ИСП
Институт Системного Программирования
им. В.П. Иванникова
Российской Академии наук

ISSN 2079-8156 (Print)
ISSN 2220-6426 (Online)

Труды Института Системного Программирования РАН

Proceedings of the Institute for System Programming of the RAS

Том 30, выпуск 2

Volume 30, issue 2

Москва 2018

ISSN 2079-8156 (Print), ISSN 2220-6426 (Online)

Труды Института системного программирования РАН

Proceedings of the Institute for System Programming of the RAS

Труды ИСП РАН – это издание с двойной анонимной системой рецензирования, публикующее научные статьи, относящиеся ко всем областях системного программирования, технологий программирования и вычислительной техники. Целью издания является формирование научно-информационной среды в этих областях путем публикации высококачественных статей в открытом доступе.

Издание предназначено для исследователей, студентов и аспирантов, а также практиков. Оно охватывает широкий спектр тем, включая, в частности, следующие:

- операционные системы;
- компиляторные технологии;
- базы данных и информационные системы;
- параллельные и распределенные системы;
- автоматизированная разработка программ;
- верификация, валидация и тестирование;
- статический и динамический анализ;
- защита и обеспечение безопасности ПО;
- компьютерные алгоритмы;
- искусственный интеллект.

Журнал издается по одному тому в год, шесть выпусков в каждом томе.

Поддерживается открытый доступ к содержанию издания, обеспечивая доступность результатов исследований для общественности и поддерживая глобальный обмен знаниями.

Труды ИСП РАН реферируются и/или индексируются в:

Proceedings of ISP RAS are a double-blind peer-reviewed journal publishing scientific articles in the areas of system programming, software engineering, and computer science. The journal's goal is to develop a respected network of knowledge in the mentioned above areas by publishing high quality articles on open access.

The journal is intended for researchers, students, and practitioners. It covers a wide variety of topics including (but not limited to):

- Operating Systems.
- Compiler Technology.
- Databases and Information Systems.
- Parallel and Distributed Systems.
- Software Engineering.
- Software Modeling and Design Tools.
- Verification, Validation, and Testing.
- Static and Dynamic Analysis.
- Software Safety and Security.
- Computer Algorithms.
- Artificial Intelligence.

The journal is published one volume per year, six issues in each volume.

Open access to the journal content allows to provide public access to the research results and to support global exchange of knowledge. **Proceedings of ISP RAS** is abstracted and/or indexed in:

УДК004.45

Адрес: 109004, г. Москва, ул. А. Солженицына, дом 25.
Телефон: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Сайт: http://www.ispras.ru/proceedings/

Address: 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.
Tel: +7(495) 912-44-25
E-mail: info-isp@ispras.ru
Web: http://www.ispras.ru/en/proceedings/

Содержание

Table of Contents

Чистая компиляция как парадигма программирования

А.В. Столяров <avst@cs.msu.ru>
О.Г. Французов <franoleg@intelib.org>
А.С. Аникина <a.anikina1993@gmail.com>
Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, вл. 1

Аннотация. В научной литературе широко представлено исследование возможностей интерпретируемого выполнения компьютерных программ. При этом противоположный подход, основанный на элиминации любых проявлений интерпретируемости, насколько можно судить, никем всерьез не исследовался. В настоящей статье рассматривается подход, предполагающий отделение всего происходящего во время исполнения программы от инструментов, использующихся программистом при ее написании, и намечается путь к построению универсального (равно пригодного как для системных, так и для прикладных задач) языка программирования.

Ключевые слова: парадигма программирования; компиляция; интерпретация; рефлексия.

DOI: 10.15514/ISPRAS-2018-30(2)-1

Для цитирования: Столяров А.В., Французов О.Г., Аникина А.С. Чистая компиляция как парадигма программирования. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 7-24. DOI: 10.15514/ISPRAS-2018-30(2)-1

1. Введение

Граница между интерпретируемым и компилируемым исполнением в наше время утратила четкость; элементы интерпретации проникают в компилируемое окружение и наоборот. Многие популярные языки программирования, такие как C# и Java, традиционно компилируются, но результатом компиляции становится не машинный код, а некоторое промежуточное представление, предназначенное для выполнения (т. е. интерпретации) виртуальной машиной. С другой стороны, современные интерпретаторы в большинстве случаев сначала переводят исходный текст программы в некоторое внутреннее представление, которое затем исполняют в режиме интерпретации. Отличия между таким компилятором и таким интерпретатором оказываются довольно эфемерными: в первом случае промежуточное представление сохраняется во внешнем файле, а виртуальная машина реализована отдельно от транслятора, во втором случае виртуальная

машина интегрирована с транслятором, что делает файл промежуточного представления необязательным.

Можно сформулировать достаточно очевидный критерий для отнесения модели исполнения к компиляции или интерпретации: интерпретатор во время исполнения программы должен сам находиться в памяти компьютера, что и отличает его от компилятора, который (сам по себе) во время исполнения не нужен. К сожалению, этот критерий не отличается высокой содержательностью. Для языков программирования, содержащих примитив EVAL и вследствие этого заведомо относящихся к интерпретируемым, таких как Lisp и (особенно) Scheme, существуют компиляторы, удовлетворяющие рассматриваемому критерию, что дает основания их сторонникам отрицать интерпретируемую сущность таких языков. То обстоятельство, что в итоговый исполняемый файл (напрямую, либо в виде динамически связываемой библиотеки) включается практически полный код интерпретатора, сторонников этой точки зрения не смущает. Более того, если рассмотреть одну из наиболее популярных в настоящее время реализаций Common Lisp – SBCL, обнаруживается, что этот транслятор работает в режиме интерпретации, а создать исполняемый файл хотя и позволяет, но крайне неочевидным путем, и размер этого файла делает использование такого режима практически бессмысленным; тем не менее, авторы SBCL в документации прямо заявляют, что их реализация является компилируемой, и предлагают «игнорировать» всех, кто говорит об интерпретируемой сущности языка Lisp.

Еще более запутывает картину применение так называемой JIT-компиляции (just-in-time compilation), при которой интерпретатор во время исполнения программы переводит некоторые ее части непосредственно в машинный код и затем исполняет его.

С другой стороны, даже в программах на таком традиционно компилируемом языке, как C, можно обнаружить элементы интерпретации: форматные строки функций семейств scanf и printf анализируются во время исполнения программы, причем анализу во время исполнения подвергается именно строка текста в том виде, в котором она написана программистом в исходном тексте программы. Конечно, форматные строки не обладают тьюринг-полнотой, их возможности примитивны, но это не исключает факта анализа фрагмента исходного текста во время исполнения. Некоторые авторы относят к элементам интерпретации также позднее связывание, реализуемое (например, в языке C++) механизмом виртуальных функций.

Больше того, можно столкнуться с утверждением, что любое исполнение является интерпретирующим, поскольку даже исполнение машинного кода центральным процессором есть не что иное, как интерпретация, причем, если вспомнить о существовании в процессорах микрокода, такое утверждение оказывается не столь далеким от истины.

В литературе, в том числе в научных работах, подробно обсуждаются возможности, получаемые при добавлении все новых и новых элементов

интерпретации. Эти возможности известны под общим термином «рефлексия». Обзор этого направления можно найти в работе [1]. Как ни странно, противоположный подход, который должен был бы основываться на элиминации интерпретирующего исполнения, в литературе не освещается вовсе.

2. Рефлексия

Намек на присутствие рефлексии можно заметить во многих современных языках программирования. Такие языки предоставляют возможность получения некоторой информации об исходном коде программы во время ее исполнения. Например, одни языки позволяют динамически определять тип переменной или объекта (операторы dynamic_cast и typeid в языке C++), а другие – получить информацию об именах переменных, функций, классов и т. д. (в таких языках как C# и Python существуют способы получения имени класса, придуманного программистом). Следует признать, что такие возможности удобны, в частности, при отладке программ.

В некоторых языках программирования присутствуют средства, позволяющие во время исполнения программы заменить одну реализацию метода класса на другую (так, в Objective-C это делается с помощью функции class_replaceMethod). Такие возможности тоже считаются рефлексией.

Само понятие рефлексии было введено Брайаном Смитом в работе [2]. Смит описывает вычисление программы в виде связей между тремя *доменами*: синтаксическим доменом (исходный код), доменом внутреннего представления программы и доменом «реального мира». Связи между доменами представлены процессами *интернализации* (internalization), *нормализации* (normalization) и *денотации* (denotation), которые соответствуют переходу от объектов исходного кода программы к объектам внутреннего представления, преобразованиям этих объектов (попросту говоря, вычислению результата) и интерпретации полученных результатов, то есть отображению объектов внутреннего представления на объекты предметной области.

При обсуждении рефлексии часто используется понятие *реификации* (reification), исходно пришедшее из философии. В данном случае под реификацией того или иного аспекта устройства программы или ее выполнения подразумевается, что к этому аспекту программе предоставляется доступ наравне с обычными данными или другими объектами, с которыми она может работать. В качестве широко распространенного примера реификации можно привести средства анализа объектов программы во время выполнения, которые обычно относят к рефлексии: к примеру, функция dir в языке Python и метод Type.GetProperties() стандартной библиотеки платформы .NET позволяют получить список атрибутов объекта. В качестве важного случая реификации следует упомянуть доступ к именам, введенным программистом, во время выполнения программы. Так, платформа .NET позволяет получить имя класса, метода, свойства и т. п. через функции стандартной библиотеки из пространства имен System.Reflection. Аналогом этого для языка Lisp будут встроенные

примитивы symbol-name и intern, которые позволяют из атома получить строку и наоборот; Prolog предусматривает для той же цели встроенный предикат name.

Брайан Смит не включал в понятие рефлексии этот аспект – по-видимому, потому, что Lisp для его работы служил отправной точкой.

Любопытно заметить, что язык C позволяет преобразовывать адреса переменных и функций в целые числа и обратно без потери информации, что также можно считать проявлением реификации.

В терминах, введенных Смитом, язык программирования поддерживает *структурную рефлексию* (structural reflection), если он позволяет анализировать или изменять объекты внутреннего домена. Если же язык позволяет вмешаться в процесс вычисления (*нормализации* по Смиту), т. е. предоставляет прямой доступ к вычислительному контексту программы, речь идет о *поведенческой рефлексии* (behavioral reflection). Смит отмечает, что для ее обеспечения от языка требуется поддержка реификации не только объектов программы, но и самих языка и вычислителя. Иначе говоря, программе через те или иные примитивы должна быть предоставлена возможность управлять интерпретатором, который ее выполняет. Для компилируемого языка должны быть полностью реифицированы как среда времени выполнения, так и транслятор, что в определенном смысле превращает компиляцию в интерпретацию, ведь транслятор теперь не может отсутствовать во время выполнения.

Поведенческая рефлексия изучена гораздо хуже, чем структурная, хотя отдельные ее элементы уже давно встречаются в языках программирования; в качестве примера можно назвать *продолжения* (continuations) языка Scheme. Для платформы .NET относительно недавно появился набор инструментов под общим названием Roslyn, включающий в себя компиляторы C# и Visual Basic, управляемые через библиотечные функции [3].

Имея доступ к управлению собственным исполнителем, программа, написанная на языке с поддержкой поведенческой рефлексии, может во время вычисления изменять структуры, которые вычисляют ее саму. Это может привести к несовместимости изменений, внесенных рефлективным кодом, и изменений, внесенных интерпретатором. Такой феномен называется *интроспективным наложением* (introspective overlap). Для борьбы с этим феноменом Смит в своей работе ввел понятие *рефлективной башни* (reflective tower), которая представляет собой стек из интерпретаторов, где первый интерпретатор исполняет исходную программу, второй интерпретатор исполняет первый и т. д. Таким образом, для поддержания возможностей поведенческой рефлексии требуется наличие не только интерпретатора программы в памяти, а их потенциально бесконечное количество.

Интересно отметить, что Смит для обозначения составляющих рефлективной башни ввел термин *рефлективная обрабатывающая программа* (reflective processor program); это более общее понятие, нежели «интерпретатор», но, так или иначе, это некоторый программно реализованный исполнитель, который должен присутствовать в памяти на момент вычисления.

3. Недостатки интерпретируемого исполнения

Когда речь идет о недопустимости или нежелательности интерпретации в тех или иных случаях, чаще всего в качестве *единственного* недостатка интерпретации почему-то (иногда неявно) рассматривают потери в эффективности; сторонники интерпретируемого исполнения тратят много сил и красноречия на обоснование возможности эффективной интерпретации, то есть такой, которая либо вообще не уступает компилируемому исполнению, либо уступает настолько незначительно, чтобы потерями в эффективности (в основном по времени исполнения, реже припоминают также эффективность использования оперативной памяти) можно было пренебречь.

Конечно, практика показывает, что эффективность интерпретируемого исполнения, несмотря на все усилия его сторонников, в подавляющем большинстве случаев оставляет желать много лучшего, но дело, как ни странно, не в этом. Сравнительно низкая эффективность по времени исполнения представляет собой существенный, но отнюдь не единственный и даже, возможно, не главный недостаток интерпретации; мы попытаемся провести краткий обзор других ее недостатков.

Одно из основных преимуществ интерпретируемой парадигмы – гибкость во время исполнения программы. Возможность в любой момент воспользоваться рефлексией для того, чтобы доинтерпретировать какой-то фрагмент кода или поработать со структурами данных, создает определенный потенциал, которого языки, тяготеющие к компилируемости и статичности, лишены; но любая возможность имеет свою цену. Повышенная гибкость программы во время исполнения приводит к увеличению **хрупкости кода**. Чем больше нетривиальной логики относится на этап выполнения программы, тем труднее становится тестирование. Компилируемые языки в чем-то ограничивают программиста, но эти ограничения служат страховкой хотя бы от части ошибок – по крайней мере, синтаксические ошибки будут обнаружены на этапе компиляции. В то же время получить синтаксическую ошибку на этапе выполнения программы, написанной на интерпретируемом языке – явление обычное.

Известен метод борьбы с этой проблемой – стопроцентное покрытие модульными тестами (что само по себе – прекрасная практика), но если в дело идут механизмы рефлексии, то нельзя быть уверенным, что даже при стопроцентном покрытии нет граничного случая, когда логика работы кода окажется нарушена. Интерпретируемая парадигма при этом не позволяет исключить даже ошибок синтаксиса или ситуаций отсутствия поля или метода в объекте.

Механизмы рефлексии позволяют расширить интерфейс объекта во время выполнения, «по месту» – такая техника носит не очень уважительное название *«латания по-обезьяньи»* «monkey patching» [4]. Ее применение также служит

источником хрупкости, однако среди программистов на Ruby, тем не менее, эта техника считается нормальной и допустимой.

Логичным продолжением стремления обнаружить как можно больше ошибок как можно раньше (например, на этапе трансляции) можно считать инструменты статического анализа кода. Таких инструментов существует множество, они могут, например, указывать на возможные проблемы (lint) или, интегрируясь с текстовым редактором, упрощать процесс написания кода, предоставляя средства навигации, автодополнения и т. д. Гибкость интерпретируемых языков ограничивает возможности такого класса инструментов, увеличивая **сложность статического анализа** вплоть до полной невозможности его выполнения. Задача выдачи списка атрибутов объекта для компилируемых языков решается относительно просто на базе статического анализа типа. В рамках интерпретируемой парадигмы потребовалось бы полностью выполнить программу до указанной точки, чтобы точно ответить на вопрос, какие атрибуты в итоге у объекта окажутся.

Одним из препятствий к использованию интерпретируемых языков программирования оказывается необходимость наличия интерпретатора на компьютере конечного пользователя. Интерпретаторы сами по себе относятся к достаточно сложным программным инструментам, а их авторы зачастую совершенно не стремятся к упрощению процедуры развертывания их продуктов, предполагая, что их пользователь – профессиональный программист. Это приводит к появлению **языковых экосистем**. Конечный пользователь, который уже по тем или иным причинам работает с программными средствами на некотором интерпретируемом языке, при выборе новых инструментов отдает предпочтение тем, которые написаны на уже «освоенном» им языке, поскольку при их внедрении ему не придется тратить время на освоение еще одной языковой экосистемы.

Многие интерпретируемые языки программирования (в качестве примеров можно привести Perl, Python и Ruby) имеют собственные пакетные решения для установки как прикладных программ и средств, так и библиотек (зависимостей). Молчаливо предполагается, что конечный пользователь должен уметь пользоваться «экосистемным» пакетным менеджером и владеть некоторым базовым набором знаний и умений для работы внутри экосистемы. Так, краткое руководство по установке и настройке популярного генератора статических сайтов Jekyll обещает, что начать работу с инструментом можно будет «за считанные секунды», при этом первой командой, которую предлагается выполнить в руководстве, оказывается `gem install jekyll bundler`. Эта команда требует наличия настроенного разработческого окружения Ruby, но в руководстве Jekyll вопрос того, как, для начала, добиться работоспособности этой команды, остается за кадром.

Дело усугубляется тем, что «экосистемные» пакетные менеджеры конфликтуют с пакетными менеджерами операционной системы, при этом для конечного пользователя часто оказывается невозможно избежать использования

«экосистемного» средства, даже если он задастся такой целью. Документация на Jekyll, к примеру, заявляет в качестве системных требований наличие не только Ruby «со всеми заголовочными файлами», но и компилятор GCC с системой сборки make [5]. Таким образом, граница между средой времени выполнения, очевидно необходимой интерпретируемым языкам, и окружением разработчика оказывается фактически стертой.

На другом полюсе спектра решений ─ подход, когда программа, написанная на интерпретируемом языке, вместе со всеми своими зависимостями и интерпретатором упаковывается в дистрибутивный комплект, обеспечивающий возможность установки конечным пользователем. Такой подход часто применяется для массовых продуктов, которые требуется распространять среди пользователей ОС Windows; например, именно так организован дистрибутив системы контроля версий Mercurial, несмотря на то, что этот инструмент очевидным образом предназначен для профессиональных программистов.

Этот подход решает часть проблем, но и он не лишен недостатков: помимо неоправданно большого размера дистрибутивного комплекта, распространяемая таким образом программа оказывается развернута в своем собственном экземпляре экосистемы и может, например, не иметь доступа к пакетам, установленным стандартными «экосистемными» средствами.

Когда у одного из авторов статьи возникла необходимость развернуть на сервере с ОС Windows систему контроля версий Mercurial и систему отслеживания замечаний Trac, воспользоваться штатными дистрибутивными комплектами каждой из систем не удалось. Обе системы написаны на языке Python, но при установке из дистрибутивных комплектов модули Mercurial, предоставляющие его API, оказались не доступны системе Trac из-за того, что каждая из двух систем использовала свой отдельный интерпретатор языка со своим набором модулей-пакетов. Единственным возможным вариантом оказалась установка всех пакетов вручную из исходного кода, по модели, более пригодной для окружения разработчика, чем конечного пользователя; при этом пришлось пожертвовать производительностью, отказавшись от собственных расширений системы Mercurial, так как для их сборки потребовалось бы разворачивать на активно эксплуатируемой серверной машине еще и систему программирования на базе языка C.

Для программ, имеющих существенный объем исходного кода, интерпретирующее исполнение может оказаться непригодным из-за **длительного времени запуска программы**, ведь фактически трансляцию программы приходится при каждом запуске производить заново. Естественно, современные интерпретаторы стараются эту проблему так или иначе решить, переводя программу во внутреннее представление по частям по мере надобности, либо сохраняя сгенерированное внутреннее представление (полностью или частично) для использования во время последующих запусков; факта существования проблемы все это не отменяет.

4. Библиотечная поддержка времени исполнения

Ситуация резко осложняется, когда написанная программа предназначена к непосредственной работе с аппаратурой и не может полагаться на операционную систему: либо эта программа сама представляет собой часть операционной системы (например, драйвер), либо операционная система отсутствует, как в случае прошивок для микроконтроллеров. В таких условиях оказываются неприемлемы многие особенности трансляторов, на первый взгляд не имеющие отношения к интерпретирующему исполнению, такие как необходимость библиотечной поддержки времени исполнения.

Так, авторы стандартов языка C, волюнтаристски включив в спецификацию языка огромный пласт библиотечных возможностей (так называемые *стандартные библиотеки*), были, тем не менее, вынуждены признать, что все это не может быть обязательной частью языка, и определили два возможных окружения – обычное, предполагающее наличие операционной системы и библиотечной поддержки (hosted), и *самостоятельное* (freestanding), для которого стандарт требует поддержки только семи заголовочных файлов, ни один из которых не вводит библиотечных функций. Следует заметить, что признание допустимости самостоятельного окружения было вынужденным шагом. Стандартная библиотека не используется и не может быть использована в ядре операционной системы. Потребовав на уровне стандарта языка, чтобы стандартная библиотека предоставлялась всегда и везде, пришлось бы считать, что для написания ядер операционных систем используется некий «нестандартный диалект» языка C; между тем, этот язык был изначально создан специально для написания ядра ОС Unix.

Появившийся несколько лет назад язык Rust изначально включал ряд высокоуровневых возможностей вплоть до сборки мусора (путем подсчета ссылок [6]); создатели Rust вовремя осознали, что такой подход существенно ограничивает область применения языка, и к моменту выпуска версии 1.0 предусмотрели аналог «самостоятельного» окружения языка C, для чего спектр возможностей языка пришлось урезать.

В контексте нашей статьи здесь интересен тот факт, что транслятор, ориентированный на компилируемое исполнение, может оказаться непригоден в конкретной задаче из-за особенностей библиотек, на наличие которых он полагается. Чаще всего внутреннее устройство компиляторов не подвергается документированию такого уровня, который бы позволил пользователю заменить версии библиотек времени исполнения на более подходящие; как следствие, в определенных случаях неподходящим оказывается весь компилятор целиком. При внимательном рассмотрении вопрос *допустимости требования о наличии во время исполнения программы компонентов транслятора* (включая библиотеку времени исполнения) оказывается обобщением вопроса о допустимости тех или иных элементов интерпретации.

5. Компилируемое выполнение как парадигма

При проектировании программы с расчетом на интерпретируемое исполнение практически отсутствуют ограничения на средства, поддерживающие выполнение программы. Во время исполнения оказываются при желании доступны средства манипуляции состоянием программы как объектом («структурная рефлексия»); в памяти присутствует интерпретатор, так что можно средствами выполняющейся программы формировать новые фрагменты для нее же самой и сразу их выполнять (примитив EVAL; в частности, конфигурационные файлы могут быть написаны на том же языке, что и сама программа), имеется информация об особенностях исходного текста программы – например, о введенных в нем именах, даже если сам исходный текст как таковой почему-то недоступен; как уже отмечалось, сторонники рефлексивного окружения на этом не останавливаются, требуя, чтобы язык допускал средства манипуляции программой, то есть чтобы программным образом во время исполнения можно было вносить в программу изменения так же, как во время написания программы их вносит программист. Можно заметить, что при этом вообще стирается грань между временем написания и временем исполнения программы.

Очевидно, что мысль о применении средств такого рода может возникнуть лишь в рамках определенного стиля мышления, что позволяет говорить о *парадигме* интерпретируемого исполнения. Аналогичным образом, по-видимому, будет корректно рассмотреть *парадигму компилируемого исполнения*; как ни странно, авторам статьи не удалось найти ни одной работы, в которой бы вводились такие термины. Между тем, определенные элементы мышления, которое можно было бы обозначить как парадигму компилируемого исполнения, встречаются на практике, и достаточно часто. Для примера можно рассмотреть ситуацию интерпретатора, встроенного в программу, написанную на компилируемом языке. К такому решению часто прибегают в ситуации, когда для настройки программы оказывается недостаточно простых конфигурационных файлов и требуется давать ей указания на тьюринг-полном языке; в качестве встраиваемых часто используют такие языки, как Lua, Tcl, разнообразные диалекты Лиспа и т. п. Многие программисты на интуитивном уровне предполагают, что такой интерпретатор, встроенный в основную программу, может выполнять программы, предоставленные конечным пользователем и предназначенные для целей настройки основной программы, но при этом встроенный интерпретируемый язык не следует применять для написания частей *самой программы*; все ее возможности следует реализовать на основном языке проекта, даже если такая реализация окажется более трудоемкой. Интересно, что в большинстве случаев не удается найти никаких технических аргументов в пользу реализации той или иной особенности именно на основном, а не встроенном языке, речь в лучшем случае идет о некой «концептуальной целостности». Не следует, однако, недооценивать концептуальную целостность. Программирование, как известно, представляет собой чрезвычайно тяжелый вид

интеллектуальной деятельности, и культура мышления способна оказать существенную помощь в борьбе со сложностью осмысления компьютерных программ.

Аналогично тому, как сторонники парадигмы рефлексии определяют несколько уровней возможностей, в контексте парадигмы компилируемого исполнения можно указать различные уровни строгости соответствия программы этой парадигме. Если рефлексия предполагает наличие и доступность инструментов программиста во время исполнения программы, то компиляция, по-видимому, должна предполагать их отсутствие; если продвинуться еще дальше, следует потребовать, чтобы язык программирования и инструментарий программиста допускал четкое разделение между особенностями программы, видимыми пользователю, и решениями, принимаемыми программистом в процессе ее реализации; иначе говоря, должна быть обеспечена как минимум *возможность* того, чтобы выбираемые программистом способы и методы достижения целей никак не влияли на все, что видит конечный пользователь программы.

В качестве первого и наиболее очевидного уровня соответствия используемых инструментов парадигме компилируемого исполнения можно предложить требование отсутствия транслятора в памяти компьютера во время исполнения программы; транслятор должен позволять создание исполняемого файла, запуск которого возможен в отсутствие транслятора. На следующем уровне можно потребовать отсутствия в памяти не только транслятора, но и отдельных его элементов, таких как лексический и синтаксический анализатор, генератор кода и прочее. Широко известен подход к синтезу исполняемого файла путем соединения значительной части компонентов интерпретатора с исходным текстом программы или тем или иным его внутреннем представлением; такой подход удовлетворяет первому уровню требований, но уже не удовлетворяет второму.

Заметив, что библиотеки очевидным образом представляют собой часть системы программирования, причем многие из них (не только «стандартные») прилагаются к транслятору, можно довести два предыдущих уровня требований до определенного логического завершения, потребовав для начала, чтобы генерируемый исполняемый файл не зависел (или по крайней мере *мог* не зависеть) от наличия каких бы то ни было компонентов системы программирования, в том числе динамически подгружаемых библиотек; переходя к следующему уровню, придется потребовать, чтобы и в сам исполняемый файл не включались никакие библиотеки кроме тех, которые в явном виде затребовал программист, а минимально допустимое множество таких библиотек было пустым. Удовлетворить такому требованию может лишь язык программирования, из которого исключены (вытеснены в библиотеку) любые возможности, требующие сколько-нибудь нетривиальной реализации на уровне машинного кода.

Отдельно следует упомянуть имена (идентификаторы), вводимые программистом в исходном тексте программы. В ранних диалектах языков,

ориентированных на обработку символьной (слабоструктурированной) информации, таких как Lisp, Prolog, Planner и т. п., отсутствовали строковые константы как отдельная сущность; в роли слов естественного языка (элементов текста) выступали обычные идентификаторы (атомы). Впоследствии от такого использования атомов отказались; все современные диалекты Лиспа включают строковые константы как отдельную сущность, есть они и в наиболее современных реализациях языка Prolog, таких как SWI-Prolog. Можно сделать достаточно логичное предположение о причинах отказа от атомов в роли строк. В эпоху, когда языки рассматриваемой категории только возникли, практически не существовало *конечных пользователей*; любой человек, работающий с компьютером, был программистом. Появление конечных пользователей позволило осознать, что имена, используемые программистом в программе, и строки, видимые пользователю, предназначены для принципиально разных целей, относятся, если угодно, к разным предметным областям и не должны смешиваться – хотя бы из соображений концептуальной чистоты.

Из сказанного вытекает очередное требование к компилируемому стилю исполнения: работа программы не должна никак зависеть от выбранного программистом набора имен (идентификаторов). Можно сказать, что программа, полностью соответствующая парадигме компилируемого исполне-ния, должна быть *устойчива к альфа-преобразованию*: при любом переименовании идентификаторов в исходном тексте программы, если такое переименование не нарушает очевидного требования о сохранении уникальности идентификаторов, результаты работы программы не должны никак измениться.

Отметим, что всем перечисленным требованиям удовлетворяют в наше время разве что языки ассемблеров. В частности, компиляторы языка C обычно зависят от пусть и небольшого, но далеко не пустого множества библиотечных функций; так, компилятор gcc имеет специальный режим для создания «самостоятельных» исполняемых файлов (флаг -ffreestanding), но даже в этом режиме на этапе компоновки может потребоваться подключение библиотеки libgcc: например, перемножение 64-битных целых чисел на 32-битных процессорах этот компилятор реализует через вызов библиотечной функции. Кроме того, в документации к компилятору перечислены несколько функций, на наличие которых он полагается, несмотря на режим создания «самостоятельных» исполняемых файлов (например, memcpy). Более того, язык C, строго говоря, не обладает устойчивостью к альфа-преобразованиям за счет наличия в макропроцессоре возможности превратить идентификатор в его имя (строковую константу). Например, в программе можно объявить такой макрос:

```
#define GETIDNAME(x) (#x)
```

и затем использовать следующий вызов функции printf:

```
int myvar;
  /* ... */
printf("%s\n", GETIDNAME(myvar));
```

Такой вызов выведет строку «myvar»; если переменную myvar в программе переименовать, то, очевидно, изменится и выдаваемая программой строка.

Впрочем, даже достижение уровня требований, которым удовлетворяют только языки ассемблеров, не обязывает нас остановиться. Следующее требование оказывается очевидным как с точки зрения эффективности исполнения, так и с точки зрения самодисциплины программиста, но, тем не менее, соблюдать его в современных условиях практически невозможно. Итак: *любые вычисления и преобразования, которые могут быть выполнены во время компиляции, не должны переноситься на время исполнения программы*. Нарушением этого принципа будет, в частности, интерпретация форматных строк в функциях printf/scanf. К сожалению, язык С не обладает изобразительными средствами, которые позволили бы адекватно (без чрезмерных трудностей для программиста) создать какую-то структуру данных, содержащую все необходимые сведения о формате ввода или вывода в таком виде, который не требовал бы лишнего анализа (например, перевода чисел из строкового представления, при том что само число уже известно во время компиляции).

В терминах содержания исполняемого файла можно сформулировать еще два требования к компиляции. Во-первых, система программирования должна позволять создать такой исполняемый файл, чтобы никакими средствами анализа нельзя было определить, какие конкретно инструменты были задействованы для его создания. Это требование позволяет надеяться, что в исполняемом файле *действительно не останется ничего лишнего*, обусловленного только выбранными инструментами и методами решения задачи и не имеющего отношения к решенной задаче как таковой. Во-вторых, можно потребовать, чтобы система программирования позволяла сформировать *любой* исполняемый файл, корректный с точки зрения целевой платформы. Среди существующих инструментов такими свойствами обладают только ассемблеры. Исполняемый файл, созданный компилятором языка С, позволяет определить, какой был использован компилятор, причем в большинстве случаев – с точностью до версии.

Модель трансляции программ, удовлетворяющую всем перечисленным требованиям, мы назовем *чистой компиляцией* или *чисто компилируемым исполнением*. Очевидно, что для практического применения этой модели потребуется новый язык программирования, поскольку существующие языки высокого уровня (даже язык С) введенным требованиям заведомо не удовлетворяют, а предложение о более активном использовании ассемблеров можно всерьез не рассматривать; впрочем, как мы видели, для применения противоположной по смыслу *полной рефлексии* тоже не годится ни один существующий язык программирования.

Отметим, что «интерпретация» машинного кода центральным процессором в принятых терминах никак не противоречит введенной модели, поскольку ни сам центральный процессор, ни его микрокод не являются инструментами программиста.

6. Заключение

На протяжении всего времени существования программирования как дисциплины наблюдалась тенденция повышения уровня абстракций в языках программирования, обусловленная снижением трудоемкости создания программ, во всяком случае, на стадиях кодирования и отладки. Использование неких программно реализованных посредников между программой и вычислительной машиной – интерпретаторов – оказалось наиболее простым и действенным способом перехода от концепций, навязываемых машиной, к абстрактным вычислительным моделям высокого уровня.

На сегодняшний день понятия *высокоуровневости* и *интерпретируемости* языков программирования во многих случаях едва ли не отождествляются. Большинство современных языков программирования высокого уровня так или иначе вбирает в себя свойства, присущие парадигме интерпретации; это могут быть как чисто интерпретируемые языки (Python, Ruby, JavaScript), так и компилируемые, но обладающие развитыми возможностями рефлексии, которые обеспечиваются интерпретацией промежуточного представления (Java, C#).

Оплотом компилируемого исполнения остаются сравнительно немногочисленные области, в которых интерпретация или полностью неприменима, или по тем или иным причинам очевидно нежелательна. В современных условиях чаще всего для таких областей применяется язык C, причем в некоторых случаях единственной альтернативой ему оказывается программирование на языке ассемблера, которое в силу его чрезвычайно высокой трудоемкости можно всерьез не рассматривать.

Трудоемкость программирования на C ниже, чем для языков ассемблера, но все еще такова, что для использования C нужны серьезные причины. Этот язык применяют, в частности, когда во время исполнения заведомо невозможно присутствие какой бы то ни было программной инфраструктуры, как в случае ядер операционных систем и прошивок микроконтроллеров. Язык C также выбирают при наличии повышенных требований к безопасности, поскольку это едва ли не единственный язык, допускающий осмысленный ручной аудит исходных текстов. Кроме того, C позволяет создавать программы практически настолько же эффективными по времени и памяти, насколько это вообще возможно на конкретной используемой аппаратуре. Наконец, сколь бы странно это ни выглядело, именно на C удается добиться наибольшей степени переносимости программ (на уровне исходных текстов) между разнообразными аппаратными платформами и операционными системами.

В подавляющем большинстве случаев применения языка C такой выбор оказывается вынужденным; о разнообразных недостатках этого языка написано огромное количество текстов, но определяющей особенностью здесь можно считать чрезвычайную выразительную бедность этого языка, требующую большого количества ручной работы, от которой невозможно избавиться никакими ухищрениями. Вызывает недоумение тот факт, что среди сотен новых

языков программирования, появившихся позже языка C, ему до сих пор не нашлось адекватной замены.

В этом плане показателен язык Go, в котором его создатели, в том числе Кен Томпсон и Роб Пайк, по их собственному утверждению, пытались исправить ошибки, допущенные при создании C. Этот язык использует сборку мусора как основную модель управления памятью, что полностью исключает применение Go в большинстве ситуаций, в которых сейчас используется C. На примере языка Go можно понять важный факт: низкоуровневость языка программирования, или, если говорить в терминах, введенных выше, его особенности, продиктованные стремлением к чисто компилируемому исполнению, следует рассматривать не как «ошибки» дизайна языка, но, напротив, как обязательное свойство для языка программирования, который смог бы, наконец, заменить неуклюжий и очевидным образом морально устаревший язык C.

Обилие «высокоуровневых» (читай – интерпретируемых до той или иной степени) языков программирования и всего одного языка, широко используемого для «системных» задач (чистого C) подспудно формирует массовое убеждение, что при «системность» языка неизбежно влечет трудоемкость работы на нем, а снижение трудоемкости программирования возможно только внедрением в язык программирования возможностей, далеких от базового вычислителя.

Авторы появившегося не так давно языка Rust попытались (возможно, непреднамеренно) опровергнуть это утверждение, создав язык, пригодный как для низкоуровневого, так и для высокоуровневого программирования, не пожертвовав сформулированными выше свойствами. В частности, компилятор Rust позволяет создавать программы, работающие в самостоятельном окружении, а развитые возможности автоматического управления памятью не требуют поддержки времени выполнения. Цена этого – высокая сложность языка Rust и, как следствие, компилятора. Тем не менее, Rust демонстрирует, что высокоуровневый и выразительный язык программирования может оставаться в рамках парадигмы чисто компилируемого исполнения. Сможет ли Rust со временем заменить C – вопрос открытый; часть его возможностей не допускает тривиальной реализации, и это может стать препятствием для его использования в таких ситуациях, когда программисту требуется предсказуемость порождаемого машинного кода.

Достаточно любопытен в этом плане язык C++, который на ранних стадиях своего развития, оставаясь низкоуровневым, позволял благодаря поддержке абстрактных типов данных и перегрузки операций создавать абстракции произвольного уровня сложности. После введения в язык таких возможностей, как обработка исключений и идентификация типов во время исполнения (RTTI), C++ оказался безнадежно далек от соответствия принципам чистой компиляции; кроме того, даже на начальном этапе становления C++ не мог считаться чисто компилируемым (в смысле, введенном в нашей статье), поскольку, во-первых, обладал практически всеми недостатками чистого C (кроме разве что

интерпретации форматных строк printf/scanf), и, во-вторых, включал достаточно сложную реализацию виртуальных функций.

В результате последовательного принятия все более и более сложных «стандартов» C++ этот язык как явление к настоящему времени может рассматриваться только как язык высокого уровня, заведомо непригодный для задач системного программирования, несмотря на то, что среди всех популярных ныне высокоуровневых языков C++ остается единственным *компилируемым в машинный код*; все остальные либо интерпретируются, либо транслируются в код промежуточной виртуальной машины. В контексте нашей статьи здесь скорее важен тот факт, что, оставаясь компилируемым, C++ позволяет исключить рутинную ручную работу, хорошо известную программистам, пишущим на чистом C. Делается это за счет библиотечных расширений, опирающихся на поддержку в C++ полноценных абстрактных типов данных, которые позволяют контролировать не только выполнение обычных операций над объектами таких типов, но и их присваивание, копирование при передаче через параметры и при возврате из функций и т. п.

Если учесть как негативный, так и положительный опыт языка C++, можно создать новый язык, допускающий, с одной стороны, чистую компиляцию в терминах, введенных выше, но при этом, с другой стороны, позволяющий ввести сколь угодно сложные абстракции. Такой язык мог бы одинаково хорошо (при условии адекватного выбора используемых библиотек, различного в каждом конкретном случае) подходить для решения как «системных», так и для прикладных задач, то есть быть в полном смысле *универсальным*.

Список литературы

[1]. Malenfant J., Jacques M., Demers F.-N. A Tutorial on Behavioral Reflection and its Implementation. Proceedings of Reflection 96, 1997, pp. 1-20. Доступно по ссылке: http://www2.parc.com/csl/groups/sda/projects/reflection96/docs/malenfant/malenfant.pdf

[2]. Smith B. C. Procedural Reflection in Programming Languages. Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the Massachusetts Institute of Technology, February 1982, 762 p. Доступно по ссылке: http://repository.readscheme.org/ftp/papers/bcsmith-thesis.pdf

[3]. McAllister N. Microsoft's Roslyn: Reinventing the compiler as we know it. IDG News Service. Дата обращения 20.10.2011 (online). Доступно по ссылке: https://www.infoworld.com/article/2621132/microsoft-net/microsoft-s-roslyn--reinventing-the-compiler-as-we-know-it.html

[4]. Limi A., Hathaway S. Monkey patch. Plone Foundation. Дата обращения 03.07.2008 (online). Доступно по ссылке: http://web.archive.org/web/20080604220320/http://plone.org/documentation/glossary/monkeypatch

[5]. Installation – Jekyll. Дата обращения 01.09.2017 (online). Доступно по ссылке: https://jekyllrb.com/docs/installation/

[6]. The Rust Language Tutorial (version 0.4). Дата обращения 30.04.2017 (online). Доступно по ссылке: https://static.rust-lang.org/doc/0.4/tutorial.html

Pure Compiled Execution as a Programming Paradigm

A.V. Stolyarov <avst@cs.msu.ru>
O.G. Frantsuzov <franoleg@intelib.org>
A.S. Anikina <a.anikina1993@gmail.com>
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia

Abstract. Interpreted execution of computer programs, its capabilities and advantages is well-covered in the computer science literature. Its key feature is reflection: the ability to access and modify the source code at run time. At the same time, interpreted execution has its shortcomings: lower performance; higher code fragility caused by the possibility to change code during run time; more complicated static analysis; runtime-tied ecosystems. And in some cases like embedded systems, runtimes and interpreted code are impractical or impossible, and compiled code with zero dependencies is the only option. Pure compiled execution can be treated as a paradigm directly opposite to reflection-powered interpretation. If the primary trait of interpreted execution is reflection, then pure compilation should cleanly separate development time and run time. This implies no part of translator being available during run time, no requirements for runtime libraries availability, and, finally, no dependence on the implementation details like variable names. While interpretation is wildly popular, compiled execution can be a conscious choice not only for low-level applications, but other cases as well. The dichotomy between low-level languages and expressive reflection-enabled language is a false one. It should be possible to create an expressive purely compiled programming language, and such a language might be equally suitable both for system programming and application development.

Keywords: programming paradigm; compilation; interpretation; reflection

DOI: 10.15514/ISPRAS-2018-30(2)-1

For citation: Stolyarov A.V., Frantsuzov O.G., Anikina A.S. Pure Compilation as a Programming Paradigm. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 2, 2018, pp. 7-24 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-1

References

[1]. Malenfant J., Jacques M., Demers F.-N. A Tutorial on Behavioral Reflection and its Implementation. Proceedings of Reflection 96, 1997, pp. 1-20. Available at: http://www2.parc.com/csl/groups/sda/projects/reflection96/docs/malenfant/malenfant.pdf

[2]. Smith B. C. Procedural Reflection in Programming Languages. Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy at the Massachusetts Institute of Technology, February 1982, 762 p. Available at: http://repository.readscheme.org/ftp/papers/bcsmith-thesis.pdf

[3]. McAllister N. Microsoft's Roslyn: Reinventing the compiler as we know it. IDG News Service, October 20, 2011 (online). Available at: https://www.infoworld.com/article/2621132/microsoft-net/microsoft-s-roslyn--reinventing-the-compiler-as-we-know-it.html

[4]. Limi A., Hathaway S. Monkey patch. Plone Foundation, Retrieved 2008-07-03 (online). Available at: http://web.archive.org/web/20080604220320/http://plone.org/documentation/glossary/monkeypatch

[5]. Installation – Jekyll. Retrieved 2017-09-01 (online). Available at: https://jekyllrb.com/docs/installation/

[6]. The Rust Language Tutorial (version 0.4). Retrieved 2017-04-30 (online). Available at: https://static.rust-lang.org/doc/0.4/tutorial.html

Распараллеливание реализаций сугубо последовательных алгоритмов

[1] *А.Б. Бугеря <shurabug@yandex.ru>*
[2] *Е.С. Ким <eugene.kim@ispras.ru>*
[2] *М.А. Соловьев <icee@ispras.ru>*
[1] *Институт прикладной математики им. М.В. Келдыша РАН, 125047, Россия, г. Москва, Миусская пл., д. 4*
[2] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*

Аннотация. Работа посвящена теме распараллеливания программ в особо сложных случаях – когда используемый алгоритм является сугубо последовательным, параллельных альтернатив используемому алгоритму нет, а время его выполнения неприемлемо велико. Рассматриваются различные методы распараллеливания программных реализаций таких алгоритмов и балансировки получающейся вычислительной нагрузки, позволяющие получить значительное ускорение выполнения прикладных программ, в которых используются сугубо последовательные алгоритмы. Приведенные методы иллюстрируются практикой их применения к двум алгоритмам, используемым в среде динамического анализа программ. Основная цель данной работы – показать, что использование в программной реализации сугубо последовательного алгоритма не означает неизбежность его последовательного выполнения. Предложенные методы распараллеливания реализаций таких алгоритмов и балансировки получающейся вычислительной нагрузки могут способствовать созданию эффективной параллельной программы, полностью использующей предоставленные ей аппаратные возможности современных вычислительных систем.

Ключевые слова: параллельное программирование; распараллеливание программ; балансировка вычислительной нагрузки

DOI: 10.15514/ISPRAS-2018-30(2)-2

Для цитирования: Бугеря А.Б., Ким Е.С., Соловьев М.А. Распараллеливание реализаций сугубо последовательных алгоритмов. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 25-44. DOI: 10.15514/ISPRAS-2018-30(2)-2

1. Введение

В современном мире имеется огромное количество различных задач, решение которых требует наличия значительных вычислительных мощностей. Такие задачи имеются во всех областях науки и промышленности, в бизнесе и даже в сфере индивидуального применения. Типичными примерами таких

ресурсоемких задач могут служить решение задач математической физики численными методами (например, моделирование процессов, происходящих в ядерном реакторе), моделирование физических, химических и биологических процессов с огромным количеством взаимодействующих сущностей различной природы, анализ и преобразования графов, поиск и анализ информации в базах данных или в информационных потоках, статический и динамический анализ программ. Постоянно появляются новые задачи подобного рода.

Время выполнения программ, решающих такие задачи, может оказаться критически большим – зачастую таким, что это становится неприемлемым для использования такой программы. И даже просто весьма долгое выполнение каких-то задач, скажем, анализа программы при исследовании ее аналитиком в интерактивном режиме, существенно снизит производительность труда и увеличит время решения задачи. Поэтому не будет преувеличением сказать, что задача разработки эффективных программ имеет важное стратегическое значение. В данной работе мы не будем касаться вопроса скорости работы аппаратной части вычислительного комплекса или вопросов создания новых алгоритмов, быстрее решающих поставленную задачу. Наша цель – ускорить процесс решения программой поставленной задачи путем оптимизации самой программы. При этом особый интерес представляют программы, реализующие последовательный алгоритм решения задачи, который не имеет параллельного аналога.

2. Ускорение процесса выполнения программы

В процессе решения задачи оптимизации программы с целью ускорить время ее выполнения можно выделить следующие основные направления.

2.1 Выбор алгоритма

Выбор оптимального алгоритма, позволяющего решать поставленную задачу в заданном окружении (окружение – это в первую очередь целевой аппаратный вычислительный комплекс, а также предполагаемый набор входных данных). Забегая вперед, надо отметить, что если предполагается распараллеливание программы (см. разд. 3), то и выбор алгоритма должен производиться с учетом этого: некоторые алгоритмы могут быть эффективно распараллелены, а некоторые – нет. Зачастую бывает так, что для решения одной и той же задачи существует несколько алгоритмов. Одни из них более простые и эффективные в последовательном исполнении, другие же могут быть более сложными, менее эффективными в последовательном исполнении, но зато допускающими эффективное параллельное выполнение.

2.2 Реализация алгоритма

Оптимальная реализация выбранного алгоритма. Это немаловажный шаг в процессе построения эффективной программы, но в рамках данной работы мы его подробно рассматривать не будем. Отметим только, что, прежде чем переходить к распараллеливанию программы и оптимизации ее параллельной версии, необходимо добиться того, чтобы последовательная версия работала оптимально: исключить повторные вычисления (кэшировать уже вычисленные данные для повторного использования, когда это возможно), исключить постоянное выделение/освобождение памяти (использовать пул предварительно выделенной памяти в таком случае), и провести прочие тому подобные оптимизации.

2.3 Распараллеливание реализации алгоритма

Распараллеливание реализации выбранного алгоритма. Когда все возможности увеличения скорости работы последовательной программы уже исчерпаны, а результат все же оставляет желать лучшего, помочь может только распараллеливание программы и последующее ее выполнение в окружении, эффективно поддерживающем выполнение параллельных программ. Параллельные вычислительные системы – это различные аппаратные решения, которые при поддержке соответствующих программных средств реализуют тем или иным способом одновременное (параллельное) выполнение различных команд, или одной и той же команды с различными наборами данных. Главная цель использования параллельных вычислений – повышение скорости вычислений за счет их параллельного выполнения. Главным критерием качества распараллеливания вычислений является сокращение общего времени решения задачи.

Идея распараллеливания вычислений базируется на том, что большинство задач может быть разделено на набор меньших независимых (или хотя бы мало зависимых) друг от друга подзадач, которые могут быть решены одновременно. Решая такой набор подзадач на параллельной вычислительной системе, можно добиться существенного уменьшения времени работы всей программы. Существует два основных типа распараллеливания (выделения подзадач): по управлению, когда в общей задаче можно выделить несколько независимых решаемых подзадач, каждая из которых выполняется со своим набором данных, и по данным, когда массив всех обрабатываемых данных делится на части, и с каждой такой частью данных решается одна и та же задача.

Очевидно, что возможности распараллеливания по управлению достаточно сильно ограничены: обычно бывает сложно найти в решаемой задаче хотя бы несколько ресурсоемких различных подзадач, которые могут быть выполнены параллельно. А случаи, когда таких подзадач десятки и более – скорее исключения, подтверждающие правило. Поэтому при распараллеливании по

управлению рассчитывать можно лишь на некоторое ускорение – не больше, чем в такое количество раз (в идеальном случае), сколько удалось выделить подзадач. На практике ускорение будет еще меньшим, определяться временем выполнения наиболее длительной подзадачи плюс накладные расходы по организации подзадач и их взаимодействия. Тем не менее, при отсутствии возможности распараллеливания по данным, на вычислительных системах с небольшим количеством параллельных вычислителей (а сейчас все даже бытовые компьютеры и мобильные устройства являются такими благодаря использованию многоядерных процессоров) такой метод распараллеливания может быть вполне успешно применен.

Возможности распараллеливания по данным выглядят гораздо более привлекательными. Объемы обрабатываемых данных в ресурсоемких задачах обычно огромны (или, по крайней мере, значительны). Если все обрабатываемые данные можно разделить на множество наборов (для типичных задач в идеале – по числу имеющихся в целевой вычислительной системе вычислителей), которые могут быть обработаны хотя бы на одном шаге обработки независимо, то, выполняя обработку всех полученных наборов данных параллельно, можно получить ускорение, в идеальном случае приближенное к числу имеющихся в целевой вычислительной системе вычислителей. Конечно, не так часто встречаются реальные прикладные задачи, в которых обрабатываемые данные можно разделить на совсем независимые группы. Для разрешения этих зависимостей приходится прибегать к синхронизации, передаче данных между вычислителями и прочим дополнительным действиям, что, конечно, снижает эффективность распараллеливания. Но, тем не менее, во многих случаях распараллеливание по данным, даже при наличии зависимости между данными, достаточно эффективно для его успешного практического применения.

При наличии зависимостей между данными, не позволяющими добиться высокой эффективности при распараллеливании обработки данных на весь набор имеющихся вычислителей, но показывающих неплохую эффективность при использовании нескольких вычислителей, хорошим решением может оказаться комбинирование подходов распараллеливания по данным и по управлению.

3. Сугубо последовательные алгоритмы

К сожалению, в ресурсоемких прикладных задачах нередко встречается ситуация, когда применяемый алгоритм является сугубо последовательным, то есть не позволяющим произвести распараллеливание ни по управлению, ни по данным. Например, когда никаких сущностей, допускающих параллельное выполнение, в этом алгоритме нет, а процесс обработки данных на каждом шаге зависит от всех обработанных данных на всех предыдущих шагах. И альтернативы применяемому алгоритму тоже нет.

В данной статье в качестве примеров будут рассмотрены алгоритмы и их программные реализации, которые были использованы авторами при

разработке среды динамического анализа программ по бинарным трассам [1]. Среда динамического анализа программ по бинарным трассам разрабатывается в Институте системного программирования им. В.П. Иванникова РАН. Целевая аппаратная платформа для выполнения среды динамического анализа – мощная персональная рабочая станция с достаточно большим количеством процессорных ядер (12 и более) и большим объемом общей оперативной памяти. Анализируемые бинарные трассы могут быть очень большого объема – до 300 Гб в сжатом виде. Среда динамического анализа представляет в помощь аналитику широкий набор алгоритмов предварительного анализа бинарных трасс, повышающих уровень представления исследуемых программ и значительно облегчающих труд аналитика [2, 3]. На огромных объемах обрабатываемых данных алгоритмы предварительного анализа могут выполняться значительное время – вплоть до нескольких суток. Поэтому, несомненно, задача ускорения работы таких алгоритмов в среде динамического анализа программ является крайне актуальной. Некоторые используемые алгоритмы хорошо могут быть распараллелены по данным. Это, например, различные алгоритмы поиска, алгоритм разметки трассы на процессы, потоки выполнения и зоны. Некоторые же алгоритмы не могут быть распараллелены из-за зависимости по данным. Рассмотрим типичные примеры таких алгоритмов подробнее.

3.1 Алгоритм построения графа потока данных

С небольшим упрощением граф потока данных представляет собой несвязанный направленный граф, в котором вершины представляют собой ячейки памяти или регистры (в дальнейшем будем называть их элементами). Дуга от одной вершины к другой означает передачу информации от соответствующего первого элемента ко второму, и дуга имеет пометку – номер шага исследуемой трассы, на котором произошла передача информации. Например, если на шаге s1 в регистр EAX было прочитано 4 байта памяти по адресу 0x12345678, на шаге s2 к регистру EAX было прибавлено значение регистра EDX, и на шаге s3 содержимое регистра EAX было записано в память по адресу 0x87654321, то соответствующий граф выглядит следующим образом, рис.1:

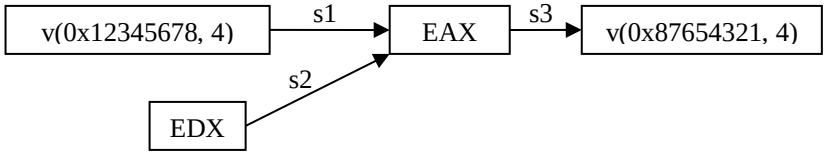


Рис. 1. Пример графа потока данных
Fig. 1. Data flow graph example

С помощью такого графа становится возможным отслеживать распространение помеченных данных, что используется в ряде других прикладных алгоритмов, таких как восстановление буфера, построение слайса и прочих.

Алгоритм построения графа потока данных работает следующим образом. На каждом шаге есть множество всех известных на этот момент элементов. Изначально это множество пусто. Рассматривается очередной шаг исследуемой программы. Если в нем есть чтение и/или запись каких-то элементов, то проверяется для каждого такого элемента, есть ли его пересечение с элементами текущего множества. Если пересечения нет, то элемент добавляется в множество. Если же пересечение есть, то возможны два варианта. Первый – когда элемент в точности соответствует какому-то элементу из множества. В этом случае множество не меняется. Второй вариант – когда элемент частично пересекается с одним или несколькими элементами множества. Тогда, если это запись элемента, то элемент включается в множество и в граф, а затем начинают рассматриваться все такие пересекаемые элементы множества. Пересекаемый элемент удаляется из множества. Если пересечение составляло лишь часть элемента, то оставшиеся части образуют новые элементы, которые вновь включаются в множество и в граф, и создаются дуги от изначального элемента множества к вновь добавленным. Если же это было чтение элемента, то читаемый элемент разбивается на несколько элементов-пересечений, и каждый такой получившийся элемент рассматривается в операции чтения по отдельности. Он уже либо не пересекается с множеством, либо в точности соответствует одному элементу из множества. И, в завершение, создаются дуги между элементами, участвующими в операции текущей инструкции.

Именно постоянное «дробление» элементов на части и их обратная «склейка» при появлении элемента, объемлющего два или более других, являются основной причиной, почему алгоритм построения графа потока данных не может быть распараллелен. Ведь если попробовать распараллелить алгоритм по данным, то во всех обработчиках, кроме первого, получится, что трасса просматривается без предыстории, и нет информации, какие элементы должны сейчас находиться в текущем множестве и какие дуги должны быть созданы на текущем шаге.

3.2 Алгоритм построения стека вызовов

Стек вызовов предоставляет аналитику в каждой точке трассы информацию о находящихся выше вызовах текущего выполняемого потока. Для каждого вызова указана точка вызова, точка возврата и находящийся выше вызов. В дальнейшем, после распознавания модулей, построения функций и восстановления параметров вызовов, эта информация будет дополнена именем функции и фактическими параметрами вызова, и такой стек вызовов будет выглядеть так же, как будто бы исследуемая программа выполняется под

отладчиком и остановлена в какой-то точке. Пример построенного стека вызовов приведен на рис. 2.

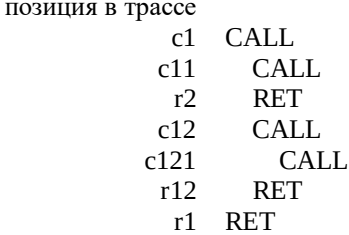


Рис. 2. Пример построенного стека вызовов
Fig. 2. Call stack example

С некоторыми упрощениями алгоритм построения стека вызовов можно описать следующим образом. Последовательно просматривается вся трасса, от первой до последней инструкции. При обнаружении на очередном шаге инструкции вызова подпрограммы, сохраняются ее параметры: шаг, идентификатор выполняющегося потока, адрес выполняющейся инструкции плюс ее размер (фактически, ожидаемый адрес инструкции после возврата из вызова). Ближайшая сохраненная ранее инструкция вызова подпрограммы с таким же идентификатором выполняющегося потока (если таковая имеется) полагается вышележащим вызовом, и эта информация также сохраняется.

Когда же на очередном шаге обнаруживается инструкция возврата из подпрограммы, предпринимается попытка сопоставить ее с одной из сохраненных ранее инструкцией вызова. Для этого определяется адрес инструкции, следующей за инструкцией возврата, и затем в обратном порядке просматриваются сохраненные ранее инструкции. Для каждой сохраненной инструкции сравнивается идентификатор выполняющегося потока с сохраненным, и, если они совпадают, проверяется, не совпадает ли адрес инструкции, следующей за инструкцией возврата, с сохраненным ожидаемым адресом инструкции после возврата. Если адреса совпадают, то инструкции вызова и возврата полагаются парными, т.е. соответствующими вызову подпрограммы и возврату из нее. Все инструкции вызовов с таким же идентификатором текущего потока, сохраненные после объявленной парной инструкции вызова (если таковые имеются), полагаются не имеющими своей парной инструкции возврата (такое часто встречается в коде ядер операционных систем) и образующими пару с той же «чужой» инструкцией возврата, для которой была только что найдена пара. Все такие пары сохраняются как результат и, в завершение, все инструкции вызова, которые на данном шаге образовали пару, удаляются из сохраненных ранее инструкций вызова. Дальнейший просмотр сохраненных ранее инструкций вызова для текущей инструкции возврата прекращается.

Если для текущей инструкции возврата парная инструкция вызова не находится (такое также часто встречается в коде ядер операционных систем), то такая инструкция возврата игнорируется.

Приведенный выше алгоритм позволяет построить хорошо согласованный стек вызовов для исследуемой программы даже и тогда, когда выполнение программы прерывается выполнением кода операционной системы.

Очевидно, что для правильного построения стека вызовов необходима история всех предыдущих вызовов, иначе, если начать обработку инструкций трассы не с самого начала, возможно не найти для текущей инструкции возврата соответствующей инструкции вызова, а также возможно не найти вышележащий вызов. Поэтому алгоритм построения стека вызовов является сугубо последовательным и распараллеливанию не подлежит.

4. Методы распараллеливания сугубо последовательных алгоритмов

Так что же делать, если другого алгоритма, решающего поставленную задачу, не существует, а имеющийся является сугубо последовательным, работает долго на значительных объемах данных, и крайне необходимо его ускорить? Казалось бы, выхода нет? Но это не всегда так, если попробовать взглянуть на поставленную задачу немного по-другому: распараллелить надо не алгоритм, а его реализацию. Пусть алгоритм так и останется последовательным, но если в его реализации в программе суметь найти возможности параллельного выполнения каких-то действий, можно попробовать добиться ускорения выполнения задачи, и, зачастую, весьма существенного.

4.1 Метод 1: все равно «разрезать», а потом «склеить»

Суть данного метода достаточно проста. Несмотря на то, что для корректного выполнения алгоритма в каждой рассматриваемой точке необходимо иметь какой-то набор данных, построенный по всем предыдущим точкам, все равно распределить обрабатываемые данные на равные непрерывные части по числу имеющихся вычислителей. Затем каждую часть данных обрабатывать параллельно так, как будто это первая часть и никакой предыстории нет. После того, как все части обработаны, надо осуществить «склейку» и коррекцию полученных результатов. Этот процесс выполняется последовательно: вначале первая часть со второй дают промежуточный результат, затем к нему также «склеивается» третья часть и так далее.

Для осуществления процесса «склейки» берется набор данных, построенный по предыдущей части, и проверяется, как он повлиял на результат, построенный по «подклеиваемой» части. Возможно, при этом придется заново обрабатывать начальную часть данных «подклеиваемой» части и корректировать полученный ранее результат. Но если эта заново обработанная начальная часть данных мала по сравнению со всем объемом

«подклеиваемой» части, и/или повторная обработка уже опирается на полученный результат и выполняется на порядок быстрее первичной, то все равно, несмотря на повторную обработку каких-то данных, можно добиться существенного ускорения выполнения всего алгоритма в целом. Последовательность процесса «склейки» также обусловлена тем, что помимо коррекции результата необходима корректировка текущего набора данных, полученного в конце «подклеиваемой» части, с учетом текущего набора данных, полученного в конце предыдущей части. Это необходимо для корректного «подклеивания» следующей части.

Очевидно, что формат представления частичного результата алгоритма, а также формат представления текущего набора данных должен позволять производить процесс их коррекции.

Вернемся к рассмотренному выше алгоритму построения графа потока данных. Разделим всю обрабатываемую трассу на равные части по числу имеющихся вычислителей, и начнем обработку каждой части с пустым множеством известных элементов. Но при этом для всех частей, кроме первой, будем дополнительно запоминать случаи, когда рассматриваемого элемента в текущем множестве элементов не было. Именно эти случаи, возможно, в дальнейшем потребуют коррекции.

Затем, после завершения обработки первой и второй части, можно начинать процесс их «склейки». Для этого начинаем по одному исследовать элементы текущего множества, образовавшегося в конце первой части. Возможны следующие три варианта:

1. Если такой элемент из множества конца первой части не входит в множество элементов, впервые появившихся во второй части, то такой элемент добавляется во множество текущих элементов конца второй части, так как, получается, что такой элемент во второй части не участвовал ни в одной операции.
2. Если такой элемент из множества конца первой части полностью совпадает с каким-то элементом из множества впервые появившихся элементов во второй части, данные дуг этих элементов объединяются, так как это один и тот же элемент.
3. И, гораздо более сложный случай, когда элемент из множества конца первой части пересекается с одним или более элементом из множества впервые появившихся элементов во второй части. Не будем приводить здесь полностью весь алгоритм работы, отметим лишь, что в этом случае требуется корректировка результата второй части, в котором присутствуют пересекаемые элементы, вплоть до того момента, когда эти пересекаемые элементы не были вновь объединены операцией записи объемлющего элемента, равного элементу из множества впервые появившихся элементов во второй части.

После завершения работы над «склейкой» второй части можно приступать к аналогичному процессу по отношению к полученному результату «склейки» второй части и модифицированному множеству текущих элементов конца второй части, с одной стороны, с результатом третьей части, с другой. И так далее, вплоть до «склейки» последней части. В итоге получается тот же результат, что и был бы получен в случае последовательной обработки всего набора данных.

А вот распараллелить алгоритм построения стека вызовов методом разрезания и склеивания не представляется возможным. Все дело в том, что при завершении обработки первой части с большой вероятностью текущий стек незавершенных вызовов будет непустым. Более того, также с немалой вероятностью вызовы, находящиеся в нем, не будут полностью удалены из стека до конца обрабатываемой трассы. Типичный пример – вызов функции main(). Это приводит к тому, что вторую и последующие части придется фактически просматривать заново для корректировки результата. Очевидно, что никакого выигрыша от такого «распараллеливания» получено быть не может.

4.2 Метод 2: выделить независимые сущности в массиве обрабатываемых данных и обработать их независимо

Суть этого метода в том, что для параллельной обработки весь массив данных не разбивается на равные части из подряд идущих данных, а среди обрабатываемых данных выделяются части, возможно, разных размеров, каждая часть может состоять из множества разрозненных данных, но такие, что все эти части могут быть обработаны параллельно.

Вернемся к алгоритму построения стека вызовов. Очевидно, что стек вызовов строится независимо для каждого потока, присутствующего в исследуемой трассе. Поэтому можно для каждого потока параллельно получить множество инструкций, относящихся к этому потоку, обработать их, и получить частичный результат, относящийся к этому потоку. А затем, уже последовательно, объединить все полученные частичные результаты в итоговый, упорядочивая в нем все вызовы, полученные в частичных результатах, в порядке появления их в трассе.

Предложенный метод имеет существенный недостаток. Эффективность его применения сильно зависит от того, насколько выделенные независимые сущности в массиве обрабатываемых данных отличаются друг от друга по объему данных в каждой сущности. Например, если в исследуемой трассе имеется поток, занимающий большую часть трассы (или даже всю, если, например, исследуется однопотоковая встроенная система), то эффективность такого распараллеливания будет крайне низка.

Но в случаях, когда удается выделить хотя бы несколько сравнимых по объему существенных объемов данных, применение данного метода может дать весьма хороший результат.

4.3 Метод 3: распараллелить предварительную обработку данных

Попробуем взглянуть на задачу ускорения работы реализации какого-то сугубо последовательного алгоритма немного по-другому. Пусть алгоритм сугубо последователен, и его реализация тоже не может быть распараллелена предложенными выше методами. Но, обычно, в реализации каждого алгоритма, помимо действий по реализации собственно шагов алгоритма, есть какая-то предварительная работа. Например, в реализации алгоритма построения стека вызовов прежде, чем поместить инструкцию вызова в текущий стек вместе с ожидаемым адресом инструкции после возврата, нужно найти эту очередную выполненную инструкцию вызова и определить ее параметры. А это тоже занимает немалое время. Так почему бы не попробовать распараллелить эту работу, и предоставить последовательной части реализации алгоритма данные уже в предварительно обработанном виде, оптимизированном для их быстрой обработки этой последовательной частью?

Для этого в реализации алгоритма выделяется два типа рабочих процессов. Первый тип, существующий в единственном экземпляре, выполняет исключительно последовательную часть алгоритма. Рабочие процессы второго типа организуются в количестве имеющихся в системе аппаратных вычислителей. Для взаимодействия между процессами организуется очередь заданий. Каждое задание представляет собой некий объем данных для предварительной обработки. Рабочий процесс второго типа берет себе очередное задание из очереди, обрабатывает его, формирует набор предварительно обработанных данных, помечает задание как выполненное и берет себе следующее свободное задание из очереди. Рабочий процесс первого типа ждет готовности задания в голове очереди, по его готовности удаляет его из очереди и начинает его обработку. Схема работы очереди представлена на рис. 3.

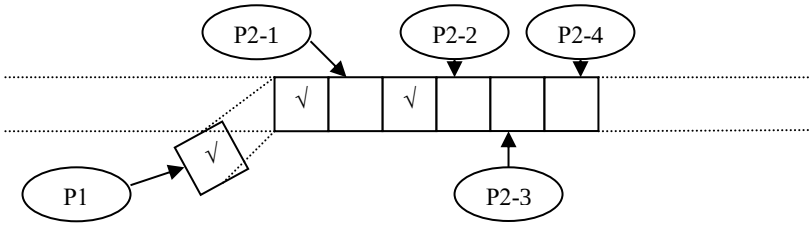


Рис. 3. Схема работы очереди заданий
Fig. 3. Job queue working diagram

Очевидно, что общая производительность такой реализации алгоритма не будет выше производительности того единственного рабочего процесса, реализующего последовательную часть на уже подготовленных данных. И, если он не будет успевать обрабатывать все подготовленные данные, очередь будет сильно разрастаться. Подробнее организация очереди, балансировка нагрузки и распределение вычислительных ресурсов между рабочими процессами первого и второго типа будут рассмотрены ниже.

4.4 Комбинирование методов 2 и 3

Вернемся к методу 2, когда в массиве обрабатываемых данных можно выделить параллельно обрабатываемые сущности, но одна из таких сущностей занимает существенный объем от общих данных. Например, в обрабатываемой трассе один из потоков занимает 60% от всего объема трассы. Тогда, если распараллеливать обработку такой трассы по методу 2, иметь более чем два параллельно выполняющихся обработчиков не нужно – пока один из них будет обрабатывать самый большой поток, второй обработчик должен обработать все остальные. Таким образом, максимальное ускорение, которое возможно получить в данной ситуации – это менее чем в два раза.

Если же распараллелить предварительную обработку данных по методу 3, то единственный рабочий процесс первого типа, осуществляющий последовательную часть работы алгоритма, может не справляться с потоком предварительно обработанных данных, производимых множеством рабочих потоков второго типа. Но ничто не мешает применить одновременно оба метода. Вначале, следуя методу 2, выделяются независимые сущности в массиве данных, и пусть среди них будут такие, которые занимают существенный объем от общих данных. Вначале выясняем, по объему данных, сколько таких сущностей имеет смысл обрабатывать параллельно (подробнее этот процесс будет описан ниже, в главе про балансировку вычислительной нагрузки). И, если таких сущностей оказывается более одной, но существенно менее имеющихся в вычислительной системе вычислителей, к каждой такой параллельно обрабатываемой сущности далее применяем метод 3. То есть организуем несколько (по числу сущностей) очередей, в каждой из которых есть один рабочий поток первого типа и один или более рабочих потоков второго типа. При этом общее число рабочих потоков второго типа должно не превышать число имеющихся вычислителей. Когда какая-то очередь заканчивает свою работу, другие очереди могут увеличить число своих рабочих потоков второго типа.

Вернемся к примеру, когда один из потоков занимает 60% от всего объема трассы. Допустим, что выполнение происходит на 12-ядерной рабочей станции. Тогда, применяя метод 2, мы получим только 2 занятых ядра (а в конце работы и вовсе будет только одно). 10 ядер будут простаивать. Применение метода 3 даст следующую картину: 1 рабочий процесс первого типа и 11 рабочих процессов второго типа. Справится ли один процесс с

последовательной обработкой данных, предоставленных одиннадцатью процессами предварительной обработки? Как показывает практика, в алгоритме построения стека – нет. И значительная часть ядер опять будет простаивать. А вот применение комбинированного подхода дает уже гораздо более сбалансированную картину: 2 потока первого типа, у каждого – по 5 потоков второго типа. И (опять же в нашем примере – в случае алгоритма построения стека) загрузка всей системы становится близка к 100%, все имеющиеся вычислительные ресурсы используются эффективно, общее выполнение алгоритма ускоряется в несколько раз.

5. Балансировка вычислительной нагрузки

В отличие от простого распараллеливания по данным, когда весь объем данных делится на равные части и распределяется по имеющимся вычислителям, при использовании описанных выше методов распараллеливания сугубо последовательных алгоритмов, гораздо более сложных, с различными типами взаимодействующих вычислительных процессов, задача правильной балансировки вычислительной нагрузки между вычислительными процессами становится крайне актуальной и совсем не тривиально решаемой. От правильной балансировки вычислительной нагрузки зависит успешное решение задачи в целом – ускорение выполнения реализации алгоритма. Несбалансированная вычислительная нагрузка может свести на нет все усилия по распараллеливанию.

Балансировка вычислительной нагрузки может производиться как статически, путем первоначального распределения данных, так и динамически – во время выполнения реализации алгоритма, путем перераспределения части данных и/или путем временного приостанавливания отдельных вычислительных процессов. Предпочтительнее, по возможности, использовать оба подхода.

Рассмотрим различные методы балансировки вычислительной нагрузки, которые могут быть применены в зависимости от используемого метода распараллеливания сугубо последовательных алгоритмов.

5.1 Статическая балансировка с весами

При использовании метода 1 (все равно «разрезать», а потом «склеить») статическое распределение данных между вычислителями должно быть, казалось бы, простое – всем поровну, как при простом распараллеливании. Но это не так, если учесть, что после процесса параллельной обработки всех данных предстоит еще процесс «склейки», а он будет производиться последовательно от первой части к последней. Для того чтобы начать процесс «склейки», необходима готовность первой и второй части, а третья и последующие части могут продолжать при этом обрабатываться. Только после того, как будет завершен процесс «склейки» первой и второй части, нужна будет готовность третьей.

Таким образом, идеальным распределением нагрузки будет такое, когда каждая последующая часть будет готова как раз к тому моменту, когда будет завешен процесс «склейки» предыдущей части. Для достижения этого каждой части нужно назначить вес обрабатываемых данных. Первая и вторая части получают наименьший вес, далее – равномерное нарастание веса вплоть до максимального у последней части. Как правильно выбрать соотношение минимального и максимального веса – зависит от алгоритма и от типичного времени для процесса «склейки» и, видимо, должно подбираться экспериментально. Для алгоритма построения графа потока данных эти значения оказались следующими (при наличии достаточного количества вычислительных ядер): от 0.7 для первой и второй части до 1 для последней.

5.2 Статическая балансировка упорядочиванием обрабатываемых данных

При использовании метода 2 балансировка вычислительной нагрузки также производится статически. Для начала надо оценить объем обрабатываемых данных в каждой независимой сущности и упорядочить сущности по уменьшению этого объема. Именно в этом порядке сущности будут выдаваться вычислителям для обработки: сначала те, которые имеют больший объем данных, затем – меньший. Вычислительный процесс берет очередную сущность с головы очереди, обрабатывает ее, и берет следующую. Сущности с наибольшими объемами данных таким образом будут обработаны в первую очередь, а конце работы останется набор сущностей с наименьшими объемами данных, что даст и в конце работы алгоритма равномерную загрузку вычислительных процессов.

Кроме того, перед началом работы следует оценить необходимое количество вычислительных процессов. При более-менее равномерном распределении объема данных между сущностями, и когда таких сущностей достаточно много, вычислительных процессов должно быть столько, сколько в вычислительной системе имеется аппаратных вычислителей. Но если имеется сущность (или несколько сущностей, но меньше, чем число аппаратных вычислителей), объем данных которой (которых) имеет в общем объеме данных превалирующее значение, то необходимо ограничить число вычислительных процессов. Так как все равно при таком неравномерном распределении данных общее время работы алгоритма будет определяться временем работы вычислительного процесса с сущностью с наибольшим объемом данных, а другие вычислительные процессы, обработав остальные сущности, затем будут простаивать. В то время как изначальное снижение числа вычислительных процессов снизит их конкуренцию за общие ресурсы (например, доступ к памяти или файлам на жестком диске) и, таким образом, увеличит скорость работы вычислительного процесса с сущностью с наибольшим объемом данных.

Для оценки количества необходимых вычислительных процессов предлагается использовать следующую формулу. Пусть имеется *n* сущностей, $n > 1$ и $p_i, i = 1..n$ – объем обрабатываемых данных *i*-той сущности, такой, что $p_1 \geq p_2 \geq \ldots \geq p_n$. Тогда минимальное значение *k*, при котором будет выполняться условие $\sum_{i=1}^{k-1} p_i \geq \sum_{i=k}^{n} p_i$ и есть число необходимых вычислительных процессов для параллельной обработки всех сущностей.

5.3 Динамическая балансировка очереди заданий

При использовании метода 3 балансировка вычислительной нагрузки производится динамически, путем управления очередью заданий и количеством активных рабочих процессов второго типа. Все данные для предварительной обработки делятся на сравнительно небольшие подряд идущие куски. Рабочих процессов второго типа организуется столько, сколько в вычислительной системе имеется аппаратных вычислителей. Рабочий процесс второго типа в начале своей работы, или закончив свое предыдущее задание, обращается к очереди за очередным заданием. Ему оформляется очередной, пока еще не выданный на обработку кусок данных в виде задания, это задание помещается в конец очереди и выдается на обработку рабочему процессу. Рабочий процесс, выполнив задание, помечает его в очереди как выполненное и обращается за новым.

Рассмотрим теперь, что будет происходить, когда рабочий процесс первого типа, собственно осуществляющий обработку предварительно подготовленных данных согласно алгоритму, не будет успевать обрабатывать подготовленные ему данные и будет выбирать с головы очереди выполненные задания медленнее, чем очередь будет пополняться с хвоста рабочими процессами второго типа. Очевидно, что при таком развитии событий очередь будет разрастаться, занимая ресурсы оперативной памяти. А когда все данные будут предварительно подготовлены и помещены в очередь, рабочие процессы второго типа прекратят свою работу, и останется выполняться только один рабочий процесс первого типа, который будет долго еще разбирать подготовленные ему данные. Помимо излишнего потребления оперативной памяти это также приведет к тому, что, пока все рабочие процессы второго типа работали, они конкурировали, в том числе и с рабочим процессом первого типа, за системные ресурсы. И, таким образом, рабочий процесс первого типа, от которого собственно зависит время выполнения всего алгоритма, в условиях конкуренции работал гораздо медленнее, чем мог бы работать при меньшей конкуренции.

Для решения этой проблемы предлагается достаточно простое решение – ограничить размер очереди по сумме выданных и выполненных заданий. Когда рабочий процесс второго типа обращается к очереди за следующим

заданием, проверяется, сколько сформированных заданий (в сумме как выполненных, так и тех, над которыми еще ведется работа другими процессами второго типа) находится в очереди. Если их меньше установленного максимального размера очереди, очередное задание выдается обратившемуся процессу. Если же максимальный размер очереди достигнут, то обратившийся рабочий процесс второго типа приостанавливается.

Когда рабочий процесс первого типа вынимает из головы очереди выполненное задание с предварительно подготовленными данными, тем самым уменьшая размер очереди на единицу, проверяется, нет ли приостановленных рабочих процессов второго типа. И, если есть, то любой один из них запускается, и ему выдается очередное задание. Таким способом достигается, что в каждый момент времени выполняется ровно столько рабочих процессов второго типа, чтобы создать непрерывную загрузку рабочему процессу первого типа, и не более того. За счет этого уменьшается конкуренция процессов за ресурсы, и рабочий процесс первого типа, а вместе с ним и весь алгоритм, выполняются максимально быстро.

Максимальный размер очереди должен быть таким, чтобы, с одной стороны, обеспечить непрерывную загрузку подготовленными данными рабочего процесса первого типа, а с другой стороны – не потреблять неоправданно системные ресурсы. Как показала практика, установка максимального размера очереди как трехкратное количество рабочих процессов второго типа полностью соответствует этим критериям.

Обратная ситуация, когда рабочий процесс первого типа успевает работать быстрее, чем рабочие процессы второго типа подготавливают ему предварительно обработанные данные, не требует никакой балансировки. Притом, что рабочих процессов второго типа должно быть столько, сколько в вычислительной системе имеется аппаратных вычислителей, следует, что и загрузка системы в таком случае будет близка к 100%.

5.4 Комбинированная балансировка при использовании комбинированных методов

Очевидно, что при использовании комбинирования методов распараллеливания 2 и 3, балансировка вычислительной нагрузки также должна производиться комбинировано. Вначале необходимо статически, точно так же, как для метода 2, отсортировать сущности по объему данных и определить необходимое количество независимых вычислительных процессов. Затем, в каждом таком процессе, создать очередь заданий и динамически управлять ей, как описано выше. Каждая такая очередь должна управляться независимо от других, с одним исключением: если в одной из очередей появляются приостановленные рабочие процессы второго типа, а в другой – нет, возможно организовать передачу приостановленного процесса в другую очередь, но с обязательным возвращением его обратно в случае, если

потом, возможно, ситуация изменится, и размер первой очереди начнет уменьшаться.

6. Результаты применения методов распараллеливания сугубо последовательных алгоритмов

В данной главе приведены временные характеристики выполнения алгоритма построения графа потока данных и алгоритма построения стека вызовов на различных обрабатываемых данных и при различном количестве предоставляемых программе аппаратных процессорных ядер. Все измерения производились на персональной рабочей станции следующей конфигурации: 2 процессора Intel Xeon E5-2690 v2, по 10 вычислительных ядер в каждом, поддержка гипертрединга включена, размер оперативной памяти – 192 Гб.

График времени выполнения алгоритма построения графа потока данных приведен на рис. 4 для одной трассы архитектуры x86, размера 2.6 Гб (370 398 536 шагов).

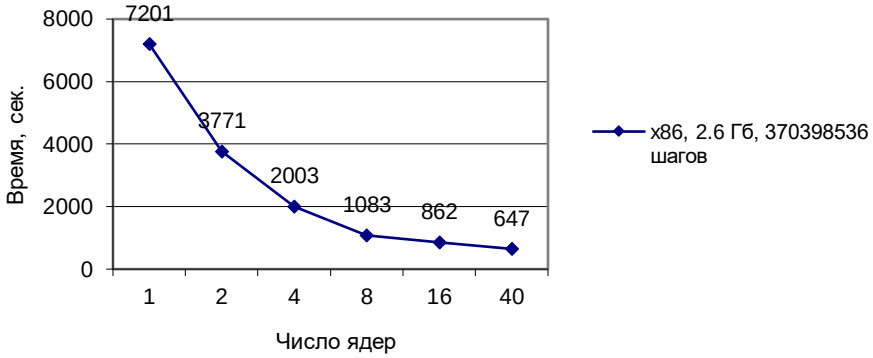


Рис. 4. Время выполнения алгоритма построения графа потока данных
Fig. 4. Dependency graph building time

На трассах для других архитектур и объемов график времени выполнения алгоритма ведет себя подобным образом. Параллельная реализация алгоритма построения графа потока данных, выполненная по методу 1, демонстрирует хорошую масштабируемость, давая в итоге при задействии всех 20 аппаратных процессорных ядер (40 – это с гипертредингом) ускорение более чем в 10 раз по сравнению с последовательным выполнением.

Далее приведены графики времени выполнения алгоритма построения стека вызовов для различных трасс. При распараллеливании реализации этого алгоритма вначале производится оценка имеющихся потоков в исследуемой трассе по методу 2. Затем выбирается метод распараллеливания: 2, 3 или их

комбинация. Для каждого метода в качестве примера приводится по одной трассе.

На рис. 5 приведены нормированные (время выполнения последовательной версии алгоритма взято за 100) графики времени выполнения алгоритма построения стека вызовов для следующих трасс:

- Та же трасса, что и на рис. 4, архитектуры x86, размера 2.6 Гб (370 398 536 шагов). Здесь и далее также приведем еще один параметр – в трассе имеется 192 потока выполнения, для обработки которых можно задействовать 61 параллельный вычислитель. Таким образом, эта трасса имеет хорошие возможности для параллельной обработки, и при распараллеливании реализации алгоритма применялся метод 2.
- Трасса архитектуры MIPS, размера 3.3 Гб (1 933 615 005 шагов), в трассе имеется единственный поток выполнения, поэтому применение к ней метода 2 невозможно, при распараллеливании реализации алгоритма применялся метод 3.
- Трасса архитектуры ARM, размера 1.9 Гб (641 679 867 шагов), в трассе имеется 96 потоков выполнения, но для обработки их можно задействовать только 2 параллельных вычислителей, так как один из потоков занимает более половины трассы. Поэтому при распараллеливании реализации алгоритма применялась комбинация методов 2 и 3.

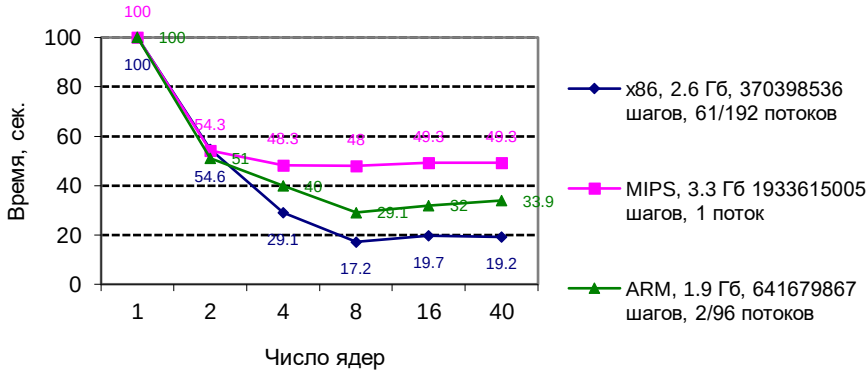


Рис. 5. Время выполнения алгоритма построения стека вызовов
Fig. 5. Call stack building time

Трасса x86, имеющая множество равнообъемных потоков выполнения, показывает хорошие результаты распараллеливания, уменьшая время своего выполнения вплоть до использования 8 процессорных ядер. Дальнейшее увеличение количества процессорных ядер никакого увеличения скорости работы уже не приносит. По-видимому, дело здесь уже в производительности

жесткого диска, с которого и читается сама трасса, и пишется частичный результат для каждого потока. Тем не менее, увеличение скорости работы более чем в 5 раз по сравнению с последовательным вариантом для выполнения такого алгоритма – мы полагаем, очень хороший результат.

Среди приведенных примеров тяжелее всего с распараллеливанием обработки трассы MIPS, имеющей только один поток. Применяется метод 3, и на единственный рабочий процесс первого типа при увеличении числа процессорных ядер пропорционально увеличивается число рабочих процессов второго типа, осуществляющих предварительную подготовку данных. Как видно по графику, использование их в количестве более четырех уже не приводит к росту производительности – единственный рабочий процесс первого типа уже не справляется с получающейся нагрузкой. Тем не менее, увеличение скорости работы более чем в 2 раза по сравнению с последовательным вариантом достигнуто даже в таком, совсем плохо подходящем для распараллеливания случае.

А вот если в трассе появляется возможность использования еще хотя бы одного рабочего процесса первого типа (трасса ARM), и распараллеливание выполняется комбинированным методом, то и результат получается гораздо лучше – увеличение скорости работы вплоть до использования 8 процессорных ядер и общее ускорение почти в 4 раза по сравнению с последовательным вариантом.

7. Заключение

Основная цель данной работы – показать, что использование в программной реализации сугубо последовательного алгоритма не означает неизбежность его последовательного выполнения. Предложенные в статье методы распараллеливания реализаций таких алгоритмов и балансировки получающейся вычислительной нагрузки могут поспособствовать созданию эффективной параллельной программы, полностью использующей предоставленные ей аппаратные возможности современных вычислительных систем.

Список литературы

[1]. В.А. Падарян, А.И. Гетьман, М.А. Соловьев, М.Г. Бакулин, А.И. Борзилов, В.В. Каушан, И.Н. Ледовских, Ю.В. Маркин, С.С. Панасенко. Методы и программные средства, поддерживающие комбинированный анализ бинарного кода. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 251-276. DOI: 10.15514/ISPRAS-2014-26(1)-8.

[2]. В.А. Падарян. О представлении результатов обратной инженерии бинарного кода. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 31-42. DOI: 10.15514/ISPRAS-2017-29(3)-3.

[3]. Alexander Getman, Vartan Padaryan, Mikhail Solovyev. Combined approach to solving problems in binary code analysis. Proceedings of the 9th International Conference on Computer Science and Information Technologies (CSIT), 2013, pp. 295-297.

Parallelization of implementations of purely sequential algorithms

[1] *A.B. Bugerya <shurabug@yandex.ru>*
[2] *E.S. Kim <eugene.kim@ispras.ru>*
[2] *M.A. Solovev <icee@ispras.ru>*
[1] *Keldysh Institute of Applied Mathematics of the Russian Academy of Sciences, Miusskaya sq., 4, Moscow, 125047, Russia*
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*

Abstract. The work is dedicated to the topic of parallelizing programs in especially difficult cases - when the used algorithm is purely sequential, there are no parallel alternatives to the algorithm used, and its execution time is unacceptably high. Various parallelization methods for software implementations of such algorithms and resulting computational load balancing are considered, allowing to obtain significant performance acceleration for application programs using purely sequential algorithms. The above methods are illustrated by the practice of their application to two algorithms used in a dynamic binary code analysis toolset. The main goal of this paper is to show that the use of a purely sequential algorithm in a software implementation does not necessarily imply inevitability of its sequential execution. The proposed methods of parallelizing implementations of such algorithms and balancing the resulting computational load can help to develop efficient parallel program that fully utilize the hardware capabilities of modern computing systems.

Keywords: parallel programming; program parallelization; computational load balancing.

DOI: 10.15514/ISPRAS-2018-30(2)-2

For citation: Bugerya A.B., Kim E.S., Solovev M.A. Parallelization of implementations of purely sequential algorithms. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 2, 2018, pp. 25-44 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-2

References

[1]. V.A. Padaryan, A.I. Getman, M.A. Solovyev, M.G. Bakulin, A.I. Borzilov, V.V. Kaushan, I.N. Ledovskich, U.V. Markin, S.S. Panasenko. Methods and software tools for combined binary code analysis. Trudy ISP RAN/Proc. ISP RAS, 2014, vol. 26, issue 1, pp. 251-276 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-8.

[2]. V.A. Padaryan. On representation used in the binary code reverse engineering. Trudy ISP RAN/Proc. ISP RAS, 2017, vol. 29, issue 3, pp. 31-42 (in Russian). DOI: 10.15514/ISPRAS-2017-29(3)-3.

[3]. Alexander Getman, Vartan Padaryan, Mikhail Solovyev. Combined approach to solving problems in binary code analysis. Proceedings of the 9th International Conference on Computer Science and Information Technologies (CSIT), 2013, pp. 295-297.

Преобразование типизированных функций в реляционную форму

П.А. Лозов <lozov.peter@gmail.com>
Д.Ю. Булычев <dboulytchev@math.spbu.ru>
Санкт-Петербургский государственный университет,
198504, Россия, Санкт-Петербург, Университетский пр., д. 28

Аннотация. Реляционное программирование является подходом, позволяющим исполнять программы в различных "направлениях" для получения различных сценариев поведения по одной реляционной спецификации. В данной статье рассмотрена задача автоматического преобразования функциональных программ в реляционные. Представлен метод преобразования типизированных функций в реляционную форму, а также доказательство его статической и динамической корректности. Также в статье обсуждаются ограничения предложенного метода, представлена реализация метода для подмножества языка OCaml и проведена оценка эффективности метода на ряде реалистичных примеров.

Ключевые слова: функциональное программирование; реляционное программирование; генерация программ.

DOI: 10.15514/ISPRAS-2018-30(2)-3

Для цитирования: Лозов П.А., Булычев Д.Ю. Преобразование типизированных функций в реляционную форму. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 45-64. DOI: 10.15514/ISPRAS-2018-30(2)-3

1. Введение

Реляционное программирование основано на построении программы в виде набора отношений [1]. Реляционная программа может быть исполнена в различных "направлениях", то есть независимо от того, какие из аргументов отношения известны, а какие необходимо найти. Это делает возможным, например, вычисление обратных функций. Хотя многие языки логического программирования, такие как Prolog, Mercury [2] или Curry [3], позволяют использовать некоторые реляционные эффекты, был создан miniKanren [4] – специальный язык для реляционного программирования. Изначально данный язык был небольшим предметно-ориентированным расширением языка Scheme/Racket, его минимальная реализация [5] содержала менее ста строк кода. Впоследствии miniKanren нашел применение, будучи встроенным во

многие языки программирования, среди которых Haskell, Standard ML и OCaml.

Непосредственная разработка реляционных программ является сложной задачей, требующей от разработчика определенных навыков. Однако во многих случаях требуемая реляционная программа может быть получена из некоторой функциональной программы автоматически. Таким образом, появляется задача автоматического преобразования функциональных программ в реляционные. Отметим, что единственный существующий подобный метод **Unnesting** [6] не полон и не был реализован.

Основным вкладом данной статьи является метод реляционного преобразования, который может быть применён к типизированным программам общего вида. Для описания этих программ используется компактный ML-подобный язык (является подмножеством OCaml), оснащенный системой типов Хиндли-Милнера с let-полиморфизмом [7].

Оставшаяся часть статьи организована следующим образом. В разд. 2 приведен краткий обзор схожих работ. Разд. 3 посвящена реляционному программированию и языку miniKanren. Разд. 4 содержит описание синтаксиса, правил вывода типов и операционной семантики как для ML-подобного языка, так и для его реляционного расширения. В разд. 5 представлены формальные правила для реляционного преобразования, а также доказательства сохранения типизации и семантики. В разд. 6 представлены реализация описанного метода и несколько примеров результата преобразования.

2. Схожие работы

Взаимодействие декларативных языков программирования (а реляционное программирование является декларативным) с языками более прагматичными (например, Java или OCaml) является естественной задачей. Успешное решение это задачи позволяет совместить выразительность декларативного языка с оптимизированностью и обширной функциональностью императивного или функционального языка. В качестве примера можно упомянуть работы [8; 9], где для описания ограничений на метамодель REAL [10] используется декларативный язык OCL (Object Constraint Language), который позволил кратко и понятно описать необходимые ограничения, однако оказался непригоден для работы с реальными данными. Поэтому ограничения на языке OCL были преобразованы в скрипты, написанные на императивном языке JavaScript.

В контексте данной работы наиболее показательным декларативным языком является Prolog. Во-первых, между ним и языком miniKanren много общего; во-вторых, задача взаимодействия этого языка с другими хорошо исследована. Прежде всего, для языка Prolog существуют методы компиляции в более низкоуровневые языки, например в язык C [11; 12]. В [14] представлен метод преобразования из Prolog в Java, позволяющий увеличить эффективность

исполнения декларативной программы, включающий метод декомпиляции Java Bitecode в Prolog [14]. В [15] представлен подход, встраивающий Prolog в Java, в [16] аналогично, Prolog встроен в C#. Тем не менее, не смотря на все эти подходы, проблема взаимодействия декларативных и императивных языков программирования далека до полного разрешения.

В случае взаимодействия функционального языка с реляционным можно выделить обратную задачу: преобразование функциональных программ в реляционную форму. Решение данной задачи позволит, с одной стороны, использовать привычный язык программирования (в нашем случае OCaml) для описания функций, а, с другой стороны, исполнять функциональные программы реляционно (в частности, моделировать вычисление обратных функций). Для данной задачи было предложено решение Unnesting [6], которое, однако, рассматривает только случай нетипизированных программ и работает для специальных примеров. Более того, оно не был реализовано.

3. Реляционное программирование и язык miniKanren

Основной особенностью языка miniKanren [1; 6] является отсутствие различий между аргументами и результатом, что позволяет исполнять программы в различных “направлениях”. Такой подход имеет практическое значение: некоторые задачи формулируются гораздо проще, если рассматривать их как запросы к реляционной программе. Существует целый ряд примеров, подтверждающих это наблюдение. К примеру, задача вывода типов для просто-типизированного лямбда-исчисления или задача о выявлении населенности какого-либо типа могут быть сформулированы в виде запросов к более простой в реализации реляционной программе проверки корректности типов. Другим примером может послужить задача генерации “квайнов” [17] – программ, результатом исполнения которых являются они сами. Данная задача может быть представлена как запрос к реляционному интерпретатору, также более простому в сравнении с изначальной задачей. Наконец, генератор всех перестановок элементов данного списка выразим в виде запроса к реляционной сортировке списка [18]. В контексте данной статьи будет использоваться конкретная реализация языка miniKanren – DSL на основе Objective Caml4, называемый OCanren [19]. Данный язык соответствует оригинальной реализации miniKanren [5], но OCanren дополнен конструкцией “Disequality constraint” [20].

Основной синтаксической единицей как реляционного языка miniKanren, так и языка OCanren является цель (*goal*). Для построения элементарных целей используется синтаксическая унификация (t1 ≡ t2) и обратная к ней операция disequality constraint ($t_1 \not\equiv t_2$). Для построения сложных целей используются дизъюнкция ($g_1 \vee g_2$), конъюнкция ($g_1 \wedge g_2$) и операция введения “свежей переменной” (**`fresh`** (*x*) *g*). Результат выполнения реляционной программы представляется в виде потока данных, из которого можно запрашивать

решения. В качестве примера рассмотрим реляционную программу сложения чисел Пеано:

```
1 type num = O | S of num
2
3 let rec add a b c =
4   (a ≡ O ∧ b ≡ c) ∨
5   (fresh (a' c')
6      (a ≡ S a') ∧
7      (add a' b c') ∧
8      (c ≡ S c'))
```

Интерпретировать отношение "**add a b c**" нужно как следующее утверждение: "сумма **a** и **b** равняется **c**". Действительно, в случае, когда **a** равняется нулю, **b** в точности совпадает с **c** (строка 4). Также **a** можно представить в виде **S a'** (строка 7) – для этого понадобится "свежая" переменная **a'** (строка 5). Также понадобится дополнительная переменная **c'** для обозначения суммы **a'** и **b**, что выражается в виде "**add a' b c'**" (строка 8). Остается указать, что с должно быть на единицу больше, чем **c'** (строка 9).

```
fresh (x) add (S O) (S O) x ⇒ [x = S (S O)]
(a)

fresh (x) add (S (S O)) x (S (S (S O))) ⇒ [x = S O]
(b)

fresh (x y) add x y (S (S O)) ⇒ [ x = O,      y = S (S O);
                                  x = S O,    y = S O;
                                  x = S (S O), y = O ]
(c)

fresh (x) add (S(S (S O))) x (S (S O)) ⇒ []
(d)
```

Рис. 1. Примеры использования отношения **add**
Fig. 1. Examples of using relation **add**

Рассмотрим несколько различных целей, построенных с помощью отношения **add**, и изображенных на рис. 1. В каждом примере отношение принимает следующие возможные аргументы: выражения-константы, "свежие" переменные, внедренные с помощью конструкции **fresh**. Прежде всего отношение **add** можно использовать для сложения двух чисел. Для этого в качестве аргументов ему необходимо передать эти числа и "свежую"

переменную. В этом случае результатом вычисления цели будет поток, содержащий сумму этих чисел. На рис. 1a приведен пример суммы двух единиц. Помимо сложения с помощью данного отношения можно произвести вычитание, передав уменьшаемое в качестве третьего аргумента, вычитаемое в качестве первого аргумента и "свежую" переменную, олицетворяющую разность, в качестве второго аргумента. На рис. 1b приведен пример разности чисел 3 и 2. Также отношение **add** позволяет сгенерировать все пары слагаемых для фиксированной суммы. Для достижения этой цели передадим отношению две “свежих” переменных и число. Полученный после вычисления цели поток будет содержать все пары корректных слагаемых. На рис. 1c приведён пример генерации слагаемых для суммы, равной двум. Наконец, данное отношение позволяет выявить некорректные аргументы, ведь получение в качестве результата вычисления цели пустого потока сигнализирует об отсутствии правильных решений. На рис. 1d приведен пример попытки прибавить к трем неотрицательное число и получить два.

4. Входной язык и его реляционное расширение

Рассмотрим формальное описание ML-подобного функционального языка, используемого в качестве входного языка для реляционного преобразования. Формальное описание состоит из синтаксиса, правил вывода типов и семантики.

$$
\begin{array}{lcl}
\mathcal{E} & = & x \\
 & & \lambda x.e \\
 & & e_1\ e_2 \\
 & & C^n(e_1, \ldots, e_n) \\
 & & \underline{\texttt{true}} \\
 & & \underline{\texttt{false}} \\
 & & \underline{\texttt{let}}\ x = e_1\ \underline{\texttt{in}}\ e_2 \\
 & & \underline{\texttt{let}}\ \underline{\texttt{rec}}\ x = e_1\ \underline{\texttt{in}}\ e_2 \\
 & & e_1 = e_2 \\
 & & \underline{\texttt{match}}\ e\ \underline{\texttt{with}}\ \{p_i \to e_i\} \\
 & & \\
\mathcal{P} & = & C^n(x_1, \ldots, x_n)
\end{array}
$$

Рис. 2. Синтаксис входного языка
Fig. 2. Syntax of input language

Синтаксис исходного функционального языка показан на рис. 2. Данный язык является расширением лямбда-исчисления конструкторами с фиксированной размерностью C^n, двумя предопределенными конструкторами **true** и **false**, операцией синтаксического сравнения "=", шаблонами *p* и конструкциями

сопоставления с образцом, а также выражениями для рекурсивных/нерекурсивных let-ссылок.

При использовании сопоставления с образцом доступны только шаблоны вида $C^n(x_1, ..., x_n)$. Данное ограничение несущественно, так как шаблоны общего вида выразимы с помощью описанных выше. Также запрещен специальный шаблон wildcard (обозн. "_"), позволяющий игнорировать сопоставляемую ему часть выражения.

Данный язык оснащен системой вывода типов Хиндли-Милнера, правила которой описаны в прил. А. Система вывода типов поддерживает типовые переменные, функциональные типы, а также набор неявно определенных алгебраических типов данных T^k, причем каждый конструктор C^n принадлежит ровно одному типу, и конструкторы **`true`** и **`false`** принадлежат выделенному алгебраическому типу **`bool`**.

Семантика данного языка представлена в прил. В и является системой переходов между состояниями. Отношение перехода

$$\langle S, e\rangle \to \langle S', e'\rangle$$

описывает один шаг вычисления выражения e в стеке контекстов S, после которого будет получено новое выражение e' с обновленным стеком контекстов S'. Контекст представляет из себя выражение с уникальной дырой; неформально говоря, стек контестов описывает путь вычисления выражения от внешнего уровня до места, где в текущий момент остановлено вычисление. Для контекста C и выражения e обозначим $C[e]$ – полное выражение без дыр, полученное путем подстановки e вместо уникальной дыры в C. Для состояния $\langle C_1 : \ldots : C_n, e\rangle$ полным выражением является $C_n[...[C_1[e]]...]$, которое является промежуточным результатом вычисления.

Наконец, выражение e вычисляется к результирующему значению v если

$$\langle \varepsilon, e\rangle \to^* \langle \varepsilon, v\rangle,$$

где ε – пустой стек, "$\to^*$" – рефлексивно-транзитивное замыкание отношения "$\to$".

$$\begin{array}{ll} \mathcal{E} \mathrel{+}= & \underline{\texttt{fresh}}\ (x)\ e \\ & e_1 \equiv e_2 \\ & e_1 \not\equiv e_2 \\ & e_1 \vee e_2 \\ & e_1 \wedge e_2 \end{array}$$

Рис. 3. Синтаксис реляционного расширения
Fig. 3. Syntax of relational extension

Реляционное расширение добавляет пять стандартных конструкций языка miniKanren для построения целей, синтаксис которых отображен на рис. 3. Вследствие добавления конструкций miniKantren к конструкциям функционального языка, становится возможным построение всевозможных

смешанных выражений, к примеру, конъюнкция (λx.x ∧ λy.y). Для устранения подобных некорректных выражений была расширена система типов для исходного языка, что описано в прил. C. Фактический, данный подход следует реализации языка OCaml, где строгая система типов позволяет исключить большинство некорректных программ во время компиляции. Также система типов была дополнена специальным типом $\mathfrak{G}$, олицетворяющим результат вычисления отношения.

Семантика расширенного языка представлена в прил. D. Прежде всего, было расширено состояние: помимо стека контекстов и текущего выражения состояние теперь содержит множество использованных семантических переменных Σ и реляционное состояние σ. Семантические переменные вводятся и заменяют синтаксические переменные после каждого исполнения конструкции **`fresh`**. Реляционное состояние используется при исполнении унификации и desequality constraint. Все существующие правила исходного языка дополняются множеством семантических переменных и реляционным состоянием, но не используют их.

5. Преобразование функциональных программ в реляционную форму

Прежде чем описать метод преобразования функциональных программ в реляционные, сформулируем несколько ограничений для входных программ. Функциональные программы, как правило, оперируют значениями высшего порядка, в то время как miniKanren ограничен унификацией первого порядка. Поэтому не всякая функциональная программа может быть преобразована в реляционную форму. Неформально говоря, необходимо исключить значения, которые содержат в своей структуре значения высшего порядка. Это выражается в виде следующих ограничений на преобразовываемую программу:

- тип любого конструктора должен содержать либо типовые переменные, либо типовые константы;
- конструкторы и полиморфная операция сравнения могут быть применены только к значениям первого порядка;
- все **`match`**-выражения должны быть первого порядка.

Первые два ограничения сужают полиморфизм для реляционных программ: все типовые переменные могут быть заменены только на типы выражений первого порядка (это ограничение, конечно, достаточно, но не необходимо). Третье ограничение несущественно и введено только для упрощения представленного ниже преобразования в реляционную форму. Действительно, если **`match`**-выражение имеет тип высшего порядка, то его всегда можно преобразовать, используя η-расширение:

$$\texttt{match}\ e\ \texttt{with}\ \{p_i\ \ \to e_i\} \rightsquigarrow \lambda \bar{x}.\texttt{match}\ e\ \texttt{with}\ \{p_i\ \ \to e_i\,\bar{x}\},$$

где $\bar{x}$ – это вектор новых переменных, отсутствующих в выражениях e, e_i и p_i. Отметим, что реализация, описанная в разделе 6, исполняет это расширение для match-выражений, если оно является выражением высшего порядка. Это единственный случай, когда для преобразования используется тип функциональной программы и η-расширение.

Основная идея преобразования может быть проиллюстрирована на уровне типов: выражение типа t в исходном языке будет преобразовано в выражение типа $[\![t]\!]^t$ в реляционном расширении языка, где преобразование $[\![\blacksquare]\!]^t$ определяется следующим образом:

$$\begin{array}{rcl} [\![g]\!]^t & = & g \to \mathfrak{G} \\ [\![t_1 \to t_2]\!]^t & = & [\![t_1]\!]^t \to [\![t_2]\!]^t \end{array}.$$

Другими словами, выражение первого порядка будет преобразовано в одноместную функцию, возвращающую значения типа $\mathfrak{G}$. Неформальная семантика данной функции состоит в том, чтобы сопоставить аргументу исходное значение. Например, константа **`Nil`** будет преобразована в функцию $(\lambda\, q\, .\, q \equiv \texttt{Nil})$.

Теперь рассмотрим преобразование выражений, обозначаемое $[\![\blacksquare]\!]^c$. Данное преобразование определеяется рекуррентно набором правил вида

$$[\![A]\!]^c = B,$$

где A – исходная функциональная программа, B – реляционная программа, являющаяся результатом преобразования. Ниже представлены правила для всех конструкций входного функционального языка.

$$\begin{array}{rcl} [\![x]\!]^c & = & x \\ [\![\lambda x.e]\!]^c & = & \lambda x.[\![e]\!]^c \\ [\![f\ e]\!]^c & = & [\![f]\!]^c\ [\![e]\!]^c \\ [\![\underline{\texttt{let}}\ x = e_1\ \underline{\texttt{in}}\ e_2]\!]^c & = & \underline{\texttt{let}}\ x = [\![e_1]\!]^c\ \underline{\texttt{in}}\ [\![e_2]\!]^c \\ [\![\underline{\texttt{let}}\ \underline{\texttt{rec}}\ f = \lambda x.e_1\ \underline{\texttt{in}}\ e_2]\!]^c & = & \underline{\texttt{let}}\ \underline{\texttt{rec}}\ f = [\![\lambda x.e_1]\!]^c\ \underline{\texttt{in}}\ [\![e_2]\!]^c \end{array}$$

Первые пять правил не меняют структуру выражения, применяя преобразование к подвыражениям.

$$[\![C^k(e_1, \ldots, e_k)]\!]^c \quad = \quad \lambda\ q.\underline{\texttt{fresh}}\ (q_1 \ldots q_k) \quad \begin{array}{l} ([\![e_1]\!]^c\ q_1) \ \wedge \\ \ldots \\ ([\![e_k]\!]^c\ q_k) \ \wedge \\ (q \equiv C^n(q_1, \ldots, q_k)) \end{array}$$

В случае, если преобразовываемое выражение является конструктором, все его аргументы e_i являются выражениями первого порядка. Следовательно, их реляционные образы будут одноместными функциями, возвращающими цель. Для вычисления этих значений необходимо создать набор “свежих” переменных (по одной, для каждого выражения) и передать их образам в качестве аргументов. Все образы с переменными соединяем оператором конъюнкции. Результатом преобразования всего конструктора также должна

быть одноместная функция, возвращающая цель, поэтому окружаем абстракцией по переменной q полученное выше выражение, а также с помощью унификации связываем переменную q с конструктором, примененным к созданным ранее переменным.

$$\left[\!\left[\begin{array}{l} \underline{\texttt{match}}\ \ e\ \ \underline{\texttt{with}} \\ \quad \{C_i^{n_i}(x_1^i, \ldots, x_{n_i}^i)\ \rightarrow\ e_i\} \end{array} \right]\!\right]^c \quad = \quad \begin{array}{l} \lambda\ q.\underline{\texttt{fresh}}\ \ (q_e) \\ \quad (\llbracket e \rrbracket^c\ q_e)\ \ \wedge \\ \quad \bigvee_i\ ((\underline{\texttt{fresh}}\ \ (q_1^i \ldots q_{n_i}^i) \\ \qquad\qquad (q_e \equiv C_i^{n_i}(q_1^i, \ldots, q_{n_i}^i))\ \ \wedge \\ \qquad\qquad (\lambda\ x_1^i \ldots x_{n_i}^i . \llbracket e_i \rrbracket^c) \\ \qquad\qquad\quad (\equiv q_1^i)\ \ \ldots\ \ (\equiv q_{n_i}^i)\ \ q \\ \qquad\qquad)) \end{array}$$

Правило для преобразования сопоставления с образцом работает аналогичным образом. Во-первых, *скрутини* (разбираемое значение e) должно быть выражением первого порядка (так как оно сопоставляется конструкторам). Создадим “свежую” переменную q_e и свяжем её со значением *скрутини* так же, как и в предыдущем случае. Далее, для каждой ветки создадим несколько “свежих” переменных q_{ij} (по одной для каждой переменной в образце данной ветки) и выразим сопоставление образца с помощью оператора дизъюнкции, используя эти переменные и соответствующий конструктор. Наконец, тело ветки e_i – это выражение со свободными переменными, соответствующими тем, что указаны в образце. Поэтому преобразуем выражение e_i и окружим результат абстракциями, замыкающими все эти переменные и получим функцию. Теперь необходимо связать q_{ij} со свободными переменными из образца. Для этого применим описанную выше функцию к функциям, возвращающим цель ($\equiv q_{ij}$). В конечном итоге получим функцию, возвращающую цель, которую применим ко внешней переменной q, олицетворяющей результат исходного сопоставления с образцом.

$$\llbracket e_1 = e_2 \rrbracket^c \quad = \quad \lambda\ q.\underline{\texttt{fresh}}\ (q_1\ q_2) \quad \begin{array}{l} \llbracket e_1 \rrbracket^c\ q_1\ \ \wedge \\ \llbracket e_2 \rrbracket^c\ q_2\ \ \wedge \\ ((q_1 \equiv q_2\ \ \wedge q \equiv \underline{\texttt{true}})\ \ \vee \\ \ (q_1 \not\equiv q_2\ \ \wedge q \equiv \underline{\texttt{false}}) \\ ) \end{array}$$

Последнее правило следует тому же шаблону: оба аргумента полиморфного сравнения преобразуются в функции, возвращающие цель, причем их аргументы будут иметь одинаковый тип выражения первого порядка. Применим эти функции к “свежим” переменным и выполним разбор двух случаев: сравниваемые выражения равны, либо не равны. Отметим, что это единственный случай использования конструкции disequality constraint.

Интересным свойством данного преобразования в реляционную форму является сохранение выражения неизменным в том случае, когда оно не содержит конструкторов, сравнения и сопоставления с образцом. Таким образом, множество полезных функций высшего порядка – применение,

композиция, неподвижная точка – уже являются реляционными и могут быть использованы в реляционных спецификациях без изменений.

Другое свойство состоит в том, что это преобразование в реляционную форму является композиционной (действительно, реляционный образ применения есть применение реляционных образов). Это означает, что реляционное преобразование совместимо с раздельной компиляцией – несколько исходных файлов могут быть преобразованы независимо, не теряя возможности работать должным образом при их объединении.

Также, интересным является тот факт, что результат преобразования в реляционную форму исполняется детерминировано в прямом направлении. Таким образом преобразование в реляционную форму вызывает константное замедление при прямом исполнении.

Для подтверждения корректности преобразования $[\![\blacksquare]\!]^c$ были сформулированы и доказаны следующие теоремы.

Теорема 1. (Статическая корректность). Если выражение e имеет тип t в исходном языке, тогда $[\![e]\!]^c$ с имеет тип $[\![t]\!]^t$ в реляционном расширении. Другими словами, преобразование в реляционную форму переводит правильно типизированные программы в правильно типизированные. Доказано с помощью структурной индукции.

Теорема 2. (Частичная семантическая корректность). Если выражение первого порядка e имеет тип t и e вычисляется до некоторого значения v $\left(e \xrightarrow{func} v\right)$, тогда **fresh** ($x$) ($[\![e]\!]^c$ x) вычисляется до подстановки θ, и $\theta(\mathfrak{s}) = v$, где $\mathfrak{s}$ семантическая переменная, ассоциированная с x на первом шаге вычисления. Доказано с помощью симуляции.

6. Апробация

Описанное в предыдущем разделе метод преобразования функциональных программ в реляционные был реализован на языке OCaml. В качестве входного языка используется подмножество OCaml, описанное в разделе 4.

При реализации были выявлены две проблемы. Во-первых, результат преобразования содержит множество λ-абстракций, многие из которых могут быть применены немедленно. Для их устранения был добавлен дополнительный опциональный проход по абстрактному синтаксическому дереву преобразованной программы, который выполняет β-редукцию везде, где это возможно. Данная оптимизация значительно улучшает качество конвертируемых программ как с точки зрения читаемости, так и по производительности. Далее, в нашей первоначальной реализации слишком много значений преобразовывались в функции и, как результат, во время исполнения их тела вычислялись несколько раз с существенным ухудшением производительности. Нами была улучшена реализация путем внедрения в результирующую реляционную программу принудительного вычисления

аргументов первого порядка для каждого содержащегося в программе вызова функции.

В качестве первого примера преобразования рассмотрим реализацию функции конкатенации для списков (см. рис. 4a). Результат преобразования (рис. 4b) несколько отличается от классического реализации реляционного отношения конкатенации списков. Основное различие происходит от функционализации примитивных значений: в то время как обычные **`append`°** работает со значениями-списками, преобразованный вариант использует функции, возвращающие цель. Таким образом, классическая **`append`°** для аргументов **`x`**, **`y`** и **`q`** может быть выражено с помощью преобразованного в качестве **`append`° `(≡ x) (≡ y) q`**.

Далее мы продемонстрируем возможность выполнимости реляционных форм в различных направлениях на следующих примерах:

- интерпретатор высшего порядка для лямбда-исчисления, принимающий в качестве аргумента функцию поиска подвыражения, к которому необходимо применить бета-редукцию;
- алгоритм Хиндли-Милнера [7] для вывода наиболее общего типа.

```
val append : α list →                  val appendᵒ : (α llist → 𝔊) →
             α list →                               (α llist → 𝔊) →
             α list                                 α llist → 𝔊
let rec append = λ a.λ b.              let rec appendᵒ a b q1 =
  match a with                           fresh (q2)
  | Nil          → b                       (a q2) ∧
  | Cons (h, t) →                          (((q2 ≡ Nil) ∧ (b q1)) |||
      Cons (h, append t b)                   (fresh (q3 q4)
                                               (q2 ≡ Cons (q3, q4)) ∧
                                               (fresh (q6 q7)
                                                 (q6 ≡ q3) ∧
                                                 (q1 ≡ Cons (q6, q7)) ∧
                                                 (appendᵒ (≡ q4) b q7))))
          (a)                                              (b)
```

Рис. 4. Пример преобразования функции в отношение
Pic. 4. Example of relational conversion

6.1 Интерпретатор высшего порядка для лямбда-исчисления

Как было сказано во введении, одним из применений языка miniKanren является разработка реляционных интерпретаторов [6; 18; 21]. Отличительной особенностью данного интерпретатора является его аргумент высшего порядка, который используется для поиска подвыражения, к которому применима бета-редукция. Таким образом, в зависимости от этого аргумента

можно получить интерпретатор с различными стратегиями вычисления: *call-by-name*, *call-by-value*, нормальный порядок редуциии и др. Реализованный функциональный интерпретатор и функции редукции имеют следующую сигнатуру:

```
val eval : (term → split) → term → term
val call_by_name  : term → split
val call_by_value : term → split
val normal_order  : term → split
```

где **`term`** – выражение лямбда-исчисления в нотации Де Брюэна; **`split`** – пара из выражения и контекста. Функция высшего порядка **`eval`** принимает в качестве первого аргумента функцию, определяющую порядок редукции, в качестве второго аргумента – выражение, которое необходимо редуцировать.

После преобразования будут получены отношения со следующими сигнатурами:

```
val eval^o : ((term → 𝔊) → split → 𝔊) → (term → 𝔊) →
             term → 𝔊
val call_by_name^o  : (term → 𝔊) → split → 𝔊
val call_by_value^o : (term → 𝔊) → split → 𝔊
val normal_order^o  : (term → 𝔊) → split → 𝔊
```

Полученное с помощью реляционного преобразования отношение позволяет непосредственно интерпретировать лямбда-выражения.

```
eval^o normal_order^o (≡ ‘(λ 0) 1‘) q ⇝ [q ↦ ‘1‘]
eval^o call_by_name^o (≡ ‘0 ((λ 0) 1)‘) q ⇝ [q ↦ ‘0 ((λ 0) 1)‘]
eval^o call_by_value^o (≡ ‘0 ((λ 0) 1)‘) q ⇝ [q ↦ ‘0 1‘]
```

В качестве примера рассмотрим три различных запроса с лямбда-выражениями и отношениями редукции. Во всех трех случаях запрос был успешно выполнен; для каждого лямбда-выражения была построена корректная нормальная форма, соответствующая выбранной стратегии редукции.

Также реляционная форма **`eval°`** позволяет генерировать из нормальных форм (возможно, бесконечный) поток выражений, из которых эта нормальная форма была получена с помощью переданного отношения, определяющего стратегию вычисления.

```
eval^o normal_order^o (≡ q) (‘λ 0‘) ⇝ [
    q ↦ ‘λ 0‘;
    q ↦ ‘(λ 0) (λ 0)‘;
    q ↦ ‘λ ((λ 1) 0)‘;
    q ↦ ‘(λ 0) ((λ 0) (λ 0))‘; ...]
eval^o call_by_name^o (≡ q) (‘λ 0‘) ⇝ [
    q ↦ ‘λ 0‘;
    q ↦ ‘(λ 0) (λ 0)‘;
    q ↦ ‘(λ 0) ((λ 0) (λ 0))‘;
    q ↦ ‘(λ λ0) 0‘; ...]
```

В представленных выше запросах задана нормальная форма и стратегия редукции. Каждый из запросов порождает поток лямбда-выражений с заданной нормальной формой. В этом и последующих примерах результаты вычисления могут содержать свободные переменные, которые обозначаются числом в квадрате и интерпретируются как произвольное значение заданного типа.

Отметим, что аргумент, определяющий стратегию редукции, породить нельзя, так как он является функцией высшего порядка. Данное ограничение является следствием ограниченности синтаксической унификации.

6.2 Вывод типов Хиндли-Милнера

Данный алгоритм [22] по лямбда-выражению вычисляет наиболее общий тип этого выражения.

Функция **type_inference**, являющаяся реализацией алгоритма вывода типов и отношение **type_inference°**, являющееся результатом преобразования **type_inference**, имеют следующие типы:

$$\underline{\texttt{val}}\ \texttt{type\_inference} \ : \texttt{term} \to \texttt{typ}$$
$$\underline{\texttt{val}}\ \texttt{type\_inference}^o \ : (\texttt{term} \to \mathfrak{G}) \to \texttt{typ} \to \mathfrak{G}$$

Полученное с помощью реляционного преобразования отношение можно использовать для вычисления типа выражения.

$$\texttt{type\_inference}^o\ (\equiv\ `\lambda x \to x`)\ \texttt{q} \rightsquigarrow [\texttt{q} \mapsto\ `a \to a`]$$

Данный запрос для заданного выражения порождает корректный тип. Также реляционную форму **type_inference°** можно использовать для решения проблемы населенности типа.

$$\texttt{type\_inference}^o\ (\equiv \texttt{q})\ \ 'a' \rightsquigarrow \bot$$
$$\texttt{type\_inference}^o\ (\equiv \texttt{q})\ \ 'a \to a' \rightsquigarrow [$$
$$\texttt{q} \mapsto\ '\lambda\ \boxed{0} \to \boxed{0}';$$
$$\texttt{q} \mapsto\ '\lambda\ \boxed{0} \to (\lambda\ \boxed{1} \to \boxed{1})\ \boxed{0}';$$
$$\texttt{q} \mapsto\ '\lambda\ \boxed{0} \to \underline{\texttt{let}}\ \boxed{1} = \boxed{2}\ \underline{\texttt{in}}\ \boxed{0}'\ (\boxed{0} \not\equiv \boxed{1});$$
$$\texttt{q} \mapsto\ '(\lambda\ \boxed{0} \to \boxed{0})\ (\lambda\ \boxed{1} \to \boxed{1})';\ \ldots]$$

В первом запросе задан ненаселенный тип. Это подтверждает результат вычисления запроса отсутствием найденных выражений. Второй запрос породил бесконечный поток выражений, следовательно, заданный тип населен.

Наконец, данную реляционную форму можно использовать для достраивания выражения с дыркой таким образом, чтобы оно имело заданный тип.

$$\texttt{type\_inference}^o\ (\equiv\ `\underline{\texttt{let}}\ \texttt{f} = \square\ \underline{\texttt{in}}\ \texttt{f}\ (\lambda\ \texttt{x} \to \texttt{f}\ \texttt{x})`)\ \ `a \to a` \rightsquigarrow$$
$$[\square \mapsto\ `\lambda\ \boxed{0} \to \boxed{0}`;\ \ldots]$$

Данный запрос порождает выражения, которые можно подставить на место □, после чего выражение будет корректно типизироваться.

7. Выводы

В данной работе представлен метод для преобразования типизированных функциональных программ в реляционную форму. Во многих случаях данное преобразование позволяет избежать утомительного переписывания функциональных спецификаций в реляционную форму и сосредоточиться на реляционных спецификациях только тогда, когда их получение из функций невозможно или нежелательно.

Наш метод преобразования применим только к функциональным программам с ограниченным полиморфизмом, вследствие ограничений синтаксической унификации, используемой в языке miniKanren.

Апробация показала, что полученные с помощью метода преобразования реляционные формы можно исполнять для вычисления произвольного аргумента первого порядка. Более того, любой аргумент первого прядка можно определить частично, что позволяет гибко уточнять запрос к реляционной форме.

Список литературы

[1]. Friedman D. P., E.Byrd W., Kiselyov O. The Reasoned Schemer. MIT Press, 2005.
[2]. Язык Mercury. URL: https://mercurylang.org (дата обращения 09.04.2018).
[3]. Язык Curry. URL: http://www-ps.informatik.uni-kiel.de/currywiki (дата обращения 09.04.2018).
[4]. Язык miniKanren. URL: http://minikanren.org (дата обращения 09.04.2018).
[5]. Hemann J., Friedman D. P. μKanren: A Minimal Core for Relational Programming. Workshop on Scheme and Functional Programming, 2013.
[6]. Byrd W. E. Relational Programming in miniKanren: Techniques, Applications, and Implementations. Ph.D. thesis, Indiana University, Bloomington, 2009.
[7]. Pierce B. Types and Programming Languages. MIT Press, 2002.
[8]. Кознов Д. В. Методология и инструментарий предметно-ориентированного моделирования. Диссертация на соискание учёной степени доктора технических наук, СПбГУ, 2016.
[9]. Ольхович Л., Кознов Д. Метод автоматической валидации UML-спецификаций на основе OCL. Программирование, 2003, том 29, № 6, стр. 44–50.
[10]. Терехов А.Н., Романовский К.Ю., Кознов Д.В., Долгов П.С., Иванов А.Н.. RTST++: методология и CASE-средство для разработки информационных систем и программного обеспечения для систем реального времени. Программирование, 1999, том 25, № 5.
[11]. Codognet P., Diaz D. WAMCC: Compiling Prolog to C. The MIT Press, 1995, pp. 317–331.
[12]. Henderson F., Somogyi Z. Compiling mercury to high-level C code. In Computational Complexity, 2002, pp. 197–212.
[13]. Banbara M., Tamura N., Inoue K. Prolog Cafe: A prolog to Java translator system. Lecture Notes in Computer Science, vol. 4369, 2006, pp. 1–11.
[14]. G´omez-Zamalloa M., Albert E., Puebla G. Decompilation of Java bytecode to Prolog by partial evaluation. Information and Software Technology, 2009, vol. 51, № 10, pp. 1409–1427.

[15]. Calejo M. InterProlog: Towards a Declarative Embedding of Logic Programming in Java. JELIA 2004: Logics in Artificial Intelligence, pp. 714-717.
[16]. J. Cook J. P#: A concurrent Prolog for the .NET framework. Software Practice and Experience, vol. 34. № 9, 2004, pp. 815-845.
[17]. Byrd W. E., Holk E., Friedman D. P. miniKanren, Live and Untagged: Quine Generation via Relational Interpreters (Programming Pearl), Workshop on Scheme and Functional Programming, 2012.
[18]. Kosarev D., Boulytchev D. Typed Embedding of a Relational Language in OCaml. ACM SIGPLAN Workshop on ML, 2016.
[19]. Язык OCanren. URL: http://github.com/dboulytchev/ocanren (дата обращения 09.04.2018).
[20]. Alvis C. E., Willcock J. J., Byrd W. E. cKanren: miniKanren with Constraints, Workshop on Scheme and Functional Programming, 2011.
[21]. Byrd W. E., Ballantyne M., Rosenblatt G., Might M. A Unified Approach to Solving Seven Programming Problems (Functional Pearl). Proc. ACM Program. Lang, 2017, vol. 1, ICFP, pp. 8:1–8:26.
[22]. Barendregt H. Lambda Calculi with Types. Handbook of Logic in Computer Science, Volume II, Oxford University Press, 1993.

Приложения

A. Правила типизации для входного языка

Типы:

$$
\begin{array}{lcll}
\mathcal{X} & = & \alpha, \beta, \ldots & \text{(типовые переменные)} \\
\mathcal{D} & = & \texttt{bool}, T^n, \ldots & \text{(конструкторы типов данных)} \\
\mathcal{T} & = & \alpha \mid T^k(t_1, \ldots, t_k) \mid t_1 \to t_2 & \text{(типы)} \\
\mathcal{S} & = & \forall \bar{\alpha}.t & \text{(схемы типов)}
\end{array}
$$

Правила типизации:

$$\Gamma \vdash \underline{\texttt{true}}, \underline{\texttt{false}} : \texttt{bool} \qquad [\textsc{Bool}_T]$$

$$\frac{\Gamma \vdash e_1 : t \quad \Gamma \vdash e_2 : t}{\Gamma \vdash e_1 = e_2 : \texttt{bool}} \qquad [\textsc{Eq}_T]$$

$$\frac{\Gamma \vdash e_i : t_i^C}{\Gamma \vdash C^n(e_1, \ldots, e_n) : t^C} \qquad [\textsc{Constr}_T]$$

$$\Gamma, x : \forall \bar{\alpha}.t \vdash x : t[\bar{\alpha} \leftarrow \bar{t'}] \qquad [\textsc{Var}_T]$$

$$\frac{\Gamma \vdash f : t_1 \to t_2 \quad \Gamma \vdash e : t_1}{\Gamma \vdash f\ e : t_2} \qquad [\textsc{App}_T]$$

$$\frac{\Gamma, x : t_1 \vdash f : t_2}{\Gamma \vdash \lambda x.f : t_1 \to t_2} \qquad [\textsc{Abs}_T]$$

$$\frac{\Gamma \vdash e_1 : t_1 \quad \Gamma, x : \forall \bar{\alpha}.t_1 \vdash e_2 : t}{\Gamma \vdash \underline{\texttt{let}}\ x = e_1\ \underline{\texttt{in}}\ e_2 : t}, \ \bar{\alpha} = FV(t_1) \setminus FV(\Gamma) \qquad [\textsc{Let}_T]$$

$$\frac{\Gamma, f : t_1 \vdash \lambda x.e_1 : t_1 \quad \Gamma, f : \forall \bar{\alpha}.t_1 \vdash e_2 : t}{\Gamma \vdash \underline{\texttt{let}}\ \underline{\texttt{rec}}\ f = \lambda x.e_1\ \underline{\texttt{in}}\ e_2 : t}, \ \bar{\alpha} = FV(t_1) \setminus FV(\Gamma) \qquad [\textsc{LetRec}_T]$$

$$\frac{\Gamma \vdash e : t^C \quad \Gamma, x_1^i : t_1^{C_i}, \ldots, x_{k_i}^i : t_{k_i}^{C_i} \vdash e_i : t}{\Gamma \vdash \underline{\texttt{match}}\ e\ \underline{\texttt{with}}\ \{C_i^{k_i}(x_1^i, \ldots, x_{k_i}^i) \to e_i\} : t} \qquad [\textsc{Match}_T]$$

B. Семантика входного языка

Значения:

$$\mathcal{V} = C^n(v_1, \ldots, v_n) \mid \lambda x.e \mid \mu f \lambda x.e \mid \underline{\texttt{true}} \mid \underline{\texttt{false}}$$

Контексты:

$$\mathcal{C} = \square\ e \mid v\ \square \mid \underline{\texttt{let}}\ x = \square\ \underline{\texttt{in}}\ e \mid \underline{\texttt{match}}\ \square\ \underline{\texttt{with}}\ \{p_i \to e_i\} \mid C^n(\bar{v}, \square, \bar{e}) \mid \square = e \mid v = \square$$

Стек контекстов:

$$\mathcal{S} = \epsilon \mid \mathcal{C} : \mathcal{S}$$

Состояния:

$\langle \mathcal{S}, e \rangle$ (стек контекстов, выражение); $\langle \epsilon, e \rangle$ (начальное сотояние); $\langle \epsilon, v \rangle$ (финальное состояние)

Переходы:

$$\langle C : \mathcal{S},\ v \rangle \to \langle \mathcal{S},\ C[v] \rangle \qquad [\textsc{Value}]$$

$$\langle \mathcal{S},\ f\ e \rangle \to \langle \square\ e : \mathcal{S},\ f \rangle \qquad [\textsc{AppL}] \qquad \langle \mathcal{S},\ v\ e_2 \rangle \to \langle v\ \square : \mathcal{S},\ e_2 \rangle \qquad [\textsc{AppR}]$$

$$\langle \mathcal{S},\ e_1 = e_2 \rangle \to \langle \square = e_2 : \mathcal{S},\ e_1 \rangle \qquad [\textsc{EqL}] \qquad \langle \mathcal{S},\ v = e \rangle \to \langle v = \square : \mathcal{S},\ e \rangle \qquad [\textsc{EqR}]$$

$$\langle \mathcal{S},\ v = v \rangle \to \langle \mathcal{S},\ \underline{\texttt{true}} \rangle \qquad [\textsc{EqTrue}]$$

$$\langle \mathcal{S},\ v_1 = v_2 \rangle \to \langle \mathcal{S},\ \underline{\texttt{false}} \rangle,\ v_1 \neq v_2 \qquad [\textsc{EqFalse}]$$

$$\langle \mathcal{S},\ (\lambda x.e)\ v \rangle \to \langle \mathcal{S},\ e[x \leftarrow v] \rangle \qquad [\textsc{Beta}]$$

$$\langle \mathcal{S},\ (\mu f \lambda x.e)\ v \rangle \to \langle \mathcal{S},\ e[f \leftarrow \mu f \lambda x.e,\ x \leftarrow v] \rangle \qquad [\textsc{Mu}]$$

$$\langle \mathcal{S},\ C^n(v_1, \ldots, v_{k-1}, e_k, \ldots, e_n) \rangle \to \langle C^n(v_1, \ldots, v_{k-1}, \square, \ldots, e_n) : \mathcal{S},\ e_k \rangle \qquad [\textsc{Constr}]$$

$$\langle \mathcal{S},\ \underline{\texttt{let}}\ x = e_1\ \underline{\texttt{in}}\ e_2 \rangle \to \langle \underline{\texttt{let}}\ x = \square\ \underline{\texttt{in}}\ e_2 : \mathcal{S},\ e_1 \rangle \qquad [\textsc{Let}]$$

$$\langle \mathcal{S},\ \underline{\texttt{let}}\ x = v\ \underline{\texttt{in}}\ e \rangle \to \langle \mathcal{S},\ e[x \leftarrow v] \rangle \qquad [\textsc{LetVal}]$$

C. Правила типизации для реляционного расширения

Типы:

$$\begin{array}{lcll} \mathcal{L} & = & \alpha^o \mid (T^n(l_1, \ldots, l_n))^o & \text{(логические переменные)} \\ \mathcal{T} & += & \mathfrak{G} & \end{array}$$

Правила типизации:

$$\frac{\Gamma, x : l \vdash e : \mathfrak{G}}{\Gamma \vdash \underline{\texttt{fresh}}\ (x)\ e : \mathfrak{G}} \qquad [\textsc{Fresh}_T]$$

$$\frac{\Gamma \vdash e_1 : l \quad \Gamma \vdash e_2 : l}{\Gamma \vdash e_1 \equiv e_2 : \mathfrak{G}} \qquad [\textsc{Unify}_T] \qquad \frac{\Gamma \vdash e_1 : l \quad \Gamma \vdash e_2 : l}{\Gamma \vdash e_1 \not\equiv e_2 : \mathfrak{G}} \qquad [\textsc{Disequality}_T]$$

$$\frac{\Gamma \vdash e_1 : \mathfrak{G} \quad \Gamma \vdash e_2 : \mathfrak{G}}{\Gamma \vdash e_1 \wedge e_2 : \mathfrak{G}} \qquad [\textsc{Conjunction}_T] \qquad \frac{\Gamma \vdash e_1 : \mathfrak{G} \quad \Gamma \vdash e_2 : \mathfrak{G}}{\Gamma \vdash e_1 \vee e_2 : \mathfrak{G}} \qquad [\textsc{Disjunction}_T]$$

D. Семантика для реляционного расширения

Семантические переменные:

$$\mathfrak{S} = \mathfrak{s}_1, \mathfrak{s}_2, \ldots$$

$\Sigma, \Sigma' \cdots \subset 2^{\mathcal{S}}$ (множества выделенных семантических переменных)

$\langle \Sigma', \mathfrak{s} \rangle \leftarrow \underline{\texttt{new}}\ \Sigma,\ \Sigma' = \Sigma \cup \{\mathfrak{s}\},\ \mathfrak{s} \notin \Sigma$ (выделение новой семантической переменной)

Значения:

$$\mathcal{V} \mathrel{+}= \underline{\texttt{success}} \mid \mathfrak{s}$$

Контексты:

$$\mathcal{C} \mathrel{+}= \square \equiv e \mid v \equiv \square \mid \square \not\equiv e \mid v \not\equiv \square \mid \square \wedge e \mid e \wedge \square$$

Состояния:

$\langle \Sigma, \mathcal{S}, e, \sigma \rangle$ (мн-во семмантических переменных, стек контекстов, выражение, логическое состояние)

$\langle \emptyset, \epsilon, e, \iota \rangle$ (начальное сотояние)

Переходы:

$$\langle \Sigma,\ \mathcal{S},\ \underline{\texttt{fresh}}\ (x)\ e,\ \sigma \rangle \rightsquigarrow \langle \Sigma',\ \mathcal{S},\ e[x \leftarrow \mathfrak{s}],\ \sigma \rangle,\ \langle \Sigma', \mathfrak{s} \rangle \leftarrow \underline{\texttt{new}}\ \Sigma \qquad [\textsc{Fresh}]$$

$$\langle \Sigma,\ \mathcal{S},\ e_1 \equiv e_2,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ \square \equiv e_2 : \mathcal{S},\ e_1,\ \sigma \rangle \qquad [\textsc{UnifyL}]$$

$$\langle \Sigma,\ \mathcal{S},\ v \equiv e,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ v \equiv \square : \mathcal{S},\ e,\ \sigma \rangle \qquad [\textsc{UnifyR}]$$

$$\langle \Sigma,\ \mathcal{S},\ v_1 \equiv v_2,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ \mathcal{S},\ \underline{\texttt{success}},\ \sigma' \rangle,\ \mathbf{unify}\,(\sigma,\ v_1,\ v_2) = \sigma' \qquad [\textsc{Unify}]$$

$$\langle \Sigma,\ \mathcal{S},\ e_1 \not\equiv e_2,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ \square \not\equiv e_2 : \mathcal{S},\ e_1,\ \sigma \rangle \qquad [\textsc{DisEqL}]$$

$$\langle \Sigma,\ \mathcal{S},\ v \not\equiv e,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ v \not\equiv \square : \mathcal{S},\ e,\ \sigma \rangle \qquad [\textsc{DisEqR}]$$

$$\langle \Sigma,\ \mathcal{S},\ v_1 \not\equiv v_2,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ \mathcal{S},\ \underline{\texttt{success}},\ \sigma' \rangle,\ \mathbf{diseq}\,(\sigma,\ v_1,\ v_2) = \sigma' \qquad [\textsc{DisEq}]$$

$$\langle \Sigma,\ \mathcal{S},\ e_1 \vee e_2,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ \mathcal{S},\ e_1,\ \sigma \rangle \qquad [\textsc{DisjL}]$$

$$\langle \Sigma,\ \mathcal{S},\ e_1 \vee e_2,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ \mathcal{S},\ e_2,\ \sigma \rangle \qquad [\textsc{DisjR}]$$

$$\langle \Sigma,\ \mathcal{S},\ e_1 \wedge e_2,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ \square \wedge e_2 : \mathcal{S},\ e_1,\ \sigma \rangle \qquad [\textsc{ConjStartL}]$$

$$\langle \Sigma,\ \mathcal{S},\ e_1 \wedge e_2,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ e_1 \wedge \square : \mathcal{S},\ e_2,\ \sigma \rangle \qquad [\textsc{ConjStartR}]$$

$$\langle \Sigma,\ \mathcal{S},\ \underline{\texttt{success}} \wedge e,\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ \mathcal{S},\ e,\ \sigma \rangle \qquad [\textsc{ConjL}]$$

$$\langle \Sigma,\ \mathcal{S},\ e \wedge \underline{\texttt{success}},\ \sigma \rangle \rightsquigarrow \langle \Sigma,\ \mathcal{S},\ e,\ \sigma \rangle \qquad [\textsc{ConjR}]$$

Conversion Typed Functions into Relational Form

P. Lozov <lozov.peter@gmail.com>
D. Boulytchev <dboulytchev@math.spbu.ru>
St. Petersburg State University,
Universitetski pr., 28, 198504 St. Petersburg, Russia

Abstract. Relational programming is an approach that allows you to execute programs in different "directions" to get different behaviors from one relational specification. The direct development of relational programs is a complex task, requiring the developer to have certain skills. However, in many cases, the required relational program can be obtained from a certain functional program automatically. In this paper, the problem of automatic conversion

of functional programs into relational ones is considered. The main contribution of the paper is the method of converting typed functions into a relational form, as well as proving its static and dynamic correctness. This method can be applied to typed programs of a general kind. To describe these programs, a compact ML-like language (a subset of OCaml) is used, equipped with a Hindley-Milner type system with let-polymorphism Also, the paper discusses the limitations of the proposed method, presents an implementation for a subset of the OCaml language, and evaluates the method on a number of realistic examples.

Keywords: functional programming; relational programming; conversion of programs.

DOI: 10.15514/ISPRAS-2018-30(2)-3

For citation: Lozov P., Boulytchev D. Conversion Typed Functions into Relational Form. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 2, 2018, pp. 45-64 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-3

References

[1]. Friedman D. P., E.Byrd W., Kiselyov O. The Reasoned Schemer. MIT Press, 2005.
[2]. Mercury Language. URL: https://mercurylang.org (accessed 09.04.2018).
[3]. Curry Language. URL: http://www-ps.informatik.uni-kiel.de/currywiki (дата обращения 09.04.2018).
[4]. miniKanren Language. URL: http://minikanren.org (accessed 09.04.2018).
[5]. Hemann J., Friedman D. P. μKanren: A Minimal Core for Relational Programming. Workshop on Scheme and Functional Programming, 2013.
[6]. Byrd W. E. Relational Programming in miniKanren: Techniques, Applications, and Implementations. Ph.D. thesis, Indiana University, Bloomington, 2009.
[7]. Pierce B. Types and Programming Languages. MIT Press, 2002.
[8]. Koznov D. V. Methodology and tools for object-oriented modeling. PhD Thesis, SPBU, 2016 (in Russian).
[9]. Ol'khovich L., Koznov D.V. Ocl-based Automated Validation Method For Uml Specifications. Programming and Computer Software, 2003, vol. 29, № 6, pp. 323–327. DOI: 10.1023/B:PACS.0000004132.42846.11.
[10]. Terekhov A. N., Romanovskii K. Yu., Koznov D. V., Dolgov P. S., Ivanov A. N. RTST++: Methodology and a CASE Tool for the Development of Information Systems and Software For Real-Time Systems. Programming and Computer Software, 1999, vol. 25, № 5, pp. 276–281.
[11]. Codognet P., Diaz D. WAMCC: Compiling Prolog to C. The MIT Press, 1995, pp. 317–331.
[12]. Henderson F., Somogyi Z. Compiling mercury to high-level C code. In Computational Complexity, 2002, pp. 197–212.
[13]. Banbara M., Tamura N., Inoue K. Prolog Cafe: A prolog to Java translator system. Lecture Notes in Computer Science, vol. 4369, 2006, pp. 1–11.
[14]. G´omez-Zamalloa M., Albert E., Puebla G. Decompilation of Java bytecode to Prolog by partial evaluation. Information and Software Technology, 2009, vol. 51, № 10, pp. 1409–1427.
[15]. Calejo M. InterProlog: Towards a Declarative Embedding of Logic Programming in Java. JELIA 2004: Logics in Artificial Intelligence, pp. 714-717.

[16]. J. Cook J. P#: A concurrent Prolog for the .NET framework. Software Practice and Experience, vol. 34. № 9, 2004, pp. 815-845.
[17]. Byrd W. E., Holk E., Friedman D. P. miniKanren, Live and Untagged: Quine Generation via Relational Interpreters (Programming Pearl), Workshop on Scheme and Functional Programming, 2012.
[18]. Kosarev D., Boulytchev D. Typed Embedding of a Relational Language in OCaml. ACM SIGPLAN Workshop on ML, 2016.
[19]. Язык OCanren. URL: http://github.com/dboulytchev/ocanren (accessed 09.04.2018).
[20]. Alvis C. E., Willcock J. J., Byrd W. E. cKanren: miniKanren with Constraints, Workshop on Scheme and Functional Programming, 2011.
[21]. Byrd W. E., Ballantyne M., Rosenblatt G., Might M. A Unified Approach to Solving Seven Programming Problems (Functional Pearl). Proc. ACM Program. Lang, 2017, vol. 1, ICFP, pp. 8:1–8:26.
[22]. Barendregt H. Lambda Calculi with Types. Handbook of Logic in Computer Science, Volume II, Oxford University Press, 1993.

Автоматизированная генерация декодеров машинных команд

Н.Ю. Фокина <nfokina@ispras.ru>
М.А. Соловьев <icee@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В работе предложен метод автоматизированной генерации декодеров машинных команд широкого класса процессорных архитектур с использованием транслятора языка ассемблера целевой архитектуры. Реализована программная система, использующая предложенный метод для генерации декодеров машинных команд различных архитектур. Система была протестирована на нескольких микроконтроллерах: PIC16F877A, AVR, Tricore, H8/300H.

Ключевые слова: декодер; микроконтроллер; бинарный код; автоматизированная генерация; система команд.

DOI: 10.15514/ISPRAS-2018-30(2)-4

Для цитирования: Фокина Н.Ю., Соловьев М.А. Автоматизированная генерация декодеров машинных команд. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 65-80. DOI: 10.15514/ISPRAS-2018-30(2)-4

1. Введение

Существует большое число различных архитектур процессоров. Большая часть из них приходится на микроконтроллеры. Для разработки машинно-зависимого системного программного обеспечения (компилятор, отладчик, компоновщик и т.д.), поддерживающего данные архитектуры, необходимо наличие декодера машинных команд данного процессора. Декодером называют инструмент, восстанавливающий текстовое представление машинной команды. Иногда декодер также извлекает другие свойства, например, зависимости по данным внутри инструкции, что необходимо для более глубокого анализа бинарного кода. В отличие от дизассемблера, производящего анализ всего исполняемого файла, декодер обрабатывает отдельные инструкции.

Таким образом, декодирование является подзадачей при дизассемблировании. Также декодер применяется как составная часть других инструментов: средств анализа бинарного кода, мониторов виртуальных машин, систем динамической двоичной трансляции и т.д.

Для подавляющего большинства архитектур существующие инструменты позволяют восстановить лишь текстовое представление инструкций. Это существенно усложняет разработку системного программного обеспечения, поскольку требуется добавлять поддержку каждой архитектуры отдельно. Необходимо обеспечить единообразный интерфейс для декодирования инструкций различных архитектур. Под единообразностью понимается возможность получения информации о мнемонике, числе и типе операндов инструкции, а также о каждом операнде (для операндов-регистров – имя регистра, для операндов-констант и смещений в памяти – соответствующее значение).

В некоторых случаях может отсутствовать, быть неполной или устаревшей документация, описывающая архитектуру системы команд (Instruction Set Architecture, ISA) процессора. При этом в набор системных программ для большинства архитектур входит ассемблер. С его помощью можно установить соответствие между инструкциями и задаваемыми ими кодировками. Эту информацию можно использовать при разработке декодера.

Поскольку написание декодера вручную является ресурсоемким, и полученный в результате инструмент может содержать ошибки, целесообразно генерировать декодер автоматически. Довольно широкое распространение получили специализированные языки описания системы команд, первым из которых был разработанный в 1997 году SLED/NJMC [1,2]. Их использование позволяет избежать написания системных программ (в частности, декодеров) вручную.

Однако можно добиться большей автоматизации, извлекая информацию о архитектуре системы команд из существующего транслятора языка ассемблера. Тогда для генерации декодера используется не низкоуровневое описание кодировок команд, а лишь описание синтаксиса языка ассемблера. Это позволяет существенно упростить работу аналитика и ускорить разработку декодеров. Это особенно актуально при разработке большого числа однотипных инструментов, поскольку при таком подходе аналитик может приступать к описанию очередной архитектуры, в то время как генератор автоматически извлекает информацию о кодировках команд.

В данной статье предложен метод, позволяющий автоматически на основе запусков транслятора языка ассемблера генерировать декодеры команд процессоров. Архитектура системы команд целевого процессора должна удовлетворять следующим требованиям:

- битовые поля, кодирующие операнды, не пересекаются (т.е. каждому операнду соответствует некоторое битовое поле);
- операнды-константы кодируются в дополнительном коде;
- значения констант лежат на непрерывном интервале и кратны некоторой степени 2 (например, последовательность .. -2, 0, 2, 4 .. удовлетворяет данному требованию, а .. -4, 0, 2, 4 .. или 0, 1, 3, 5 .. – нет).

Разработанный метод лег в основу программной системы, описание которой также приводится в настоящей статье.

Дальнейшее изложение построено следующим образом. В разд. 2 приводится обзор работ, смежных с данной. В разд. 3 дается общее описание разработанной системы. В разд. 4 и 5 более подробно рассматриваются генерация описания кодировок команд и декодирование соответственно. В разд. 6 приводятся результаты тестирования разработанной системы. Разд. 7 содержит заключение.

2. Обзор родственных работ

Решаемая задача является родственной по отношению к двум достаточно независимых группам работ. В первую группу входят работы по созданию инструментов, позволяющих восстанавливать соответствие между ассемблерными инструкциями и их кодировками; во вторую – разработка адаптируемых декодеров, использующих машинно-независимые алгоритмы для декодирования команд различных процессорных архитектур.

К первой группе относится инструмент Derive [3], который позволяет автоматически восстанавливать кодировки ассемблерных инструкций, а также генерировать описания данных инструкций. Работа инструмента основана на использовании ассемблера целевой архитектуры. Пользователем задается описание целевой архитектуры: перечень регистров, перечень мнемоник, описание синтаксиса языка ассемблера. Путем многократного запуска и анализа результатов работы ассемблера инструмент определяет, какие биты кодировки соответствуют мнемонике и операндам инструкции. Если отображение задается достаточно сложным образом, инструмент приостанавливает работу и запрашивает описание инструкции у пользователя.

Несколько другой подход был предложен в серии работ К. Коллберга [4, 5], посвященной разработке адаптируемого (self-retargetable) компилятора. Путем компиляции небольших программ, написанных на языке C, с помощью системных компилятора и ассемблера целевого процессора строится описание системы команд данной архитектуры, необходимое для машинно-зависимой генерации кода. Однако для многих микроконтроллеров компилятор C отсутствует, а некоторые не могут эмулировать абстрактную C-машину в силу ограниченности аппаратных ресурсов.

В ИСП РАН была разработана система MetaDSP [6], предназначенная для разработки кросс-инструментов для микроконтроллеров, включающих в себя ассемблер, дизассемблер, симулятор и профилировщик. Система ориентирована на интерактивную работу и имеет мощный графический интерфейс. Схема работы MetaDSP заключается в следующем: пользователь составляет файл описания архитектуры, который транслируется в файлы спецификации на языке ISE. На основе этих описаний генерируются требуемые инструменты.

Авторами статьи [7] был разработан язык описания архитектуры системы команд Rosetta. Язык основан на регулярных выражениях, позволяющих с

помощью одного описания задать группу инструкций, имеющих сходный тип битовых полей. Перед генерацией декодера производится предобработка, и каждое из описаний транслируется в группу задаваемых им кодировок. В процессе предобработки в заданные описания подставляются все допустимые значения параметров. Из полученного набора кодировок строится дерево. Кодировки в дереве группируются при наличии в них одинаковых битовых последовательностей; для каждой группы в дерево добавляется вершина. Таким образом, в листьях дерева хранятся конкретные кодировки. Полученное дерево сжимается, т.е. в нем выделяются общие поддеревья, и на его основе генерируется декодер (в виде С-кода).

Более сложные варианты деревьев разбора представлены в [8] и [9]. Построение дерева декодирования, описанное в работе [8], основывается на градиентном алгоритме, т.е. в каждом новом поколении ветвление производится по максимальному различающемуся числу значащих битов. В [9] развиваются идеи, предложенные в предыдущей работе: при построении дерева разбора учитываются не только значащие, но и незначащие биты. Последние определяют вероятность обнаружения задаваемой ими инструкции и, таким образом, определяют положение соответствующей вершины в дереве разбора.

Проведенный обзор работ позволяет сделать вывод о необходимости реализации нового инструмента, в котором бы отсутствовали недостатки рассмотренных работ. В частности, почти все рассмотренные инструменты требуют высокой квалификации от оператора. Форматы файлов спецификации архитектуры не только сложны для написания, но и предполагают наличие у оператора глубоких знаний специфики описываемой архитектуры. Интерактивность систем облегчает взаимодействие с пользователем, но, вместе с тем, принуждает его к постоянному контролю за системой.

Кроме того, все рассмотренные инструменты (за исключением MetaDSP) ориентированы на применение к крупным, широко распространенным архитектурам (Intel x86, ARM, SPARC, MIPS) и не учитывают специфики архитектуры большинства микроконтроллеров: простые способы кодирования команд (отсутствие нетривиальных преобразований операндов при кодировании), небольшое количество операндов в каждой из инструкций, и, зачастую, ограниченные наборы команд и регистров.

3. Архитектура системы

Для реализации системы требуется решение двух основные подзадач:

1) генерация описания кодировок команд целевой архитектуры;
2) машинно-независимое декодирование.

При решении первой подзадачи каждой ассемблерной инструкции целевой архитектуры ставится в соответствие ее двоичная кодировка, строится некоторое отображение, позволяющее в дальнейшем декодировать команды данного процессора. Вторая подзадача – декодирование поступающих команд с

использованием информации, полученной в результате решения первой подзадачи.

Система состоит из трех модулей:

- модуль генерации описания кодировок команд;
- конвертер, осуществляющий преобразование файла описания кодировок команд из двоичного формата в текстовый формат и обратно;
- библиотека декодирования.

Разработанные модули могут работать независимо, однако предполагается следующий сценарий использования системы:

1) генерация описания кодировок команд;
2) анализ и модификация файла описания кодировок команд;
3) декодирование инструкций целевой архитектуры.

Диаграмма потока работ приведена на рис. 1. Подчеркнутыми надписями на схеме обозначены компоненты разработанной программной системы. Числа над стрелками обозначают порядок выполнения; числа в квадратных скобках обозначают необязательные этапы работы.

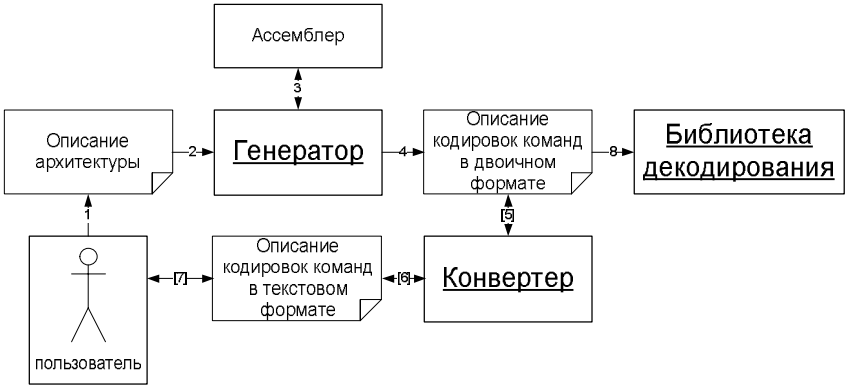


Рис. 1. Диаграмма потока работ
Fig. 1. Workflow diagram

Сначала пользователь на основе документации подготавливает файл описания архитектуры. На основе этого файла производится генерация входных данных для транслятора языка ассемблера. Полученные с его помощью кодировки обрабатываются, и в результате генерируется файл описания кодировок команд (в двоичном формате). Данный файл можно либо непосредственно использовать для декодирования, либо с помощью конвертера преобразовать в текстовый формат и вручную изменить. Затем полученный текстовый файл нужно

преобразовать обратно в двоичный формат и также использовать для декодирования.

В дальнейших разделах приводится развернутое описание разработанных компонентов. При этом используется следующая терминология:

- *команда* – управляющая конструкция целевого процессора;
- *инструкция* – ассемблерное представление корректной команды;
- *кодировка* – двоичное представление команды.

4. Генерация описания кодировок команд

В результате работы генератора по заданному пользователем описанию процессорной архитектуры с использованием ассемблера целевой архитектуры строится описание кодировок команд процессора.

4.1 Файл описания архитектуры

Описание синтаксиса языка ассемблера задается в виде шаблонов инструкций, где в соответствующие позиции вместо переменных подставляются имена регистров и целочисленные константы, а также мнемоники, задающие команду. Группировка шаблонов по синтаксическим признакам позволяет существенно упростить написание входного файла. Более того, можно задавать как неполное (возможно, с ущербом для последующего декодирования), так и избыточное описание архитектуры. Это позволяет не описывать в точности, операнды какого типа необходимы, а перебрать все возможные комбинации типов операндов и выяснить, какие из них используются в инструкции.

Пример описания синтаксиса языка ассемблера представлен на рис. 2. Любая строка, содержащая ключевые слова (*opcode*, *operand*), является шаблоном. Все символы шаблона, не входящие в состав ключевых слов, при генерации входных файлов для транслятора языка подставляются непосредственно. За каждым из шаблонов следует список относящихся к нему мнемоник. Вместо ключевого слова *opcode* генератором будут подставлены все следующие за ним мнемоники, вместо слова *operand* – операнды (регистры из списка регистров и константы). Список имен регистров процессора задается пользователем.

```
1  opcode operand, @(operand, operand)
2  MOV.B MOV.W MOV.L
3  opcode #operand, operand
4  BAND BIAND BIST BIXOR BNOT BOR BSET BTST BXOR CMP.B
5  CMP.W CMP.L DEC.W DEC.L INC.W INC.L LDC.B MOV.B
6  MOV.W MOV.L OR.B OR.W OR.L ORCSUB.W SUB.L SUBS SUBX
7  XOR.B XOR.W XOR.L XOR.C
8  opcode operand, @operand
9  BNOT BSET BTST MOVTPE
10 Opcode
11 EEPMOV RTE RTS SLEEP
```

Рис. 2. Фрагмент описания синтаксиса языка ассемблера (H8/300H)
Fig. 2. Fragment of the assembly language syntax description (H8/300H)

4.2 Алгоритм генерации описания кодировок команд

Основным этапом генерации декодера является генерация описания кодировок команд заданной архитектуры. Для получения этого описания используется ассемблер целевой архитектуры.

Алгоритм генерации описания можно разделить на 4 этапа:

1) разбор входных файлов (описание синтаксиса языка ассемблера, описание набора регистров);
2) генерация возможных перестановок операндов, сохранение необработанных кодировок;
3) обработка: определение опкодов, масок и типов операндов;
4) сохранение результатов в выходной файл.

Обычно операнды инструкции разделяют на три класса:

- константа;
- регистр или прямо указанный элемент памяти;
- элемент памяти, адресуемый косвенно.

Для получения текстового представления инструкции важна лишь синтаксическая запись операнда. Таким образом, целесообразно выделять только два типа операндов: (1) регистры; (2) константы.

Поскольку синтаксически регистры обозначаются именами, операндом-регистром является такой операнд, который может принимать значение из определенного множества текстовых строк. Таким образом, регистрами считаются любые именованные значения (в том числе, именованные константы); константами – целые шестнадцатеричные числа.

Производится перебор всех возможных комбинаций операндов; множество регистров перебирается полностью, множество констант – побитово, т.е.

подставляются только такие константы, в которых единичным является только 1 бит (степени 2), а также 0. Данный подход позволяет существенно повысить скорость анализа. Поскольку известно, что все операнды инструкции лежат на непрерывных интервалах, каждый из битов, кодирующих операнд, существенно влияет на его значение. Это позволяет перебирать не все комбинации битов, а только отдельные их позиции, в результате чего сложность перебора сокращается с экспоненциальной до линейной. Ноль также необходим, поскольку при его отсутствии невозможно определить операнды, кодируемые одним битом (например, флаги).

Кодировки одной инструкции, имеющие разную битовую длину, рассматриваются раздельно. Это также позволяет упростить и ускорить анализ, поскольку не требуется в явном виде хранить информацию о значимости каждого бита кодировки и анализировать незначащие биты.

Кодировки сохраняются в таблицу, отдельную для каждой инструкции (см. рис. 3). Каждая новая корректная кодировка добавляется в таблицу, в столбец, соответствующий порядковому номеру операнда, в строку, соответствующую его значению. Ячейки таблицы заполняются парами (опкод; маска). В данном случае опкодом называется часть кодировки, задающая операцию (т.е. код операции, КОП) и операнд, соответствующий данной ячейке.

Если ячейка пуста, кодировка сохраняется в ней; корректными считаются все ее биты, т.е. все биты маски единичные. Если ячейка не пуста, то нужно изменить маску кодировки, т.е. корректными битами должны оставаться только те, которые постоянны для данной ячейки.

После того, как таблица построена, данные из нее используются для определения кода операции и масок операндов.

Определение кода операции производится следующим образом. Вначале производится перебор всех значений каждого операнда. *instr* – некоторый столбец построенной таблицы, т.е. позиция операнда фиксирована.

Инициализация производится при обнаружении первого ненулевого значения:

```
mask = instr[i].mask;
opcode = instr[i].opcode;
```

Для каждого последующего ненулевого значения (`j != i`):

```
mask &= ~((opcode & mask) ^ (instr[j].opcode & instr[j].mask));
opcode &= mask;
```

КОП/маска инструкции определяется как *побитовое и* КОП/масок для каждого из ее операндов. Полученные таким образом КОП и маска не зависят ни от одного из операндов инструкции и определяют закодированную с их помощью команду.

	операнд 1 [reg]	операнд 2 [imm]	
r0	(0xf800; 0xfff8)	(0xf800; 0xfe0f)	0
r1	(0xf810; 0xfff8)	(0xf801; 0xfe0f)	2^0
r2	(0xf820; 0xfff8)	(0xf802; 0xfe0f)	2^1
r3	(0xf830; 0xfff8)	(0xf804; 0xfe0f)	2^2
r4	(0xf840; 0xfff8)	NULL	2^3
r5	(0xf850; 0xfff8)	NULL	2^4
r6	(0xf860; 0xfff8)	NULL	2^5
r7	(0xf870; 0xfff8)	NULL	2^6
r8	(0xf880; 0xfff8)	NULL	2^7
r9	(0xf890; 0xfff8)	NULL	2^8
r10	(0xf8a0; 0xfff8)	NULL	2^9
	.....	.....	

Рис. 3. Пример таблицы кодировок для инструкции `BLD reg, imm` *(AVR)*
Fig. 3. Example of an encoding table for the instruction `BLD reg, imm` *(AVR)*

Определение масок операндов принципиально не отличается от определения опкода. При обнаружении первого ненулевого значения операнда:

```
op_mask = 0;
temp = instr[i].opcode & instr[i].mask;
```

Для всех последующих ненулевых значений (`j != i`):

```
op_mask |= temp ^(instr[j].opcode & instr[j].mask);
temp &= ~op_mask;
```

Поскольку, в соответствии с налагаемыми ограничениями, битовые поля кодировок не пересекаются, маска операнда содержит все биты, кодирующие операнд.

4.3 Особенности трансляторов языка ассемблера

Ассемблеры для некоторых архитектур имеют особенности, некритичные для их использования при трансляции, но влияющие на генерацию файла описания. Одной из таких особенностей является дублирование кодировок. Ассемблер не считает ошибочным задание в качестве операнда слишком большого значения константы, а генерирует кодировку, где данный операнд закодирован как какая-либо меньшая допустимая константа (как правило, 0). Поскольку данная инструкция с меньшей константой также корректна, возникает дублирование, и при декодировании такая кодировка может быть разобрана как правильно, так и неправильно.

Например, для архитектуры PIC16F877A (ассемблер gpasm), некорректная из-за переполнения второго операнда инструкция INCF 0x0, 0x80 кодируется как 0xA0, что на самом деле соответствует инструкции INCF 0x0, 0x0. В таких случаях учитывается только первое (с меньшей константой) вхождение данной кодировки.

Еще одной особенностью является совпадение кодировок регистров. В некоторых случаях синтаксисом языка ассемблера допускается задание имени регистра как константы-номера в регистровом файле. В таком случае, если определенный операнд может задаваться и регистром, и константой, и типы остальных операндов попарно совпадают, необходимо выяснить, не является ли избыточной кодировка, в которой данный операнд является непосредственно заданной константой.

Для обнаружения такой ситуации используется следующий алгоритм: если для инструкции найдутся такие отображения, что для заданного операнда множество допустимых кодировок непосредственно заданных констант является подмножеством допустимых кодировок регистров, и при этом размеры кодировок равны, а также кодировки всех остальных операндов совпадают и по типу, и по значениям, необходимо пометить то отображение, где данный операнд имеет константный тип. После проверки всех отображений, помеченные отображения необходимо удалить как избыточные.

Например, инструкция архитектуры AVR (ассемблер avr-as) ADD $r2, $r3 также может быть задана как ADD 0x2, 0x3. Из-за этого полученный файл описания кодировок команд содержит дублирующиеся инструкции (см. рис. 4). После преобразования будет сгенерирован файл, согласно которому операндами инструкции ADD могут являться только регистры (что соответствует действительности).

Однако, если операнды-регистры всегда кодируются числами (как, например, в архитектуре PIC16F877A), предложенное преобразование может не работать. В инструкциях CALL и GOTO адреса перехода кодируются целыми 11-битными числами, в то время как регистровый файл содержит 512 регистров (т.е. регистр кодируется не более чем 9 битами). Таким образом, из-за того, что число регистров недостаточно, чтобы покрыть все множество значений данного операнда, в любом случае будет сгенерировано два описания каждой из данных инструкций.

```
1  add reg_1, reg_1
2  000011baaaaabbbb
3
4  add reg_1, imm
5  000011baaaaabbbb
6
7  add imm, reg_1
8  000011baaaaabbbb
9
10 add imm, imm
11 000011baaaaabbbb
```

```
1  add reg_1, reg_1
2  000011baaaaabbbb
```

Рис. 4. Фрагмент сгенерированного описания до и после преобразования (AVR)
Fig. 4. Fragment of the generated description before and after the transformation (AVR)

5. Декодирование

Процесс декодирования представляет собой обработку последовательных запросов к библиотеке декодирования. Правила декодирования задаются с помощью файла описания кодировок команд, сгенерированного ранее.

При декодировании инструкции необходимо получить результат следующего вида:

- информацию о корректности кодировки;
- если кодировка корректна, необходимо получить ее ассемблерное представление.

В свою очередь, обработка каждого запроса состоит из следующих этапов:

1) определение мнемоники;
2) определение операндов.

При этом, поскольку несколько инструкций, имеющих одинаковый опкод и маску, могут различаться в зависимости от операндов, при декодировании операндов может понадобиться повторный поиск и декодирование мнемоники. Также частным случаем является ситуация, когда одному опкоду соответствует несколько мнемоник. Это чаще всего происходит, если архитектура системы команд предусматривает наличие более высокоуровневых команд для облегчения разработки прикладных программ.

Например, в архитектуре AVR существует команда CLR (очистить бит в регистре), кодировка которой совпадает с кодировкой команды ANDI (*побитовое и* с константой), поскольку данные команды семантически эквивалентны. При возникновении подобных коллизий выбирается первая обнаруженная подходящая мнемоника.

По опкоду и маске определяется соответствующая им мнемоника. Для этого среди всех известных пар опкод/маска находится такая, которая соответствовала бы переданной кодировке, и длина которой совпадала бы с длиной переданного буфера.

После того, как определена мнемоника, декодируются операнды. Операнды-регистры и операнды-константы обрабатываются по-разному. Для констант выбираются требуемые биты переданной инструкции и сжимаются до непрерывной последовательности (т.е. незначащие биты опкода удаляются). Если же операнд имеет регистровый тип, после определения его индекса описанным выше способом осуществляется поиск в соответствующей таблице, где индексом является полученное число, значением – текстовая строка, содержащая имя регистра.

Ключевым фактором, определяющим скорость декодирования, является структура данных, используемая для хранения описания кодировок команд. Кроме того, на общее время работы декодера влияет, хотя и намного менее существенно, время, требуемое на десериализацию файла описания кодировок команд. Таким образом, подходят следующие структуры данных:

1. **Список.** Звенья представляют собой структуры, содержащие информацию о кодировках команд (опкод, маска, кодировки операндов) и инструкциях (мнемоника и типы операндов). Поиск осуществляется с помощью линейного прохода по всем кодировкам. Достоинствами данной структуры являются простота ее реализации и низкие накладные расходы, затрачиваемые на десериализацию описания инструкций. Тем не менее, из-за линейной временной сложности операции поиска требуемой инструкции, эта структура данных практически применима лишь для архитектур с небольшим набором инструкций (не более 30-50).
2. **Список деревьев**. Для реализации алгоритма поиска инструкции, имеющего менее чем линейную временную сложность, на рассматриваемом множестве должно быть задано отношение порядка. В связи с тем, что поиск осуществляется одновременно по опкоду и маске, требуется, чтобы заданная операция отношения существенно учитывала обе данных зависимости. В общем случае это невозможно. Поскольку количество масок существенно меньше, чем количество опкодов, целесообразно сгруппировать все инструкции с одинаковыми масками в двоичные деревья поиска, обеспечив таким образом логарифмическое время поиска инструкции внутри каждого дерева, т.е. при фиксированной маске. Таким образом, вершины, в которых совпадают маски, объединяются в сбалансированные деревья. Корни деревьев произвольным образом связываются в список. Поиск осуществляется по списку линейно, далее для каждого из деревьев осуществляется бинарный поиск по опкоду. Сложность десериализации при таком

подходе увеличивается незначительно – с $O(1)$ для списка до $O(m * log(k))$, где m – число различных масок, k – число инструкций с данной маской. Такая временная сложность приемлема с учетом того, что данная операция осуществляется однократно, и величины m и k сравнительно невелики.

В реализации поддерживаются обе предложенных структуры данных, но по умолчанию используется список деревьев, что обусловлено его большей эффективностью при больших размерах набора инструкций.

6. Результаты тестирования

Реализованная программная система была протестирована на архитектурах PIC16F877A, AVR, Tricore, H8/300H.

В табл. 1 приведено время генерации декодеров для различных архитектур. Временем генерации считается суммарное время, затраченное оператором на написание входных файлов (описание синтаксиса инструкций, описание набора регистров и набора инструкций), а также время работы непосредственно генератора.

Стоит отметить, что в силу экспоненциальной сложности алгоритма наибольшее влияние на время работы оказывает количество операндов в каждой из инструкций. На практике максимально допустимое число операндов – 3, что делает систему применимой для большого числа современных микроконтроллеров.

Менее существенно на время работы влияет размер набора регистров и размер набора инструкций. Большим размером набора регистров обусловлено, в частности, увеличение времени работы генератора при генерации описания PIC16F877A (512 регистров) по сравнению с AVR (32 регистра). Тем не менее, для всех рассмотренных архитектур время генерации декодера не превышает суток и занимает существенно меньше времени, чем написание соответствующего инструмента вручную.

Табл. 1. Время генерации файла описания кодировок команд
Table 1. Time of generation of an encoding description file

Архитектура	Время работы, чч:мм		
	Пользователь	Генератор	Общее
PIC16F877A	00:30	02:18	02:48
AVR	00:30	00:25	00:55
Tricore	03:00	18:39	21:39
H8/300H	01:30	03:14	04:44

Табл. 2 и рис. 5 отражают скорость работы полученного декодера по сравнению с существующими инструментами для соответствующих архитектур.

Разработанный инструмент позволяет получать декодеры, превосходящие написанные вручную по скорости в несколько раз.

Табл. 2. Сравнительная характеристика декодеров
Table 2. Comparative characteristics of decoders

Архитектура	Скорость декодирования, инстр/с		Соотношение скоростей
	Разработанный инструмент	Эталонный декодер	
PIC16F877A	26 222.79	3 886.15 (gpdasm)	6.75
AVR	1 762.71	865.22 (avr-objdump)	2.04
Tricore	753.99	368.78 (objdump 2.13)	2.07
H8/300H	1 845.34	496.06 (h8300-hms-objdump)	3.72

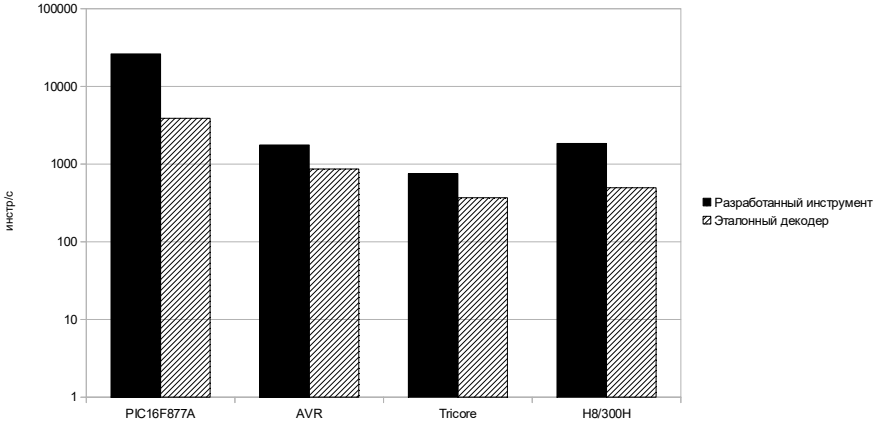


Рис. 5. Сравнительная характеристика декодеров
Fig. 5. Comparative characteristics of decoders

Тестирование производилось на машине Intel Xeon E3-1240 v2 3.40 ГГц с 8 Гб ОЗУ, платформа Ubuntu Linux 14.04 x86_64.

7. Заключение

В настоящей работе предложен метод автоматизированной генерации декодеров машинных команд; метод был реализован в виде программной системы, включающей в себя генератор описания кодировок команд, библиотеку декодирования, а также конвертер, позволяющий преобразовывать полученное описание кодировок команд из двоичного формата в текстовый и обратно, что делает систему более гибкой.

Разработанные инструменты позволяют существенно упростить поддержку большого числа различных процессорных архитектур, поскольку предложенный формат файлов не требует высокой квалификации оператора. Вместе с тем, автоматизированная генерация декодеров производится существенно быстрее, чем при ручном или полуавтоматическом (описание кодировок команд на некотором языке вручную) написании соответствующего инструмента.

Программная система была протестирована на нескольких целевых архитектурах. Скорость декодирования полученных библиотек во всех случаях выше, чем у стандартных инструментов, находящихся в открытом доступе.

Список литературы

[1]. Ramsey N., Fernandez M.F. The New Jersey Machine-code Toolkit. Proceedings of the USENIX Technical Conference, 1995. pp. 289-302.

[2]. Ramsey N., Fernandez M.F. Specifying Representations of Machine Instructions. ACM Transactions on Programming Languages and Systems, 19(3), 1997. pp. 492-524.

[3]. Hsieh W.C., Engler D.R., Back G. Reverse-Engineering Instruction Encodings. Proceedings of the General Track: 2002 USENIX Annual Technical Conference, 2001. pp. 133-145.

[4]. Collberg C.S. Reverse Interpretation + Mutation Analysis = Automatic Retargeting. Proceedings of the ACM SIGPLAN 1997 Conference on Programming Language Design and Implementation, 1997. pp. 57-70. DOI: 10.1145/258916.258922.

[5]. Collberg C.S. Automatic Derivation of Compiler Machine Descriptions. ACM Transactions on Programming Languages and Systems, 24(4), 2002. pp. 369-408. DOI: 10.1145/567097.567100.

[6]. Рубанов В.В., Михеев А.С. Интегрированная среда описания системы команд встраиваемых процессоров. Труды ИСП РАН, том 9, 2006 г., стр. 143-158.

[7]. Krishna R., Austin T. Efficient Software Decoder Design. IEEE Computer Society Technical Committee on Computer Architecture Newsletter, 2001.

[8]. Theiling H. Generating Decision Trees for Decoding Binaries. Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems, 2001. pp. 112-120. DOI: 10.1145/384197.384213.

[9]. Qin W., Malik S. Automated Synthesis of Efficient Binary Decoders for Retargetable Software Toolkits. Proceedings of the 40th Annual Design Automation Conference, 2003. pp. 764-769. DOI: 10.1109/DAC.2003.1219122.

Automated generation of machine instruction decoders

N.Yu. Fokina <nfokina@ispras.ru>
M.A. Solovev <icee@ispras.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. This paper proposes a method of automated generation of machine instruction decoders for various processor architectures, mainly microcontrollers. Only minimal, high-level

input from user is required: a set of assembly instruction templates and a list of register names. The method utilises the target architecture assembler to reveal the mapping of assembly-level instructions onto their binary encodings by mutating variables in the templates. The recovered mapping is then used as the central part of the architecture-independent decoder. The developed tools allow to significantly simplify the support of a large number of different processor architectures, since the proposed file format does not require high skill of the operator. At the same time, automated generation of decoders is performed much faster than manual or semi-automatic (description of the command character encodings in a certain language manually) development of a corresponding tool. A system based on the proposed method has been implemented and tested over a set of four microcontroller architectures: PIC16F877A, AVR, Tricore, H8/300H. The speed of decoding of our system is in all cases higher than that of standard tools that are in the public domain

Keywords: decoder; microcontroller; binary code; automated generation; instruction set.

DOI: 10.15514/ISPRAS-2018-30(2)-4

For citation: Fokina N.Yu., Solovev M.A. Automated generation of machine instruction decoders. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue. 2, 2018, pp. 65-80 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-4

References

[1]. Ramsey N., Fernandez M.F. The New Jersey Machine-code Toolkit. Proceedings of the USENIX Technical Conference, 1995. pp. 289-302.

[2]. Ramsey N., Fernandez M.F. Specifying Representations of Machine Instructions. ACM Transactions on Programming Languages and Systems, 19(3), 1997. pp. 492-524.

[3]. Hsieh W.C., Engler D.R., Back G. Reverse-Engineering Instruction Encodings. Proceedings of the General Track: 2002 USENIX Annual Technical Conference, 2001. pp. 133-145.

[4]. Collberg C.S. Reverse Interpretation + Mutation Analysis = Automatic Retargeting. Proceedings of the ACM SIGPLAN 1997 Conference on Programming Language Design and Implementation, 1997. pp. 57-70. DOI: 10.1145/258916.258922.

[5]. Collberg C.S. Automatic Derivation of Compiler Machine Descriptions. ACM Transactions on Programming Languages and Systems, 24(4), 2002. pp. 369-408. DOI: 10.1145/567097.567100.

[6]. Rubanov V.V., Mikheev A.S. Integrated Environment for Embedded Processors Instruction Set Description. Trudy ISP RAN/Proc. ISP RAS, 2006, vol. 9, pp. 143-158 (in Russian).

[7]. Krishna R., Austin T. Efficient Software Decoder Design. IEEE Computer Society Technical Committee on Computer Architecture Newsletter, 2001.

[8]. Theiling H. Generating Decision Trees for Decoding Binaries. Proceedings of the ACM SIGPLAN workshop on Languages, compilers and tools for embedded systems, 2001. pp. 112-120. DOI: 10.1145/384197.384213.

[9]. Qin W., Malik S. Automated Synthesis of Efficient Binary Decoders for Retargetable Software Toolkits. Proceedings of the 40th Annual Design Automation Conference, 2003. pp. 764-769. DOI: 10.1109/DAC.2003.1219122.

Алгоритм удаления невидимых поверхностей на основе программных проверок видимости

В.И. Гонахчян <pusheax@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Визуализация больших трехмерных сцен занимает существенное время. Для сокращения количества обрабатываемых объектов используются методы удаления невидимых поверхностей. В статье рассматривается семейство интерактивных методов удаления невидимых поверхностей, обладающих пространственной и временной когерентностью. Наиболее распространенным является метод с использованием аппаратных запросов видимости. Однако посылка и получение результатов запросов видимости занимают значительное время при большом количестве объектов. Предлагается алгоритм удаления невидимых поверхностей на основе программных проверок видимости относительно составленного на графическом процессоре буфера глубины. Предлагается эвристика для определения высоты иерархии, соответствующей наибольшей эффективности проверок видимости. Предложенный алгоритм позволяет сократить количество команд визуализации, что улучшает производительность визуализации трехмерных сцен с большим количеством объектов по сравнению с алгоритмом, основанным на аппаратных запросах видимости.

Ключевые слова: визуализация трехмерных сцен; проверка видимости; окто-дерево.

DOI: 10.15514/ISPRAS-2018-30(2)-5

Для цитирования: Гонахчян В.И. Алгоритм удаления невидимых поверхностей на основе программных проверок видимости. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 81-98. DOI: 10.15514/ISPRAS-2018-30(2)-5

1. Введение

Визуализация больших трехмерных сцен является сложной вычислительной задачей. Для ускорения визуализации используются методы удаления невидимых поверхностей [1], [2]. Рассмотрим наиболее распространенные методы удаления невидимых поверхностей и некоторые их недостатки при визуализации больших трехмерных сцен.

В работе [3] предложен алгоритм удаления невидимых поверхностей на основе иерархического z-буфера. Z-буфер делится на 4 части так, что в каждой из них записывается самое дальнее значение z. Процесс деления повторяется

до пикселей. Z-пирамида позволяет быстро исключить треугольники с помощью сравнения минимального значения z треугольника и максимального значения z в области. При программной реализации происходит затратное обновление пирамиды при добавлении новых объектов, что снижает эффективность при работе с большими сценами.

В работе [4] предлагается составлять BSP-дерево с объектами статических архитектурных сцен и декомпозицией по осям архитектурных сцен. Получается иерархия, которая соответствует структуре комнат в здании. Информация о видимости хранится в графе комнат и порталов. Видимость комнат определяется путем растеризации порталов. Однако для составления графа требуется много вычислительных ресурсов, и этот метод работает только на статических сценах. Коммерческая программа Umbra вычисляет воксельное представление сцены [5]. Пустые воксели используются как порталы между разными частями сцены. Программная растеризация порталов используется для определения видимости разных частей сцены. Этот алгоритм эффективно находит загороженные объекты в статических сценах и широко используется в компьютерных играх.

В работе [6] используются маски покрытия (coverage mask) для определения видимости. На вход подается массив полигонов в порядке удаления от камеры (front-to-back). Происходит рекурсивное деление изображения на квадро-дерево до тех пор, пока видимость полигонов не может быть определена для каждого квадранта. Главное преимущество этого подхода заключается в низких требованиях к памяти и отсутствие перезаписи пикселей. Однако требуется специализированное аппаратное обеспечение для эффективной реализации.

В работе [7] предлагается использовать иерархические маски покрытия для отбраковки невидимых объектов. Сначала происходит растеризация потенциально блокирующих видимость объектов, затем обход иерархии, и выполняются проверки видимости. Алгоритм позволяет делать приблизительное определение видимости, когда порог видимости меньше единицы. Главный недостаток алгоритма заключается в сложном процессе отбора блокирующих объектов, который выбирает большие объекты с маленьким числом полигонов.

В данной статье рассматриваются методы удаления невидимых поверхностей, обладающих пространственной и временной когерентностью видимости, на основе аппаратных и программных проверок видимости [8]11]. Аппаратный запрос видимости – это способ нахождения видимых граней полиэдра на графическом процессоре [12]. Запрос видимости останавливается, когда находит первую видимую грань. Пространственная когерентность позволяет определить видимость объектов в области пространства (узле дерева). Например, если здание невидимо, то объекты внутри здания также невидимы. Временная когерентность позволяет определить видимость в будущем по текущей видимости. Например, если объект виден в текущем кадре, и камера

движется непрерывно, то можно считать его видимым следующие несколько кадров.

В работе [8] рассматриваются способы оптимальной посылки аппаратных запросов видимости с использованием расширения NV_occlusion_query. Результаты видимости в текущем кадре используются в следующем кадре. Главные проблемы с производительностью возникают из-за простаивания процессора (CPU starvation) и графического процессора (GPU starvation). Результаты запросов видимости проверяются в следующем кадре, чтобы избежать простаивания процессора. Ранее видимые объекты визуализируются в начале текущего кадра, чтобы избежать простаивания графического процессора. Авторы использовали k-мерное дерево, построенное с применением эвристики на основе площади узлов дерева [13]. Запросы видимости посылаются на листья и по результатам определяется видимость верхних уровней иерархии. Этот метод подходит для визуализации сложных трехмерных сцен, если взять иерархию, которая не требует существенных затрат при движении объектов.

В работе [9] предлагается использовать программную растеризацию для проверки видимости вместо аппаратных запросов видимости. Сначала происходит растеризация треугольников больших объектов на центральном процессоре, составляется z-буфер (full sized tiled z-buffer). Затем проверяется видимость ограничивающих параллелепипедов остальных объектов сначала относительно суммарного z-буфера, который задает максимальное значение z в пределах полоски (tile). Если треугольники загорожены относительно суммарного z-буфера, то объект невидим. Иначе требуется полноценная растеризация треугольников и сравнение глубины относительно полноразмерного буфера глубины. Это метод позволяет делать проверки видимости без синхронизации с графическим процессором, но для него требуется отбор блокирующих объектов, который лучше всего делается вручную. Растеризация блокирующих объектов с большим количеством полигонов может быть довольно дорогой, а версия объекта с низким уровнем детализации дает приблизительные результаты.

В работе [10] предложен вероятностный критерий для уменьшения количества аппаратных запросов видимости, который позволяет не посылать запросы видимости, когда визуализация объектов в узле иерархии дешевле. В работе [11] предлагаются дальнейшие способы уменьшения запросов видимости путем посылки одного запроса на группу узлов дерева. Авторы использовали иерархию p-hbvo (polygon-based hierarchical bounding volume decomposition [14], которая хорошо справляется со статическими сценами, но не подходит для динамических трехмерных сцен.

В данной статье предложен алгоритм, который является развитием идей, описанных в работах [8], [9]. Для хранения объектов сцены используется окто-дерево с множественными ссылками [15]. Окто-деревья имеют многочисленные приложения в компьютерной графике и вычислительной

геометрии: удаление объектов, не попадающих в усеченную пирамиду камеры, поиск соседей, определение коллизий, планирование движения [16][17][18].

Оставшаяся часть статьи имеет следующую структуру. В разд. 2 описаны основные формулы, которые используются в программной растеризации и проверках видимости. В разд. 3 описан предлагаемый алгоритм удаления невидимых поверхностей. В разд. 4 представлены результаты сравнения производительности предлагаемого алгоритма и алгоритма на основе аппаратных запросов видимости. В разд. 5 приводятся основные выводы.

2. Проверки видимости в программном режиме

В алгоритме, предлагаемом в данной статье, используется код для программной растеризации, основанный на векторных инструкциях [19]. Поскольку скорость проверок видимости имеет большое значение, и этому уделено недостаточно внимания в работе [9], рассмотрим более подробно то, как осуществляются программные проверки видимости.

Для определения принадлежности точки треугольнику используется функция ребра $\mathrm{F}_{AB}(\mathrm{x},\mathrm{y}) = (\mathrm{y}_A - \mathrm{y}_B)\mathrm{x} + (\mathrm{x}_B - \mathrm{x}_A)\mathrm{y} + (\mathrm{x}_A\ \mathrm{y}_B - \mathrm{x}_B\ \mathrm{y}_A)$ [20]. $\mathrm{F}_{AB}(\mathrm{x},\mathrm{y}) > 0$, если точка (x,y) находится слева от отрезка AB, $\mathrm{F}_{AB}(\mathrm{x},\mathrm{y}) = 0$, если точка (x,y) находится на прямой, проходящей через AB, $\mathrm{F}_{AB}(\mathrm{x},\mathrm{y}) < 0$, если точка (x,y) находится справа от отрезка AB. Треугольник ABC состоит из трех отрезков: AB, BC, CA. Точка (x,y) принадлежит треугольнику, когда $\mathrm{F}_{AB}(\mathrm{x},\mathrm{y}) \geq 0$, $\mathrm{F}_{BC}(\mathrm{x},\mathrm{y}) \geq 0$, $\mathrm{F}_{CA}(\mathrm{x},\mathrm{y}) \geq 0$.

Глубина z(p) точки p вычисляется с помощью барицентрической интерполяции: $\mathrm{z(p)} = \alpha * \mathrm{z(A)} + \beta * \mathrm{z(B)} + \gamma * \mathrm{z(C)}$, где $\alpha = \frac{\mathrm{F}_{BC}(\mathrm{p})}{2*\mathrm{S}_{ABC}}$, $\beta = \frac{\mathrm{F}_{CA}(\mathrm{p})}{2*\mathrm{S}_{ABC}}$, $\gamma = \frac{\mathrm{F}_{AB}(\mathrm{p})}{2*\mathrm{S}_{ABC}}$. Поскольку $\alpha + \beta + \gamma = 1$, $\mathrm{z(p)} = \mathrm{z(A)} + \beta * (\mathrm{z(B)} - \mathrm{z(A)}) + \gamma * (\mathrm{z(C)} - \mathrm{z(A)})$.

Точка p является видимой, если $\mathrm{z(p)} \leq \mathrm{depth\_buffer(p)}$; ближняя плоскость соответствует 0, дальняя плоскость соответствует 1 в буфере глубины. Треугольник ABC считается видимым, если видима хотя бы одна точка из ограничивающего прямоугольника треугольника ABC. Октант окто-дерева считается видимым, если виден хотя бы один треугольник октанта.

Для эффективных проверок видимости используются следующие методы: проекция вершин AABB, инкрементальное обновление коэффициентов функции ребра, использование векторных инструкций SSE. Рассмотрим более подробно каждую из них.

Для наивной проекции вершин AABB требуется умножить матрицу проекции на 8 вершин (128 умножений). Рассмотрим две вершины $\mathrm{v}_0(\mathrm{x},\mathrm{y},\mathrm{z})$, $\mathrm{v}_1(\mathrm{x} + \Delta\mathrm{x},\mathrm{y},\mathrm{z})$. Заметим, что $\mathrm{M} * \mathrm{v}_1 = \mathrm{M} * \mathrm{v}_0 + \mathrm{M} * (\Delta\mathrm{x}, 0, 0)$. Для вычисления $\mathrm{M} * \mathrm{v}_1$ дополнительно требуется посчитать $\mathrm{m}_{00} * \Delta\mathrm{x}, \mathrm{m}_{10} * \Delta\mathrm{x}, \mathrm{m}_{20} * \Delta\mathrm{x}, \mathrm{m}_{30} * \Delta\mathrm{x}$.

Таким образом, для трансформации AABB требуется $16 + 4 * 3 = 28$ умножений, что значительно ускоряет трансформацию октантов.

Рассмотрим метод вычисления функции ребра для соседних пикселей треугольника (x, y), (x+1, y), $F_{AB}(x + 1, y) = (y_A - y_B)(x + 1) + (x_B - x_A)y + (x_A\, y_B - x_B * y_A) = F_{AB}(x, y) + (y_A - y_B)$. Таким образом, для перехода к следующему пикселю на данной строке требуется одно сложение. Для треугольника ABC нужно посчитать коэффициенты $(y_A - y_B), (x_B - x_A), (x_A\, y_B - x_B\, y_A)$ один раз, а потом просто добавлять их при обходе пикселей внутри ограничивающего прямоугольника ABC.

Векторные инструкции SSE позволяют выполнять 4 операции за одну инструкцию. AABB состоит из 12 треугольников. Поместив координаты вершин треугольников в SSE регистры, можно за одну инструкцию вычислять 4 коэффициента функции ребра (для 4 треугольников). При обходе пикселей треугольника рассматриваются сразу 4 соседних пикселя.

Рассмотрим суммарный буфер глубины, в котором проставлено максимальное значение z для каждой группы пикселей (tile). Для отбора видимых объектов с отбраковкой на основе буфера глубины производится проекция вершин объекта на экран и проверка относительно суммарного буфера глубины. Если находятся треугольник и группа пикселей, пересекающаяся с треугольником, такие что $\min Z_\Delta < \max Z_{tile}$, то треугольник считается видимым относительно суммарного буфера глубины. В этом случае делается проверка видимости каждого пикселя внутри треугольника. Если таких треугольников нет, то объект считается загороженным и вычисление глубины каждого пикселя не требуется (см. рис. 1). Проверки относительно буфера глубины осуществляются в основной памяти на CPU, что разгружает GPU.

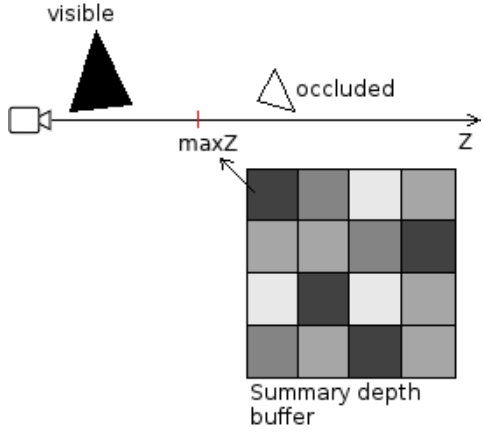


Рис. 1. Отбраковка треугольников на основе суммарного буфера глубины
Fig. 1. Rejecting triangles using summary depth buffer.

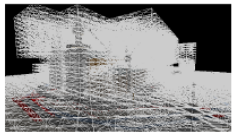
Описанные методы позволяют значительно сократить время проверок видимости октантов.

3. Алгоритм удаления невидимых поверхностей

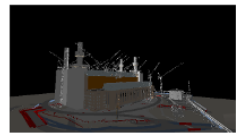
Предлагаемый алгоритм является развитием алгоритмов, описанных в работах [8], [9]. Однако, в отличие от Coherent hierarchical culling, аппаратные запросы видимости не используются. В отличие от Software occlusion culling, выполняются проверки видимости октантов в следующем кадре, т. е. учитывается пространственная и временная когерентность. Сначала рассмотрим основные шаги предлагаемого алгоритма, потом опишем отличия от алгоритма на основе аппаратных проверок видимости [21], эвристику для определения уровня иерархии, технику консервативного понижения глубины вершин октанта для устранения мерцаний.

Объекты сцены помещаются в окто-дерево с множественными ссылками. Для каждого кадра выполняются следующие шаги алгоритма. Сначала скачивается буфер глубины предыдущего кадра с графического процессора. Для этого нужно завершить визуализацию всех посланных объектов. Хотя это и вызывает простой процессора, затрачиваемое время оказывается меньше, чем затраты на посылку и прием запросов видимости при использовании аппаратных запросов видимости.

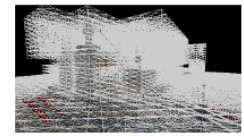
Hardware queries (АПВ)



1. Get occlusion query results that determine octant visibility.

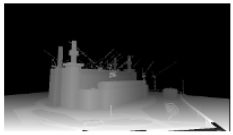


2. Render visible.

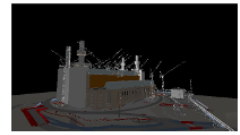


3. Send occlusion queries for each octant on last level.

Software queries (ППВ)



1. Download depth buffer of the previous frame. Perform software visibility checks.



2. Render visible.

Рис. 2. Основные этапы работы алгоритмов АПВ и ППВ
Fig. 2. Main stages of hardware and software occlusion culling algorithms.

Далее формируется массив октантов нижнего уровня, которые попадают в область видимости камеры (frustum culling). Потом выполняются программные проверки видимости относительно полученного буфера глубины, которые выполняются параллельно на всех доступных ядрах процессора с доступом к буферу глубины только на чтение. Результаты проверок видимости записываются в массив байт. Затем визуализируются объекты внутри видимых октантов.

На рис. 2 показаны этапы, которые занимают основное время при выполнении алгоритмов АПВ (аппаратные проверки видимости) и ППВ (программные проверки видимости). Более подробно АПВ описан в [21], главное отличие от ППВ – использование объектов “opengl query” для проверок видимости октантов [12]. Этап 2 занимает примерно одинаковое время, потому что результаты видимости в обоих случаях одинаковые. Этап 1 выполняется быстрее в ППВ при большом количестве уровней иерархии, потому что загрузка буфера глубины – разовая операция, которая происходит после составления буфера предыдущего кадра (см. табл. 1). Также ППВ дает прирост производительности за счет сокращения количества посылаемых на графический процессор команд (этап 3).

Табл. 1. Сравнение времени выполнения основных этапов АПВ и ППВ при визуализации сцены 2
Table 1. Comparison of execution time of hardware and software occlusion queries when rendering scene 2

Высота окто-дерева	Загрузка буфера глубины	Программные проверки видимости	Получение результатов аппаратных проверок видимости	Отправка аппаратных запросов видимости
3	1.2 мс	0.09 мс	0.24 мс	0.54 мс
4	1.2 мс	0.15 мс	1.42 мс	2.3 мс
5	1.2 мс	0.35 мс	8.25 мс	15.26 мс
6	1.2 мс	2.1 мс	57.9 мс	106.9 мс
7	1.2 мс	11.8 мс	-	-

Высота окто-дерева сильно влияет на производительность предлагаемого алгоритма. Рассмотрим эвристику для выбора оптимальной высоты окто-дерева. Пусть количество октантов $\mathrm{N} = 8^{m}\mathrm{C}$, где m – количество уровней окто-дерева, C – степень разреженности окто-дерева. Рассмотрим типовую сцену, в которой M объектов распределены равномерно по кубу из N октантов. Будем считать, что в камеру попадают три стороны куба.

Пусть $\mathrm{n} = \sqrt[3]{N}$, тогда сторона куба содержит $\frac{n^2}{N}M$ видимых объектов, три стороны содержат $\frac{\mathrm{n}^2+\mathrm{n}(\mathrm{n}-1)+(\mathrm{n}-1)(\mathrm{n}-1)}{N}M$ видимых объектов, визуализация видимых объектов занимает время $\frac{\mathrm{n}^2+\mathrm{n}(\mathrm{n}-1)+(\mathrm{n}-1)(\mathrm{n}-1)}{N}MT_{obj}$, проверки видимости занимают время NT_{check}. Предполагается, что суммарное время посылки октантов и объектов является узким местом работы алгоритма, а графический процессор справляется с визуализацией объектов. Эти формулы позволяют определить даст ли прирост производительности визуализации переход на следующий уровень.

При переходе на следующий уровень количество проверяемых октантов увеличится, а количество видимых объектов трех сторон куба уменьшится в два раза за счет деления по оси, перпендикулярной соответствующей стороне куба. Таким образом, количество проверок видимости увеличится, а количество объектов, которые посылаются на визуализацию, сократится. Это позволяет сделать примерную оценку высоты окто-дерева, которая дает минимальное время составления кадра: $(N_{next} - N_{prev})T_{check} < \frac{\mathrm{n}^2+\mathrm{n}(\mathrm{n}-1)+(\mathrm{n}-1)(\mathrm{n}-1)}{2N_{prev}}MT_{obj}$.

В табл. 2 приведены эвристические оценки высоты окто-дерева и соответствующие времена составления кадров, когда в область видимости камеры попадает вся сцена. Результаты показали, что предложенную эвристику можно использовать при визуализации трехмерных сцен с большим количеством объектов.

Табл. 2. Определение высоты окто-дерева для эффективных проверок видимости
Table 2. Determination of octree depth for effective occlusion culling

	Сцена 1	Сцена 2	Сцена 3	Сцена 4
Время составления кадра при высоте 4	77.5 мс	206.5 мс	113.0 мс	100.4 мс
Время составления кадра при высоте 5	55.1 мс	147.0 мс	96.3 мс	84.0 мс
Время составления кадра при высоте 6	54.5 мс	118.5 мс	98.0 мс	57.7 мс
Время сост. кадра при высоте 7	93.0 мс	138.7 мс	162.0 мс	125.5 мс
Эвристическая	6	6	6	6

оценка высоты окто-дерева				

Из-за того, что глубина z вычисляется по-разному на CPU и на GPU, накапливаются ошибки округления чисел с плавающей точкой, и тот же самый пиксель имеет другую глубину z в проверках видимости. Это может вызывать мерцания, когда октант попеременно то видим, то невидим. Для корректных проверок видимости нужно консервативно занизить глубину, вычисленную на CPU.

Рассмотрим задачу определения равенства двух чисел с плавающей точкой. Числа x и y можно считать равными, если $|x - y| < K * epsilon * |x + y|$, где epsilon примерно равняется 10^{-5} (разница между наименьшим числом с плавающей точкой больше единицы и единицей). Однако глубина может отличаться совсем мало, поскольку она пропорциональна $\frac{1}{z}$. Кроме того, неизвестно, какое значение K стоит брать, чтобы превысить накопившуюся ошибку округления.

Другой вариант – взять часть расстояния между ближней и дальней гранью октанта вдоль z для консервативного понижения глубины вершин. После проекции $z' = \frac{2fn}{(f-n)z} + \frac{f+n}{f-n}$, где n – расстояние до ближней плоскости, f – расстояние до дальней плоскости, z – расстояние от камеры до октанта вдоль оси Z. Посчитаем расстояние между гранями октанта: $\Delta z = \frac{2fn}{(f-n)}(\frac{1}{(z-a)} - \frac{1}{(z+a)})$, где a – характерный размер октанта. В предлагаемом алгоритме все вершины октанта приближаются к камере на расстояние $\frac{1}{10}\Delta z$. Это позволяет устранить мерцания во многих случаях. Преимущество этого подхода состоит в том, что расчет проводится не для каждого пикселя, а для всех вершин октанта один раз во время проверки видимости.

4. Сравнение производительности

Сравнивается время составления кадра при визуализации сцен с помощью двух алгоритмов, которые отличаются методами проверки видимости. В первом алгоритме используются аппаратные проверки видимости (АПВ). Во втором алгоритме, который предлагается в данной работе, используются программные проверки видимости (ППВ). Кроме того, приводятся результаты алгоритма отсечения объектов, не попадающих в область видимости камеры (Frustum Culling). Характеристики тестовой системы: Intel Core i7-7700 3600MHz 8192Kb L3, Intel HD Graphics 630, 16GB DDR4 2400MHz. Тест заключается в измерении времени составления кадра при движении камеры по сцене. Тестовые сцены (см. рис. 3–6):

- сцена 1: 10,827,713 треугольников, 71,961 объектов; примерно половину объема сцены занимают довольно большие объекты с маленьким количеством полигонов;
- сцена 2: 31,462,818 треугольников, 270,431 объектов;
- сцена 3: 11,536,541 треугольников, 109,991 объектов;
- сцена 4: 10,154,304 треугольников, 221,796 объектов; искусственная сцена, которая состоит из 36 зданий; в каждом здании имеется большая группа объектов, которая в большинстве случаев оказывается загороженной.

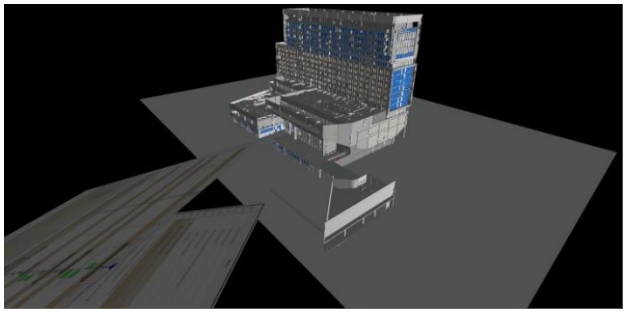


Рис. 3. Сцена 1 – архитектурная сцена из 10.8 миллионов треугольников
Fig. 3. Scene 1 – architectural scene that contains 10.8 million triangles

Рис. 4. Сцена 2 – архитектурная сцена из 31.5 миллионов треугольников
Fig. 4. Scene 2 – architectural scene that contains 31.5 million triangles

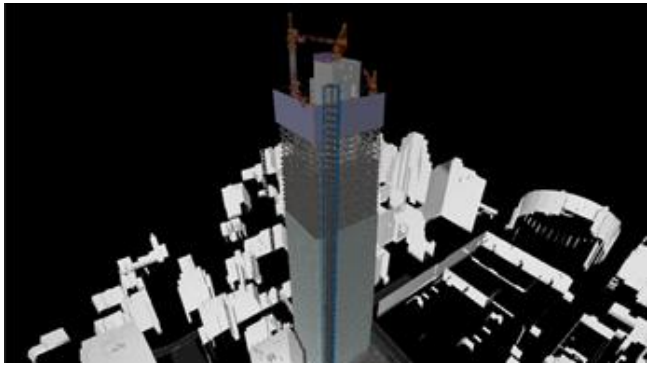


Рис. 5. Сцена 3 – архитектурная сцена из 11.5 миллионов треугольников
Fig. 5. Scene 3 – architectural scene that contains 11.5 million triangles.

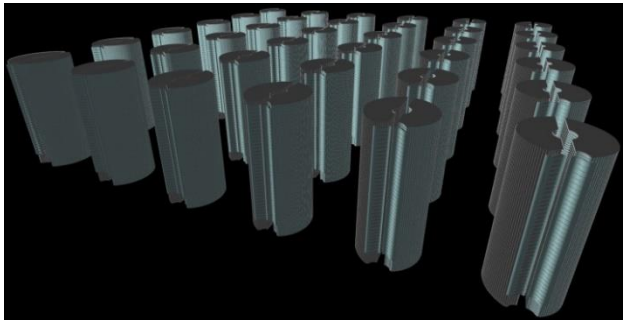


Рис. 6. Искусственная сцена со зданиями из 10.2 миллионов треугольников
Fig. 6. Artificial scene 3 with buildings having in total 10.2 million triangles

На рис. 7-10 и в табл. 3 показана производительность сравниваемых алгоритмов при проходе по тестовым сценам. Во всех тестах алгоритм ППВ показал выигрыш в производительности. Это происходит по двум причинам. Во-первых, выполнение проверок видимости в программном режиме происходит достаточно быстро, это позволило в алгоритме ППВ использовать окто-дерево с большей высотой, что, в свою очередь, сократило количество видимых объектов. Во-вторых, в ППВ посылается значительно меньше команд визуализации за счет выполнения проверок видимости на центральном процессоре.

Скачки на графиках связаны с изменением количества видимых объектов при движении камеры по сцене. Значительный объем сцены 1 занимают плоскости, на октанты которых посылается большое количество запросов видимости. За счет более быстрых проверок видимости ППВ оказывается эффективнее. В сцене 4 содержится большое количество объектов внутри зданий, и уменьшенный размер октантов позволяет сильно сократить количество визуализируемых объектов.

Хотя предложенный алгоритм дает существенный прирост производительности на рассмотренных сценах и данной конфигурации вычислительной системы, существуют и другие сценарии, в которых иерархические проверки видимости не имеют смысла. Например, когда в сцене имеется несколько тысяч объектов, которые всегда на виду, стоит использовать отсечение объектов, не попадающих в область видимости камеры, и не тратить лишние ресурсы на хранение иерархии и проверки видимости.

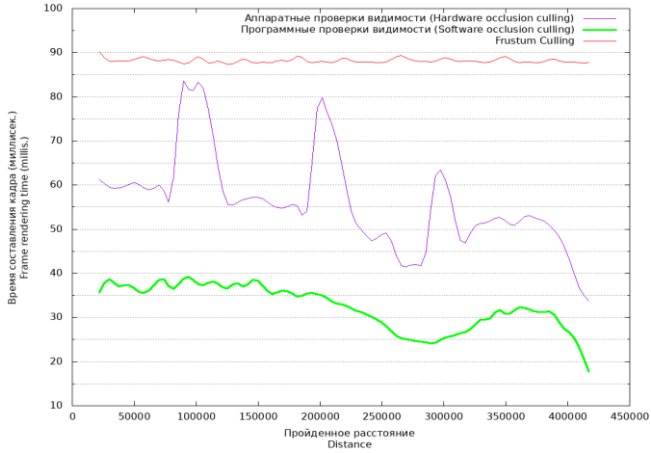


Рис. 7. Производительность во время прохода по сцене 1
Fig. 7. Performance during camera walkthrough of scene 1.

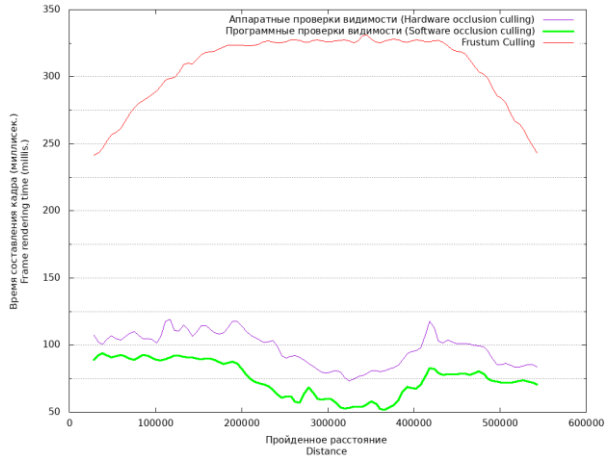


Рис. 8. Производительность во время прохода по сцене 2
Fig. 8. Performance during camera walkthrough of scene 2.

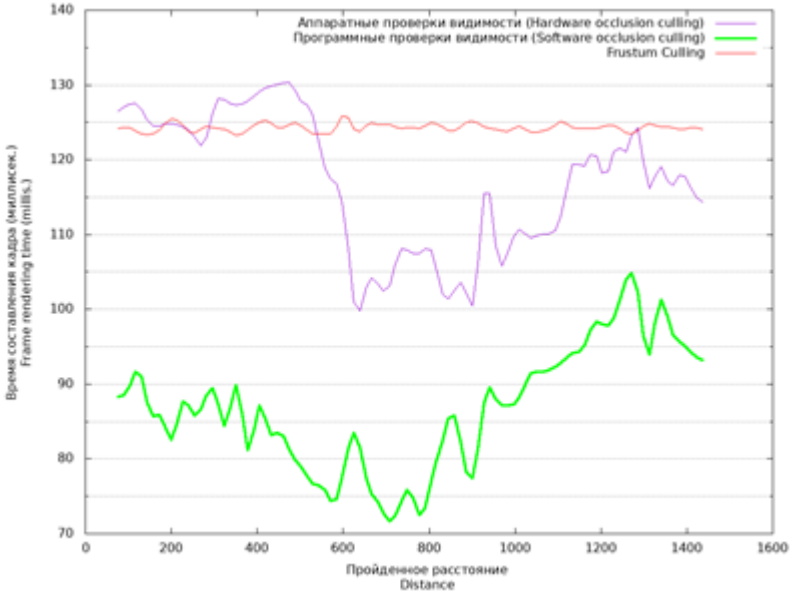


Рис. 9. Производительность во время прохода по сцене 3
Fig. 9. Performance during camera walkthrough of scene 3

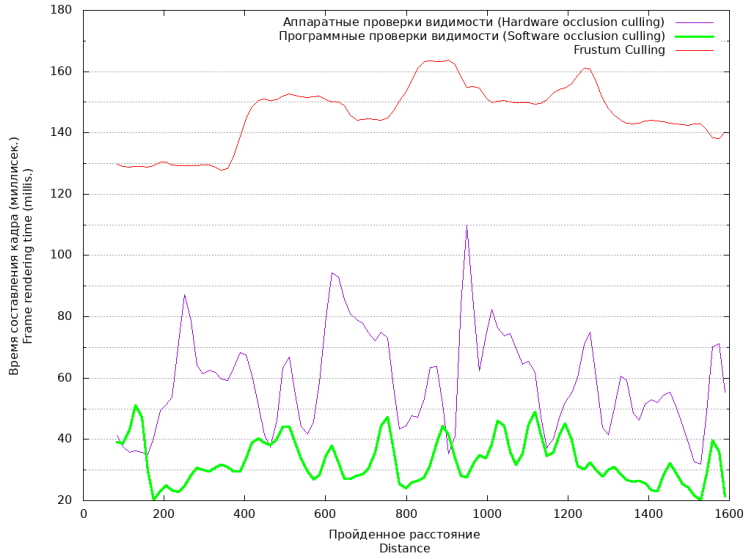


Рис. 10. Производительность во время прохода по сцене 4
Fig. 10. Performance during camera walkthrough of scene 4

Табл. 3. Среднее время составления кадра во время прохода камеры по сценам
Table 3. Average frame rendering time during scene walkthrough

Алгоритм	Сцена 1, мс	Сцена 2, мс	Сцена 3, мс	Сцена 4, мс
Frustum culling	88.2	305.0	124.3	145.2
АПВ	56.8	98.2	117.2	58.4
ППВ	33.5	77.2	86.9	34.5

5. Заключение

В работе предложен алгоритм удаления невидимых поверхностей на основе программных проверок видимости и окто-дерева. Описаны основные техники, которые позволяют ускорить проверки видимости на CPU. Предложена эвристика выбора уровня окто-дерева, который соответствует наиболее эффективным проверкам видимости и показана ее применимость на практике. Также предложена техника консервативного понижения глубины вершин октанта для устранения мерцаний. Предложенный алгоритм эффективнее справляется с визуализацией архитектурных сцен, чем алгоритм на основе аппаратных проверок видимости.

Список литературы

[1]. Bittner, J. and Wonka, P., 2003. Visibility in computer graphics. Environment and Planning B: Planning and Design, Vol. 30, No.5, pp.729–755.
[2]. Cohen-Or D. et al, 2003. A survey of visibility for walkthrough applications. IEEE Transactions on Visualization and Computer Graphic,Vol. 9, No. 3, pp. 412–431.
[3]. Greene, N. et al, 1993. Hierarchical Z-buffer visibility. Proceedings of the 20th annual conference on Computer graphics and interactive techniques. ACM.
[4]. Teller, S., Sequin, C., 1991. Visibility preprocessing for interactive walkthroughs. Computer Graphics (Proceedings of SIGGRAPH 91), Vol. 25, No. 4, pp. 61–69.
[5]. Next Generation Occlusion Culling. http://www.gamasutra.com/view/feature/164660/sponsored_feature_next_generation_.php?print=1 (дата обращения 09.04.2018).
[6]. Greene, N., 1996. Hierarchical polygon tiling with coverage masks. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. pp. 65–74.
[7]. Zhang, H. et al, 1997. Visibility culling using hierarchical occlusion maps. Proceedings of the 24th annual conference on Computer graphics and interactive techniques. pp. 77–88.
[8]. Bittner, J. et al, 2004. Coherent hierarchical culling: Hardware occlusion queries made useful. In Computer Graphics Forum, Vol. 23, No.3, pp. 615–624.
[9]. Chandrasekaran C. et al, 2013–2016. Software Occlusion Culling. https://software.intel.com/en-us/articles/ (дата обращения 09.04.2018).
[10]. Guthe, M. et al, 2006. Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries. In Eurographics Symposium on Rendering. pp. 207–214.

[11]. Mattausch, O. et al, 2008. CHC++: Coherent Hierarchical Culling Revisited. EUROGRAPHICS, Vol. 27, No. 3.
[12]. GLAPI/glBeginQuery. https://www.opengl.org/wiki/GLAPI/glBeginQuery (дата обращения 09.04.2018).
[13]. Macdonald, J. D., Booth, K. S., 1990. Heuristics for ray tracing using space subdivision. Visual Computer, Vol. 6, No. 6, pp. 153–165.
[14]. Meissner, M. et al, 2001. Generation of Decomposition Hierarchies for Efficient Occlusion Culling of Large Polygonal Models. In Vision, Modeling, and Visualization, Vol. 1, pp. 225–232.
[15]. Pharr M., Jakob W., Humphreys G. Physically based rendering: From theory to implementation. Morgan Kaufmann, 2016, 1233 p.
[16]. Morozov S., Semenov V., Tarlapan O., Zolotov V. (2018) Indexing of Hierarchically Organized Spatial-Temporal Data Using Dynamic Regular Octrees. In: Petrenko A., Voronkov A. (eds) Perspectives of System Informatics. PSI 2017. Lecture Notes in Computer Science, vol 10742, pp. 276–290.
[17]. Золотов В.А., Петрищев К.С., Семенов В.А. Исследование методов пространственного индексирования динамических сцен на основе регулярных октодеревьев. Программирование, 2016, № 6, стр. 59–66.
[18]. V.A. Semenov, K.A. Kazakov, V.A. Zolotov. Effective spatial reasoning in complex 4D modeling environments. eWork and eBusiness in Architecture, Engineering and Construction, eds. A.Mahdavi, B. Martens, R. Scherer, CRC Press, Taylor & Francis Group, London, UK, 2015, pp. 181–186.
[19]. Software Occlusion Culling Sample Application. https://github.com/GameTechDev/OcclusionCulling (дата обращения 09.04.2018).
[20]. Marschner, S. and Shirley, P., 2015. Fundamentals of computer graphics (3rd ed.). CRC Press, pp.45–49.
[21]. Gonakhchyan V. Comparison of hierarchies for occlusion culling based on occlusion queries. In Proceedings of the GraphiCon 2017 conference on Computer Graphics and Vision, pp. 32-36.

Occlusion culling algorithm based on software visibility checks

V.I. Gonakhchyan <pusheax@ispras.ru>
Ivannikov Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Rendering of 3D scenes with big number of objects is computationally intensive. Occlusion culling methods are used to decrease the number of handled objects. We consider interactive occlusion culling methods that have spatial and time coherence. We propose algorithm to increase rendering performance by using occlusion checks implemented in software mode. We propose heuristic to determine hierarchy level that corresponds to the most efficient occlusion checking. The algorithm is compared with the algorithm based on hardware occlusion queries. Checking for occlusion on CPU avoids transmission overhead between CPU and GPU and as a result improves rendering performance of 3d scenes with big number of objects. Section 1 provides an overview of related work as well as general

purposes of given paper and its structure. Section 2 describes the basic formulas that are used in software rasterization and visibility checks. Section 3 describes the proposed algorithm for removing invisible surfaces. Section 4 presents the results of comparing the performance of the proposed algorithm and the algorithm based on hardware visibility requests. Section 5 summarizes the main conclusions.

Keywords: 3d rendering; occlusion query; octree.

DOI: 10.15514/ISPRAS-2018-30(2)-5

For citation: Gonakhchyan V.I. Occlusion culling algorithm based on software visibility checks. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue. 2, 2018, pp. 81-98 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-5

References

[1]. Bittner, J. and Wonka, P., 2003. Visibility in computer graphics. Environment and Planning B: Planning and Design, Vol. 30, No.5, pp.729–755.

[2]. Cohen-Or D. et al, 2003. A survey of visibility for walkthrough applications. IEEE Transactions on Visualization and Computer Graphic,Vol. 9, No. 3, pp. 412–431.

[3]. Greene, N. et al, 1993. Hierarchical Z-buffer visibility. Proceedings of the 20th annual conference on Computer graphics and interactive techniques. ACM.

[4]. Teller, S., Sequin, C., 1991. Visibility preprocessing for interactive walkthroughs. Computer Graphics (Proceedings of SIGGRAPH 91), Vol. 25, No. 4, pp. 61–69.

[5]. Next Generation Occlusion Culling. http://www.gamasutra.com/view/feature/164660/sponsored_feature_next_generation_.php?print=1 (accessed 09.04.2018).

[6]. Greene, N., 1996. Hierarchical polygon tiling with coverage masks. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. pp. 65–74.

[7]. Zhang, H. et al, 1997. Visibility culling using hierarchical occlusion maps. Proceedings of the 24th annual conference on Computer graphics and interactive techniques. pp. 77–88.

[8]. Bittner, J. et al, 2004. Coherent hierarchical culling: Hardware occlusion queries made useful. In Computer Graphics Forum, Vol. 23, No.3, pp. 615–624.

[9]. Chandrasekaran C. et al, 2013–2016. Software Occlusion Culling. https://software.intel.com/en-us/articles/ (accessed 09.04.2018).

[10]. Guthe, M. et al, 2006. Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries. In Eurographics Symposium on Rendering. pp. 207–214.

[11]. Mattausch, O. et al, 2008. CHC++: Coherent Hierarchical Culling Revisited. EUROGRAPHICS, Vol. 27, No. 3.

[12]. GLAPI/glBeginQuery. https://www.opengl.org/wiki/GLAPI/glBeginQuery (accessed 09.04.2018).

[13]. Macdonald, J. D., Booth, K. S., 1990. Heuristics for ray tracing using space subdivision. Visual Computer, Vol. 6, No. 6, pp. 153–165.

[14]. Meissner, M. et al, 2001. Generation of Decomposition Hierarchies for Efficient Occlusion Culling of Large Polygonal Models. In Vision, Modeling, and Visualization, Vol. 1, pp. 225–232.

[15]. Pharr M., Jakob W., Humphreys G. Physically based rendering: From theory to implementation. Morgan Kaufmann, 2016, 1233 p.

[16]. Morozov S., Semenov V., Tarlapan O., Zolotov V. (2018) Indexing of Hierarchically Organized Spatial-Temporal Data Using Dynamic Regular Octrees. In: Petrenko A., Voronkov A. (eds) Perspectives of System Informatics. PSI 2017. Lecture Notes in Computer Science, vol 10742, pp. 276–290.

[17]. Zolotov V.A., Petrishchev K.S., Semenov V.A. Methods of spatial indexing of dynamic scenes based on regular octrees. Programming and Computer Software, vol. 42, No. 6, 2016, pp. 375–381. DOI: 10.1134/S0361768816060098

[18]. V.A. Semenov, K.A. Kazakov, V.A. Zolotov. Effective spatial reasoning in complex 4D modeling environments. // eWork and eBusiness in Architecture, Engineering and Construction, eds. A.Mahdavi, B. Martens, R. Scherer, CRC Press, Taylor & Francis Group, London, UK, 2015, pp. 181–186.

[19]. Software Occlusion Culling Sample Application. https://github.com/GameTechDev/OcclusionCulling (accessed 09.04.2018).

[20]. Marschner, S. and Shirley, P., 2015. Fundamentals of computer graphics (3rd ed.). CRC Press, pp.45–49.

[21]. Gonakhchyan V. Comparison of hierarchies for occlusion culling based on occlusion queries. In Proceedings of the GraphiCon 2017 conference on Computer Graphics and Vision, pp. 32-36.

Организация полностью самопроверяемой схемы встроенного контроля на основе метода логического дополнения до равновесного кода «2 из 4»

[1] *Д.В. Ефанов <TrES-4b@yandex.ru>*
[2] *В.В. Сапожников <port.at.pgups@gmail.com>*
[2] *Вл.В. Сапожников <at.pgups@gmail.com>*
[2] *Д. В. Пивоваров <pivovarov.d.v.spb@gmail.com>*
[1] *ООО «ЛокоТех-Сигнал»,*
107113, Россия, г. Москва, 3-я Рыбинская ул., д. 18, стр. 22
[2] *Петербургский государственный университет путей сообщения Императора Александра I,*
190031, Россия, г. Санкт-Петербург, Московский пр., д. 9

Аннотация. Рассматривается задача синтеза самопроверяемой схемы встроенного контроля с оптимизацией структурной избыточности на основе использования метода логического дополнения до равновесного кода «2 из 4». Разработан способ доопределения значений контрольных функций, позволяющий пошагово устанавливать их вид и при этом обеспечивать решение задачи тестирования соответствующих элементов сложения по модулю два и схемы тестера. При этом в значения функций вводятся неопределенности, что позволяет минимизировать сами функции, и соответственно, упрощать схему блока контрольной логики.

Ключевые слова: схема встроенного контроля; логическое дополнение; равновесный код; код «2 из 4».

DOI: 10.15514/ISPRAS-2018-30(2)-6

Для цитирования: Ефанов Д.В., Сапожников В.В., Сапожников Вл.В., Пивоваров Д.В. Организация полностью самопроверяемой схемы встроенного контроля на основе метода логического дополнения до равновесного кода «2 из 4». Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 99-112. DOI: 10.15514/ISPRAS-2018-30(2)-6

1. Введение

При построении отказоустойчивых компонентов систем автоматического управления широко применяют самопроверяемые схемы встроенного контроля, организуя для объекта диагностирования систему функционального

контроля [1–3]. В такой системе реализуется стратегия рабочего диагностирования, когда входные рабочие воздействия на объект диагностирования одновременно являются и тестовыми, а отключение его от работы не является необходимым [4].

Системы функционального контроля в реальных приложениях должны обеспечивать 100%-ное обнаружения ошибок из заданного класса. Причем схема встроенного контроля в составе системы функционального контроля должна быть полностью самопроверяемой: любая неисправность из заданного класса должна обнаруживаться в момент первого ее проявления на контрольных выходах в виде формирования защитной комбинации [5, 6].

При синтезе систем функционального контроля используются равномерные блочные коды с избыточностью, не превышающей количества рабочих выходов объекта диагностирования. К таким кодам относятся различные коды с суммированием (коды Бергера [7] и их модификации) и равновесные коды [8]. В процессе синтеза системы функционального контроля используют особенности обнаружения ошибок выбранным равномерным кодом. Например, классические коды Бергера и равновесные коды обладают свойством обнаружения любых монотонных и асимметричных ошибок.

Это свойство позволяет использовать данные коды в процессе построения системы функционального контроля следующим образом [9]:

1) осуществлять поиск групп монотонно (или и монотонно, и асимметрично) независимых выходов объекта диагностирования с последующим их контролем на основе соответствующего кода и объединением контрольных выходов получаемых подсхем контроля на входах самопроверяемой схемы сжатия парафазных сигналов;
2) преобразовывать структурную схему объекта диагностирования в схему с единой группой монотонно (или и монотонно, и асимметрично) независимых выходов с последующим контролем их на основе соответствующего кода.

Такой способ, как показано в современной мировой литературе по синтезу контролепригодных компонентов систем управления [10–13], применяется довольно часто.

В системе функционального контроля объект диагностирования $F(x)$ снабжается самопроверяемой схемой встроенного контроля в составе блока контрольной логики $G(x)$ и полностью самопроверяемого тестера TSC [14]. Схема блока контрольной логики строится таким образом, чтобы на выходах обоих блоков $F(x)$ и $G(x)$ в процессе нормальной эксплуатации формировались только кодовые слова заранее выбранного блочного кода. Тестер контролирует принадлежность поступающего на его входы кодового слова выбранному коду, и при его нарушении формирует сигнал ошибки [15].

В [16, 17] описана структурная схема системы функционального контроля, включающая еще один блок, помимо обозначенных выше, – блок логического

дополнения, образованный каскадом сумматоров по модулю два (*XORs*). Структура системы функционального контроля, полученная на основе метода логического дополнения, является более «гибкой», чем классическая структура [14], с точки зрения обеспечения самопроверяемости схемы контроля и структурной избыточности получаемого дискретного устройства.

2. Контроль логических устройств на основе 2/4-кода

В [23] отмечены существенные преимущества использования 2/4-кода при организации контроля логических устройств автоматики и вычислительной техники по сравнению с другими равновесными кодами. Прежде всего, они заключаются в следующем:

1) тестер 2/4-кода (2/4-*TSC*) имеет простую структуру (рис. 1) и требует для полной проверки всего четырех комбинаций: {0011; 1100; 1001; 0110} [24];
2) Для преобразования любого четырехбитного вектора, формируемого на выходах блока $F(x)$, в кодовое слово 2/4-кода потребуется изменение максимум двух функций, а значит, блок логического дополнения будет иметь наименьшую сложность, а блок контрольной логики – всего два выхода [25];
3) эксперименты [26, 27] показывают, что использование 2/4-кода для контроля многовыходных логических устройств дает меньшую по сложности схему контроля, чем любых других равновесных кодов.

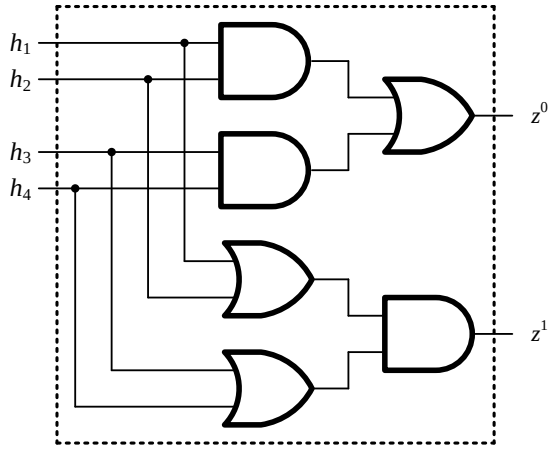


Рис. 1. Наиболее простая структурная схема 2/4-TSC
Fig. 1. The simplest structural circuit of 2/4-TSC

Структурная схема системы функционального контроля на основе метода логического дополнения до равновесного 2/4-кода изображена на рис. 2. Эта

структура является базовой и используется при контроле группы из четырех выходов. Информационный вектор $<f_4\ f_3\ f_2\ f_1>$ преобразуется в кодовое слово 2/4-кода $<h_4\ h_3\ h_2\ h_1>$ следующим образом: $h_1 = f_1$, $h_2 = f_2$, $h_3 = f_3 \oplus g_3$, $h_4 = f_4 \oplus g_4$. Таким образом, в блоке логического дополнения преобразуются только две рабочие функции на элементах XOR_3 и XOR_4.

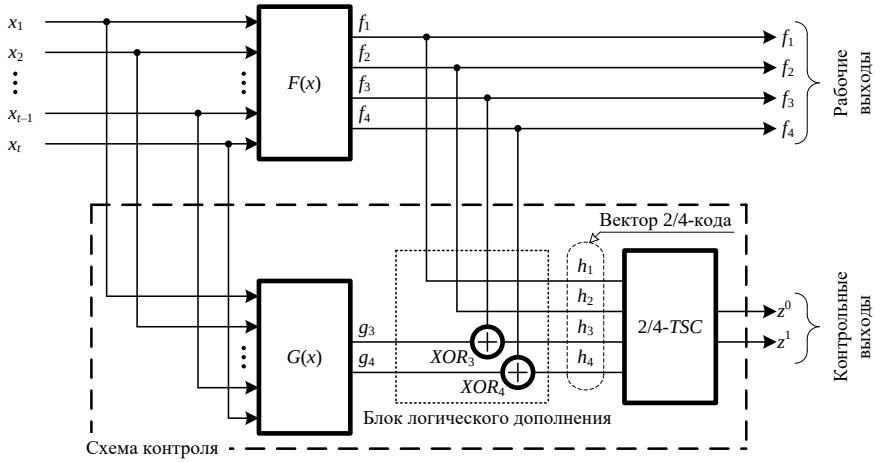


Рис. 2. Структурная схема системы контроля комбинационного логического устройства на основе 2/4-кода

Fig. 2. The structural scheme of the control system of the combination logic device based on 2/4-code

Для обеспечения полной самопроверяемости системы функционального контроля необходимо обеспечивать формирование хотя бы по разу всех кодовых слов 2/4-кода, а также формирование хотя бы по разу всех тестовых комбинаций каждого элемента *XOR* – комбинаций {00; 01; 10; 11} [28]. Кроме того, блоки $F(x)$ и $G(x)$ должны быть проверяемыми, то есть любая неисправность из заданного класса должна проявляться на выходах в виде искажений значений. Далее приводится способ построения системы функционального контроля, учитывающий обозначенные выше особенности тестирования ее компонентов.

3. Способ построения ССВК на основе 2/4-кода

Рассмотрим следующий подход к построению самопроверяемой схемы контроля на основе 2/4-кода, ориентированный на доопределении значений двух контрольных функций пошагово, исходя из обеспечения условий тестируемости элементов *XOR* блока логического дополнения и 2/4-*TSC*.

Будем рассматривать способ на примере организации схемы контроля для комбинационного логического устройства, заданного в виде таблицы

истинности (табл. 1). Устройство имеет четыре входа x_1, x_2, x_3 и x_4 и четыре выхода f_1, f_2, f_3 и f_4.

Шаг 1. На начальном этапе выбираются две рабочие функции, которые не будут дополняться и будут напрямую соединены с входами тестера. Эти функции выбираются из соображений обеспечения формирования тестового множества для 2/4-*TSC* {0011; 1100; 1001; 0110}, а именно: хотя бы на одном входном наборе должны быть сформированы хотя бы по разу информационные векторы < f_1 f_2 f_3 f_4 > = < 0 0 ~ ~>, < 0 1 ~ ~>, < 1 0 ~ ~>, < 1 1 ~ ~>. В противном случае окажется невозможным формирование всех тестовых комбинаций для 2/4-*TSC*. В рассматриваемом примере в качестве не дополняемых функций могут быть выбраны функции f_1 и f_2. Таким образом, $h_1 = f_1$ и $h_2 = f_2$.

Шаг 2. Определяются условия дополнения функции f_3, которые должны учитывать и предыдущий шаг алгоритма. Значение функции h_3 должно выбираться исходя из выражения

$$h_3 = \begin{cases} 0, & \text{если} \quad h_1 h_2 = 1; \\ 1, & \text{если} \quad h_1 \vee h_2 = 0; \\ \sim, & \text{если} \quad h_1 \oplus h_2 = 1. \end{cases}$$

Шаг 3. Из предыдущего шага следует, что:

$$g_3 = \begin{cases} f_3, & \text{если} \quad h_3 = 0; \\ \overline{f_3}, & \text{если} \quad h_3 = 1; \\ \sim, & \text{если} \quad h_3 = \sim . \end{cases}$$

Выполнение третьего шага для рассматриваемого примера представлено в табл. 1.

Табл. 1. Первый этап получения значений контрольных функций
Table 1. The first stage of obtaining the values of check functions

No.	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	h_1	h_2	h_3	h_4	g_3	g_4
0	0	0	0	0	0	1	1	0	0	1	~		~	
1	0	0	0	1	0	0	0	1	0	0	1		1	
2	0	0	1	0	0	1	0	1	0	1	~		~	
3	0	0	1	1	1	0	1	1	1	0	~		~	
4	0	1	0	0	0	0	1	0	0	0	1		0	
5	0	1	0	1	0	0	0	0	0	0	1		1	
6	0	1	1	0	0	1	1	0	0	1	~		~	
7	0	1	1	1	0	0	1	0	0	0	1		0	
8	1	0	0	0	1	0	1	1	1	0	~		~	

9	1	0	0	1	1	1	0	1	1	1	0		0	
10	1	0	1	0	0	0	0	0	0	0	1		1	
11	1	0	1	1	1	0	0	1	1	0	~		~	
12	1	1	0	0	0	1	1	1	0	1	~		~	
13	1	1	0	1	0	0	0	0	0	0	1		1	
14	1	1	1	0	0	0	1	1	0	0	1		0	
15	1	1	1	1	0	0	1	0	0	0	1		0	

Шаг 4. Частично определенная функция g_3 минимизируется, например, по методу Карно (рис. 3). Минимизированная функция g_3 имеет вид:

$$g_3 = \overline{x_1}\,\overline{x_2} \vee \overline{x_2}x_3 \vee x_2\overline{x_3}x_4.$$

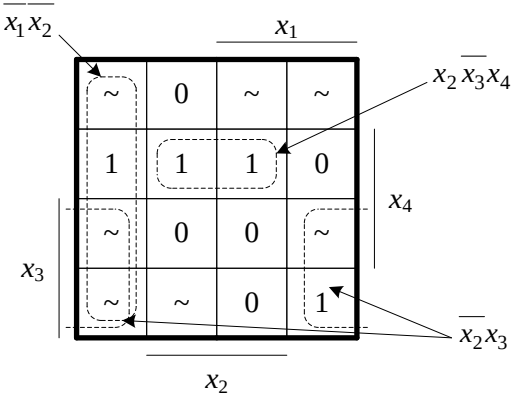


Рис. 3. Минимизация частично определенной функции g_3
Fig. 3. The minimization of a partially defined function g_3

Шаг 5. Вычисляется функция $h_3 = f_3 \oplus g_3$. Результат занесен в таблицу 2.

Табл. 2. Второй этап получения значений контрольных функций
Table 2. The second stage of obtaining the values of check functions

No.	x_1	x_2	x_3	x_4	f_1	f_2	f_3	f_4	h_1	h_2	h_3	h_4	g_3	g_4
0	0	0	0	0	0	1	1	0	0	1	0	1	1	1
1	0	0	0	1	0	0	0	1	0	0	1	1	1	0
2	0	0	1	0	0	1	0	1	0	1	1	0	1	1
3	0	0	1	1	1	0	1	1	1	0	0	1	1	0
4	0	1	0	0	0	0	1	0	0	0	1	1	0	1
5	0	1	0	1	0	0	0	0	0	0	1	1	1	1
6	0	1	1	0	0	1	1	0	0	1	1	0	0	0
7	0	1	1	1	0	0	1	0	0	0	1	1	0	1

8	1	0	0	0	1	0	1	1	1	0	1	0	0	1
9	1	0	0	1	1	1	0	1	1	1	0	0	0	1
10	1	0	1	0	0	0	0	0	0	0	1	1	1	1
11	1	0	1	1	1	0	0	1	1	0	1	0	1	1
12	1	1	0	0	0	1	1	1	0	1	1	0	0	1
13	1	1	0	1	0	0	0	0	0	0	1	1	1	1
14	1	1	1	0	0	0	1	1	0	0	1	1	0	0
15	1	1	1	1	0	0	1	0	0	0	1	1	0	1

Шаг 6. Необходимо проверить, все ли тестовые комбинации для XOR_3 сформированы после выполнения процедуры однозначного определения функции g_3. Для этого следует вычислить четыре проверяющие функции, устанавливающие наличие или отсутствие входных наборов, на которых формируются тестовые комбинации элемента сложения по модулю два:

$$p_1^3 = f_3 g_3 = \overline{x}_1\overline{x}_2\overline{x}_3 x_4 \vee \overline{x}_1\overline{x}_2 x_3\overline{x}_4 \vee \overline{x}_1 x_2\overline{x}_3 x_4 \vee$$
$$\vee\, x_1\overline{x}_2\overline{x}_3\overline{x}_4 \vee x_1\overline{x}_2 x_3 x_4 \vee x_1 x_2\overline{x}_3 x_4 \neq 0;$$

$$p_2^3 = f_3\overline{g}_3 = \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4 \vee \overline{x}_1\overline{x}_2 x_3 x_4 \neq 0;$$

$$p_3^3 = \overline{f}_3 g_3 = \overline{x}_1 x_2\overline{x}_3\overline{x}_4 \vee \overline{x}_1 x_2 x_3\overline{x}_4 \vee \overline{x}_1 x_2 x_3 x_4 \vee x_1\overline{x}_2\overline{x}_3\overline{x}_4 \vee$$
$$\vee\, x_1 x_2\overline{x}_3\overline{x}_4 \vee x_1 x_2 x_3\overline{x}_4 \vee x_1 x_2 x_3 x_4 \neq 0.$$

$$p_4^3 = \overline{f}_3\overline{g}_3 = x_1\overline{x}_2\overline{x}_3 x_4 \neq 0.$$

Так как все четыре функции не равны 0, то элемент XOR_3 тестируется.

Если же проверка показала отсутствие какой-либо тестовой комбинации для элемента XOR_3, то выполняется коррекция функции g_3.

Шаг 7. Коррекция функции g_3 требует доопределения данной функции на неопределенных значениях в соответствии с требуемой тестовой комбинацией. Для этого по формуле $\tilde{g_3} = h_1 \oplus h_2$ находятся те входные наборы, на которых функция g_3=~. Далее осуществляется доопределение функции g_3 на установленных входных наборах для обеспечения формирования соответствующей тестовой комбинации элемента XOR_3.

В случае если процедура коррекции значения функции g_3 не дает возможности формирования всех проверяющих комбинаций, осуществляется замена функции g_3 на g_4. Действия шагов 2 – 7 повторяются. В случае, если результат также не достигнут, в качестве не дополняемых функций выбирают другие функции и процедура повторяется.

Шаг 8. Определяются значения функции g_4:

$$g_4 = \begin{cases} f_4, & \text{если} \quad h_1h_2 \vee h_1h_3 \vee h_2h_3 = 1; \\ \overline{f_4}, & \text{если} \quad h_1h_2 \vee h_1h_3 \vee h_2h_3 \neq 1. \end{cases}$$

Условия выбора значений функции g_4 определяются исходя из проверки наличия в уже определенных значениях разрядов кодового слова $< h_1\, h_2\, h_3 \sim >$ наличия двух единичных значений. Так как функция g_3 однозначно определена, функция g_4 также определяется однозначно (см. табл. 2).

Шаг 9. Выполняется процедура проверки формирования полного множества тестовых комбинаций для элемента XOR_4:

$$p_1^4 = f_4 g_4 = \overline{x_1} x_2 x_3 \overline{x_4} \neq 0;$$

$$p_2^4 = f_4 \overline{g_4} = \overline{x_1 x_2 x_3}\, \overline{x_4} \vee x_1 \overline{x_2 x_3 x_4} \vee$$

$$\vee\, x_1 \overline{x_2 x_3} x_4 \vee x_1 \overline{x_2} x_3 x_4 \vee x_1 x_2 \overline{x_3 x_4} \neq 0;$$

$$p_3^4 = \overline{f_4} g_4 = \overline{x_1 x_2 x_3} x_4 \vee \overline{x_1 x_2} x_3 x_4 \vee x_1 x_2 x_3 \overline{x_4} \neq 0.$$

$$p_4^4 = \overline{f_4}\, \overline{g_4} = \overline{x_1 x_2 x_3 x_4} \vee \overline{x_1} x_2 \overline{x_3 x_4} \vee \overline{x_1} x_2 \overline{x_3} x_4 \vee \overline{x_1} x_2 x_3 x_4 \vee$$

$$\vee\, x_1 \overline{x_2 x_3 x_4} \vee x_1 x_2 \overline{x_3 x_4} \vee x_1 x_2 x_3 x_4 \neq 0.$$

Поскольку все проверяющие функции не равны нулю, элемент XOR_4 полностью тестируется.

Шаг 10. Проверяется наличие среди кодовых слов $< h_1\, h_2\, h_3\, h_4 >$ всех тестовых комбинаций 2/4-*TSC*. В рассматриваемом случае они формируются.

Таким образом, представленная последовательность действий по доопределению значений контрольных функций, позволяет однозначно установить значения контрольных функций на всех входных наборах, а значит, полностью позволяет синтезировать схему контроля.

6. Заключение

Представленный способ построения схемы встроенного контроля для комбинационных логических устройств позволяет достичь свойства ее самопроверяемости за счет последовательного доопределения значений контрольных функций.

Недостатком способа следует признать большое количество операций по доопределению и ограниченность реальным числом входных переменных (не более 30 – 35 – это пределы мощности современных вычислительных систем). При большом количестве входных переменных (более 35) следует применять декомпозицию устройства и выделение отдельных подсхем в его составе. Еще одним недостатком является то, что способ ограничен для его использования при построении схем контроля для логических устройств с малых числом

входов (например, при 3 входах), так как имеется малое число вариантов дополнений.

Способ построения схемы контроля по методу логического дополнения для 2/4-кода универсален, и с учетом того факта, что 2/4-*TSC* имеет простую структуру, позволяет синтезировать схемы встроенного контроля со структурной избыточностью, не превышающей избыточность схемы контроля при использовании метода дублирования. Такое заключение может быть сделано на основании большого количества экспериментов по применению равновесных кодов при организации самопроверяемых схем контроля, в том числе, опубликованных в известных источниках [26].

Список литературы

[1]. Kubalík P., Kubátová H. Parity Codes Used for On-Line Testing in FPGA. Acta Polytechnika, 2005, Vol. 45, No. 6, pp. 53-59.

[2]. Ubar R., Raik J., Vierhaus H.-T. Design and Test Technology for Dependable Systems-on-Chip (Premier Reference Source). Information Science Reference, Hershey – New York, IGI Global, 2011, 578 p.

[3]. Borecký J., Kohlík M., Kubátová H. Parity Driven Reconfigurable Duplex System. Microprocessors and Microsystems, 2017, Vol. 52, pp. 251-260, doi: 10.1016/j.micpro.2017.06.015.

[4]. А.В. Дрозд, В.С. Харченко, С.Г. Антощук, Ю.В. Дрозд, М.А. Дрозд, Ю.Ю. Сулима. Рабочее диагностирование безопасных информационно-управляющих систем. Под ред. А.В. Дрозда и В.С. Харченко. Харьков: Национальный аэрокосмический университет им. Н. Е. Жуковского «ХАИ», 2012, 614 с.

[5]. Пархоменко П.П., Согомонян Е.С. Основы технической диагностики (оптимизация алгоритмов диагностирования, аппаратурные средства). М.: Энергоатомиздат, 1981, 320 с.

[6]. Согомонян Е.С., Слабаков Е.В. Самопроверяемые устройства и отказоустойчивые системы. М.: Радио и связь, 1989, 208 с.

[7]. Berger J.M. A Note on Error Detecting Codes for Asymmetric Channels. Information and Control, 1961, vol. 4, issue 1, pp. 68-73, doi:10.1016/S0019-9958(61)80037-5.

[8]. Freiman C.V. Optimal Error Detection Codes for Completely Asymmetric Binary Channels. Information and Control, 1962, Vol. 5, Issue 1, pp. 64-71, doi: 10.1016/S0019-9958(62)90223-1.

[9]. Ефанов Д.В., Сапожников В.В., Сапожников Вл.В. Условия обнаружения неисправности логического элемента в комбинационном устройстве при функциональном контроле на основе кода Бергера. Автоматика и телемеханика, 2017, №5, С. 152-165.

[10]. Sogomonyan E.S., Gössel M. Design of Self-Testing and On-Line Fault Detection Combinational Circuits with Weakly Independent Outputs. Journal of Electronic Testing: Theory and Applications, 1993, Vol. 4, Issue 4, Pp. 267-281, doi:10.1007/BF00971975.

[11]. Busaba F.Y., Lala P.K. Self-Checking Combinational Circuit Design for Single and Unidirectional Multibit Errors. Journal of Electronic Testing: Theory and Applications, 1994, Vol. 5, Issue 1, Pp. 19-28, DOI: 10.1007/BF00971960.

[12]. Matrosova A.Yu., Levin I., Ostanin S.A. Self-Checking Synchronous FSM Network Design with Low Overhead. VLSI Design, 2000, Vol. 11, Issue 1, Pp. 47-58, DOI: 10.1155/2000/46578.

[13]. Ostanin S. Self-Checking Synchronous FSM Network Design for Path Delay Faults. Proceedings of 15th IEEE East-West Design & Test Symposium (EWDTS`2017), Novi Sad, Serbia, September 29 – October 2, 2017, pp. 696-699, doi: 10.1109/EWDTS.2017.8110129.

[14]. Nicolaidis M., Zorian Y. On-Line Testing for VLSI – A Compendium of Approaches. Journal of Electronic Testing: Theory and Applications, 1998, №12, Pp. 7-20, DOI: 10.1023/A:1008244815697.

[15]. Piestrak S.J. Design of Self-Testing Checkers for Unidirectional Error Detecting Codes, Wrocław: Oficyna Wydawnicza Politechniki Wrocławskiej, 1995, 111 p.

[16]. Сапожников В.В., Сапожников Вл.В., Дмитриев А.В., Морозов А.В., Гессель М. Организация функционального контроля комбинационных схем методом логического дополнения. Электронное моделирование, 2002, Том 24, №6, С. 52-66.

[17]. Гессель М., Морозов А.В., Сапожников В.В., Сапожников Вл.В. Логическое дополнение – новый метод контроля комбинационных схем. Автоматика и телемеханика, 2003, №1, С. 167-176.

[18]. Saposhnikov Vl.V., Dmitriev A., Goessel M., Saposhnikov V.V. Self-Dual Parity Checking – a New Method for on Line Testing. Proceedings of 14th IEEE VLSI Test Symposium, 28 April – 1 May 1996, Princeton, NJ, USA, pp. 162-168, doi: 10.1109/VTEST.1996.510852.

[19]. Dmitriev A., Saposhnikov V., Saposhnikov Vl., Goessel M., Moshanin V., Morosov A. New Self-Dual Circuits for Error Detection and Testing. VLSI Design, 2000, Vol. 11, Issue 1, Pp. 1-21, DOI: 10.1155/2000/84720.

[20]. Göessel M., Ocheretny V., Sogomonyan E., Marienfeld D. New Methods of Concurrent Checking: Edition 1, Dordrecht: Springer Science+Business Media B.V., 2008, 184 p.

[21]. Sen S.K. A Self-Checking Circuit for Concurrent Checking by 1-out-of-4 code with Design Optimization using Constraint Don't Cares. National Conference on Emerging trends and advances in Electrical Engineering and Renewable Energy (NCEEERE 2010), Sikkim Manipal Institute of Technology, Sikkim, held during 22-24 December, 2010.

[22]. Das D.K., Roy S.S., Dmitiriev A., Morozov A., Gössel M. Constraint Don't Cares for Optimizing Designs for Concurrent Checking by 1-out-of-3 Codes. Proceedings of the 10th International Workshops on Boolean Problems, Freiberg, Germany, September, 2012, pp. 33-40.

[23]. Сапожников В.В., Сапожников Вл.В., Ефанов Д.В. Построение самопроверяемых структур систем функционального контроля на основе равновесного кода «2 из 4». Проблемы управления, 2017, №1, с. 57-64.

[24]. Сапожников В.В., Сапожников Вл.В. Самопроверяемые дискретные устройства, СПб: Энергоатомиздат, 1992, 224 с.

[25]. Sapozhnikov V., Sapozhnikov Vl., Efanov D. Concurrent Error Detection of Combinational Circuits by the Method of Boolean Complement on the Base of «2-out-of-4» Code. Proceedings of 14th IEEE East-West Design & Test Symposium (EWDTS`2016), Yerevan, Armenia, October 14-17, 2016, pp. 126-133, doi: 10.1109/EWDTS.2016.7807677.

[26]. Пивоваров Д.В. Построение систем функционального контроля многовыходных комбинационных схем методом логического дополнения по равновесным кодам. Автоматика на транспорте, 2018, Том 4, №1, С. 130-148.

[27]. Сапожников В.В., Сапожников Вл.В., Ефанов Д.В., Пивоваров Д.В. Синтез систем функционального контроля многовыходных комбинационных схем на основе метода логического дополнения. Вестник Томского государственного университета. Управление, вычислительная техника и информатика, 2017, №4, С. 69-80, doi: 10.17223/19988605/41/9.

[28]. Аксёнова Г.П. Необходимые и достаточные условия построения полностью проверяемых схем свертки по модулю 2. Автоматика и телемеханика, 1979, №9, С. 126-135.

The organization of the totally self-checking integrated control circuit based on the Boolean complement method up to «2-out-of-4» constant-weight code

[1] *D.V. Efanov <TrES-4b@yandex.ru>*
[2] *V.V. Sapozhnikov <port.at.pgups@gmail.com>*
[2] *Vl.V. Sapozhnikov <at.pgups@gmail.com>*
[2] *D.V. Pivovarov <pivovarov.d.v.spb@gmail.com>*
[1] *OOO «LocoTech-Signal»,*
18, box 22, 3ed Rybinskaya st., Moscow, Russia, 107113
[2] *Emperor Alexandex I St. Petersburg state transport university,*
9, Moscowsky ave., St. Petersburg, Russia 190031

Abstract. The article considers the problem of the synthesis of a self-checking integrated control circuit with optimization of structural redundancy using the Boolean complement method up to 2-out-of-4 constant-weight code. A method for determining the values of control functions is developed, which makes it possible to set their appearance step by step, this ensures the solution of the problem of testing the corresponding elements of addition by modulo two and the tester circuit. In this case, uncertainties are introduced into the values of functions, which makes it possible to minimize the functions themselves, and, accordingly, simplify the circuit of check logic block. The method of constructing the control scheme by the method of logical addition for the 2/4-code is universal, and taking into account the fact that 2/4-TSC has a simple structure, it allows synthesizing embedded control schemes with structural redundancy not exceeding the redundancy of the control scheme when using the duplication method. Such a conclusion can be made on the basis of a large number of experiments on the use of equilibrium codes in the organization of self-verified control schemes

Keywords: integrated control circuit; Boolean complement; constant-weight code; «2-out-of-4» code.

DOI: 10.15514/ISPRAS-2018-30(2)-6

For citation: Efanov D.V., Sapozhnikov V.V., Sapozhnikov Vl.V., Pivovarov D.V. The organization of the totally self-checking integrated control circuit based on the Boolean complement method up to «2-out-of-4» constant-weight code. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue. 2, 2018, pp. 99-112 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-6

References

[1]. Kubalík P., Kubátová H. Parity Codes Used for On-Line Testing in FPGA. Acta Polytechnika, 2005, Vol. 45, No. 6, pp. 53-59.

[2]. Ubar R., Raik J., Vierhaus H.-T. Design and Test Technology for Dependable Systems-on-Chip (Premier Reference Source). – Information Science Reference, Hershey – New York, IGI Global, 2011, 578 p.

[3]. Borecký J., Kohlík M., Kubátová H. Parity Driven Reconfigurable Duplex System. Microprocessors and Microsystems, 2017, Vol. 52, pp. 251-260, doi: 10.1016/j.micpro.2017.06.015.

[4]. Objects and Methods of On-Line Testing for Safe Instrumentation and Control Systems / A.V. Drozd, V.S. Kharchenko, S.G. Antoshchuk, Ju.V. Drozd, M.A. Drozd, Yu.Yu. Sulima. Kharkov, National Aerospace University "KhAI", 2012, 614 p. (in Russian).

[5]. Parkhomenko P.P., Sogomonyan E.S. Technical Diagnosis Fundamentals (Diagnostic Algorithm Optimization, Apparatus Means). Moscow: Energoatomizdat, 1981, 320 p. (in Russian).

[6]. Sogomonyan, E.S., Slabakov E.V. Self-Checking Devices and Fault-Tolerant Systems. Moscow: Radio & Communication, 1989, 208 p. (in Russian).

[7]. Berger J.M. A Note on Error Detecting Codes for Asymmetric Channels. Information and Control, 1961, vol. 4, issue 1, pp. 68-73, doi:10.1016/S0019-9958(61)80037-5.

[8]. Freiman C.V. Optimal Error Detection Codes for Completely Asymmetric Binary Channels. Information and Control, 1962, Vol. 5, Issue 1, pp. 64-71, doi: 10.1016/S0019-9958(62)90223-1.

[9]. Efanov D.V., Sapozhnikov V.V., Sapozhnikov Vl.V. Conditions for Detecting a Logical Element Fault in a Combination Device under Concurrent Checking Based on Berger`s Code. Automation and Remote Control, 2017, Vol. 78, Issue 5, pp. 891-901, doi: 10.1134/S0005117917050113.

[10]. Sogomonyan E.S., Gössel M. Design of Self-Testing and On-Line Fault Detection Combinational Circuits with Weakly Independent Outputs. Journal of Electronic Testing: Theory and Applications, 1993, Vol. 4, Issue 4, pp. 267-281, doi: 10.1007/BF00971975.

[11]. Busaba F.Y., Lala P.K. Self-Checking Combinational Circuit Design for Single and Unidirectional Multibit Errors. Journal of Electronic Testing: Theory and Applications, 1994, Vol. 5, Issue 1, pp. 19-28, doi: 10.1007/BF00971960.

[12]. Matrosova A.Yu., Levin I., Ostanin S.A. Self-Checking Synchronous FSM Network Design with Low Overhead. VLSI Design, 2000, Vol. 11, Issue 1, pp. 47-58, doi: 10.1155/2000/46578.

[13]. Ostanin S. Self-Checking Synchronous FSM Network Design for Path Delay Faults. Proceedings of 15th IEEE East-West Design & Test Symposium (EWDTS`2017), Novi Sad, Serbia, September 29 – October 2, 2017, pp. 696-699, doi: 10.1109/EWDTS.2017.8110129.

[14]. Nicolaidis M., Zorian Y. On-Line Testing for VLSI – A Compendium of Approaches. Journal of Electronic Testing: Theory and Applications, 1998, Issue 12, pp. 7-20, doi: 10.1023/A:1008244815697.

[15]. Piestrak S.J. Design of Self-Testing Checkers for Unidirectional Error Detecting Codes. – Wrocław: Oficyna Wydawnicza Politechniki Wrocławskiej, 1995, 111 p.

[16]. Sapozhnikov V.V., Sapozhnikov Vl.V., Dmitriev A.V., Morozov A.V., Göessel M. Organization of Functional Checking of Combinational Circuits by the Logic Complement Method, Yelektronnoje modelirovanije [Electronic Modeling], 2002, Vol. 24, Issue 6, pp. 51-66. (in Russian).

[17]. Goessel M., Morozov A.V., Sapozhnikov V.V., Sapozhnikov Vl.V. Logic Complement, a New Method of Checking the Combinational Circuits. Automation and Remote Control, 2003, Vol. 64, Issue 1, pp. 153-161, doi: 10.1023/A:1021884727370.

[18]. Saposhnikov Vl.V., Dmitriev A., Goessel M., Saposhnikov V.V. Self-Dual Parity Checking – a New Method for on Line Testing. Proceedings of 14th IEEE VLSI Test Symposium, 28 April – 1 May 1996, Princeton, NJ, USA, pp. 162-168, doi: 10.1109/VTEST.1996.510852.

[19]. Dmitriev A., Saposhnikov V., Saposhnikov Vl., Goessel M., Moshanin V., Morosov A. New Self-Dual Circuits for Error Detection and Testing. VLSI Design, 2000, Vol. 11, Issue 1, pp. 1-21, doi: 10.1155/2000/84720.

[20]. Göessel M., Ocheretny V., Sogomonyan E., Marienfeld D. New Methods of Concurrent Checking: Edition 1. – Dordrecht: Springer Science+Business Media B.V., 2008, 184 p.

[21]. Sen S.K. A Self-Checking Circuit for Concurrent Checking by 1-out-of-4 code with Design Optimization using Constraint Don't Cares. National Conference on Emerging trends and advances in Electrical Engineering and Renewable Energy (NCEEERE 2010), Sikkim Manipal Institute of Technology, Sikkim, held during 22-24 December, 2010.

[22]. Das D.K., Roy S.S., Dmitiriev A., Morozov A., Gössel M. Constraint Don't Cares for Optimizing Designs for Concurrent Checking by 1-out-of-3 Codes. Proceedings of the 10th International Workshops on Boolean Problems, Freiberg, Germany, September, 2012, pp. 33-40.

[23]. Sapozhnikov V.V., Sapozhnikov Vl.V., Efanov D.V. Design of Self-Checking Concurrent Error Detection Systems Based on "2-out-of-4" Constant-Weight Code. Problemy upravlenija [Control Sciences], 2017, Issue 1, Pp. 57-64. (in Russian).

[24]. Sapozhnikov V.V., Sapozhnikov Vl.V. Self-Checking Discrete Devices. St. Petersburg: Energoatomizdat, 1992, 224 p. (in Russian).

[25]. Sapozhnikov V., Sapozhnikov Vl., Efanov D. Concurrent Error Detection of Combinational Circuits by the Method of Boolean Complement on the Base of «2-out-of-4» Code. Proceedings of 14th IEEE East-West Design & Test Symposium (EWDTS`2016), Yerevan, Armenia, October 14-17, 2016, pp. 126-133, doi: 10.1109/EWDTS.2016.7807677.

[26]. Pivovarov D.V. Formation of concurrent error detection systems in multiple-output combinational circuits using the Boolean complement method based on constant-weight codes. Avtomatika na transporte [Automation on transport], 2018, Vol. 4, Issue 1, pp. 130-148. (in Russian).

[27]. Sapozhnikov V.V., Sapozhnikov Vl.V., Efanov D.V., Pivovarov D.V. Synthesis of concurrent error detection systems of multioutput combinational circuits based on Boolean complement method. Vestnik Tomskogo gosudarstvennogo universiteta: Upravlenije, vychislitel`naya technika I informatika [Tomsk State University Journal of

Control and Computer Science], 2017, Issue 4, pp. 69-80, doi: 10.17223/19988605/41/9. (in Russian).

[28]. Aksjonova G.P. Necessary and Sufficient Conditions for Design of Completely Checkable Modulo 2 Convolution Circuits. Automation and Remote Control, 1979, Vol. 40, Issue 9, pp. 1362-1369.

Обзор расширяемого протокола аутентификации и его методов

[1] *А.В. Никешин <alexn@ispras.ru>*
[2] *В.З. Шнитман vzs@ispras.ru*
[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.
[2] *Московский физико-технический институт,*
141700, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. Данная статья представляет собой обзор расширяемого протокола аутентификации (Extensible Authentication Protocol, EAP), специфицированного комитетом Internet Engineering Task Force, IETF, и предоставляющего эффективный механизм встраивания в него различных методов аутентификации, а также обзор собственно методов аутентификации EAP, часть из которых была стандартизована в спецификациях IETF. Показано разнообразие механизмов, используемых для реализации сервиса аутентификации. Работа выполнялась при поддержке РФФИ, проект № 16-07-00603 «Верификация функций безопасности и оценка устойчивости к атакам реализаций протокола аутентификации EAP».

Ключевые слова: безопасность; аутентификация; контроль доступа; EAP, методы EAP

DOI: 10.15514/ISPRAS-2018-30(2)-7

Для цитирования: Никешин А.В., Шнитман В.З. Обзор расширяемого протокола аутентификации и его методов. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 113-148. DOI: 10.15514/ISPRAS-2018-30(2)-7

1. Введение

Аутентификация является наиболее важным сервисом безопасности, поскольку все другие сервисы безопасности зависят от него. Этот сервис направлен против угрозы маскарада – угрозы, которая может сделать возможной реализацию множества угроз. Аутентификация дает гарантию подлинности, т.е. представляет собой средство получения уверенности в том, что люди или сущности являются именно теми, за кого или за что они себя выдают. Легитимный владелец некоторых идентификационных данных называется принципалом. Может потребоваться аутентифицировать принципалов различных физических видов, например, людей, части оборудования, или работающие в вычислительной системе приложения.

Существует огромное число различных методов аутентификации. Аутентификация связана со сценарием, в котором некоторая сторона (претендент) представила идентификационные данные (identity) принципала и заявляет о том, что является этим принципалом. Аутентификация позволяет некоторой другой стороне (верификатору) убедиться в том, что это заявление является легитимным. Аутентификация широко применяется в системах контроля доступа к сетям и ресурсам вычислительных систем. В этом контексте значительный интерес представляет расширяемый протокол аутентификации (Extensible Authentication Protocol, EAP), специфицированный IETF в RFC 3748 [1], обеспечивающий эффективный механизм встраивания в него различных методов аутентификации, а также собственно методы аутентификации EAP, часть из которых была стандартизована в спецификациях IETF. В данной статье в разделах 2 и 3 представлены обзоры протокола EAP и методов EAP, соответственно.

Работа выполнялась при поддержке РФФИ, проект № 16-07-00603 «Верификация функций безопасности и оценка устойчивости к атакам реализаций протокола аутентификации EAP».

2. Основные особенности EAP

Протокол EAP относится к протоколам аутентификации, поддерживающим расширения своей функциональности за счет добавления новых методов [1]. EAP обычно работает непосредственно на базе протоколов канального уровня типа PPP [2] или IEEE 802 [3], не требуя использования протокола IP [4]. EAP может применяться на выделенных и коммутируемых каналах, как в проводных, так и в беспроводных сетях. Данный протокол использует пошаговую схему обработки сообщений: новый запрос отправляется только после получения ответа на предыдущий. Поэтому данный протокол не предназначен для передачи больших объемов данных. EAP обеспечивает собственную поддержку повторной передачи сообщений и избавления от дубликатов, но она основана на гарантированном порядке доставки сообщений протоколом нижележащего уровня. Соответственно, обработка пакетов, доставленных с нарушением порядка, не поддерживается.

Архитектура EAP использует следующие роли для сетевых узлов:

- аутентификатор (authenticator) –сетевой узел, начинающий процесс аутентификации;
- партнер (peer) – сетевой узел, отвечающий на запросы аутентификатора;
- внутренний сервер аутентификации (backend authentication server) – выделенный сервер, предоставляющий услуги аутентификации (выполняет методы EAP) для аутентификатора;
- сервер EAP – объект, реализующий методы EAP; он либо размещается на сервере аутентификации, либо является частью

аутентификатора.

Одним из преимуществ архитектуры EAP является ее гибкость. Выбор конкретного механизма аутентификации происходит после того, как аутентифицирующая сторона (аутентификатор) получит дополнительную информацию от партнерского узла. Вместо необходимости каждый раз обновлять аутентификатор для поддержки нового метода аутентификации архитектура EAP позволяет использовать выделенный сервер, который поддерживает различные методы аутентификации, а сам аутентификатор просто пересылает сообщения от других узлов такому серверу.

Общая схема работы протокола выглядит следующим образом (рис. 1). Клиент (партнер) запрашивает доступ к некоторому ресурсу, обращаясь к системе доступа (аутентификатору). Аутентификатор передает запрос с данными клиента серверу EAP. Сервер EAP запрашивает дополнительные данные у клиента. Обмен сообщениями между клиентом и сервером EAP продолжается до тех пор, пока выбранный метод аутентификации не завершится успешно или с ошибкой. Аутентификатор на основании результата аутентификации, полученному от сервера EAP, принимает решение о предоставлении клиенту доступа к запрошенному ресурсу. Таким образом, аутентификатор исполняет роль посредника между клиентом и сервером EAP.

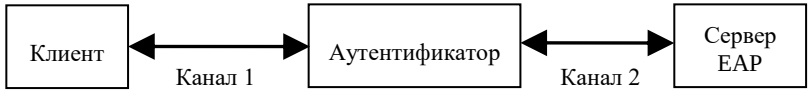


Рис. 1. Общая схема работы EAP
Fig. 1. General flow chart of EAP

Первый коммуникационный канал между клиентом и аутентификатором может быть как проводным, так и беспроводным. Довольно часто в роли клиента выступает мобильное устройство, а в роли аутентификатора – точка доступа. Пересылка пакетов EAP между клиентом и аутентификатором осуществляется посредством инкапсуляции пакетов EAP в протокол нижележащего уровня, например, в протокол PPP при использовании каналов точка-точка, или в протокол EAPOL (EAP over LAN) в сетях IEEE 802 [5]. Второй канал между аутентификатором и сервером EAP, как правило, является проводным, однако в некоторых случаях (например, в сценариях роуминга), между ними могут размещаться дополнительные объекты пересылки сообщений.

Указанная схема допускает различные вариации. Например, сам сервер EAP может располагаться как на аутентификаторе, так и на выделенном узле. Второй вариант позволяет упростить управление доступом к нескольким разделяемым ресурсам, что очень важно в целом ряде случаев. Многие сетевые устройства обычно имеют довольно ограниченные аппаратные

ресурсы и, например, не могут хранить в памяти информацию о большом числе пользователей. Кроме того, число пользователей может регулярно меняться, и возникает необходимость в единой базе данных. В такой системе от аутентификатора конкретного ресурса требуется только поддержка базовой функциональности протокола EAP. А реализация всех методов аутентификации и база данных с информацией обо всех пользователях и их правах доступа находятся в единой подсистеме – сервере аутентификации.

Пересылка пакетов EAP между внутренним сервером аутентификации и клиентом осуществляется посредством инкапсуляции пакетов EAP в протокол аутентификации, авторизации и учета (Authentication, Authorization, and Accounting, AAA), который выполняется между аутентификатором и внутренним сервером аутентификации. В качестве протокола AAA обычно применяются протоколы RADIUS [6] и Diameter [7]. При этом, однако, увеличивается объем сетевого трафика.

Возможна и комбинированная схема, в которой аутентификатор поддерживает некоторые методы аутентификации, а для других методов используется выделенный сервер. Также сервер EAP может не хранить данные для аутентификации клиентов и обращаться за ними к внешней базе данных. Таким образом, одним из важных свойств EAP является независимость от режима работы аутентификатора, т.е. любой метод EAP работает одинаково во всех аспектах независимо от того, работает ли аутентификатор в режиме ретрансляции или нет.

Другими важными свойствами протокола EAP являются независимость от среды, независимость от метода и независимость от набора шифров.

Протокол EAP первоначально разрабатывался для использования с протоколом точка-точка (Point-to-Point Protocol, PPP) [2]. Впоследствии его начали использовать для аутентификации доступа в проводных сетях IEEE 802 [5] и в беспроводных сетях IEEE-802.11 [8] и IEEE-802.16e [9]. Он применяется также в качестве одного из способов аутентификации в протоколе управления ключами в Интернет (Internet Key Exchange Protocol version 2, IKEv2) [10]. Таким образом, одной из целей создания EAP являлось обеспечение функционирования его методов поверх любого нижележащего уровня, т.е. методы EAP в процессе своего выполнения не должны пользоваться информацией конкретного нижележащего уровня, например, MAC-адресами.

Независимость от метода означает, что благодаря обеспечению режима ретрансляции аутентификатор может поддерживать любой метод, реализованный на партнере и сервере, а не только локально реализованные методы.

По существу, независимость от набора шифров обеспечивает независимость от среды. Поскольку наборы шифров разных нижележащих уровней отличаются друг от друга, для обеспечения независимости от среды требуется,

чтобы экспортируемый ключевой материал имел достаточную длину и энтропию для работы с любым набором шифров.

При аутентификации по протоколу EAP могут одновременно использоваться разные среды передачи данных и, как следствие, EAP будет выполняться через разные стеки коммуникационных протоколов. Ниже на рис. 2 показан пример стеков протоколов при использовании EAP поверх беспроводного канала (WLAN) между клиентом и аутентификатором (точка доступа) и проводного канала (LAN) между аутентификатором и сервером EAP. При этом, если аутентификатор работает в режиме ретрансляции, то ему не требуется поддерживать сами методы аутентификации, поэтому у него нет уровня методов EAP.

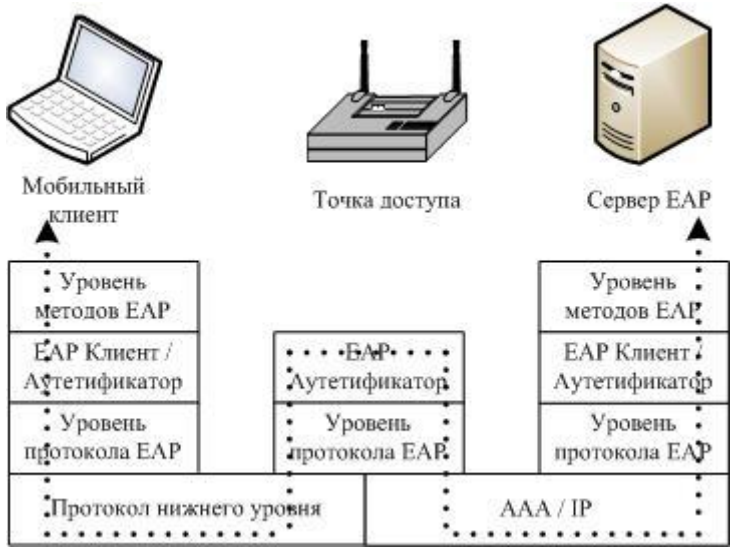


Рис. 2. Стеки протоколов EAP
Fig. 2. EAP Protocol Stacks.

Можно отметить следующие преимущества протокола.

- Протокол EAP может поддерживать множество механизмов аутентификации без предварительного согласования используемого метода.
- Устройства сетевого доступа не обязаны понимать каждый метод аутентификации и могут действовать как посредники для внутреннего сервера аутентификации. Аутентификатор, например, может поддерживать часть методов EAP или самостоятельно обеспечивать аутентификацию только локальных сетевых узлов, и, в то же время,

работать в режиме посредника для внешних узлов и непонятных методов аутентификации.

- Разделение аутентификатора и внутреннего сервера аутентификации упрощает управление доступом к сети и политиками безопасности.

3. Классификация и обзор методов EAP

В настоящее время протокол EAP широко используется в самых разных сетевых средах. В IANA (Internet Assigned Numbers Authority) зарегистрировано несколько десятков расширений данного протокола, реализующих широкий набор методов аутентификации и использующих различные средства аутентификации (пароли, сертификаты, токены и др.) [11].

В данном обзоре рассматриваются только методы EAP, спецификации которых зафиксированы в документах IETF RFC. Другие методы EAP, указанные в [11], либо остались на уровне черновых документов IETF (ietf-drafts), не получив статуса IETF RFC, либо вообще не документированы.

Спецификация EAP [1] определяет четыре основных типа сообщений: *Request* (запрос), *Response* (ответ), *Success* (успех), *Failure* (неудача)

Сообщения *Request* и *Response* содержат однобайтовое поле *Type* предназначенное для согласования метода аутентификации. Несколько значений поля *Type* зарезервированы для служебных сообщений:

1 Identity

Используется для запроса идентификатора клиента. Спецификация требует использовать данный тип запроса исключительно для целей маршрутизации и выбора подходящего метода EAP. Как правило, это первое сообщение от аутентификатора клиенту.

2 Notification

Необязательный тип сообщений. Используется для передачи аутентификатором сообщений, выводимых на экран клиента. Может передаваться в любое время во время обмена EAP.

3 Nak (Response only)

Используется в ответных сообщениях партнера, когда предложенный аутентификатором метод EAP не поддерживается. Партнер может указать в ответном сообщении желаемые методы EAP.

254 Expanded Types

Используется для расширения множества используемых методов EAP (когда множество значений однобайтового поля *Type* исчерпано). Также может использоваться для согласования специальных методов EAP, не предназначенных для общего использования (например, методов, ориентированных на конкретное оборудование).

255 Experimental use

Данный тип сообщений используется для экспериментов и тестирования (например, разработчиками нового метода EAP). Спецификация никак не ограничивает формат таких сообщений.

Остальные значения поля *Type* определяют конкретный метод EAP. Таким образом, стандарт EAP определяет единый механизм встраивания новых методов EAP в общую архитектуру протокола.

На рис.3 представлен типовой обмен сообщениями EAP (предполагается, что аутентификатор работает в режиме ретрансляции сообщений, т.е. все сообщения запроса, за исключением начального запроса идентификационных данных, и сообщение Success/Failure посылаются сервером EAP, а все ответные сообщения от партнера пересылаются серверу EAP). В первом сообщении аутентификатор (A) запрашивает идентификатор партнера (клиента, C), который в ответном сообщении указывает свои идентификационные данные. После получения этого ответа сервер (S) выбирает метод EAP и посылает первое сообщение соответствующего типа.

Если клиент поддерживает и принимает предложенный сервером метод EAP, он отвечает соответствующим сообщением того же типа. В противном случае клиент посылает сообщение Nak, и сервер EAP либо выбирает другой метод, либо прекращает выполнение EAP сообщением Failure. Количество парных сообщений запрос/ответ, пересылаемых между сервером EAP и клиентом, определяется выбранным методом EAP. Последнее сообщение является индикацией успешного или неудачного результата аутентификации.

C←*A* : EAP-Request / Identity
C→*S* : EAP-Response / Identity (id)

C←*S* : EAP-Request / EAP-method
C→*S* : EAP-Response / EAP-method

C←*S* : EAP-Request / EAP-method
C→*S* : EAP-Response / EAP-method

...

C←*S* : EAP- Success/Failure

Рис. 3. Типовой обмен сообщениями EAP
Fig. 3. Typical Message Flow of an EAP Execution

Для методов EAP, обеспечивающих установление ключей, спецификация определяет несколько типов ключевого материала. Все методы EAP, обеспечивающие вычисление ключей, создают главный сеансовый ключ (Master Session Key, MSK) и расширенный главный сеансовый ключ (Extended Master Session Key, EMSK). Дополнительно могут создаваться временные ключи EAP (Transient EAP Keys, TEKs), которые используются для защиты текущего обмена EAP. Ключи MSK, EMSK предназначены для использования за рамками метода EAP (например, другими приложениями).

Ключ MSK экспортируется нижележащим уровням и может транспортироваться аутентификатору для последующего вычисления дополнительных ключей, обеспечивающих защиту целостности и шифрование канала передачи данных между клиентом и аутентификатором. Ключ EMSK остается на сервере и резервируется для будущего использования. Ключи MSK, EMSK не должны использоваться непосредственно для защиты данных, а должны применяться только для вычисления других временных ключей.

Временные ключи EAP (TEKs) используются исключительно внутри конкретного метода EAP. Их количество, иерархию, вычисление и использование определяет спецификация данного метода. Например, метод EAP может использовать отдельные ключи для взаимной аутентификации и дополнительные ключи для создания криптографического канала. Для вычисления ключей обычно используется заранее распределенный ключ или методы Диффи-Хеллмана.

Представленные ниже методы можно разделить на четыре группы.

- *Методы, использующие схему «запрос-отклик» (challenge-response)*. Сервер отправляет клиенту запрос с некоторыми данными. Клиент на основе этих данных и собственных заранее полученных удостоверяющих данных (пароль, секретный ключ, токен и др.) формирует уникальный ответ, корректность которого позволяет серверу аутентифицировать клиента.
- *Методы, использующие заранее распределенный секретный ключ (shared secret key)*, на основе которого затем создается общий ключ безопасности (например, используя обмен Диффи-Хэллмана), уникальный для данного сеанса, который, в свою очередь, используется (непосредственно или через дополнительные криптографические ключи) для защиты сетевого обмена. Некоторые методы из других групп также можно отнести к данной группе (например, EAP-AKA использует как механизм challenge-response, так и симметричную криптографию).
- *Методы, использующие для целей аутентификации, как основного назначения методов EAP, механизмы создания защищенного канала*. Протоколы TLS и IKE предназначены, в первую очередь, для безопасной передачи данных [12], [10]. Основной функциональностью этих протоколов является создание защищенного канала между узлами сети, в процессе которого осуществляется взаимная аутентификация партнеров. Эту особенность и используют соответствующие методы EAP: начальная фаза создания защищенного канала используется исключительно для аутентификации партнеров и создания криптографических ключей, предусмотренных спецификацией EAP. Использование самого

защищенного канала для передачи данных находится за рамками спецификации рассматриваемых методов EAP.

- *Туннельные методы, создающие сначала криптографически защищенный туннель, внутри которого используются другие методы аутентификации*. На туннельные методы не распространяется запрет (по причине безопасности) спецификации RFC 3748 на использование нескольких методов аутентификации в процессе одного выполнения EAP. Туннельные методы предназначены для защиты от атак всех внутренних методов (в частности, позволяют использовать внутри туннеля небезопасные парольные методы аутентификации), а также полезны при необходимости многоуровневой аутентификации партнера (например, при последовательной аутентификации устройства, а затем – пользователя).

К сожалению, предложенная классификация не является очень строгой, поскольку, как уже было отмечено, некоторые методы можно отнести к разным группам.

3.1 Методы, использующие схему «запрос-отклик»

3.1.1 MD5-Challenge

Тип 4. RFC 3748.

Аналог протокола PPP CHAP (RFC1994) с алгоритмом MD5 [13]. Аутентификатор отправляет клиенту произвольный набор байтов. Клиент вычисляет из них контрольную сумму, используя заранее полученный секретный ключ и алгоритм MD5, и отправляет результат обратно. В настоящее время EAP-MD5 считается запрещенным: он не обеспечивает ни взаимную аутентификацию, ни вычисление ключей. Он обладает высокой уязвимостью к активным атакам прямого подбора и атакам по словарю.

3.1.2 One-Time Password (OTP)

Тип 5. RFC 3748.

Данный метод использует механизм одноразовых паролей (RFC2289) [14]. Аутентификатор отправляет клиенту некоторый набор байтов и дополнительные данные, используемые для выбора соответствующего алгоритма и дальнейших вычислений. Клиент, используя полученные данные и свой ключ безопасности, вычисляет контрольную сумму, применяя несколько раз заданную хэш-функцию.

3.1.3 Generic Token Card (GTC)

Тип 6. RFC 3748.

Данный метод использует механизм одноразовых паролей (RFC2289) [14]. Аутентификатор отправляет клиенту некоторый набор байтов и дополнительные данные, используемые для выбора соответствующего алгоритма и дальнейших вычислений. Клиент, используя полученные данные

и свой ключ безопасности, вычисляет контрольную сумму, применяя несколько раз заданную хэш-функцию.

3.1.4 EAP-POTP (EAP Protected One-Time Password Protocol)

Тип 32. RFC 4793 [15].

Данный метод использует генераторы одноразовых паролей. Это расширенный вариант метода EAP-GTC (Generic Token Card) [1]. В качестве генератора паролей может выступать как физическое устройство, так и программное обеспечение. Метод не зависит от применяемого генераторами паролей алгоритма. В отличие от EAP-GTC позволяет проводить взаимную аутентификацию и создавать криптографические ключи. Метод EAP-POTP не следует путать с EAP-OTP [1], который описывает конкретный алгоритм создания одноразовых паролей.

Базовый режим данного метода обеспечивает только аутентификацию клиента и обычно используется внутри защищенного канала. Расширенный режим применяется для взаимной аутентификации клиента и сервера, криптографической защиты сообщений и создания криптографических ключей. Также существует возможность быстрого возобновления сеанса (session resumption).

Спецификация метода определяет более десятка атрибутов, используемых в сообщениях для согласования различных параметров аутентификации: версия метода, параметры аутентификатора, алгоритм для генерации паролей, идентификатор клиента, идентификатор ключа генератора паролей (Token Key Identifier), криптографические алгоритмы и др.

Данный метод определяет пять криптографических ключей:

Ключ K_MAC используется для взаимной аутентификации партнеров и защиты целостности сообщений.

Ключ K_ENC используется для криптографической защиты некоторых данных.

Ключ SRK используется для быстрого возобновления сеанса.

Ключи MSK, EMSK предназначены для использования за рамками данного метода EAP (см. RFC 3748 [1]).

Обмен сообщениями при аутентификации клиента (рекомендуется использовать внутри защищенного канала). Как показано на рис. 4, клиент отправляет данные для аутентификации в блоке User Identifier TLV.

C←S : EAP-Request / Type=Identity
C→S : EAP-Response / Type=Identity

C←S : EAP-Request / Type=OTP-X
Version TLV: Highest=0,Lowest=0
OTP TLV: P=0,C=0,N=0,T=0,E=0,R=0
C→S : EAP-Response / Type=OTP-X
Version TLV: Highest=0

OTP TLV: P=0,C=0,N=0,T=0,E=0,R=0
Authentication Data=V1
User Identifier TLV: User Identifier=V2

C←S : EAP- Success

Рис. 4. Основной режим, односторонняя аутентификация
Fig. 4. Basic Mode, Unilateral Authentication

На рис.5 представлен обмен сообщениями при взаимной аутентификации.

C←S : EAP-Request / Type=Identity
C→S : EAP-Response / Type=Identity

C←S : EAP-Request / Type=OTP-X
Version TLV: Highest=0,Lowest=0
Server-Info TLV: N=0, Session Identifier=V1,
Server Identifier=V2, Nonce=V3
OTP TLV: P=1,C=0,N=0,T=0,E=0,R=0
Pepper Length=0, Iteration Count=V4
C→S : EAP-Response / Type=OTP-X
Version TLV: Highest=0
OTP TLV: P=1,C=0,N=0,T=0,E=0,R=0, Pepper Length=0,
Iteration Count=V4, Authentication Data=V5
User Identifier TLV: User Identifier=V6
Token Key Identifier TLV: Token Key Identifier=V7

C←S : EAP-Request/Type=OTP-X
Confirm TLV: C=0, Authentication Data=V8
Pepper Identifier=V9, Encrypted Pepper=V10
C→S : EAP-Response / Type=OTP-X
Confirm TLV: (no data)

C←S : EAP- Success

Рис. 5. Взаимная аутентификация без возобновления сеанса
Fig. 5. Mutual Authentication without Session Resumption

3.1.5 EAP-SIM (EAP GSM Subscriber Identity Modules)

Тип 18. RFC 4186 [16].

Метод использует параметры и алгоритмы SIM-карты для аутентификации и создания криптографических ключей. Данный метод основан на механизмах аутентификации GSM. GSM – стандарт мобильных сетей второго поколения. Алгоритмы A3/A8, используемые SIM-картой и GSM оператором, принимают 128-битное случайное число RAND и секретный ключ с SIM-карты Ki в качестве входных данных и выдают 32-битное хэш-значение SRES и 64-битный ключ Kc, используемые для аутентификации и шифрования данных [17].

В отличие от стандарта GSM, данный метод EAP-SIM не использует ключ Kc непосредственно для шифрования данных из-за его слабой криптографической стойкости. Вместо этого несколько значений RAND используются для генерации нескольких ключей Kc, которые затем объединяются для создания более сильных ключей безопасности. EAP-SIM обеспечивает взаимную аутентификацию партнеров, защиту целостности сообщений, криптографическую защиту некоторых данных, а также механизм быстрой повторной аутентификации.

Использование метода EAP-SIM требует наличия на стороне клиента специализированного устройства для работы с SIM-картами. При этом процесс аутентификации проходит прозрачно для клиента, ему не требуется вводить какие-либо данные.

На рис.6 представлена процедура полной аутентификации:

C←S : EAP-Request / Identity
C→S : EAP-Response / Identity

C←S : EAP-Request / SIM/Start (AT_VERSION_LIST)
C→S : EAP-Response / SIM/Start
(AT_NONCE_MT, AT_SELECTED_VERSION)

C←S : EAP-Request / SIM/Challenge
(AT_RAND, AT_MAC)

Партнер выполняет алгоритмы GSM, проверяет AT_MAC и вычисляет сеансовые ключи

C→S : EAP-Response / SIM/ Challenge (AT_MAC)

C←S : EAP- Success

Рис. 6. Полная процедура аутентификации EAP-SIM
Fig. 6. EAP-SIM full authentication procedure

Все параметры сеанса передаются в сообщениях в виде атрибутов. Спецификация определяет два десятка атрибутов, такие как версия метода EAP-SIM, запрос идентификатора клиента, значения RAND, контрольная сумма сообщения, параметры алгоритма шифрования и сами зашифрованные данные, информационные сообщения, сообщения об ошибке и др.

Сообщение EAP-Response/Identity обычно содержит IMSI (International Mobile Subscriber Identity) SIM-карты клиента [18]. Получив от клиента это сообщение, сервер EAP посылает партнеру пакет EAP-Request / SIM/Start, содержащий в атрибуте AT_VERSION_LIST список версий EAP-SIM, которые он поддерживает. Партнер в ответном сообщении указывает выбранную им версию протокола в атрибуте AT_SELECTED_VERSION, а также выбранное им случайное число в атрибуте AT_NONCE_MT.

Затем, получив от клиента сообщение EAP-Response/SIM/Start, сервер запрашивает несколько GSM триплетов (RAND, SRES, Kc), как правило, в

центре аутентификации оператора сети. Из триплетов создаются ключи безопасности, а значения RAND отправляются клиенту в атрибуте AT_RAND в следующем сообщении, которое включает также код аутентификации в атрибуте AT_MAC.

Приняв это сообщение партнер выполняет алгоритмы GSM, вычисляет копию MAC и проверяет ее на совпадение со значением, присланным сервером, а также вычисляет сеансовые ключи. В случае несовпадения кодов MAC партнер посылает серверу сообщение об ошибке. При совпадении кодов MAC посылка следующего сообщения серверу означает, что партнер успешно аутентифицировал сервер и что обмен EAP соответствует локальной политике партнера. В это сообщение включается код MAC, покрывающий содержимое пакета в конкатенации со значениями SRES партнера. Сервер EAP проверяет корректность MAC и посылает сообщение EAP-Success, подтверждающее успешное завершение аутентификации.

Механизм быстрой повторной аутентификации не задействует алгоритмы A3/A8 и инфраструктуру GSM, используя ключи, созданные во время полной аутентификации, что позволяет экономить вычислительные ресурсы.

3.1.6 EAP-AKA (EAP Method for 3rd Generation Authentication and Key Agreement)

Тип 23. RFC 4187 [19].

Метод использует параметры и алгоритмы SIM-карты для аутентификации и создания криптографических ключей. Данный метод основан на механизмах аутентификации и согласования ключей AKA, используемых в мобильных сетях третьего поколения (3G) UMTS и CDMA2000 [20],[21]. Механизмы AKA, основанные на алгоритмах с симметричными ключами, отличаются от используемых в сетях второго поколения GSM и основанном на них методе EAP-SIM (в частности, использованием более стойких ключей безопасности). AKA используется в SIM-картах типов USIM (UMTS Subscriber Identity Module) и (R)UIM ((Removable) User Identity Module).

Процесс аутентификации выглядит следующим образом (рис. 7). Сервер запрашивает идентификатор клиента, и затем, используя заранее распределенный секретных ключ и счетчик сообщений, создает криптографический набор, называемый вектором аутентификации (authentication vector): случайное значение RAND, хэш-значение AUTH (используется клиентом для аутентификации сервера), ожидаемое хэш-значение клиента XRES, два 128-битных ключа IK/CK. RAND и AUTH передаются клиенту. Клиент таким же образом создает свой криптографический набор, проверяет значение AUTH и отправляет серверу свое хэш-значение RES. Если взаимная аутентификация прошла успешно, ключи IK/CK используются для создания ключей безопасности данного сеанса.

C←S : EAP-Request/Identity

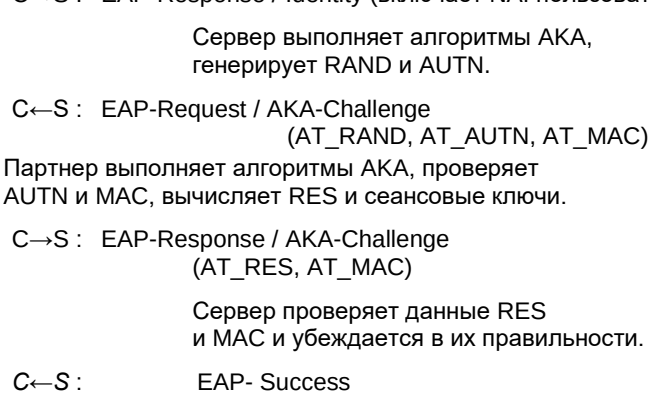


Рис. 7. Полная процедура аутентификации EAP-AKA
Fig. 7. EAP- AKA full authentication procedure.

Все параметры сеанса передаются в сообщениях в виде атрибутов. Спецификация определяет два десятка атрибутов, такие как запрос идентификатора клиента, значения RAND и AUTH, контрольная сумма сообщения, параметры алгоритма шифрования и сами зашифрованные данные, информационные сообщения, сообщения об ошибке и др.

Сообщение клиента EAP-Response/Identity обычно содержит IMSI (International Mobile Subscriber Identity) SIM-карты клиента для радиосетей или NAI (Network Access Identifier) для сервисов потоковых данных (IP multimedia service) [22].

Вектор аутентификации может быть получен сервером из центра аутентификации оператора сети.

Механизм быстрой повторной аутентификации не задействует алгоритмы AKA и инфраструктуру, используя ключи, созданные во время полной аутентификации, что позволяет экономить вычислительные ресурсы.

3.1.7 EAP-AKA' (Improved EAP-AKA)

Тип 50. RFC 5448 [23].

Данный метод является небольшой модификацией метода EAP-AKA. Он использует новый механизм создания криптографических ключей, связывая создаваемые внутри метода ключи и имя сети доступа (access network). Также заменена базовая хэш-функция: SHA-256 вместо SHA-1.

В спецификации RFC 5448 описан также способ противодействия атакам понижения уровня, которые может проводить злоумышленник (человек посередине) при согласовании оконечными точками методов EAP-AKA и EAP-AKA', чтобы вынудить их использовать менее стойкий метод, по сравнению с методом, которым они могут обе воспользоваться. В частности, для EAP-AKA определяется новый механизм, позволяющий оконечным

точкам выяснить возможности друг друга. Таким образом, предполагается, что метод EAP-AKA' всегда является предпочтительным.

3.2 Методы с общим секретным ключом

3.2.1 EAP-PSK (A Pre-Shared Key EAP Method)

Тип 47. RFC 4764 [24].

Метод EAP, основанный на заранее распределенных ключах безопасности. Обеспечивает взаимную аутентификацию партнеров и создание сеансовых криптографических ключей.

При его разработке ставились следующие цели:

- простота настройки и использования: метод использует единственный криптографический алгоритм AES-128 и фиксированный формат сообщений (отсутствуют поля формата Тип-Длина-Значение / Type-Length-Value);
- широкое применение: метод может использоваться в любых типах сетей, включая беспроводные.
- Защищенность: метод разработан для использования в незащищенных сетях.
- Расширяемость: в процессе аутентификации создается защищенный канал, который может использоваться для расширения функциональности метода.

На рис.8 представлен стандартный обмен сообщениями.

C←*S* : EAP-Request / EAP-PSK (Flags||RAND_S||ID_S)
C→*S* : EAP-Response / EAP-PSK (Flags||RAND_S||RAND_P||MAC_P||ID_P)
C←*S* : EAP-Request / EAP-PSK (Flags||RAND_S||MAC_S||PCHANNEL_S_0)
C→*S* : EAP-Response / EAP-PSK (Flags||RAND_S||PCHANNEL_P_1)

Рис. 8. Стандартная аутентификация EAP-PSK
Fig. 8. EAP-PSK Standard Authentication

Все сообщения EAP-PSK включают некоторый вид заголовка, состоящего из поля Flags и 16-байтного случайного числа RAND_S, которое посылается сервером и служит в качестве идентификатора сеанса. В первом сообщении сервер посылает RAND_S и указывает свой идентификатор ID_S. Во втором сообщении партнер посылает свое 16-байтное случайное число RAND_P, свой идентификатор ID_P и аутентифицирует себя путем демонстрации своей возможности вычислить правильный код аутентификации сообщения MAC_P, который зависит от ключа аутентификации AK, идентификаторов партнера и сервера, а также от RAND_S и RAND_P.

Третье сообщение служит для обеспечения аутентификации сервера партнером, в котором сервер демонстрирует свою возможность вычислить правильный код аутентификации сообщения MAC_S, зависящий от ключа AK, идентификатора сервера и RAND_P, а также устанавливает защищенный канал (поле PCHANNEL_S_0), подтверждая, что он вычислил сеансовые ключи, и индицируя защищенный результат аутентификации. Четвертое сообщение, посылаемое партнером серверу, завершает установку защищенного канала (поле PCHANNEL_P_1), подтверждает вычисление сеансовых ключей на стороне партнера и обеспечивает индикацию защищенного результата аутентификации.

Спецификация позволяет при необходимости продолжить обмен сообщениями в рамках созданного защищенного канала.

На рис. 9 представлена иерархия ключей, используемых данным методом.

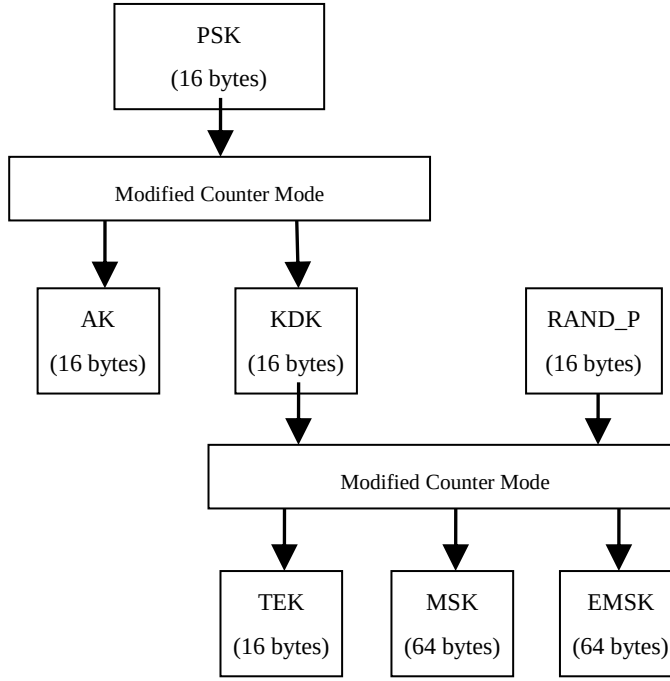


Рис. 9. Иерархия ключей EAP-PSK
Fig. 9. EAP-PSK Key Hierarchy

В EAP-PSK используется единственный криптографический примитив – блочный шифр AES-128. На его основе реализуется алгоритм вычисления кода аутентификации сообщений (MAC).

Ключ AK (Authentication Key) используется для взаимной аутентификации партнеров.

Ключ KDK (Key-Derivation Key) используется для создания сеансовых ключей (TEK, MSK, EMSK).

Ключ TEK (Transient EAP Key) используется для создания защищенного канала.

Ключи MSK, EMSK предназначены для использования за рамками данного метода EAP (например, другими приложениями) в соответствии с требованиями спецификации EAP (RFC 3748).

3.2.2 EAP-SAKE (EAP Method for Shared-secret Authentication and Key Establishment)

Тип 48. RFC 4763 [25].

Метод EAP, основанный на заранее распределенных ключах безопасности. В отличие от метода EAP-PSK, позволяет использовать различные криптографические алгоритмы. Основан на протоколе взаимной аутентификации, предложенном Михиром Белуаром и Филиппом Рогвеем [26-27].

На рис.10 представлен стандартный обмен сообщениями.

C←S : EAP-Request/ SAKE/Challenge
(AT_RAND_S, AT_SERVERID)
C→S : EAP-Response / SAKE/Challenge
(AT_RAND_P, AT_PEERID, AT_SPI_P, AT_MIC_P)

C←S : EAP-Request / SAKE/Confirm
(AT_SPI_S, AT_ENCR_DATA, AT_MIC_S)
C→S : EAP-Response / SAKE/ Confirm
(AT_MIC_P)

C←S : EAP- Success

Рис. 10. Процедура аутентификации EAP-SAKE (с согласованием набора шифров)
Fig. 10. EAP-SAKE Authentication Procedure (with ciphersuite negotiation).

AT_MIC_P, AT_MIC_S – контрольные суммы сообщений, обеспечивающие целостность сообщений и взаимную аутентификацию партнеров.

AT_SPI_P, AT_SPI_S – атрибуты, предназначенные для согласования криптографического набора.

AT_ENCR_DATA – зашифрованные данные.

Атрибуты AT_SPI_P, AT_SPI_S, AT_ENCR_DATA являются необязательными и включаются, если требуется обеспечить конфиденциальность данных.

Спецификация определяет несколько необязательных дополнительных атрибутов, расширяющих функциональность метода.

На рис.11 представлена иерархия ключей, используемых данным методом.

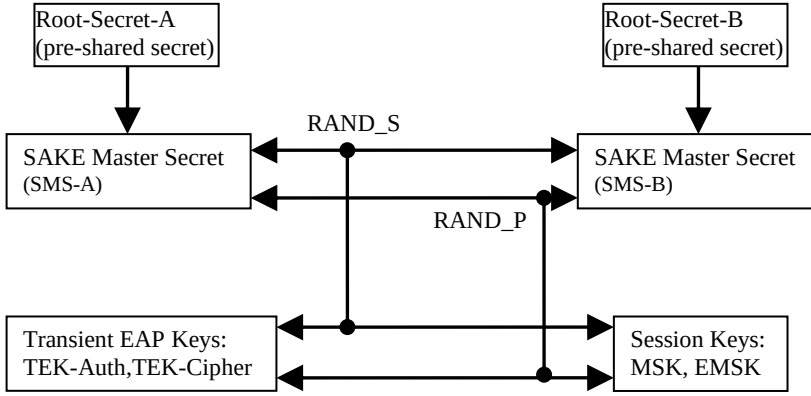


Рис. 11. Иерархия ключей метода EAP-SAKE
Fig. 11. EAP-SAKE Key Hierarchy

Общий ключ (pre-shared secret) делится на две части: Root-Secret-A и Root-Secret-B.

Ключи SMS-A, SMS-B используются для создания сеансовых ключей - TEK, MSK, EMSK.

Ключи TEK используются только внутри данного метода EAP для вычисления контрольных сумм и криптографической защиты.

Ключи MSK, EMSK предназначены для использования за рамками данного метода EAP (например, другими приложениями).

3.2.3 EAP-GPSK (EAP Generalized Pre-Shared Key Method)

Тип 51. RFC 5433 [28].

Простой метод EAP, основанный на заранее распределенных секретных ключах и обеспечивающий взаимную аутентификацию и создание криптографических ключей. К особенностям метода можно отнести простоту реализации, упрощенные криптографические вычисления на основе симметричных ключей, минимальное количество сообщений и поддержку нескольких криптографических алгоритмов (в спецификации метода определены два набора шифров). На рис. 12 представлен обмен сообщениями в случае успешной взаимной аутентификации участников.

C←S : EAP-Request/Identity
C→S : EAP-Response/Identity

C←S : EAP-Request
(ID_Server, RAND_Server, CSuite_List)
C→S : EAP-Response
SEC_SK(ID_Peer, ID_Server, RAND_Peer, RAND_Server,
CSuite_List, CSuite_Sel, [ENC_PK(PD_Payload_Block)])

C←S : EAP-Request
SEC_SK(RAND_Peer, RAND_Server, ID_Server, CSuite_Sel,
[ENC_PK(PD_Payload_Block)])
C→S : EAP-Response
SEC_SK([ENC_PK(PD_Payload_Block)])

C←S : EAP- Success

Рис. 12. Успешный обмен EAP-GPSK
Fig. 12. EAP-GPSK Successful Exchange

Во время выполнения EAP-GPSK партнер и сервер обмениваются неповторяющимися значениями (нонсами RAND_Peer и RAND_Server), которые используются вместе с заранее распределенным ключом для получения иерархии ключей EAP. Поэтому безопасность установления ключей (рис. 13) зависит от используемой функции вычисления ключей (key derivation function, KDF), и случайности указанных неповторяющихся значений.

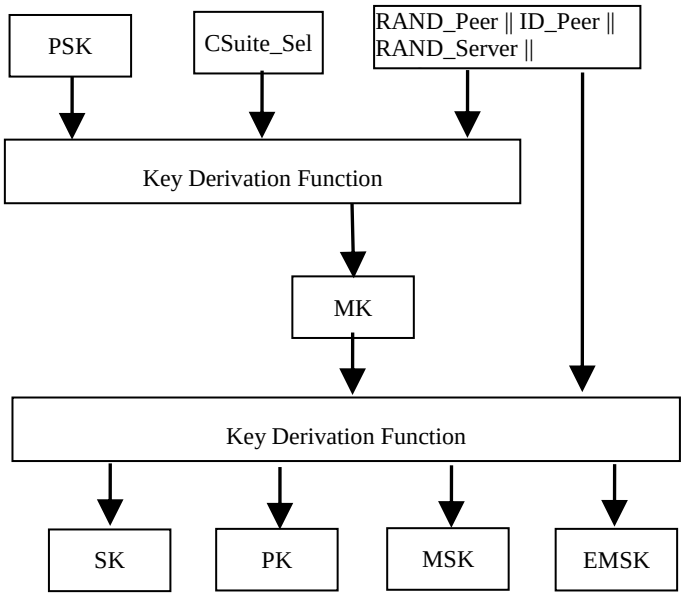


Рис. 13. Вычисление ключей в EAP-GPSK
Fig. 13. EAP-GPSK Key Derivation

Сервер предлагает список поддерживаемых алгоритмов через атрибут CSuite_List.

Атрибут SEC_SK обеспечивает защиту целостности сообщения.

Необязательный атрибут ENC_PK обеспечивает криптографическую защиту некоторых данных.

Метод использует несколько ключей безопасности:

Ключ PSK – заранее распределенный секретный ключ.

Ключ MK - используется для создания других ключей сеанса.

Ключ SK - используется для защиты целостности сообщений.

Ключ PK - используется для шифрования данных.

Ключи MSK, EMSK предназначены для использования за рамками данного метода EAP (см. RFC 3748 [1]).

3.2.4 EAP-pwd (EAP Authentication Using Only a Password)

Тип 52. RFC 5931 [29].

Использование пароля до сих пор является одним из самых распространенных способов аутентификации в Интернет с присущими ему многочисленными проблемами безопасности. Метод EAP-pwd использует в качестве входных данных пароль клиента. На его основе, используя криптографию дискретных логарифмов (discrete logarithm cryptography), создаются и согласовываются ключи безопасности с заданной криптографической стойкостью, которые в дальнейшем используются для защиты сеанса и взаимной аутентификации клиента и сервера [30]. Данный механизм аутентификации обеспечивает защиту от различных типов атак, включая атаки по словарю, поскольку исходный пароль не передается по сети.

Метод определяет три пары сообщений: Identity, Commit, Confirm. В начале обмена исходный пароль преобразуется в бинарную строку в соответствии с заданными в EAP-pwd правилами, что должно обеспечить идентичные исходные данные на стороне клиента и сервера.

C←S : EAP-pwd-ID/Request
C→S : EAP-pwd-ID/Response

C←S : EAP-pwd-Commit/Request
C→S : EAP-pwd-Commit/Response

C←S : EAP-pwd-Confirm/Request
C→S : EAP-pwd-Confirm/Response

C←S : EAP- Success

Рис. 14. Успешный обмен EAP-pwd
Fig. 14. A Successful EAP-pwd Exchange

Успешный обмен EAP-pwd показан на рис. 14. Сообщения Identity используются для обмена идентификаторами и согласования криптографического набора. Также сервер может использовать эти сообщения, чтобы сообщить клиенту о необходимости предварительных преобразований пароля. В сообщениях Commit передаются данные для создания общего криптографического ключа. Третья пара сообщений Confirm

используется для верификации созданных ключей и, как следствие, взаимной аутентификации сторон. Сообщения содержат проверочные блоки данных, защищенные с помощью согласованных криптографических алгоритмов и созданных ключей безопасности.

3.2.5 EAP-EKE (EAP Authentication Method Based on the Encrypted Key Exchange Protocol)

Тип 53. RFC 6124 [31].

Еще один метод, использующий в качестве входных данных обычный пароль клиента. Во многом повторяет EAP-pwd, из отличий: другой способ генерации общего ключа безопасности (используется механизм защищенного согласования ключей (Encrypted Key Exchange (EKE) на основе обмена Диффи-Хеллмана) и отсутствие требований предварительного преобразования исходного пароля к общему виду [32].

На основе исходного пароля создаются и согласовываются ключи безопасности с заданной криптографической стойкостью, которые в дальнейшем используются для защиты сеанса и взаимной аутентификации клиента и сервера. Также стоит отметить, что исходный пароль не передается по сети, а созданный из пароля общий ключ сеанса не используется напрямую – из него вычисляются другие ключи, которые и используются для защиты сообщений

Метод определяет три пары сообщений: Identity, Commit, Confirm. Сообщения Identity используются для обмена идентификаторами и согласования криптографического набора. В сообщениях Commit передаются данные для создания общего криптографического ключа. Третья пара сообщений Confirm используется для верификации созданных ключей и, как следствие, взаимной аутентификации сторон. Сообщения содержат проверочные блоки данных, защищенные с помощью согласованных криптографических алгоритмов и созданных ключей безопасности. Успешный обмен EAP-EKE показан на рис. 14.

```
C←S :  EAP-pwd-ID/Request
                ID_S, CryptoProposals
C→S :  EAP-pwd-ID/Response
                ID_P, CryptoSelection

C←S :  EAP-pwd-Commit/Request
                Encr(Password, y_s)
C→S :  EAP-pwd-Commit/Response
                Encr(Password, y_p),  Prot(Ke, Ki, Nonce_P)

C←S :  EAP-pwd-Confirm/Request
                Prot(Ke, Ki, Nonce_S | Nonce_P), Auth_S
C→S :  EAP-pwd-Confirm/Response
                Prot(Ke, Ki, Nonce_S), Auth_P

C←S :  EAP- Success
```

Рис. 15. Успешный обмен EAP-EKE
Fig. 15. A Successful EAP-EKE Exchange

3.3 Методы, использующие механизмы создания защищенного канала для целей аутентификации

3.3.1 EAP-TLS (EAP TLS Authentication Protocol)

Тип 13. RFC 5216 [33].

Метод EAP-TLS использует фазу рукопожатия протокола TLSv1.1 (TLS handshake) для взаимной аутентификации клиента и сервера на основе сертификатов, а также для согласования дополнительных ключей безопасности [34]. Метод поддерживает стандартный механизм возобновления сеанса TLS. На рис.16 представлен успешный обмен EAP-TLS.

C←S : EAP-Request/Identity
C→S : EAP-Response/Identity (MyID)

C←S : EAP-Request/EAP-TLS (TSL start)
C→S : EAP-Response/EAP-TLS (ClientHello)

C←S : EAP-Request/EAP-TLS
(ServerHello, Certificate, [ServerKeyExchange],
CertificateRequest, ServerHelloDone)
C→S : EAP-Response/EAP-TLS
(Certificate, ClientKeyExchange, CertificateVerify,
ChangeCipherSpec, Finished)

C←S : EAP-Request/EAP-TLS
(ChangeCipherSpec, Finished)
C→S : EAP-Response/EAP-TLS

C←S : EAP- Success

Рис. 16. Успешный обмен EAP-TLS
Fig. 16. A Successful EAP-TLS Exchange

Получив идентификатор партнера, сервер EAP должен ответить пакетом EAP-TLS с установленным флагом S (Start), начинающим обмен рукопожатия TLS. В своем сообщении ClientHello партнер посылает случайное число ClientHello.random, максимальный номер версии TLS, которую он поддерживает, предлагаемый набор криптографических алгоритмов и метод сжатия, а также идентификатор сеанса, Session ID, (в случае попытки продолжения сеанса TLS).

Сервер отвечает пакетом, инкапсулирующим одну или несколько записей TLS. Сообщение рукопожатия ServerHello, содержит выбранные из предложенных клиентом версию TLS, идентификатор сеанса (Session ID), криптографические алгоритмы и метод сжатия, а также свое случайное число ServerHello.random. Выбранная версия TLS должна быть максимальной из поддерживаемых. Если сервер находит идентификатор сеанса в своем кэше, то

это означает возобновление ранее установленного сеанса TLS. В противном случае сервер выбирает значение Session ID для установления нового сеанса TLS.

Сообщение Certificate содержит цепочку сертификатов либо для открытого ключа обмена ключами (RSA или ДиффиХеллмана), либо открытого ключа подписи (RSA или DSS). В последнем случае в этот обмен должно быть включено сообщение рукопожатия ServerKeyExchange. Если сервер аутентифицирован, он может запросить сертификат клиента (CertificateRequest). Кроме того, сервер посылает сообщение ServerHelloDone, указывающее, что фаза приветствия завершена. После чего ждет ответа клиента.

Если был соответствующий запрос сервера, клиент посылает свой сертификат. Затем следует сообщение ClientKeyExchange, содержимое которого зависит от выбранного в сообщениях ClientHello и ServerHello алгоритма с открытым ключом. Если сертификат клиента используется для цифровой подписи, то посылается подписанное сообщение CertificateVerify для явной проверки подлинности сертификата. Клиент посылает сообщение об изменении состояния ChangeCipherSpec и заменяет текущее состояние ожидаемым. В завершение клиент посылает сообщение Finished, защищенное только что согласованными алгоритмами и содержащее код аутентификации (MAC), вычисленный над всеми предыдущими сообщениями обмена.

Сервер расшифровывает полученное сообщение Finished, проверяет правильность MAC (в случае ошибки соединение разрывается). Сервер отвечает своим сообщением изменения состояния ChangeCipherSpec, заменяет текущее состояние ожидаемым и посылает заключительное сообщение Finished, применяя согласованные алгоритмы и ключи.

Клиент также расшифровывает и проверяет полученное сообщение, о чем сообщает серверу в предпоследнем сообщении.

Следует отметить, что EAP-TLS в течение достаточно длительного времени был единственным понятным и надежным механизмом, обеспечивающим взаимную аутентификацию между клиентом и сервером EAP. Однако широкое его распространение оказалось ограниченным в связи со сложностью обеспечения клиентов сертификатами. Создание и поддержка инфраструктуры открытых ключей оказалось для многих организаций исключительно сложной задачей.

3.3.2 EAP-IKEv2

Тип 49. RFC 5106 [35].

Данный метод основан на протоколе обмена ключами IKEv2 (Internet Key Exchange Protocol version 2) и обеспечивает взаимную аутентификацию партнеров и согласование ключей безопасности на основе различных аутентификационных данных: паролей, заранее распределенных ключей или сертификатов. Протокол IKEv2 обеспечивает согласование

криптографического набора, который затем используется для защиты сетевого обмена: шифрования данных, защиты целостности сообщений, генерации различных криптографических ключей [10]. Формат сообщений метода EAP аналогичен сообщениям протокола IKEv2. Также поддерживается быстрое возобновление сеанса. Метод EAP-IKEv2 не поддерживает инкапсуляцию других методов EAP.

R←I : EAP-Request/Identity
R→I : EAP-Response/Identity (Id)

R←I : EAP-Request (HDR, SAi, KEi, Ni)
R→I : EAP-Response (HDR, SAr, KEr, Nr, [CERTREQ], [SK{IDr}])

R←I : EAP-Request (HDR, SK{IDi, [CERT], [CERTREQ], [NFID], AUTH})
R→I : EAP-Response (HDR, SK{IDr, [CERT], AUTH})

R←I : EAP- Success

Рис. 17. Полный успешный обмен по протоколу EAP-IKEv2
Fig. 17. EAP-IKEv2 Full Successful Protocol Run.

В EAP-IKE2 предполагается, что сервер EAP исполняет роль инициатора (I), а партнер - роль ответчика (R). Семантика и формат сообщений EAP-IKE2 подобны, хотя и не совпадают полностью с семантикой и форматом сообщений IKE2. Полное выполнение протокола EAP-IKE2 включает пару сообщений запрос/ответ для выяснения идентификатора партнера и две пары сообщений запрос/ответ, за которыми следует либо сообщение EAP-Success, либо сообщение EAP-Failure.

Первые два сообщения являются стандартными сообщениями запроса идентификатора партнера и соответствующего ответа. С третьего сообщения начинается реальный аутентификационный обмен. Оно содержит индекс параметров безопасности инициатора (Security Parameter Index, SPI) в заголовке EAP-IKE2 (HDR), набор криптографических алгоритмов, которые сервер предполагает использовать для защиты трафика EAP-IKE2 (шифрования и защиты целостности) и вычисления сеансового ключа. Этот набор алгоритмов кодируется в блоке данных (Security Association, SAi). В блоках данных KEi и Ni пересылаются открытое значение Диффи-Хеллмана и одноразовый номер (нонс) инициатора.

Получив это сообщение, партнер генерирует ненулевое значение SPI ответчика, выбирает криптографический набор из множества, предложенного инициатором, и сообщает о своём выборе четвертым сообщением в блоке SAr, завершает обмен Диффи-Хеллмана, пересылая свое открытое значение в блоке KEr, и посылает свой одноразовый номер Nr.

После получения инициатором этого сообщения каждый из участников, используя результат обмена Диффи-Хеллмана, может найти порождающий секретный ключ SKEYSEED, на основе которого вычисляются все

необходимые ключи в соответствии со спецификацией IKEv2 [10]. Для всех последующих сообщений будет обеспечиваться защита целостности и шифрование всего содержимого, кроме заголовка. Для этого используются ключи SK_a (аутентификация) и SK_e (шифрование) соответственно, получаемые из SKEYSEED. Для каждого направления вычисляются свои ключи. Кроме того, вычисляется величина SK_d, на основе которой впоследствии будет генерироваться ключевой материал для CHILD_SA. Именно ключи SK_e и SK_a используются для шифрования и обеспечения целостности данных в обозначении SK{...}.

В зависимости от используемых для аутентификации удостоверяющих данных локальная политика партнера может потребовать включения в четвертое сообщение необязательного блока данных CERTREQ, который запрашивает сертификат открытого ключа сервера, а в случае использования для аутентификации симметричной криптографии как на стороне сервера, так и на стороне партнера последний должен включить в это сообщение необязательный блок данных SK{IDr}, который содержит его идентификатор EAP-IKE2, зашифрованный и защищенный целостностью в блоке данных Encrypted. Этот идентификатор партнера необходим серверу для выбора правильного симметричного ключа или пароля для создания блока данных AUTH в пятом сообщении.

Пятое сообщение, посылаемое сервером, содержит заголовок EAP-IKE2, за которым следует единственный блок данных Encrypted. Инициатор должен в этот блок данных Encrypted встроить по крайней мере два блока данных: блок данных Identification, содержащий EAP-IKE2 идентификатор инициатора, и блок данных Authentication (AUTH), который доказывает знание секрета, связанного с этим идентификатором, и обеспечивает целостность своего предыдущего сообщения. Инициатор также, возможно, посылает свой сертификат в блоке CERT и запрашивает сертификат партнера, указывая в блоке CERTREQ список сертификационных центров, которым инициатор доверяет. Если в сообщение были включены сертификаты, то первый из них должен содержать открытый ключ для проверки AUTH. Кроме того, в этот блок данных Encrypted может быть встроен блок данных Next Fast-Reconnect Identifier (NFRI), который спецификацией предполагалось использовать для быстрого повторного соединения.

После получения пятого сообщения ответчик (партнер EAP) аутентифицирует инициатора (сервер EAP). Для этого он осуществляет необходимый контроль расшифровывая блок данных Encrypted, проверяя его целостность и выясняя, содержит ли блок данных AUTH ожидаемое значение. Если все проверки прошли успешно, то ответчик отправляет шестое сообщение, которое содержит заголовок EAP-IKE2 и блок данных Encrypted, в который в свою очередь встраиваются блоки данных, показанные на рис.17.

После получения шестого сообщения инициатор (сервер EAP) аутентифицирует ответчика (партнера EAP). Как и в предыдущем случае

проверки зависят от выбранного способа аутентификации, локальной политики и должны включать расшифровывание и проверку блока данных Encrypted, а также проверку того, что блок данных AUTH содержит ожидаемое значение. Если в четвертое сообщение был включен блок данных SK{IDr}, то сервер EAP должен также гарантировать, что блок данных IDr в шестом сообщении совпадает с соответствующим блоком данных из четвертого сообщения.

Если аутентификация прошла успешно, то ответчику отправляется седьмое сообщение EAP-Success. После успешного выполнения протокола EAP-IKE2 сервер EAP и партнер EAP генерируют ключи MSK и EMSK в соответствии с разделом 5 спецификации [35].

3.4 Туннельные методы

3.4.1 EAP-TTLSv0 (EAP Tunneled TLS Authenticated Protocol Version 0)

Тип 21. RFC 5281 [36].

Метод EAP-TLS использует протокол TLS для взаимной аутентификации клиента и сервера. Метод EAP-TTLS расширяет функциональность последнего, используя защищенный канал TLS, созданный в результате обмена рукопожатия (TSL handshake), для безопасной передачи дополнительных данных между клиентом и сервером.

Информационный обмен данного метода состоит из двух фаз: фазы рукопожатия (handshake phase) и фазы передачи данных (data phase). Первая фаза использует стандартный механизм протокола TLS для аутентификации сервера клиентом (и, при желании сервера, для взаимной аутентификации) на основе сертификатов. В результате стороны согласовывают криптографический набор и ключи безопасности для создания защищенного соединения.

Во второй фазе созданный криптографический канал используется для передачи произвольных данных, а также, при необходимости, для аутентификации клиента сервером (или взаимной аутентификации) с помощью произвольного механизма аутентификации (может использоваться как какой-либо метод EAP, так и независимый протокол: PAP, CHAP, MS-CHAP-V2). Также во второй фазе могут последовательно применяться несколько механизмов аутентификации. Кроме того, метод EAP-TTLS позволяет создавать криптографические ключи для использования за пределами данного метода (ключи MSK, EMSK). В сообщениях EAP-TTLS для передачи данных используются атрибуты AVP (attribute-value pairs) совместимые с протоколами RADIUS и DIAMETR [37, 38].

Метод определяет сетевые объекты (рис. 18), взаимодействующие между собой.

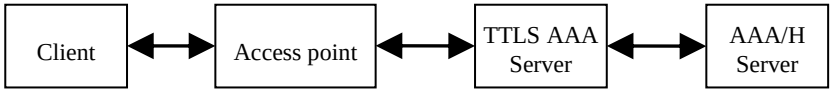


Рис. 18. Модель взаимодействия объектов метода EAP-TTLSv0
Fig. 18. EAP-TTLSv0 Architectural Model

Точка доступа (access point) и серверы TTLS и AAA/H могут располагаться как на одном сетевом узле, так и на нескольких. Как правило, между клиентом и точкой доступа отсутствуют предустановленные средства безопасности передачи данных. Ключи MSK и EMSK, создаваемые внутри метода EAP и предназначенные для использования за пределами метода (см. RFC 3748 [1]), могут быть переданы точке доступа сервером TTLS и могут в дальнейшем использоваться для криптографической защиты соединения между точкой доступа и клиентом.

На рис. 19 представлена многоуровневая модель сообщений метода EAP-TTLS, где каждый уровень сообщения инкапсулирует вышележащий уровень.

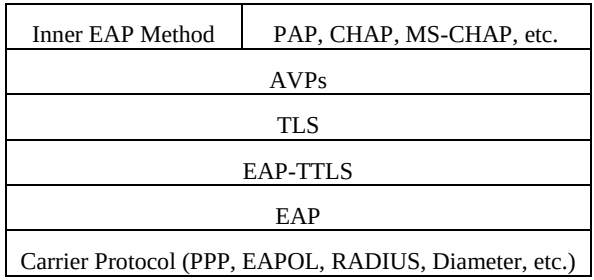


Рис. 19. Модель уровневой организации протокола
Fig. 19. Protocol Layering Model.

EAP-TTLS поддерживает стандартный механизм возобновления сеанса TLS (TLS session resumption). Однако данный механизм не должен использоваться, если клиент не смог пройти успешную аутентификацию во второй фазе метода. Несоблюдение данного требования сервером TTLS может привести к серьезному нарушению безопасности.

Схема сетевого обмена при успешной аутентификации (EAP/MD5-Challenge) показана на рис. 20.

C←S : EAP-Request/Identity
C→S : EAP-Response/Identity (Id)

C←S : EAP-Request/TTLS-start
C→S : EAP-Response/TTLS (ClientHello)

C←S : EAP-Request/TTLS (ServerHello, Certificate,
ServerKeyExchange, ServerHelloDone)

C→S : EAP-Response/TTLS (ClientKeyExchange, ChangeCipherSpec, Finished)

C←S : EAP-Request/TTLS (ChangeCipherSpec, Finished)
C→S : EAP-Response/TTLS (EAP-Response/Identity)

C←S : EAP-Request/TTLS (EAP-Request/MD5-Challenge)
C→S : EAP-Response/TTLS (EAP-Response/ MD5-Challenge)

C←S : EAP- Success

Рис. 20. Успешная аутентификация посредством туннелируемого метода EAP/MD5-Challenge
Fig. 20. Successful Authentication via Tunneled EAP/MD5-Challenge

3.4.2 EAP-FAST (Flexible Authentication via Secure Tunneling EAP Method)

Тип 43. RFC 4851 [39].

Метод EAP-FAST использует протокол TLS для создания защищенного канала, внутри которого затем используются другие методы аутентификации. Для передачи данных внутри TLS-туннеля используются TLV-объекты (Type-Length-Value). Чтобы уменьшить использование вычислительных ресурсов, EAP-FAST поддерживает механизм быстрого возобновления сеанса TLS; в этом случае сервер не хранит параметры сеанса, а передает их клиенту в виде специальной структуры (ticket) [40].

Метод определяет сетевые объекты (рис. 21), взаимодействующие между собой

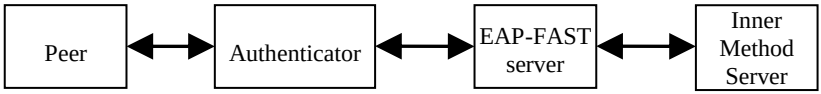


Рис. 21. Модель взаимодействия объектов метода EAP-FAST
Fig. 21. EAP-FAST Architectural Model.

При этом, объекты аутентификатор, сервер EAP-FAST и сервер внутреннего метода могут располагаться как на одном сетевом узле, так и на нескольких.

На рис. 22. представлена многоуровневая модель сообщений метода EAP-FAST, где каждый уровень сообщения инкапсулирует вышележащий уровень.

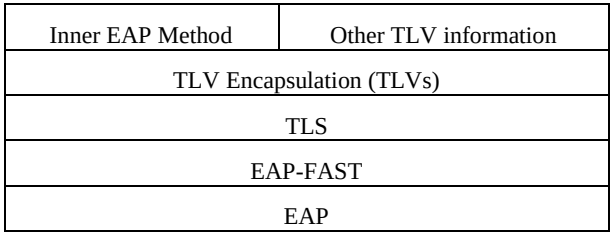


Рис. 22. Модель уровневой организации протокола EAP-FAST
Fig. 22. EAP-FAST Protocol Layering Model.

Вариант сетевого обмена с успешной аутентификацией при возобновлении сеанса показан на рис. 23.

C←S : EAP-Request/Identity
C→S : EAP-Response/Identity (MyID)

C←S : EAP-Request/EAP-FAST (S=1, A-ID)
C→S : EAP-Response/ EAP-FAST
(ClientHello with PAC-Opaque in SessionTicket extension)

C←S : EAP-Request/ EAP-FAST
(ServerHello, ChangeCipherSpec, Finished)
C→S : EAP-Response/ EAP-FAST
(ChangeCipherSpec, Finished)

Канал TLS установлен (последующие сообщения, посылаемые по каналу TLS, инкапсулируются в EAP-FAST):

C←S : EAP Payload TLV (EAP-Request/EAP-GTC(Challenge))
C→S : EAP Payload TLV (EAP-Response/EAP-GTC
(ответ содержит имя пользователя и пароль))

Необязательные дополнительные обмены (режим нового пин-кода, изменение пароля и т.п.):

C←S : Intermediate-Result TLV (Success)
Crypto-Binding TLV (Request)
C→S : Intermediate-Result TLV (Success)
Crypto-Binding TLV (Response)

C←S : Result TLV (Success)
[Optional PAC TLV]
C→S : Result TLV (Success)
[PAC TLV Acknowledgment]

Канал TLS уничтожается (сообщения посылаются в виде открытого текста)

C←S : EAP- Success

Рис. 23. Успешная аутентификация EAP-FAST
Fig. 23. EAP-FAST Successful Authentication

В тех случаях, когда внутри TLS-туннеля последовательно применяются несколько методов аутентификации, каждый такой метод заканчивается сообщением Intermediate-Result TLV/Crypto-Binding TLV, обеспечивающим индикацию результата и криптографическое связывание с последующим методом.

3.4.3 TEAP (Tunnel Extensible Authentication Protocol Version 1)

Тип 55. RFC 7170 [41].

Метод TEAP использует протокол TLS для создания защищенного канала со взаимной аутентификацией партнеров, внутри которого затем используются другие методы EAP. Для передачи данных внутри TLS-туннеля используются TLV-объекты (Type-Length-Value). TEAP может использоваться с любым транспортным протоколом, поддерживающим аутентификацию EAP.

На момент выхода данного стандарта были зарегистрированы несколько методов EAP, использующих TLS-туннель для защиты других методов аутентификации: PEAP (Protected EAP), EAP-TTLS (EAP tunneled TLS), EAP-FAST (EAP Flexible Authentication via Secure Tunneling) [42],[36],[39]. Однако, ни один из них не имеет статуса «Интернет Стандарта». Кроме того, в документе RFC 6678 сформулированы требования к методам EAP, использующим защищенный канал для последующей аутентификации (tunnel-based EAP method) [43].

Метод TEAP получил статус стандарта (PROPOSED STANDARD). В качестве его основы был выбран метод EAP-FAST, который был переработан в соответствии с требованиями RFC 6678, улучшена гибкость метода (особенно в отношении согласования криптографических алгоритмов), обновлены некоторые устаревшие спецификации, на которые ссылается EAP-FAST (в частности версия протокола TLS изменена на последнюю v1.2). Поэтому в плане архитектуры и сетевого обмена TEAP во многом повторяет EAP-FAST.

Ниже перечислены основные отличия от EAP-FAST:

- TEAP должен поддерживать последнюю версию протокола TLSv1.2 [12];
- криптографические ключи создаются в соответствии с требованиями RFC 5705, а соответствующие криптографические функции согласовываются через обмен TLS [44];
- TEAP полностью совместим с требованиями RFC 5077 (расширение TLS для быстрого возобновления сеанса с хранением состояния сеанса на стороне клиента) [45];
- добавлены дополнительные атрибуты для передачи метаданных и связывания каналов (channel binding);
- добавлена поддержка простого пароля в качестве одного из внутренних методов аутентификации.

4. Заключение

Протокол EAP в совокупности с его методами представляет собой мощное средство для обеспечения аутентификации и контроля доступа к сетям и ресурсам вычислительных систем. В данной статье рассмотрены основные особенности самого протокола и методов EAP. Показано разнообразие механизмов, используемых для реализации сервиса аутентификации. В ней не

рассматриваются вопросы обеспечения безопасности EAP и методов EAP, которые с нашей точки зрения заслуживают отдельной публикации.

Список литературы

[1]. IETF RFC 3748. B. Aboba, et al. Extensible Authentication Protocol (EAP). June 2004. Доступно по ссылке: https://tools.ietf.org/html/rfc3748

[2]. IETF RFC 1661. W. Simpson. The Point-to-Point Protocol (PPP). July 1994. Доступно по ссылке: https://tools.ietf.org/html/rfc1661

[3]. IEEE Standard 802, Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks: Overview and Architecture", 1990.

[4]. IETF RFC 791, Internet Protocol, September 1981. Доступно по ссылке: https://tools.ietf.org/html/rfc791

[5]. IEEE Standard 802.1X-2010 - IEEE Standard for Local and metropolitan area networks--Port-Based Network Access Control, 2010.

[6]. IETF RFC 3579. B. Aboba and P. Calhoun. RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP). September 2003. Доступно по ссылке: https://tools.ietf.org/html/rfc3579

[7]. IETF RFC 4072. Eronen, et al. Diameter Extensible Authentication Protocol (EAP) Application. August 2005. Доступно по ссылке: https://tools.ietf.org/html/rfc4072

[8]. IEEE Standard 802.11-2007, Institute of Electrical and Electronics Engineers, "Standard for Local and metropolitan area networks - specific requirements – part 11: Wireless LAN Medium Access Control and Physical Layer specifications", 2007.

[9]. IEEE Standard 802.16e-2005, Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands. December 2005.

[10]. IETF RFC 4306. Kaufman, C., Ed. Internet Key Exchange (IKEv2) Protocol. December 2005. Доступно по ссылке: https://tools.ietf.org/html/rfc4306

[11]. Extensible Authentication Protocol (EAP) Registry, Доступно по ссылке: http://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml, 25.04.2018

[12]. IETF RFC 5246. Dierks, T. and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. August 2008. Доступно по ссылке: https://tools.ietf.org/html/rfc5246

[13]. IETF RFC 1994. W. Simpson. PPP Challenge Handshake Authentication Protoco. August 1996. Доступно по ссылке: https://tools.ietf.org/html/rfc1994

[14]. IETF RFC 2289. N. Haller, et al. A One-Time Password System. February 1998. Доступно по ссылке: https://tools.ietf.org/html/rfc2289

[15]. IETF RFC 4793. M. Nystroem. The EAP Protected One-Time Password Protocol (EAP-POTP). February 2007. Доступно по ссылке: https://tools.ietf.org/html/rfc4793

[16]. IETF RFC 4186. Haverinen & Salowey. Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM). January 2006. Доступно по ссылке: https://tools.ietf.org/html/rfc4186

[17]. European Telecommunications Standards Institute, "GSM Technical Specification GSM 03.20 (ETS 300 534): "Digital cellular telecommunication system (Phase 2); Security related network functions"", August 1997.

[18]. European Telecommunications Standards Institute, "GSM Technical Specification GSM 03.03 (ETS 300 523): "Digital cellular telecommunication system (Phase 2); Numbering, addressing and identification"", April 1997.

[19]. IETF RFC 4187. Arkko & Haverinen. Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA). January 2006. Доступно по ссылке: https://tools.ietf.org/html/rfc4187
[20]. 3rd Generation Partnership Project, "3GPP Technical Specification 3GPP TS 33.102 V5.1.0: "Technical Specification Group Services and System Aspects; 3G Security; Security Architecture (Release 5)"", December 2002.
[21]. 3rd Generation Partnership Project 2, "3GPP2 Enhanced Cryptographic Algorithms", September 2003.
[22]. 3rd Generation Partnership Project, "3GPP Technical Specification 3GPP TS 23.003 V6.8.0: "3rd Generation Parnership Project; Technical Specification Group Core Network; Numbering, addressing and identification (Release 6)"", December 2005.
[23]. IETF RFC 5448. Arkko, et al. Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA'). May 2009. Доступно по ссылке: https://tools.ietf.org/html/rfc5448
[24]. IETF RFC 4764. F. Bersani and H. Tschofenig. The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method. January 2007. Доступно по ссылке: https://tools.ietf.org/html/rfc4764
[25]. IETF RFC 4763. M. Vanderveen and H. Soliman. Extensible Authentication Protocol Method for Shared-secret Authentication and Key Establishment (EAP-SAKE). November 2006. Доступно по ссылке: https://tools.ietf.org/html/rfc4763
[26]. M. Bellare and P. Rogaway. Entity Authentication and key distribution. In Advances in Cryptology - Crypto 93 Proceedings, pages 232-249, 1993.
[27]. M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In Proc. 27th Annual Symposium on the Theory of Computing, pages 57-66, 1995.
[28]. IETF RFC 5433. Clancy & Tschofenig. Extensible Authentication Protocol - Generalized Pre-Shared Key (EAP-GPSK) Method. February 2009. Доступно по ссылке: https://tools.ietf.org/html/rfc5433
[29]. IETF RFC 5931, Harkins & Zorn. Extensible Authentication Protocol (EAP) Authentication Using Only a Password. August 2010. Доступно по ссылке: https://tools.ietf.org/html/rfc5931
[30]. Barker, E., Johnson, D., and M. Smid. Recommendations for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication 800-56A, March 2007.
[31]. IETF RFC 6124. Sheffer, et al. An EAP Authentication Method Based on the Encrypted Key Exchange (EKE) Protocol. February 2011. Доступно по ссылке: https://tools.ietf.org/html/rfc6124
[32]. Bellovin, S. and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. Proc. IEEE Symp. on Research in Security and Privacy , May 1992.
[33]. IETF RFC 5216. Simon, et al. The EAP-TLS Authentication Protocol. March 2008. Доступно по ссылке: https://tools.ietf.org/html/rfc5216
[34]. IETF RFC 4346. Dierks, T. and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. April 2006. Доступно по ссылке: https://tools.ietf.org/html/rfc4346
[35]. IETF RFC 5106. Tschofenig, et al. The Extensible Authentication Protocol-Internet Key Exchange Protocol version 2 (EAP-IKEv2) Method. February 2008. Доступно по ссылке: https://tools.ietf.org/html/rfc5106

[36]. IETF RFC 5281. Funk & Blake-Wilson. Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0). August 2008. Доступно по ссылке: https://tools.ietf.org/html/rfc5281
[37]. IETF RFC 2865. Rigney, C., Willens, S., Rubens, A., and W. Simpson. Remote Authentication Dial In User Service (RADIUS). June 2000. Доступно по ссылке: https://tools.ietf.org/html/rfc2865
[38]. IETF RFC 3588. Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko. Diameter Base Protocol. September 2003. Доступно по ссылке: https://tools.ietf.org/html/rfc3588
[39]. IETF RFC 4851. Cam-Winget, et al. The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST). May 2007. Доступно по ссылке: https://tools.ietf.org/html/rfc4851
[40]. IETF RFC 4507. Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. May 2006. Доступно по ссылке: https://tools.ietf.org/html/rfc4507
[41]. IETF RFC 7170. Zhou, et al. Tunnel Extensible Authentication Protocol (TEAP) Version 1. May 2014. Доступно по ссылке: https://tools.ietf.org/html/rfc7170
[42]. Microsoft Corporation. [MS-PEAP]: Protected Extensible Authentication Protocol (PEAP). December 2017. Доступно по ссылке: https://msdn.microsoft.com/en-us/library/cc238354.aspx, 25.04.2018
[43]. IETF RFC 6678. Hoeper, K., Hanna, S., Zhou, H., and J. Salowey. Requirements for a Tunnel-Based Extensible Authentication Protocol (EAP) Method. July 2012. Доступно по ссылке: https://tools.ietf.org/html/rfc6678
[44]. IETF RFC 5705. Rescorla, E. Keying Material Exporters for Transport Layer Security (TLS). March 2010. Доступно по ссылке: https://tools.ietf.org/html/rfc5705
[45]. IETF RFC 5077. Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. January 2008. Доступно по ссылке: https://tools.ietf.org/html/rfc5077

The review of Extensible Authentication Protocol and its methods

[1] *A.V. Nikeshin <alexn@ispras.ru>*
[1,2] *V.Z. Shnitman <vzs@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Moscow Institute of Physics and Technology (State University), 9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia*

Abstract. Authentication is associated with a scenario, in which some party (the applicant) presented the identity of the principal and states that this is the principal. Authentication allows some other party (verifier) to make sure that this statement is legitimate. Authentication is widely used in access control systems to networks and resources of computing systems. In this context, of considerable interest is the Extensible Authentication Protocol (EAP), specified by the IETF in RFC 3748, which provides an effective mechanism for embedding various authentication methods into it, as well as the proper methods of EAP

authentication, some of which were standardized in specifications IETF. This article is a review of Extensible Authentication Protocol (EAP) and its methods, specified by IETF. EAP provide an effective flexible authentication mechanism that can be easily expanded with new authentication methods. The variety of mechanisms used to implement the authentication service are shown. The work was performed under support of the Russian Foundation for Basic Research, research grant № 16-07-00603 "The verification of security functionality of the EAP authentication protocol and evaluation of the robustness of its implementations against attacks".

Keywords: security; authentication; access control; EAP; EAP methods.

DOI: 10.15514/ISPRAS-2018-30(2)-7

For citation: Nikeshin A.V., Shnitman V.Z. The review of Extensible Authentication Protocol and its methods. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue. 2, 2018, pp. 113-148 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-7

References

[1]. IETF RFC 3748. B. Aboba, et al. Extensible Authentication Protocol (EAP). June 2004. Доступно по ссылке: https://tools.ietf.org/html/rfc3748
[2]. IETF RFC 1661. W. Simpson. The Point-to-Point Protocol (PPP). July 1994. Available at https://tools.ietf.org/html/rfc1661
[3]. IEEE Standard 802, Institute of Electrical and Electronics Engineers, "Local and Metropolitan Area Networks: Overview and Architecture", 1990.
[4]. IETF RFC 791, Internet Protocol, September 1981. Available at https://tools.ietf.org/html/rfc791
[5]. IEEE Standard 802.1X-2010 - IEEE Standard for Local and metropolitan area networks--Port-Based Network Access Control, 2010.
[6]. IETF RFC 3579. B. Aboba and P. Calhoun. RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP). September 2003. Available at https://tools.ietf.org/html/rfc3579
[7]. IETF RFC 4072. Eronen, et al. Diameter Extensible Authentication Protocol (EAP) Application. August 2005. Available at https://tools.ietf.org/html/rfc4072
[8]. IEEE Standard 802.11-2007, Institute of Electrical and Electronics Engineers, "Standard for Local and metropolitan area networks - specific requirements – part 11: Wireless LAN Medium Access Control and Physical Layer specifications", 2007.
[9]. IEEE Standard 802.16e-2005, Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands. December 2005.
[10]. IETF RFC 4306. Kaufman, C., Ed. Internet Key Exchange (IKEv2) Protocol. December 2005. Available at https://tools.ietf.org/html/rfc4306
[11]. Extensible Authentication Protocol (EAP) Registry, Available at http://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml, 25.04.2018
[12]. IETF RFC 5246. Dierks, T. and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. August 2008. Available at https://tools.ietf.org/html/rfc5246
[13]. IETF RFC 1994. W. Simpson. PPP Challenge Handshake Authentication Protoco. August 1996. Available at https://tools.ietf.org/html/rfc1994
[14]. IETF RFC 2289. N. Haller, et al. A One-Time Password System. February 1998. Available at https://tools.ietf.org/html/rfc2289

[15]. IETF RFC 4793. M. Nystroem. The EAP Protected One-Time Password Protocol (EAP-POTP). February 2007. Available at https://tools.ietf.org/html/rfc4793
[16]. IETF RFC 4186. Haverinen & Salowey. Extensible Authentication Protocol Method for Global System for Mobile Communications (GSM) Subscriber Identity Modules (EAP-SIM). January 2006. Available at https://tools.ietf.org/html/rfc4186
[17]. European Telecommunications Standards Institute, "GSM Technical Specification GSM 03.20 (ETS 300 534): "Digital cellular telecommunication system (Phase 2); Security related network functions"", August 1997.
[18]. European Telecommunications Standards Institute, "GSM Technical Specification GSM 03.03 (ETS 300 523): "Digital cellular telecommunication system (Phase 2); Numbering, addressing and identification"", April 1997.
[19]. IETF RFC 4187. Arkko & Haverinen. Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA). January 2006. Available at https://tools.ietf.org/html/rfc4187
[20]. 3rd Generation Partnership Project, "3GPP Technical Specification 3GPP TS 33.102 V5.1.0: "Technical Specification Group Services and System Aspects; 3G Security; Security Architecture (Release 5)"", December 2002.
[21]. 3rd Generation Partnership Project 2, "3GPP2 Enhanced Cryptographic Algorithms", September 2003.
[22]. 3rd Generation Partnership Project, "3GPP Technical Specification 3GPP TS 23.003 V6.8.0: "3rd Generation Parnership Project; Technical Specification Group Core Network; Numbering, addressing and identification (Release 6)"", December 2005.
[23]. IETF RFC 5448. Arkko, et al. Improved Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA'). May 2009. Available at https://tools.ietf.org/html/rfc5448
[24]. IETF RFC 4764. F. Bersani and H. Tschofenig. The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method. January 2007. Available at https://tools.ietf.org/html/rfc4764
[25]. IETF RFC 4763. M. Vanderveen and H. Soliman. Extensible Authentication Protocol Method for Shared-secret Authentication and Key Establishment (EAP-SAKE). November 2006. Available at https://tools.ietf.org/html/rfc4763
[26]. M. Bellare and P. Rogaway. Entity Authentication and key distribution. In Advances in Cryptology - Crypto 93 Proceedings, pages 232-249, 1993.
[27]. M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In Proc. 27th Annual Symposium on the Theory of Computing, pages 57-66, 1995.
[28]. IETF RFC 5433. Clancy & Tschofenig. Extensible Authentication Protocol - Generalized Pre-Shared Key (EAP-GPSK) Method. February 2009. Available at https://tools.ietf.org/html/rfc5433
[29]. IETF RFC 5931, Harkins & Zorn. Extensible Authentication Protocol (EAP) Authentication Using Only a Password. August 2010. Available at https://tools.ietf.org/html/rfc5931
[30]. Barker, E., Johnson, D., and M. Smid. Recommendations for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. NIST Special Publication 800-56A, March 2007.
[31]. IETF RFC 6124. Sheffer, et al. An EAP Authentication Method Based on the Encrypted Key Exchange (EKE) Protocol. February 2011. Available at https://tools.ietf.org/html/rfc6124

[32]. Bellovin, S. and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. Proc. IEEE Symp. on Research in Security and Privacy , May 1992.
[33]. IETF RFC 5216. Simon, et al. The EAP-TLS Authentication Protocol. March 2008. Available at https://tools.ietf.org/html/rfc5216
[34]. IETF RFC 4346. Dierks, T. and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. April 2006. Available at https://tools.ietf.org/html/rfc4346
[35]. IETF RFC 5106. Tschofenig, et al. The Extensible Authentication Protocol-Internet Key Exchange Protocol version 2 (EAP-IKEv2) Method. February 2008. Available at https://tools.ietf.org/html/rfc5106
[36]. IETF RFC 5281. Funk & Blake-Wilson. Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0). August 2008. Available at https://tools.ietf.org/html/rfc5281
[37]. IETF RFC 2865. Rigney, C., Willens, S., Rubens, A., and W. Simpson. Remote Authentication Dial In User Service (RADIUS). June 2000. Available at https://tools.ietf.org/html/rfc2865
[38]. IETF RFC 3588. Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko. Diameter Base Protocol. September 2003. Available at https://tools.ietf.org/html/rfc3588
[39]. IETF RFC 4851. Cam-Winget, et al. The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST). May 2007. Available at https://tools.ietf.org/html/rfc4851
[40]. IETF RFC 4507. Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. May 2006. Available at https://tools.ietf.org/html/rfc4507
[41]. IETF RFC 7170. Zhou, et al. Tunnel Extensible Authentication Protocol (TEAP) Version 1. May 2014. Available at https://tools.ietf.org/html/rfc7170
[42]. Microsoft Corporation. [MS-PEAP]: Protected Extensible Authentication Protocol (PEAP). December 2017. Available at https://msdn.microsoft.com/en-us/library/cc238354.aspx, 25.04.2018
[43]. IETF RFC 6678. Hoeper, K., Hanna, S., Zhou, H., and J. Salowey. Requirements for a Tunnel-Based Extensible Authentication Protocol (EAP) Method. July 2012. Available at https://tools.ietf.org/html/rfc6678
[44]. IETF RFC 5705. Rescorla, E. Keying Material Exporters for Transport Layer Security (TLS). March 2010. Available at https://tools.ietf.org/html/rfc5705
[45]. IETF RFC 5077. Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig. Transport Layer Security (TLS) Session Resumption without Server-Side State. January 2008. Available at https://tools.ietf.org/html/rfc5077

Синтаксический анализ графов с использованием конъюнктивных грамматик

Р.Ш. Азимов <rustam.azimov19021995@gmail.com>
С.В. Григорьев <Semen.Grigorev@jetbrains.com>
Санкт-Петербургский государственный университет,
199034, Россия, г. Санкт-Петербург, Университетская наб., д. 7/9

Аннотация. Графы используются в качестве структуры данных для представления больших объемов информации в компактной и удобной для анализа форме в различных областях – биоинформатике, графовых базах данных, статическом анализе кода и др. При этом оказывается необходимо вычислять запросы к большим графам с целью выявления зависимостей между их вершинами. Ответом на такие запросы обычно является множество всех троек (A, *m*, *n*), для которых существует путь в графе от вершины *m* до вершины *n* такой, что метки на ребрах этого пути образуют строку, выводимую из нетерминала A в некоторой контекстно-свободной грамматике. Говорят, что такой тип запросов вычисляется с использованием реляционной семантики запросов. Кроме того, существуют конъюнктивные грамматики, образующие более широкий класс грамматик, чем традиционные контекстно-свободные грамматики. Использование конъюнктивных грамматик в задаче синтаксического анализа графов позволяет формулировать более сложные запросы к графу и решать более широкий круг задач. Известно, что задача вычисления запросов к графу с использованием реляционной семантики и конъюнктивных грамматик является неразрешимой. В данной работе предлагается алгоритм, вычисляющий приближенное решение этой задачи, а именно, аппроксимацию сверху. Предложенный алгоритм основан на матричных операциях, и проведенные эксперименты показывают, что его производительность может быть существенно повышена, благодаря использованию вычислений на графическом процессоре.

Ключевые слова: синтаксический анализ графов; конъюнктивные грамматики; транзитивное замыкание; матричные операции; вычисления на графическом процессоре.

DOI: 10.15514/ISPRAS-2018-30(2)-8

Для цитирования: Азимов Р.Ш., Григорьев С.В. Синтаксический анализ графов с использованием конъюнктивных грамматик. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 149-166. DOI: 10.15514/ISPRAS-2018-30(2)-8

1. Введение

Графы используются во многих областях, например, в биоинформатике [1], в графовых базах данных [2], при статическом анализе программ [3], в навигационных сервисах для средств визуального моделирования [4]. При этом оказывается необходимым вычислять запросы к большим графам с целью выявления сложных зависимостей между их вершинами. Результатом вычисления таких запросов является множество неявных отношений между вершинами графа, то есть путей [5].

Естественно помечать ребра графа символами из некоторого конечного алфавита и выделять пути с помощью формальных грамматик над тем же алфавитом (регулярные выражения, контекстно-свободные грамматики). Говорят, что такой тип запросов вычислен с использованием реляционной семантики запросов. Наиболее популярными являются запросы, которые используют контекстно-свободные (КС) грамматики. Также существуют конъюнктивные грамматики [6], задающие более широкий класс языков, чем контекстно-свободные.

Использование конъюнктивных грамматик в задаче синтаксического анализа графов позволяет формулировать более сложные запросы к графам и решать более широкий круг задач, например, задачи поиска псевдонимов и уязвимостей в исходном коде [3]. Необходимо отметить, что задача вычисления запросов к графу с использованием реляционной семантики и конъюнктивных грамматик является неразрешимой [5]. Один из распространенных способов найти приближенное решение неразрешимой задачи – построить аппроксимацию решения.

В данной работе предложен алгоритм, вычисляющий приближенное решение задачи синтаксического анализа графов с использованием реляционной семантики запросов и конъюнктивных грамматик. Предложенный алгоритм основан на матричных операциях, что позволяет повысить производительность, используя для вычислений графический процессор.

Статья организована следующим образом. В разд. 2 приведены основные определения, связанные с задачей синтаксического анализа графов, и рассмотрены основные подходы данной области; в разд. 3 проанализированы существующие решения данной задачи; в разд. 4 представлен алгоритм синтаксического анализа графов, использующий реляционную семантику запросов и конъюнктивные грамматики, а также доказана его корректность; в разд. 5 работа предложенного алгоритма разобрана на небольшом примере; в разд. 6 представлены результаты проведенных экспериментов; заключение и направления будущих исследований приведены в разд. 7.

2. Обзор

В этом разделе мы определим задачу синтаксического анализа графов и обсудим основные подходы, применяемые для ее решения.

Пусть Σ – конечное множество терминальных символов. *Помеченным графом* будем называть пару $D = (V, E)$, где V является множеством вершин, а $E \subseteq V \times \Sigma \times V$ – множеством ребер с метками из алфавита Σ. Для пути π в графе D мы будем использовать обозначение $l(\pi)$, чтобы указать на слово, полученное конкатенацией меток на ребрах данного пути. Кроме того, запись $m\pi n$ будет обозначать, что в рассматриваемом графе существует путь из вершины $m \in V$ в вершину $n \in V$.

Результатом работы алгоритма синтаксического анализа графов с использованием формальной грамматики G обычно является множество всех троек (A, m, n), для которых путь $m\pi n$ таков, что строка $l(\pi)$ выводима из нетерминала A в грамматике G. Говорят, что такой тип запросов вычислен с использованием *реляционной семантики запросов* [5].

Традиционно в качестве грамматики G используются регулярные выражения [2, 7, 8, 9, 10]. Но в последнее время стало популярным использовать КС-грамматики [5, 11, 12, 13], так как некоторые полезные запросы не могут быть описаны с помощью регулярных грамматик. Примером таких запросов являются классические запросы поиска всех вершин в графе, находящихся на одном уровне иерархии [14]. Все рассмотренные алгоритмы синтаксического анализа графов принимают на вход КС-грамматики в *нормальной форме Хомского* [15].

Существует ряд алгоритмов синтаксического анализа графов с использованием реляционной семантики запросов и КС-грамматик [5, 12, 13], которые основаны на методе динамического программирования. Данные алгоритмы обобщают такие алгоритмы синтаксического анализа, как CYK [16, 17] и Earley [18]. В работе [5] для заданного графа $D = (V, E)$ и КС-грамматики $G = (N, \Sigma, P)$ для каждого $A \in N$ определяются *контекстно-свободные отношения* $R_A \subseteq V \times V$, являющиеся множеством пар вершин, между которыми существует путь, образующий строку, выводимую из нетерминала A. Такие отношения определяются следующим образом:

$$R_A = \{(n, m) \mid \exists n\pi m (l(\pi) \in L(G_A)\},$$

где $L(G_A)$ – это язык, порожденный грамматикой G со стартовым нетерминалом A.

В [5] представлен алгоритм, вычисляющий контекстно-свободные отношения R_A. Кроме того, существует алгоритм синтаксического анализа графов с использование реляционной семантики запросов и КС-грамматик, вычисляющий данные контекстно-свободные отношения R_A используя матричное транзитивное замыкание [11]. Данный алгоритм обобщает алгоритм Вэлианта [19] и сводится к умножению булевых матриц.

Также существуют *конъюнктивные грамматики* [6], образующие более широкий класс грамматик, чем контекстно-свободные. *Конъюнктивная грамматика* – это тройка $G = (N, \Sigma, P)$, где N – конечное множество

нетерминальных символов, Σ – конечное множество терминальных символов и P – конечное множество правил следующего вида:

- $A \to B_1 C_1 \& \dots \& B_m C_m$, для $m \geq 1, A, B_i, C_i \in N$,
- $A \to x$, для $A \in N, x \in \Sigma$.

Как и в случае КС-грамматик, мы рассматриваем только конъюнктивные грамматики в бинарной нормальной форме [20], так как для каждой конъюнктивной грамматики можно построить эквивалентную ей грамматику в данной форме. Кроме того, мы не выделяем стартовый нетерминал, так как его можно будет определить во время синтаксического анализа графа.

Мы будем писать $A \to^* w$, чтобы указать, что строка $w \in \Sigma^*$ может быть получена из нетерминала A некоторой последовательностью применений правил конъюнктивной грамматики. Отношение $\to$ определим следующим образом.

- При применении правила $(A \to B_1 C_1 \& \dots \& B_m C_m) \in P$ любой подтерм A любого терма может быть заменен подтермом $B_1 C_1 \& \dots \& B_m C_m$:
$$\dots A \dots \to \dots (B_1 C_1 \& \dots \& B_m C_m) \dots$$
- Конъюнкция нескольких одинаковых строк из Σ^* может быть перезаписана одной строкой, т.е. для любого $w \in \Sigma^*$
$$\dots (w \& \dots \& w) \dots \to \dots w \dots$$

Языком, задаваемым конъюнктивной грамматикой $G = (N, \Sigma, P)$ со стартовым нетерминалом $S \in N$, будем называть $L(G_S) = \{w \in \Sigma^* | S \to^* w\}$.

Определение контекстно-свободных отношений R_A естественным образом обобщается до определения конъюнктивных отношений R_A путем замены КС-грамматики на конъюнктивную. Таким образом, задача синтаксического анализа графов с использованием реляционной семантики запросов и конъюнктивных грамматик сводится к поиску конъюнктивных отношений R_A для всех нетерминалов A.

Но также известен факт, что данная задача является неразрешимой [5]. Поэтому единственный известный алгоритм для решения данной задачи, описанный в [3], дает приближенный результат. Данный алгоритм используется для статического анализа программ и аппроксимирует сверху конъюнктивные отношения R_A, то есть точное решение является подмножеством предложенного.

3. Существующие работы

Имеется ряд алгоритмов для синтаксического анализа графов с использованием реляционной семантики запросов и КС-грамматик [5, 12, 13], которые, однако, демонстрируют низкую производительность на больших графах, так как имеют в худшем случае кубическую временную сложность. Одной из самых популярных техник, используемых для увеличения производительности при работе с большими объемами данных, является использование графического процессора. В работе [11] представлен алгоритм синтаксического анализа

графов, использующий реляционную семантику запросов и КС-грамматики и вычисляющий матричное транзитивное замыкание с применением матричных операций. Кроме того, известно, что для вычислений матричных операции можно эффективно использовать графический процессор [21].

Однако все перечисленные алгоритмы работают только с КС-грамматиками и не позволяют обрабатывать запросы, описанные с помощью конъюнктивных грамматик. В работе [3] описан алгоритм, работающий с конъюнктивными грамматиками. Но данный алгоритм принимает на вход только определенный подкласс конъюнктивных грамматик, а именно, линейные конъюнктивные грамматики, которые имеют не более одного нетерминального символа в каждом конъюнкте правила.

Таким образом, задача построения алгоритма синтаксического анализа графов, использующего реляционную семантику запросов и работающего с произвольными конъюнктивными грамматиками, является открытой.

4. Алгоритм

В данном разделе мы представляем алгоритм синтаксического анализа графов, использующий реляционную семантику запросов и конъюнктивные грамматики. Предложенный алгоритм находит аппроксимацию сверху для конъюнктивных отношений R_A, вычисляя некоторое матричное транзитивное замыкание.

Пусть даны помеченный граф $D = (V, E)$ и конъюнктивная грамматика $G = (N, \Sigma, P)$. Определим *конъюнктивное матричное умножение* $a \circ b = c$, где a и b – матрицы подходящего размера, элементами которых являются подмножества множества нетерминалов N, следующим образом:

$$c_{i,j} = \{A \mid \exists (A \to B_1 C_1 \& \dots \& B_m C_m) \in P, \text{т.ч.} (B_k, C_k) \in d_{i,j}\},$$

где матрица d является результатом декартового произведения матриц a и b, т.е. $d_{i,j} = \bigcup_{k=1}^{n} a_{i,k} \times b_{k,j}$.

Кроме того, определим *конъюнктивное транзитивное замыкание* квадратной матрицы a как $a^{conj} = a^{(1)} \cup a^{(2)} \cup \dots$, где $a^{(i)} = a^{(i-1)} \cup (a^{(i-1)} \circ a^{(i-1)})$ для $i \geq 2$ и $a^{(1)} = a$.

Чтобы построить аппроксимацию сверху для конъюнктивных отношений R_A, мы построим матрицу T, используя ребра графа D и грамматику G. Для начала пронумеруем вершины графа D от 0 до (|V| - 1) и будем ассоциировать вершины с их номерами. Инициализируем каждый элемент |V|×|V| матрицы T пустым множеством $\emptyset$. Далее, для каждых i и j мы присваиваем в ячейки матрицы следующие значения:

$$T_{i,j} = \{A_k \mid \exists x(((i, x, j) \in E) \wedge ((A_k \to x) \in P))\}.$$

Наконец, мы вычисляем конъюнктивное транзитивное замыкание $T^{conj} = T^{(1)} \cup T^{(2)} \cup \dots$, где $T^{(i)} = T^{(i-1)} \cup (T^{(i-1)} \circ T^{(i-1)})$ для $i \geq 2$, а $T^{(1)}$ – это

построенная нами матрица T. Спецификация алгоритма представлена на листинге 1.

```
/* D - Входной помеченный граф
/* G – Входная конъюнктивная грамматика
/* T – Результат
1  n ← количество вершин графа D
2  E ← множество помеченных ребер графа D
3  P ← множество правил грамматики G
4  T ← матрица n × n с каждым элементом равным ∅
5  for each (i, x, j) ∈ E
6     T_{i,j} ← T_{i,j} ∪ {A| (A → x) ∈ P}
7  while матрица T изменяется do
8     T ← T ∪ (T ∘ T) /* вычисление транзитивного замыкания
```

Листинг 1. Алгоритм синтаксического анализа графов
Listing 1. A path querying algorithm

Для представленного алгоритма справедливы следующие утверждения.

Лемма 1. Пусть даны помеченный граф $D = (V, E)$ и конъюнктивная грамматика $G = (N, \Sigma, P)$ в нормальной форме. Рассмотрим вершины i, j такие, что $(i, j) \in R_A$ и существует путь $i\pi j$ такой, что для строки $l(\pi)$ существует дерево вывода грамматики $G = (N, \Sigma, P)$ с некоторым стартовым нетерминалом A с высотой $h \leq k$. Тогда $A \in T_{i,j}^{(k)}$.

Доказательство.

База индукции. Покажем, что утверждение леммы верно при $k = 1$. Для любых i, j и для любого нетерминала $A \in N$, если $(i, j) \in R_A$ и существует путь $i\pi j$ такой, что для строки $l(\pi)$ существует дерево вывода грамматики $G = (N, \Sigma, P)$ со стартовым нетерминалом A с высотой $h \leq 1$, то путь π состоит из единственного ребра, ведущего из вершины i в вершину j, и $(A \to x) \in P$, где $x = l(\pi)$. Таким образом, $A \in T_{i,j}^{(1)}$ и утверждение леммы при $k = 1$ доказано.

Индукционный переход. Предположим, что утверждение леммы верно для всех $k \leq (p - 1)$, где $p \geq 2$. Покажем, что утверждение леммы также верно при $k = p$.

Пусть существует путь $i\pi j$, образующий строку $l(\pi)$, для которой существует дерево вывода грамматики $G = (N, \Sigma, P)$ со стартовым нетерминалом A с высотой $h \leq p$. Если $h < p$, то утверждение леммы верно по индукционному предположению. Пусть $h = p$, и правило, использованное для корня этого дерева выглядит так: $(A \to B_1 C_1 \& \dots \& B_m C_m)$. Тогда все поддеревья для нетерминалов $(B_1, C_1, \dots B_m, C_m)$ имеют высоту меньше, чем p. Пусть каждая пара нетерминалов в конъюнктах $(B_1 C_1, \dots B_m C_m)$ разбивает в вершинах

$t_1, \dots, t_m$ путь $i\pi j$ на два подпути. По индукционному предположению получаем, что $B_x \in T_{i,t_x}^{(p-1)}$, а $C_x \in T_{t_x,j}^{(p-1)}$ для всех $1 \le x \le m$. Таким образом, все пары нетерминалов $(B_1, C_1), \dots (B_m, C_m)$ принадлежат $d_{i,j}$, где $d_{i,j} = \bigcup_{k=1}^{n} T_{i,k}^{(p-1)} \times T_{k,j}^{(p-1)}$. Значит, по определению конъюнктивного матричного умножения получаем, что $A \in T_{i,j}^{(p)}$, где $T^{(p)} = T^{(p-1)} \cup (T^{(p-1)} \circ T^{(p-1)})$. Таким образом, утверждение леммы также верно для $k = p$, ч. т. д.

Теорема 1. (*Корректность алгоритма*). Пусть даны помеченный граф $D = (V, E)$ и конъюнктивная грамматика $G = (N, \Sigma, P)$ в нормальной форме. Тогда для любых i, j и для любого нетерминала $A \in N$, если $(i, j) \in R_A$, то $A \in T_{i,j}^{conj}$.

Доказательство.

Пусть $(i, j) \in R_A$. По определению конъюнктивных отношений R_A получаем, что существует путь $i\pi j$ такой, что строка $l(\pi)$ принадлежит языку, порожденному грамматикой G со стартовым нетерминалом A. Рассмотрим некоторое дерево вывода этой грамматики для строки $l(\pi)$. Пусть высота этого дерева равна h. Тогда, по лемме 1 получаем, что $A \in T_{i,j}^{(h)}$. А так как $T^{conj} = T^{(1)} \cup T^{(2)} \cup \dots$, то $A \in T_{i,j}^{conj}$, ч. т. д.

Теорема 2. (*Конечность алгоритма*). Пусть даны помеченный граф $D = (V, E)$ и конъюнктивная грамматика $G = (N, \Sigma, P)$ в нормальной форме. Тогда конъюнктивное транзитивное замыкание матрицы T, построенной по графу D и грамматике G, может быть вычислено за конечное число операций.

Доказательство.

Если для некоторого $p \ge 2$, $T^{(p)} = T^{(p-1)}$, то для любого $k \ge p$, $T^{(k)} = T^{(p-1)}$ и $T^{conj} = T^{(p-1)}$. Таким образом, для нахождения матрицы T^{conj} достаточно вычислять матрицы $T^{(i)}$ до тех пор, пока не встретятся две подряд идущие одинаковые матрицы. По определению конъюнктивного транзитивного замыкания, в каждой ячейке матрицы $T^{(i+1)}$ количество нетерминалов больше либо равно, чем в соответствующей ячейке матрицы $T^{(i)}$. А так как общее количество нетерминалов в любой матрице $T^{(i)}$ не может быть больше, чем $|V|^2|N|$, то $T^{conj} = T^{|V|^2|N|}$. Таким образом, матрица T^{conj} может быть вычислена за конечное число операций, ч. т. д.

Таким образом, мы показали, как аппроксимация сверху всех конъюнктивных отношений R_A может быть найдена с помощью вычисления конъюнктивного транзитивного замыкания T^{conj}.

5. Пример работы алгоритма

В данном разделе мы продемонстрируем работу предложенного алгоритма на небольшом примере.

Пример основан на грамматике G со стартовым нетерминалом S и правилами, представленными на рис. 1.

$$S \to AB\&DC$$
$$A \to a$$
$$B \to BC$$
$$B \to b$$
$$C \to c$$
$$D \to AD$$
$$D \to b$$

Рис. 1. Пример входной конъюнктивной грамматики
Fig. 1. An example of the input conjunctive grammar

Конъюнкт AB порождает язык $L_{AB} = \{abc^*\}$, а конъюнкт DC порождает язык $L_{DC} = \{a^*bc\}$. Таким образом, конъюнктивная грамматика G со стартовым нетерминалом S порождает язык $L_S = L_{AB} \cap L_{DC} = \{abc\}$.

Рассмотрим работу предложенного алгоритма с входной грамматикой G и графом D, представленным на рис. 2.

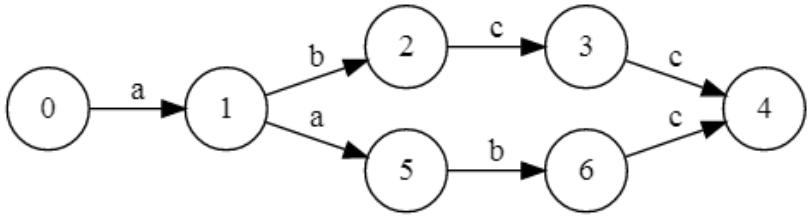


Рис. 2. Пример входного графа
Fig. 2. An example of the input graph

Сначала матрица T инициализируется в строках 5-6 листинга 1. Мы будем индексировать состояния матрицы T. Обозначим через T_0 состояние матрицы T, после инициализации. Матрица T_0 приведена на рис. 3.

$$\begin{pmatrix} \emptyset & \{A\} & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{B,D\} & \emptyset & \emptyset & \{A\} & \emptyset \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

Рис. 3. Матрица T_0
Fig. 3. The matrix T_0

Далее в строках 7-8 листинга 1 вычисляются последующие состояния матрицы T до тех пор, пока не найдется такое $k \geq 1$, что $T_k = T_{k-1}$. Таким образом, матрица T_k будет является результатом работы алгоритма.

Для вычисления матрицы $T_1 = T_0 \cup (T_0 \circ T_0)$ необходимо вычислить матрицу d, являющуюся декартовым произведением матрицы T_0 с собой же. Матрица d представлена на рис. 4.

$$\begin{pmatrix} \emptyset & \emptyset & \{(A,B),(A,D)\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \{(B,C),(D,C)\} & \emptyset & \emptyset & \{(A,B),(A,D)\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{(C,C)\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{(B,C),(D,C)\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \end{pmatrix}$$

Рис. 4. Матрица d
Fig. 4. The matrix d

По матрице d строится матрица $T_1 = T_0 \cup (T_0 \circ T_0)$, представленная на рис. 5. Новые нетерминалы матрицы T_1, по сравнению с матрицей T_0, выделены жирным шрифтом.

$$\begin{pmatrix} \emptyset & \{A\} & \{\mathbf{D}\} & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \{B,D\} & \{\mathbf{B}\} & \emptyset & \{A\} & \{\mathbf{D}\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{\mathbf{B}\} & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

Рис. 5. Матрица $T_1 = T_0 \cup (T_0 \circ T_0)$
Fig. 5. The matrix $T_1 = T_0 \cup (T_0 \circ T_0)$

Далее вычисляются последующие состояния матрицы T до T_4 включительно, так как $T_4 = T_3$. Все последующие состояния матрицы T представлены на рис. 6. Новые нетерминалы матрицы T_i, по сравнению с матрицей T_{i-1}, выделены жирным шрифтом.

$$T_2 = \begin{pmatrix} \emptyset & \{A\} & \{D\} & \{\mathbf{S}\} & \emptyset & \emptyset & \{\mathbf{D}\} \\ \emptyset & \emptyset & \{B,D\} & \{B\} & \{\mathbf{S},\mathbf{B}\} & \{A\} & \{D\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

$$T_3 = \begin{pmatrix} \emptyset & \{A\} & \{D\} & \{S\} & \{\mathbf{S}\} & \emptyset & \{D\} \\ \emptyset & \emptyset & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

$$T_4 = \begin{pmatrix} \emptyset & \{A\} & \{D\} & \{S\} & \{S\} & \emptyset & \{D\} \\ \emptyset & \emptyset & \{B,D\} & \{B\} & \{S,B\} & \{A\} & \{D\} \\ \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset & \emptyset \\ \emptyset & \emptyset & \emptyset & \emptyset & \{B\} & \emptyset & \{B,D\} \\ \emptyset & \emptyset & \emptyset & \emptyset & \{C\} & \emptyset & \emptyset \end{pmatrix}$$

Рис. 6. Последующие состояния матрицы T
Fig. 6. Remaining states of the matrix T

Таким образом, результатом работы предложенного алгоритма является матрица T_4. Далее мы можем сконструировать множества R'_A, которые являются аппроксимацией сверху конъюнктивных отношений R_A для всех нетерминалов $A \in N$, т.е. $R'_A \supseteq R_A$. Например, так как в ячейке (0,3) результирующей матрицы T_4 имеется нетерминал S, то $(0,3) \in R'_S$. Сконструированные множества R'_A для всех нетерминалов $A \in N$ представлены на рис. 7.

$$\begin{aligned} R'_S &= \{(0,3),(0,4),(1,4)\} \\ R'_A &= \{(0,1),(1,5)\} \\ R'_B &= \{(1,2),(1,3),(1,4),(5,4),(5,6)\} \\ R'_C &= \{(2,3),(3,4),(6,4)\} \\ R'_D &= \{(0,2),(0,6),(1,2),(1,6),(5,6)\} \end{aligned}$$

Рис. 7. Аппроксимация конъюнктивных отношений
Fig. 7. An approximation of the conjunctive relations

Данный пример демонстрирует, что не всегда удается получить точное решение. Например, пара вершин (0,4) принадлежит R'_S, хотя не существует пути из вершины 0 в вершину 4, образующего строку, выводимую из нетерминала S (таковой является единственная строка *abc*). Лишние пары вершин добавляются в том случае, если существуют различные пути из вершины *i* в вершину *j*, которые суммарно соответствуют конъюнктам одного правила, но не существует пути из вершины *i* в вершину *j*, который одновременно соответствовал бы всем конъюнктам этого правила. Например, для конъюнктов правила $S \to AB\&DC$ существует путь из вершины 0 в

вершину 4, образующий строку *abcc*, и существует также путь из вершины 0 в вершину 4, образующий строку *aabc*. Первый путь соответствует конъюнкту *AB*, так как строка *abcc* принадлежит языку $L_{AB} = \{abc^*\}$, а второй путь соответствует конъюнкту *DC*, так как строка *aabc* принадлежит языку $L_{DC} = \{a^*bc\}$. Однако, очевидно, что не существует пути из вершины 0 в вершину 4, образующего строку *abc*.

6. Эксперименты

В этом разделе мы приводим результаты экспериментов, целью которых является демонстрация практической применимости предложенного алгоритма и возможности его значительного ускорения при использовании для вычислений графического процессора. Для достижения данной цели мы реализовали алгоритм двумя способами: вычисляя матричные операции на ЦПУ и вычисляя их на графическом процессоре. Мы запустили данные реализации на классических конъюнктивных грамматиках [6] и случайно сгенерированных графах.

6.1 Постановка экспериментов

Эксперименты проводились на рабочей станции со следующими характеристиками: операционная система – Microsoft Windows 10 Pro x64-based PC, ЦПУ – Intel i7-4790, 3601 Mhz, 4 Core(s), 4 Logical Processor(s), оперативная память – 16 GB, графический процессор – NVIDIA GeForce GTX 1070 (CUDA Cores: 1920, Core clock: 1556 MHz, Memory data rate:8008 MHz, Memory interface: 256-bit, Memory bandwidth: 256.26 GB/s, Dedicated video memory: 8192 MB GDDR5).

Было выполнено две реализации алгоритма на языке программирования F# [22].

- Реализация *onCPU* использует ЦПУ для вычисления матричных операций. Для хранения матриц в разреженном формате и вычисления матричных операций использована библиотека Math.Net Numerics [23].
- Реализация *onGPU* использует графический процессор для вычисления матричных операций. Для вычисления матричных операций на графическом процессоре использована обёртка CUSPARSE-библиотеки, взятую из библиотеки managedCuda [24].

Сравнение производительности двух выполненных реализаций позволит определить эффективность ускорения предложенного алгоритма, с использованием вычислений матричных операций на графическом процессоре.

Для экспериментов мы использовали два запроса к случайно сгенерированным графам, соответствующим двум классическим конъюнктивным грамматикам.

Запрос 1. Данный запрос основан на конъюнктивной грамматике G со стартовым нетерминалом S, которая порождает язык $\{a^n b^n c^n | n > 0\}$. Правила данной грамматики представлены на рис. 8.

$$S \to AB\&DC$$
$$A \to AA \mid a$$
$$B \to bBc \mid bc$$
$$C \to CC \mid c$$
$$D \to aDb \mid ab$$

Рис. 8. Конъюнктивная грамматика, порождающая язык $\{a^n b^n c^n | n > 0\}$
Fig. 8. An example of the input conjunctive grammar for language $\{a^n b^n c^n | n > 0\}$

Представленная грамматика приводится в бинарную нормальную форму и подается на вход алгоритма вместе со случайно сгенерированными графами, метки на ребрах которых принадлежат алфавиту $\Sigma = \{a, b, c\}$. Предложенный алгоритм выделяет пары вершин в найденной аппроксимации конъюнктивного отношения R'_S, как потенциальные, между которыми существует путь, формирующий строку из языка $\{a^n b^n c^n | n > 0\}$.

Запрос 2. Данный запрос основан на конъюнктивной грамматике G со стартовым нетерминалом S, которая порождает язык $\{wcw \mid w \in \{a, b\}^*\}$. Правила данной грамматики представлены на рис. 9.

$$S \to C\&D$$
$$C \to aCa \mid aCb \mid bCa \mid bCb \mid c$$
$$D \to aA\&aD \mid bB\&bD \mid cE$$
$$A \to aAa \mid aAb \mid bAa \mid bAb \mid cEa$$
$$B \to aBa \mid aBb \mid bBa \mid bBb \mid cEb$$
$$E \to aE \mid bE \mid \epsilon$$

Рис. 9. Конъюнктивная грамматика, порождающая язык $\{wcw \mid w \in \{a, b\}^*\}$
Fig. 9. An example of the input conjunctive grammar for language $\{wcw \mid w \in \{a, b\}^*\}$

Представленная грамматика также приводится в бинарную нормальную форму и генерируются графы с метками на ребрах из алфавита $\Sigma = \{a, b, c\}$. Предложенный алгоритм выделяет пары вершин в найденной аппроксимации конъюнктивного отношения R'_S, как потенциальные, между которыми существует путь, формирующий строку из языка $\{wcw \mid w \in \{a, b\}^*\}$.

6.2 Результаты

Результаты вычислений запроса 1 представлены в табл. 1, где $|V|$ является количеством вершин сгенерированного графа, $|E|$ – количеством ребер, *#results* – это количество пар вершин в найденной аппроксимации

конъюнктивного отношения R'_S. Для двух выполненных реализаций в соответствующих столбцах представлено время работы алгоритма в миллисекундах.

Табл. 1. Результаты вычислений запроса 1
Table 1. Evaluation results for query 1

|V|	**|E|**	**#results**	**onCPU (ms)**	**onGPU (ms)**
100	*25*	*0*	*2*	*7*
100	*75*	*0*	*10*	*20*
100	*200*	*79*	*101*	*213*
1000	*250*	*1*	*265*	*25*
1000	*750*	*13*	*2781*	*102*
1000	*2000*	*731*	*12050*	*347*
10000	*2500*	*4*	*26595*	*41*
10000	*7500*	*136*	*241087*	*213*
10000	*20000*	*4388*	*1305177*	*1316*

Результаты вычислений запроса 2 представлены в табл. 2. Для двух предложенных реализаций в соответствующих столбцах представлено время работы алгоритма в миллисекундах. Время работы реализации *onCPU* для входного графа с 10000 вершин и 10000 ребер опущено из-за слишком низкой производительности данной реализации на больших графах.

6.3 Анализ результатов

Результаты вычислений запроса 1 показывают, что реализация, вычисляющая матричные операции на ЦПУ, более производительная только на нескольких небольших графах. Это связано с временными затратами на выделение памяти и обмен данными с графическим процессором. Кроме того, прирост производительности при использовании графического процессора для вычисления матричных операций значительно увеличивается с ростом размера входного графа.

Табл. 2. Результаты вычислений запроса 2
Table 2. Evaluation results for query 2

|V|	**|E|**	**#results**	**onCPU (ms)**	**onGPU (ms)**
100	*25*	*9*	*14*	*67*

100	*75*	*29*	*114*	*129*
100	*100*	*47*	*254*	*483*
1000	*250*	*82*	*2566*	*127*
1000	*750*	*279*	*21394*	*530*
1000	*1000*	*438*	*64725*	*1951*
10000	*2500*	*829*	*268843*	*257*
10000	*7500*	*2796*	*3380046*	*1675*
10000	*10000*	*27668*	*–*	*3017*

Аналогично, результаты вычислений запроса 2 показывают, что реализация *onCPU* более производительная по сравнению с реализацией *onGPU* только на нескольких небольших графах и прирост производительности при использовании графического процессора также значительно увеличивается с ростом размера входного графа. Кроме того, сравнение результатов вычислений запросов 1 и 2 показывает, что вычисление запроса 2 требует значительно большего времени, чем вычисление запроса 1 при аналогичных входных графах. Одной из причин этого является наличие большего количества правил в грамматике, приведенной в нормальную форму, для запроса 2, чем в аналогичной грамматике для запроса 1.

Таким образом, поставленные эксперименты демонстрируют практическую применимость предложенного алгоритма и показывают наличие возможности его значительного ускорения благодаря использованию вычислений на графическом процессоре.

7. Заключение

В данной работе был предложен алгоритм, вычисляющий приближенное решение задачи синтаксического анализа графов с использованием реляционной семантики запросов и конъюнктивных грамматик. Кроме того, были проведены эксперименты, демонстрирующие практическую применимость предложенного алгоритма и возможность повысить его производительность, используя для вычислений графический процессор.

Кроме того, мы можем определить несколько открытых проблем для будущих исследований. В данной работе была рассмотрена только одна семантика запросов – реляционная. Но существуют другие семантики запросов [25], в которых необходим предоставлять один или даже все пути, образующие строки, выводимые из нетерминалов входной грамматики. Может ли предложенный алгоритм быть обобщен до данных семантик запросов?

Также, существуют Булевы грамматики [26], которые принадлежат более широкому и выразительному классу грамматик, чем конъюнктивные.

Существует также матричный алгоритм [27] синтаксического анализа строк, использующий Булевы грамматики. Может ли наш алгоритм быть обобщен до алгоритма синтаксического анализа графов использующего реляционную семантику запросов и Булевы грамматики?

Список литературы

[1]. Anderson J. W. et al. Quantifying variances in comparative RNA secondary structure prediction. BMC bioinformatics, vol. 14, № 1, 2013.
[2]. Mendelzon A., Wood P. Finding Regular Simple Paths in Graph Databases. SIAM J. Computing, vol. 24, № 6, 1995, pp. 1235–1258.
[3]. Zhang Q., Su Z. Context-sensitive data-dependence analysis via linear conjunctive language reachability. Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, 2017, pp. 344–358.
[4]. Кознов Д.В., Ларчик Е.В., Терехов А.Н. Трансформация динамических представлений в предметно-ориентированном визуальном моделировании. Программирование, том 41, № 4, 2015 г., стр. 3-12
[5]. Hellings. J. Conjunctive context-free path queries. In: Proc. of ICDT'14, 2014, pp.119–130.
[6]. Okhotin A. Conjunctive grammars. Journal of Automata, Languages and Combinatorics, vol. 6, № 4, 2001, pp. 519–535.
[7]. Abiteboul S., Vianu V. Regular path queries with constraints. In Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1997 pp. 122–133.
[8]. Fan W., Li J., Ma S., Tang N, and Wu Y. 2011. Adding regular expressions to graph reachability and pattern queries. In 27th Data Engineering International Conference, 2011, pp. 39–50.
[9]. Nolé M. and Sartiani C. Regular path queries on massive graphs. In Proceedings of the 28th International Conference on Scientific and Statistical Database Management, 2016.
[10]. Reutter J., Romero M., and Vardi M. Regular queries on graph databases. Theory of Computing Systems, vol. 61, № 1, 2017, pp. 31–83
[11]. Azimov R. Sh., Grigorev S. V. Context-Free Path Querying by Matrix Multiplication. arXiv preprint, arXiv:1707.01007v2, 2017.
[12]. Sevon P., Eronen L. Subgraph queries by context-free grammars. Journal of Integrative Bioinformatics, vol. 5, № 2, 2008.
[13]. Zhang X. et al. Context-free path queries on RDF graphs. International Semantic Web Conference, 2016, pp. 632–648.
[14]. Abiteboul S., Hull R., Vianu V. Foundations of databases: the logical level. Addison-Wesley Longman Publishing Co., Inc., 1995.
[15]. Chomsky N. On certain formal properties of grammars. Information and control, vol. 2, № 2, 1959, pp. 137–167.
[16]. Kasami T. An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages. Report of University of Hawaii, Contract No. AF 19(628)-4379, No. 2, July, 1965.
[17]. Younger D.H. Recognition and parsing of context-free languages in time n3, Information and control, vol. 10, № 2, 1967, pp. 189–208.
[18]. Grune D., Jacobs C. J. H. Parsing Techniques (Monographs in Computer Science). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[19]. Valiant L.G. General context-free recognition in less than cubic time. Journal of computer and system sciences, vol., №2, pp. 308– 315.
[20]. Okhotin A. Conjunctive and Boolean grammars: the true general case of the context-free grammars. Computer Science Review, vol. 9, 2013. pp. 27–59.
[21]. Che S., Beckmann B.M., Reinhardt S.K. Programming GPGPU Graph Applications with Linear Algebra Building Blocks. International Journal of Parallel Programming, vol. 45, Issue 3, June 2017, pp 657–679.
[22]. Syme D., Granicz A., and Cisternino A. Expert F# 3.0. Springer, 2012.
[23]. The Math.Net Numerics WebSite. Доступно по ссылке: https://numerics.mathdotnet.com/, 20.03.2018.
[24]. The managedCuda library. Доступно по ссылке: https://kunzmi.github.io/managedCuda/, 20.03.2018.
[25]. Hellings J. Querying for Paths in Graphs using Context-Free Path Queries. arXiv preprint arXiv:1502.02242, 2015.
[26]. Okhotin A. 2004. Boolean grammars. Information and Computation, vol. 194, issue 1, 2004, pp. 19–48.
[27]. Okhotin A. Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science, vol. 516, 2014, pp. 101–120.

Path querying using conjunctive grammars

R.Sh. Azimov <rustam.azimov19021995@gmail.com>
S.V. Grigorev <Semen.Grigorev@jetbrains.com>
Saint Petersburg State University,
7/9, Universitetskaya nab., St. Petersburg, 199034, Russia

Abstract. Graphs are used as a data structure to represent large volumes of information in a compact and convenient for analysis form in many areas: bioinformatics, graph databases, static code analysis, etc. In these areas, it is necessary to evaluate a queries for large graphs in order to determine the dependencies between the nodes. The answer to such queries is usually a set of all triples (A, m, n) for which there is a path in the graph from the vertex m to the vertex n, such that the labels on the edges of this path form a string derivable from the nonterminal A in some context-free grammar. This type of query is calculated using relational query semantics and context-free grammars but there are conjunctive grammars, which form a broader class of grammars than traditional context-free grammars. The use of conjunctive grammars in the path querying allows us to formulate more complex queries to the graph and solve a wider class of problems. It is known that the path querying using relational query semantics and conjunctive grammars is undecidable problem. In this paper, we propose a path querying algorithm which uses relational query semantics and conjunctive grammars, and approximates the exact solution. The proposed algorithm is based on matrix operations, and our evaluation shows that it is possible to significantly improve the performance of the algorithm by using a graphics processing unit.

Keywords: path querying; conjunctive grammars; transitive closure; matrix operations; GPGPU.

DOI: 10.15514/ISPRAS-2018-30(2)-8

For citation: Azimov R.Sh., Grigorev S.V. Path querying using conjunctive grammars. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue. 2, 2018, pp. 149-166 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-8

References

[1]. Anderson J. W. et al. Quantifying variances in comparative RNA secondary structure prediction. BMC bioinformatics, vol. 14, № 1, 2013.
[2]. Mendelzon A., Wood P. Finding Regular Simple Paths in Graph Databases. SIAM J. Computing, vol. 24, № 6, 1995, pp. 1235–1258.
[3]. Zhang Q., Su Z. Context-sensitive data-dependence analysis via linear conjunctive language reachability. Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, 2017, pp. 344–358.
[4]. Koznov D.V., Larchik E.V., Terekhov A.N. View to view transformations in domain specific modeling. Programming and Computer Software, vol. 41, № 4, 2015, pp. 208-214. DOI: 10.1134/S0361768815040039.
[5]. Hellings. J. Conjunctive context-free path queries. In: Proc. of ICDT'14, 2014, pp.119–130.
[6]. Okhotin A. Conjunctive grammars. Journal of Automata, Languages and Combinatorics, vol. 6, № 4, 2001, pp. 519–535.
[7]. Abiteboul S., Vianu V. Regular path queries with constraints. In Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, 1997 pp. 122–133.
[8]. Fan W., Li J., Ma S., Tang N, and Wu Y. 2011. Adding regular expressions to graph reachability and pattern queries. In 27th Data Engineering International Conference, 2011, pp. 39–50.
[9]. Nolé M. and Sartiani C. Regular path queries on massive graphs. In Proceedings of the 28th International Conference on Scientific and Statistical Database Management, 2016.
[10]. Reutter J., Romero M., and Vardi M. Regular queries on graph databases. Theory of Computing Systems, vol. 61, № 1, 2017, pp. 31–83
[11]. Azimov R. Sh., Grigorev S. V. Context-Free Path Querying by Matrix Multiplication. arXiv preprint, arXiv:1707.01007v2, 2017.
[12]. Sevon P., Eronen L. Subgraph queries by context-free grammars. Journal of Integrative Bioinformatics, vol. 5, № 2, 2008.
[13]. Zhang X. et al. Context-free path queries on RDF graphs. International Semantic Web Conference, 2016, pp. 632–648.
[14]. Abiteboul S., Hull R., Vianu V. Foundations of databases: the logical level. Addison-Wesley Longman Publishing Co., Inc., 1995.
[15]. Chomsky N. On certain formal properties of grammars. Information and control, vol. 2, № 2, 1959, pp. 137–167.
[16]. Kasami T. An Efficient Recognition and Syntax-Analysis Algorithm for Context-Free Languages. Report of University of Hawaii, Contract No. AF 19(628)-4379, No. 2, July, 1965.
[17]. Younger D.H. Recognition and parsing of context-free languages in time n3, Information and control, vol. 10, № 2, 1967, pp. 189–208.
[18]. Grune D., Jacobs C. J. H. Parsing Techniques (Monographs in Computer Science). Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[19]. Valiant L.G. General context-free recognition in less than cubic time. Journal of computer and system sciences, vol., №2, pp. 308– 315.
[20]. Okhotin A. Conjunctive and Boolean grammars: the true general case of the context-free grammars. Computer Science Review, vol. 9, 2013. pp. 27–59.
[21]. Che S., Beckmann B.M., Reinhardt S.K. Programming GPGPU Graph Applications with Linear Algebra Building Blocks. International Journal of Parallel Programming, vol. 45, Issue 3, June 2017, pp 657–679.
[22]. Syme D., Granicz A., and Cisternino A. Expert F# 3.0. Springer, 2012.
[23]. The Math.Net Numerics WebSite. Доступно по ссылке: https://numerics.mathdotnet.com/, 20.03.2018.
[24]. The managedCuda library. Доступно по ссылке: https://kunzmi.github.io/managedCuda/, 20.03.2018.
[25]. Hellings J. Querying for Paths in Graphs using Context-Free Path Queries. arXiv preprint arXiv:1502.02242, 2015.
[26]. Okhotin A. 2004. Boolean grammars. Information and Computation, vol. 194, issue 1, 2004, pp. 19–48.
[27]. Okhotin A. Parsing by matrix multiplication generalized to Boolean grammars. Theoretical Computer Science, vol. 516, 2014, pp. 101–120.

Проблема отката в ориентированной распределенной системе

И.Б. Бурдонов <igor@ispras.ru>[1]
А.С. Косачев <kos@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.

Аннотация. Для распределенной системы, в основе которой лежит ориентированный граф без кратных ребер и петель, рассматривается проблема отката (backtracing problem): как передать сообщение от конца дуги в ее начало. Ставится задача создания на графе структуры, позволяющей передавать сообщение из конца любой дуги в ее начало по кратчайшему пути. Такая структура в каждой вершине a задаётся отображением номера вершины b в номер исходящей дуги, через которую проходит кратчайших путь от a к b. В частности, такое отображение позволяет симулировать в ориентированных распределенных системах алгоритмы решения задач на графе, разработанные для неориентированных распределенных систем. Это увеличивает время работы таких алгоритмов (в тактах, где время пересылки сообщения по дуге не превосходит 1 такт) не более чем в k раз, где k диаметр графа, $k < n$, и n – число вершин графа. В разделе 2 описывается используемая асинхронная модель распределенной системы. Раздел 3 содержит основные определения и обозначения, а раздел 4 – постановку задачи. В разделе 5 описываются два вспомогательных алгоритма коррекции поддеревьев, применение которых позволяет строить остовные деревья кратчайших путей: прямого дерева, ориентированного от корня графа, и обратного дерева, ориентированного к корню графа. Раздел 6 содержит описание различных способов передачи сообщений по графу. В разделе 7 предлагаются два алгоритма построения в памяти автомата корня графа описаний прямого и обратного остовных деревьев кратчайших путей, а в разделе 8 – основанные на них алгоритмы построения требуемого отображения: «быстрый» алгоритм с оценками $T=\boldsymbol{O}(n)$ и $N=\boldsymbol{O}(n)$ и «экономный» алгоритм с оценками $T=\boldsymbol{O}(n^2)$ и $N=\boldsymbol{O}(1)$, где T – время (в тактах) работы алгоритма, N – число сообщений, одновременно передаваемых по дуге. В разделе 9 доказано, что эти оценки времени не улучшаемые. В разделе 10 «быстрый» алгоритм модифицируется для синхронной модели с $N=1$. Заключение подводит итоги и намечает направления дальнейших исследований.

Ключевые слова: ориентированный граф; корневой граф; нумерованный граф; распределенные алгоритмы; задачи на графах; проблема отката; кратчайшие пути.

DOI: 10.15514/ISPRAS-2018-30(2)-9

[1] Работа частично поддержана проектом РФФИ № 17-07-00682 А.

Для цитирования: Бурдонов И.Б., Косачев А.С. Проблема отката в ориентированной распределенной системе. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 167-194. DOI: 10.15514/ISPRAS-2018-30(2)-9

1. Введение

Распределенная система – это граф, в вершинах которого располагаются вычислительные единицы (автоматы), которые могут обмениваться между собой сообщениями, посылаемыми по ребрам графа. Если граф ориентированный, то ориентированное ребро называется дугой и соответствует симплексному каналу передачи сообщений: сообщение может передаваться по дуге только в одном направлении: от начала дуги к концу дуги. Это создает проблему отката (*backtracking problem*): как передать сообщение от конца дуги в ее начало?

Первоначально эта проблема изучалась в контексте обхода ориентированного графа роботом (конечным автоматом), двигающимся по дугам графа [1][2][3]. Модель с одним двигающимся роботом эквивалентна «инвертированной» модели с одним сообщением, передаваемым по графу, и идентичными роботами, находящимися в вершинах графа: робот в вершине, принимая сообщение по некоторой входящей дуге, посылает одно сообщение по одной исходящей дуге или завершает работу алгоритма. Длина обхода ориентированного графа без кратных ребер равна $\boldsymbol{O}(n^2)$, где n – число вершин. Однако из-за проблемы отката оценка алгоритмического обхода увеличивается, в зависимости от алгоритма, до $\boldsymbol{O}(n^2 log n)$ [1], $\boldsymbol{O}(n^2 log log n)$ [2], а наилучший известный результат равен $\boldsymbol{O}(n^2 log^* n)$ [3], где log^* – итерированный логарифм (количество итеративных логарифмирований аргумента, необходимых для того, чтобы результат стал меньше или равен 1).

В данной работе мы исследовали проблему отката для распределенной системы, в которой может быть несколько сообщений, одновременно передаваемых по дугам графа. Граф предполагается *нумерованным*: вершинам присвоены номера от 1 до n. Граф *корневой*, т.е. имеется выделенная вершина, называемая *корнем* графа, с которой начинается и в которой заканчивается выполнение алгоритма. Граф *упорядоченный*: дуги, выходящие из вершины перенумерованы от 1 до полустепени исхода этой вершины. Заметим, что упорядоченность графа необходима для того, чтобы автомат в вершине мог указывать исходящую дугу, по которой он хочет передать сообщение. Сообщение перемещается по дуге за время не более 1 такта.

Мы поставили задачу создания на графе структуры, позволяющей передавать сообщение из конца любой дуги в ее начало по кратчайшему пути, т.е. за минимальное время в наихудшем случае, когда по каждой дуге сообщение пересылается 1 такт. В памяти автомата каждой вершины a эта структура задается *отображением* номера вершины b в номер исходящей дуги, через которую проходит кратчайший путь от a к b. В частности, такое отображение позволяет симулировать в ориентированных распределенных системах

алгоритмы решения задач на графе, разработанные для неориентированных распределенных систем (например, [4][5]). Вместо прохода по одной дуге в обратном направлении проходится ориентированный путь из конца дуги в её начало, что увеличивает время работы таких алгоритмов не более чем в k раз, где $k < n$ диаметр графа (максимальное расстояние между вершинами).

В разд. 2 описывается используемая асинхронная модель распределенной системы. Разд. 3 содержит основные используемые определения и обозначения, а разд. 4 – постановку задачи. В разд. 5 описываются два вспомогательных алгоритма коррекции поддеревьев, применение которых в последующих алгоритмах позволяет строить остовные деревья кратчайших путей: прямого дерева, ориентированного от корня графа, и обратного дерева, ориентированного к корню графа. Разд. 6 содержит описание пяти различных способов передачи сообщений по графу, которые используются в алгоритмах, описываемых ниже.

В разд. 7 предлагаются два алгоритма построения в памяти автомата корня графа описаний прямого и обратного остовных деревьев кратчайших путей. Эти алгоритмы отличаются соотношением времени T работы и числа N сообщений, одновременно передаваемых по дуге: в «быстром» алгоритме $T=\boldsymbol{O}(n)$ и $N=\boldsymbol{O}(n)$, а в «экономном» алгоритме $T=\boldsymbol{O}(n^2)$ и $N=\boldsymbol{O}(1)$.

Разд. 8 содержит описание двух основных алгоритмов построения требуемого отображения, т.е. отображения в каждой вершине a номера другой вершины b в номер первой дуги на кратчайшем пути из a в b. «Быстрый» алгоритм имеет оценки $T=\boldsymbol{O}(n)$ и $N=\boldsymbol{O}(n)$, «экономный» алгоритм имеет оценки $T=\boldsymbol{O}(n^2)$ и $N=\boldsymbol{O}(1)$. В разд. 9 доказано, что эти оценки не улучшаемы. Разд. 10 посвящён синхронной модели, в которой любое сообщение перемещается по любой дуге ровно за 1 такт. Заключение подводит итоги и намечает направления дальнейших исследований.

2. Модель

Граф распределенной системы предполагается ориентированным, сильно связным, нумерованным, корневым и упорядоченным без кратных дуг и петель. *Емкостью* дуги называется число сообщений, которые могут одновременно перемещаться по дуге. В рассматриваемой модели предполагается неограниченная емкость дуги. Это означает, что автомат в вершине не получает каких-либо «извещений» о том, что исходящая дуга уже полностью «заполнена» или, наоборот, «освободилась». Но для предлагаемых алгоритмов мы будем давать оценку числа сообщений на дуге.

Будем считать, что в одном срабатывании автомат принимает ровно одно сообщение ровно по одной входящей дуге и посылает по каждой исходящей дуге не более одного сообщения. Для удобства описания алгоритма мы будем иногда говорить, что в одном срабатывании автомат посылает по одной исходящей дуге несколько (но конечное число) сообщений $m_1,m_2,...m_e$, имея в виду, что они «склеиваются» в одно сообщение $m = m_1 \cdot m_2 \cdot ... m_e$. Число e таких

сообщений во всех предлагаемых ниже алгоритмах не превосходит 3. Принимая такое «склеенное» сообщение m, автомат запоминает его в своей памяти и обрабатывает его компоненты $m_1,m_2,...m_e$ последовательно, как если бы он принял их в последовательных e срабатываниях. Мы будем пренебрегать временем срабатывания автомата в вершине, т.е. будем оценивать сложность алгоритмов как время перемещения сообщений по дугам.

Для краткости там, где это не приведёт к недоразумениям, мы будем вместо «автомат в вершине» или «память автомата в вершине» говорить просто «вершина» или «память вершины». Память вершины будет рассматриваться как набор *переменных*, а сообщение – как набор *параметров*.

Мы будем давать следующие оценки алгоритмов: T – время работы алгоритма, A – размер памяти вершины как сумма размеров (в битах) переменных, M – размер сообщения как сумма размеров (в битах) параметров сообщения, N – число сообщений, одновременно передаваемых по одной дуге. Время перемещения сообщения по одной дуге считается ограниченным сверху одним тактом.

3. Определения и обозначения

Определения:

- *Маршрут* – последовательность дуг графа, в которой конец не последней дуги совпадает с началом следующей дуги.
- *Путь* – маршрут, в котором никакие две дуги не имеют общего конца, и конец последней дуги не совпадает с началом первой дуги.
- *Расстояние* от вершины a до вершины b – длина кратчайшего пути от вершины a до вершины b.
- *Прямое дерево* – дерево с выделенным корнем дерева, в котором все дуги ориентированы от корня дерева.
- *Обратное дерево* – дерево с выделенным корнем дерева, в котором все дуги ориентированы к корню дерева.
- *Прямое дерево кратчайших путей* – такой подграф, являющийся прямым деревом, что расстояние от корня дерева до каждой вершины дерева по дереву и по графу одинаковы.
- *Обратное дерево кратчайших путей* – такой подграф, являющийся обратным деревом, что расстояние от каждой вершины дерева до корня дерева по дереву и по графу одинаковы.
- *Остов графа, остовное дерево* – подграф, являющийся деревом и содержащий все вершины графа.

Далее граф G, лежащий в основе распределенной системы, предполагается ориентированным сильно-связным простым нумерованным упорядоченным корневым графом. В переменных вершин и параметрах сообщений описание

вершины – это ее номер, а описание дуги – это тройка чисел (a,i,b), где a – номер начала (начальной вершины) дуги, i – номер дуги в вершине a, b – номер конца (конечной вершины) дуги. Связный подграф будем задавать множеством таких троек чисел, представляющих все его дуги. В тривиальном случае, когда связный подграф не имеет дуг, он состоит из одной вершины, и будет задаваться синглетоном, содержащим номер этой вершины. Маршрут будет задаваться последовательностью дуг (троек чисел); маршрут нулевой длины – пустая последовательность <>.

Обозначения:

- r – корень графа.
- $d^-(a)$ – полустепень исхода вершины a (число исходящих из нее дуг).
- $d^+(a)$ – полустепень захода вершины a (число входящих в нее дуг).
- $a—i\rightarrow b$ – дуга, задаваемая тройкой (a,i,b).
- $a—i\rightarrow$ – означает, что для некоторого b существует дуга $a—i\rightarrow b$.
- $a\rightarrow b$ – означает, что для некоторого i существует дуга $a—i\rightarrow b$.
- $V(H)$ – множество вершин, инцидентных дугам множества дуг H.

4. Постановка задачи

Для того, чтобы в графе G с n вершинами из каждой вершины можно было попасть в каждую вершину по кратчайшему пути, нужно отображение $f(G) : [1..n]\times[1..n]\rightarrow[0..n\text{-}1]$, которое каждой паре вершины с номерами a и b ставит в соответствие номер дуги, с которой начинается кратчайший путь из a в b, если $a\neq b$, или 0 в противном случае. В каждой вершине $a\in[1..n]$ достаточно хранить часть этого отображения как отображение $f_a(G) : [1..n]\rightarrow[1..d^-(a)]$ такое, что $\forall b\in[1..n]\ f_a(G)(b) = f(G)(a,b)$. Если граф G подразумевается, мы будем вместо $f(G)$ и $f_a(G)$ писать просто f и f_a.

Размер памяти вершины a, необходимый для хранения такого отображения, равен $\boldsymbol{O}(nlogd^-(a)) = \boldsymbol{O}(nlogn)$, что на порядок меньше памяти для хранения всего неупорядоченного графа $\boldsymbol{O}(n^2)$ (матрица смежности $[1..n]\times[1..n]\rightarrow\{0,1\}$) и, тем более, упорядоченного графа $\boldsymbol{O}(n^2logn)$ (матрица смежности с указанием номеров дуг $[1..n]\times[1..n]\rightarrow[0..n]$) [6]. Поскольку отображение имеет размер $\boldsymbol{O}(nlogn)$, то в любом алгоритме размер памяти вершины $A = \boldsymbol{\Omega}(nlogn)$.

Мы ставим задачу: построить отображение f и разместить в каждой вершине a отображение f_a. В этой статье мы предлагаем четыре результата, графически изображенные на рис. 1 и описываемые в следующих разделах.

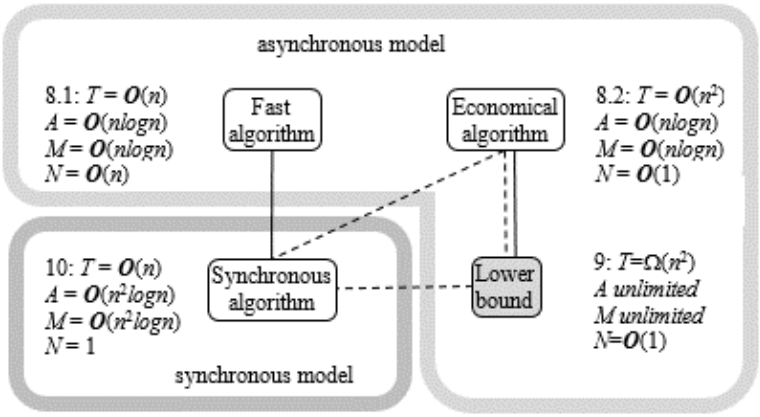


Рис. 1. Алгоритмы построения отображения и их оценки
Fig. 1. Algorithms for constructing a map and their estimates

5. Коррекция деревьев

5.1. Коррекция прямого дерева

Пусть все вершины графа H достижимы из вершины x, и имеется прямое дерево $D \subseteq H$ с корнем в x. Определим коррекцию дерева, когда становится известным о существовании дуги $a—i\to b \in H \setminus D$ с началом в D, т.е. $a \in V(D)$. Если $b \notin V(D)$, то дуга $a—i\to b$ добавляется в D: $D := D \setminus \{a\} \cup \{a—i\to b\}$. Если $b \in V(D)$, то в D есть единственная дуга $a`—i`\to b$, заканчивающаяся в b. Если по дереву D расстояние от x до a, увеличенное на 1, меньше, чем расстояние от x до b, то дуга $a—i\to b$ заменяет в D дугу $a`—i`\to b$: $D := D \cup \{a—i\to b\} \setminus \{a`—i`\to b\}$. Следующее утверждение очевидно, и мы приводим его без доказательства: если такая коррекция применяется последовательно для каждой дуги графа, то в итоге будет получено прямое остовное (для H) дерево кратчайших путей с корнем в x.

5.2. Коррекция обратного дерева

Пусть из всех вершин графа H достижима вершина x, и имеется обратное дерево $D \subseteq H$ с корнем в x. Определим коррекцию дерева, когда становится известным о существовании дуги $a—i\to b \in H \setminus D$ с концом в D, т.е. $b \in V(D)$. Если $a \notin V(D)$, то дуга $a—i\to b$ добавляется в D: $D := D \setminus \{b\} \cup \{a—i\to b\}$. Если $a \in V(D)$, то в D есть единственная дуга $a—i`\to b`$, начинающаяся в a. Если по дереву D расстояние от b до x, увеличенное на 1, меньше, чем расстояние от a до x, то дуга $a—i\to b$ заменяет в D дугу $a—i`\to b`$: $D := D \cup \{a—i\to b\} \setminus \{a—i`\to b`\}$. Следующее утверждение очевидно, и мы приводим его без доказательства: если такая коррекция применяется

последовательно для каждой дуги графа, то в итоге будет получено обратное остовное (для H) дерево кратчайших путей с корнем в x.

6. Способы передачи сообщений

В описываемых ниже алгоритмах используется пять типов передачи сообщений. Мы определим эти способы здесь, указывая параметры сообщения и переменные в вершинах, которые используются в этих способах передачи. В дальнейшем будем просто ссылаться на способ передачи того или иного сообщения, указывая также дополнительные параметры.

6.1. Рассылка из корня

Параметры сообщения: отсутствуют. Это сообщение должно пройти по каждой дуге графа ровно один раз. В каждой вершине x имеется булевская переменная $s(x)$; вначале $s(x) = \textbf{\textit{false}}$. Сообщение создается корнем r и посылается им по каждой исходящей из корня дуге графа. Когда вершина $x \neq r$ получает сообщение, она анализирует переменную $s(x)$. Если $s(x) = \textbf{\textit{false}}$, сообщение пересылается дальше по каждой исходящей из x дуге графа и $s(x) := \textbf{\textit{true}}$. Если $s(x) = \textbf{\textit{true}}$, ничего не делается. Заметим, что условие $s(x) = \textbf{\textit{false}}$ является единственным условием, которое должно быть выполнено в каждой вершине перед началом работы любого алгоритма; можно считать, что это часть начального состояния вершины.

6.2. Множественная рассылка

Параметры сообщения: номер вершины x, создавшей сообщение. Это сообщение распространяется по графу аналогично рассылке из корня, но только создателем сообщения может быть другая вершина, и таких вершин может быть несколько. В каждой вершине x имеется множество $S(x)$ вершин-создателей, от которых получено сообщение; вначале $S(x) = \varnothing$. Когда вершина x создаёт сообщение, $S(x) := S(x) \cup \{x\}$. Когда вершина $x \neq y$ получает сообщение, созданное вершиной y, она анализирует множество $S(x)$. Если $y \notin S(x)$, сообщение пересылается дальше по каждой исходящей из x дуге графа и $S(x) := S(x) \cup \{y\}$. Множество $S(x)$ можно задавать битовой шкалой длины n.

6.3. Пересылка по прямому дереву

Параметры сообщения: описание прямого дерева $F(x)$ с корнем в вершине x, номер вершины $y \in V(F(x))$. Сообщение передается по прямому дереву $F(x)$ от x до y. Когда вершина $z \neq y$ получает сообщение, она пересылает его дальше по исходящей из z дуге дерева $F(x)$, ведущей в y, т.е. первой дуге на пути от x до y в дереве $F(x)$.

6.4. Пересылка по обратному дереву

Параметры сообщения: описание обратного дерева $R(x)$ с корнем в вершине x. Сообщение передается по обратному дереву $R(x)$ до вершины x. Когда вершина $z \neq x$ получает сообщение, она пересылает его дальше по исходящей из z дуге дерева $R(x)$.

6.5. Сбор по обратному дереву

Параметры сообщения: описание обратного дерева $R(x)$ с корнем в вершине x, булевский признак q. В каждой вершине z имеются счетчик $c(z)$ и переменная $m(z)$, в которой может храниться сообщение; вначале $c(z) = 0$. Из каждой вершины $z \in V(R(x)) \backslash \{x\}$ будет послано одно сообщение по исходящей из z дуге дерева $R(x)$, но только после того, как вершина z получит сообщение по каждой входящей в z дуге дерева $R(x)$ и сама создаст сообщение. Для подсчета числа сообщений используется счетчик $c(z)$, который увеличивается на 1, когда вершина z получает сообщение или создает его сама. Полученное или созданное вершиной z сообщение запоминается в переменной $m(z)$, если там уже не запомнено сообщение с $q = \textbf{\textit{true}}$. Когда $c(z)$ на 1 больше числа дуг дерева $R(x)$, входящих в z, запомненное в $m(z)$ сообщение посылается по исходящей из z дуге дерева $R(x)$. В результате вершина x получит ровно одно сообщение по каждой входящей в x дуге дерева $R(x)$. Среди полученных вершиной x сообщений (включая сообщение, созданное вершиной x) будет сообщение с $q = \textbf{\textit{true}}$ тогда и только тогда, когда хотя бы одна вершина из $V(R(x))$ создала сообщение с $q = \textbf{\textit{true}}$. Такой способ передачи сообщений будем называть «сбором по обратному остову».

7. Алгоритмы построения в корне описаний прямого и обратного остовных деревьев кратчайших путей

В этом разделе предложены два алгоритма построения в корне описаний прямого F и обратного R остовных деревьев кратчайших путей, а в каждой вершине x – описания множества $In(x)$ входящих дуг. Оценки: в «быстром» алгоритме $T = \boldsymbol{O}(n)$, $N = \boldsymbol{O}(n)$, в «экономном» алгоритме $T = \boldsymbol{O}(n^2)$, $N = \boldsymbol{O}(1)$, остальные оценки совпадают: $A = \boldsymbol{O}(nlogn)$ и $M = \boldsymbol{O}(nlogn)$.

7.1. «Быстрый» алгоритм

Идея алгоритма заключается в следующем. Сначала рассылкой из корня по графу распространяется сообщение «Старт». Оно пройдет по каждой дуге ровно один раз, что требует времени не более n тактов. Это сообщение накапливает в себе описание пройденного им маршрута Fp. Когда сообщение приходит в вершину x, Fp запоминается в ней. Множество маршрутов, запомненных в x, образует подграф $H(x)$, состоящий из прямого дерева с корнем в r и множества $In(x)$ дуг, ведущих из его листьев в x. Поэтому число дуг в $H(x)$ ограничено по порядку n, что ограничивает размер описания $H(x)$

порядком *nlogn*. Получив «Старт» с параметром *Fp*, вершина создает сообщение «Возврат», которое распространяется по графу множественной рассылкой, что гарантирует доставку сообщения в корень. На одной дуге может оказаться $\boldsymbol{O}(n)$ сообщений «Возврат». «Возврат» содержит как параметр *Fp* (из сообщения «Старт»), и тоже накапливает в себе описание пройденного им маршрута *Rp*. Получая «Старт» с *Fp* (это цикл) или «Возврат» с *Fp* и *Rp* (вместе образуют цикл), корень формирует описание текущих прямого F и обратного R деревьев на множестве вершин $V = V(F) = V(R)$; размер описания тоже ограничен по порядку *nlogn*. Корень в цикле организует опрос вершин в старт-стопном режиме.

Сообщение «Опрос» передается рассылкой по дереву F, а корень запоминает число $w := |V|$ вершин, которые получат «Опрос». Получив «Опрос», вершина x создает сообщение «Ответ», в котором указывает $H(x)$ и полустепень исхода $d^-(x)$. «Ответ» передается пересылкой по дереву R. Поэтому на одной дуге может оказаться $\boldsymbol{O}(n)$ сообщений «Ответ». Корень, получая $H(x)$, образованный маршрутами от корня до $x \in V$, корректирует (см. п.5.1 и п.5.2) описания деревьев F и R (соответственно меняется V) так, чтобы это были деревья кратчайших путей. Корень запоминает $D^+(x) := |In(x)|$ и $D^-(x) := d^-(x)$. Когда корень получит «Ответ» от всех w вершин, проверяется условие завершения алгоритма: 1) полустепень исхода $d^-(x)$ запомнена в $D^-(x)$ для каждой $x \in V$, т.е. $D^-(x) > 0$, 2) суммарное число известных входящих дуг равно суммарному числу исходящх дуг $\Sigma_{x\in V} D^+(x) = \Sigma_{x\in V} D^-(x)$.

7.1.1. Начало работы

Корень r инициализирует $S(r) := \{r\}$, $F := \{r\}$, $R := \{r\}$, $D^-(r) := d^-(r)$, $D^+(r) := 0$, $In(r) := \varnothing$. Если нет дуг, исходящих из r, алгоритм заканчивается. В противном случае создается сообщение «Старт» и посылается по каждой исходящей из r дуге j с параметрами $(<>, r, j)$.

7.1.2. Сообщение «Старт», рассылка из корня

Параметры сообщения при передаче по дуге $a—i\to x$: a, i, Fp – маршрут, пройденный сообщением от r до a. Обозначим $Fp` := Fp\cdot<a—i\to x>$. Вершина $x \neq r$, получив «Старт» первый раз ($s(x) = \boldsymbol{false}$), инициализирует $S(x) := \{x\}$, $H(x) := Fp`$, $In(x) := \{a—i\to x\}$, пересылает «Старт» по каждой исходящей из x дуге j с параметрами $(Fp`, x, j)$, и создает сообщение «Возврат», которое тоже посылает по каждой исходящей из $x$ дуге $j$ с параметрами $(Fp`, <>, x, j)$. При повторном ($s(x) = \boldsymbol{true}$) получении «Старта» $H(x) := H(x) \cup Fp`$, $In(x) := In(x) \cup \{a—i\to x\}$, «Старт» дальше не посылается и «Возврат» не создается. Корень r, получая «Старт», увеличивает $D^+(r) := D^+(r)+1$, $In(r) := In(r) \cup \{a—i\to r\}$, для каждого $y \in V(\boldsymbol{Im}(Fp))\backslash V$ добавляет $D^+(y) := 0$ и $D^-(y) := 0$. Затем корректируется F (см. п.5.1) перебором дуг $Fp`$ от начала к концу, и корректируется $R$ (см. п.5.2) перебором дуг $Fp`$ от конца к началу.

После этого корень создает сообщение «Опрос» с параметрами (F, R) и посылает его по исходящим из корня дугам F, запоминая $w := |V|$.

7.1.3. Сообщение «Возврат», множественная рассылка

Параметры сообщения при передаче по дуге $a—i\rightarrow z$: a, i, Fp, Rp, где Fp –это маршрут, пройденный сообщением «Старт» от r до некоторой вершины x, которая создает сообщение «Возврат», Rp – это маршрут, пройденный сообщением «Возврат» от вершины x до вершины a. Обозначим $Rp` := Rp\cdot<a—i\rightarrow z>$. Когда «Возврат» приходит в вершину z, проверяется, является ли это сообщение первым или повторным от вершины x. Если $x\notin S(z)$, это первое сообщение. Тогда, если $z \neq r$, «Возврат» пересылается дальше по каждой исходящей из z дуге j с параметрами $(Fp, Rp`, z, j)$. Если $z = r$, корень для каждого $y\in V(\boldsymbol{Im}(Fp)\cup\boldsymbol{Im}(Rp))\backslash V$ добавляет $D^+(y) := 0$ и $D^-(y) :=0$, корректирует $F$ (см. п.5.1), перебирая дуги $Fp\cdot Rp`$ от начала к концу, и корректирует R (см. п.5.2), перебирая дуги $Fp\cdot Rp`$ от конца к началу. Повторный ($x\in S(z)$) «Возврат» игнорируется.

7.1.4. Сообщение «Опрос», рассылка по дереву F

Параметры сообщения: F, R, где F прямое, а R обратное остовные деревья кратчайших путей с корнем в r. Когда сообщение «Опрос» приходит в вершину x, оно пересылается дальше по всем исходящим из x дугам дерева F. Кроме того, вершина x создает сообщение «Ответ», которое посылается по исходящей из x дуге дерева R.

7.1.5. Сообщение «Ответ», пересылка по дереву R

Параметры сообщения при создании в вершине x: R, $H(x)$, $d^-(x)$. Когда вершина $z \neq r$ получает «Ответ», она пересылает его по исходящей из z дуге дерева R. Когда корень получает «Ответ», он запоминает $D^+(x) := |In(x)|$, $D^-(x) := d^-(x)$, для каждого $y\in V(H(x))\backslash V$ добавляет $D^+(y) := 0$ и $D^-(y) :=0$. Далее корректируется F (см. п.5.1) перебором дуг из $H(x)$ от корня к x, и корректируется R (см. п.5.2) перебором дуг из $H(x)$ от x к корню. Корень дожидается w «Ответов» и проверяет условие конца работы.

7.1.6. Конец алгоритма

Конец работы определяет корень при выполнении условия: 1) $\forall x\in V\, D^-(x) > 0$, 2) $\Sigma_{x\in V} D^+(x) = \Sigma_{x\in V} D^-(x)$. Если условие не выполнено, корень снова создает и рассылает сообщение «Опрос», запоминая $w = |V|$.

Покажем, что алгоритм заканчивается через конечное время (в тактах) и определим его оценки. За время $t_1 \le n$ сообщение «Старт» пройдет по каждой дуге, в том числе по каждой входящей дуге корня. После этого за время $t_2 \le n$-1 сообщение «Возврат» от каждой вершины дойдет до корня. Поэтому через время не более $t_1 + t_2$ будет $V = V(G)$. Сообщения «Опрос» проходят дерево F за время не более n-1, а сообщения «Ответ» проходят дерево R за время не более n-1. Поэтому цикл «Опрос»-«Ответ» длится не более $2(n$-1)

тактов. Поэтому через время не более $t_1 + t_2 + 2(n\text{-}1)$ начнется очередной цикл, в котором каждая вершина $x \neq r$ создает «Ответ» с $|In(x)| = d^+(x)$. После получения корнем всех «Ответов» на этот «Опрос» будет выполнено условие конца работы. Следовательно, время работы алгоритма $T \le t_1+t_2+2(n\text{-}1)+2(n\text{-}1) = \boldsymbol{O}(n)$. Из описания алгоритма следует, что $N = \boldsymbol{O}(n)$, $A = \boldsymbol{O}(n\log n)$, $M = \boldsymbol{O}(n\log n)$.

Покажем, что в конце работы алгоритма F и R остовные деревья кратчайших путей. В этот момент времени $V = V(G)$, т.е. деревья F и R остовные. Кроме того, для каждой вершины $x \in V$ в корне хранится $D^-(x) = d^-(x)$. Поэтому условие окончания алгоритма может быть выполнено только в том случае, когда $D^+(x) = d^+(x)$ для каждой вершины $x \in V$. Следовательно, для каждой дуги $y \text{—} i \to x$ корень получил такой «Ответ» от x, что $y \text{—} i \to x \in H(x)$. В этот момент времени $x \in V$, а в $H(x)$ существует путь, начинающийся в корне, последняя дуга которого – это дуга $y \text{—} i \to x$. Это значит, что каждая дуга графа участвует в коррекции деревьев F и R. Следовательно, по утверждениям в 5.1 и 5.2 F и R деревья кратчайших путей.

7.2. «Экономный» алгоритм

Идея этого алгоритма заключается в такой модификации «быстрого» алгоритма из п.7.1, чтобы уменьшить оценку N с $O(n)$ до $O(1)$ за счет увеличения времени работы не более чем в n раз. Причиной оценки $N = O(n)$ являются способы передачи сообщения «Возврат» – множественная рассылка, и сообщения «Ответ» – пересылка по дереву R. Сообщение «Возврат» мы удалим. Теперь у нас не будет времени t_2 и гарантии того, что корень узнает о всех вершинах графа через время $t_1 + t_2 = \boldsymbol{O}(n)$. Корень будет узнавать о вершинах только при получении «Старт» и «Ответ». «Ответ» на данный «Опрос» будет передаваться не пересылкой по дереву R, а сбором по дереву R. При создании «Ответа» вершиной x признак $q := (D^+(x) < |\, In(x)|)$ показывает, что множество $H(x)$ (и, тем самым, и множество $In(x)$) изменилось по сравнению с тем, которое уже попало в корень.

В результате при каждом опросе в корень придет только один «Ответ» по каждой входящей в корень дуге дерева R, причем признак q в нем будет истинен тогда и только тогда, когда в соответствующем поддереве дерева F хотя бы в одной вершине x изменилось $H(x)$. Заметим, что при ложном признаке q в «Ответе» от x не нужны параметры $H(x)$ и $d^-(x)$, но эта оптимизация не влияет на оценки алгоритма. За все время работы алгоритма от каждой вершины x может придти только один «Ответ» с истинным признаком q и $|In(x)| = d^+(x)$. Поэтому после времени t_1 будет не более n циклов «Опрос»-«Ответ», после чего условие конца работы будет выполнено.

7.2.1. Начало работы

Корень r инициализирует $F := \{r\}$, $R := \{r\}$, $D^-(r) := d^-(r)$, $D^+(r) := 0$, $In(r) := \varnothing$. Затем корень посылает «Старт» по каждой исходящей из r дуге j с параметрами $(<>, r, j)$, если такие дуги есть. Если дуг нет, то конец алгоритма.

7.2.2. Сообщение «Старт», рассылка из корня

Параметры сообщения при передаче по дуге $a—i\rightarrow x$: a, i, Fp – маршрут, пройденный сообщением от r до a. Обозначим $Fp` := Fp\cdot<a—i\rightarrow x>$. Вершина $x\neq r$, получив «Старт» первый раз ($s(x)$ = ***false***), инициализирует $H(x) := Fp`$, $In(x) := \{a—i\rightarrow x\}$, пересылает «Старт» по каждой исходящей из x дуге j с параметрами $(Fp`, x, j)$. При повторном ($s(x)$ = ***true***) получении «Старта» $H(x) := H(x) \cup Fp`$, $In(x) := In(x) \cup \{a—i\rightarrow x\}$, «Старт» дальше не посылается. Корень, получая «Старт», увеличивает $D^+(r) := D^+(r)+1$, $In(r) := In(r) \cup \{a—i\rightarrow r\}$, для каждого $y\in V(\mathbf{Im}(Fp))\backslash V$ добавляет $D^+(y) := 0$ и $D^-(y) :=0$, корректирует F (см. п.5.1), перебирая дуги $Fp`$ от начала к концу, и корректирует $R$ (см. п.5.2), перебирая дуги $Fp`$ от конца к началу. Корень сбрасывает счетчик $c(r) := 1$, создает сообщение «Опрос» с параметрами (F, R, D^+) и посылает его по исходящим из корня дугам F.

7.2.3. Сообщение «Опрос», рассылка по дереву F

Параметры сообщения: F, R, D^+. Когда сообщение «Опрос» приходит в вершину x, оно пересылается дальше по всем исходящим из x дугам дерева F. Также создается сообщение «Ответ» с $q = (D^+(x) = |In(x)|)$. Если x – листовая вершина R, «Ответ» посылается по исходящей из x дуге дерева R. Иначе «Ответ» запоминается в $m(x)$ и $c(x) := 1$.

7.2.4. Сообщение «Ответ», сбор по дереву R

Параметры сообщения при создании в вершине x: признак q, R, $H(x)$, $d^-(x)$. Когда вершина $z \neq r$ получает «Ответ», $c(z) := c(z)+1$. Если q = ***true***, вершина z запоминает «Ответ» в $m(z)$ вместо ранее запомненного «Ответа». Далее, если счетчик $c(z)$ на 1 больше числа дуг дерева R, входящих в вершину z, запомненный в $m(z)$ «Ответ» посылается по исходящей из z дуге дерева R. Когда корень получает сообщение, $c(r) := c(r)+1$. Если признак q = ***true***, корень запоминает $D^+(x) := |In(x)|$, $D^-(x) := d^-(x)$, для каждого $y\in V(H(x))\backslash V$ добавляет $D^+(y) := 0$ и $D^-(y) := 0$. Далее корректируется F (см. п.5.1) перебором дуг из $H(x)$ от корня к x, и корректируется R (см. п.5.2) перебором дуг из $H(x)$ от x к корню. Если $c(r)$ меньше или равен числа дуг дерева R, входящих в корень, корень продолжает ждать «Ответы», иначе проверяет условие конца работы.

7.2.5. Конец алгоритма

Конец работы определяет корень при выполнении условия: 1) $\forall x\in V\, D^-(x) > 0$, 2) $\Sigma_{x\in V} D^+(x) = \Sigma_{x\in V} D^-(x)$. Если условие не выполнено, корень снова создает и рассылает сообщение «Опрос», сбрасывая счетчик $c(r) := 1$.

Покажем, что алгоритм заканчивается через конечное время и определим его оценки. За время $t_1 \leq n$ сообщение «Старт» пройдет по каждой дуге, в том числе по каждой входящей дуге корня. От каждой вершины x сообщение «Ответ» с q = ***true*** и $|In(x)| = d^+(x)$ может придти в корень не более одного раза. Поэтому таких сообщений «Ответ» может придти в корень не более n. Сообщения «Опрос» проходят дерево F за время не более n-1, а сообщения

«Ответ» проходят дерево R за время не более n-1. Поэтому цикл «Опрос»-«Ответ» длится не более 2(n-1) тактов. Первый цикл «Опрос»-«Ответ», начинающийся после времени t_1, начнется не позже времени t_1+2(n-1), а всего циклов, начинающихся после времени t_1, будет не более n. Следовательно, время работы алгоритма $T \le t_1+2(n-1)+n(2(n-1)) = \boldsymbol{O}(n^2)$. Из описания алгоритма следует, что на одной дуге может находиться либо не более одного сообщения «Старт», либо не более одного сообщения «Опрос» и (если это дуга из $F \cap R$) не более одного сообщения «Ответ». Поэтому $N \le 2 = \boldsymbol{O}(1)$. Также из описания алгоритма следует, что $A = \boldsymbol{O}(n\log n)$, $M = \boldsymbol{O}(n\log n)$.

Покажем, что в конце работы алгоритма F и R остовные деревья кратчайших путей. По условию конца работы для каждой вершины $x \in V$ будет $D^-(x) = d^-(x)$, и $\Sigma_{x\in V} D^+(x) = \Sigma_{x\in V} D^-(x)$. Из описания алгоритма следует, что для каждой дуги $y \text{—} i \rightarrow x$, учтенной в $D^+(x)$, $y \in V$. Поэтому условие конца работы может быть выполнено только в том случае, когда $D^+(x) = d^+(x)$ и $V = V(G)$. Тем самым, в конце работы алгоритма деревья F и R остовные. Кроме того, для каждой дуги $y \text{—} i \rightarrow x$ корень получит такой «Ответ» от x, что $q = \textbf{\textit{true}}$ и $y \text{—} i \rightarrow x \in H(x)$. В этот момент времени $x \in V$, а в $H(x)$ существует путь, начинающийся в корне, последняя дуга которого – это дуга $y \text{—} i \rightarrow x$. Это значит, что каждая дуга графа участвует в коррекции деревьев F и R. Следовательно, по утверждениям в 5.1 и 5.2 F и R деревья кратчайших путей.

8. Алгоритмы построения отображения

Мы рассмотрим два алгоритма построения отображения f, которое для каждой вершины a и каждой другой вершины b ставит в соответствие номер дуги, с которой начинается кратчайший путь из a в b. У этих алгоритмов одинаковые оценки $A = \boldsymbol{O}(n\log n)$ и $M = \boldsymbol{O}(n\log n)$, но разные оценки T и N: в «быстром» алгоритме оценки $T = \boldsymbol{O}(n)$ и $N = \boldsymbol{O}(n)$, в «экономном» алгоритме оценки $T = \boldsymbol{O}(n^2)$ и $N = \boldsymbol{O}(1)$.

В каждом из этих алгоритмов размер сообщения $M = \boldsymbol{O}(n\log n)$. Легко модифицировать алгоритмы, чтобы уменьшить оценку M в n раз до $\boldsymbol{O}(\log n)$, увеличив в n раз оценку N. Для этого достаточно при передаче сообщения размером $\boldsymbol{O}(n\log n)$ разбивать его на серию из n мини-сообщений размером $\boldsymbol{O}(\log n)$ с пометкой последнего мини-сообщения в серии. Эти мини-сообщения посылаются по дуге в одном срабатывании автомата подряд от первого до последнего. Вершина принимает мини-сообщения и сохраняет их в своей памяти, пока не получит последнее в серии мини-сообщение. Аналогично можно уменьшить оценку M до $\boldsymbol{O}(1)$, увеличив в $n\log n$ раз оценку N.

Идея алгоритмов заключается в следующем. Сначала применяется один из двух алгоритмов построения в корне описаний прямого F и обратного R деревьев кратчайших путей, а в каждой вершине x – описания $In(x)$ множества входящих дуг. Затем корень организует доставку в каждую вершину x описания деревьев F и R, а из каждой вершины y в каждую вершину x

описания множества входящих дуг $In(y)$. По F и R вершина x строит описание прямого $F(x)$ и обратного $R(x)$ остовов с корнем в x. Получая от вершины y множество $In(y)$, вершина x для каждой дуги из $In(y)$ корректирует деревья $F(x)$ и $R(x)$. В конечном итоге эти деревья станут деревьями кратчайших путей с корнем в x, что позволяет вершине x создать требуемое отображение f_x.

8.1. «Быстрый» алгоритм

Применяется «быстрый» алгоритм построения в корне описаний деревьев F и R (см. п.7.1). Доставка из корня в каждую вершину описания деревьев F и R, а из каждой вершины в каждую вершину описания множества входящих дуг выполняются параллельно.

8.1.1. Начало работы

После выполнения «быстрого» алгоритма построения в корне описаний деревьев корень r создает сообщение «Остовы» с параметрами (F, R, $F(r) = F$, $In(r)$), которое посылает по каждой исходящей из корня дуге дерева F. Также корень инициализирует счетчик $ce := 0$, показывающий число сообщений «Конец», полученных корнем. Предполагается, что в каждой вершине $x \neq r$ имеется счетчик $ct(x)$ числа сообщений «Остовы», полученных вершиной x. Вначале $ct(x) = 0$.

8.1.2. Сообщение «Остовы», рассылка по дереву F(x)

Параметры сообщения, созданного вершиной x: F, R, $F(x)$, $In(x)$. Вершина z, получив «Остовы», пересылает его дальше по каждой исходящей из z дуге дерева $F(x)$, и увеличивает счетчик $ct(z) := ct(z)+1$. Далее проверяется, первое ли это сообщение «Остовы», полученное вершиной z. Если $ct(z) = 1$, вершина z строит по F и R прямое остовное дерево $F(z)$ с корнем в z, корректирует его по каждой дуге из $In(z) \cup In(x)$, создает сообщение «Остовы» с параметрами (F, R, $F(z)$, $In(z)$), и посылает его по каждой исходящей из z дуге дерева $F(z)$. Если $ct(z) > 1$, вершина z только корректирует $F(z)$ по каждой дуге из $In(x)$. В любом случае вершина z проверяет, получила ли она «Остовы» от всех вершин $x \neq z$. Если $ct(z) = |V|-1$, вершина z по дереву $F(z)$ строит требуемое отображение f_z, создает сообщение «Конец» с параметрами (R, z), которое посылает по исходящей из z дуге дерева R.

8.1.3. Сообщение «Конец», пересылка по дереву R

Параметры сообщения, созданного вершиной x: R, x. Вершина $z \neq r$, получив «Конец», пересылает его дальше по исходящей из z дуге дерева R. Корень, получая сообщение «Конец», увеличивает счетчик $ce := ce+1$. Далее проверяется условие конца работы.

8.1.4. Конец алгоритма

Конец работы алгоритма проверяется корнем по условию: $ce = |V|-1$.

Покажем, что алгоритм заканчивается через конечное время и определим его оценки. «Быстрый» алгоритм построения в корне описаний деревьев имеет оценки $T = \boldsymbol{O}(n)$, $A = \boldsymbol{O}(n\log n)$, $M = \boldsymbol{O}(n\log n)$, $N = \boldsymbol{O}(n)$. После этого сообщение «Остовы» рассылкой по прямому дереву F за время $t_1 \le n\text{-}1$ дойдет от корня до каждой вершины. Следовательно, каждая вершина x получит первое сообщение «Остовы» за время не более t_1, и создаст свое сообщение «Остовы». После этого от каждой вершины x до каждой другой вершины сообщение «Остовы» рассылкой по прямому дереву $F(x)$ дойдет за время не более $t_2 \le n\text{-}1$. После этого от каждой вершины до корня сообщение «Конец» пересылкой по обратному дереву R дойдет за время не более $t_3 \le n\text{-}1$. Тем самым, алгоритм закончится через время $T = \boldsymbol{O}(n)$. Из описания алгоритма следует, что $A = \boldsymbol{O}(n\log n)$, $M = \boldsymbol{O}(n\log n)$. Каждая вершина создает только одно сообщение «Остовы» и только одно сообщение «Конец». Поэтому $N = \boldsymbol{O}(n)$.

Покажем, что в конце работы алгоритма в каждой вершине z будет создано требуемое отображение f_z. Для этого достаточно показать, что прямое дерево $F(z)$ является деревом кратчайших путей. При получении первого сообщения «Остовы», созданного вершиной x, вершина z получает прямой F и обратный R остовы. По ним она строит прямой остов $F(z)$ с корнем в z, который корректируется по дугам из $In(z) \cup In(x)$. Далее до конца работы вершина z получит «Остовы» от каждой вершины $y \ne z$ и $y \ne x$ с параметром $In(y)$. Тем самым, вершина z получит описание всех дуг графа. Поскольку $F(z)$ остов, его коррекция по любой дуге графа всегда выполнима. Следовательно, в конце работы $F(z)$ является деревом кратчайших путей.

8.2. «Экономный» алгоритм

Применяется «экономный» алгоритм построения в корне описания деревьев F и R (см. п.7.2). Корень организует цикл по всем вершинам. Для каждой вершины x выполняется доставка из корня в x описания деревьев F и R, а из x в каждую вершину описания множества входящих дуг $In(x)$.

8.2.1. Начало работы

После выполнения «экономного» алгоритма построения в корне описания деревьев корень проверяет $|V| = 1$. Если корень единственная вершина, конец алгоритма. Иначе корень r выбирает любую вершину $x \ne r$, инициализирует множество вершин $SV := \{r, x\}$, создает сообщение «Остовы» с параметрами $(x, F, R, In(r))$, которое посылает по исходящей из корня дуге дерева F, ведущей в x. Также корень инициализирует счетчик $ce := 0$, показывающий число сообщений «Конец», полученных корнем. Предполагается, что в каждой вершине $x \ne r$ имеется счетчик $ci(x)$ числа сообщений «Дуги», полученных вершиной x, учитывающий также одно сообщение «Остовы» от корня, если вершина x его получала. Вначале $ci(x) = 0$.

8.2.2. Сообщение «Остовы», пересылка по дереву F

Параметры сообщения: x, F, R, $In(r)$. Вершина $z \neq x$, получив «Остовы», пересылает его дальше по исходящей из z дуге дерева F, ведущей в x. Когда сообщение «Остовы» получит вершина x, она увеличивает счетчик $ci(x) := ci(x)+1$, после чего проверяет его. Если $ci(x) = 1$, вершина x строит по F и R прямое остовное дерево $F(x)$ с корнем в x, и корректирует его по каждой дуге из $In(x) \cup In(r)$. Если $ci(x) > 1$, вершина x только корректирует $F(x)$ по каждой дуге из $In(r)$. В любом случае, если $ci(x) = |V|-1$, вершина x по дереву $F(x)$ строит требуемое отображение f_x. В любом случае вершина x создает сообщение «Дуги» с параметрами ($F(x)$, R, $In(x)$), и посылает его по каждой исходящей из x дуге дерева $F(x)$.

8.2.3. Сообщение «Дуги», рассылка по прямому дереву F(x)

Параметры сообщения, созданного вершиной x: $F(x)$, R, $In(x)$. Вершина z, получив «Дуги», пересылает его дальше по каждой исходящей из z дуге дерева $F(x)$, увеличивает счетчик $ci(z) := ci(z)+1$, после чего проверяет его. Если $ci(z) = 1$, вершина z строит по F и R прямое остовное дерево $F(z)$ с корнем в z, и корректирует его по каждой дуге из $In(z) \cup In(x)$. Если $ci(z) > 1$, вершина z только корректирует $F(z)$ по каждой дуге из $In(x)$. В любом случае, если $ci(z) = |V|-1$, вершина z по дереву $F(z)$ строит требуемое отображение f_z. В любом случае, вершина z создает сообщение «Конец» с параметрами (R, q = ***true***), которое передается сбором по обратному дереву R: если z листовая вершина дерева R, сообщение посылается по исходящей из z дуге дерева R, иначе сообщение не посылается до тех пор, пока вершина z не получит сообщения «Конец» по всем входящим в z дугам дерева R.

8.2.4. Сообщение «Конец», сбор по обратному дереву R

Параметры сообщения: R, q = ***true***. Вершина $z \neq r$, получив «Конец», пересылает его дальше сбором по обратному дереву R: если z листовая вершина дерева R, сообщение посылается по исходящей из z дуге дерева R, иначе сообщение не посылается до тех пор, пока вершина z не получит сообщения «Конец» по всем входящим в z дугам дерева R. Корень, получая сообщение «Конец», увеличивает счетчик $ce := ce+1$. Если $ce = |In(r)|$ – число входящих в корень дуг дерева R, проверяется условие конца работы.

8.2.5. Конец алгоритма

Конец работы алгоритма проверяется корнем по условию: $SV = V$. Если условие не выполнено, корень выбирает любую вершину $x \in V \setminus SV$, добавляет ее во множество вершин $SV := SV \cup \{x\}$, сбрасывает счетчик $ce := 0$, создает сообщение «Остовы» с параметрами (x, F, R, $In(r)$), и посылает его по исходящей из корня дуге дерева F, ведущей в x.

Покажем, что алгоритм заканчивается через конечное время и определим его оценки. «Экономный» алгоритм построения в корне описаний деревьев имеет оценки $T = \boldsymbol{O}(n^2)$, $A = \boldsymbol{O}(nlogn)$, $M = \boldsymbol{O}(nlogn)$, $N = \boldsymbol{O}(1)$. После этого корень

перебирает все вершины $x \neq r$ в цикле, число проходов цикла равно n-1. На каждом проходе цикла сообщение «Остовы» пересылкой по прямому дереву F за время $t_1 \leq n$-1 дойдет от корня до выбранной вершины x. После этого от вершины x до каждой другой вершины сообщение «Дуги» рассылкой по прямому дереву $F(x)$ дойдет за время не более $t_2 \leq n$-1. После этого от каждой вершины до корня сообщение «Конец» сбором по обратному дереву R дойдет за время не более $t_3 \leq n$-1. Тем самым, один проход цикла требует времени $t_1 + t_2 + t_3 \leq 3(n$-1), и алгоритм закончится через время $T = \boldsymbol{O}(n^2)$. Из описания алгоритма следует, что $A = \boldsymbol{O}(n\log n)$, $M = \boldsymbol{O}(n\log n)$, $N = \boldsymbol{O}(1)$.

Покажем, что в конце работы алгоритма в каждой вершине z будет создано требуемое отображение f_z. Для этого достаточно показать, что прямое дерево $F(z)$ является деревом кратчайших путей. При получении вершиной z первого сообщения «Остовы» от корня или «Дуги» от вершины x вершина z получает прямой F и обратный R остовы. По ним она строит прямой остов $F(z)$ с корнем в z, который корректируется по дугам из $In(z) \cup In(r)$ или $In(z) \cup In(x)$. Далее до конца работы вершина z получит «Дуги» от каждой вершины $y \neq z$ и $y \neq r$ с параметром $In(y)$. Тем самым, вершина z получит описание всех дуг графа. Поскольку $F(z)$ остов, его коррекция по любой дуге графа всегда выполнима. Следовательно, в конце работы $F(z)$ является деревом кратчайших путей.

9. Нижние оценки сложности

Для «быстрого» алгоритма построения отображения оценка $T = \boldsymbol{O}(n)$ не может быть улучшена, поскольку, очевидно, ограничена снизу временем распространения сообщения от одной вершины до другой, т.е. диаметром графа, который может достигать величины n-1. В этом разделе мы докажем, что для «экономного» алгоритма построения отображения оценка $T = \boldsymbol{O}(n^2)$ также не может быть улучшена: для алгоритмов построения отображения с оценками $A = unlimited$, $M = unlimited$, $N = \boldsymbol{O}(1)$ имеет место $T = \Omega(n^2)$.

Нам достаточно доказать, что алгоритм, который строит требуемое отображение f_r только в корне r с оценкой $N = \boldsymbol{O}(1)$, имеет оценку $T = \Omega(n^2)$. Произвольный алгоритм будем называть *корневым*, если на любом графе в конце его работы в памяти корня r построено отображение f_r. Оценка $N = \boldsymbol{O}(1)$ означает, что существует такая константа k, что $N \leq k$; ее будем считать емкостью дуги.

До сих пор мы говорили о том, что сообщение проходит маршрут «без задержек»: когда вершина принимает сообщение по дуге маршрута, она сразу же посылает сообщение по следующей дуге маршрута, т.е. не дожидаясь каких-либо еще сообщений. Теперь такой маршрут будем называть *главным маршрутом* сообщения, но нам понадобится более общее понятие маршрута, проходимого сообщением, возможно, с какими-то задержками. Для любого алгоритма будем говорить, что в момент времени t *сообщение прошло маршрут* как последовательность смежных дуг $P = x_1 \to x_2 \to \ldots \to x_{p-1} \to x_p$, если

существует неубывающая последовательность моментов времени $t_1 \le t`_2 \le t_2 \le ... \le t`_{p-1} \le t_p \le t`_p = t$ такая, что для $i = 1..p\text{-}1$ вершина $x_i$ послала сообщение по дуге $x_i \to x_{i+1}$ в момент времени $t_i$, а вершина $x_{i+1}$ получила это сообщение в момент времени $t`_{i+1}$. Из этого определения следует, что в общем случае сообщение в момент времени t может пройти несколько маршрутов, заканчивающихся в разных вершинах, и в одной вершине могут заканчиваться несколько из этих маршрутов. Это объясняется тем, что сообщения могут «размножаться», когда вершина, получив сообщение, посылает сообщения по нескольким исходящим дугам, и «склеиваться», когда вершина не посылает сообщение, пока не получит несколько сообщений. Будем говорить, что *вершина получает сообщение от вершины* x, если она получает сообщение, прошедшее маршрут, начинающийся в x. Будем говорить, что алгоритм *покрывает дуги графа*, если для каждой дуги графа при работе алгоритма есть сообщение, которое проходит маршрут, проходящий по этой дуге и заканчивающийся в корне.

Утверждение 1. Если алгоритм корневой, то он покрывает дуги графа.

Доказательство: Допустим противное. Пусть $\mathcal{A}$ корневой алгоритм, а G – граф, в котором есть дуга $a—j\to$, и при работе алгоритма $\mathcal{A}$ нет сообщения, которое проходит маршрут, проходящий по этой дуге и заканчивающийся в корне. До конца работы алгоритма $\mathcal{A}$ корень получал сообщения, маршруты которых – это маршруты, заканчивающиеся в корне. Поэтому корень может определить конец работы алгоритма $\mathcal{A}$, только на основании этих маршрутов, включая номера дуг и номера вершин, через которые проходили эти маршруты, а также полустепени исхода этих вершин. Рассмотрим граф $G`$, который отличается от графа $G$ тем, что добавлена новая вершина $b$ и новая дуга $b—1\to a$, а дуга $a—j\to$ перенаправлена в вершину $b$, т.е. это дуга $a—j\to b$. Тогда должно быть: $b \notin \boldsymbol{Im}(f_r(G))$ и $b \in \boldsymbol{Im}(f_r(G`))$. На обоих графах в алгоритме $\mathcal{A}$ корень r получит одинаковые маршруты и, следовательно, будут построены одинаковые отображения в корне, чего быть не может, если алгоритм $\mathcal{A}$ корневой. Утверждение доказано.

В алгоритме, не нарушающем емкость дуги k, вершина может посылать сообщение по дуге только в том случае, когда на этой дуге гарантированно меньше k сообщений. При произвольном времени прохождения сообщений по дугам такую гарантию может дать только получение сообщения-подтверждения, прошедшего по этой дуге и далее по циклу вернувшегося в начало дуги. Если вершина послала по дуге несколько сообщений, подтверждать нужно каждую из таких пересылок. Проблема в том, что сообщения могут «размножаться» и «склеиваться».

Опишем формальную модель подтверждения пересылок. Для этого модифицируем алгоритм так, чтобы вершина нумеровала пересылки сообщений по исходящим дугам и учитывала возникающие циклы передачи сообщений. Пересылка задается тройкой (v,j,h), где v – номер вершины, j –

номер исходящей из нее дуги, h – номер пересылки из этой вершины. В каждое сообщение m добавляется множество $A(m)$ всех пересылок этого сообщения. В каждую вершину с номером v добавляется максимальный номер $h_{max}(v)$ пересылки из этой вершины, множество $B(v)$ всех неподтвержденных пересылок из этой вершины, а также множество $A(v)$ пересылок из сообщений, принятых этой вершиной после отправки ею последнего сообщения. Вначале $h_{max}(v) = 0$, $B(v) = \varnothing$, $A(v) = \varnothing$. Можно считать, что в корне эти переменные инициализируются в начале работы, а в любой другой вершине при получении первого сообщения. Когда вершина с номером v посылает сообщение m по дуге с номером j, она увеличивает номер неподтвержденной пересылки: $h_{max}(v) := h_{max}(v)+1$, запоминает эту пересылку как неподтвержденную $B(v) := B(v) \cup \{ (v, j, h_{max}(v)) \}$, добавляет ее в сообщение $A(m) := A(v) \cup \{ (v, j, h_{max}(v)) \}$, затем посылает сообщение m по дуге j и «забывает» пересылки из принятых сообщений $A(v) := \varnothing$. Когда вершина с номером v получает сообщение m, она подтверждает неподтвержденные пересылки, имеющиеся в сообщении, $B(v) := B(v) \setminus A(m)$ и запоминает пересылки из сообщения $A(v) := A(v) \cup A(m)$. Будем говорить, что сообщение m является *подтверждением* в вершине с номером v для исходящей дуги с номером j, если при получении этого сообщения множество $\{ (v,j,h) \in B(v) \mid h = 1.. \}$ неподтвержденных пересылок по этой дуге уменьшается. Алгоритм не нарушает емкость k дуги с номером j, исходящей из вершины с номером v, тогда и только тогда, когда вершина посылает сообщение по этой дуге только при условии, что число неподтвержденных пересылок по этой дуге меньше емкости дуги: $|\{ (v,j,h) \in B(v) \mid h = 1.. \}| < k$. Будем говорить, что *алгоритм следует правилу подтверждения для емкости дуги* k, если вершина, послав по дуге p сообщений и получив q подтверждений, посылает следующее сообщение по этой дуге только при условии $p-q < k$. Отсюда непосредственно следует следующее утверждение.

Утверждение 2. При произвольном времени прохождения сообщений по дугам алгоритм не нарушает емкость дуги k тогда и только тогда, когда алгоритм следует правилу подтверждения для емкости дуги k.

Рассмотрим класс графов $\boldsymbol{G} = \{ G_n \mid n \ge 2 \}$ на рис. 2 со всеми возможными способами нумерации вершин и дуг. Числа, записанные в вершинах на рис. 2 будем называть *индексами*. Когда мы будем говорить «вершина i», мы будем иметь в виду «вершина с индексом i». Если $n\text{-}1 \ge j > i \ge 1$, то будем говорить, что вершина i (с меньшим индексом) расположена правее вершины j, а вершина j (с большим индексом) – левее вершины i. Когда вершина $i = 1..n\text{-}2$ получает сообщение по дуге $i+1 \to i$, будем говорить, что она получает сообщение слева. Обозначим: $v(i)$ – номер вершины i, $a(i)$ – номер дуги $0 \to i$.

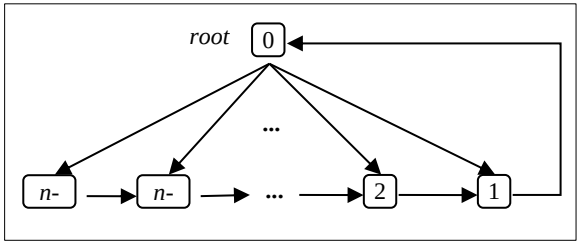


Рис. 2. Пример графа
Fig. 2. Example graph

Из утверждения 1 следует, что нам достаточно доказать, что алгоритм, не нарушающий емкости дуги k и покрывающий все дуги графа на классе $\boldsymbol{G}$ с емкостью дуг k, имеет оценку $T = \Omega(n^2)$. Из утверждения 2 следует, что нам достаточно рассматривать алгоритмы, которые следуют правилу подтверждения для емкости дуги k. Для оценки времени работы алгоритма будем рассматривать только случай, когда время пересылки единичное: каждое сообщение по каждой дуге пересылается за 1 такт. Итак, нам достаточно доказать, что алгоритм, следующий правилу подтверждения для емкости дуги k и покрывающий все дуги графа на классе $\boldsymbol{G}$ с емкостью дуг k и единичным временем пересылки, имеет оценку $T = \Omega(n^2)$. Далее по умолчанию мы будем рассматривать только такие алгоритмы. Через $t(n)$ будем обозначать время работы алгоритма.

Из устройства графа G_n следует, что подтверждением в вершине $i \neq 0$ является сообщение, которое прошло цикл, т.е. маршрут $i \rightarrow i\text{-}1 \rightarrow ... \rightarrow 1 \rightarrow 0 \rightarrow [... \rightarrow 0 \rightarrow] i$, где в квадратные скобки заключена часть маршрута, которая может отсутствовать. Сообщение будем называть $\{i\}$-сообщением, если $i \neq 0$ – это самая левая из вершин, через которые проходило сообщение. Очевидно, что вершина i получает $\{i\}$-сообщение по дуге $0 \rightarrow i$, т.е. от корня.

Будем говорить, что в вершине $i \neq 0$ происходит *склейка* $\{j`\}$-сообщения с $\{j\}$-сообщением, где $j` > j = i$, если первое сообщение, которое посылает вершина i, это $\{j`\}$-сообщение. Это происходит либо из-за того, что ранее полученное вершиной $\{i\}$-сообщение ожидает в вершине $\{j`\}$-сообщения, либо из-за того, что ранее полученное $\{j`\}$-сообщение ожидает в вершине $\{i\}$-сообщения, либо из-за того, что вершина, не получив $\{i\}$-сообщения, получает $\{j`\}$-сообщение и посылает его дальше. Во всех случаях будет послано $\{j`\}$-сообщение, с которым «склеено» $\{i\}$-сообщение.

Будем говорить, что в вершине $i \neq 0$ происходит *склейка* $\{j`\}$-сообщения с $\{j\}$-сообщением, где $j` > j > i$, если эта вершина посылает $\{j`\}$-сообщение тогда, когда она еще не послала $\{j\}$-сообщение. Это происходит из-за того, что ранее полученное $\{j\}$-сообщение ожидает в вершине $i$ $\{j`\}$-сообщения. Будет послано $\{j`\}$-сообщение, с которым «склеено» $\{j\}$-сообщение.

Утверждение 3. Для любого алгоритма можно так подобрать нумерацию дуг и вершин графа G_n, чтобы ни в какой вершине $i \neq 0$ не было склейки сообщений до получения этой вершиной первого подтверждения.

Доказательство: Рассмотрим работу алгоритма на графе G_n при некоторой нумерации вершин и дуг. Выберем самую левую вершину, в которой происходит склейка до получения этой вершиной первого подтверждения, и самую первую по времени склейку. Пусть это будет вершина $i \neq 0$, и она склеивает $\{j`\}$-сообщение с $\{j\}$-сообщением, где $j` > j$, в момент времени t.

Заметим, что любое сообщение, получаемое вершиной i слева до момента склейки в вершине i, не может быть уже склеено в некоторой вершине $x > i$. Действительно, поскольку мы выбрали самую левую вершину i, где происходит склейка до получения этой вершиной первого подтверждения, склейка в вершине x должна была происходить после получения вершиной x первого подтверждения. Но в этом случае склеенное сообщение, посланное вершиной x и дошедшее до вершины i, является подтверждением и для вершины i, а этого не может быть по правилу выбора вершины i.

Вершина i не может получить слева больше k сообщений до получения вершиной i+1 первого подтверждения. Но если вершина i+1 получает такое подтверждение, а потом посылает сообщение, то это сообщение будет подтверждением и для вершины i. Тем самым, вершина i не может получить слева больше k сообщений до получения вершиной i первого подтверждения.

Если бы было $j` > j+1$, то для некоторого $x$ такого, что $j` > x > j$, либо $\{j`\}$-сообщение склеивалось бы с $\{x\}$-сообщением в вершине левее $i$, либо $\{x\}$-сообщение склеивалось бы с $\{j\}$-сообщением в вершине $i$ до получения ею $\{j`\}$-сообщения. Поэтому, поскольку каждое сообщение, которое вершина i получает слева, не склеено ни с каким сообщением, и мы выбрали самую первую по времени склейку в вершине i, должно быть $j` = j+1$.

Итак, мы рассматриваем склейку $\{j+1\}$-сообщения с $\{j\}$-сообщением в вершине i. А поскольку мы выбрали первую по времени склейку, очевидно, j минимально. Поэтому возможны три типа склейки.

1) $j = i$, вершина i, еще не получив $\{i\}$-сообщение, получает $\{i+1\}$-сообщение.
2) $j = i$, вершина i сначала получает $\{i\}$-сообщение, но не посылает его дальше, а потом получает $\{i+1\}$-сообщение.
3) $j > i$, вершина i сначала посылает последовательно $\{x\}$-сообщения, где $x = i..j-1$, потом получает $\{j\}$-сообщение, но не посылает его дальше, а потом получает $\{j+1\}$-сообщение.

Тип 1 (рис. 3). Вершина i, еще не получив $\{i\}$-сообщение, получает $\{i+1\}$-сообщение. Тогда вершина i+1 получала $\{i+1\}$-сообщение (от корня) и посылала его дальше. Поскольку время пересылки единичное, корень посылал сообщение по дуге $0—a(i+1)\rightarrow i+1$, но не посылал сообщение по дуге $0—a(i)\rightarrow i$ или послал его позже (иначе оно пришло бы в вершину i раньше).

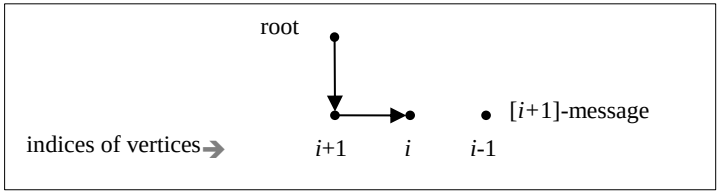


Рис. 3. Склейка сообщений типа 1
Fig. 3. Gluing messages of type 1

Рассмотрим момент времени $t_{0,i+1}$, когда корень посылает сообщение по дуге $0—a(i+1)\rightarrow i+1$. К этому моменту времени корень мог получить сообщения от вершин самого правого отрезка в диапазоне индексов $[1..s_{i+1}]$ (если $s_{i+1} = 0$, диапазон пустой). Должно быть $s_{i+1} < i$, поскольку в противном случае $\{i+1\}$-сообщение будет подтверждением в вершине i, т.е. к моменту склейки вершина i получит подтверждение. Поэтому, учитывая, что на классе $\boldsymbol{G}$ у всех вершин, кроме корня, полустепень исхода равна 1, можно считать, что корню в момент времени $t_{0,i+1}$ известная следующая информация:

$I_{0,i+1}=(v(0), \dot{d}(0), v(1) , a(1),..., v(s_{i+1}), a(s_{i+1}))$.

Поскольку $s_{i+1} < i$, если мы поменяем местами номера дуг $a(i)\leftrightarrow a(i+1)$, то информация $I_{0,i+1}$ не изменится, но сообщение от корня будет приходить не в вершину $i+1$, а в вершину i. Поэтому вершина i сначала получит $\{i\}$-сообщение, а не сообщение слева. После такого изменения нумерации вершин и дуг в вершине i не будет склейки типа 1.

Тип 2 (рис. 4). Вершина i сначала получает $\{i\}$-сообщение, но не посылает его дальше, а потом получает $\{i+1\}$-сообщение. Тогда вершина $i+1$ получала $\{i+1\}$-сообщение и посылала его дальше.

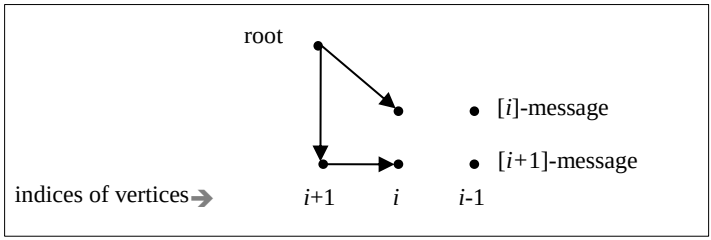


Рис. 4. Склейка сообщений типа 2
Fig. 4. Gluing messages of type 2

Рассмотрим моменты времени $t_{0,i+1}$ ($t_{0,i}$), когда корень посылает сообщение по дуге $0—a(i+1)\rightarrow i+1$ ($0—a(i)\rightarrow i$). К моменту времени $t_{0,i+1}$ ($t_{0,i}$) корень мог получить сообщения от вершин самого правого отрезка в диапазоне индексов $[1..s_{i+1}]$ ($[1..s_i]$). Должно быть $s_{i+1} < i$ и $s_i < i$, поскольку в противном случае $\{i+1\}$-сообщение или $\{i\}$-сообщение будет подтверждением в вершине i, т.е. к

моменту склейки вершина i получит подтверждение. В моменты времени $t_{0,i+1}$ и $t_{0,i}$ корню известна следующая информация:

в момент времени $t_{0,i+1}$: $I_{0,i+1}$=($v(0), d^-(0), v(1), a(1), ..., v(s_{i+1}), a(s_{i+1})$),

в момент времени $t_{0,i}$: $I_{0,i}$ =($v(0), d^-(0), v(1), a(1), ..., v(s_i), a(s_i)$).

Вершина i+1 (i) получает {i+1}-сообщение ({i}-сообщение) от корня через один такт в момент времени $t_{0,i+1}$+1 ($t_{0,i}$+1). Поскольку в вершине i+1 не происходит склейки, а в вершине i не происходит склейки типа 1, это будет первое сообщение, получаемое вершиной. Поэтому в эти моменты времени вершинам известна следующая информация:

для вершины i+1: $I_{i+1,i+1}$ =($v(0), d^-(0), v(1), a(1), ..., v(s_{i+1}), a(s_{i+1}), v(i+1), a(i+1)$),

для вершины i: $I_{i,i}$ =($v(0), d^-(0), v(1), a(1), ..., v(s_i), a(s_i), v(i), a(i)$,).

Поскольку $s_{i+1} < i$ и $s_i < i$, если мы поменяем местами номера вершин $v(i) \leftrightarrow v(i+1)$ и номера дуг $a(i) \leftrightarrow a(i+1)$, то информация в корне $I_{0,i+1}$ и $I_{0,i}$ не изменится, но сообщение от корня будет приходить не в вершину i+1, а в вершину i. Поэтому информация для вершин i и i+1 поменяется местами $I_{i+1,i+1} \leftrightarrow I_{i,i}$. Поэтому теперь первое сообщение, которое получит вершина i, будет {i}-сообщением, и оно сразу посылается дальше. После такого изменения нумерации вершин и дуг в вершине i не будет склейки типа 1 и 2.

Тип 3 (рис. 5). Вершина i сначала посылает последовательно {x}-сообщения, где $x = i..j$-1, потом получает {j}-сообщение, но не посылает его дальше, а потом получает {j+1}-сообщение.

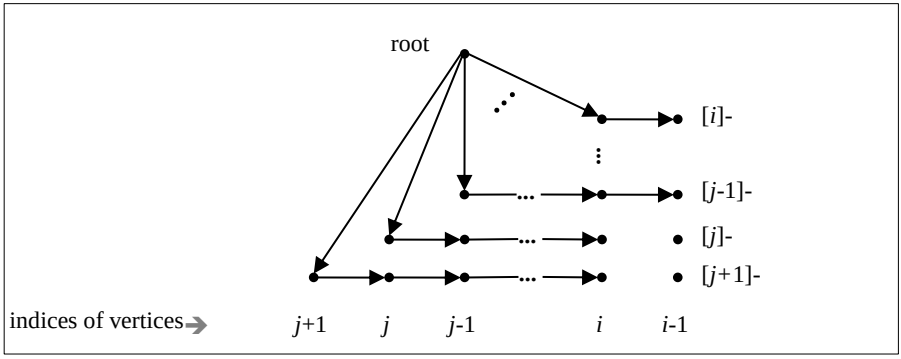


Рис. 5.Склейка сообщений типа 3
Fig. 5. Gluing messages of type 3

Для $y = i..j$+1 вершина y получала и посылала {y}-сообщение. Рассмотрим момент времени $t_{0,y}$, когда корень посылает сообщение по дуге $0—a(y)\rightarrow y$. К этому моменту времени корень мог получить сообщения от вершин самого правого отрезка в диапазоне индексов $[1..s_y]$. Должно быть $s_y < i$, поскольку в противном случае {y}-сообщение будет подтверждением в вершине i, т.е. к моменту склейки вершина i получит подтверждение. В момент времени $t_{0,y}$ корню известна следующая информация:

$I_{0,y}$=($v(0), d^-(0), v(1), a(1), ..., v(s_y), a(s_y)$),

Вершина y получает $\{y\}$-сообщение (от корня) через один такт в момент времени $t_{0,y}$+1. Поскольку в вершине $y > i$ не происходит склейки, а в вершине $y = i$ не происходит склейки типа 1 и 2, это будет первое сообщение, получаемое вершиной y. Поэтому в момент времени $t_{0,y}$+1 вершине y известна следующая информация:

$I_{y,y}$=($v(0), d^-(0), v(1), a(1), ..., v(s_y), a(s_y)$, $v(y), a(y)$).

Теперь рассмотрим момент времени $t_{y,j}$, когда вершина $y = i..j$ получает $\{j\}$-сообщение. Это сообщение проходит маршрут от вершины j до вершины y, поэтому в момент времени $t_{y,j}$ вершине y известна следующая информация:

$I_{y,j}$ =($v(0), d^-(0), v(1), a(1), ..., v(s_y^{max}), a(s_y^{max})$, $v(y), a(y), ..., v(j), a(j)$),

где $s_y^{max} = max\{ s_i, ..., s_y \}$.

Поскольку $s_y < i$, если мы поменяем местами номера вершин по циклу $v(j+1) \to v(j) \to ... \to v(i) \to v(j+1)$ и номера дуг по циклу $a(j+1) \to a(j) \to ... \to a(i) \to a(j+1)$, то информация в корне $I_{0,y}$ не изменится для каждого $y = i..j+1$, но сообщение от корня будет приходить не в вершину $y > i$, а в вершину y-1, и не в вершину $y = i$, а в вершину j+1. Поэтому в момент времени $t_{0,y}$+1 в вершине $y = i..j$ окажется та информация, которая была в вершине y+1, т.е. $I_{y+1, y+1}$, а в вершине $y = j$+1 окажется та информация, которая была в вершине i, т.е. $I_{i, i}$. В результате в момент времени $t_{i,j}$ в вершине i окажется та информация, которая была в вершине i+1, т.е. $I_{i+1,j}$. Поэтому теперь вершина i, получая $\{j\}$-сообщение, посылает его дальше. После такого изменения нумерации вершин и дуг в вершине i не будет склейки типа 3 для данного j, а также всех меньших его, поскольку j выбиралось минимальным. Также вершина i, по-прежнему будет получать в качестве первого сообщения $\{i\}$-сообщение и сразу посылать его дальше (как до изменения делала вершина i+1), поэтому в вершине i не будет склеек типов 1 и 2.

Итак, мы нашли такие изменения нумерации дуг и вершин, которые приводят к отсутствию в вершине i склейки типов 1 и 2, а также к отсутствию склейки типа 3 или к увеличению минимального j для типа 3. Поскольку число вершин конечно, мы можем повторять эти изменения до тех пор, пока при работе данного алгоритма не останется склеек. Утверждение доказано.

Утверждение 4. Время работы алгоритма на графе G_n равно $\Omega(n^2)$.

Доказательство: По утверждению 3 можно считать, что в вершине $i = 1..n$-1 не происходит склеек сообщений до получения этой вершиной первого подтверждения. Очевидно, корень получит $\{1\}$-сообщение через $t_1 \geq 2$ такта. Обозначим $x_1 = 1$. Пусть вершина x_j посылает $1 \leq y_j \leq k$ сообщений до того, как она перестает посылать сообщения и начинает ждать первого подтверждения. Обозначим $x_{j+1} = x_j + y_j$, тогда $x_{j+1} \leq x_j + k$. Если $x_{j+1} \leq n$-1, то, поскольку склеек нет, $\{x_{j+1}\}$-сообщение не может быть послано из вершины x_j до получения ею первого подтверждения. Обозначим через t_j время от начала работы, через которое корень получает $\{x_j\}$-сообщение. Только после этого не менее чем

через 1 такт вершина x_j получит подтверждение, и только после этого $\{x_{j+1}\}$-сообщение может пройти путь до корня длиной не менее x_j. Следовательно, $t_{j+1} \ge t_j + 1 + x_j$. Имеем:

$x_1 = 1, \qquad t_1 \ge 2,$

$x_1+1 \le x_2 \le x_1+k, \qquad t_2 \ge t_1 + 1 + x_1,$

$x_2+1 \le x_3 \le x_2+k, \qquad t_3 \ge t_2 + 1 + x_2,$

...

$x_{u-1}+1 \le x_u \le x_{u-1}+k, \quad t_u \ge t_{u-1} + 1 + x_{u-1}.$

Тогда $i \le x_i$ и $x_u \le 1+k(u-1)$. Пусть $n \ge k + 1$. Выберем u максимальное так, чтобы оставаться в диапазоне индексов, т.е. $0 \le n-1-k < x_u \le n-1$. Тогда $n-1-k < 1+k(u-1)$, что влечет $u > (n-2)/k$. Очевидно, $t(n) \ge t_u$. Имеем:

$t(n) \ge 2 + (1+x_1) + (1+x_2) + (1+x_3) + ... + (1+x_u) = 2 + u + (x_1+x_2+x_3+...+x_u) \ge$

$\ge 2 + u + (1+2+3+...+u) = 2 + u + u(u+1)/2 = 2+u(u+3)/2 >$

$> 2+((n-2)/k)(((n-2)/k)+3)/2 = (n-2)^2/(2k^2)+3(n-2)/2k+2 \ge n^2/(3k^2) = \Omega(n^2).$

Утверждение доказано.

10. Синхронная модель

До сих пор мы рассматривали асинхронную модель, в которой время пересылки сообщения по дуге может быть произвольным в диапазоне [0..1] и даже меняться со временем. В синхронной модели время пересылки считается равным 1 такту и не меняется. В то же время следует отметить, что под синхронной моделью понимается не просто модель с единичным временем пересылки, а с дополнительным свойством: вершина принимает сразу все сообщения, дошедшие до нее по входящим дугам. Это соответствует понятию *раунда* как единицы измерения времени. За один раунд каждая вершина принимает все пришедшие к ней по входящим дугам сообщения и посылает все сообщения по исходящим из нее дугам.

Такой режим работы можно симулировать в модели, где вершина за одно «срабатывание» принимает только одно сообщение, если вместе с сообщением вершина получает булевский признак «последнего сообщения», указывающий, является ли это сообщение последним или не последним из сообщений, дошедших до вершины и еще не принятых ею.

«Быстрый» алгоритм построения отображения (см. 8.1) легко модифицируется в синхронной модели так, что получаются оценки $T = O(n)$, $A = O(n^2 log n)$, $M = O(n^2 log n)$, $N = 1$. Для этого достаточно, чтобы вершина принимала сообщения до тех пор, пока не получит сообщение с признаком «последнее сообщение». Если в оригинальном алгоритме, приняв сообщение m, вершина должна была послать по дуге j сообщение $m`$, то в модифицированном алгоритме сообщение не посылается, а в памяти вершины запоминается пара $(m`, j)$. После получения сообщения с признаком «последнее сообщение» по

каждой дуге j посылается не более чем одно сообщение, склеенное из тех сообщений m`, которые были запомнены в паре с дугой j.

При единичном времени пересылки все сообщения, посланные в вершину на предыдущем раунде, одновременно доходят до вершины на следующем раунде через 1 такт, время между приемом первого и последнего из этих сообщений равно нулю, и по каждой исходящей дуге посылается не более одного сообщения. Поэтому $N = 1$, и сохраняется оценка $T = O(n)$. Остальные оценки увеличиваются не более чем в n раз: $A = O(n^2 log n)$, $M = O(n^2 log n)$, что объясняется необходимостью хранить в памяти вершины несколько (но не более n) сообщений и склеивать из нескольких (но не более n) сообщений одно сообщение. Более точно, в оригинальном алгоритме на одной дуге может оказаться не более одного сообщения каждого типа от одной вершины, а число типов ограничено. Таким образом, в модифицированном алгоритме в вершине дополнительно хранятся оригинальные сообщения не более чем от n вершин с ограниченным числом сообщений от каждой из них, а «склеенное» сообщение состоит из оригинальных сообщений не более чем от n вершин с ограниченным числом сообщений от каждой из них. Поскольку в оригинальном алгоритме $A = O(n log n)$ и $M = O(n log n)$, в модифицированном алгоритме получаются оценки $A = O(n^2 log n)$ и $M = O(n^2 log n)$.

Другим способом симуляции синхронной работы является сигнал «освобождение дуги j», который вершина получает, когда на дуге j не остается сообщений, не дошедших до конца дуги. В этом случае вершина, принимая сообщения, не посылает сообщение по дуге j до тех пор, пока не получит сигнал «освобождение дуги j». Возможно также, что имеется только общий сигнал «освобождение всех исходящих дуг» [7]. Для соответствующей модификации «быстрого» алгоритма построения отображения при единичном времени пересылки получаем такие же оценки $T = O(n)$, $A = O(n^2 log n)$, $M = O(n^2 log n)$, $N = 1$. Более того, эти оценки остаются верными для произвольного и даже меняющегося времени пересылки в диапазоне [0..1]. В этом случае время ожидания между приемом сообщения вершиной и его посылкой по исходящей дуге не превышает времени пересылки по этой дуге, т.е. 1 такта, что ведет к увеличению времени T не более чем в 2 раза.

11. Заключение

В разд. 8 для асинхронной модели распределенной системы предложены два алгоритма построения отображения, позволяющего пересылать сообщения между вершинами по кратчайшим путям. «Быстрый» алгоритм имеет оценки $T = \boldsymbol{O}(n)$, $A = \boldsymbol{O}(n log n)$, $M = \boldsymbol{O}(n log n)$, $N = \boldsymbol{O}(n)$, «экономный» алгоритм имеет оценки $T = \boldsymbol{O}(n^2)$, $A = \boldsymbol{O}(n log n)$, $M = \boldsymbol{O}(n log n)$, $N = \boldsymbol{O}(1)$. В разд. 9 доказано, что при таких оценках числа N одновременно передаваемых по дуге сообщений оценки времени T не улучшаемы при неограниченных размерах памяти вершин и сообщений: $T = \boldsymbol{\Theta}(n)$ при $N = \boldsymbol{O}(n)$ и $T = \boldsymbol{\Theta}(n^2)$ при $N = \boldsymbol{O}(1)$.

В разд. 10 показано, как «быстрый» алгоритм можно модифицировать для синхронной модели при условии, что имеется либо признак «последнее сообщение», либо сигнал «освобождение дуги *j*» или «освобождение всех исходящих дуг». В этом случае $T = \boldsymbol{O}(n)$, $A = O(n^2 log n)$, $M = O(n^2 log n)$, $N = 1$.

В то же время, если время пересылки равно 1 такту и не меняется, но признака «последнее сообщение» и сигналов «освобождение дуги *j*» или «освобождение всех исходящих дуг» нет, вопрос об оценке времени работы алгоритма с ограниченной емкостью дуги $N = O(1)$ остается открытым. Другое направление дальнейших исследований – это поиск эффективных алгоритмов решения задач на ориентированных графах, не сводимых к симуляции алгоритмов для неориентированной распределенной системы.

Список литературы

[1]. Y. Afek and E. Gafni, Distributed Algorithms for Unidirectional Networks, SIAM J. Comput., vol. 23, No. 6, 1994, pp. 1152-1178.

[2]. И.Б.Бурдонов. Обход неизвестного ориентированного графа конечным роботом. Программирование, 2004, №4, стр.11-34.

[3]. И.Б.Бурдонов. Проблема отката по дереву при обходе неизвестного ориентированного графа конечным роботом. Программирование, 2004, №6, стр.6-29.

[4]. И. Бурдонов, А. Косачев. Общий подход к решению задач на графах коллективом автоматов. Труды Института системного программирования РАН, том 29, вып. 2, 2017 г., стр. 27-76. DOI: 10.15514/ISPRAS-2017-29(2)-2.

[5]. И. Бурдонов, А. Косачев. Распределённые алгоритмы на корневых неориентированных графах. Труды Института системного программирования РАН, том 29, вып. 5, 2017 г., стр. 283-310. DOI: 10.15514/ISPRAS-2017-29(5)-14.

[6]. И. Бурдонов, А. Косачев. Размер памяти для хранения упорядоченного корневого графа. Труды Института системного программирования РАН, том 29, вып. 2, 2017 г., стр. 7-26. DOI: 10.15514/ISPRAS-2017-29(2)-1.

[7]. И.Б. Бурдонов, А.С. Косачев, В.В. Кулямин. Параллельные вычисления на графе. Программирование, 2015, №1, стр. 3-20.

Directed distributed system: Backtracking problem

I.B. Burdonov <igor@ispras.ru>
A.S. Kossatchev <kos@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. For a distributed system based on a directed graph without multiple edges and loops, the backtracing problem is considered: how to transfer a message from the final vertex of the arc to its initial vertex. The task is to create a structure on the graph that allows the message to be transmitted from the final vertex of the arc to its initial vertex in the minimum time, i.e. on the shortest path. Such a structure at each vertex *a* is given by mapping the number of vertex *b* to the number of the outgoing arc through which the shortest path passes from *a* to *b*. In particular, such a mapping makes it possible to simulate, in directed

distributed systems, algorithms for solving problems on a graph, developed for unditected distributed systems. This increases the running time of such algorithms by not more than k times, where k does not exceed the diameter of the graph, $k < n$, where n is the number of vertices of the graph. Section 2 describes the asynchronous model of the distributed system used. Section 3 contains the basic definitions and notation, and Section 4 – the statement of the problem. Section 5 describes two auxiliary algorithms for subtree correction, the application of which makes it possible to construct spanning trees of shortest paths: a out-tree and an in-tree. Section 6 contains a description of the various methods for transmitting messages over the graph. In Section 7, two algorithms are proposed for constructing in the memory of the graph root automaton the descriptions of spanning out- and in- shortest path trees, and in Section 8, the algorithms for constructing the required mapping based on them: a "fast" algorithm with $T = \boldsymbol{O}(n)$ and $N = \boldsymbol{O}(n)$ and an "economical" algorithm with $T = \boldsymbol{O}(n^2)$ and $N = \boldsymbol{O}(1)$, where T is the running time of the algorithm, N is the number of messages simultaneously transmitted along the arc. In Section 9 it is proved that these estimates of time are not improved. In Section 10, the "fast" algorithm is modified for a synchronous model with $N = 1$. The conclusion sums up and outlines directions for further research.

Keywords: directed graph; rooted graph; numbered graph; distributed algorithms; graph problems; backtracing problem; shortest paths.

DOI: 10.15514/ISPRAS-2018-30(2)-9

For citation: Burdonov I.B., Kossatchev A.S. Directed distributed system: Backtracking problem. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue. 2, 2018, pp. 167-194 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-9

References

[1]. Y. Afek and E. Gafni, Distributed Algorithms for Unidirectional Networks, SIAM J. Comput., Vol. 23, No. 6, 1994, pp. 1152-1178.

[2]. I.B.Bourdonov. Traversal of an Unknown Directed Graph by a Finite Robot. Programming and Computer Software, vol. 30, No. 4, 2004, pp. 188-203.

[3]. I.B.Bourdonov. Backtracking Problem in the Traversal of an Unknown Directed Graph by a Finite Robot. Programming and Computer Software, vol. 30, No. 4, 2004, pp. 305-322.

[4]. I.Burdonov, A.Kossatchev. General approach to solving problems on graphs by collective automata. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. 27-76 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-2.

[5]. I.Burdonov, A.Kossatchev. Distributed algorithms on root undirected graphs. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017, pp. 283-310 (in Russian). DOI: 10.15514/ISPRAS-2017-29(5)-14.

[6]. I.Burdonov, A.Kossatchev. The size of the memory for storing the ordered root graph. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 2, 2017, pp. 7-26 (in Russian). DOI: 10.15514/ISPRAS-2017-29(2)-1.

[7]. I.B. Burdonov, A.S. Kossatchev, V.V. Kulyamin. Parallel Computations on a Graph. Programming and Computer Software, vol. 41, No. 1, 2015, pp. 1-13. DOI: 10.1134/S0361768815010028.

Онтология предметной области «Удобство использования программного обеспечения»

А.А. Сытник <as@sstu.ru>
Т.Э. Шульга <shulga@sstu.ru>
Н.А. Данилов <Nikita_Danilov@outlook.com>
Саратовский государственный технический университет имени Ю. А. Гагарина,
410054, г.Саратов, ул.Политехническая, д. 77

Аннотация. В статье представлена онтология предметной области «Удобство использования программного обеспечения». Описываются преимущества, которые может дать ее использование при анализе и оценке удобства использования программных продуктов. Представлены диаграмма классов предлагаемой онтологии и текстовое описание входящих в ее состав классов, объектных свойств и свойств данных. Приводятся примеры вопросов, на которые может отвечать онтология. Предлагается классификация возможных методик анализа и оценки удобства использования на основе представленной онтологии и описываются некоторые из них.

Ключевые слова: удобство использования программного обеспечения; пользовательский интерфейс; онтология; онтологический инжиниринг

DOI: 10.15514/ISPRAS-2018-30(2)-10

Для цитирования: Сытник А.А., Шульга Т.Э., Данилов Н.А. Онтология предметной области «Удобство использования программного обеспечения». Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 195-214. DOI: 10.15514/ISPRAS-2018-30(2)-10

1. Введение

Стандарты ГОСТ Р ИСО 9241-210–2016 [1] и ISO 9241-11:1998 [2] определяют пригодность использования как свойство системы, продукции или услуги, при наличии которого установленный пользователь может применить продукцию в определенных условиях использования для достижения установленных целей с необходимой результативностью, эффективностью и удовлетворенностью. В профессиональной среде программистов более устоявшимся термином для данного свойства программного продукта является термин юзабилити (usability). Уровень удобства использования программного интерфейса влияет на качество всего программного обеспечения в целом [3]. Заметим, что в российских стандартах качества

программных средств термин «пригодность использования» упоминается как «удобство использования» и «практичность».

Существуют различные подходы и методики, применяемые для анализа удобства использования [4]. Однако, большинство из них основывается на эвристическом подходе, то есть на заранее собранных рекомендациях и гипотезах относительно того, как пользователь взаимодействует с программным интерфейсом системы. По этой причине их нельзя назвать в достаточной мере формализованными и их эффективность во многом зависит от уровня эксперта. Отсутствует и общая модель хранения данных активности пользователя, которая бы позволила свободно обмениваться данными, собираемыми в процессе тестирования удобства использования, и применять эти данные для ее анализа.

Возможным решением указанных проблем является разработка модели предметной области удобства использования в виде онтологии. Тенденции развития семантического веба позволяют утверждать, что онтология, как модель представления знаний, имеет ряд преимуществ. Согласно определению консорциума W3C, под онтологией понимается формальная модель представления знаний в некоторой предметной области, описывающая типы объектов (классы), взаимосвязи между ними (свойства), и способы совместного использования классов и свойств (аксиомы) [5].

Онтологии, публикуемые в вебе в стандартных форматах, позволяют упростить процесс распространения знаний и их повторного использования. Кроме того, модель онтологии подразумевает возможность ее последующего расширения или уточнения с целью использования в любых программных приложениях определенной предметной области.

Заметим, что в последние годы большое количество научных работ посвящено проблемам онтологического моделирования в разных предметных областях (например, [6-9]). В данной работе описывается онтология предметной области «Удобство использования программного обеспечения» и приводится описание вариантов ее возможного использования в методиках анализа юзабилити. При этом из трех групп метрик оценки удобства использования, определенных в стандартах [1,2] − результативность, эффективность, удовлетворенность, − в работе описываются методики оценки удобства использования, ориентированные на эффективность, а именно, на время выполнения задач пользователя.

2. Обзор онтологий

Основным объектом исследования при оценке удобства использования являются действия пользователя [1], процесс его работы с программной системой, то есть его активность: движение курсора мыши, клики клавиш мыши, нажатие клавиш клавиатуры, различные формы взаимодействия с сенсорным экраном и т.п. Таким образом, для оценки удобства использования

необходимо аккумулировать данные активности пользователей с их привязкой к пользовательскому интерфейсу.

Существующие онтологии позволяют описать интерфейс, например, [9,10] или способы взаимодействия пользователя с интерфейсом [11]. Однако, они не пригодны для последующего накопления данных о самом взаимодействии. Обоснуем данный вывод, приведя краткий обзор этих онтологий.

Энн Блэндфорд (Ann Blandford) и Томас Грин (Thomas Green) еще в 1997 году предложили онтологическую модель построения эскизов (Ontological Sketch Model, OSM) [11]. Они ставили перед собой цель разработать такой подход к оценке удобства использования ПО, который был бы основан на теоретических результатах исследований научных сообществ, но мог быть использован командами дизайнеров в промышленности. При использовании описанного ими подхода эксперт формирует структурированное, но простое представление анализируемого программного обеспечения на основе упрощенной модели онтологии. Онтология OSM покрывает три аспекта дизайна системы: сущности (entities), действия (actions), взаимосвязи (relationships). Ниже они рассмотрены на примере текстового редактора.

Сущность – понятие или объект, который пользователь должен знать (символ, слово, параграф, ширина колонки) и который обладает следующими свойствами:

- атрибуты (attributes) – дополнительные характеристики которыми обладает сущность, например, символ имеет шрифт, размер;
- доступность (accessibility) – на уровне понимания пользователя, уровне устройства или системы и общие;
- релевантность (relevance) – релевантность сущности по отношению к определенной области и/или к устройству, например, слово релевантно к области написания текста, а полоса прокрутки (англ. scrollbar) релевантна к текстовому редактору;
- видимость (persistent visibility) – может ли пользователь увидеть сущность, существует она условно или прозрачна;
- маскировка (disguise) – обладает ли сущность понятным названием или символикой.

Действие – то, что пользователь может совершать. Действие имеет следующие свойства:

- название действия;
- сущность(и), над которой(ми) совершается действие;
- эффект (effect), получаемый после совершения действия;
- контекст (context), контекстная информация.

Взаимосвязь – связь между сущностями, обладает свойствами:

- тип (type) – «состоит из», «влияет», «ограничивает» или любой другой;
- сущность(и) – две или более сущности, имеющие взаимосвязь.

Валерия Грибова в своей работе «Модель онтологии предметной области "Графический пользовательский интерфейс"», опубликованной в 2005 году [10], предложила модель, значительно более сложную, чем OSM, так как основная идея ее использования – формирование декларативной модели пользовательского интерфейса на основе универсальных моделей онтологий и последующая автоматическая генерация исполнимого кода интерфейса. Для описания модели пользовательского интерфейса разработана подробная модель онтологии «графический пользовательский интерфейс», которая описывает графические интерфейсные элементы, их свойства и связь друг с другом для формирования диалога с пользователем, основанном на экранных формах. Базовая универсальная онтология включает в себя более 50 классов, где классы содержат 10 и более свойств. Среди различных видов визуальных средств графического пользовательского интерфейса (ГПИ) выделяются две основные группы – окна и оконные элементы управления, и три дополнительные – панели управления, оконные меню и вспомогательные средства. Фрагмент иерархии элементов ГПИ (классов) представлен на рис. 1.

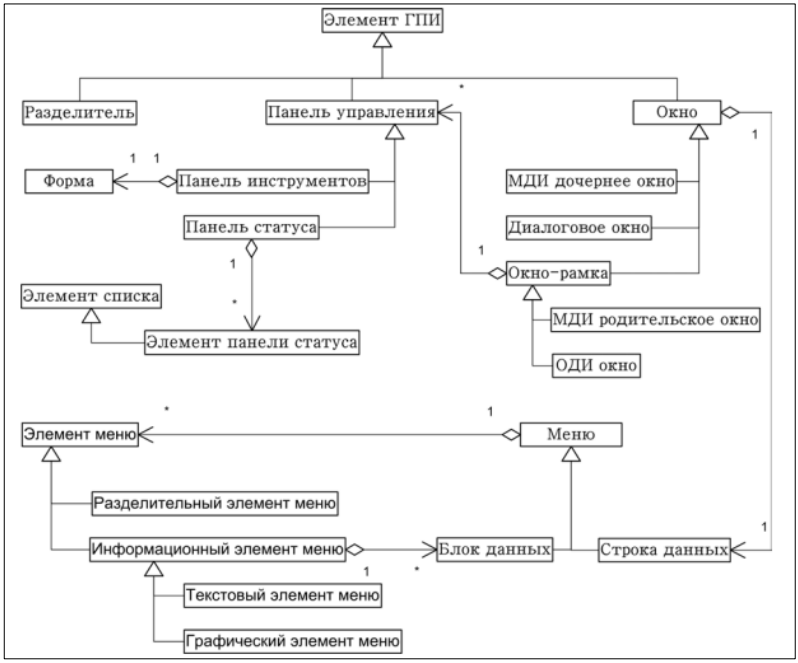


Рис. 1. Фрагмент иерархии элементов ГПИ
Fig. 1. A fragment of the GUI elements hierarchy

Элемент ГПИ – класс, описывающий общие для всех элементов ГПИ свойства, свойства: X (экранная координата левого верхнего угла элемента по оси X в пикселях, тип: целое число); Y (экранная координата левого верхнего

угла элемента по оси Y в пикселях, тип: целое число); ширина (ширина элемента в пикселях, тип: целое число); высота (высота элемента в пикселях, тип: целое число); отображаемость (признак, отображается ли элемент на экране, тип: булевский); доступность (признак, доступен ли элемент для взаимодействия с пользователем, тип: булевский); меню (каждому элементу соответствует [0, 1] контекстных меню в виде блока данных, тип: блок данных) и т.д.

Окно – класс, описывающий свойства области экрана, с помощью которой пользователь имеет возможность получить визуальное представление определенного аспекта решаемой задачи, суперкласс: «Элемент ГПИ», свойства: иконка (каждому рамочному элементу соответствует 0 или 1 небольших изображений, находящихся в левом верхнем углу экрана, которые используются для графической идентификации окна, тип: образ) и т.д.

В каталоге открытых связанных словарей (Linked Open Vocabularies) содержится онтология «Пользовательский интерфейс» (user interface, UI) [9] под авторством Тимоти Джона Бернерс-Ли (Timothy John Berners-Lee), английского ученого, одного из создателей всемирной паутины, автора концепции семантического веба. Онтология достаточно простая (24 класса, 27 свойств и 4 экземпляра), но содержит важные базовые понятия из предметной области пользовательского интерфейса, такие как: форма (англ. Form); поле (англ. Field) и т.д., фрагмент модели представлен на рис. 2.

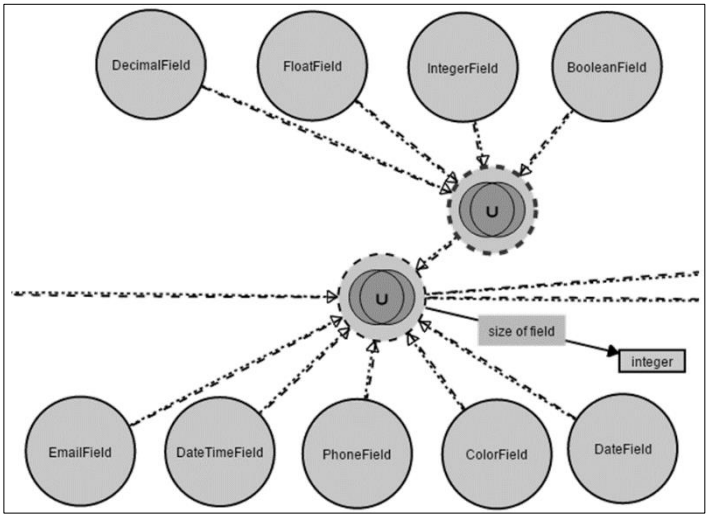


Рис. 2. Фрагмент онтологии пользовательского интерфейса
Fig. 2. A fragment of the User Interface ontology

Таким образом, рассмотренные онтологии позволяют описать интерфейс анализируемого программного обеспечения, но не содержат классов и

свойств, описывающих данные активности пользователей с привязкой к пользовательскому интерфейсу: движение курсора мыши, клики клавиш мыши, нажатие клавиш клавиатуры, различные формы взаимодействия с сенсорным экраном и т.п.

Существуют и другие онтологии, упоминаемые в научных работах и связанные в той или иной мере с предметными областями «Пользовательский интерфейс» и «Удобство использования ПО», но не все из них удается найти в открытых каталогах онтологий. Онтологии, связанные непосредственно с терминами «Удобство использования» и «Данные активности пользователя», найти в открытых научных источниках не удалось. Однако рассмотренные онтологии могут служить отправной точкой при разработке общей онтологии предметной области «Удобство использования программного обеспечения».

3. Описание онтологии предметной области «Удобство использования»

Онтология предметной области «Удобство использования программного обеспечения» в первую очередь предназначена для фиксации данных о взаимодействии пользователя с программной системой посредством графического пользовательского интерфейса. При оценке удобства использования необходимы данные именно об активности пользователя. Учитывая современное развитие технологий, можно выделить огромное количество способов и видов взаимодействия пользователя с пользовательским интерфейсом.

Подробное описание пользовательского интерфейса, в свою очередь, с детализацией всех элементов по их типам и характеристикам в данный момент, по мнению авторов, не представляется необходимым. Для решения базовых задач оценки удобства использования достаточно общего разбиения пользовательского интерфейса на интересующие эксперта участки (регионы). Таким образом, предметная область «пользовательский интерфейс» не покрывается данной онтологией.

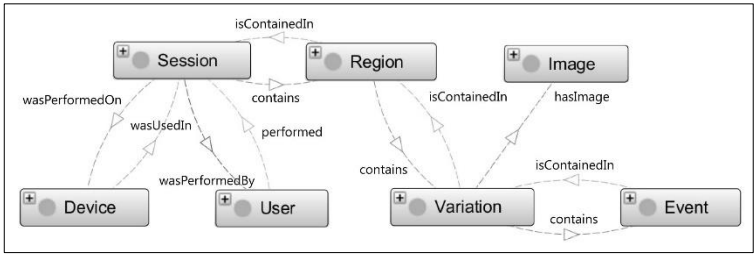


Рис. 3. Фрагмент структуры классов онтологии «Удобство использования программного обеспечения»
Fig. 3. Classes structure fragment of the "Software Usability" ontology

Опишем основные понятия, используемые в рассматриваемой предметной области и взаимосвязи между ними, т.е. классы и свойства предлагаемой онтологии (рис. 3). Заметим, что на рис. 3 в виде дуг отображены только основные объектные свойства онтологии (описывающие отношения между экземплярами классов, англ. Object Properties). Свойства данных (описывающие отношения экземпляров класса с литеральными значениями, англ. класса DataType Properties) описаны ниже.

Класс Session предназначен для описания сессии − временного промежутка, в течение которого пользователь взаимодействует с программной системой. В рамках сессии накапливаются и хранятся все данные об активности пользователя, события, действия пользователя и вспомогательная информация. Свойства данных hasStartDateTime и hasEndDateTime позволяют указать границы промежутка времени.

Класс Device описывает устройство, на котором выполнялась сессия. Device в текущей реализации являются простым классом, содержащим лишь базовую информацию для понимания «где» выполнялась сессия работы с программной системой. Экземпляры устройств привязываются к сессиям свойством wasUsedIn, которое является инверсивным по отношению к wasPerformedOn. Свойство данных hasName позволяет указать краткое имя для устройства.

Класс User описывает пользователя, которым выполнялась сессия. User в текущей реализации являются простым классом, содержащим лишь базовую информацию для понимания «кем» выполнялась сессия работы с программной системой. Экземпляры пользователей привязываются к сессиям свойством performed, инверсивным по отношению к wasPerformedBy. Свойство данных hasName позволяет указать краткое имя пользователя.

Для каждого экземпляра сессии, должен быть указан строго один экземпляр устройства, на котором она совершалась (объектное свойство wasPerformedOn), и строго один экземпляр пользователя, которым она совершалась (объектное свойство wasPerformedBy).

Класс Region описывает регион, т.е. область пользовательского интерфейса, например, окно целиком, либо его отдельную часть. Регион связывается с сессией транзитивным свойством isContainedIn. Экземпляры сессий связываются с коллекцией экземпляров регионов с помощью объектного транзитивного свойства contains. Свойство данных hasName позволяет указать имя региона.

Класс Variation предназначен для описания вариации региона, так как каждый регион, в свою очередь, может иметь одну или более вариаций. Под вариацией региона в общем случае может пониматься уникальное сочетание параметров высоты и ширины изображения региона, либо их диапазон. Выделение вариаций необходимо при анализе адаптивного пользовательского интерфейса, когда внешний вид региона и, возможно, функциональность, меняется в зависимости от его размеров. Например, крупные кнопки с подробными надписями могут заменяться на более мелкие с пиктограммами и

без надписей (рис. 4). Минимальные и максимальные границы ширины и высоты определяются свойствами данных hasMinWidth, hasMaxWidth, hasMinHeight, hasMaxHeight. Свойство данных hasName определяет название вариации. Вариация связывается с регионом свойством isContainedIn. Регион связывается со всеми своими вариациями объектным свойством contains.

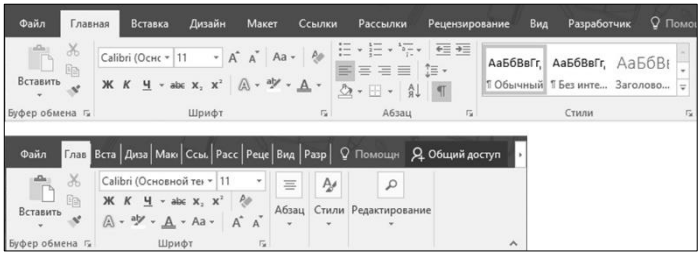


Рис. 4. Пример вариаций региона панели инструментов текстового процессора MS Word
Fig. 4. Region variations example for the MS Word toolbar

Класс Image используется для описания изображения. При оценке удобства использования и анализе данных активности пользователя может потребоваться знание того, как выглядел пользовательский интерфейс при работе с ним, т.е. его изображение. Например, кроссплатформенное прикладное программное обеспечение работает в самых разных условиях и на разных операционных системах. Общепринятой практикой является что приложение «подстраивается» под условия работы.

Для каждого изображения создается экземпляр класса Image, который связывается строго с одним экземпляром вариации. Свойства данных hasWidth и hasHeight позволяют определить ширину и высоту изображения. Свойства данных hasDpiX и hasDpiY определяют разрешение изображения. Объектное свойство hasImage описывает прямую связь вариации с изображением, а инверсивное ему объектное свойство wasImaged описывает обратную.

Класс Event представляет собой базовый класс для событий, которые могут происходить при взаимодействии пользователя с программной системой. Вариация имеет связь contains со всеми событиями. Объектное свойство isContainedIn позволяет указать строго одну вариацию в которой произошло событие.

Событием может быть любое действие пользователя, команда и т.д. Набор информации о событии зависит от его конкретного типа. Общим для всех событий является свойство данных hasDateTime, которое указывает, когда событие произошло. Класс Event можно считать абстрактным, т.к. он не несет информации о типе события. Для анализа требуется работать с детализированными типами событий, подклассами класса Event. Опишем структуру подклассов событий (рис. 5).

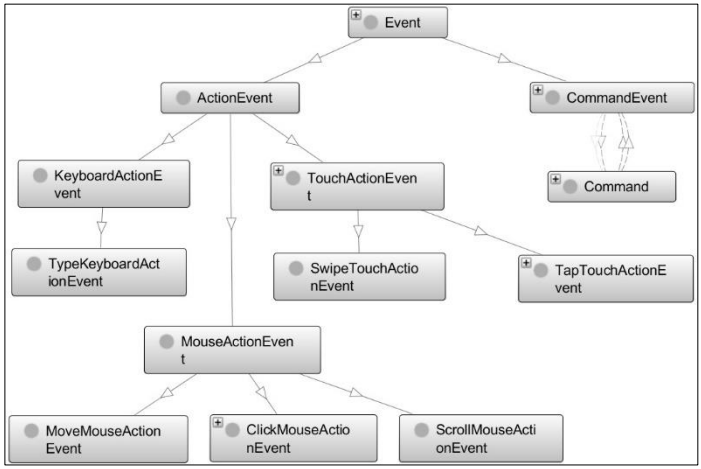


Рис. 5. Подклассы класса Event онтологии «Удобство использования программного обеспечения»
Fig. 5. Event subclasses of the "Software Usability" ontology

Класс ActionEvent является подклассом Event и является надклассом всех классов событий действий. Они служат для фиксации действий пользователя, которые относятся к механическому взаимодействию пользователя с программным интерфейсом (движение курсора мыши, клики), обычно, не несущим связи с функциональностью программного продукта и функциональными требованиями напрямую.

Подкласс KeyboardActionEvent описывает класс для событий действий, связанных с манипуляцией пользователя с клавиатурой.

Подкласс TypeKeyboardActionEvent описывает событие нажатие клавиши клавиатуры.

Подкласс MouseActionEvent описывает события действий, связанных с манипуляцией пользователя с мышью.

Подкласс ClickMouseActionEvent соответствуют событию нажатия клавиши мыши. Для события необходимо указание информации о координатах курсора мыши, где произошло событие, с помощью свойств hasInRegionX и hasInRegionY. Подклассы SingleClickMouseEvent и DoubleClickMouseEvent описывают одинарный и двойной клики соответственно.

Подкласс MoveMouseActionEvent соответствует событию движения курсора мыши. Свойства hasInRegionX и hasInRegionY позволяют указать новые координаты курсора мыши после перемещения.

Подкласс TouchActionEvent описывает события действий, связанных с взаимодействием пользователя с сенсорным экраном (тачскрин, англ. touchscreen), например, дисплеем телефона или планшетного компьютера.

Подкласс TapTouchActionEvent описывает касание пользователем сенсорного экрана. Можно считать условным аналогичным ClickMouseActionEvent, но для сенсорного экрана.

Подклассы SingleTapTouchActionEvent, DoubleTapTouchActionEvent, LongTapTouchActionEvent, HoldTapTouchActionEvent описывают соответственно: одиночное быстрое касание, двойное быстрое касание, одинарное долгое касание и касание с удержанием. Подобная детализация важна в случае сенсорных экранов, поскольку способ касания определяет ответную реакцию программного обеспечения.

Подкласс SwipeTouchActionEvent описывает событие способа взаимодействия пользователя с интерфейсом, когда пользователь проводит пальцем или пальцами в одном направлении.

Класс CommandEvent является подклассом класса Event и описывает события команд, или командные события. Они служат для фиксации факта вызова определенной команды (функции) пользователем.

Класс Command описывает команду, которая является функциональным действием пользователя (обычно, связанным с функциональными требованиями), подразумевающим ответную реакцию программной системы. Свойство hasName позволяет указать название команды.

Все командные события связываются с конкретными экземплярами команд свойством wasAssociatedWith, а команды с командными событиями, в свою очередь, свойством wasInvokedIn. Для упрощения последующего использования онтологии, данная связь дополнительно дублируется транзитивными свойствами contains и isContainedIn, где событие содержит команду, а команда содержится в событии.

4. Использование онтологии при анализе и оценке удобства использования

Предложенная онтология может быть использована при анализе и оценке удобства использования пользовательских интерфейсов программного обеспечения. Для этого могут потребоваться различные выборки данных, связанные с различными событиями. Важно отметить, что не для каждого выполненного действия гарантируется исполнение команды. Перемещение курсора, в большинстве случаев, не вызывает выполнение какой-либо команды. Клик курсора мыши в районе кнопки может не привести к выполнению команды, если кнопка была неактивна или если пользователь «промахнулся». Однако, независимо от этого любое механическое взаимодействие пользователя с интерфейсом напрямую затрачивает его ресурсы, такие как время и внимание.

Прежде всего, приведем примеры вопросов, на которые может отвечать описанная онтология.

- Какие команды совершены за сессию?

- Какие действия совершены в регионе?
- Сколько кликов мышью совершено в регионе?
- Какие команды совершены пользователями в регионе?
- Какие вариации региона встречаются в сессии?

Допустим, существует регион с названием «SomeRegion» (свойство данных hasName). Приведем пример SPARQL-запроса, отвечающего на вопрос «Какие команды совершены в регионе ‘SomeRegion’?»:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX usa:
<http://www.semanticweb.org/nikita_danilov/ontologies/usability#>
SELECT DISTINCT ?cname
WHERE {
    ?r rdf:type usa:Region .
    ?r usa:hasName ?rname .
    ?r usa:hasName
"SomeRegion"^^<http://www.w3.org/2001/XMLSchema#string> .
    ?v rdf:type usa:Variation .
    ?ce rdf:type usa:CommandEvent .
    ?c rdf:type usa:Command .
    ?c usa:hasName ?cname .
    ?r usa:contains+ ?c }
```

Транзитивность свойства contains позволяет избежать перечисления полной вложенности сущностей (вариации в регионе, командное событие в вариации, команда в командном событии).

Если требуется подсчитать в целом по всем имеющимся данным сколько раз была вызвана каждая команда в каждом регионе, то SPARQL-запрос будет выглядеть следующим образом:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX usa:
<http://www.semanticweb.org/nikita_danilov/ontologies/usability#>
SELECT ?rname ?cname (count(?c) as ?count)
WHERE {
    ?r rdf:type usa:Region .
    ?r usa:hasName ?rname .
    ?v rdf:type usa:Variation .
    ?ce rdf:type usa:CommandEvent .
    ?c rdf:type usa:Command .
    ?c usa:hasName ?cname .
    ?r usa:contains+ ?c }
GROUP BY ?rname ?cname
ORDER BY ?rname ?cname
```

Подобные запросы могут использоваться в составе методик анализа и оценки удобства использования. Авторами предлагается их классификация в зависимости от базовой метрики удобства использования [1], на которую они ориентированы. Всего стандарт ISO 9241-11:1998 [2] определяет 3 группы таких метрик (показателей).

- Результативность (англ. effectiveness) – степень реализации

запланированной деятельности и достижения запланированных результатов.

- Эффективность (англ. efficiency) – связь между достигнутым результатом и использованными ресурсами.
- Удовлетворенность (англ. satisfaction) – комфорт и приемлемость использования, отсутствие дискомфорта при использовании продукции или положительное отношение к ней.

Результативность и эффективность основаны на достаточно формальных показателях, таких как количество выполненных задач и объем затраченных ресурсов. Поэтому, по мнению авторов, данные метрики возможно формализовать с достаточной степенью точности.

Результативность – точность и полнота, с которой пользователи достигают поставленных целей, успешность выполнения промежуточных задач необходимых для достижения цели. Ориентированные на результативность (effectiveness-oriented) методики могут быть связаны непосредственно показателями с успешности выполнения задач. Существуют различные показатели результативности. Стандарт ISO 9241-11:1998 [2] определяет, например, такие как: процент достигнутых целей, процент пользователей, успешно завершивших задачу, средняя точность выполненных задач.

Важно отметить, что понятие «задача» в предметной области «Удобство использования программного обеспечения» является комплексным и вариативным. В общем виде, под задачей понимается деятельность, необходимая пользователю для достижения определенной поставленной цели [1]. Однако, критерии определения успешного выполнения задачи зависят от анализируемого программного обеспечения, конкретного региона(ов), функциональности и целей исследования. Под успешным выполнением задачи (успехом) может пониматься как сложное действие, вроде завершения оформления товара в интернет-магазине, так и достаточно простое вроде отправки документа на печать.

Методики, ориентированные на результативность, потребуют добавления в онтологию таких классов как: задача, успех, ошибка и т.д. На данный момент, авторами представляется это излишним усложнением и требующим более тщательного экспериментального исследования. Поэтому, методики данной группы детально не описываются в рамках представленной работы.

Удовлетворенность оценивается как отношение к использованию продукта, так и восприятие пользователем таких показателей, как экономичность, полезность или легкость в изучении. Данная метрика во многом зависит от субъективной оценки. Поэтому, в настоящий момент она не рассматривается авторами в качестве возможной для формализации.

Далее рассмотрим детально методики, ориентированные на эффективность (efficiency-oriented), в виду их наибольшего потенциала, по мнению авторов.

5. Методики оценки удобства использования, ориентированные на эффективность

Эффективность – отношение израсходованных ресурсов к точности и полноте, с которой пользователи достигают поставленных целей.

Ориентированные на эффективность методики связывают успешность выполнения задач с затрачиваемыми ресурсами. Как и для результативности, существуют детализированные показатели эффективности. Стандарт ISO 9241-11:1998 [2] определяет, например, следующие показатели:

- время необходимое на завершение задачи;
- задачи, выполненные в единицу времени;
- финансовая стоимость выполнения задачи.

Вероятно, самой наглядной и известной методикой в данной категории является построение и анализ тепловых карт на основе выборки данных по действиям пользователей. Тепловые карты известны довольно давно и функционал для их построения представляется различным программным обеспечением и веб-сервисами. Однако, формат хранения данных активности пользователей обычно закрыт. Использование онтологии в данном случае упростит обмен данными активности пользователей за счет использования общей модели данных, представленной онтологией.

Ранее авторами был разработан метод построения тепловых карт на основе данных активности пользователей [12,13]. Кроме того, было разработано программное обеспечение для сбора и анализа данных активности пользователей на основе представленной онтологии [14]. Их совместная апробация проведена на базе программно-аппаратного комплекса «Лего-машина Тьюринга» [15], ранее разработанного авторами для демонстрации принципов работы абстрактного исполнителя. Программная часть указанного комплекса является прикладным программным обеспечением и относится к категории настольных приложений для операционных систем семейства Windows. В программную часть было внедрено программное обеспечения для сбора данных активности пользователей.

Полученные данные позволили построить тепловые карты регионов и провести анализ удобства использования. Например, тепловая карта, представленная на рисунке 6, демонстрирует множественные попытки взаимодействия пользователей с полосой прокрутки, что является индикатором ошибки для данного региона, т.к. пользователь пытается взаимодействовать с неактивным элементом интерфейса. В то же время, отсутствие «промахов» пользователями мимо кнопок означает отсутствие ошибок в основном сценарии взаимодействия с программным интерфейсом.

Авторами в данный момент разрабатывается так же методика, нацеленная на поиск ранее неизвестных повторяющихся последовательностей команд. Наличие таких последовательностей может сигнализировать о существовании совокупностей команд, требующих реорганизации.

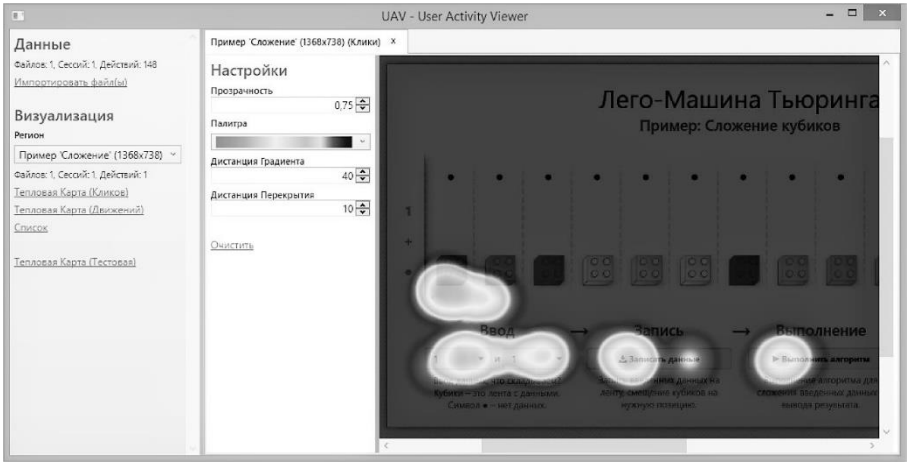


Рис. 6. Тепловая карта одного из регионов комплекса «Лего-машина Тьюринга»
Fig. 6. Heatmap of one of the regions of the “LEGO Turing machine” complex

Допустим, определено множество команд {1, 2, 3, 4}. Имеется выборка использованных команд в хронологическом порядке из данных активности пользователя за одну сессию:

{2, 1, 4, 1, 2, 3, 1, 2, 1, 1, 2}

Последовательность {1, 2} часто встречается в сессии, т.е. команды {1} и {2} часто выполняются последовательно. Данное наблюдение может сигнализировать о необходимости детального анализа указанных команд. Возможно, что команда {1} лишь открывает доступ к часто используемой {2} и необходимо повысить доступность {2}. Либо необходимо создать новую команду {1+2}, которая бы выполняла автоматически обе команды.

Разработанная онтология позволит проводить детальный анализ статистики затраченного времени и действий в конкретном регионе, возможно, с разбиением на различные категории пользователей или устройств. Обнаружение, например, избыточных действий (перемещений курсора), может сигнализировать о необходимости перегруппировки элементов пользовательского интерфейса. Возможен сбор и анализ различной статистики для ранее известных категорий пользователей, с целью лучшего их понимания. Например, выявление частоты использования «горячих клавиш».

Для выбора конкретного решения потребуется привлечение эксперта и экспертный анализ. Оно будет зависеть от типа программного обеспечения, его функциональности, категории пользователей и множества других факторов, которые должен будет учесть эксперт по удобства использования. Однако, применение подобных методик позволит автоматизированным образом выявить все участки программной системы, требующие детального

анализа. Это, в свою очередь, сэкономит время на поиски таких участков и во многом уменьшит субъективность при их отборе.

6. Наполнение онтологии экземплярами

Для качественного анализа и проверки адекватности предложенной онтологии требуются достаточно большие наборы данных, собранные для отдельных типов интерфейсов. Предполагается, что данные в общем случае, будут собираться автоматически, то есть необходимы инструменты (программное обеспечение) для автоматического наполнения онтологии экземплярами на основе данных активности пользователей приложений с различными типами интерфейсов. При этом формат и способ хранения данных могут зависеть от конкретной реализации. Главное, чтобы данные содержали в себе всю информацию соответственно структуре классов представленной онтологии и были доступны для последующего преобразования в онтологический вид.

Авторами предлагается использовать RDF-модель (Resource Description Framework) в стандартной RDF/XML нотации для упрощения взаимодействия различного программного обеспечения при сборе, передаче и анализе данных активности пользователей.

В рамках данной работы эксперимент по проверки адекватности онтологии был проведен для прикладного программного обеспечения, относящегося к категории настольных приложений для операционных систем семейства Windows. Разработанное авторами программное обеспечение [15] позволяет собирать базовые данные о действиях пользователя, взаимодействующего с таким типом интерфейсов, а именно: передвижение курсора мыши, одинарный клик мыши, вызванная команда и т.п. Собранные данные могут сохраняться в RDF/XML формате, соответствующем разработанной онтологии. В рамках эксперимента данные собирались для 6 программных приложений, в том числе упомянутого выше комплекса «Лего-машина Тьюринга». После этого данные из различных источников были использованы для построения тепловых карт отдельных регионов приложений.

Например, приведенная выше тепловая карта (рис. 6) построена на основе собранных таким образом данных, фрагмент которых выглядит следующим образом:

```
<rdf:RDF
  xmlns="http://ontologies/usability#"
  xml:base="http://ontologies/usability"
  xmlns:us="http://ontologies/usability#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <Session
    us:hasStartDateTime="2017-10-12T13:18:58+04:00"
    us:hasEndDateTime="2017-10-12T13:45:51+04:00"
    us:hasUid="ca0ad458-d3ab-4cd3-8192-5a29511b0f68">
    <wasPerformedBy>
      <User us:hasName="Школьник" />
```

```
        </wasPerformedBy>
        <contains rdf:parseType="Collection">
          <Region us:hasName="Пример: Сложение кубиков">
            <contains rdf:parseType="Collection">
              <Variation>
                <contains rdf:parseType="Collection">
                  <SingleClickMouseEvent
                    us:hasDateTime="2017-10-12T13:20:00+04:00"
                    us:hasInRegionX="27"
                    us:hasInRegionY="627" />
                  <SingleClickMouseEvent
                    us:hasDateTime="2017-10-12T13:20:04+04:00"
                    us:hasInRegionX="28"
                    us:hasInRegionY="625" />
                  ...
                </contains>
              </Variation>
            </contains>
          </Region>
        </contains>
      </Session>
    </rdf:RDF>
```

Пример данных содержит одну сессию с датой начала 2017-10-12T13:18:58+04:00, датой окончания 2017-10-12T13:45:51+04:00 и уникальным идентификатором «ca0ad458-d3ab-4cd3-8192-5a29511b0f68». Сессия исполнена пользователем с именем «Школьник» и содержит один регион с именем «Пример: Сложение кубиков» с одной вариацией по-умолчанию без имени. Вариация содержит два события типа «SingleClickMouseEvent» (одинарный клик мыши), прочие событие опущены для краткости примера.

Согласно экспертной оценке, построенные на основе онтологических данных тепловые карты могут эффективно использоваться при анализе удобства использования программного обеспечения.

Таким образом, показано, что предложенная онтология адекватна структуре данных активностей пользователей прикладного программного обеспечения, относящегося к категории настольных приложений для операционных систем семейства Windows.

7. Заключение

Описанная версия онтологии «Удобство использования программного обеспечения» предназначена для представления данных активности пользователей прикладного программного обеспечения. Проведенные эксперименты показали, что онтология может быть использована для анализа удобства использования настольных приложений для операционных систем семейства Windows, и в том числе для построения тепловых карт.

Кроме того, онтология может быть использована и расширена любыми исследователями, занимающимися проблемами удобства использования

программных интерфейсов, в частности в рамках экспериментов с различными типами интерфейсов. Авторы будут рады замечаниям и предложениям по уточнению и развитию онтологии от специалистов в данной предметной области и готовы к сотрудничеству.

В данный момент проходит процесс публикации онтологии в вебе под открытой лицензией в одном из синтаксисов языка OWL, а также осуществляется разработка формальных методов ее использования для оценки удобства использования программных продуктов.

Список литературы

[1]. Федеральное агентство по техническому регулированию и метрологии, ГОСТ Р ИСО 9241-210–2016, "Эргономика взаимодействия человек-система – Часть 210: Человеко-ориентированное проектирование интерактивных систем". Информационный портал по стандартизации. Стандартинформ, 2017.

[2]. ISO 9241-11:1998, ISO 9241-11:1998. Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. ISO, URL: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-1:v1:en (дата обращения 10.01.2017).

[3]. Федеральное агентство по техническому регулированию и метрологии ГОСТ-28806-90: Качество программных средств. Термины и определения. Информационный портал по стандартизации, Стандартинформ, 2017.

[4]. Данилов Н.А., Шульга Т.Э. Обзор моделей онтологий предметных областей «пользовательский интерфейс» и «юзабилити». Проблемы управления, обработки и передачи информации (УОПИ-2015): сб. тр. IV Междунар. науч. конф.: в 2 т. под ред. А.А. Львова и М.С. Светлова, Саратов: Издательский Дом «Райт-Экспо», 2015, том 1, стр. 214-218.

[5]. Linked Data Glossary. W3C Working Group Note 27 June 2013. URL: http://www.w3.org/TR/2013/NOTE-ld-glossary-20130627/#ontology (дата обращения 01.-02.2017).

[6]. Когаловский М.Р. Системы доступа к данным, основанные на онтологиях. Программирование, том 38, № 4, 2012 г., стр. 55-77.

[7]. Астраханцев Н.А., Турдаков Д.Ю. Методы автоматического построения и обогащения неформальных онтологий. Программирование, том 39, № 1, 2013 г., стр. 23-34.

[8]. Шульга Т.Э., Вагарина Н.С., Мельникова Н.И., Мищенко Д.А. Модели и инструменты представления пространственно-временных данных в семантическом вебе. Известия Самарского научного центра Российской академии наук, том 18, № 4-4, 2016 г., стр. 844-851.

[9]. G. Atemezing, J. Berners-Lee. A user interface ontology (ui). Linked Open Vocabularies, 2014, URL: http://lov.okfn.org/dataset/lov/vocabs/ui (дата обращения 20.10.2016).

[10]. Грибова В.В., Тарасов А.В. Модель онтологии предметной области «Графический Пользовательский Интерфейс». Интеллектуальные системы, 2005, №1(9).

[11]. A. Blandford, T. Green. OSM: an ontology-based approach to usability evaluation. Proceedings of Workshop on Representations. Queen Mary & Westfield College, 1997, Июль.

[12]. Danilov N., Shulga T., Frolova N., Melnikova N., Vagarina N., Pchelintseva E. Software usability evaluation based on the user pinpoint activity heat map. Advances in Intelligent Systems and Computing, vol. 465, 2016, pp. 217-225.

[13]. Данилов Н. А., Шульга Т. Э. Метод построения тепловой карты на основе точечных данных об активности пользователя приложения. Прикладная информатика, том. 10, № 2 (56), 2015 г., стр. 49-58.

[14]. Шульга Т.Э., Данилов Н.А. Свидетельство о государственной регистрации программ для ЭВМ № 2014662094 «Программный комплекс для сбора и визуализации данных активности пользователя настольного приложения» от 6 октября 2014 г.

[15]. Шульга Т. Э., Данилов Н. А., Валявский В. А., Митрофанов А. А., Мурзабеков А. М. Свидетельство о государственной регистрации программы для ЭВМ №2015610568 «Лего-машина Тьюринга» от 13 января 2015 г.

Ontology of the “Software Usability” Domain

A.A. Sytnik <as@sstu.ru>
T.E. Shulga <shulga@sstu.ru>
N.A. Danilov <Nikita_Danilov@outlook.com>
Yuri Gagarin State Technical University of Saratov, 77, Politechnicheskaya st., Saratov, 410054, Russia

Abstract. The article presents the ontology of the "Software usability" domain. Authors provides the review of existing ontologies for “user interface” and “software” domains. The article describes the advantages that can give its use in the analysis and evaluation of software products usability, e.g.: opened data structure for more easily data sharing. The paper presents a class diagram of the proposed ontology and a text description for the classes, object properties and data properties. Presented diagrams contains basic classes (device, session, user, region, variation, image) and subclasses for the event class. Examples of questions that can be answered by ontology are given. e.g.: “What commands are made by users in the region?” and “What variations of the region are in the session?”. Also, the article presents the SPARQL queries that may be used for this ontology. The classification of possible methods of analysis and evaluation of usability on the basis of the ontology is proposed and some of them are described. An example of the user activity data in a standard RDF/XML format is demonstrated. Described ontology "Software Usability" designed for the user activity data representation. Experiments have shown that ontology can be used to analyze the usability of desktop applications for Windows operating systems, including the construction of heat maps. Heat map based on the collected user activity data in is shown. In addition, ontology can be used and extended by any researchers involved in the problems of ease of use of software interfaces, in particular in experiments with different types of interfaces. The authors will welcome comments and suggestions on the refinement and development of ontology from experts in the domain and are ready to cooperate.

Keywords: software usability; user interface; ontology; ontological engineering.

DOI: 10.15514/ISPRAS-2018-30(2)-10

For citation: Sytnik A.A., Shulga T.E., Danilov N.A. Ontology of the “Software Usability” Domain. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue. 2, 2018, pp. 195-214 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-10

References

[1]. Federal Agency for Technical Regulation and Metrology, GOST R ISO 9241-210–2016, "Ergonomics of human-system interaction. Part 210. Human-centred design for interactive systems". Standardization information portal. Standartinform, 2017 (in Russian).

[2]. ISO 9241-11:1998, ISO 9241-11:1998. Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability. ISO, URL: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-1:v1:en.

[3]. Federal Agency for Technical Regulation and Metrology, GOST-28806-90: Software quality, Terms and definitions. Standardization information portal, Standartinform, 2017 (in Russian).

[4]. Danilov N.A., Shulga T.E. Review of ontology models of "user interface" and "usability" domains. Problemy upravlenija, obrabotki i peredachi informacii (UOPI-2015): sb. tr. IV Mezhdunar. nauch. konf.: v 2 t. pod red. A.A. L'vova i M.S. Svetlova [Proc. of conference “Problems of information management, processing and transmission (UOPI-2015)”], Saratov: Publishing House «Rajt-Jekspo», 2015, vol. 1, pp. 214-218 (in Russian).

[5]. Linked Data Glossary. W3C Working Group Note 27 June 2013. URL: http://www.w3.org/TR/2013/NOTE-ld-glossary-20130627/#.

[6]. Kogalovsky M.R. Ontology-based data access systems. Programming and Computer Software, vol. 38, issue 4, 2012, pp. 167-182. DOI: 10.1134/S0361768812040032.

[7]. Astrakhantsev N.A., Turdakov D.Y. Automatic construction and enrichment of informal ontologies: A survey. Programming and Computer Software, vol. 39, issue 1, 2013, pp. 34-42. DOI: 10.1134/S0361768813010039.

[8]. Shulga T.E., Vagarina N.S., Melnikova N.I., Mishhenko D.A. [Models and tools for presenting spatiotemporal data in a semantic web], Izvestija Samarskogo nauchnogo centra Rossijskoj akademii nauk, 2016, vol. 18, issue 4-4, pp. 844-851.

[9]. G. Atemezing, J. Berners-Lee. A user interface ontology (ui). Linked Open Vocabularies, 2014, URL: http://lov.okfn.org/dataset/lov/vocabs/ui (acessed 20.10.2016).

[10]. Gribova V.V., Tarasov A.V. The ontology model of the “Graphical User Interface” domain. [Intelligent system], Intellektual'nye Sistemy, 2005, issue 1(9) (in Russian).

[11]. A. Blandford, T. Green. OSM: an ontology-based approach to usability evaluation. Proceedings of Workshop on Representations. Queen Mary & Westfield College, 1997, Июль.

[12]. Danilov N., Shulga T., Frolova N., Melnikova N., Vagarina N., Pchelintseva E. Software usability evaluation based on the user pinpoint activity heat map. Advances in Intelligent Systems and Computing, 2016, vol. 465, pp. 217-225 (in Russian).

[13]. Danilov N.A., Shulga T.E. Constructing a heat map based on the point data of the application user’s activity. Applied informatics, 2015, vol. 10, issue 2 (56), pp. 49-58 (in Russian).

[14]. Shulga T.E., Danilov N.A. Certificate of state registration of computer programs № 2014662094 "Software package for the collection and visualization of the desktop application user activity data», 2014, 6 October.

[15]. Shulga T. Je., Danilov N. A., Valjavskij V. A., Mitrofanov A. A., Murzabekov A. M. Certificate of state registration of computer programs №2015610568 «LEGO Turing machine», 2015, 13 January.

Активное обучение и краудсорсинг: обзор методов оптимизации разметки данных

[1,2]*Гилязев Р.А. <gilyazev@ispras.ru>*
[1,3,4]*Турдаков Д.Ю. <turdakov@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*
[2] *Московский физико-технический институт, 141700, Московская область, г. Долгопрудный, Институтский пер., 9*
[3]*Московский государственный университет имени М.В. Ломоносова, 119991 ГСП-1, Москва, Ленинские горы*
[4] *Национальный исследовательский университет Высшая школа экономики, 101000, Москва, ул. Мясницкая, д. 20*

Аннотация. Качественные аннотированные коллекции являются ключевым элементом при построении систем, использующих машинное обучение. В большинстве случаев создание таких коллекций предполагает привлечение к разметке данных людей, а сам процесс является дорогостоящим и утомительным для аннотаторов. Для оптимизации этого процесса был предложен ряд методов, использующих активное обучение и краудсорсинг. В статье приводится обзор существующих подходов, обсуждается их комбинированное применения, а также описываются существующие программные системы, предназначенные для упрощения процесса разметки данных.

Ключевые слова: активное обучение; краудсорсинг; аннотация корпусов; крауд-вычисления

DOI: 10.15514/ISPRAS-2018-30(2)-11

Для цитирования: Гилязев Р.А., Турдаков Д.Ю. Активное обучение и краудсорсинг: обзор методов оптимизации разметки данных. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 215-250. DOI: 10.15514/ISPRAS-2018-30(2)-11

1. Введение

Увеличение размера обучающей выборки позволяет улучшить точность и полноту большинства прикладных решений, основанных на методах машинного обучения с учителем. Однако для получения обучающих выборок в большинстве задач приходится прибегать к ручной разметке данных, что приводит к удорожанию конечных решений, особенно если необходимо привлекать экспертов в предметной области. Для оптимизации процесса

разметки было предложено несколько подходов, которые будут рассмотрены далее.

Одним из направлений работ, позволяющих оптимизировать процесс разметки, является активное обучение. Методы активного обучения направлены на поиск самых информативных примеров для классификатора. На каждой итерации из неразмеченного множества каким-либо алгоритмом выбирается один пример, он предоставляется оракулу (эксперту) на разметку и классификатор заново обучается на обновленном наборе тренировочных примеров. Подробный обзор методов предложен в работе [37]. Активное обучение – важная техника, позволяющая существенного снизить объем размеченных данных, что подкрепляется как экспериментами так и теоретическим обоснованием.

При активном обучении подразумевается, что в каждый момент времени разметку производит только один человек – эксперт в задаче, который и является оракулом. Но часто важно распараллелить нагрузку, что позволяет разметить больше примеров. В этом помогают краудсорсинговые платформы: Amazon Mechanical Turk (MTurk)[1], CrowdFlower[2] , Яндекс.Толока[3] и др. Их цель объединить работников и работодателей. Каждый участник платформы может выложить задание, например, в виде набора из нескольких примеров для разметки, другой участник или исполнитель, увидев интересное ему задание, выполняет его за определенную плату, существенно меньшую стоимости привлечения эксперта.

Основным недостатком такого подхода является низкий уровень качества выполненных заданий. Так Килгаррифф [18] выделил три причины некачественной разметки: неоднозначность данных, плохое руководство для аннотаторов и недостаток мотивации или знаний у аннотатора.

Тем не менее, исследования показывают, что польза от использования краудсорсинга есть, и результат, как правило, достигается при агрегации ответов нескольких аннотаторов на одном и том же примере. Например, С. Новак и др. [31] рассмотрели задачу многоклассовой разметки картинок с привлечением экспертов и обычных исполнителей, используя краудсорсинговую систему MTurk. Авторы, померив несколько статистик согласия, пришли к выводу, что при качественном руководстве для аннотаторов нет смысла в нескольких метках для объекта, если аннотаторами выступают эксперты. В противном случае, несмотря на неплохую точность разметчиков (каппа-статистика была на пороге допустимого), разметка одного задания несколькими разметчиками позволяет получить существенно более качественный датасет.

Район Сноу и др. [40] рассмотрели различные задачи текстовой классификации. Данные были размечены с привлечением экспертов и

[1] https://www.mturk.com/

[2] https://www.crowdflower.com/

[3] https://toloka.yandex.ru/

аннотаторов из MTurk. Измерив корреляцию Пирсона между и теми и другими авторы пришли к выводу, что привлеченные аннотаторы размечают данные заметно хуже экспертов, но их учет позволяет улучшить результаты, в частности, в некоторых задачах обычное усреднение меток не-экспертов сравнимо с результатами одного эксперта уже при 4 аннотаторах.

Похожие результаты получили и другие исследователи, которые дали ответ на вопрос – что важнее при построение тренировочного множества для классификатора: покрытие или качественная разметка [2, 20]. То есть существуют две опции: потратить все ресурсы на то, чтобы разметить как можно больше данных по одному разу, или разметить небольшое количество примеров, но каждый несколькими аннотаторами. Лучшей стратегией является выбор качественной разметки при низком качестве исполнителей, а при высоком – покрытия.

Краудсорсинг широко применяется при решении задач, не поддающихся автоматическим вычислениям и требующих человеческих усилий. На пути получения максимальной пользы при использовании краудплатформ встают три фактора. Первый из них – это качество, то есть нужны алгоритмы, которые наилучшим образом определят настоящие метки из имеющихся. При этом, естественно, необходимо помнить о стоимости разметки – решить задачу увеличением числа аннотаторов для одного примера не всегда разумно – это второй параметр. И, в-третьих, иногда первоочередным фактором является быстрое получение размеченного корпуса, тогда необходимо минимизировать временные задержки при выполнении участниками задания.

Большинство работ направлены на изучение первых двух факторов, и существующие решения обычно учитывают оба. Существующие работы можно сгруппировать несколькими способами. Возможные группировки представлены в табл. 1. Основой алгоритма вывода меток в краудсорсинге является модель аннотатора. Чаще всего это одно или несколько чисел, характеризующих его надежность или компетентность на каждом классе. В дополнение к этому иногда моделируется зависимость аннотаторов, например, марковской сетью. Реже – для каждого аннотатора обучается отдельный классификатор. В некоторых алгоритмах нет явной модели аннотатора, но различие между ними важно. В остальных случаях предполагается, что исполнители равнозначны.

Еще один вид группировки – это способ описания примера. Многие алгоритмы вывода не связаны напрямую с машинным обучением, а направлены лишь на качественную разметку, таким образом признаковое описание объектов встречается не часто. Тем не менее учитывать характеристики примера необходимо. Поэтому моделируются его сложность – чем она выше тем, меньше правильных ответов ожидается получить и распределение тем – вектор характеризующий к каким заранее определенным темам относится задание.

Табл. 1. Классификация алгоритмов вывода меток
Table 1. Classification of ground truth inference algorithms

Моделирование аннотатора	Уровень компетентности	[4][44]
	Матрица ошибок	[3][19][34][53]
	Параметры классификатора	[14] [12]
	Структура зависимости	[19][42]
	Вектор компетентности(в зависимости от темы задания)	[5][51]
	Неявное моделирование	[9][15]
Моделирование примера	Вектор признаков	[14][34][42][45]
	Сложность	[17][44]
	Распределение тем	[5][51]
Вывод меток	Голосование большинством	[4][11][16][40]
	Максимизация правдоподобия EM алгоритмом	[3][34][43][44][53]
	Итерационный процесс	[4][15][51]
	Вывод в графической модели	[19][25][42]
	Решение оптимизационной задачи	[14][41]
Инициализация параметров	На основе результатов разметки тестовых примеров	[4][11][16][17][40][51]
	Голосование большинством	[3][47][34]
	Случайная инициализация	[15][19][34][42]
Распределение заданий	Случайное	[3][14][34][44][47]
	Итеративное	[7][11][38][43][45][54]
	Онлайн	[5][12][20] [51][53]
Тестирование качества алгоритмов	Качество вывода меток	[3][8][9][25][51]
	Качество классификатора, обученного на крауд-данных	[12][14][34][42]

Следующий способ группировки – это алгоритм вывода меток. Самым простым является голосование большинством и его улучшения. В других алгоритмах учитывается модель аннотатора и вывод происходит более сложными путями.

Важным этапом является инициализация параметров модели, например, компетентностей аннотаторов. От этого во многом зависит точность решения. Для максимального использования ресурсов и применения активного обучения важно оптимальное распределение заданий.

Наконец, есть два способа тестирования качества алгоритмов: качество вывода меток и качество классификатора обученного на полученных данных.

Вопрос скорости получения размеченного корпуса не столь популярен у исследователей. В работе [10] выделяют три вида временных задержек.

Первая из них связана со временем, которое проходит с момента отправки задания до начала его выполнения. Сюда входит время на привлечение исполнителя, изучения им руководства для аннотаторов и, возможно, обучение (прохождение тренировочных заданий). Второй вид – это непосредственно выполнение задания. И третий, которого мы коснемся в вопросе об онлайн распределении заданий, – это задержки связанные с исполнением алгоритма генерирующего задания, например, время выполнения итерации активного обучения.

Активное обучение и краудсорсинг (под краудсорсингом здесь подразумевается любое распараллеливание процесса), пожалуй, единственные методы оптимизации процесса разметки. И возникает естественный вопрос – как их объединить, чтобы достичь большего результата? В частности, как в этом случае распределить задания между участниками?

Предметом этой работы является обзор методов разметки данных с использованием краудсорсинга и способов применения активного обучения, когда имеется несколько параллельно работающих исполнителей. Также задачей является исследование фреймворков реализующих эти методы.

Далее статья организована следующим образом. В следующем разделе обсуждаются существующие обзоры по релевантным темам. В разд. 3 освещается вопрос обеспечения качества в краудсорсинге. В четвертом разделе описаны способы распределения заданий между участниками и онлайн взаимодействие пользователей и системы. Разд. 5 посвящен существующим фреймворкам, которые обеспечивают работу с крауд-вычислениями. Разд. 6 содержит заключение.

2. Существующие обзоры

Тема выявления истинных меток возникла относительно недавно, вместе с первыми краудсорсинговыми платформами, и привлекает внимание исследователей до сих пор. Это отмечает А.В. Пономарев в обзоре [55]. В работе рассматриваются методы обеспечения качества в крауд-вычислениях. Под крауд-вычислениями здесь понимается более широкая область, чем разметка примеров для задач классификации, – по сути, это любая обработка информации. Автор провел аналитическое исследование проблемы и выделил широкий спектр существующих методов решения: методы согласования (consensus), методы проектирования потока работ, методы централизованного назначения работ, теоретико-игровые методы, методы, основанные на учете свойств заданий, и методы, основанные на анализе действий пользователя и воздействии на него.

Подробное описание методов краудсорсинга приведено в обзоре [48]. Глобально авторы выделяют два вида работ: те, которые никак не используют признаки объектов, и работы, вывод в которых так или иначе связан с моделью машинного обучения. В последнем случае рассмотрены и методы активного обучения, но не затрагивается проблема параллельной разметки.

Также проведена классификация методов по основным предположениям, использующимся в выводе.

В работе [23] произведена классификация каждого из типов моделей в краудсорсинге. Затронуты такие темы как: моделирование заданий, управление качеством, стоимостью и временными задержками. Особое внимание уделено крауд-операторам, которые охватывают практически все виды крауд-вычислений. Такими операторами являются: операторы выбора, сортировки, агрегации, сопоставления объектов и другие. Для каждого оператора приведены способы управления качеством. Также дан краткий обзор существующих крауд-платформ и фреймворков, упрощающих работу с ними, оперируя информацией как реляционной базой данных.

Смежной (или даже более общей чем выявление истины из крауд-данных) является тема выявления правды из нескольких источников. Последними могут выступать, например, новостные издания, сайты в сети или другие источники информации. Формально задача ставится так: есть множество источников S, множество объектов O и высказывания v_o^s, $o \in O$ – объект, к которому относится высказывание, $s \in S$ – источник. Для каждого объекта o есть истина v_o^*, у каждого источника есть надежность w_s. И, таким образом, имея множество объектов O и высказывания для них нужно найти истину, попутно выводя w_s.

В обзоре [24] методы решения такой задачи делят на три класса:

- итеративные: так как процесс вывода истинных меток и надежностей связаны, то часто сначала оценивают метки, а потом веса источников и так до сходимости;
- основанные на оптимизации, где в общем виде решается задача:
$$\arg\min_{v_o^*, w_s} \sum_{o \in O} \sum_{s \in S} w_s d(v_o^s, v_o^*)$$
где d - некоторая функция расстояния.
- основанные на вероятностных графических моделях.

Все эти методы похожи на методы выявления истинных меток (а некоторые совпадают), но большое внимание здесь уделяется зависимости источников между собой. Например, часто бывает, что одно издание копирует другое и дублироваться могут как ложные, так и истинные высказывания.

Многие исследователи описывают алгоритм вывода как итерационный процесс [23, 24]. Сначала инициализируются параметры модели, затем до сходимости повторяются два шага: вывод меток и переоценка параметров (см. Алгоритм 1).

Имеется несколько открытых реализаций и сравнений методов по данной теме. Одними из первых были системы BATC (2013) [13] и SQUARE (2013) [39]. Авторы обзора [48] реализовали более новые методы в системе CEKA (2015). Среди всех таких работ выделяется статья [52] со сравнением 17 алгоритмов и

подробным описанием исследуемых данных. Общей тенденцией являются неплохие результаты алгоритмов, основанных на EM и его модификациях.

К сожалению, во всех обзорах методы, учитывающие признаки примера (связанные с машинным обучением), не сравниваются, в виду дороговизны реалистичного эксперимента и малого количества подходящих датасетов.

3. Основные методы вывода меток в краудсорсинге

В этом разделе рассматриваются методы агрегации меток в предположении, что система никак не влияет на процесс распределения заданий и всегда, за исключением оговоренных случаев, анализ производится после сбора всех ответов исполнителей. Начнем с формальной постановки задачи. Будем следовать классической постановке, встречающейся во многих работах [48, 55].

Пусть имеется N объектов $x_1, ..., x_N$, каждый их которых принадлежит одному из J классов $\{1, ..., J\}$. Также имеется K аннотаторов, каждый из которых некоторым объектам поставил в соответствие класс. Т.е. дана матрица меток $y_i^j \in \{0, ..., J\}$, где $i \in \{1, ..., N\}, j \in \{1, ..., K\}$, класс 0 означает, что соответствующий аннотатор не предоставил метки для примера i. Задача состоит в том, чтобы имея множество меток $\{y_i^j\}_{j=1}^K$ для каждого примера i предсказать правильную метку y_i (здесь предполагается, что она существует и единственна), т. е. нужно минимизировать эмпирический риск:

$$R = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(y_i = \hat{y}_i)$$

Здесь $\hat{y}_i$ – предсказание метки, а $\mathbb{1}(x)$ – индикаторная функция, принимающая значение 1 при истинном агрументе x и 0 – в противном случае.

Очевидным решением является голосование большинством: для каждого примера выбирают метку, которая встречается чаще всего для данного примера.

Этот способ еще называют мажоритарным голосованием (majority voting (MV)). Предположим, что $2k + 1$ аннотаторов предоставили метку для некоторого примера i, причем каждый ставит правильную метку с одинаковой точностью p. Тогда вероятность того, что при мажоритарном голосовании получится правильный ответ:

$$\sum_{j=1}^{k} \binom{2k+1}{j} p^{2k+1-i} (1-p)^i$$

Эта формула часто используется для оценки необходимого числа меток, чтобы получить заданную точность [26].

Интересно вспомнить знаменитую теорему Кондорсье о жюри присяжных, в которой утверждается, что если число присяжных(аннотаторов) стремится к бесконечности, то при $p > 0.5$ вероятность выбрать правильный ответ стремится к 1, а при $p < 0.5$ к 0.

Как уже было сказано, краудсорсингу присущ шум, и метки получаются от людей с разным уровнем компетентности и опыта. Поэтому такой подход не всегда хорош, поскольку каждый участник вносит одинаковый вклад в итоговый ответ. Логичным кажется модифицировать метод, добавив вес w_j каждому аннотатору как уровень его надежности, который отражает вероятность предоставить правильную метку для произвольного примера, а затем проводить голосование с весами:

$$y_i = \arg\max_{c \in [1,\dots,J]} \left(\sum_{j=1}^{K} w_j \mathbb{1}(y_i^j = c) \right)$$

В [50] детально рассмотрены алгоритмы голосования, если для каждого аннотатора известны q_j – вероятности предоставления правильного ответа, и доказано, что оптимальным является байесовское голосование, где вероятность класса пропорциональна его правдоподобию:

$$Pr(y_i = c) \propto \prod_{j=1}^{K} q_j^{\mathbb{1}(y_{ij}=c)} \left(\frac{1-q_j}{J-1} \right)^{\mathbb{1}(y_i^j \neq c)}$$

Нормировка позволяет также оценить вероятности классов.

Оценка характеристики аннотатора, например, значений q_j, является основной задачей. Часто для её нахождения используют тестовое множество примеров, с заранее известными ответами [11, 16, 40]. Иногда ненадежных аннотаторов вообще исключают из системы [21]. В системе ZenCrowds [4] привлекали разметчиков для решения задачи связывания именованных сущностей (entity linking). Аннотатор j описывался одним числом q_j – долей правильных ответов. На старте этот параметр получался из результатов разметки тестового множества, если такового не было, то полагалось $q_j = 0.5$. Затем до сходимости проводились итерации из двух шагов. Для каждого примера метка определялась взвешенным голосованием среди надежных аннотаторов, для этого выбирался порог надежности. После веса q_j считались заново в предположении, что полученные до этого метки истинны. В общем виде такая процедура описывается Алгоритмом 1.

input: матрица ответов L, L_{ij} - ответ участника с номером j на примере i
output: метки $\hat{\mathbf{y}} = [\hat{y}_1, ..., \hat{y}_N]$ и качество участников $\mathbf{q} = [\hat{q}_1, ..., \hat{q}_K]$
1: Инициализировать $\mathbf{q}$
2: Выполнять до сходимости:
3: for i:=1 to N do:
4: Оценить $\hat{y}_i$ на основе $\mathbf{q}$ и L
5: for j:=1 to K do:
6: Оценить q_j на основе $\hat{\mathbf{y}}$ и L

Алгоритм 1. Итеративный вывод
Algorithm 1. Iterative ground truth inference

Но этот подход не решает проблему того, что надежность может меняться в зависимости от данных или от истинной метки, к тому же привлечение экспертов для разметки теста вызывает дополнительные траты бюджета. Веса надежности являются лишь частью сложной модели, но тем не менее эта идея является ключевой при выводе истинных меток.

Дэвид и Скини [3] рассматривают надежность в более широком смысле. Они предложили для каждого аннотатора вычислять матрицу ошибок: $\pi_{qj}^{(k)}$ – вероятность того, что аннотатор k поставит метку j, если настоящей меткой является q. Теперь аннотатор характеризуется только матрицей ошибок. Обозначим p_{ij} априорное распределение классов для примера i.

Пусть $n_{il}^{(k)}$ – количество раз, которое аннотатор поставил метку l примеру i (предполагалось, что участник мог разметить один и тот же пример несколько раз). Тогда правдоподобие вероятностной модели запишется так:

$$\prod_{i=1}^{N}\left(\sum_{j=1}^{J} p_{ij}\prod_{k=1}^{K}\prod_{l=1}^{J}(\pi_{jl}^{(k)})^{n_{il}^{(k)}}\right)$$

Здесь использовались два важных предположения:

1. метка аннотатора не зависит от примера, а зависит только от истинной метки;
2. аннотаторы предоставляют метки независимо друг от друга.

Для решения задачи используют EM-алгоритм для нахождения параметров, максимизирующих правдоподобие. Итерации происходят следующим образом: при фиксированных $\pi_{qj}^{(k)}$ оцениваются вероятности классов, а затем при фиксированных p_{ij} находятся матрицы ошибок, максимизирующие правдоподобие.

Такую модель называют DS (анаграмма первых букв авторов). Очевидны недостатки модели: EM-алгоритм не гарантирует сходимость к оптимальному решению; необходимо правильно выбрать начальное параметры $\pi_{qj}^{(k)}$; нигде не используется сам пример x_i (в модели авторов не было признаков). И конечно, предположения 1 и 2 не всегда оправданы.

Эта идея в дальнейшем развивалась многими авторами и породила целый класс алгоритмов, которые так или иначе рассматривают модель аннотатора и часто решают задачу путем применения EM-алгоритма.

В работе Жанга и др. [49] для инициализации матрицы ошибок используют оригинальный подход, основанный на спектральном методе. Аннотаторы разбиваются на три группы, и для каждой группы вычисляются усредненные ответы исполнителей. Затем методом моментов оцениваются матрицы ошибок для каждой группы, как будто бы имеется всего 3 участника. На основе полученных оценок находятся начальные приближения для всех аннотаторов.

В алгоритме, предложенном Райкаром и др. [34], рассматривается задача с двумя классами. В таком случае модель аннотатора записывается двумя параметрами:

- чувствительность $\alpha^j = Pr[y^j = 1|y = 1]$ – вероятность того, что аннотатор j верно определит положительный класс и
- специфичность $\beta^j = Pr[y^j = 0|y = 0]$ – вероятность верно определить отрицательный класс.

В оригинальной модели (DS) нигде не использовался сам пример x_i. В работе [34] этот недостаток устраняется введением модели логистической регрессии:

$$Pr[y = 1|\mathbf{x}, \mathbf{w}] = \sigma(\mathbf{w}^T\mathbf{x})$$

где $\sigma(z) = \frac{1}{1+e^{-z}}$. Теперь метка зависит не только от векторов $\boldsymbol{\alpha}$ и $\boldsymbol{\beta}$, но и от $\mathbf{x}$ и вектора весов $\mathbf{w}$, и правдоподобие запишется следующим образом:

$$Pr[D|\theta] = \prod_{i=1}^{N} Pr[y_i^1, ..., y_i^K|\mathbf{x}_i, \boldsymbol{\alpha}, \boldsymbol{\beta}, \mathbf{w}]$$

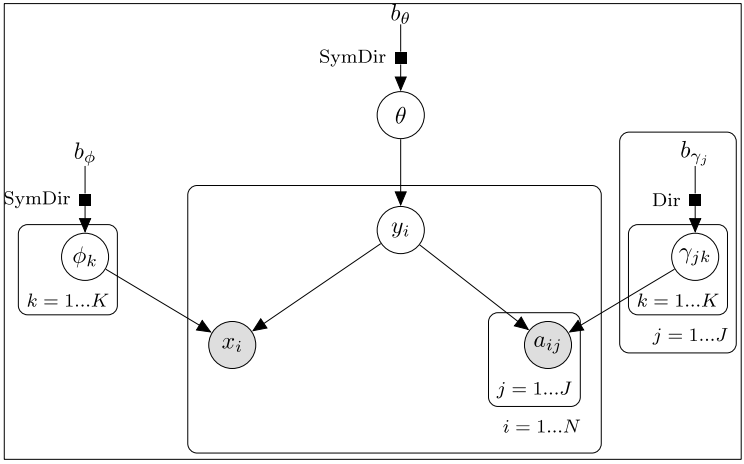


Рис. 1. Графическая модель вывода меток. Закрашенные ячейки соответствуют наблюдаемым переменным

Fig. 1. Graphical model for inferring ground truth labels. Shaded cells correspond to observable variables

$$\forall i : y_i|\boldsymbol{\theta} \sim Categorical(\boldsymbol{\theta})$$

$$\forall i : x_i|y_i, \boldsymbol{\psi} \sim Multinomial(|x_i|_1, \boldsymbol{\psi}_{\boldsymbol{y_i}})$$

$$\forall i : x_i|y_i, \boldsymbol{\psi} \sim Multinomial(|x_i|_1, \boldsymbol{\psi}_{\boldsymbol{y_i}})$$

Дальше выражение преобразуется с учетом предположений (1) и (2), и его логарифм представляется следующим образом:

$$ln(Pr[D|\theta]) = \sum_{i=1}^{n} y_i ln(p_i) a_i + (1 - y_i) ln(1 - p_i) b_i$$

где

$$p_i = \sigma(\mathbf{w}^T \mathbf{x}_i)$$
$$a_i = \prod_{j=1}^{K} [\alpha_j]^{y_i^j} [1 - \alpha_j]^{1 - y_i^j}$$
$$b_i = \prod_{j=1}^{K} [\beta_j]^{1 - y_i^j} [1 - \beta_j]^{y_i^j}$$

Максимум находится EM-алгоритмом: на E шаге оцениваются y_i, на M шаге значения α_i и β_i можно вычислить явно, а $\mathbf{w}$ находится градиентным спуском.

Для инициализации y_i используется мажоритарное голосование. Также отмечается, что алгоритм будет работать, если не все аннотаторы предоставили метки каждому примеру, и метод можно обобщить на любой вероятностный классификатор. Тестирование на наборе из нескольких реальных задач показывает, что метод превосходит мажоритарное голосование.

Однако, с другими методами сравнение не проводилось. Более детально метод и его расширения, включая задачу регрессии, описаны в поздней статье [35].

Важным дополнением является байесовское расширение метода. Если заранее имеются какие-либо предпочтения к аннотаторам, то авторы предполагают, что параметры $\boldsymbol{\alpha}$ и $\boldsymbol{\beta}$ принадлежат Beta распределению. Затем вместо ML ищется MAP оценка. Похожий подход встречается во многих работах. В общем случае вероятностные предположения накладываются на все переменные и система описывается более сложной графической моделью, вывод в которой уже не удается произвести EM-алгоритмом. На помощь приходят методы Монте Карло по схеме марковской цепи (Markov Chain Monte Carlo, MCMC).

Так, например, в работе [19] вводятся дополнительные параметры (предполагается, что матрица ошибок и распределение классов принадлежат распределению Дирихле);

$$\pi_j^{(k)} \sim Dir(\alpha_{j,1}^{(k)}, ..., \alpha_{j,J}^{(k)})$$
$$\mathbf{p} \sim Dir(\nu_1, ..., \nu_J)$$

Сами параметры $\alpha_{j,l}^{(k)}$ порождены экспоненциальным распределением $Exp(\lambda_{j,l}^{(k)})$. Тогда апостериорное распределение при условии независимости аннотаторов будет выглядеть так:

$$P(\boldsymbol{p}, \pi, y, \alpha | c) = \prod_{i=1}^{N} (p_{y_i} \prod_{j=1}^{M} \pi_{y_i, c_i^{(k)}}^{(k)}) p(\boldsymbol{p}|\nu) p(\pi|\alpha) p(\alpha|\lambda)$$

Вывод производится итерационно по схеме Гиббса. Также рассматривается случай зависимости участников, которая моделируется марковской сетью. Графическая модель из работы [42] учитывает признаки объекта. Имеется конечный набор факторов – линейных весов, соответствующих вектору признаков. Аннотатор характеризуется бинарной суммой этих факторов, а вероятность предоставить положительный класс описывается пробит-регрессией. В работе [8] введены параметры для моделирования распределения примеров в случае задачи текстовой классификации. Схема модели представлена на рисунке 1.

Интересный подход предложен Каргером и др. [15]. Рассматривается задача с двумя классами: 1 и -1. Также предполагается истинность допущений 1 и 2. Далее строится случайный (l, r) – регулярный двудольный граф $G(\{t_i\}_{i=1}^{m} \cup \{w_j\}_{j=1}^{n}, E)$, l – число вопросов для каждого примера, r – количество примеров которые размечает каждый аннотатор, число аннотаторов n определяется из $lm = rn$. Этот граф определяет распределение заданий для аннотаторов.

Вывод производится итеративно. Используются два типа сообщений $x_{i\to j}$, $y_{j\to i}$, где $(i,j) \in E$. $y_{j\to i}^{(0)}$ инициализируются случайно из нормального распределения $\mathcal{N}(1,1)$. Затем для заданного k_{max} и полученных меток $\{L_{ij}\}_{(i,j)\in E}$ выполняется k_{max} шагов:

- для всех $(i,j) \in E$:

$$x_{i\to j}^{k} \leftarrow \sum_{j' \in \delta(i)\setminus j} L_{ij'} y_{j'\to i}^{(k-1)}$$

- для всех $(i,j) \in E$:

$$y_{j\to i}^{k} \leftarrow \sum_{i' \in \delta(j)\setminus i} L_{i'j} x_{i'\to j}^{(k)}$$

$\delta(u)$ – соседи вершины u. Итоговые метки:

$$x_i = \sum_{j\in\delta(i)} L_{ij} y_{j\to i}^{(k_{max}-1)}$$

Значения $x_{i\to j}$, $y_{j\to i}$ легко интерпретируются: $x_{i\to j}$ – метка для примера i, полученная голосованием всех аннотаторов, кроме j-го, а $y_{j\to i}$ – надежность участника j, найденная без учета его предсказания для i-го примера. Авторы приводят теоретические обоснования корректности метода и асимптотики сходимости. Сравнения с мажоритарным голосованием и DS доказывают эффективность метода на синтетических данных. Однако в работе Лиу и др. [25] подчеркивается, что метод сложно обобщить на различные модели аннотатора, и нет уверенности в работе на реальных данных. Был предложен более общий подход, основанный на алгоритме распространения доверия.

Каргер утверждает, что его метод похож на алгоритм поиска собственного вектора матрицы LL^T. Разница только в том, что здесь одно из слагаемых не учитывается.

Гош и др. [9] предложили метод, полностью основанный на вычислении собственного вектора LL^T. Мотивация подхода следующая. Пусть есть два класса 1 и -1. Участник i размечает верно с вероятностью q_j, $\boldsymbol{y}$ – столбец настоящих меток, и A – матрица разметок. Тогда легко посчитать, что $E(L) = \boldsymbol{y}(2\boldsymbol{q} - 1)^T$, а $E(LL^T) = \kappa \boldsymbol{y}\boldsymbol{y}^T + (n - \kappa)I$, где $\kappa = \sum_j (2q_j - 1)$, I – единичная матрица. Максимальное собственное значение матрицы $E(LL^T)$ есть $\kappa||\boldsymbol{y}||^2 + (n - \kappa)$, а $\boldsymbol{y}$ – собственный вектор. Таким образом, в качестве вектора предсказаний алгоритм возвращает собственный вектор LL^T, соответствующий максимальному собственному значению.

Заметим, что не все подходы учитывают вектор признаков примера: сложно объединить модель вывода меток и какой-либо алгоритм машинного обучения. Тем не менее, находятся и другие способы различать объекты. Так, улучшить точность агрегации меток позволяют предположения о сложности примера.

В работе Уайтхилла и др. [44] решалась задача классификации картинок на 2 группы. Здесь параметр $1/\beta_i \in [0, +\infty)$, где $1/\beta_i = +\infty$ говорит о том, что картинка настолько сложная, что даже эксперт разметит ее правильно с вероятностью $1/2$, а $1/\beta_i = 0$ – что любой разметчик поставит этой картинке верный класс. Параметр $\alpha_j \in (-\infty, +\infty)$ отвечает за надежность аннотатора, $\alpha = +\infty$ соответствует идеальному разметчику, а $\alpha = -\infty$ – аннотатору, который всегда дает неправильный класс, 0 соответствует случайному выбору. Вероятность того, что метка l_{ij} для примера i от аннотатора j истинная, определяется как сигмоида $\sigma(\alpha_j \beta_i)$:

$$Pr[l_{ij} = y_i | \alpha_j, \beta_i] = \frac{1}{1 + e^{-\alpha_j \beta_i}}$$

Неизвестные параметры находятся все тем же EM-алгоритмом. Авторы назвали систему GLAD.

Похожим образом параметр сложности примера используется в системе ELICE [17].

Для начальной оценки параметров используются достоверно известные метки для n объектов корпуса. По этим меткам оцениваются надежности аннотаторов как разница числа верно и неверно размеченных примеров

$$\alpha_j = \frac{1}{n} \sum_{i=1}^{n} [\mathbb{1}(y_i = y_i^j) - \mathbb{1}(y_i \neq y_i^j)]$$

и сложность примера:

$$\beta_i = \frac{1}{M} \sum_{j=1}^{M} [\mathbb{1}(y_i = y_i^j)]$$

Для неразмеченных объектов метка сначала оценивается мажоритарным голосованием с весами α_j из первого шага. Затем полученные метки используются для оценки параметра β_i, и, наконец, итоговая метка получается так:

$$F_i = sign[\frac{1}{M}\sum_{j=1}^{M}\sigma(\alpha_j\beta_i)y_i^j]$$

Эксперименты с симуляцией разных типов разметчиков показали, что алгоритм более устойчив к большому числу шумных меток, эффективно работая когда всего 20% аннотаторов работают качественно. При этом экспертам понадобилось разметить всего 20 объектов. Подход применим только для двухклассовой задачи.

Далее была предложена модификация метода ELICE-2 [16], в которой извлекается польза от оппозиционных участников, заведомо неправильно выполняющих аннотации. Алгоритм похож на предыдущую версию, только параметры α_j и β_i домножаются на $(1 - E(p))$, где $E(p) = -plog(p) - (1-p)log(1-p)$ – энтропия доли настоящих меток для аннотатора или примера соответственно. Ясно, что случайные разметчики будут иметь высокую энтропию, хорошие и оппозиционные – низкую, но надежность последних так же, как и в ELICE, будет отличаться знаком. Финальная формула теперь учитывает неправильные метки оппозиционных разметчиков:

$$F_i = sign[\frac{1}{M}\sum_{j=1}^{M}\sigma(|c\alpha_j\beta_i|) * l_{ij} * sign(\alpha_j\beta_i)]$$

В системах DOCS [51] и CDAS [26] раскрывается идея того, что компетентность исполнителя связана с темой задания. В DOCS для каждого задания (примера) вводится вектор доменов – распределение по выделенным темам, а каждый исполнитель описывается набором чисел – компетентностью на каждом домене. Вывод происходит по классическому итерационному сценарию, с учетом доменов. В CDAS строится граф похожести заданий: если аннотатор хорошо справился с определенным заданием, то скорее всего он справится и с аналогичным. Идеи авторов систем DOCS и CDAS описываются подробнее в дальнейших разделах.

Имеются несколько подходов, которые строят классификатор явно, не производя вывод меток. Шенг и др. [38] предложили учитывать все собранные метки. Если для примера i есть L_i меток, то из него получается L_i обучающих примеров: каждой со своей меткой и примеру присваивается вес $1/L_i$, который обрабатывается классификатором.

Каджино [14] предложил алгоритм, который обобщает логистическую регрессию на случай нескольких аннотаторов. Для простоты рассматриваются два класса. Общая модель описывается как $\sigma(\boldsymbol{w_o}^T\boldsymbol{x})$. Для каждого аннотатора j строится своя модель: $\sigma(\boldsymbol{w}_j^T\boldsymbol{x})$, где параметр $\boldsymbol{w}_j$ рассматривается как

отклонение от общей модели, что выражается следующим априорным распределением:

$$Pr[\boldsymbol{w}_0|\eta] \sim \mathcal{N}(0, \eta^{-1}I)$$
$$Pr[\boldsymbol{w}_j|\boldsymbol{w}_0, \lambda] \sim \mathcal{N}(\boldsymbol{w}_0, \lambda^{-1}I)$$

Затем, как обычно, максимизируется логарифм апостериорного распределения, что эквивалентно такой функции ошибки:

$$-\sum_{j=1}^{J}\sum_{i\in I_j} l(y_i^j, \sigma(\boldsymbol{w}_j^T\boldsymbol{x}_i)) + \lambda/2\sum_{j=1}^{J}||\boldsymbol{w}_j - \boldsymbol{w}_0||^2 + \eta/2||\boldsymbol{w}_0||^2$$

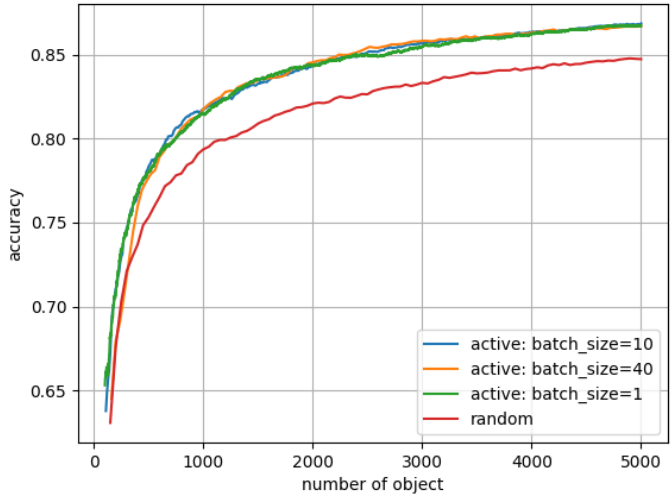
$l(y,p) = -ylog(p) - (1-y)log(1-p)$ – кросс-энтропия. Оптимизация происходит следующим образом. Если даны $\boldsymbol{w}_j$, то $\boldsymbol{w}_0$ выписывается аналитически. Оптимизация по $\boldsymbol{w}_j$ независима и делается по отдельности. То есть для каждого $\boldsymbol{w}_j$ делается шаг оптимизации, затем считается $\boldsymbol{w}_0$.

В конце разлела коснемся практической стороны темы. Сравнения [13, 48, 52] наиболее популярных методов показывают, что нет явного лидера: при двух классах неплохо работают DS [3] и его модификации (RY [34]), а сложные модели с большим количеством параметров не всегда применимы. Иногда совсем простые методы бывают эффективны. Например, метод [47], основанный на кластеризации, оказался успешней остальных в сравнении [48] на многоклассовых задачах. Каждый пример здесь описывается вектором размерности $|J|$ – число классов. Каждая компонента – количество меток соответствующего класса. Ответ получается кластеризацией этих векторов на $|J|$ классов алгоритмом к-средних, центрами кластеров инициализируются те примеры, для которых максимально число голосов за данный класс.

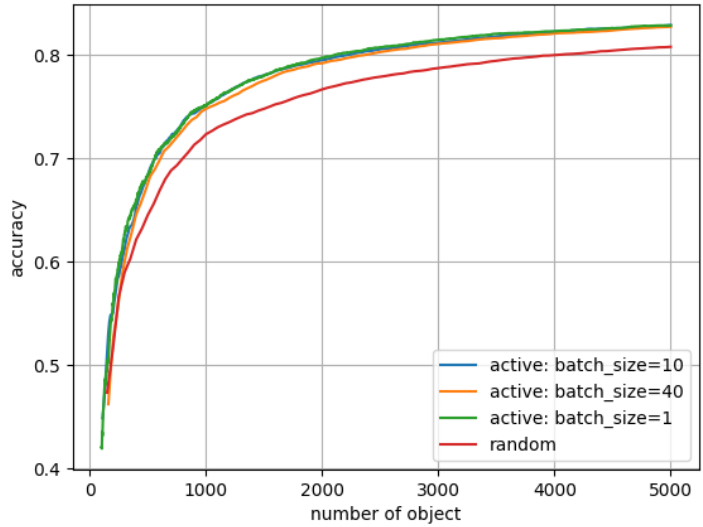
4. Распределение заданий

Как уже было сказано, почти все методы предыдущего раздела предполагали простейший сценарий распределения заданий. Заранее выбиралось множество объектов, разметку которых мы хотим получить. Затем задания, сформированные по несколько примеров в каждом, отправлялись на краудплатформу, каждое по несколько раз. После выполнения всех заданий результаты обрабатывались тем или иным алгоритмом. Другими словами, разметка происходила в оффлайн режиме.

В этом разделе будут рассмотрены различные варианты распределения заданий, в которых разметка происходит в итерационном режиме, т.е. анализ производится не после аннотации всех объектов, а после каждой итерации разметки. Такой способ позволяет эффективно использовать доступные ресурсы, в частности, применить активный выбор примеров и аннотаторов.



(a) IMDB, 2 класса
(a) IMDB, 2 classes



(b) Lenta.ru, 8 классов
(b) Lenta.ru, 8 classes

Рис. 2: Влияние размера пакета на качество: (a) Анализ тональности отзывов о фильмах с сайта IMDB, 2 класса; (b) Классификация новостей с сайта lenta.ru на 8 классов

Fig. 2. Tradeoff between batch size and model quality: (a) Sentiment analysis of movies review from IMDB, 2 classes; (b) News classification from lenta.ru, 8 classes

Классическое активное обучение предполагает, что примеры поставляются на разметку по одному. Эффективность в данном случае подкрепляется также теоретическими обоснованиями.

Но на практике это долго: каждый раз нужно переобучать классификатор и неясно как быть если метки приходят из краудсорсинга (отправлять задания с одним примером было бы затратно). Поэтому нередко генерируют не один пример, а пакет, состоящий из нескольких заданий. Существующие эксперименты подтверждают разумность этого шага. Например, в [17] размеры пакета 10–40 считаются приемлемыми.

То же показывают и наши эксперименты на нескольких задачах текстовой классификации. Вместо привлечения аннотаторов использовались датасеты с заранее известными истинными метками, таким образом происходила симуляция активного обучения. В качестве алгоритма активного обучения был выбран один из наиболее простых и популярных методов – для пополнения обучающего множества выбирались те примеры, на которых вероятностный классификатор был наименее уверен, а именно, с наименьшей разностью вероятностей двух наиболее популярных предсказанных классов. Классификатор – логистическая регрессия, признаки – «мешок» слов. В качестве датасетов были выбраны следующие:

- анализ тональности отзывов с двумя классами, IMDB, датасет из 25 тыс. примеров;
- классификация новостей с сайта Lenta.ru на 8 групп, около 500 тыс. примеров

В обоих случаях классы сбалансированы. На рис. 2 приведены результаты. Показано изменение метрики точности от числа примеров в обучающем множестве в зависимости от режима: активное обучение с различными размерами пакета и случайный выбор. Результаты усреднены по нескольким (трем) запускам итераций. Графики показывают, что использование активного обучения дает прирост в метрике качества по сравнению со случайной разметкой, и при размере пакета 10 получается результат, сравнимый с классическим подходом, где размер пакета 1.

Существующие подходы к пакетному распределению заданий можно разделить на две группы. Первая – итерационное планирование: планировщик после обработки очередной порции примеров, выбирает подходящие объекты или аннотатора и отправляет задание на крауд-платформу. Во второй группе работ, исполнитель сам сообщает о своей готовности выполнить разметку, и система поставляет ему примеры в онлайн режиме. В случае одного аннотатора оба подхода соответствуют традиционному активному обучению. Интерес представляет параллельная работа нескольких исполнителей.

4.1 Итеративная разметка

Наиболее простым подходом является варьирование числа меток для каждого примера в зависимости от его сложности. Так, в [20] число аннотаторов на один пример определяется динамически: после того как пример размечен несколько раз, подсчитывается согласие аннотаторов на нём. Разметка происходит до тех пор, пока не будет достигнут консенсус или не будет превышен допустимый порог числа повторений. В [29] это число для каждого примера считается заранее: все объекты кластеризуются, представители кластеров размечаются в тестовом режиме, на основе согласия на выбранных примерах итоговое число определяется для каждого кластера.

Более результативным является применение активного обучения в том или ином виде. В одной из первых работ [38] на эту тему примеры для разметки выбираются активно, а все собранные метки обрабатываются классификатором. Для итеративного выбора примера предлагается несколько эвристик. Первая считает неопределенность на основе имеющихся меток для примера – в простом случае это может быть энтропия меток. Вторая использует какой-либо алгоритм активного обучения. Эти эвристики объединяются подсчетом среднего геометрического неопределенностей полученных в обоих методах.

Большинство же работ, обсуждаемых в этом разделе, направлены на выбор аннотатора и примера либо одновременно, либо сначала примера, а потом аннотатора. То есть новое задание выдается конкретному человеку, который вероятно справится с ним лучше остальных.

Одной из таких работ является [45]. Метка аннотатора t на примере x_i моделируется нормальным распределением с центром в истинном классе:

$$p(y_i^{(t)}; x_i, z_i) = \mathcal{N}(y_i^{(t)}; y_i, \sigma_t(x_i));$$

y_i – истинная метка.

$$\sigma_t(x_i) = \frac{1}{1 + exp(-\boldsymbol{w}_t^T \boldsymbol{x}_i - \gamma_t)}.$$

Вероятность положительного класса рассматривается как логистическая регрессия:

$$p(z = 1|x_i) = \frac{1}{1 + exp(-\boldsymbol{\alpha}^T \boldsymbol{x}_i - \beta)}$$

Параметры аннотаторов $\boldsymbol{w}_i^T$, γ_t и веса регрессии $\boldsymbol{\alpha}$, β находятся поиском оценки максимального правдоподобия EM-алгоритмом.

Далее выбираются примеры, для которых текущая модель не уверена:

$$\arg\min_x (0.5 - p(y|x))^2$$

Решением является гиперплоскость $\boldsymbol{\alpha}^T \boldsymbol{x}_i + \beta = 0$. Затем ищется аннотатор с наименьшей дисперсией $\sigma_t(x)$. В итоге получается такая задача оптимизации:

$$\min_{x,p}(C(\boldsymbol{\alpha}^T\boldsymbol{x}+\beta)+p^T[\boldsymbol{w}_1,...,\boldsymbol{w}_T]x+p^T\gamma)$$

$p=[p_1,...,p_T]$ – распределение аннотаторов, $\sum_{i=1}^{T}p_i=1$, $\gamma=[\gamma_1,...,\gamma_T]$.

Таким образом, одновременно находятся и пример x^*, и аннотатор к нему. Конечно, x^* может не оказаться среди данных. Тогда выбирается ближайший к нему по Евклидовой метрике.

Нетрудно выделить общую схему алгоритмов активного обучения с несколькими аннотаторами. Сначала из неразмеченного множества выбирается самый информативный пример:

$$x_{i+1}=\arg\min_{x\in X_u}A(x)$$

Затем выбирается подходящий аннотатор, который больше всего уверен в нем:

$$t_{i+1}=\arg\max_{t\in T}Q_t(x_{i+1})$$

И в конце цикла все необходимые параметры переоцениваются.

Следовательно, нужно подобрать функции неуверенности классификатора $A(x)$ и надежности исполнителей $Q_t(x)$; при этом желательно, чтобы классификатор и аннотаторы были связаны. В качестве A и Q в работе [7] используется энтропия предсказаний классов и линейная комбинация признаков соответственно, в работе [54] – SVM с радиальными базисными функциями в обоих случаях.

Одним из недостатков таких подходов является то, что модель склонна отдавать предпочтения одним и тем же исполнителям. Зависимость модели от одного аннотатора является нежелательным эффектом: предсказания алгоритма будут смещены, и в какой-то момент модель начнет считать все метки этого исполнителя истинными. К тому же у других участников пропадает возможность проявить себя.

Родригес и др. [36] предложили для оценки характеристики исполнителя использовать два параметра: α_j – чувствительность и β_j – специфичность, а для разметки выбирать исполнителя с наибольшим ожиданием предоставить правильный ответ:

$$j^*=\arg\max_j[\alpha_j p(y=1|x^*,L,Y)+\beta_j(1-p(y=1|x^*,L,Y))]$$

L – множество размеченных примеров, Y – ответы исполнителей. В качестве $p(y=1|x,L,Y)$ используется адаптированная к нескольким аннотаторам модель гауссовского классификатора [33]. Пример x^* выбирается активным обучением. Поскольку α_j и β_j вычисляются исходя из оцененных меток, предлагается не учитывать метки аннотатора при оценке его параметров. Например, если чувствительность вычисляется следующим образом:

$$\alpha_j=\frac{\sum_{i=1}^{N}y_i^j p(y_i=1|L,Y)}{\sum_{i=1}^{N}p(y_i=1|L,Y)}$$

то вероятности $p(y_i = 1|L, Y)$ заменяются на $p(y_i = 1|L\backslash L^j, Y\backslash Y^j)$. Таким образом решается проблема зависимости от одного исполнителя.

```
L, U - множество размеченных и неразмеченных примеров
A - множество аннотаторов
Iter - число итераций
1: for i:=1 to Iter do:
     // Выбрать пример для разметки из множества U с учетом имеющихся ответов в L
2:   x = U.sample(L)
     // Выбрать аннотатора для разметки x, на основе имеющихся ответов в L
3:   j = A.choice(x, L)
     // Получить метку для x
4:   y = get_answer(j, x)
5:   L.Update(x, y)
```

Алгоритм 2. Активное обучение с выбором аннотатора
Algorithm 2. Active learning with annotator selection

Иногда выбираются сразу нескольких аннотаторов, или аннотаторы выбираются случайно – пропорционально надежности. Некоторые алгоритмы рассчитаны на возможность привлечения специалистов: если согласие исполнителей низкое, то пример может быть предоставлен на экспертную оценку [11, 30], а в работе [46] после выбора примера сразу решается, следует ли привлечь эксперта или же обычного исполнителя к его разметке.

Велиндер и Перона [43] реализовали онлайн версию EM-алгоритма. Аннотатор описывается вероятностью предоставления правильного ответа на каждом классе. Далее вводится несколько типов аннотаторов: E – эксперты, B – множество аннотаторов, дающих некачественные ответы, и остальные. Итерации происходят следующим образом: очередной пример предоставляется на разметку аннотатору из множества E (если оно пусто, то любому аннотатору не из B). Затем оцениваются $p(y)$ – апостериорное распределение классов данного объекта, что соответствует E шагу. Процедура продолжается до тех пор, пока выполняется условие $\max_y p(y) < \tau$, где τ – порог. То есть разметка продолжается до тех пор, пока нет уверенности в какой-либо метке, либо пока не достигнуто максимальное число шагов. После заново оцениваются параметры исполнителей (M шаг), и на основе этого пересчитываются множества E и B.

Отметим, что онлайн выбор аннотатора в некотором смысле относится к задаче многоруких бандитов, но отклик (награда) не может быть вычислен явно – сложно определить полезность исполнителя по одному ответу. Поэтому приходится использовать различные эвристики. Так, в [27] модель в каждый момент времени находится в одном из двух режимов: исследование (exploration) – оценка компетентности исполнителей или использование (exploitation).

Пусть $E(t)$ – множество примеров, которые были размечены в первом режиме до момента времени t. Если $|E(t)| < D_1 log(t)$ или имеется пример $x_k \in E(t)$, размеченный меньше чем $D_2 log(t)$ раз, то в момент $t+1$ на разметку всем исполнителям предоставляется либо новый пример, либо x_k соответственно. Затем взвешенным голосованием переоцениваются истинные метки и оцениваются надежности исполнителей. Это режим исследования. В режиме использования случайно выбирается новый пример и предоставляется самым надежным участникам.

В работе [41] предложен подход к решению задачи привлечения фриланс-работников с неизвестным рейтингом и разной стоимостью. В этом случае отклик оценивается как качество проделанной работы. Отличие от задачи многоруких бандитов состоит в ограничении на количество выполненных заданий одним участником (может физически не хватить времени) и также в том, что за один раз можно дать одно и то же задание многим людям (дернуть за несколько ручек).

Формально задача в работе [41] ставится так: даны бюджет B и участники со стоимостью одного задания c_i, ограничением на количество заданий l_i и неизвестным распределением полезности. Необходимо распределить задания так, чтобы максимизировать сумму полезностей при ограничении на бюджет.

Решение делится на 2 фазы. На первом этапе выбирается такое ϵ, что, пока это возможно, тратится ϵB бюджета. Участники упорядочиваются по возрастанию c_i и по циклу получают задания. За задание выставляется оценка. Для каждого исполнителя оценивается математическое ожидание полезности $\hat{\mu}_i$ – среднее оценок. На втором этапе при найденной полезности $\hat{\mu}_i$, стоимости c_i, бюджете $(1-\epsilon)B$ и ограничениях l_i эвристическими методами решается известная в теории сложности вычислений задача о рюкзаке.

4.2 Онлайн разметка

С точки зрения организации процесса разметки итеративное планирование неэффективно использует доступные ресурсы. Нужно ждать, пока конкретный исполнитель выполнит задание, нет возможности загрузить работой свободных аннотаторов, и тем самым параллельная разметка затруднительна. Если разметку выполняют не участники крауд-платформ, а заинтересованные люди с высокой компетентностью, оптимизация процесса становится критичной.

Хотелось бы, чтобы планирование выглядело так. Скажем, каждый участник готов уделить аннотированию несколько минут в день. Он в удобное ему время заходит в систему и выполняет задание в режиме онлайн: система выдает по одному или по несколько примеров, обрабатывает ответы и с минимальной задержкой предоставляет следующее задание. Под такие критерии попадает простейший случай разметки – достаточно выбирать

задания случайным образом, принимая во внимание только то, что одному человеку допустимо разметить не больше одного примера. Но существуют ли оптимизированные методы онлайн разметки, если целью является построение качественного классификатора в задаче? В частности, возможно ли применение активного обучения в таком случае?

Процесс онлайн активного обучения с несколькими аннотаторами можно организовать в виде двух очередей. Первая очередь содержит примеры, ожидающие разметку, она пополняется пакетами, состоящими из нескольких примеров. Вторая очередь хранит ответы участников. Когда обработано определенное количество ответов, запускается очередная итерация активного обучения, и очередь обновляется.

Такой асинхронный процесс позволяет сократить время ожидания аннотатора, пока система генерирует следующий вопрос, но, к сожалению, он неэффективен, если качество аннотаций невысокое, и каждый пример требуется разметить несколькими людьми, прежде чем перейти к следующей итерации. Такая схема обсуждается в [10] и [20]. Когда пользователь открывает задание на крауд-платформе, система перенаправляет его на сервер заказчика, где объекты для разметки предоставляются в режиме онлайн.

В работе [12] для распределения заданий было предложено весь неразмеченный датасет D заранее разбивать на части $D = D_1 \cup ... \cup D_K$, $D_i = U_i \cup \mathcal{L}_i$, где K – число исполнителей, $\mathcal{L}$ и U_i – множества размеченных и неразмеченных примеров соответственно. Каждый пример попадает в одно и то же число частей: $|\{D_k | x_i \in D_k\}| = m, \forall x_i \in D$. Для каждого датасета D_i обучается отдельный классификатор $f_i(x)$. Функция ошибки $L(D)$ пытается одновременно оптимизировать все классификаторы $f_i(x)$:

$$L(D) = \sum_{i=1}^{K} \sum_{x_j \in \mathcal{L}_i} L_i(y_i^j, f_i(x_j)) + \\ + \sum_{1 \le i \ne j \le K} \sum_{x_k \in D_i \cap \mathcal{L}_j} L_{ij}(y_k^j, f_i(x_k)) + \\ + \lambda \sum_{i=1}^{K} \Omega(||f_i||_H)$$

Здесь L_i – функция ошибки классификатора i, а L_{ij} учитывает ошибку классификатора i на примерах, размеченных участником j и содержащихся в множестве $\mathcal{L}_i$; последнее слагаемое отвечает за регуляризацию. Таким образом, классификаторы не являются независимыми.

Разметка происходит активным обучением. Для исполнителя i выбирается пример, который находится ближе всего к границе решающего правила $f_i(x)$. После разметки параметры, связанные с аннотатором i оптимизируются согласно формуле; остальные параметры считаются фиксированными. В

качестве классификаторов в работе [12] используется метод опорных векторов. Итоговый классификатор получается усреднением алгоритмов $f_i(x)$.

input: множество неразмеченных примеров U
output: метки $\hat{\mathbf{y}} = [\hat{y}_1, ..., \hat{y}_N]$, качество участников $\mathbf{q} = [\hat{q}_1, ..., \hat{q}_K]$
L - матрица ответов, M - матрица распределений классов для всех примеров
1: Инициализировать $\mathbf{q}$
2: Выполнять:
// Получить номер исполнителя
3: j = get_requestor()
// Выбрать подходящий пример на основе q_j и априорных распределений M
4: i = U.sample(q_j, M)
5: L.Update(i, j, get_answer(j, x_i))
// Переоценить M на основе имеющихся ответов L и надежностей $\mathbf{q}$
6: M.Update(L, $\mathbf{q}$)
// Для каждого исполнителя j оценить q_j на основе M и L
7: for j:=1 to K do:
8: q_j.Update(L, M)

Алгоритм 3. Вывод в онлайн разметке
Algorithm 3. Inference in online annotation

Существуют несколько систем, которые поддерживают онлайн разметку произвольных данных, но не связаны с построением классификатора: DOCS [51], QASQA [53], iCrowd [5]. В качестве вывода обычно применяется итерационная схема с байесовским голосованием, по сути это онлайн-версия Алгоритма 1: при получении очередной аннотации примера сначала переоценивается ожидаемая метка для этого объекта, а затем с учетом обновленной метки заново вычисляются параметры исполнителей (т.е. в отличие от Алгоритма 1 параметры оцениваются после каждой аннотации, а не после получения всех аннотаций).

Опишем, как происходит назначение задания. В каждый момент времени для всех примеров хранится текущее апостериорное распределение меток в виде матрицы $M_{i,k}$, содержащей вероятности того, что пример i принадлежит классу k. Для исполнителя j оцениваются ожидаемые ответы: $Q_{ia}^{(j)}$ – вероятность предоставить класс a примеру i. В качестве априорного распределения классов выбирается матрица M. В простейшем случае, когда исполнитель описывается только вероятностью q_j предоставить правильный ответ, значения $Q_{ia}^{(j)}$ вычисляются так:

$$Q_{ia}^{(j)} = q_j M_{i,a} + \frac{1 - q_j}{J - 1}(1 - M_{i,a})$$

Для каждого возможного ответа a переоценивается матрица M:

$$M_{i,k}|^a \propto M_{i,k}(q_j)^{\mathbb{1}(a=k)}(\frac{1 - q_j}{J - 1})^{\mathbb{1}(a \neq k)}$$

В итоге выбираются те примеры, на которых максимально изменение метрики неопределенности. Например, в DOCS это разность текущей и ожидаемой энтропии: $H(M_i) - H(M_i^|)$, где

$$H(M_i^|) = \sum_{a=1}^{J} H(M_i^{|a}) Q_{ia}^{(j)}$$

Алгоритм 3 описывает подход в общем виде.

5. Обзор фреймворков

5.1 Крауд-платформы

В этом подразделе описываются несколько известных платформ для работы с крауд-вычислениями. В таких платформах есть два типа участников: заказчики, которые публикуют задания, и участники-исполнители или аннотаторы. Задания, как правило, представляются в виде набора из нескольких примеров для которых требуется аннотация, их называют HIT (Human Intelligence Task). Это могут быть различные задачи классификации, машинного перевода, сопоставления сущностей и многие другие.

Опишем подробнее использование одной из таких платформ – Яндекс.Толока. Перед публикацией заданий нужно создать проект и написать инструкцию к выполнению заданий. Затем необходимо оформить интерфейс в виде HTML текста. Задания можно добавлять набором по несколько штук, называемых пулом. Для каждого пула указывается максимальное время выполнения, а также перекрытие – количество пользователей, которые должны выполнить задание. Агрегация меток производится большинством голосов.

Для дополнительного контроля качества добавляются контрольные задания с заранее известными ответами, которые для исполнителя внешне ничем не отличаются от обычных. Таким образом, имеется возможность блокирования пользователей, которые либо часто ошибаются на контрольных вопросах, либо выполняют задания подозрительно быстро. Имеется также возможность добавления обучающих заданий, являющихся квалификационным тестом. После запуска в системе отображается прогресс по задаче.

Наиболее популярными англоязычными краудсорсинговыми платформами являются MTurk и CrowdFlower. Они предоставляют готовые шаблоны для оформления заданий, кроме того есть интерфейс для построения собственного дизайна задания средствами CSS и Javascript. Кроме обычного подхода, применяемого в Яндекс.Толока, существуют несколько других способов использования системы. Во-первых, имеются API, позволяющие выполнять операции добавления заданий, получать различные статистики и ответы на задания с помощью высокоуровневых языков программирования. Во-вторых, при выполнении задания платформа может перенаправлять пользователей на

сайт заказчика. Это может быть особенно полезно при использовании онлайн методов разметки.

5.2 Крауд-оптимизаторы

Существует несколько систем, предназначенных для упрощения и оптимизации работы с краудплатформами: хранения и обработки информации об аннотаторах и полученных аннотаций, формирования заданий и отправки их на крауд-платформу, улучшения качества вывода меток. Такие системы выступают посредниками между платформой и заказчиком. Эти системы уже упоминались в предыдущих частях. Важно, что они имеют косвенное отношение к разметке тренировочного множества для алгоритмов машинного обучения, их задачей является получение аннотаций для заданного набора примеров. И признаки объектов практически не используются.

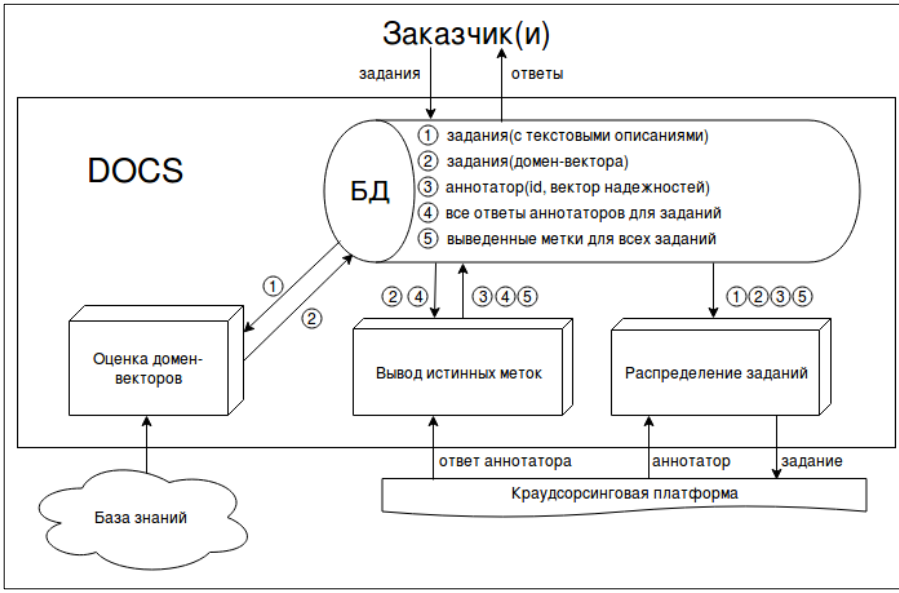


Рис. 3. Архитектура DOCS
Fig. 3. DOCS architecture

В самой простой такой системе Askit [1] не вводятся дополнительные параметры, и алгоритм работает только с полученными метками. Из множества примеров для повторной разметки выбираются те, у которых максимальна мера неопределенности. Для каждой возможной метки примера вычисляется энтропия всех меток, если к ним добавить новую. Для подсчета

неопределенности предлагается два способа: максимум этих значений и их среднее. Система CDAS [26] моделирует надежности исполнителей и определяет необходимое число вопросов для достижения определенного качества.

К другим подходам применима схема, описанная в предыдущей части для онлайн разметки. В QASCA [53] аннотатор моделируется вероятностью предоставить верный класс, а вероятность ответить любым неверным классом считается одинаковой. В качестве метрики выбирается не разность энтропии, а разность вероятностей наиболее уверенных классов в ожидаемом от аннотатора распределении и текущем.

В iCrowd [5] и DOCS [51] для оценки ожидаемых ответов $Q_{ia}^{(j)}$ используются более сложные методы, связанные с идеей зависимости компетентности исполнителя от темы задания. В iCrowd строится взвешенный граф похожести примеров.

В DOCS объект и аннотатор описываются векторами из нескольких чисел, соответствующих доменам. Объект – это вектор распределения доменов, аннотатор – вектор вероятностей предоставить правильный ответ на каждом домене. Возникает три задачи.

Во-первых, по имеющимся примерам нужно определить их вектора доменов. Постановка аналогична постановке задачи тематического моделирования [56]. Но авторы вместо использования стандартных методов предлагают свой подход. Для примера выделяют все сущности. Затем для каждой сущности находят распределения концептов. Эти два шага осуществляются с помощью готового викификатора. Каждый концепт имеет бинарный вектор домена (принадлежит/не принадлежит). Чтобы для данного набора концептов посчитать домен-вектор, нужно сложить эти бинарные вектора по всем сущностям и нормализовать. А чтобы посчитать вектор для всего примера, нужно найти математическое ожидание по распределению концептов.

Во-вторых, после получения каждой аннотации нужно переоценить надежности и метки. Это производится стандартным итеративным алгоритмом, но с учетом доменов. Сначала для каждого домена байесовским голосованием оценивается распределение меток для всех примеров. Окончательное распределение получается взвешенным суммированием этих распределений пропорционально весам доменов. Затем переоцениваются надежности аннотаторов на каждом домене. Для ускорения переоцениваются только параметры, непосредственно связанные с новой аннотацией – распределение меток соответствующего примера и компетентности аннотаторов, разметивших этот пример.

И третья задача – подбор задания участнику. Если он еще не выполнил ни одного задания, то предоставляется тестовое множество примеров. Иначе оцениваются метки, которые ожидаются от аннотатора (в зависимости от его

компетентности на доменах) и выбираются те примеры, для которых максимальна разность энтропии имеющихся меток и ожидаемых.

Авторы DOCS провели подробные сравнения системы с Askit, QASCA, iCrowd на крауд-платформе MTurk и с алгоритмами тематического моделирования для выявления доменов. Причём для каждой системы запускались независимые процессы разметки. Результаты показали, что DOCS выигрывает у конкурентов по всем показателям.

5.3 Крауд-СУБД

Популярными являются крауд системы, которые рассматривают данные в виде реляционных баз Qurk [28], Deco [32], CDB [22], CrowdOp [6]. В общем случае они устроены следующим образом. Пользователь загружает в систему данные в виде таблиц, возможно с пропущенными значениями, и вводит обычный SQL запрос. Система анализирует запрос, затем генерирует план в виде простых вопросов на краудплатформу и производит крауд оптимизации при исполнении этого плана. В одной из первой такой системе CrowdDB вводят три типа вопросов: заполнение пропущенных значений, сопоставление одинаковых объектов (join) и сравнение объектов по какомулибо показателю.

Важной задачей таких систем является построение оптимального плана запросов. Так, в Deco (2012), CrowdOP (2015) оптимизация строится на основе деревьев, а в недавней работе CDB (2017) на основе взвешенного графа объектов.

6. Заключение

Тема разметки данных с помощью краудсорсинга активно изучается последние несколько лет. За это время появилось много различных методов и реализаций. Так как основной особенностью крауд-систем является различный уровень компетентности и мотивированности исполнителей, то главная черта решений – это оценка уровня надежности исполнителей.

Оффлайн разметка является наиболее исследованной темой, это подтверждает наличие большого количества обзоров и сравнений [23, 48, 52, 55]. Для задач бинарной классификации лидерами остаются алгоритмы основанные на модели матрицы ошибок [3] и его модификации. Для многоклассовых задач имеется небольшое количество алгоритмов. Проблемой остается разрыв между разметкой данных и построением тренировочного множества для классификатора – практически все алгоритмы слабо учитывают признаковое описание примеров.

Более успешным является использование итеративного процесса разметки. Полагаем, что дальнейший фокус исследований будет смещен в эту сторону, в частности, к методам активного обучения. Здесь необходимо устранить эффект зависимости от одного исполнителя. Онлайн активное обучение

лишено этого недостатка, и также позволяет распараллелить работу исполнителей. Однако такие методы встречаются редко.

Также заметим, что акцент в современных работах смещается в сторону практических реализаций, набирают популярность крауд-оптимизаторы и их аналоги. К сожалению, основной целью таких работ является не построение классификатора для данной задачи, а точная разметка данных.

Список литературы

[1]. Rubi Boim, Ohad Greenshpan, Tova Milo, Slava Novgorodov, Neoklis Polyzotis, and Wang-Chiew Tan. Asking the right questions in crowd data sourcing. In Proc. of the 28th International Conference on Data Engineering (ICDE), 2012, pp 1261–1264.

[2]. Anthony Brew, Derek Greene, and Pa´draig Cunningham. Using crowdsourcing and active learning to track sentiment in online media. In Proc. of the 19th European Conference on Artificial Intelligence, pp. 145–150.

[3]. P. Dawid, A. M. Skene, A. P. Dawidt, and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. Applied Statistics, vol. 28, № 1, 1979, pp. 20-28.

[4]. Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudr´e-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In Proc. of the 21st international conference on World Wide Web, 2012, pp. 469–478.

[5]. Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-lee Tan, and Jianhua Feng. icrowd: An adaptive crowdsourcing framework. In Proc. of the ACM SIGMOD International Conference on Management of Data, 2015, pp. 1015–1030.

[6]. Ju Fan, Meihui Zhang, Stanley Kok, Meiyu Lu, and Beng Chin Ooi. Crowdop: Query optimization for declarative crowdsourcing systems. IEEE Transactions on Knowledge and Data Engineering, vol. 27, № 8, 2015, pp. 2078–2092.

[7]. Meng Fang, Xingquan Zhu, Bin Li, Wei Ding, and Xindong Wu. Self-taught active learning from crowds. In Proc. of the 12th International Conference on Data Mining (ICDM), 2012, pp. 858–863.

[8]. Paul Felt, Robbie Haertel, Eric K Ringger, and Kevin D Seppi. Momresp: A bayesian model for multi-annotator document labeling. In Proc. of the Ninth International Conference on Language Resources and Evaluation, 2014. pp. 3704–3711.

[9]. Arpita Ghosh, Satyen Kale, and Preston McAfee. Who moderates the moderators?: crowdsourcing abuse detection in user-generated content. In Proc. of the 12th ACM conference on Electronic commerce, 2011, pp. 167–176.

[10]. Daniel Haas, Jiannan Wang, Eugene Wu, and Michael J. Franklin. Clamshell: Speeding up crowds for low-latency data labeling. Proc. of the VLDB Endowment, vol. 9, № 4, 2015, pp. 372–383.

[11]. Shuji Hao, Steven C. H. Hoi, Chunyan Miao, and Peilin Zhao. Active crowdsourcing for annotation. In Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, vol. II, 2015, pp. 1–8.

[12]. Gang Hua, Chengjiang Long, Ming Yang, and Yan Gao. Collaborative active learning of a kernel machine ensemble for recognition. In Proc. of the IEEE International Conference on Computer Vision (ICCV), 2013, pp. 1209– 1216.

[13]. Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Lam Ngoc Tran, and Karl Aberer. An evaluation of aggregation techniques in crowdsourcing. In Proc. of the International Conference on Web Information Systems Engineering, 2013, pp. 1–15.

[14]. Hiroshi Kajino, Yuta Tsuboi, and Hisashi Kashima. A convex formulation for learning from crowds. Transactions of the Japanese Society for Artificial Intelligence, vol. 27, № 3, , 2012, pp. 133–142.

[15]. David R Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In Proc. of the Neural Information Processing Systems 2011 Conference (Advances in neural information processing systems 24), 2011, pp. 1953–1961.

[16]. Faiza Khan Khattak. Toward a Robust and Universal Crowd Labeling Framework. PhD Thesis, Columbia University, 2017, 168 p.

[17]. Faiza Khan Khattak and Ansaf Salleb-Aouissi. Quality control of crowd labeling through expert evaluation. In Proceedings of the NIPS 2nd Workshop on Computational Social Science and the Wisdom of Crowds, vol. 2, 2011, 5 p.

[18]. Adam Kilgarriff and Adam Kilgarriff. Gold standard datasets for evaluating word sense disambiguation programs. Computer Speech and Language, vol. 12, № 3, 1998, pp. 453-472.

[19]. Hyun-Chul Kim and Zoubin Ghahramani. Bayesian classifier combination. In Proc. of the Fifteenth International Conference on Artificial Intelligence and Statistics, 2012, pp. 619–627, 2012.

[20]. Florian Laws, Christian Scheible, and Hinrich Schütze. Active learning with amazon mechanical turk. In Proc. of the Conference on Empirical Methods in Natural Language Processing, 2011, pp. 1546–1556.

[21]. Kyumin Lee, James Caverlee, and Steve Webb. The social honeypot project: protecting online communities from spammers. In Proc. of the 19th International Conference on World Wide Web, 2010, pp 1139–1140.

[22]. Guoliang Li, Chengliang Chai, Ju Fan, Xueping Weng, Jian Li, Yudian Zheng, Yuanbing Li, Xiang Yu, Xiaohang Zhang, and Haitao Yuan. Cdb: optimizing queries with crowd-based selections and joins. In Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data, 2007, pp. 1463–1478.

[23]. Guoliang Li, Jiannan Wang, Yudian Zheng, and Michael J Franklin. Crowdsourced data management: A survey. IEEE Transactions on Knowledge and Data Engineering, vol. 28, № 9, 2016, pp. 2296–2319.

[24]. Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. A survey on truth discovery. ACM SIGKDD Explorations Newsletter, vol. 17, № 2, 2016, pp. 1–16.

[25]. Qiang Liu, Jian Peng, and Alexander T Ihler. Variational inference for crowdsourcing. In Proc. of the Neural Information Processing Systems 2011 Conference (Advances in neural information processing systems 25), 2012, pp. 692–700.

[26]. Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. Cdas: a rowdsourcing data analytics system. Proc. of the VLDB Endowment, vol. 5, № 10, 2012, pp. 1040–1051.

[27]. Yang Liu and Mingyan Liu. An online learning approach to improving the quality of crowd-sourcing. ACM SIGMETRICS Performance Evaluation Review, vol. 43, 2015, pp. 217–230.

[28]. Adam Marcus, Eugene Wu, David R Karger, Samuel Madden, and Robert C Miller. Crowdsourced databases: Query processing with people. Proc. of the 5th Conference on Innovative Data Systems Research (CIDR). 2011, pp. 211-214.

[29]. Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. Scaling up crowdsourcing to very large datasets: a case for active learning. Proc. of the VLDB Endowment, vol. 8, № 2, 2014, pp. 125–136.

[30]. An Thanh Nguyen, Byron C Wallace, and Matthew Lease. Combining crowd and expert labels using decision theoretic active learning. In Proc. of the Third AAAI Conference on Human Computation and Crowdsourcing, 2015, pp. 120-129.

[31]. Stefanie Nowak and Stefan Ru¨ger. How reliable are annotations via crowdsourcing: A study about interannotator agreement for multi-label image annotation. In Proc. of the International Conference on Multimedia Information Retrieval, 2010, pp. 557– 566.

[32]. Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: declarative crowdsourcing. In Proceedings of the 21st ACM International Conference on Information and Knowledge Management, 2012, pp. 1203–1212.

[33]. Carl Edward Rasmussen. Gaussian processes in machine learning. In Advanced Lectures on Machine Learning, Lecture Notes in Computer Science, vol 3176, 2004, pp. 63–71.

[34]. Vikas C Raykar, Shipeng Yu, Linda H Zhao, Anna Jerebko, Charles Florin, Gerardo Hermosillo Valadez, Luca Bogoni, and Linda Moy. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In Proc. of the 26th Annual international conference on machine learning, 2009, pp. 889–896.

[35]. Vikas C Raykar, Shipeng Yu, Linda H Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. Learning from crowds. Journal of Machine Learning Research, vol. 11, 2010, pp. 1297–1322.

[36]. Filipe Rodrigues, Francisco Pereira, and Bernardete Ribeiro. Gaussian process classification and active learning with multiple annotators. In Proc. of the International Conference on Machine Learning, 2014, pp. 433–441.

[37]. Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin– Madison, 2009, 65 p.

[38]. Victor S. Sheng, Foster J. Provost, and Panagiotis G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 614– 622.

[39]. Aashish Sheshadri and Matthew Lease. Square: A benchmark for research on computing crowd consensus. In Proc. of the First AAAI Conference on Human Computation and Crowdsourcing, 2013, pp. 156-164.

[40]. Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fast– but is it good?: Evaluating non-expert annotations for natural language tasks. In Proc. of the Conference on Empirical Methods in Natural Language Processing, 2008, pp. 254–263.

[41]. Long Tran-Thanh, Sebastian Stein, Alex Rogers, and Nicholas R Jennings. Efficient crowdsourcing of unknown experts using bounded multi-armed bandits. Artificial Intelligence, vol. 214, issue 1, 2014, pp. 89–111.

[42]. Fabian L Wauthier and Michael I Jordan. Bayesian bias mitigation for crowdsourcing. In In Proc. of the Neural Information Processing Systems 2011 Conference (Advances in neural information processing systems 24), 2011, pp. 1800–1808.

[43]. Peter Welinder and Pietro Perona. Online crowdsourcing: rating annotators and obtaining cost-effective labels. In Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, 2010, pp. 25–32.
[44]. Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In Proc. of the Neural Information Processing Systems 2009 Conference (Advances in neural information processing systems 22), 2009, pp. 2035–2043.
[45]. Yan Yan, Romer Rosales, Glenn Fung, and Jennifer G Dy. Active learning from crowds. In Proc. of the 28th International Conference on International Conference on Machine Learning, 2011, pp. 1161–1168.
[46]. Chicheng Zhang and Kamalika Chaudhuri. Active learning from weak and strong labelers. In Proc. of the Neural Information Processing Systems 2015 Conference (Advances in neural information processing systems 28), 2015, pp. 703–711.
[47]. Jing Zhang, Victor S Sheng, Jian Wu, and Xindong Wu. Multi-class ground truth inference in crowdsourcing with clustering. IEEE Transactions on Knowledge and Data Engineering, vol. 28, № 4, 2016, pp. 1080–1085.
[48]. Jing Zhang, Xindong Wu, and Victor S Sheng. Learning from crowdsourced labeled data: a survey. Artificial Intelligence Review, vol. 46, № 4, 2016, pp. 543–576.
[49]. Yuchen Zhang, Xi Chen, Denny Zhou, and Michael I Jordan. Spectral methods meet em: A provably optimal algorithm for crowdsourcing. In Proc. of the Neural Information Processing Systems 2014 Conference (Advances in neural information processing systems 27), 2014, pp. 1260–1268.
[50]. Yudian Zheng, Reynold Cheng, Silviu Maniu, and Luyi Mo. On optimality of jury selection in crowdsourcing. In Proceedings of the 18th International Conference on Extending Database Technology, 2015, pp. 193-204.
[51]. Yudian Zheng, Guoliang Li, and Reynold Cheng. Docs: a domain-aware crowdsourcing system using knowledge bases. Proc. of the VLDB Endowment, vol. 10, № 4, 2016, pp. 361– 372.
[52]. Yudian Zheng, Guoliang Li, Yuanbing Li, Caihua Shan, and Reynold Cheng. Truth inference in crowdsourcing: is the problem solved? Proc. of the VLDB Endowment, vol. 10, № 5, 2017, pp. 541–552.
[53]. Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. Qasca: A quality-aware task assignment system for crowdsourcing applications. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2015, pp. 1031–1046.
[54]. Jinhong Zhong, Ke Tang, and Zhi-Hua Zhou. Active learning from crowds with unsure option. In Proc. of the 24th International Conference on Artificial Intelligence, 2015, pp. 1061–1068.
[55]. Пономарев А. В. Методы обеспечения качества в системах крауд-вычислений: аналитический обзор. Труды СПИИРАН, вып. 54, 2017, pp. 152–184. DOI: 10.15622/sp.54.7
[56]. Коршунов А., Гомзин А. Тематическое моделирование текстов на естественном языке. Труды ИСП РАН, 23, 2012, стр. 215-244. DOI: 10.15514/ISPRAS-2012-23-13

Active learning and crowdsourcing: a survey of annotation optimization methods

[1,2]*R. A. Gilyazev <gilyazev@ispras.ru>*
[1,3,4]*D. U. Turdakov <turdakov@ispras.ru>*
[1]*Ivannikov Institute for System Programming of the RAS,*
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia
[2]*Moscow Institute of Physics and Technology (State University),*
9, Institutskiy per., Dolgoprudny,
[3]*Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia Moscow Region, 141700, Russia
[4]*National Research University Higher School of Economics (HSE),*
20, Myasnitskaya Ulitsa, Moscow, 101000, Russia

Abstract. High quality labeled corpora play a key role to elaborate machine learning systems. Generally, creating of such corpora requires human efforts. So, annotation process is expensive and time-consuming. Two approaches that optimize the annotation are active learning and crowdsourcing. Methods of active learning are aimed at finding the most informative examples for the classifier. At each iteration from the unplaced set, one algorithm is chosen by an algorithm, it is provided to the oracle (expert) for the markup and the classifier is newly trained on the updated set of training examples. Crowdsourcing is widely used in solving problems that can not be automated and require human effort. To get the most out of using crowdplatforms one needs to to solve three problems. The first of these is quality, that is, algorithms are needed that will best determine the real labels from the available ones. Of course, it is necessary to remember the cost of markup - to solve the problem by increasing the number of annotators for one example is not always reasonable - this is the second problem. And, thirdly, sometimes the immediate factor is the rapid receipt of the marked corpus, then it is necessary to minimize the time delays when the participants perform the task. This paper aims to survey existing methods based on this approaches and techniques to combine them. Also, the paper describes the systems that help to reduce the cost of annotation.

Keywords: active learning; crowdsourcing; learning from crowds; annotation; ground truth inference

DOI: 10.15514/ISPRAS-2018-30(1)-11

For citation: Gilyazev R.A., Turdakov D.Y. Active learning and crowdsourcing: a survey of data markup optimization methods. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 2, 2018, pp. 215-250 (in Russian). DOI: 10.15514/ISPRAS-2018-30(1)-11

References

[1]. Rubi Boim, Ohad Greenshpan, Tova Milo, Slava Novgorodov, Neoklis Polyzotis, and Wang-Chiew Tan. Asking the right questions in crowd data sourcing. In Proc. of the 28th International Conference on Data Engineering (ICDE), 2012, pp 1261–1264.

[2]. Anthony Brew, Derek Greene, and Pa´draig Cunningham. Using crowdsourcing and active learning to track sentiment in online media. In Proc. of the 19th European Conference on Artificial Intelligence, pp. 145–150.
[3]. P. Dawid, A. M. Skene, A. P. Dawidt, and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. Applied Statistics, vol. 28, № 1, 1979, pp. 20-28.
[4]. Gianluca Demartini, Djellel Eddine Difallah, and Philippe Cudr´e-Mauroux. Zencrowd: leveraging probabilistic reasoning and crowdsourcing techniques for large-scale entity linking. In Proc. of the 21st international conference on World Wide Web, 2012, pp. 469–478.
[5]. Ju Fan, Guoliang Li, Beng Chin Ooi, Kian-lee Tan, and Jianhua Feng. icrowd: An adaptive crowdsourcing framework. In Proc. of the ACM SIGMOD International Conference on Management of Data, 2015, pp. 1015–1030.
[6]. Ju Fan, Meihui Zhang, Stanley Kok, Meiyu Lu, and Beng Chin Ooi. Crowdop: Query optimization for declarative crowdsourcing systems. IEEE Transactions on Knowledge and Data Engineering, vol. 27, № 8, 2015, pp. 2078–2092.
[7]. Meng Fang, Xingquan Zhu, Bin Li, Wei Ding, and Xindong Wu. Self-taught active learning from crowds. In Proc. of the 12th International Conference on Data Mining (ICDM), 2012, pp. 858–863.
[8]. Paul Felt, Robbie Haertel, Eric K Ringger, and Kevin D Seppi. Momresp: A bayesian model for multi-annotator document labeling. In Proc. of the Ninth International Conference on Language Resources and Evaluation, 2014. pp. 3704–3711.
[9]. Arpita Ghosh, Satyen Kale, and Preston McAfee. Who moderates the moderators?: crowdsourcing abuse detection in user-generated content. In Proc. of the 12th ACM conference on Electronic commerce, 2011, pp. 167–176.
[10]. Daniel Haas, Jiannan Wang, Eugene Wu, and Michael J. Franklin. Clamshell: Speeding up crowds for low-latency data labeling. Proc. of the VLDB Endowment, vol. 9, № 4, 2015, pp. 372–383.
[11]. Shuji Hao, Steven C. H. Hoi, Chunyan Miao, and Peilin Zhao. Active crowdsourcing for annotation. In Proc. of the IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, vol. II, 2015, pp. 1–8.
[12]. Gang Hua, Chengjiang Long, Ming Yang, and Yan Gao. Collaborative active learning of a kernel machine ensemble for recognition. In Proc. of the IEEE International Conference on Computer Vision (ICCV), 2013, pp. 1209– 1216.
[13]. Nguyen Quoc Viet Hung, Nguyen Thanh Tam, Lam Ngoc Tran, and Karl Aberer. An evaluation of aggregation techniques in crowdsourcing. In Proc. of the International Conference on Web Information Systems Engineering, 2013, pp. 1–15.
[14]. Hiroshi Kajino, Yuta Tsuboi, and Hisashi Kashima. A convex formulation for learning from crowds. Transactions of the Japanese Society for Artificial Intelligence, vol. 27, № 3, , 2012, pp. 133–142.
[15]. David R Karger, Sewoong Oh, and Devavrat Shah. Iterative learning for reliable crowdsourcing systems. In Proc. of the Neural Information Processing Systems 2011 Conference (Advances in neural information processing systems 24), 2011, pp. 1953–1961.
[16]. Faiza Khan Khattak. Toward a Robust and Universal Crowd Labeling Framework. PhD Thesis, Columbia University, 2017, 168 p.

[17]. Faiza Khan Khattak and Ansaf Salleb-Aouissi. Quality control of crowd labeling through expert evaluation. In Proceedings of the NIPS 2nd Workshop on Computational Social Science and the Wisdom of Crowds, vol. 2, 2011, 5 p.

[18]. Adam Kilgarriff and Adam Kilgarriff. Gold standard datasets for evaluating word sense disambiguation programs. Computer Speech and Language, vol. 12, № 3, 1998, pp. 453-472.

[19]. Hyun-Chul Kim and Zoubin Ghahramani. Bayesian classifier combination. In Proc. of the Fifteenth International Conference on Artificial Intelligence and Statistics, 2012, pp. 619–627, 2012.

[20]. Florian Laws, Christian Scheible, and Hinrich Schütze. Active learning with amazon mechanical turk. In Proc. of the Conference on Empirical Methods in Natural Language Processing, 2011, pp. 1546–1556.

[21]. Kyumin Lee, James Caverlee, and Steve Webb. The social honeypot project: protecting online communities from spammers. In Proc. of the 19th International Conference on World Wide Web, 2010, pp 1139–1140.

[22]. Guoliang Li, Chengliang Chai, Ju Fan, Xueping Weng, Jian Li, Yudian Zheng, Yuanbing Li, Xiang Yu, Xiaohang Zhang, and Haitao Yuan. Cdb: optimizing queries with crowd-based selections and joins. In Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data, 2007, pp. 1463–1478.

[23]. Guoliang Li, Jiannan Wang, Yudian Zheng, and Michael J Franklin. Crowdsourced data management: A survey. IEEE Transactions on Knowledge and Data Engineering, vol. 28, № 9, 2016, pp. 2296–2319.

[24]. Yaliang Li, Jing Gao, Chuishi Meng, Qi Li, Lu Su, Bo Zhao, Wei Fan, and Jiawei Han. A survey on truth discovery. ACM SIGKDD Explorations Newsletter, vol. 17, № 2, 2016, pp. 1–16.

[25]. Qiang Liu, Jian Peng, and Alexander T Ihler. Variational inference for crowdsourcing. In Proc. of the Neural Information Processing Systems 2011 Conference (Advances in neural information processing systems 25), 2012, pp. 692–700.

[26]. Xuan Liu, Meiyu Lu, Beng Chin Ooi, Yanyan Shen, Sai Wu, and Meihui Zhang. Cdas: a rowdsourcing data analytics system. Proc. of the VLDB Endowment, vol. 5, № 10, 2012, pp. 1040–1051.

[27]. Yang Liu and Mingyan Liu. An online learning approach to improving the quality of crowd-sourcing. ACM SIGMETRICS Performance Evaluation Review, vol. 43, 2015, pp. 217–230.

[28]. Adam Marcus, Eugene Wu, David R Karger, Samuel Madden, and Robert C Miller. Crowdsourced databases: Query processing with people. Proc. of the 5th Conference on Innovative Data Systems Research (CIDR). 2011, pp. 211-214.

[29]. Barzan Mozafari, Purna Sarkar, Michael Franklin, Michael Jordan, and Samuel Madden. Scaling up crowdsourcing to very large datasets: a case for active learning. Proc. of the VLDB Endowment, vol. 8, № 2, 2014, pp. 125–136.

[30]. An Thanh Nguyen, Byron C Wallace, and Matthew Lease. Combining crowd and expert labels using decision theoretic active learning. In Proc. of the Third AAAI Conference on Human Computation and Crowdsourcing, 2015, pp. 120-129.

[31]. Stefanie Nowak and Stefan Ru¨ger. How reliable are annotations via crowdsourcing: A study about interannotator agreement for multi-label image annotation. In Proc. of the International Conference on Multimedia Information Retrieval, 2010, pp. 557– 566.

[32]. Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. Deco: declarative crowdsourcing. In Proceedings of the

21st ACM International Conference on Information and Knowledge Management, 2012, pp. 1203–1212.
[33]. Carl Edward Rasmussen. Gaussian processes in machine learning. In Advanced Lectures on Machine Learning, Lecture Notes in Computer Science, vol 3176, 2004, pp. 63–71.
[34]. Vikas C Raykar, Shipeng Yu, Linda H Zhao, Anna Jerebko, Charles Florin, Gerardo Hermosillo Valadez, Luca Bogoni, and Linda Moy. Supervised learning from multiple experts: whom to trust when everyone lies a bit. In Proc. of the 26th Annual international conference on machine learning, 2009, pp. 889–896.
[35]. Vikas C Raykar, Shipeng Yu, Linda H Zhao, Gerardo Hermosillo Valadez, Charles Florin, Luca Bogoni, and Linda Moy. Learning from crowds. Journal of Machine Learning Research, vol. 11, 2010, pp. 1297–1322.
[36]. Filipe Rodrigues, Francisco Pereira, and Bernardete Ribeiro. Gaussian process classification and active learning with multiple annotators. In Proc. of the International Conference on Machine Learning, 2014, pp. 433–441.
[37]. Burr Settles. Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin– Madison, 2009, 65 p.
[38]. Victor S. Sheng, Foster J. Provost, and Panagiotis G. Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008, pp. 614– 622.
[39]. Aashish Sheshadri and Matthew Lease. Square: A benchmark for research on computing crowd consensus. In Proc. of the First AAAI Conference on Human Computation and Crowdsourcing, 2013, pp. 156-164.
[40]. Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. Cheap and fast– but is it good?: Evaluating non-expert annotations for natural language tasks. In Proc. of the Conference on Empirical Methods in Natural Language Processing, 2008, pp. 254–263.
[41]. Long Tran-Thanh, Sebastian Stein, Alex Rogers, and Nicholas R Jennings. Efficient crowdsourcing of unknown experts using bounded multi-armed bandits. Artificial Intelligence, vol. 214, issue 1, 2014, pp. 89–111.
[42]. Fabian L Wauthier and Michael I Jordan. Bayesian bias mitigation for crowdsourcing. In In Proc. of the Neural Information Processing Systems 2011 Conference (Advances in neural information processing systems 24), 2011, pp. 1800–1808.
[43]. Peter Welinder and Pietro Perona. Online crowdsourcing: rating annotators and obtaining cost-effective labels. In Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, 2010, pp. 25–32.
[44]. Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In Proc. of the Neural Information Processing Systems 2009 Conference (Advances in neural information processing systems 22), 2009, pp. 2035–2043.
[45]. Yan Yan, Romer Rosales, Glenn Fung, and Jennifer G Dy. Active learning from crowds. In Proc. of the 28th International Conference on International Conference on Machine Learning, 2011, pp. 1161–1168.
[46]. Chicheng Zhang and Kamalika Chaudhuri. Active learning from weak and strong labelers. In Proc. of the Neural Information Processing Systems 2015 Conference (Advances in neural information processing systems 28), 2015, pp. 703–711.

[47]. Jing Zhang, Victor S Sheng, Jian Wu, and Xindong Wu. Multi-class ground truth inference in crowdsourcing with clustering. IEEE Transactions on Knowledge and Data Engineering, vol. 28, № 4, 2016, pp. 1080–1085.

[48]. Jing Zhang, Xindong Wu, and Victor S Sheng. Learning from crowdsourced labeled data: a survey. Artificial Intelligence Review, vol. 46, № 4, 2016, pp. 543–576.

[49]. Yuchen Zhang, Xi Chen, Denny Zhou, and Michael I Jordan. Spectral methods meet em: A provably optimal algorithm for crowdsourcing. In Proc. of the Neural Information Processing Systems 2014 Conference (Advances in neural information processing systems 27), 2014, pp. 1260–1268.

[50]. Yudian Zheng, Reynold Cheng, Silviu Maniu, and Luyi Mo. On optimality of jury selection in crowdsourcing. In Proceedings of the 18th International Conference on Extending Database Technology, 2015, pp. 193-204.

[51]. Yudian Zheng, Guoliang Li, and Reynold Cheng. Docs: a domain-aware crowdsourcing system using knowledge bases. Proc. of the VLDB Endowment, vol. 10, № 4, 2016, pp. 361– 372.

[52]. Yudian Zheng, Guoliang Li, Yuanbing Li, Caihua Shan, and Reynold Cheng. Truth inference in crowdsourcing: is the problem solved? Proc. of the VLDB Endowment, vol. 10, № 5, 2017, pp. 541–552.

[53]. Yudian Zheng, Jiannan Wang, Guoliang Li, Reynold Cheng, and Jianhua Feng. Qasca: A quality-aware task assignment system for crowdsourcing applications. In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2015, pp. 1031–1046.

[54]. Jinhong Zhong, Ke Tang, and Zhi-Hua Zhou. Active learning from crowds with unsure option. In Proc. of the 24th International Conference on Artificial Intelligence, 2015, pp. 1061–1068.

[55]. A.V. Ponomarev. Quality Control Methods in Crowd Computing: Literature Review. SPIIRAS Proceeding, issue 54, 2017, pp. 152–184 (in Russian). DOI: 10.15622/sp.54.7.

[56]. A. Korshunov, A. Gomzin. Topic modeling in natural language texts. Trudy ISP RAN/Proc. ISP RAS, vol. 23, 2012, pp. 215-244 (in Russian). DOI: 10.15514/ISPRAS-2012-23-13.

Анализ баллистокардиограммы на граничных вычислительных узлах

[1] *А.С. Нужный <nuzhny@inbox.ru>*
[2] *А.А. Прозоров <prozoroff@mail.ru>*
[2] *В.И. Бугаев <vik362@list.ru>*
[2] *Н.Д. Шувалов <shuvalovnickolay@gmail.com>*
[2] *В.В. Подымов <valdus@yandex.ru>*
[1] *Институт проблем безопасного развития атомной энергетики РАН,*
Россия, 115191, г. Москва, Большая Тульская ул., д. 52
[2] *Московский физико-технический институт (государственный университет)*
Россия, 141701, Московская область,
г. Долгопрудный, Институтский переулок, д.9

Аннотация. В работе рассматривается бесконтактный метод анализа сердечной активности человека, основанный на регистрации и обработке баллистокардиографичекого сигнала. В измерительной установке для фиксации микроскопических движений тела используется пьезоэлектрический датчик высокой чувствительности. Появляющийся вследствие высокой чувствительности шум, существенно превышающий полезный сигнал, в дальнейшем фильтруется математическими методами. Для выделения кардио компоненты используется полосный фильтр Баттерворта. Этот подход к фильтрации полезного сигнала является более экономичным с точки зрения необходимых вычислительных ресурсов, чем сравнимые по точности методы, основанные на машинном обучении, и может быть реализован на граничном (промежуточном) вычислительном узле, к которому подключены несколько датчиков. Качество полученной после фильтрации кардио компоненты позволяет с высокой точностью выделить на ней циклы сердечной активности (сердцебиения). Предлагаемый в работе алгоритм выделения сердцебиений также обладает достаточно низкой вычислительной стоимостью, чтобы быть использованным на граничном вычислительном узле. После фильтрации данные передаются выше – в центр обработки данных (облако).[1]

Ключевые слова: баллистокардиография; сердечная активность; фильтр Баттерворта; интернет вещей**.**

DOI: 10.15514/ISPRAS-2018-30(2)-12

[1] Исследование выполнено за счет гранта Российского научного фонда (проект №17-71-10215)

Для цитирования: Нужный А.С., Прозоров А.А., Бугаев В.И., Шувалов Н.Д., Подымов В.В. Анализ баллистокардиограммы на граничных вычислительных узлах. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 251-262. DOI: 10.15514/ISPRAS-2018-30(2)-12

1. Введение

По данным правительства Москвы [1], в Москве насчитывается не менее 100 000 человек, которым требуется постоянный мониторинг давления и пульса с целью предупреждения возможного ухудшения здоровья или более тяжелых последствий. Кроме того, у большого числа людей встречаются отдельные периоды жизни, когда необходим постоянный мониторинг сердечной деятельности. Это накладывает существенную дополнительную нагрузку на медицинских и социальных работников, а также родственников пациентов.

Использование прикроватных медицинских мониторов, во многих случаях, не оправдано экономически. Диагностика на основе анализа ЭКГ-сигнала требует непосредственного физического контакта с пациентом, что причиняет ему неудобство. Разрабатываемые в настоящее время бесконтактные домашние мониторы могут позволить, с одной стороны, снизить смертность, с другой – дать возможность проводить регулярную оценку состояния здоровья пациента без понижения его бытового комфорта, а также уменьшить нагрузку на медицинский персонал.

В представленной работе рассматривается метод анализа сердечной активности путем измерения баллистокардиографического (БКГ) сигнала. Данный метод записывает механическую активность сердца, передаваемого через тело пациента и позволяет вести мониторинг бесконтактно или посредством косвенного контакта (через матрас, одеяло и т.п.).

Методики анализа сердечной активности, основанные на измерении БКГ, в научных кругах разрабатываются уже давно. Сама идея измерения механических колебаний тела пациента была предложена еще в конце 19-го века [2]. Основой измерительных установок, как правило, служил акселерометр [3] – прибор, измеряющий ускорение тела.

Однако методики, основанные на измерении БКГ сигнала, пока не получили широкого применения в медицинской практике. По мнению авторов это связано с двумя факторами: во-первых, исследователи прошлого века не располагали достаточно чувствительными измерительными приборами, позволяющими получить сигнал удовлетворительного качества, во-вторых, не был предложен адекватный математический аппарат обработки и анализа БКГ сигналов.

Сигнал нового качества удалось получить благодаря использованию пьезоэлектрической сенсорной поверхности Emfit [4]. Такие сенсоры реагируют на изменение давления тела пациента на опору и обладают большей чувствительностью, чем стандартные акселерометры. Возникающий вследствие высокой чувствительности прибора шум, вызванный внешними

воздействиями, в дальнейшем фильтруется математическими методами. Предлагаемые математические алгоритмы фильтрации и анализа сигнала имеют низкую вычислительную стоимость, что позволяет их использовать на граничных вычислительных узлах системы интернета вещей, обладающих незначительной производительностью.

2. Описание ИТ-инфрастурктуры

В качестве центрального компонента инфраструктуры решено было использовать облако. Важным аргументом в пользу облака является то, что мониторинговые данные, собираемые с разных отделений больниц в облаке, позволяют создать единую базу мониторинговых данных, которая позволит решить различные аналитические задачи из разных областей медицинской информатики.

Известны и широко применяются три варианта развертывания эластичных вычислительных архитектур: частное облако, гибридное облако и публичное облако. Использование публичного облака для сбора, обработки и хранения медицинских данных не подходит, так как это довольно чувствительная информация о пациентах и безопасность публичного облака никак не контролируется со стороны компании-арендатора. Частное облако лишено этого недостатка, как и вариант гибридного облака, если сегмент гибридного облака, в котором обрабатываются и хранятся мониторинговые данные, является частным. Частное облако подходит по причине того, что компания либо владеет аппаратными мощностями, либо арендует их и может контролировать, что позволяет ей обеспечить необходимый уровень безопасности хранения и обработки мониторинговых медицинских данных.

Вопросы «возможно ли» и «целесообразно ли» отправлять мониторинговые данные в облако перестают быть актуальными, как только появляется надежный зарезервированный канал связи с малыми временами задержки, так что разница в хранении мониторинговых данных на локальном сервере или в облаке стремится к нулю, в то время как эластичность предоставления ресурсов и готовность инфраструктуры в облаке являются значительно более высокими.

Эволюция эластичных архитектур в архитектуры интернета вещей определила три вычислительных слоя [5]: нижний слой – это устройства, снимающие информацию с источника (сенсор, актуатор, привод и т.д.), второй слой – это граничные вычислительные узлы, выполняющие функции первичной обработки данных и управления устройствами съёма информации, и третий слой – это центр обработки данных (облако), выполняющий функции сбора, хранения и анализа собранных данных. Архитектура целевой системы интернета вещей изображена на рис. 1.

В силу многочисленности устройств съёма данных, а также незначительной вычислительной мощности граничных вычислительных узлов [6], важным требованием, предъявляемым к алгоритмам непрерывной обработки сигналов,

является сниженные потребности в вычислительных ресурсах, позволяющие уменьшить потребление электроэнергии и снизить единицу мощности граничных вычислительных узлов в расчете на один подключенный источник сигнала.

Ниже будет предложен не дорогой с вычислительной точки зрения алгоритм фильтрации и разметки БКГ, предназначенный для использования на граничных вычислительных узлах. Такая конфигурация вычислительных мощностей распределенной информационно-коммуникационной системы позволит значительно снизить объем данных, передаваемых в облако и, соответственно, распределить необходимые вычисления по периметру сети сбора и обработки данных.

В качестве граничного узла используется кластер из трех Raspberry Pi 3B. На кластере производится фильтрация сырого сигнала, разметка J-пиков, расчет дискретных показателей, а также инкапсуляция потока медицинских данных в защищенное SSL-соединение. Преимуществом такого решения является его бесшумная работа и низкая стоимость.

Связь между граничными вычислительными узлами и центром обработки данных осуществляется по протоколу TCP, что позволяет использовать для подключения сенсоров уже имеющуюся инфраструктуру.

Обмен информацией между граничными узлами и облаком производится в формате JSON, что позволяет легко дополнять протокол при необходимости.

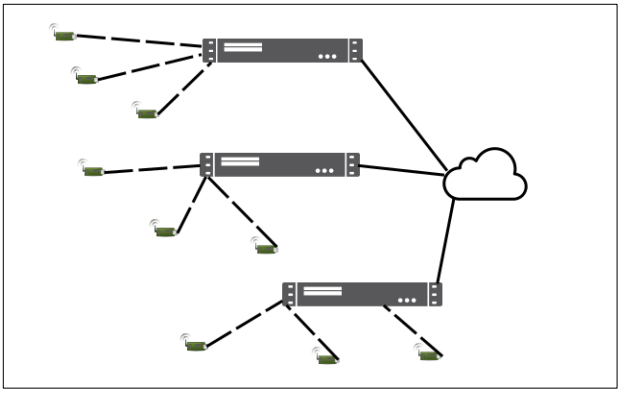


Рис. 1. Архитектура целевой системы интернета вещей
Fig. 1. Architecture of the target system of the Internet of things

3. Описание измерительного устройства

В качестве измерительного устройства использовался аппаратно-программный комплекс для бесконтактной регистрации основных биометрических показателей пациента в непрерывном режиме в состоянии лежа [7].

Программное наполнение измерительного устройства реализовано с использованием свободного программного обеспечения и готово для размещения на граничных (промежуточных) вычислительных узлах в трехкомпонентной вычислительной архитектуре интернета вещей [6].

Внешний вид устройства показан на рис. 2. Цифрами на нем обозначены:

- сенсор в чехле из плотной ткани (1);
- корпус с контроллером (2);
- коаксиальный кабель, соединяющий сенсор и корпус (3);
- Ethernet кабель, соединяющий устройство с роутером (4).

Для снятия экспериментальных данных с исследуемых пациентов была разработана система оцифровки и передачи информации на сервер через Ethernet. Экспериментальная установка включает в себя в качестве подсистемы сбора исследуемого БКГ сигнала:

- пьезоэлектрические датчики, с разработанными авторским коллективом электрометрическими усилителями к ним;
- систему передачи оцифрованного усиленного сигнала с помощью микроконтроллера, реализующего непрерывную оцифровку сигнала и управление системой;
- микрокомпьютер с усеченной системой Linux.

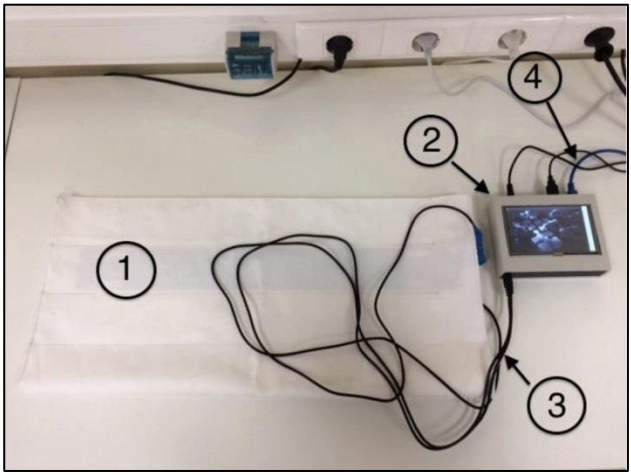


Рис. 2. Снимок измерительного устройства
Fig. 2. Photo of measuring device

4. Выделение кардио компоненты

Первоначально полученный БКГ сигнал не поддается визуальному анализу (Рис. 3, верхний график), в отличие от широко распространенного в медицинской практике электрокардиографического (ЭКГ) сигнала. Последний

передается врачу практически без предварительной обработки. Однако ЭКГ характеризует электрическую компоненту сердца, в то время как БКГ – механическую. С одной стороны это обстоятельство может указывать на то, что БКГ содержит больше информации о механике работы сердца, с другой, записанный БКГ сигнал помимо динамики, вызванной работой сердечной мышцы, содержит другие движение организма пациента, а также внешние механические шумы. Таким образом, требуется математический аппарат, позволяющий извлекать полезные компоненты БКГ. Среди других составляющих сигнала практический интерес представляет его дыхательная компонента [8].

Для фильтрации кардио компоненты были предложены различные подходы, основанные на дискретном вейвлет-преобразовании [9-11], преобразовании Гильберта [12], низкочастотной и полосной фильтрации [13-15] и т.д. В последние годы в литературе преобладает метод выделения кардио компоненты из БКГ сигнала, основанный на применении полосного фильтра Баттерворта [8, 13-15].

Фильтр Баттерворта подавляет в сигнале частоты, задаваемого его параметрами диапазона. Амплитудно-частотная характеристика этого фильтра максимально гладкая на частотах полосы пропускания и убывает практически до нуля на частотах полосы подавления [16]. Полосный фильтр Баттерворта содержит три параметра – порядок фильтра, частоту среза, отсекающую высокие частоты, и частоту среза, отсекающую низкие частоты. При правильно выбранных значениях параметров фильтрации, фильтр Баттерворта выделяет кардио компоненту сигнала, отсекая остальные его составляющие. На нижнем графике рис. 3 приведена кардио компонента, полученная полосным фильтром Баттерворта 8-го порядка с частотами пропускания между 25 и 35 Гц.

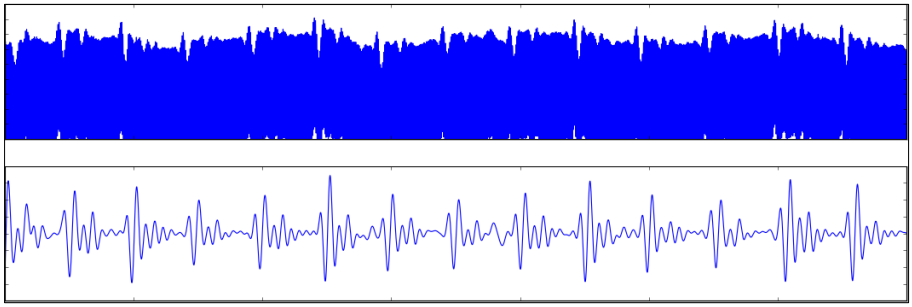


Рис. 3. На верхнем графике приведен исходный усиленный сигнал датчика, после оцифровки. На нижнем – сигнал после фильтрации полосным фильтром Баттерворта 8-го порядка с частотами пропускания от 25 до 35 Гц

Fig. 3. The upper graph shows the original amplified sensor signal, after digitization. At the bottom - the signal after filtering the band-pass filter Butterworth 8th order with transmission frequencies from 25 to 35 Hz

5. Модель распознавания кардиоциклов

Один кардиоцикл пациента, находящегося в положении лежа на спине, представляет на графике структуру, известную как HIJKLMN комплекс. Он состоит из четырех пиков и трех провалов (Рис. 4). Второй пик кардиоцикла (J-пик) у здорового человека обычно является самым высоким. Он соответствует срабатыванию левого желудочка. Интервал между сердцебиениями можно определить, как расстояние между двумя соседними J-пиками. В результате задача вычисления пульса будет сведена к задаче распознавания J-пиков на кардио компоненте БКГ сигнала.

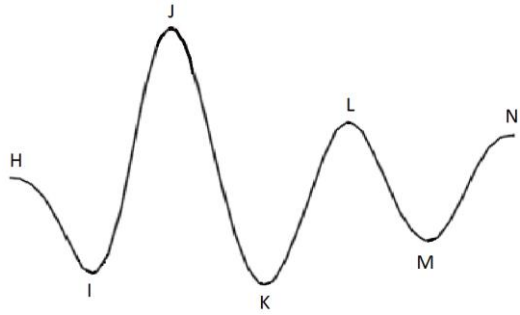


Рис. 4. Форма кардиоцикла БКГ сигнала
Fig. 4. Form of cardio circle of BCG signal

Автоматическое выделение J-пиков на выделенной в результате фильтрации кардио компоненте представляется несложной задачей. Она может быть решена в рамках модельного подхода с несколькими параметрами. Эксперименты с сигналами показали, что хорошую точность дает простейший алгоритм:

- в дискретном сигнале x_1, x_2, …, x_N выделяются точки локального экстремума – точки максимума x_i, для которых $x_{i-1} < x_i$ и $x_i \geq x_{i+1}$, и минимума – для которых $x_{i-1} > x_i$ и $x_i \leq x_{i+1}$;
- для каждого максимума *J* рассчитываем показатель: $Z(J) = X(J) - 2X(I) + X(H)$, где $X(J)$ – высота данного пика, $X(I)$ – уровень сигнала, соответствующий предшествующему ему провалу, $X(H)$ – высота предшествующего пика;
- в качестве *J*-пиков выбираем те максимумы, для которых показатель $Z(J)$ больше трех соседних пиков слева и трех соседних пиков справа.

6. Результаты работы модели распознавания кардиоциклов и их сопоставление с результатами других авторов

В тестовой выборке рассматривались сигналы, полученные от 3-х пациентов, длительностью от 7 до 10 минут каждый. На верхнем графике Рис. 5 приведен

фрагмент такого БКГ сигнала, на котором проставлены метки над J-пиками, автоматически определенные моделью. На нижнем графике показан параллельно снятый ЭКГ сигнал, опираясь на который эксперт может подтвердить правильность проставленных меток. Согласно работе [13] J-пики обычно находится в пределах 200мс-го окна после R-пиков QRS комплекса, отвечающих тому же кардиоциклу.

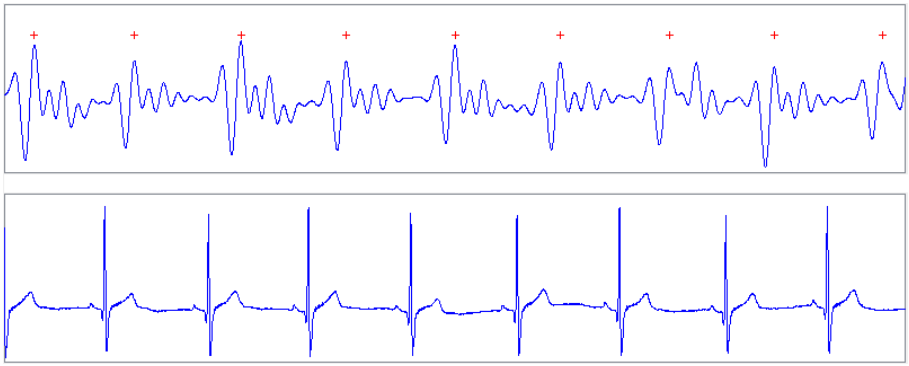


Рис. 5. На верхнем графике показана кардио компонента БКГ сигнала, метками отмечены автоматически распознанные J-пики. На нижнем – одновременно измеренная ЭКГ компонента того же пациента
Fig. 5. The upper graph shows the cardio component of the BCG *signal, the marks indicate automatically recognized J-peaks. At the bottom - simultaneously measured ECG component of the same patient*

Качество распознавания J-пиков моделью можно охарактеризовать двумя показателями: чувствительностью (sensitivity) – отношением числа правильно распознанных J-пиков к общему числу J-пиков в тестовой выборке и относительной ошибкой некорректного определения J-пика. Описанная модель демонстрирует следующие усредненные по всем пациентам значения данных показателей:

Sens = 98.7%, (1)

Err = 0.6%. (2)

В статье [17] рассматривается модель распознавания кардиоциклов, строящаяся на принципах обучения без учителя. Авторы приводят следующие значения показателей точности:

Sens = 49.2%,

Err = 0.09%,

Этот результат существенно уступает результату нашей модели по первому показателю, однако, превосходит его по второму.

В работе [15] предлагается метод, также основанный на принципах обучения без учителя. В ней приводятся значения ложноположительных и

ложноотрицательных ставок: 0.12% и 0.41% соответственно. Однако оговаривается, что столь высокие показатели точности получены для 95.94% сигнала, остальную его часть модель сочла, непригодной для анализа. С учетом этой оговорки можно заключить, что данный результат несколько уступает показателям (1) и (2).

В работе [18] приводится модель распознавания кардиоциклов, базирующаяся на вычислении кросс-корреляции анализируемого сигнала с шаблонными сигналами, и приводятся следующие значения чувствительности и положительной прогностической ценности:

Sens = 95.16%

Ppv = 94.76%

Этот результат также несколько уступает показателям (1) и (2).

Исходя из вышесказанного, можно заключить, что предложенная простая параметрическая модель распознавания кардиоциклов в БКГ сигнале демонстрирует приемлемую точность и, как минимум, не уступает, более сложным, основанным, в частности, на машинном обучении моделям. Данный результат обусловлен в первую очередь качеством получаемого сигнала.

Простота и «вычислительная дешевизна» данной модели позволяет использовать ее на граничных (промежуточных) вычислительных узлах распределенной информационно-коммуникационной системы сбора и обработки данных с множеством датчиков. Использование граничных вычислительных узлов в системе позволяет сократить объём передаваемых в облако данных в 5 раз, что позволяет существенно снизить требования к каналу связи. Кроме этого, вычисления дискретных показателей на основе непрерывного сигнала полностью выносится на граничный узел, что снимает значительную вычислительную нагрузку с серверных мощностей.

7. Заключение

В работе был рассмотрен подход к решению задачи мониторинга сердечной активности пациента, основанный на съеме и анализе БКГ сигнала. Математическая модель выделения кардиоциклов в БКГ сигнале состоит из двух этапов: на первом с помощью полосного фильтра Баттерворта выделяется кардио компонента, на втором – ищутся J-пики кардио комплексов. Точность распознавания J-пиков не уступает, а в ряде случаев превосходит показатели, приведенные в работах других авторов, использующих более сложные математические модели.

Предложенная в работе математическая модель готова для размещения на граничных (промежуточных) вычислительных узлах, что существенно сокращает требования к полосе пропускания каналов и вычислительных мощностей центров обработки данных.

Список литературы.

[1]. https://forinnovations.ru.
[2]. J.W. Gordon. Certain molar movements of the human body produced by the circulation of the blood. *J. Anatomy Physiol.*, vol. 11, 1877, pp. 533–536.
[3]. А.С. Девятисильный. Интерпретация измерений оптического акселерометра. Журнал технической физики, том 74, вып. 9, 2004, стр. 141-142.
[4]. RMFIT R-series sensor. https://www.emfit.com/r-series-sensors.
[5]. A. Ahmed and E. Ahmed. A survey on mobile edge computing. In Proc. of the 10th International Conference on Intelligent Systems and Control (ISCO), 2016, pp. 1-8.
[6]. W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu. Edge Computing: Vision and Challenges. IEEE Internet of Things Journal, vol. 3, no. 5, Oct. 2016, pp. 637-646.
[7]. Прозоров А.А., Бугаев В.И., Царенко С.В. Устройство бесконтактной регистрации основных биометрических показателей пациента в непрерывном режиме в состоянии лежа: заявка на патент РФ №2017101824 от 20 января 2017 года.
[8]. Joonas Paalasmaa. Monitoring Sleep with Force sensor Measurement. Doctoral dissertation, Department of Computer Science, Series of Publications A, report A-2014-2. 2014, 59 p.
[9]. W. Chen, X. Zhu, T. Nemoto, K. Kitamura, K. Sugitani, and D. Wei. Unconstrained monitoring of long-term heart and breath rates during sleep. Physiol. Meas., vol. 29, , 2008, pp. N1–N10.
[10]. O. Postolache, P. Silva Girao, G. Postolache, and M. Pereira. Vital signs monitoring system based on EMFi sensors and wavelet analysis. In Proc. of the Instrumentation and Measurement Technology Conference, 2007, pp. 1–4.
[11]. W. Xu, W. A. Sandham, A. C. Fisher, and M. Conway. Detection of the seismocardiogram W complex based on multiscale edges. In Proc. of the 18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 1996, pp. 1023–1024.
[12]. D. H. Phan, S. Bonnet, R. Guillemaud, and N. Y. P. T. E. Castelli. Estimation of respiratory waveform and heart rate using an accelerometer. In Proc. of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology, 2008, pp. 4916–4919.
[13]. Joan Gomez-Clapers , Albert Serra-Rocamora , Ramon Casanella , Ramon Pallas-Areny. Towards the standardization of ballistocardiography systems for J-peak timing measurement. Measurement, vol. 58, 2014, pp. 310-316
[14]. S. Junnila, A. Akhbardeh, A. V¨arri, and T. Koivistoinen. An EMFi-film sensor based ballistocardiographic chair: Performance and cycle extraction method. In Proc. IEEE Workshop on Signal Processing Systems Design and Implementation, 2005, pp. 373–377.
[15]. Christoph Bruser, Kurt Stadlthanner, Stijn de Waele, and Steffen Leonhardt. Adaptive Beat-to-Beat Heart Rate Estimation in Ballistocardiograms. IEEE Transactions on Information Technology in Biomedicine, vol. 15, issue 5, Sept. 2011, pp. 778-786.
[16]. S. Haykin. Adaptive Filter Theory. 4rd Edition. Paramus, NJ: Prentice-Hall, 2001.
[17]. J. Paalasmaa and M. Ranta. Detecting heartbeats in the ballistocardiogram with clustering. In Proc. of the ICML/UAI/COLT 2008 Workshop on Machine Learning for Health-Care Applications, 2008, pp. 1-4.
[18]. J. H. Shin, B. H. Choi, Y. G. Lim, D. U. Jeong, and K. S. Park. Automatic ballistocardiogram (BCG) beat detection using a template matching approach. In Proc. of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2008, pp. 1144–1146.

Ballistocardiogram analysis on edge computing nodes

[1] A.S. Nuzhny <*nuzhny@inbox.ru*>
[2] *A.A.Prozorov* <*prozorov.aa@mipt.ru*>
[2] *V.I. Bugaev* <*vik362@list.ru*>
[2] *N.D.Shuvalov* <*shuvalovnickolay@gmail.com*>
[2] *V.V. Podumov* <*valdus@yandex.ru*>
[1] *Nuclear Safety Institure of the Russian Academy of Sciences,*
52 Bolshaya Tulskaya st., Moscow 115191, Russia
[2] *Moscow Institute of Physics and Technology,*
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russia

Abstact. In this paper we present the contactless method of analyzing the cardiac activity of a person based on recording and analyzing a ballistic cardiogram signal. A measuring device for registration of microscopic movements of the body uses a piezoelectric sensor of high sensitivity. Due to sensor's high sensitivity, the level of background noise is higher than the signal level, so mathematical methods are used for noise reduction. Butterworth filter is used to extract cardiac signal. This approach is more computationally efficient compared to machine learning-based methods, and can be implemented on an edge computing node to which several sensors are connected. The quality of the signal obtained after filtration allows us to detect cardiac cycles. The algorithm used for detection of heartbeats proposed in this paper is also computationally simple enough to be implemented at the edge node. After preprocessing described above data is transmitted to the datacenter (cloud).

Keywords: Ballistocardiography; cardiac activity; Butterworth filter; Internet of things.

DOI: 10.15514/ISPRAS-2018-30(2)-12

For citation: Nuzhny A.S., Prozorov A.A., Bugaev V.I., Shuvalov N.D., Podumov V.V. Ballistocardiogram analysis on edge computing nodes. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 2, 2018, pp. 251-262 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-12

References.

[1]. https://forinnovations.ru.
[2]. J.W. Gordon. Certain molar movements of the human body produced by the circulation of the blood. *J. Anatomy Physiol.,* vol. 11, 1877, pp. 533–536.
[3]. A. S. Devyatisilny, "Intrapretation of optical accelerometer measurements", Journal of technical physics, vol 74, issue 9, 2004, pp. 141-142 (in Russian).
[4]. RMFIT R-series sensor. https://www.emfit.com/r-series-sensors.
[5]. A. Ahmed and E. Ahmed. A survey on mobile edge computing. In Proc. of the 10th International Conference on Intelligent Systems and Control (ISCO), 2016, pp. 1-8.
[6]. W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu. Edge Computing: Vision and Challenges. IEEE Internet of Things Journal, vol. 3, no. 5, Oct. 2016, pp. 637-646.
[7]. Prozorov A.A., Bugaev V.I., Tsarenko S.V. A device for continuous contactless measurements of major biometric indexes of a lying patient: Russian patent application №2017101824 from 20 Jan. 2017.

[8]. Joonas Paalasmaa. Monitoring Sleep with Force sensor Measurement. Doctoral dissertation, Department of Computer Science, Series of Publications A, report A-2014-2. 2014, 59 p.

[9]. W. Chen, X. Zhu, T. Nemoto, K. Kitamura, K. Sugitani, and D. Wei. Unconstrained monitoring of long-term heart and breath rates during sleep. Physiol. Meas., vol. 29, , 2008, pp. N1–N10.

[10]. O. Postolache, P. Silva Girao, G. Postolache, and M. Pereira. Vital signs monitoring system based on EMFi sensors and wavelet analysis. In Proc. of the Instrumentation and Measurement Technology Conference, 2007, pp. 1–4.

[11]. W. Xu, W. A. Sandham, A. C. Fisher, and M. Conway. Detection of the seismocardiogram W complex based on multiscale edges. In Proc. of the 18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 1996, pp. 1023–1024.

[12]. D. H. Phan, S. Bonnet, R. Guillemaud, and N. Y. P. T. E. Castelli. Estimation of respiratory waveform and heart rate using an accelerometer. In Proc. of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology, 2008, pp. 4916–4919.

[13]. Joan Gomez-Clapers , Albert Serra-Rocamora , Ramon Casanella , Ramon Pallas-Areny. Towards the standardization of ballistocardiography systems for J-peak timing measurement. Measurement, vol. 58, 2014, pp. 310-316

[14]. S. Junnila, A. Akhbardeh, A. V¨arri, and T. Koivistoinen. An EMFi-film sensor based ballistocardiographic chair: Performance and cycle extraction method. In Proc. IEEE Workshop on Signal Processing Systems Design and Implementation, 2005, pp. 373–377.

[15]. Christoph Bruser, Kurt Stadlthanner, Stijn de Waele, and Steffen Leonhardt. Adaptive Beat-to-Beat Heart Rate Estimation in Ballistocardiograms. IEEE Transactions on Information Technology in Biomedicine, vol. 15, issue 5, Sept. 2011, pp. 778-786.

[16]. S. Haykin. Adaptive Filter Theory. 4rd Edition. Paramus, NJ: Prentice-Hall, 2001.

[17]. J. Paalasmaa and M. Ranta. Detecting heartbeats in the ballistocardiogram with clustering. In Proc. of the ICML/UAI/COLT 2008 Workshop on Machine Learning for Health-Care Applications, 2008, pp. 1-4.

[18]. J. H. Shin, B. H. Choi, Y. G. Lim, D. U. Jeong, and K. S. Park. Automatic ballistocardiogram (BCG) beat detection using a template matching approach. In Proc. of the 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2008, pp. 1144–1146.

Моделирование осесимметричных течений вязкой несжимаемой жидкости методом конечных элементов с частицами PFEM-2 в программном комплексе Kratos с открытым кодом

[1] *Е.В. Смирнова <alena.davidova@ispras.ru>*
[1,2] *И.К. Марчевский <iliamarchevsky@mail.ru>*
[2] *В.О. Бондарчук <viktoriia.bondarchuk@gmail.com>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25.*
[2] *Московский государственный технический университет им. Н.Э. Баумана, 105005, Россия, Москва, ул. 2-я Бауманская, д. 5*

Аннотация. Получены необходимые соотношения для реализации алгоритма моделирования осесимметричных течений вязкой несжимаемой жидкости в методе конечных элементов с частицами PFEM-2. Осесимметричная модель реализована в программном комплексе с открытым исходным кодом Kratos на основе существующего модуля для решения плоских задач. Была проведена валидация осесимметричной модели на тестовых задачах. В качестве модельных рассмотрены задачи о течении в трубе (задача Пуазейля) и задача о моделировании падения капли в глубокий слой жидкости. Численное решение, полученное методом конечных элементов с частицами, для задачи Пуазейля показало удовлетворительное согласие с аналитическим; решение задачи о падении капли в слой глубокий жидкости в комплексе Kratos сравнивалось с результатом моделирования в программном пакете Gerris с открытым исходным кодом. Результаты расчетов также удовлетворительно согласуются.

Ключевые слова: метод конечных элементов с частицами; вязкая несжимаемая среда; осесимметричное течение; задача Пуазейля; падение капли в слой жидкости.

DOI: 10.15514/ISPRAS-2018-30(2)-13

Для цитирования: Смирнова Е.В., Марчевский И.К., Бондарчук В.О. Моделирование осесимметричных течений вязкой несжимаемой жидкости методом конечных элементов с частицами PFEM-2 в программном комплексе Kratos с открытым кодом. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 263-284. DOI: 10.15514/ISPRAS-2018-30(2)-13

1. Введение

Метод конечных элементов с частицами (Particle Finite Element Method, PFEM [1]) представляет собой сеточный эйлерово-лагранжев метод вычислительной гидродинамики несжимаемой среды. Наиболее эффективным является его модификация PFEM-2 [2], в рамках которой расчет ведется на неподвижной конечноэлементной сетке, через которую перемещаются частицы – «носители» скорости среды. Движение частиц позволяет моделировать движение среды, обусловленное конвективным переносом, а влияние остальных процессов (вязкость, градиент давления) учитывается путем решения соответствующих дифференциальных уравнений методом конечных элементов.

На сегодняшний день единственным программным комплексом, в котором реализованы PFEM/PFEM-2, является пакет Kratos с открытым исходным кодом (http://kratos-wiki.cimne.upc.edu). Kratos – конечноэлементный пакет, позволяющий решать задачи теории упругости, механики разрушения, гидродинамики и т.д. В текущей версии Kratos можно моделировать плоские и трехмерные течения жидкостей, однако возможность решения задач в осесимметричной постановке до настоящего времени не реализована.

Целью данной работы является разработка алгоритма моделирования несжимаемых осесимметричных течений методом конечных элементов с частицами и его реализация в рамках программного комплекса Kratos.

2. Постановка задачи

2.1 Постановка и принцип решения задачи методом PFEM-2

Движение вязкой несжимаемой жидкости описывается уравнениями Навье – Стокса и уравнением неразрывности [3]

$$\rho\left(\frac{\partial\vec{v}}{\partial t}+(\vec{v}\cdot\nabla)\vec{v}\right)=\nabla\cdot(\mu\nabla\vec{v})-\nabla p+\rho\vec{g};$$

$$\nabla\cdot\vec{v}=0,$$

где ρ – плотность, $\vec{v}=(u,v,w)$ – скорость среды; μ – коэффициент динамической вязкости; p – давление; $\vec{g}=(g_1,g_2,g_3)$ – вектор массовой плотности объемных сил. Система уравнений дополняется необходимыми начальными и граничными условиями, чаще всего в их роли выступают:

- условие непротекания: $\vec{v}^{\,n}|_{\Gamma_1}=0$,
- условие прилипания: $\vec{v}^{\,\tau}|_{\Gamma_2}=0$,
- заданные величины для скорости или давления: $\vec{v}|_{\Gamma_3}=\vec{v}_0$, $p|_{\Gamma_4}=p_0$.

Здесь индексы n и τ указывают на нормальные и касательные компоненты вектора скорости соответственно.

Смысл использования лагранжевых методов заключается в возможности учета конвективного слагаемого в уравнении (1) путем моделирования движения частиц – «носителей» скорости – по расчетной области. Вводя понятие материальной (субстанциональной) производной $\frac{\mathrm{d}}{\mathrm{d}t} = \frac{\partial}{\partial t} + (\vec{v} \cdot \nabla)$, учитывающей изменение соответствующей величины в лагранжевой частице, систему уравнений (1)-(2) можно записать в виде

$$\rho \frac{\mathrm{d}\vec{v}}{\mathrm{d}t} = \nabla \cdot (\mu \nabla \vec{v}) - \nabla p + \rho \vec{g};$$
$$\nabla \cdot \vec{v} = 0.$$

Полную производную по времени заменим разностным аналогом:

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} \approx \frac{f(t+\Delta t) - f(t)}{\Delta t} + O(\Delta t),$$

С учетом (4) уравнения (3) переходят в систему уравнений (в частных произвоных по пространственным координатам), которая должна решаться на каждом n-м шаге по времени:

$$\rho \frac{\vec{v}^{(n+1)}}{\Delta t} \approx \nabla \cdot (\mu \nabla \vec{v}^{(n+\theta)}) - \nabla p^{(n+1)} + \rho \vec{g} + \rho \frac{\vec{v}^{(n)}}{\Delta t},$$
$$\nabla \cdot \vec{v}^{(n+1)} = 0,$$

Параметр θ определяет способ учета «вязкого» слагаемого: $\theta = 0$ и $\theta = 1$ для явной и неявной аппроксимации соответственно.

2.2 Постановка осесимметричной задачи

Примем, что вязкая несжимаемая жидкость с плотностью ρ и вязкостью ν занимает область цилиндрической формы и находится в поле массовой силы $\vec{g}$, действующей вдоль оси области. Введем систему координат $\{r, \phi, z\}$, направив ось Oz по оси симметрии области, ось Or – перпендикулярно ей. Будем расматривать осесимметричные незакрученные течения среды ($v_\phi = 0$), когда окружное перемещение частиц равно нулю. Тогда можно перейти к двумерной задаче гидродинамики в области $\Omega = \{\{r, z\} : r \in \{r_1, r_2\}, z \in \{z_1, z_2\}\}$ (рис. 1). Течение жидкости описывается уравнениями (3), при этом все дифференциальные операторы подразумеваются действующими в цилиндрической системе координат $\{r, z\}$. Граничные условия (ГУ) соответствуют указанным выше, однако ГУ на оси области (при $r = 0$) является условие ограниченности радиальной скорости.

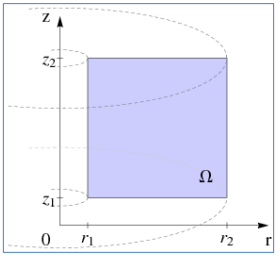


Рис. 1. Расчетная область
Fig. 1. The computational domain

3. Краткое описание метода PFEM-2

Эйлерово-лагранжев метод PFEM-2 [2, 4] предполагает рассмотрение в области течения неподвижной конечноэлементной сетки и набора частиц, движущихся сквозь ячейки сетки. Частицы перемещаются по полю скоростей, одновременно являясь его «носителями», что обеспечивает учет конвективного слагаемого из уравнения (2). При этом система уравнений (5) решается на неподвижной сетке методом конечных элементов [5]. Отметим, что при моделировании многофазных течений частицы также являются маркерами соответствующей фазы.

Расчет течения методом PFEM-2 производится по следующему алгоритму (операции 2–5 выполняются в цикле по времени).

1) В расчетной области строится конечноэлементная сетка. Задается начальное распределение частиц, при котором на каждую ячейку сетки приходится, как правило, несколько десятков частиц.
2) По известному полю скоростей с n-го шага по времени частицы переносятся вдоль линий тока в новые положения.
3) Скорости частиц проецируются (интерполируются) на узлы конечноэлементной сетки.
4) Решается система уравнений (5); найденные ранее скорости являются начальным приближением для значений скоростей на следующем шаге.
5) По найденному полю скоростей в узлах сетки корректируются скорости частиц. При необходимости некоторые частицы исключаются из расчета, а в ячейки с малым числом частиц вводятся дополнительные частицы.

Алгоритмы перечисленных операций с особенностями их реализации описаны в публикациях, посвященных PFEM-2. В данной работе приведем лишь их краткое описание, необходимое для понимания общей идеологии метода PFEM-2. При этом остановимся более подробно на решении методом конечных элементов системы уравнений (3). Для простоты будем рассматривать задачу в двумерной (плоской) постановке и считать, что массовые силы направлены вдоль второй координаты, т.е. вектор $\vec{g}$ имеет координаты $(0, g)$.

3.1 Перемещение частиц

Рассмотрим частицу жидкости с номером p. Зная координаты частицы в некоторый момент времени t_n, ее положение в момент времени $t_{n+1} > t_n$ можно определить выражением

$$\vec{x}_p^{(n+1)} = \vec{x}_p^{(n)} + \int_{t_n}^{t_{n+1}} \vec{v}^{\,t}(\vec{x}_p^{\,t})dt,$$

где $\vec{v}$ – скорость, а индекс t означает, что вычисление должно производиться непрерывно по времени. Скорость частицы в момент времени t_{n+1} определяется аналогичным образом:

$$\vec{v}_p^{(n+1)} = \vec{v}_p^{(n)} + \int_{t_n}^{t_{n+1}} \vec{a}^{\,t}(\vec{x}_p^{\,t})dt,$$

где $\vec{a}$ – ускорение. Однако уравнения (6)-(7) не могут быть решены напрямую, так как перемещение частицы зависит от ее скорости в каждый момент времени, а скорость – от координаты частицы. К тому же для решения уравнений производится дискретизация по времени, поэтому значения величин известны только в определенные моменты времени t_{n-1}, t_n, t_{n+1} и т.д. Для приближенного расчета положения частицы p в момент времени t_{n+1} из формулы (6) получаем:

$$\vec{x}_p^{(n+1)} \approx \vec{x}_p^{(n)} + \int_{t_n}^{t_{n+1}} \vec{v}^{\,n}(\vec{x}_p^{\,t})dt.$$

Такой подход фактически означает интегрирование движения частицы вдоль линии тока (т.е. в постоянном поле скоростей). Скорость частицы может быть найдена как

$$\vec{v}_p^{(n+1)} \approx \vec{v}_p^{(n)} + (1-\theta)\int_{t_n}^{t_{n+1}} \vec{a}^{(n)}(\vec{x}_p^{\,t})dt + \theta\int_{t_n}^{t_{n+1}} \vec{a}^{(n+1)}(\vec{x}_p^{\,t})dt, \quad 0 \le \theta \le 1.$$

Перемещение частиц может осуществляться двумя способами. Первый, более простой способ заключается в аппроксимации интегралов в правой части выражения (9) следующим образом (принимаем в простейшем случае $\theta = 0$ и дополнительно полагаем, что для ускорения справедливо $\vec{a}^{(n)} \approx \vec{a}^{(n+1)}$):

$$\vec{v}_p^{(n+1)} \approx \vec{v}_p^{(n)} + \vec{a}^{(n)}(\vec{x}_p^{(n)})\Delta t,$$

где $\Delta t = t_{n+1} - t_n$. Затем согласно (8) находится положение частицы выражению. Однако при использовании данного способа положение частицы

в момент времени t_{n+1} может существенно отличаться от истинного. Поэтому для повышения точности на этапе перемещения частиц применяют второй способ – интегрирование вдоль линий тока.

Данный способ предполагает перемещение частицы по известному на шаге t_n полю скорости. При этом отрезок $[t_n, t_{n+1}]$ дробится на M более мелких:

$$\left\{t_n^0, t_n^1, t_n^2, \ldots, t_n^{M-1}, t_n^M\right\},$$

где $t_n^m = t_n + m\tau,\ \tau = \Delta t/M$. Перемещение частицы вдоль линий тока происходит согласно выражению

$$\vec{x}_p^{(m+1)} = \vec{x}_p^{(m)} + \vec{v}_p^{(n)}(\vec{x}_p^{(m)})\tau, \quad 0 \le m \le M-1.$$

Иллюстрация интегрирования вдоль линий тока приведена на рис. 2. Стоит отметить, что скорость частиц, найденная таким образом, позже будет скорректирована.

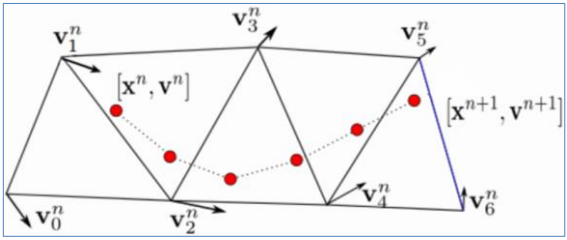


Рис. 2. Интегрирование вдоль линий тока
Fig. 2. Integration along the streamlines

3.2 Проецирование скоростей с частиц на сетку

Для описания механизма проецирования величин с частиц на узлы заданной в расчетной области сетки рассмотрим скалярное поле Λ. С каждой i-й частицей ассоциировано значение λ_i. Для того, чтобы вычислить значение Λ_j для каждого j-го узла сетки, необходимо использовать все частицы в соседних к узлу j элементах (рис. 3). Полагается, что частицы, расположенные ближе к узлу j, дают больший вклад в значение Λ_j. Для выполнения данного требования вводится весовая функция, с помощью которой будет проводиться вычисление значения поля в узлах. В качестве весовой функции выбрана стандартная функция формы элемента $N^j(\vec{x})$; при этом вес вклада в значение Λ_j от частицы i равен $N^j(\vec{x}_i) = N_i^j$.

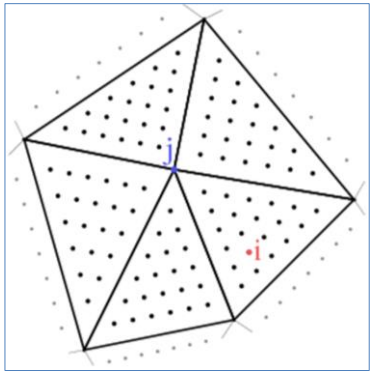


Рис. 3. Узел с индексом j и соседние к нему конечные элементы с частицами

Fig. 3. Node with the index j and the neighbouring finite elements with the particles

Таким образом, значение λ в узле j определяется выражением

$$\Lambda_j = \frac{\sum_i^n \lambda_i N_i^j}{\sum_i^n N_i^j},$$

где верхний индекс у функции формы указывает на принадлежность функции конкретному узлу конечного элемента. Спроецированная на узлы сетки с помощью (10) скорость определяется аналогичнно выражением

$$\hat{\vec{V}}_j^{n+1} = \frac{\sum_i^n \vec{v}_i^n N_i^j}{\sum_i^n N_i^j}.$$

Обозначение $(\hat{\cdot})$ указывает на то, что вычисленная скорость является первым приближением к скорости на новом шаге по времени. Данная величина уточняется в итерационном процессе решения уравнений движения.

3.3 Решение системы уравнений

При использовании метода конечных элементов (МКЭ) значение некоторой величины λ в точке $\vec{x}$ вычисляется как сумма произведений значений этой величины в узлах на значения соответствующих узлам функций формы [5]:

$$\lambda(\vec{x},t) = \sum_{j=1}^{K} \lambda_j(t) N^j(\vec{x}).$$

Здесь K – количество узлов в элементе (рассматриваем локальную нумерацию узлов в конечном), λ_j – значение величины λ в узле j, $N^j(\vec{x})$ – функция формы j-го узла.

Ограничимся рассмотрением симплексных (треугольных в плоском и тетраэдральных в пространственном случае) конечных элементов 1-го порядка с линеными функциями формы, частные производные которых постоянны. Отметим, что именно в таком виде PFEM-2 реализован в KRATOS.

Представляя u,v,p в виде (12), уравнения (5), полагая $\theta = 1$, принимают вид:

$$\frac{\rho}{\Delta t}\sum_i^K u_i^{n+1} N^i - \nabla\cdot(\mu\nabla\sum_i^K u_i^{n+1} N^i) + \nabla\sum_i^K p_i^{n+1} N^i = \frac{\rho}{\Delta t}\sum_i^K u_i^n N^i,$$

$$\frac{\rho}{\Delta t}\sum_i^K v_i^{n+1} N^i - \nabla\cdot(\mu\nabla\sum_i^K v_i^{n+1} N^i) + \nabla\sum_i^K p_i^{n+1} N^i = \rho g + \frac{\rho}{\Delta t}\sum_i^K v_i^n N^i,$$

$$\nabla\cdot\sum_i^K \vec{v}_i^{\,n+1} N^i = 0.$$

Выполняя стандартную процедуру МКЭ и умножая уравнения на функции формы N^j и далее интегрируя их по конечному элементу Ω, получаем:

$$\begin{aligned}
&\int_\Omega \frac{\rho}{\Delta t}\sum_i^K u_i^{n+1} N^j N^i\, d\Omega - \int_\Omega N^j \nabla\cdot(\mu\sum_i^K u_i^{n+1}\nabla N^i)\, d\Omega + \int_\Omega \sum_i^K p_i^{n+1} N^j \nabla N^i\, d\Omega = \\
&\qquad = \int_\Omega \frac{\rho}{\Delta t}\sum_i^K u_i^n N^j N^i\, d\Omega, \\
&\int_\Omega \frac{\rho}{\Delta t}\sum_i^K v_i^{n+1} N^j N^i\, d\Omega - \int_\Omega N^j \nabla\cdot(\mu\sum_i^K v_i^{n+1}\nabla N^i)\, d\Omega + \int_\Omega \sum_i^K p_i^{n+1} N^j \nabla N^i\, d\Omega = \\
&\qquad = \int_\Omega \rho g N^j\, d\Omega + \int_\Omega \frac{\rho}{\Delta t}\sum_i^K v_i^n N^j N^i\, d\Omega, \\
&\int_\Omega \sum_i^K (\vec{v}_i^{\,n+1}\cdot\nabla N^i)\, d\Omega = 0.
\end{aligned} \tag{14}$$

С учетом допущений о постоянстве градиентов функций для интегралов в (14) можно ввести следующие обозначения:

$$\tilde{M}_{ij} = \int_\Omega N^j N^i\, d\Omega;$$

$$\tilde{G}_{ij} = \int_\Omega N^j \nabla N^i\, d\Omega = \frac{1}{3}\nabla N^i V;$$

$$\tilde{L}_{ij} = \int_\Omega N^j \nabla(\nabla N^i) d\Omega = \int_\Omega \nabla(N^j \nabla N^i) d\Omega - \int_\Omega \nabla N^j \nabla N^i d\Omega =$$

$$= \oint_S N^j \nabla N^i \cdot \vec{n} dS - \int_\Omega \nabla N^j \nabla N^i d\Omega \approx - \int_\Omega \nabla N^j \nabla N^i d\Omega = -\nabla N^j \nabla N^i V.$$

(в последнем соотношении пренебрегли поверхностным интегралом.)

С учетом введенных обозначений систему (14) удобно записать в матричной (блочной) форме:

$$\left[\begin{array}{c|c} \frac{\rho}{\Delta t}[M] + \mu[L] & [G] \\ \hline [D] & 0 \end{array}\right] \begin{pmatrix} \{V\}^{n+1} \\ \{P\}^{n+1} \end{pmatrix} = \begin{pmatrix} \rho g\{R\} + \frac{\rho}{\Delta t}\{V\}^n \\ 0 \end{pmatrix},$$

здесь $\{V\}^n = \{u_1^n, v_1^n, u_2^n, v_2^n, u_3^n, v_3^n\}^T$ – вектор-столбец из компонент скорости в узлах конечного элемента, $\{P\}^n = \{p_i^n, p_j^n, p_k^n\}^T$ – вектор-столбец из значений давления в узлах КЭ, $\{R\} = \frac{1}{3} V\{0,1,0,1,0,1\}^T$ – вектор-столбец, компонентами которого являются нули и третья часть объема КЭ. Все матрицы, входящие в (16), состоят из соответствующих им элементов (15), упорядоченных необходимым образом. Матрицы $[M]$ и $[L]$ – квадратные размером $(2n \times 2n)$, где $n = 3$ – количество узлов в конечном элементе, а двойка указывает на количество компонент вектора скорости. Матрицы $[G]$ и $[D]$ имеют размер $(2n \times n)$ и $(n \times 2n)$ соответственно и состоят из элементов $\tilde{G}_{ij}$. По сложившейся в МКЭ традиции будем называть $[M]$ матрицей масс, $[L]$ – матрицей вязких напряжений, $[D]$ – матрицей дивергенции, $[G]$ – матрицей градиента.

Для составления глобальной системы линейных алгебраических уравнений необходимо для каждого конечного элемента записать локальную систему уравнений (16). Затем локальные матрицы с помощью специальной процедуры аггрегации «встраиваются» в глобальную матрицу [5]. Полученная в итоге система уравнений решается итерационными методами.

Для того, чтобы реализовать возможность расчета осесимметричных течений, на базе готовых модулей приложения `pfem2_application` для решения задач гидродинамики методом PFEM-2 необходимо модифицировать соответствующие программные модули, отвечающие за расчет коэффициентов соответствующих матриц и интерпретацию результатов решения.

3.4 Коррекция положения частиц

После решения системы уравнений скорости частиц, определенные изначально по значениям старого поля скоростей, должны быть обновлены. Для этого используется поправка, вносимая в скорости частиц [4]:

$$\delta\vec{v}_j^{\,n+1} = \vec{v}_j^{\,n+1} - \hat{\vec{v}}_j^{\,n+1}.$$

Утверждается, что использование для коррекции скоростей частиц величины поправки $\delta\vec{v}_j^{\,n+1}$ позволяет уменьшить численную диффузию, особенно на сетках с разноразмерными ячейками.

3.5 Об алгоритме поиска межфазной границы

Для моделирования течений со свободной поверхностью, а также многофазных течений требуется определение положения границы фаз. Для этого частицам приписывается параметр – маркер фазы, к которой они принадлежат. Положение межфазной границы при этом не должно определяться параметрами отдельных частиц. Например, на рис. 4 показано, что некоторые частицы могут располагаться не с нужной стороны границы раздела фаз.

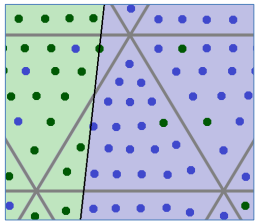


Рис. 4. Конечные элементы, частицы и положение границы раздела фаз
Fig. 4. Finite elements, particles and the interface between phases position

Предполагая, что в расчетной области присутствует две фазы, можно промаркеровать частицы легкой фазы параметром $+1$, а частицы тяжелой – параметром -1. Тогда для вычисления положения границы раздела фаз можно ввести «псевдо»-функцию уровня ϕ, изоповерхности $\phi = 0$ которой будут соответствовать межфазная граница. Значения функции ϕ в узле j конечноэлентной сектки находится по формуле

$$\phi_j = \frac{\sum_i \mathrm{sign}_i N_i^j}{\sum_i N_i^j},$$

где суммирование ведется по частицам в соседних к узлу j элементах, а sign_i определяет знак маркера i-й частицы.

4. Решение осесимметричной задачи

4.1. Представление функций формы конечных элементов и дифференциальных операторов в осесимметричной задаче

При рассмотрении плоской задачи функция формы N^i i-го узла конечного элемента $\{i,j,k\}$ в точке $P(P_1,P_2)$, лежащей внутри конечного элемента, определялась как отношение площадей треугольников $\{P,j,k\}$ и $\{i,j,k\}$. В осесимметричном случае такой функции формы соответствует функция, определяемая как отношение объемов фигур, полученных из треугольников $\{P,j,k\}$ и $\{i,j,k\}$ их вращением вокруг оси симметрии Oz (рис. 2). Объем подобной фигуры можно найти точно, но в первом приближении он равен $V_{ijk} = 2\pi r_{ijk}^{mid} S_{ijk}$, где S_{ijk} – площадь треугольника $\{i,j,k\}$, а r^{mid} – радиальная координата его средней точки. Таким образом, функция формы осесимметричного конечного элемента определяется выражением $N^i(r,z) = V_{Pjk}/V_{ijk}$. Если приближенно принять $r_{ijk}^{mid} \approx r_{Pjk}^{mid}$, то функции формы формально остаются прежними, $N^i(r,z) = S_{Pjk}/S_{ijk}$.

Рис. 2. К определению функций формы:
a – двумерные функции формы, b – осесимметричные функции формы
Fig. 2. To the shape functions definition:
a – two-dimensional shape functions, b – axisymmetric shape functions

Система линейных алгебраических уравнений, полученная после ансамблирования локальных систем (16), сохраняет свою структуру для любой двумерной постановки. Однако чтобы перейти к цилиндрическим координатам, необходимо учесть изменение правила действия дифференциальных операторов и якобиан преобразования при вычислении интегралов.

В цилиндрической системе координат $\{r,\phi,z\}$ с условием независимости всех рассматриваемых величин от окружной координаты ($\lambda(r,\phi,z) = \lambda(r,z)$) дифференциальные операторы градиента скалярного поля и дивергенции векторного поля записываются следующим образом:

$$\nabla F(r,z) = \frac{\partial F}{\partial r}\vec{e}_1 + \frac{\partial F}{\partial z}\vec{e}_2, \quad \nabla\cdot\vec{a}(r,z) = \frac{\partial a^r}{\partial r} + \frac{a^r}{r} + \frac{\partial a^z}{\partial z},$$

где $\vec{e}_1$ и $\vec{e}_2$ – орты системы координат $\{r,z\}$, а обозначения a^r и a^z отвечают соответствующим компонентам векторного поля $\vec{a}$. Якобиан перехода к цилиндрическим координатам $J = r$.

4.2. Интегрирование уравнений движения в конечно-элементном представлении

Примем следующие допущения и обозначения:

- как и ранее, считаем функции формы N^i линейными функциями координат r и z;
- обозначим за r^{mid} радиальную координату средней точки конечного элемента Ω;
- $\int_\Omega r d\Omega = \frac{\overline{V}}{2\pi} = V$, где $\overline{V}$ – объем конечного элемента, а V – нормированный объем;
- $\int_\Omega N_i(\vec{x}) r\, d\Omega \approx \overline{N}_i V$;
- в первом приближении считаем координату r постоянной на конечном элементе и равной r^{mid}.

Перейдем к интегрированию компонент матриц, входящих в локальную систему уравнений (16), и определим «добавочные члены», возникающие в осесимметричных задачах. Впоследствии эти «добавочные слагаемые» учтены в соответствующем модуле для программного комплекса Kratos.

Действие оператора градиента в цилиндрических координатах $\{r,z\}$ формально не отличается от действия градиента в «плоской» системе координат $\{x,y\}$. Таким образом, матрица $[G]$, входящая в систему (16), не изменится при переходе к цилиндрическим координатам, а при ее интегрировании производные функции формы необходимо умножить на объем конечного элемента V.

Рассмотрим **уравнение неразрывности** в цилиндрических координатах и запишем его с учетом конечноэлементного представления скорости:

$$\nabla\cdot\vec{v} = \frac{\partial v^r}{\partial r} + \frac{v^r}{r} + \frac{\partial v^z}{\partial z} \approx \sum_{j=1}^{K}\left[v_j^r\left(\frac{\partial N^j}{\partial r} + \frac{N^j}{r}\right) + v_j^z \frac{\partial N^j}{\partial z}\right] =$$

$$= \sum_{j=1}^{K}\left(v_j^r \frac{\partial N^j}{\partial r} + v_j^z \frac{\partial N^j}{\partial z}\right) + \sum_{j=1}^{K} v_j^r \frac{N^j}{r}.$$

Проинтегрируем получившееся выражение:

$$\int_{\Omega}\sum_{j=1}^{K}\left(v_j^r\frac{\partial N^j}{\partial r}+v_j^z\frac{\partial N^j}{\partial z}\right)r\,d\Omega+\int_{\Omega}\sum_{j=1}^{K}v_j^r\frac{N^j}{r}r\,d\Omega=$$

$$=\sum_{j=1}^{K}\left(v_j^r\frac{\partial N^j}{\partial r}+v_j^z\frac{\partial N^j}{\partial z}\right)V+\sum_{j=1}^{K}(v_j^r\int_{\Omega}N^j\,d\Omega).$$

Интеграл от функции формы можно посчитать аналитически или численно интегрированием по гауссовым точкам; с учетом выбранных допущений имеем

$$\sum_{j=1}^{K}(v_j^r\int_{\Omega}N^j\,d\Omega)\approx\frac{1}{3}\sum_{j=1}^{K}v_j^rS,$$

где S – площадь треугольного конечного элемента.

Выражение (18) содержит два слагаемых. Первое из них после формальной подстановки $r\to x,\ z\to y$ будет являться конечноэлементным представлением дивергенции в координатах $\{x,y\}$. Выражение (19) представляет собой «добавочное слагаемое» к матрице дивергенции $[D]$, которая учитывает переход к цилиндрической системе координат.

Рассмотрим **«вязкое» слагаемое**. Проинтегрируем его по объему конечного элемента с учетом якобиана цилиндрической системы координат:

$$\int_{\Omega}N^j\nabla(\nabla N^i)r\,d\Omega=\int_{\Omega}\nabla\cdot(rN^j\nabla N^i)d\Omega-\int_{\Omega}\nabla(N^jr)\nabla N^i d\Omega=$$

$$=\oint_{S}rN^j\nabla N^i\cdot\vec{n}\,dS-\int_{\Omega}\nabla(N^jr)\nabla N^i\,d\Omega\approx-\int_{\Omega}\nabla(N^jr)\nabla N^i\,d\Omega.$$

Для полученного интеграла можно записать приближенное выражение

$$\int_{\Omega}\nabla(N^jr)\nabla N^i\,d\Omega=\left(\frac{\partial N^i}{\partial r}\frac{\partial N^j}{\partial r}+\frac{\partial N^i}{\partial z}\frac{\partial N^j}{\partial z}\right)V+\frac{\partial N^i}{\partial r}\int_{\Omega}\frac{N^j}{r}r\,d\Omega\approx$$

$$\approx\left(\frac{\partial N^i}{\partial r}\frac{\partial N^j}{\partial r}+\frac{\partial N^i}{\partial z}\frac{\partial N^j}{\partial z}\right)V+\frac{1}{3}S\frac{\partial N^i}{\partial r}.$$

Первое слагаемое в (20) – такое же, как и в случае плоской задачи с точностью до замены обозначений координат; второе – «добавочное слагаемое», которое необходимо ввести в матрицу вязкости $[L]$ при решении задачи в осесимметричной постановке.

К интегрированию компонент **матрицы масс** вернемся ниже.

Локальную систему линейных алгебраических уравнений (16) можно записать в сокращенной форме

$$[A]\{x\}=\{f\},$$

где $[A]$ – матрица левой части; $\{x\}$ – вектор-столбец неизвестных; $\{f\}$ – вектор-столбец правой части. При переходе к осесимметричной постановке матрицу левой части можно представить в виде суммы двух матриц, одна из которых равна матрице, получающейся при решении двумерной задачи $[A]_{2D}$, а вторая учитывает описанные выше «добавочные слагаемые», возникающие из-за осесимметричности $[A]_r$:

$$[A] = [A]_{2D} + [A]_r.$$

4.3. Обеспечение устойчивости схемы

Несмотря на простоту изложенной процедуры приближенного интегрирования, решение системы уравнений, полученной ансамблированием матриц, записанных вышеописанным образом, не всегда будет устойчивым. Для этого часть слагаемых записывают в модифицированном виде.

Рассмотрим следующее слагаемое из уравнения (14):

$$\int_\Omega N^j \sum_i N^i u_i \, d\Omega.$$

При составлении локальной системы данное слагаемое распадется на три компоненты. Однако при непосредственном его интегрировании в составленной матрице не будет диагонального преобладания, что приведет к ее плохой обусловленности. Однако (22) можно преобразовать таким образом, чтобы не нарушить условие диагонального преобладания:

$$\int_\Omega N^j \left(N^1 u_1 + N^2 u_2 + N^3 u_3\right) d\Omega \approx \int_\Omega N^j u_j \left(N^1 + N^2 + N^3\right) d\Omega = u_j \int_\Omega N^j \, d\Omega \approx \frac{1}{3} u_j V.$$

При этом было принято допущение о малом различии величин скоростей в узлах конечного элемента.

Запись компонент матрицы вязкости в виде (20) также приводит к плохой обусловленности матрицы $[A]$ Авторы работы [6] предлагают использовать следующий способ вычисления компонент матрицы вязкости:

$$L_{i,j} = \mu \int_\Omega [B_i]^T [C] [B_j] d\Omega,$$

где матрицы $[B_i]$ и $[C]$ имеют следующий вид:

$$[B_i] = \begin{bmatrix} N^i_{,r} & 0 \\ 0 & N^i_{,z} \\ N^i_{,z} & N^i_{,r} \\ N^i/r & 0 \end{bmatrix}, \qquad [C] = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix}.$$

Здесь запись $N^i_{,r}$ или $N^i_{,z}$ означает дифференцирование i-й функции формы по координате r или z соответственно. При такой записи компонент матрицы вязкости в $[L]$ появляются компоненты вида

$$\int_\Omega \frac{N^i}{r}\frac{N^j}{r} r\,d\Omega = \int_\Omega \frac{N^i N^j}{r} d\Omega.$$

При точном интегрировании данного выражения возникают слагаемые вида $\ln\frac{r_i}{r_j}$, где r_i и r_j – радиальные координаты узлов i и j конечного элемента. При удалении элемента на достаточно большое расстояние от оси величина логарифма будет стремиться к нулю и данные слагаемые не будут приводить к появлению каких-либо сложностей. Однако при приближении элемента к оси величина логарифма неограниченно растет, что вносит ошибку в расчеты.

При реализации осесимметричной модели PFEM-2 при численном интегрировании выражений использовались формулы приближенного интегрирования по одной, трем и шести гауссовым точкам [5].

Стоит заметить, что при добавлении в локальную матрицу аппроксимации уравнения неразрывности в форме (18) с учетом (19) соответствующие уравнению строки получаются приблизительно одинаковыми, что также приводит к плохой обусловленности системы. Чтобы в какой-то степени решить эту проблему, в уравнение неразрывности можно добавить дополнительное слагаемое, «уравновесив» его слагаемым в правой части. К примеру, таковым может быть лапласиан давления. Запишем его в конечно-элементной формулировке, домножим на функцию формы N^j и проинтегрируем по объему конечного элемента:

$$\int_\Omega N^j \Delta p^{n+1} r\,d\Omega = p_i^{n+1}\int_\Omega N^j \Delta N^i r\,d\Omega = p_i^{n+1}\left(\oint_S N^j r \nabla N^i\cdot\vec{n}\,dS - \int_\Omega \nabla(N^j r)\cdot\nabla N^i\,d\Omega\right)\approx$$

$$\approx -p_i^{n+1}\int_\Omega \nabla(N^j r)\cdot\nabla N^i\,d\Omega = -p_i^{n+1}\left(\int_\Omega r\nabla N^j\cdot\nabla N^i\,d\Omega + \int_\Omega N^j\nabla r\cdot\nabla N^i\,d\Omega\right)\approx$$

$$\approx -p_i^{n+1}\left(\nabla N^j\cdot\nabla N^i V + \frac{1}{3}\frac{\partial N_i}{\partial r}S\right). \qquad (23)$$

Полученное выражение умножается на стабилизирующий параметр τ и добавляется в строки локальной матрицы $[A]$, соответствующие уравнению неразрывности. В соответствующие компоненты вектора правой части войдет похожее выражение. Полагая значение ∇p^n известным и в простейшем случае равным $\vec{g}$, получаем:

$$\int_{\Omega} \nabla \cdot \vec{g} r \, d\Omega \approx -\nabla N^j \cdot \vec{g} V.$$

С учетом (23)–(24) локальную систему уравнений можно записать в виде

$$\left[\begin{array}{c|c} \frac{\rho}{\Delta t}[M]+\mu[L] & [G] \\ \hline [D] & \tau[L] \end{array}\right]\begin{pmatrix} \{V\}^{n+1} \\ \{P\}^{n+1} \end{pmatrix} = \begin{pmatrix} \rho g\{R\}+\frac{\rho}{\Delta t}\{V\}^n \\ -\tau[D]g \end{pmatrix},$$

В рассматриваемой реализации PFEM-2 стабилизирующий параметр τ был выбран равным [20]

$$\tau = \left(\frac{1}{\Delta t}+\frac{4\mu}{h^2}+\frac{2v^n}{h}\right)^{-1},$$

где h – длины наименьшей грани конечного элемента; v^n – модуль средней на элементе скорости в момент времени t_n. Зависимость параметра от Δt необходима для случаев невязких течений в ячейках с нулевой скоростью.

4.4 Обеспечение баланса массы в осесимметричном случае

Так как схема метода конечных элементов не является консервативной, необходимо использовать инструмент, позволяющий корректировать объем жидкости с целью обеспечения его постоянства. Это актуально при моделировании двухфазных течений и течений со свободной границей. В программном комплексе Kratos для того, чтобы обеспечить сохранение объема жидкости, достаточно заменить в соответствующей процедуре объемы плоских конечных элементов на объемы осесимметричных.

5. Решение модельных задач

5.1 Течение жидкости в цилиндрической трубе

Рассмотрим установившееся течение вязкой несжимаемой жидкости в цилиндрической трубе радиусом R и длиной L (рис. 3). Течение происходит под действием постоянной разности давлений p_1 и p_2, заданных на концах трубы. Коэффициент динамической вязкости равен μ. На стенках трубы заданы граничные условия прилипания $\vec{v}|_{r=R} = \vec{0}$. На оси – равенство нулю радиальной скорости $v_r|_{r=0} = 0$.

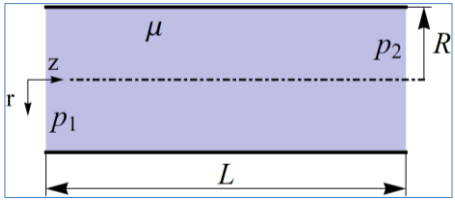


Рис. 3. Моделирование течения Пуазейля в круглой трубе

Fig. 3. Simulation of the Poiseuille flow in a pipe

Такое течение (при малых скоростях, когда обеспечиваеся ламинарный режим течения) называется течением Пуазейля и имеет аналитическое описание (профиль Пуазейля). Распределение скорости в трубе в зависимости от расстояния до оси симметрии r дается выражением

$$v(r) = \frac{p_2 - p_1}{4\mu L}(R^2 - r^2).$$

Для решения задачи выберем следующие параметры $L = 1{,}0$ м, $R = 0{,}15$ м, $\rho = 1000{,}0$ кг/м^3, $\Delta p = p_2 - p_1 = 0{,}013$ Па, $\mu = 0{,}001$ Па·с. Задача решалась при шаге пространственной дискретизации $\Delta h = 0.01$ м.

На рис. 4 приведены полученные результаты. Для построения графиков использовались данные о скоростях, взятые в центральном сечении трубы (при $z = L/2$). Из графиков видно удовлетворительное совпадение результатов расчета с аналитическим решением.

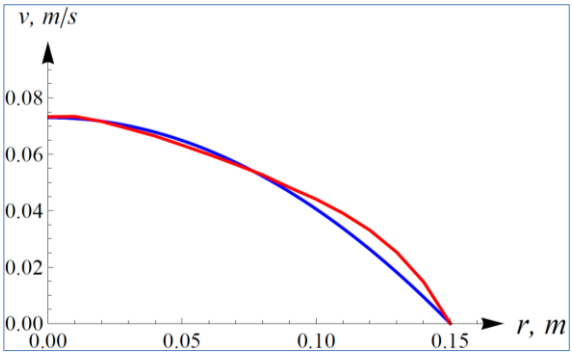


Рис. 4. График зависимости скорости от радиальной координаты (синяя кривая – профиль Пуазейля, красная кривая – результат расчета)
Fig. 4. Flow velocity dependence on the radial coordinate (blue curve – Poiseuille velocity profile; red curve – result of the flow simulation)

5.2 Падение капли в слой жидкости

Рассмотрим каплю жидкости с параметрами $\rho_f = 1179.0$ кг/м^3 и $\mu_f = 1.86 \cdot 10^{-5}$ Па·с, падающую в бассейн такой же жидкости под прямым углом (рис. 5). Скорость капли в момент соударения $v_{imp} = 2{,}56$ м/с. Параметры окружающего жидкость газа $\rho_a = 1.204$ кг/м^3 и $\mu_a = 1.90 \cdot 10^{-5}$ Па·с. На поведение жидкости в общем случае сильно влияет значение поверхностного натяжения [7], однако, так как в программном комплексе Kratos отсутствуют модели поверхностного натяжения для PFEM-2, примем коэффициент поверхностного натяжения равным нулю. На боковой и нижней стенках области заданы граничные условия прилипания, но оси – ограниченность радиальной скорости v^r, на верхней границе равенство нулю давления p.

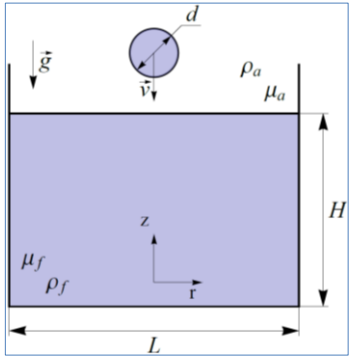


Рис. 5. Моделирование падения капли в глубокий слой жидкости

Fig. 5. Simulation of the droplet impact onto a deep pool

Задача о падении капли в слой жидкости аналитического решения не имеет, и из-за допущения о равенстве нулю коэффициента поверхностного натяжения становится невозможным сравнение полученных результатов с данными экспериментов. Поэтому решение задачи о падении капли с помощью PFEM-2 будем сравнивать с результатами численного эксперимента, поставленного в программном пакете с открытым исходным кодом Gerris [8, 9], который позволяет решать задачи гидродинамики со свободной поверхностью интегро-интерполяционным методом. Его отличием от других программ этого класса является использование структурированных динамически перестраиваемых сеток. Gerris успешно зарекомендовал себя в решении задач о падении капель как в слой жидкости, так и на твердую поверхность [10, 11].

Решения, полученные в задаче о падении капли в моменты времени $t = 0{,}0;\ 0{,}00117;\ 0{,}00234;\ 0{,}00351;\ 0{,}00468;\ 0{,}00585$ с, приведены на рис.6.

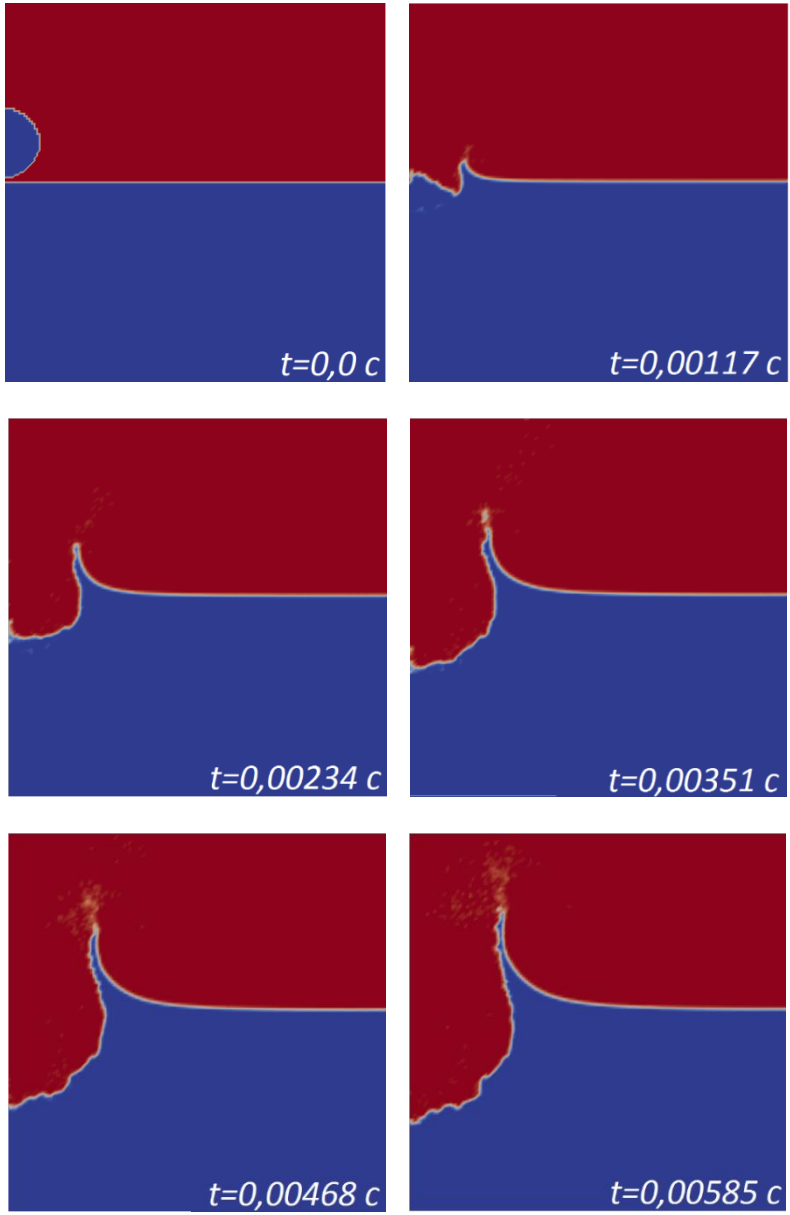


Рис. 6. Моделирование падения капли в глубокий слой жидкости
Fig. 6. Simulation of the droplet impact onto a deep pool

Для того, чтобы количественно сравнить полученные результаты, построим графики зависимости глубины кратера от времени (рис. 7). Сравнение графиков для глубины кратера показывает также удовлетворительное соответствие между результатами численных экспериментов, проведенных в двух программных пакетах, Kratos и Gerris, в основе которых лежат разные численные методы решения задач гидродинамики.

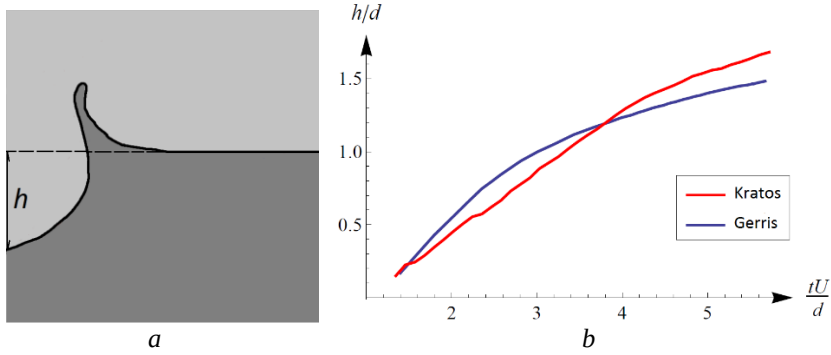


Рис. 7. Схема для нахождения глубины кратера (a) и график зависимости приведенной глубины кратера от безразмерного времени (b)
Fig. 7. The scheme for the crater depth determination (a) and the normalized crater depth dependency on dimensionless time (b)

Благодарности

Работа выполнена при частичной финансовой поддержке Российского фонда фундаментальных исследований (проект 17-08-01468)

Список литературы

[1]. Idelsohn S.R., Onate E., Pin F.D. The particle finite element method: a powerful tool to solve incompressible flows with free-surfaces and breaking waves. International journal for numerical methods in engineeringб vol. 61, No. 7, 2004, pp. 964–989. doi: 10.1002/nme.1096
[2]. Idelsohn S.R., Nigro N.M., Gimenez J.M., Rossi R., Marti J.M. A fast and accurate method to solve the incompressible Navier-Stokes equations. Engineering Computations, vol.30, No. 2, 2013, pp.197-222. doi: 10.1108/02644401311304854
[3]. Милн-Томсон Л.М. Теоретическая гидродинамика. М.: Мир, 1964. 660 стр.
[4]. Becker P. An enhanced Particle Finite Element Method with special emphasis on landslides and debris flows. PhD thesis, 2015, Universitat Politecnica de Catalunya.
[5]. Зенкевич О.С. Метод конечных элементов в технике. М.: Мир, 1975. 543 стр.
[6]. Hughes T.J.R., Liu W.K., Brooks A. Finite Element Analysis of Incompressible Viscous Flows by the Penalty Function Formulation. Journal of Computational Physics, vol. 30, No. 1, 1979, pp. 1-60. doi: 10.1016/0021-9991(79)90086-X

[7]. Yarin A.L. Drop Impact Dynamics: Splashing, Spreading, Receding, Bouncing. Annual Review of Fluid Mechanics, vol. 38, 2006, pp. 159-192. doi: 10.1146/annurev.fluid.38.050304.092144

[8]. Popinet S. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. Journal of Computational Physics, vol. 190, No. 2, . 2003, pp. 572–600. doi: 10.1016/S0021-9991(03)00298-5

[9]. Korchagova V.N., Kraposhin M.V., Marchevsky I.K., Smirnova E.V. Simulation of droplet impact onto a deep pool for large Froude numbers in different open-source codes. Journal of Physics: Conference Series, vol. 918. art. 012037, 2017. doi: 10.1088/1742-6596/918/1/012037

[10]. Agbaglaha G., Thoravala M.-J., Thoroddsen S.T., Zhanga L.V., Fezzaaa K., Deegana R.D. Drop impact into a deep pool: vortex shedding and jet formation. Journal of Fluid Mechanics, vol. 764, 2015m pp. 1-12. doi: 10.1017/jfm.2014.723

[11]. Renardy Y., Popinet S., Duchemin L., Renardy M., Zaleski S., Josserand C., Drumright-Clarke M.A., Richard D., Clanet C., Quere D. Pyramidal and toroidal water drops after impact on solid surface. Journal of Fluid Mechanics, vol. 484, . 2003, pp. 69–83. doi: 10.1017/S0022112003004142

Axisymmetric viscous incompressible flow simulation by using the Particle finite element PFEM-2 method in the open source Kratos code

[1] *E.V. Smirnova <alena.davidova@ispras.ru>*
[1,2] *I.K. Marchevsky <iliamarchevsky@mail.ru>*
[2] *V.O. Bondarchuk <viktoriia.bondarchuk@gmail.com>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Bauman Moscow State Technical University, 5, 2-nd Baumanskaya st., Moscow, 105005, Russia*

Abstract. In this paper, the particle finite element method (PFEM-2) for the simulation of the axisymmetric flows of a viscous incompressible fluid is considered. The necessary equations for the description of the axisymmetric flows by using the particle finite element method with some assumptions were obtained. The numerical model for solving axisymmetric flows of a viscous incompressible fluid was implemented in the Kratos open-source code on the basis of the existing application for solving two-dimensional problems. The numerical model for the simulation of the axisymmetric flows of a viscous incompressible fluid was validated on two problems. First of them is the Poiseuille problem about viscous flow in the pipe. The numerical solution obtained by particle finite element method are in the satisfactory agreement with the analytical solution for this problem. The second test is a problem of a droplet impact onto a deep liquid pool with similar fluid. As the comparison with analytical results is impossible, the results of particle finite element method simulation were compared with results of the numerical simulation obtained by using the open-source package Gerris (Volume of Fluid method). The results of comparing of numerical simulations of droplet impact onto a liquid pool obtained by two different codes also in satisfactory agreement.

Keywords: particle finite element method; viscous incompressible flow; axisymmetric flow;; Poiseuille flow; droplet impact onto a deep pool.

DOI: 10.15514/ISPRAS-2018-30(2)-13

For citation: Smirnova E.V., Marchevsky I.K., Bondarchuk V.O. Axisymmetric viscous incompressible flow simulation by using the Particle finite element PFEM-2 method in the open source Kratos code. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 2, 2018, pp. 263-284 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-13

References

[1]. Idelsohn S.R., Onate E., Pin F.D. The particle finite element method: a powerful tool to solve incompressible flows with free-surfaces and breaking waves. International journal for numerical methods in engineering6 vol. 61, No. 7, 2004, pp. 964–989. doi: 10.1002/nme.1096

[2]. Idelsohn S.R., Nigro N.M., Gimenez J.M., Rossi R., Marti J.M. A fast and accurate method to solve the incompressible Navier-Stokes equations. Engineering Computations, vol.30, No. 2, 2013, pp.197-222. doi: 10.1108/02644401311304854

[3]. Milne-Thomson L.M. Theoretical Hydrodynamics. London: Macmillan & Co, 1968.

[4]. Becker P. An enhanced Particle Finite Element Method with special emphasis on landslides and debris flows. PhD thesis, 2015, Universitat Politecnica de Catalunya.

[5]. Zienkiewicz O.C. The finite element method in engineering science. London: McGraw-Hill, 1971. 521 p.

[6]. Hughes T.J.R., Liu W.K., Brooks A. Finite Element Analysis of Incompressible Viscous Flows by the Penalty Function Formulation. Journal of Computational Physics, vol. 30, No. 1, 1979, pp. 1-60. doi: 10.1016/0021-9991(79)90086-X

[7]. Yarin A.L. Drop Impact Dynamics: Splashing, Spreading, Receding, Bouncing. Annual Review of Fluid Mechanics, vol. 38, 2006, pp. 159-192. doi: 10.1146/annurev.fluid.38.050304.092144

[8]. Popinet S. Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries. Journal of Computational Physics, vol. 190, No. 2, . 2003, pp. 572–600. doi: 10.1016/S0021-9991(03)00298-5

[9]. Korchagova V.N., Kraposhin M.V., Marchevsky I.K., Smirnova E.V. Simulation of droplet impact onto a deep pool for large Froude numbers in different open-source codes. Journal of Physics: Conference Series, vol. 918. art. 012037, 2017. doi: 10.1088/1742-6596/918/1/012037

[10]. Agbaglaha G., Thoravala M.-J., Thoroddsen S.T., Zhanga L.V., Fezzaaa K., Deegana R.D. Drop impact into a deep pool: vortex shedding and jet formation. Journal of Fluid Mechanics, vol. 764, 2015m pp. 1-12. doi: 10.1017/jfm.2014.723

[11]. Renardy Y., Popinet S., Duchemin L., Renardy M., Zaleski S., Josserand C., Drumright-Clarke M.A., Richard D., Clanet C., Quere D. Pyramidal and toroidal water drops after impact on solid surface. Journal of Fluid Mechanics, vol. 484, . 2003, pp. 69–83. doi: 10.1017/S0022112003004142

Математическое моделирование двумерных течений газа с использованием RKDG-метода на структурированных прямоугольных сетках

[1] *В.Н. Корчагова <v.korchagova@ispras.ru>*
[1] *И.Н. Фуфаев <fufaeviwan@yandex.ru>*
[1] *С.М. Сауткина <sofyasautkina@yandex.ru>*
[1,2] *В.В. Лукин <vvlukin@gmail.com>*
[1] *Московский государственный технический университет им. Н.Э. Баумана, 105005, Россия, Москва, ул. 2-я Бауманская, д. 5.*
[2] *Институт прикладной математики им. М.В. Келдыша РАН, 125047, Россия, Москва, Миусская пл., д. 4*

Аннотация. Работа посвящена поиску приближенного решения системы уравнений газовой динамики методом RKDG (Runge – Kutta Discontinuous Galerkin), который характеризуется высоким порядком точности по сравнению с классическим методом конечных объёмов (МКО). Вычислительный алгоритм реализован на языке C++ и верифицирован на тестовых задачах. Результаты моделирования акустического импульса на достаточно грубой сетке с кусочно-линейной аппроксимацией хорошо согласуются с аналитическим решением, в отличие от численного приближения с помощью МКО. Для задачи Сода приводится сравнение зависимости схемы от выбора численных потоков, индикатора проблемных ячеек и лимитера.

Ключевые слова: RKDG-метод; уравнения Эйлера; MUSCL-схема; WENO-схема; лимитер; индикатор.

DOI: 10.15514/ISPRAS-2018-30(2)-14

Для цитирования: Корчагова В.Н., Фуфаев И.Н., Сауткина С.М., Лукин В.В. Математическое моделирование двумерных течений газа с использование RKDG-метода на структурированных прямоугольных сетках. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 285-300. DOI: 10.15514/ISPRAS-2018-30(2)-14

1. Введение

Разрывный метод Галеркина (RKDG-метод) относится к числу наиболее эффективных численных методов, позволяющих исследовать математические модели, допускающие разрывные решения, а также развитие неустойчивостей.

Суть метода сводится к повышению точности моделирования не за счет расширения шаблона аппроксимации, а за счет увеличения порядка аппроксимации численного решения на каждой ячейке. Благодаря этому RKDG-метод получил широкое распространение в вычислительной аэроакустике [1, 2] и гидродинамике [3].

Согласно теореме Годунова алгоритм поиска численного решения RKDG-методом, обеспечивающий повышенный порядок точности, требует дополнительной монотонизации решения для подавления нефизичных осцилляций. Для этих целей обычно используют функции-ограничители, иначе называемые лимитерами [4]. Вследствие применения лимитеров порядок точности численного метода в общем случае может снижаться до первого, поэтому их следует применять лишь на тех ячейках, где нарушается монотонность решения. Для поиска таких «проблемных» ячеек используют функции-индикаторы.

Одним из способов обеспечения монотонности решения является использование MUSCL-схем (Monotonic Upstream-Centered Scheme for Conservation Laws) [5, 6]. Широкое использование подобных лимитеров связано прежде всего с простотой их реализации, однако качество разрешения разрывов при этом остается невысоким.

Методические исследования, проведенные для одномерных задач [7], показывают высокую эффективность другого подхода, связанного с использованием технологии реконструкции решения на основе подхода WENO (Weighted Essentially Non-Oscillatory). Исходные идеи, положенные в основу схем типа WENO, ограничивают применимость данных методов в многомерном случае, поскольку они требуют расширенного шаблона аппроксимации для реконструкции решения с высокой точностью [8]. Однако разработаны модификации этого алгоритма (например, WENO_S [9] и HWENO_SC [10]), которые позволяют сохранить компактность шаблона аппроксимации.

Целью работы является сравнительный анализ применения наиболее простого лимитера типа WENO (WENO_S) и MUSCL-лимитера при решении двумерных задач газовой динамики на структурированных прямоугольных сетках. Для этого разработан программный комплекс, реализующий RKDG метод с линейными базисными функциями применительно к решению системы уравнений газовой динамики в двумерной постановке.

2. Система уравнений газовой динамики

Нестационарная система уравнений газовой динамики, описывающая двумерные течения идеального сжимаемого нетеплопроводного газа, имеет вид [11]:

$$\frac{\partial \mathbf{U}}{\partial t}+\frac{\partial \mathbf{F(U)}}{\partial x}+\frac{\partial \mathbf{G(U)}}{\partial y}=\mathbf{0}, \tag{1}$$

где

$$\begin{aligned}\mathbf{U} &= [\rho,\rho u,\rho v,\rho w,e]^T,\\ \mathbf{F} &= [\rho u,\rho u^2+p,\rho uv,\rho uw,(e+p)u]^T,\\ \mathbf{G} &= [\rho v,\rho vu,\rho v^2+p,\rho vw,(e+p)v]^T.\end{aligned} \tag{2}$$

Здесь $\mathbf{U}$ – вектор консервативных переменных; $\mathbf{F}$, $\mathbf{G}$ – векторы потоков консервативных переменных; ρ – плотность; $\mathbf{v}=[u,v,w]^{\mathrm{T}}$ – вектор скорости; $e=\rho\varepsilon+\frac{1}{2}\rho(u^2+v^2+w^2)$ – полная энергия единицы объема; p – давление газа; ε – удельная внутренняя энергия.

Связь p, ρ и ε определяется уравнением состояния газа:

$$p=(\gamma-1)\rho\varepsilon, \tag{3}$$

где $\gamma>1$ – показатель адиабаты.

Систему (1) можно записать в квазилинейной неконсервативной форме:

$$\frac{\partial \mathbf{U}}{\partial t}+A\frac{\partial \mathbf{U}}{\partial x}+B\frac{\partial \mathbf{U}}{\partial y}=\mathbf{0}.$$

Эта система уравнений является гиперболической, все собственные значения матриц A и B действительны и существует полная система собственных векторов. Следовательно, матрицы могут быть диагонализированы и записаны в виде

$$\begin{aligned}A&=\Omega_R^A\Lambda^A\Omega_L^A, \qquad \Omega_R^A=\left(\Omega_L^A\right)^{-1},\\ B&=\Omega_R^B\Lambda^B\Omega_L^B, \qquad \Omega_R^B=\left(\Omega_L^B\right)^{-1},\end{aligned}$$

где $\Lambda^j=diag[\lambda_1^j,\ldots,\lambda_5^j]$, $j=A,B$ – диагональные матрицы, составленные из собственных значений матриц A и B; Ω_R^j,Ω_L^j – матрицы правых и левых собственных векторов соответственно.

Будем рассматривать систему уравнений (1) в области $[0;L_x]\times[0;L_y]\times(0;T]$ и зададим начальное условие вида

$$\mathbf{U}(x,y,0)=\mathbf{U}_0(x,y). \tag{4}$$

3. Численный метод

3.1 Общая схема RKDG

Введем на рассматриваемой пространственной области равномерную по каждому направлению прямоугольную сетку с шагами $h_x = L_x/N_x$ и $h_y = L_y/N_y$ соответственно, состоящую из $N_x \cdot N_y$ ячеек I_{ij} с центрами $(x_i, y_j) = ((i-1/2)h_x, (j-1/2)h_y)$ и границами

$$\left(x_{i\pm1/2}, y_{j\pm1/2}\right) = \left(x_i \pm h/2, y_j \pm h/2\right), \quad i = \overline{1, N_x}, \quad j = \overline{1, N_y}.$$

Для использования RKDG-метода определим пространство кусочно-непрерывных функций $V_h^k = \{p : p|_{ij} \in \mathbf{P}_k(I_{ij})\}$, которые являются многочленами степени не выше k на каждой ячейке I_{ij}. Для поиска численного решения системы (1), (4) умножим уравнения системы на тестовые функции $v(x,y) \in V_h^k$ и проинтегрируем полученные соотношения по каждой ячейке.

Приближенное решение задачи будем искать в виде

$$\mathbf{U}_h(x,y,t) = \sum_{i,j} \sum_{s=0}^{2} \mathbf{U}_{ij}^{(s)}(t) \phi_{ij}^{(s)}(x,y),$$

где $\{\phi_{ij}^{(s)}(x,y)\}_{s=0}^{2}$ – ортонормированные базисные функции, определенные на ячейке I_{ij} следующим образом:

$$\phi_{ij}^{(0)}(x,y) = \frac{1}{\sqrt{h_x h_y}}, \quad \phi_{ij}^{(1)}(x,y) = \sqrt{\frac{12}{h_x^3 h_y}}(x - x_i), \quad \phi_{ij}^{(2)}(x,y) = \sqrt{\frac{12}{h_x h_y^3}}(y - y_j).$$

Индекс «*h*» у численного решения задачи для краткости будем опускать. Используя в качестве пробных функций базисные, получим систему обыкновенных дифференциальных уравнений относительно коэффициентов $\mathbf{U}_{ij}^{(r)}$, $r = 0,1,2$:

$$\frac{d\mathbf{U}_{ij}^{(r)}(t)}{dt} - \int_{I_{ij}} \mathbf{F}_{ij} \frac{\partial \phi_{ij}^{(r)}}{\partial x} dxdy - \int_{I_{ij}} \mathbf{G}_{ij} \frac{\partial \phi_{ij}^{(r)}}{\partial y} dxdy +$$

$$+ \int_{\partial I_{ij,R}} \mathbf{F}_{ij} \phi_{ij}^{(r)} dx - \int_{\partial I_{ij,L}} \mathbf{F}_{ij} \phi_{ij}^{(r)} dx + \int_{\partial I_{ij,T}} \mathbf{G}_{ij} \phi_{ij}^{(r)} dy - \int_{\partial I_{ij,B}} \mathbf{G}_{ij} \phi_{ij}^{(r)} dy = 0, \qquad (5)$$

$$\mathbf{U}_{ij}^{(r)}(0) = \int_{I_{ij}} \mathbf{U}_0(x,y)\phi_{ij}^{(r)}dxdy, \quad k = 1,\ldots,N_x \cdot N_y, \quad r = 0,1,2.$$

Здесь $\mathbf{F}_{ij} = \mathbf{F}(\mathbf{U}_{ij}(x,y,t)), \mathbf{G}_{ij} = \mathbf{G}(\mathbf{U}_{ij}(x,y,t))$, $\partial I_{ij,L}$, $\partial I_{ij,R}$, $\partial I_{ij,T}$ и $\partial I_{ij,B}$ обозначают левую, правую, верхнюю и нижнюю границы ячейки I_{ij} соответственно.

При решении системы ОДУ (5) требуется обеспечивать порядок точности, соответствующий порядку аппроксимации по пространственным координатам. Для этого целесообразно использовать методы Рунге – Кутты, причем среди методов высокого порядка необходимо выбирать те, которые обеспечивают наименьший уровень временной немонотонности численного решения. В данной работе используется TVD-метод Рунге – Кутты второго порядка:

$$\mathbf{U}^* = \mathbf{U}^n + \tau\mathbf{L}_h(\mathbf{U}^n),$$

$$\mathbf{U}^{n+1} = \frac{1}{2}\mathbf{U}^n + \frac{1}{2}\mathbf{U}^* + \frac{1}{2}\tau\mathbf{L}_h(\mathbf{U}^*),$$

где τ – шаг по времени; $\mathbf{L}_h(\mathbf{U})$ – правая часть уравнения (5); $\mathbf{U}^n$ и $\mathbf{U}^{n+1}$ – приближенные решения на текущем и следующем временных слоях соответственно.

3.2 Численные потоки

Численное решение уравнения (1) терпит разрывы на границах ячеек, поэтому аналитические потоки $\mathbf{F}$ и $\mathbf{G}$ в уравнении (5) следует заменить численными потоками $\tilde{\mathbf{F}}$ и $\tilde{\mathbf{G}}$. Их можно вычислить по предельным значениям численного решения на ячейках, соседних к рассматриваемой границе, используя приближенные решения задачи Римана о распаде произвольного разрыва, по аналогии со схемами годуновского типа. В данной работе используются численные потоки Лакса – Фридрихса (LF), HLL и HLLC [11].

3.3 Лимитеры

С одной стороны, использование для аппроксимации решения на ячейках базисных функций высокого порядка должно приводить к повышению точности метода. Однако с другой стороны, получаемая численная схема становится немонотонной, что может выражаться в возникновении нефизичных осцилляций приближенного решения, особенно в окрестности разрывов. Для подавления таких осцилляций применяются специальные нелинейные ограничители – лимитеры [4, 8], обеспечивающие уменьшение

наклона численного решения в проблемной с точки зрения немонотонности ячейке.

В данной работе использованы лимитеры WENO_S (simple) [9] и MUSCL [13]. Лимитеры применены к численному решению покомпонентно; лимитер WENO_S может применяться двумя способами:

- путем реконструкции решения на шаблоне, состоящем из самой ячейки и четырех соседних ячеек;
- в виде полусуммы независимых реконструкций вдоль каждой из осей.

Большинство известных ограничителей, таких как, например, MUSCL (основанный на minmod-свойстве), ухудшают точность, так как ошибочно применяются в областях, где решение является гладким. Это нивелирует эффект от введения полиномиальной аппроксимации решения на соседних ячейках. Таким образом, лимитеры следует применять лишь на тех ячейках, где монотонность решения нарушается. Поэтому первый шаг алгоритма монотонизации должен быть связан с обнаружением так называемых «проблемных ячеек», т.е. поиском ячеек, производные решения на которых необходимо ограничить. В данной работе для данных целей используется индикатор проблемных ячеек KXRCF [14].

4. Результаты тестовых расчетов

4.1 Распространение акустического импульса

Рассмотрим распространение цилиндрической акустической волны, вызванное малым возмущением плотности. Начальное возмущение задано в центре прямоугольной области $[0, L_x]\times[0, L_y]$ функцией

$$\rho'(x,y)\,|_{t=0} = \varepsilon \exp\left(-2(x-0.5L_x)^2 - 2(y-0.5L_y)^2\right), \quad \varepsilon = 10^{-6}.$$

Следует отметить, что такое возмущение является точным решением системы линеаризованных уравнений Эйлера (1), записанной для малых возмущений плотности ρ', компонент скорости u', v' и давления p' (акустическое приближение):

$$\begin{aligned}
\mathbf{U} &= [\rho', \rho_0 u', \rho_0 v', \rho_0 w', p']^T, \\
\mathbf{F} &= [\rho' u_0 + \rho_0 u', \rho_0 u_0 u' + p', \rho_0 u_0 v', \rho_0 u_0 w', u_0 p' + \gamma p_0 u']^T, \\
\mathbf{G} &= [\rho' v_0 + \rho_0 v', \rho_0 v_0 u', \rho_0 v_0 v' + p', \rho_0 v_0 w', v_0 p' + \gamma p_0 v']^T,
\end{aligned}$$

где ρ_0, u_0, v_0, p_0 – заданные для невозмущенной среды значения плотности, составляющих скорости и давление соответственно. Распределение давления выбрано так, чтобы скорость звука была постоянной и равной

$c = \sqrt{\gamma}$. Скорость течения принимается нулевой, $\mathbf{v_0} = [0,0,0]^{\mathrm{T}}$; плотность и давление в невозмущенной среде полагаются равными $\rho_0 = 1$, $p_0 = 1$. Расчетная область имеет размеры $L_x = L_y = 8$. Границы области считаются открытыми, что означает, что возмущения могут свободно выходить из расчетной области.

На рис. 1 приведены результаты расчетов на сетке 20×20 ячеек при значении числа Куранта 0.2. Показан профиль решения вдоль средней линии области течения $y = L_y/2$ в момент времени $t = 2$.

Высокий уровень диссипации, присущий численному потоку LF приводит к значительному «размыванию» решения, получаемого при помощи метода конечных объемов (FVM), см. рис. 1, *a*. В то же время, кусочно-линейная аппроксимация приводит к получению решения, хорошо согласующегося с аналитическим. Менее диссипативный поток HLLC (рис. 1, *b*) обеспечивает несколько более высокое качество решения методом контрольного объема, однако при использовании кусочно-линейных базисных функций эффект от применения потока HLLC практически не заметен. Вычислительные эксперименты показывают, что численная схема остается устойчивой при величине числа Куранта не более 0.2.

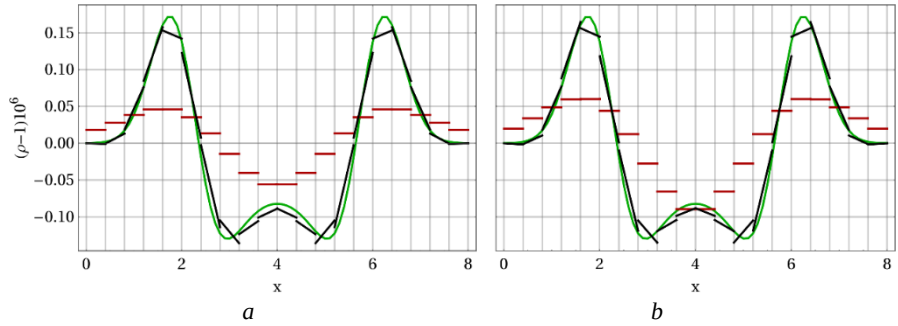


Рис. 1. Распространение акустической волны; сетка 20×20 ячеек, число Куранта $\mathrm{Co} = 0.2$, показан график плотности вдоль средней линии по направлению оси Ox в момент времени $t = 2.0$: a – для численного потока LF, b – для численного потока HLLC. Черные линии – численное решение с тремя базисными функциями (кусочно-линейная аппроксимация), красные линии – решение методом конечного объема (кусочно-постоянная аппроксимация), зеленая линия – точное аналитическое решение

Fig. 1. Propagation of acoustic pulse, mesh 20×20 cells, $\mathrm{Co} = 0.2$, density plot along the center line in x-direction, $t = 2.0$: (a) LF numerical flux, (b) HLLC numerical flux. Black line is the numerical solution with 3 basis functions (piecewise-linear solution representation), red line is FVM solution (pieciwise-constant solution representation), green line is the analytical solution

4.2 Задача Сода

Другая группа тестов основана на задаче Сода, решение которой содержит три основных типа разрывов: ударную волну, волну разрежения и контактный разрыв. Качество разрешения этих разрывов зависит от чувствительности используемого индикатора проблемных ячеек, а также от выбора алгоритма монотонизации и используемого численного потока.

Первый тестовый расчет соответствует одноосному распространению волн: предполагается, что решение распространяется вдоль оси *Ox*, тогда как в направлении оси *Oy* расчетная область состоит лишь из одной ячейки.

Начальные условия для данной задачи имеют вид

$$(\rho,u,v,w,p) = \begin{cases} (1,0,0,0,1), & x \le 0.5, \\ (0.125,\ 0,0,0,0.1), & x > 0.5. \end{cases}$$

Результаты расчетов получены для сетки, состоящей из 100 ячеек в напрвлении оси *Ox* при значении числа Куранта 0.1. Расчет выполнялся до момента времени $t = 0.2$. Параметры расчета соответствуют таковым применительно к решению одномерной задачи [7].

Первая группа результатов (рис. 2) представляет численные решения, полученные с использованием индикатора проблемных ячеек KXRCF при использовании различных численных потоков и лимитеров. На рис. 2, *a* представлено численное решение задачи Сода полученное с использованием MUSCL-лимитера и численного потока LF. Данные сравнительно простые алгоритмы позволяют достигать хорошего качества решения в смысле монотонности, однако обеспечивают низкое качество разрешения разрывов: около 6 ячеек на фронт ударной волны и 10-12 ячеек на контактном разрыве. Также наблюдается потеря точности решения внутри волны разрежения и образование выраженного «горба» перед ее фронтом.

Использование более точного численного потока HLLC позволяет повысить качество решения в целом (рис. 2, *b*). При этом несколько повышается точность моделирования волны разрежения и качество воспроизведения решения в областях его гладкости. Тем не менее, число ячеек, приходящихся на разрывы, остается сравнительно большим.

Изменения качества численного решения становятся более существенными при замене MUSCL-лимитера на лимитер WENO_S (рис. 3, 4). Этот лимитер обеспечивает размазывание ударной волны на две ячейки, контактного разрыва – на пять ячеек даже при применении такого существенно диссипативного потока, как поток LF, и в то же время сохраняет компактность шаблона аппроксимации, присущую разрывному методу Галеркина, и использует данные только с соседних (рис. 3, *a*). Согласно идее WENO-реконструкции, паразитные осцилляции полностью не исключаются, но должны убывать по амплитуде при измельчении сетки. Можно видеть эти осцилляции на гладких участках решения между разрывами. Использование

более точного численного потока, например, HLL или HLLC, позволяет дополнительно улучшить качество приближенного решения (рис. 3, *b*, 4, *a*).

Предыдущие результаты были получены при помощи первого варианта реализации методики WENO_S, использующей одновременно 4 соседних ячейки. Указанный выше подход с использованием полусуммы результатов квазиодномерного лимитирования численного решения вдоль каждой из осей оказывается менее эффективен в силу больших вычислительных затрат при меньшей разрешающей способности на разрывах по сравнению с первым вариантом.

Заметим, что монотонность решения существенно зависит от используемого индикатора проблемных ячеек. На рис. 4, *b* показаны результаты расчетов, полученные при применении лимитера на всех ячейках. В этом случае лимитер WENO_S дает более качественное решение, подавляя «горб» в голове волны разрежения и практически убирая осцилляции в области гладкости решения. В то же время, использование индикации проблемных ячеек существенно снижает вычислительную стоимость лимитирования. Индикатор KXRCF является популярным в силу относительно простоты адаптации к многомерным задачам.

Следующий тест – задача Сода с начальным положением фронта на диагонали $x + y = 1$ расчетной области – единичного квадрата. Начальные условия взяты следующим образом:

$$(\rho,u,v,w,p) = \begin{cases} (1,0,0,0,1), & x + y \le 1, \\ (0.125,\ 0,0,0,0.1), & x + y > 1. \end{cases}$$

Численное решение на рис. 5 получено в момент времени t = 0.2 на сетке 40×40 и с соотношением временного и пространственного шага 0.1. Для расчетов выбран численный поток HLLC. В качестве граничных условий взяты условия свободного вытекания. Такие граничные условия приводят к возникновению искусственных отраженных волн в окрестности углов расчетной области. В связи с этим сравнение численного и точного решений произведено вдоль линии $x = y$, где влияние таких волн пренебрежимо мало.

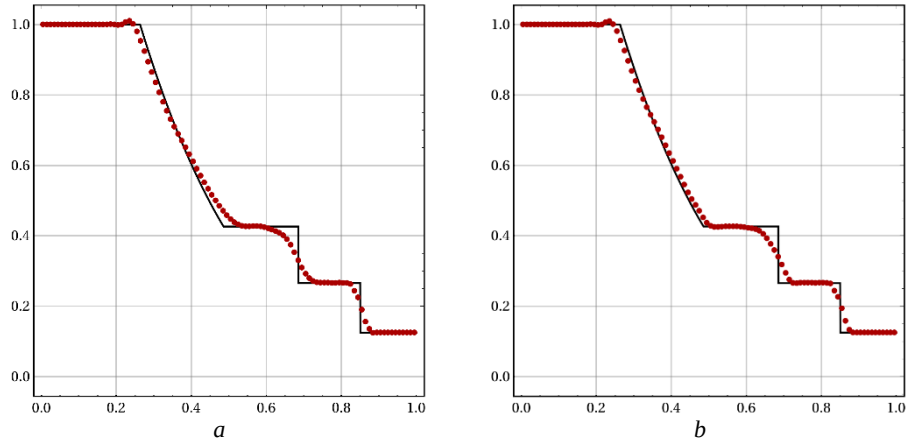


Рис. 2: Задача Сода, распределение плотности в момент времени t = 0.2: точное (черная кривая) и численное (красные точки) решения. Расчеты с лимитером MUSCL, индикатором KXRCF и численными потоками LF (a) и HLLC (b)
Fig. 2: The Sod problem, density distribution in the final moment of time: the analytical solution (black line) and numerical solution (red dots). The calculation with MUSCL approach, KXRCF indicator and numerical fluxes LF (a) and HLLC (b)

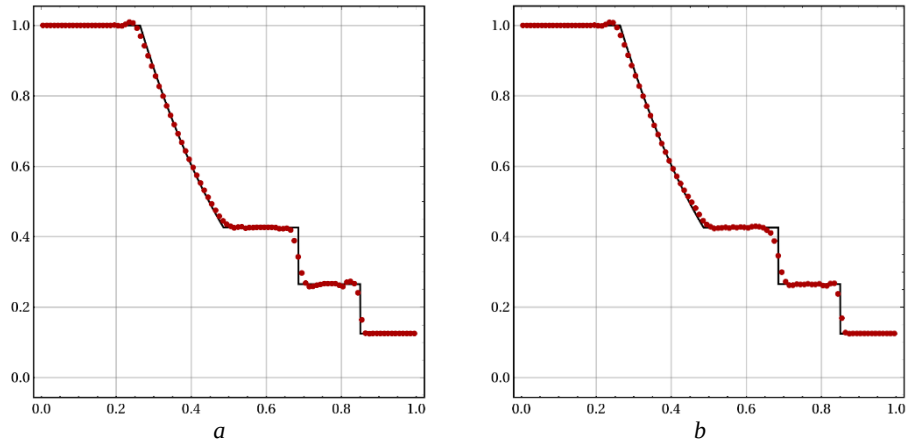


Рис. 3: Задача Сода, распределение плотности в момент времени t = 0.2: точное (черная кривая) и численное (красные точки) решения. Расчеты с индикатором KXRCF, лимитером WENO_S и численными потоками LF (a) и HLLC (b)
Fig. 3: The Sod problem, density distribution in the final moment of time: the analytical solution (black line) and numerical solution (red dots). The calculation with KXRCF indicator, WENO_S limiter and numerical fluxes LF (a) and HLL (b)

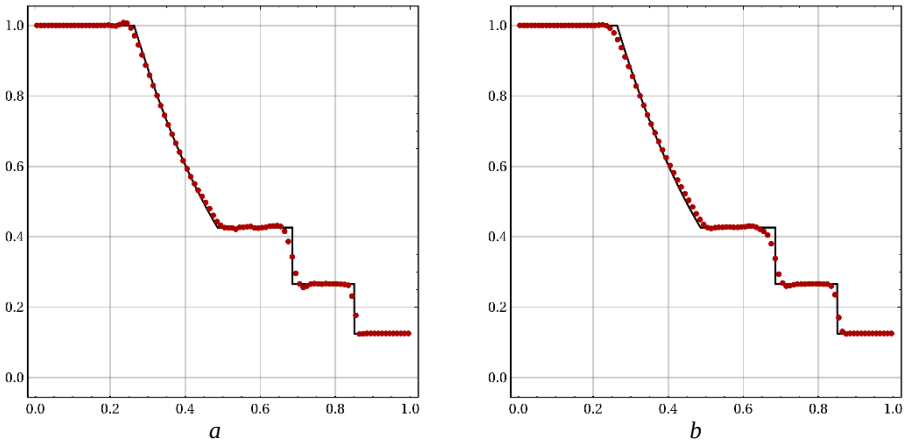


Рис. 4: Задача Сода, распределение плотности в момент времени t = 0.2*: точное (черная кривая) и численное (красные точки) решения. Расчеты с лимитером WENO_S численным потоком HLLC и индикатором KXRCF (a) и лимитированием по всем ячейкам (b)*

Fig. 4: The Sod problem. The density distribution in the final moment of time: the analytical solution (black line) and numerical solution (red dots). The calculation with WENO_S limiter and numerical fluxes HLLC. Two situations: KXRCF indicator (a) and with limitation all the cells (b)

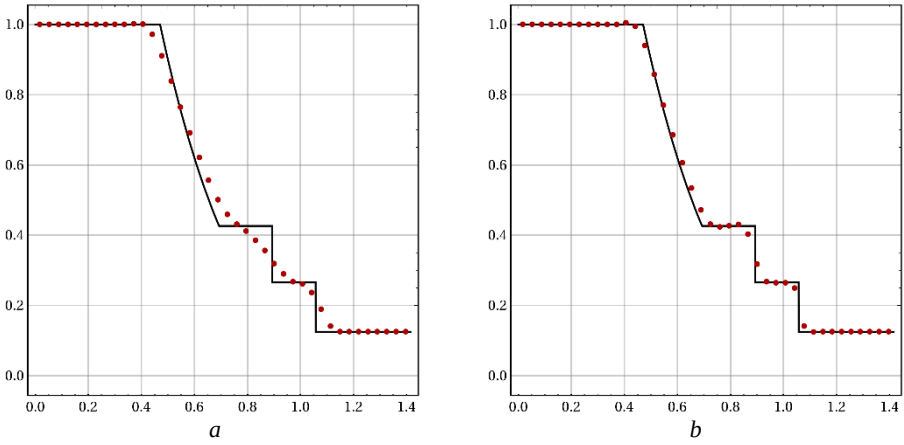


Рис. 5. Задача Сода, распределение плотности вдоль линии x = y в момент времени t = 0.2*: точное (черная кривая) и численное (красные точки) решения. Расчеты с численным потоком HLLC и лимитерами MUSCL (a) и WENO_S (b)*

Fig. 5. The Sod problem, density distribution along the line x = y in the final moment of time: the analytical solution (black line) and numerical solution (red dots). The calculation with numerical flux HLLC and limiters MUSCL (a) and WENO_S (b)

Численная схема с лимитером MUSCL имеет высокую численную вязкость (рис. 5, *a*). Напротив, лимитер WENO_S позволяет получить численное решение более приемлемого качества (рис 5, *b*). Разрешение разрывов соответствует полученным выше результатам для одноосного распространения волн.

5. Заключение

Рассмотрена реализация алгоритма RKDG метода для решения двумерных задач газовой динамики на структурированных прямоугольных сетках с использованием линейных базисных функций. В разработанном авторами статьи программном комплексе использованы численные потоки Лакса – Фридрихса, HLL и HLLC, индикатор проблемных ячеек KXRCF и лимитеры WENO_S и MUSCL. Численное решение тестовой задачи о распространении акустического возмущения хорошо согласуется с аналитическим решением даже на довольно грубой сетке, в отличие от результата применения существенно более диссипативного метода конечных объемов. Эффективность реализации двух рассмотренных лимитеров протестирована на задаче Сода в квазиодномерной постановке. Численная схема с лимитером MUSCL имеет более высокую численную вязкость на разрывах, чем с лимитером WENO_S. Показано, что использованный индикатор KXRCF может не определять некоторые немонотонности численного решения.

Благодарности

Работа выполнена при частичной финансовой поддержке РФФИ (проекты 18-01-00252 и 16-02-00656).

Список литературы

[1]. Harris R.E., Collins E., Sescu A., Luke E.A., Strutzenberg L.L., West J.S. High-order Discontinuous Galerkin and hybrid RANS/LES method for prediction of launch vehicle lift-off acoustic environments. 20th AIAA/CEAS Aeroacoustics Conference (Atlanta, GA). In Papers – American Institute of Aeronautics and Astronautics. 2014. No. 3.

[2]. Toulorge T. Efficient Runge–Kutta Discontinuous Galerkin Methods Applied to Aeroacoustic. PhD thesis. Katholieke Universiteit Leuven. 2012.

[3]. Bosnyakov I., Lyapunov S.V., Troshin A., Vlasenko V.V., Wolkov A.V. A High-Order Discontinuous Galerkin Method for External Aerodynamics. 32nd AIAA Applied Aerodynamics Conference, 2981.

[4]. Cockburn B. An Introduction to the Discontinuous Galerkin Method for Convection-Dominated Problems. Lecture Notes in Mathematics. Advanced Numerical Approximation of Nonlinear Hyperbolic Equations, No. 1697, 1998, pp. 151-268.

[5]. Димитриенко Ю. И., Коряков М. Н., Захаров А. А. Применение метода RKDG для численного решения трехмерных уравнений газовой динамики на

неструктурированных сетках. Математическое моделирование и численные методы, № 4 (8), 2015, стр. 75-91. doi: 10.18698/2309-3684-2015-4-7591

[6]. Touze C.L., Murrone A., Guillard H. Multislope MUSCL method for general unstructured meshes. Journal of Computational Physics. 2015. No. 284. Pp. 389-418. doi: 10.1016/j.jcp.2014.12.032

[7]. Галепова В.Д., Лукин В.В., Марчевский И.К., Фуфаев И.Н. Сравнительное исследование лимитеров семейства WENO и Hermite WENO для расчета одномерных течений газа методом RKDG. Препринты ИПМ им. М.В.Келдыша, № 131, 2017, 32 стр., doi:10.20948/prepr-2017-131

[8]. Qiu J., Shu C.-W. Runge – Kutta Discontinuous Galerkin Method Using WENO Limiters. SIAM J. Sci. Comput., vol. 26, № 3, 2005, pp. 907-929. doi: 10.1137/S1064827503425298

[9]. Zhong X., Shu C.-W. A simple weighted essentially nonoscillatory limiter for Runge – Kutta discontinuous Galerkin methods. J. Comp. Phys., vol. 232, 2013, pp. 397-415. doi: 10.1016/j.jcp.2012.08.028

[10]. Zhu J., Zhong X., Shu C.-W., Qiu, J.-X. Runge – Kutta Discontinuous Galerkin Method with a Simple and Compact Hermite WENO limiter. Communications in Computational Physics, vol. 19, № 4, 2016, pp. 944-969. 10.4208/cicp.070215.200715a

[11]. Toro E.F. Riemann Solvers and Numerical Methods for Fluid Dynamics. Berlin: Springer, 2009, 724 p. doi: 10.1007/b79761

[12]. Cockburn B., Shu C.-W. TVB Runge – Kutta local projection discontinuous Galerkin finite element method for scalar conservation laws II: General framework. Math. Comp., vol. 52, 1989, pp. 411-435. doi: 10.2307/2008474

[13]. Cockburn B., Shu C.-W. Runge–Kutta Discontinuous Galerkin Methods for Convection-Dominated Problems. Journal of Scientific Computing., vol. 16, issue 3, 2001, pp. 173-261. doi: 10.1023/A:1012873910884

[14]. Krivodonova L., Xin J., Remacle J.F., Chevaugeon N., Flaherty J.E. Shock detection and limiting with discontinuous Galerkin methods for hyperbolic conservation laws // Applied Numerical Mathematics, vol. 48, № 3-4, . 2004, pp. 323-338. doi: 10.1016/j.apnum.2003.11.002

[15]. Tam, C.K.W., Webb, J.C. Dispersion-Relation-Preserving Finite Difference Schemes for Computational Acoustics. Journal of Computational Physics, vol. 107, 1993, pp. 262-281. doi: 10.1006/jcph.1993.1142

On 2D gas dynamics simulation using RKDG method on structured rectangular meshes

[1] *V.N. Korchagova <v.korchagova@ispras.ru>*
[1] *I.N. Fufaev <fufaeviwan@yandex.ru>*
[1] *S.M. Sautkina <sofyasautkina@yandex.ru>*
[1,2] *V.V. Lukin <vvlukin@gmail.com>*
[1] *Bauman Moscow State Technical University, 5, 2nd Baumanskaya st., Moscow, 105005, Russia*
[2] *Keldysh Institute of Applied Mathematics, 4, Miusskaya sq., Moscow, 125047, Russia*

Abstract. In this paper, we describe the variant of the Runge – Kutta Discontinuous Galerkin (RKDG) method for numerical simulation of 2D gas dynamics flows on structured rectangular meshes. The RKDG algorithm was implemented with C++ code based on the experience in 1D case implementation and verified on simple tests. The advantage of the RKDG method over the most popular finite volume method (FVM) is discussed: three basis functions being applied in the framework of the RKDG approach lead to a considerable decrease of the numerical dissipation rate with respect to FVM. The numerical code contains implementations of Lax – Friedrichs, HLL and HLLC numerical fluxes, KXRCF troubled cell indicator and WENO_S and MUSCL slope limiters. Results of the acoustic pulse simulation on a sufficiently coarse mesh with the piecewise-linear approximation show a good agreement with the analytical solution in contrast to the FVM numerical solution. For the Sod problem results of the shock wave, rarefaction wave and the contact discontinuity propagation illustrate the dependence of the scheme resolution on the numerical fluxes, troubled cell indicator and limitation technique choice. The numerical scheme with MUSCL slope limiter has the higher numerical dissipation in comparison to one with WENO_S limiter. It is shown, that in some cases KXRCF troubled cell indicator doesn't detect numerical solution non-monotonicity.

Keywords: RKDG; Euler equations; MUSCL; WENO; limiter; indicator.

DOI: 10.15514/ISPRAS-2018-30(2)-13

For citation: Korchagova V.N., Fufaev I.N., Sautkina S.M., Lukin V.V. On 2D gas dynamics simulation using RKDG method on structured rectangular meshes. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 2, 2018, pp. 285-300 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-13

References

[1]. Harris R.E., Collins E., Sescu A., Luke E.A., Strutzenberg L.L., West J.S. High-order Discontinuous Galerkin and hybrid RANS/LES method for prediction of launch vehicle lift-off acoustic environments. 20th AIAA/CEAS Aeroacoustics Conference (Atlanta, GA). In Papers – American Institute of Aeronautics and Astronautics. 2014. No. 3.

[2]. Toulorge T. Efficient Runge–Kutta Discontinuous Galerkin Methods Applied to Aeroacoustic. PhD thesis. Katholieke Universiteit Leuven. 2012.

[3]. Bosnyakov I., Lyapunov S.V., Troshin A., Vlasenko V.V., Wolkov A.V. A High-Order Discontinuous Galerkin Method for External Aerodynamics. 32nd AIAA Applied Aerodynamics Conference, 2981.

[4]. Cockburn B. An Introduction to the Discontinuous Galerkin Method for Convection-Dominated Problems. Lecture Notes in Mathematics. Advanced Numerical Approximation of Nonlinear Hyperbolic Equations, No. 1697, 1998, pp. 151-268.

[5]. Dimitrienko Yu.I., Koryakov M.N., Zakharov A.A. Application of RKDG method for computational solution of three-dimensional gas-dynamic equations with non-structured grids. Mathematical Modeling and Computational Methods, vol. 4, No. 8, . 2015, pp. 75-91. doi: 10.18698/2309-3684-2015-4-7591 (in Russian).

[6]. Touze C.L., Murrone A., Guillard H. Multislope MUSCL method for general unstructured meshes. Journal of Computational Physics. 2015. No. 284. Pp. 389-418. doi: 10.1016/j.jcp.2014.12.032

[7]. Galepova V.D., Lukin V.V., Marchevsky I.K., Fufaev I.N. Comparative study of WENO and Hermite WENO limiters for gas flows numerical simulations using the RKDG method. KIAM Preprint № 131, Moscow, 2017, 32 p. doi:10.20948/prepr-2017-131 (in Russian).

[8]. Qiu J., Shu C.-W. Runge – Kutta Discontinuous Galerkin Method Using WENO Limiters. SIAM J. Sci. Comput., vol. 26, № 3, 2005, pp. 907-929. doi: 10.1137/S1064827503425298

[9]. Zhong X., Shu C.-W. A simple weighted essentially nonoscillatory limiter for Runge – Kutta discontinuous Galerkin methods. J. Comp. Phys., vol. 232, 2013, pp. 397-415. doi: 10.1016/j.jcp.2012.08.028

[10]. Zhu J., Zhong X., Shu C.-W., Qiu, J.-X. Runge – Kutta Discontinuous Galerkin Method with a Simple and Compact Hermite WENO limiter. Communications in Computational Physics, vol. 19, № 4, 2016, pp. 944-969. 10.4208/cicp.070215.200715a

[11]. Toro E.F. Riemann Solvers and Numerical Methods for Fluid Dynamics. Berlin: Springer, 2009, 724 p. doi: 10.1007/b79761

[12]. Cockburn B., Shu C.-W. TVB Runge – Kutta local projection discontinuous Galerkin finite element method for scalar conservation laws II: General framework. Math. Comp., vol. 52, 1989, pp. 411-435. doi: 10.2307/2008474

[13]. Cockburn B., Shu C.-W. Runge–Kutta Discontinuous Galerkin Methods for Convection-Dominated Problems. Journal of Scientific Computing., vol. 16, issue 3, 2001, pp. 173-261. doi: 10.1023/A:1012873910884

[14]. Krivodonova L., Xin J., Remacle J.F., Chevaugeon N., Flaherty J.E. Shock detection and limiting with discontinuous Galerkin methods for hyperbolic conservation laws // Applied Numerical Mathematics, vol. 48, № 3-4, . 2004, pp. 323-338. doi: 10.1016/j.apnum.2003.11.002

[15]. Tam, C.K.W., Webb, J.C. Dispersion-Relation-Preserving Finite Difference Schemes for Computational Acoustics. Journal of Computational Physics, vol. 107, 1993, pp. 262-281. doi: 10.1006/jcph.1993.114

Применение параллельных алгоритмов при численном моделировании кровотока в квазиодномерном приближении

А.Н. Авдеева <aan-bmstu@yandex.ru>
В.В. Пузикова <vvp@dms-at.ru>
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана),
105005, Россия, г. Москва, ул. 2-я Бауманская, дом 5, стр. 1

Аннотация. Главной целью современного гемодинамического моделирования является предсказание поведения давления крови в артериях, а также изучение комплексного воздействия разнообразных факторов на характеристики сердечно-сосудистой системы. Наиболее популярными при этом являются квазиодномерные модели течения крови по сосудам, позволяющие моделировать кровоток во всей сосудистой системе. Поскольку полномасштабное моделирование сердечно-сосудистой системы требует больших вычислительных затрат, актуальной является задача распараллеливания вычислений. В данной работе проведено исследование эффективности распараллеливания вычислений при численном моделировании кровотока в квазиодномерном приближении. Для простоты рассмотрена задача о моделировании течения крови в отдельном кровеносном сосуде. При построении параллельного алгоритма был применен метод декомпозиции области. В каждой подобласти задача на каждом шаге по времени расщепляется на гиперболическую и параболическую подзадачи. Для решения гиперболической подзадачи используется интегро-интерполяционный метод, основанный на схеме MUSCL. Для интегрирования по времени применяются методы Рунге-Кутты и Адамса-Башфорта второго порядка. Для решения параболической подзадачи используется метод Кранка-Николсона. На стыках подобластей интерфейсные условия образуют нелинейные системы с тремя неизвестными. Эти системы решаются при помощи метода Ньютона. С помощью профилировщика AMD CodeAnalyst была определена структура временных затрат при решении тестовой задачи в последовательном режиме. При помощи закона Амдала получены оценки максимально возможного ускорения при распараллеливании наиболее дорогостоящих с вычислительной точки зрения операций. При реализации полученного алгоритма в разработанном авторами настоящей работы программном комплексе использовались технология OpenMP и библиотека MPI. Расчеты проводились на учебно-вычислительном кластере кафедры ФН-2 «Прикладная математика» МГТУ им. Н.Э. Баумана. Результаты вычислительных экспериментов показали, что выигрыш по времени, достигаемый за счет использования библиотеки

MPI, не превышает нескольких процентов по сравнению с применением технологии OpenMP. В связи с этим, принимая во внимания простоту распараллеливания алгоритмов посредством OpenMP, можно остановить выбор на данной технологии, однако использование MPI позволяет сделать программный комплекс универсальным – работающим как на системах с общей памятью, так и на системах с распределенной памятью.

Ключевые слова: технология OpenMP; MPI; квазиодномерная модель; кровоток; метод декомпозиции области; параллельный алгоритм; кластер; метод MUSCL; расщепление Годунова**.**

DOI: 10.15514/ISPRAS-2018-30(2)-15

Для цитирования: Авдеева А.Н., Пузикова В.В. Применение параллельных алгоритмов при численном моделировании кровотока в квазиодномерном приближении. Труды ИСП РАН, том 30, вып. 2, 2018 г., стр. 301-316. DOI: 10.15514/ISPRAS-2018-30(2)-15

1. Введение

В последние годы главной причиной смертности в мире являются сердечно-сосудистые заболевания. В настоящее время значительно увеличивается количество стентирований кровеносных сосудов и ангиопластик, причем в проекте государственной программы по развитию здравоохранения планируется дальнейшее усиление мер по оказанию специализированной медицинской помощи больным с острой сосудистой патологией. Нередко для восстановления кровообращения в пораженных сосудах помимо медикаментозного лечения проводятся дорогостоящие реконструктивные операции, и зачастую невозможно объективно оценить, какой тип оперативного вмешательства будет оптимальным для конкретного пациента, а также, насколько близок будет кровоток в сосуде к нормальному после операции. Таким образом, очень важной задачей представляется не только усовершенствование методов ранней диагностики сердечно-сосудистых заболеваний, но и разработка методов, позволяющих смоделировать для конкретного пациента последствия проведения той или иной реконструктивной кардиохирургической операции.

Эффективным методом решения данной проблемы является использование вычислительного эксперимента: в виртуальной системе кровообращения можно реализовать различные сценарии операции для конкретного пациента, спрогнозировать исход той или иной тактики лечения и выбрать оптимальную [1, 2]. Использование относительно недорогих средств математического моделирования может обеспечить значительный прогресс в разработке и оптимизации формы искусственных клапанов сердца, шунтов, имплантируемых насосов крови, способов управления ими, и, самое главное, позволит исследовать их влияние на гемодинамику и спрогнозировать успешность применения данного метода терапии для конкретного пациента в дооперационный период. Прогностическую ценность и клинический

потенциал персонализированного моделирования сердечно-сосудистой системы для ряда клинически значимых параметров доказал международный исследовательский проект «euHeart» [3], который возглавляла компания Royal Philips Electronics.

Главной целью современного гемодинамического моделирования является предсказание поведения давления крови в артериях, а также изучение комплексного воздействия разнообразных факторов на характеристики сердечно-сосудистой системы. Наиболее популярными при этом являются квазиодномерные модели течения крови по сосудам [4–8], позволяющие моделировать кровоток во всей сосудистой системе. Параметры отдельных сосудов (длины сосудов, скорости распространения пульсовых волн и т.д.) берут из результатов медицинских исследований, например, цветового дуплексного сканирования артерий.

Поскольку полномасштабное моделирование сердечно-сосудистой системы требует больших вычислительных затрат, актуальной является задача распараллеливания вычислений. Вопросам разработки параллельных алгоритмов решения разнообразных задач вычислительной механики и анализу их эффективности посвящено большое число исследований [9-15]. При этом расматриваются вопросы проведения расчетов как на вычислительных системах кластерного типа при помощи библиотеки параллельных вычислений MPI [16, 17], так и на системах с общей памятью при помощи таких технологий параллельного программирования, как Intel® Cilk™ Plus [18], Intel® TBB [19], OpenMP [20].

Целью данной работы является изучение возможности эффективного распараллеливания вычислений при численном моделировании кровотока в квазиодномерном приближении. Для простоты рассматривается задача о моделировании течения крови в отдельном кровеносном сосуде. При построении параллельного алгоритма применяется метод декомпозиции области [21]. При реализации полученного алгоритма в разработанном авторами настоящей работы программном комплексе используются технология OpenMP и библиотека MPI.

2. Квазиодномерная модель кровотока в отдельном сосуде

Для простоты рассмотрим один участок эластичного кровеносного сосуда (рис. 1) в цилиндрической системе координат (r,θ,x). Внутренний радиус сосуда R зависит от координаты x и времени t. Кровь полагается однородной и несжимаемой (плотность $\rho = const$) ньютоновской жидкостью (вязкость $\nu = const$). Течение характеризуется скоростью $\vec{v} = (v_r, v_\theta, v_x)$ и давлением P. Отметим, что P – это внутреннее давление, т.е.

$$P = P_{ext} + P_{art}.$$

Здесь P_{ext} – внешнее давление, т.е. атмосферное давление, а P_{art} – трансмуральное давление, которое в медицине принято называть артериальным давлением [1, 2]. Предполагаем, что P_{ext} постоянно вдоль осевой переменной x. Считается, что силой тяжести можно пренебречь [7, 8]. Сосуд и течение крови полагаются осесимметричными, поэтому $v_\theta = 0$ и ни одна из переменных не зависит от θ. Полагаем, что течение описывается уравнением неразрывности и уравнением Навье-Стокса [7, 8]. Эти уравнения дополняются граничным условием прилипания

$$v_x(R) = 0.$$

Кроме того, предполагаем, что стенка сосуда движется только вдоль r, т.е.

$$v_r(R, x, t) = \frac{\partial R}{\partial t}.$$

Из этого уравнения следует, что давление постоянно вдоль радиуса.

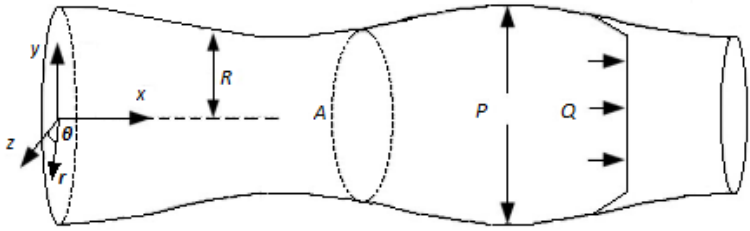


Рис. 1. Расчетная область
Fig. 1. Computational domain

Перепишем уравнение неразрывности в интегральном виде:

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0. \qquad (1)$$

Здесь Q –расход потока, A –площадь поперечного сечения:

$$Q = \int_0^R 2\pi v_x r dr, \quad A = \pi R^2.$$

Для описания вязкоупругого поведения стенки сосуда используем модель Кельвина – Фойгта. Тогда внутреннее давление можно вычислять следующим образом [7]:

$$P = P_{ext} + \beta(\sqrt{A} - \sqrt{A_0}) + \nu_s \frac{\partial A}{\partial t}. \qquad (2)$$

Здесь β – коэффициентом жесткости и ν_s – коэффициент вязкости.

В работе [7] показано, что при малых возмущений можно полагать, что

$$\frac{A}{\rho}\frac{\partial}{\partial x}\left(\nu_s \frac{\partial Q}{\partial x}\right) \approx C_\nu \frac{\partial^2 Q}{\partial x^2}.$$

Здесь $C_\nu = \frac{A\nu_s}{\rho}$. Тогда уравнение Навье–Стокса в интегральном виде с учетом (2) можно переписать следующим образом:

$$\frac{\partial Q}{\partial t} + \frac{\partial}{\partial x}\left(\frac{Q^2}{A} + \frac{\beta}{3\rho}A^{3/2}\right) - C_\nu \frac{\partial^2 Q}{\partial x^2} = -C_f \frac{Q}{A} + \frac{A}{\rho}\left(\frac{\partial(\beta\sqrt{A_0})}{\partial x} - \frac{2}{3}\sqrt{A}\frac{\partial \beta}{\partial x}\right). \quad (3)$$

Здесь $C_f = 22\pi\nu$ [7]. Уравнения (1), (3) можно записать следующим образом:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = S. \quad (4)$$

Здесь U – консервативная переменная, F – соответствующий поток, S – источник:

$$U = \begin{pmatrix} A \\ Q \end{pmatrix}, \quad F = F_c + F_\nu = \begin{pmatrix} Q \\ \frac{Q^2}{A} + \frac{\beta}{3\rho}A^{3/2} \end{pmatrix} + \begin{pmatrix} 0 \\ -C_\nu \frac{\partial Q}{\partial x} \end{pmatrix},$$

$$S = \begin{pmatrix} 0 \\ -C_f \frac{Q}{A} + \frac{A}{\rho}\left(\frac{\partial(\beta\sqrt{A_0})}{\partial x} - \frac{2}{3}\sqrt{A}\frac{\partial \beta}{\partial x}\right) \end{pmatrix}.$$

Заметим, что поток состоит из двух частей, конвективной F_c и диффузионной F_ν. В общем случае, при работе с конвективными и диффузионным потоками применяются разные численные методы. Поэтому диффузионные слагаемые часто отделяют и переносят в правую часть. Таким образом, мы можем переписать задачу (4) следующим образом:

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = S + D, \quad F = F_c, \quad D = \begin{pmatrix} 0 \\ C_\nu \frac{\partial^2 Q}{\partial x^2} \end{pmatrix}. \quad (5)$$

Запишем в общем виде, как будут выглядеть начальные и граничные условия для каждого сосуда. В качестве начального условия полагаем

$$U(x,0) = U_{\text{in}}(x).$$

На левом конце рассматриваемого сосуда зададим граничное условие первого рода, а на правом конце – условие свободного выхода потока:

$$U(0,t) = U_0(t), \qquad \frac{\partial U(L,t)}{\partial \vec{n}} = 0.$$

3. Численный метод

Как и в работах [7, 8] используем метод дробных шагов: исходную задачу (5) разделим на гиперболическую и параболическую соответственно:

$$\frac{\partial U}{\partial t}+\frac{\partial F}{\partial x}=S, \tag{6}$$

$$\frac{\partial U}{\partial t}=D. \tag{7}$$

Тогда, решение исходной задачи (5) может быть заменено последовательным решением этих двух подзадач. В данной работе используется метод расщепления Годунова: на каждом шаге по времени будем решать гиперболическую подзадачу, чтобы получить прогноз U^*, который используется в качестве начального значения в параболической подзадаче

$$U^j \overset{\Delta t,H}{\rightarrow} U^* \overset{\Delta t,P}{\rightarrow} U^{j+1}.$$

Здесь H и P означает решение гиперболической и параболической подзадач на j-м шаге по времени. Второй шаг можно рассматривать как корректор.

Для решения гиперболической подзадачи (6) используем метод контрольных объемов: каждую подобласть разделим на ячейки I_i. Для каждой такой ячейки должен выполняться закон сохранения

$$\int_{I_i}\frac{\partial U}{\partial t}dx+\int_{I_i}\frac{\partial F}{\partial x}dx=\int_{I_i}Sdx.$$

При построении дискретных аналогов производных по пространству используем MUSCL-ограничитель [22]. Для решения дифференциальное уравнение первого порядка, получающегося после дискретизации по пространству в разработанном программном комплексе реализованы два метода интегрирования по времени второго порядка: метод Рунге–Кутты и двухшаговый метод Адамса–Башфорта.

Заметим, что для A решать параболическую подзадачу (7) не требуется, так как $A^{j+1}=A^*$. Поэтому (7) принимает вид:

$$\frac{\partial Q}{\partial t}=C_v\frac{\partial^2 Q}{\partial x^2}.$$

Для решения данной задачи используем метод Кранка-Николсона:

$$\frac{Q_i^{j+1}-Q_i^j}{\Delta t}=\frac{C_v}{2}\left(\frac{Q_{i+1}^{j+1}-2Q_i^{j+1}+Q_{i-1}^{j+1}}{\Delta x^2}+\frac{Q_{i+1}^{j}-2Q_i^{j}+Q_{i-1}^{j}}{\Delta x^2}\right). \tag{8}$$

Здесь $Q^j=Q^*$, т.е. Q^j является решением гиперболической подзадачи. Разностная задача (8) решается методом прогонки.

4. Декомпозиция области

Для распараллеливания вычислений при решении задач вычислительной механики часто используют методы декомпозиции области [10]. Идея методов декомпозиции заключается в том, что расчетная область разбивается на пересекающиеся или непересекающие подобласти и исходная задача представляется в виде совокупности вспомогательных краевых задач в этих подобластях. При этом на границах подобластей, совпадающих с границами исходной расчетной области, ставятся граничные условия из исходной задачи, а на остальных границах подобластей, называемых внутренними, ставятся условия сопряжения (интерфейсные условия). Решение вспомогательных задач может осуществляться параллельно.

На рис. 2 показан пример декомплозиции расчетной области на две подобласти: в месте стыковки подобластей имеем четыре неизвестные: A_p, Q_p на выходе из родительской артерии и A_d, Q_d на входе в дочернюю. Здесь и далее нижний индекс p, d определяет принадлежность к родительской и дочерней артериям соответсвенно.

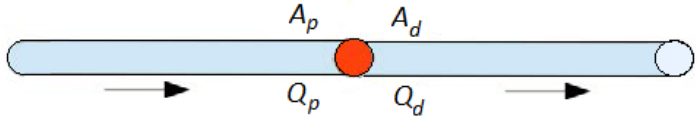


Рис. 2. Декомпозиция области
Fig. 2. Domain decomposition

В точке соединения на каждом $n+1$-м шаге по времени должны выполняться уравнение расхода потока жидкости и закон сохранения импульса:

$$Q_p^{n+1} = Q_d^{n+1}, \tag{9}$$

$$\frac{1}{2}\rho_p\left(\frac{Q_p^{n+1}}{A_p^{n+1}}\right)^2 + P_p^{n+1} = \frac{1}{2}\rho_d\left(\frac{Q_d^{n+1}}{A_d^{n+1}}\right)^2 + P_d^{n+1}. \tag{10}$$

Значения давления P_p^{n+1} и P_d^{n+1} зависят от площади поперечного сечения и вычисляются при помощи (2).

В работе [7] показано, что в физиологических условиях число Уомерсли $\mathrm{W_o}$ достаточно мало, из-за чего источниковые члены в рассматриваемой системе уравнений оказываются незначительными и в системе преобладает гиперболичность. Поэтому два недостающих условия на стыке подобластей могут получены при помощи характеристик или инвариантов Римана[22]

$$W_{1,2} = \frac{Q}{A} \pm 4c. \tag{11}$$

Здесь c – скорость Моэнса – Кортевега, которая является скоростью волны давления, т.е. скоростью пульсовой волны:

$$c = \sqrt{\frac{\beta}{2\rho} A^{\frac{1}{2}}}.$$

Два собственных значения λ_1 и λ_2 имеют противоположные знаки:

$$\lambda_{1,2} = \frac{Q}{A} \pm c.$$

Соответственно, на каждом конце вычислительной области существует ровно одна входящая характеристика и одна выходящая. Поскольку $W_{1,2}$ постоянны вдоль линий $D_t X_{1,2} = \lambda_{1,2}$ в пространственно-временной плоскости, можно получить значения $W_1^{n+1}(L_i)$ и $W_2^{n+1}(0)$, применив интерполяцию данных с n-го шага по времени:

$$W_1^{n+1}(L_i) = W_1^n(L - \lambda_1^n(L_i)\Delta t), \quad W_2^{n+1}(0) = W_2^n(-\lambda_2^n(0)\Delta t), \quad t > 0. \quad (12)$$

Таким образом, в родительской артерии значение $(W_1)_p^{n+1}$ можно вычислить по формуле (12) и потребовать, чтобы оно было равно значению $W_1(U_p^{n+1})$, которое задается соотношением (11). В итоге имеем уравнение

$$(W_1)_p^{n+1} = W_1(U_p^{n+1}). \quad (13)$$

Аналогично строится уравнение для W_2 для дочерней артерии:

$$(W_2)_d^{n+1} = W_2(U_d^{n+1}). \quad (14)$$

Таким образом, имеем систему из четырех уравнений (9), (10), (13) и (14) с четырьмя неизвестными. Ее решение даст значения неизвестной U на стыке подобластей. С учетом условия (9) уравнения (10), (13), (14) примут вид:

$$\begin{cases} \frac{1}{2}\rho\left(\frac{Q_p^{n+1}}{A_p^{n+1}}\right)^2 + P_p^{n+1} - \frac{1}{2}\rho\left(\frac{Q_p^{n+1}}{A_d^{n+1}}\right)^2 + P_d^{n+1} = 0, \\ (W_1)_p^{n+1} - W_1(A_p^{n+1}, Q_p^{n+1}) = 0, \\ (W_2)_d^{n+1} - W_2(A_d^{n+1}, Q_p^{n+1}) = 0. \end{cases}$$

Таким образом, имеем нелинейную алгебраическую систему вида

$$F(X) = 0, \quad X = (A_p^{n+1}, Q_p^{n+1}, A_d^{n+1})^T.$$

Для ее решения применим итерационный метод Ньютона. Матрица Якоби считается аналитически. В качестве начального приближения будем использовать значения неизвестных с предыдущего шага по времени

$$X^{(0)} = (A_p^n, Q_p^n, A_d^n)^T.$$

Расчетная формула для расчета $k+1$-го итерационного приближения метода Ньютона имеет следующий вид

$$X^{(k+1)} = X^{(k)} - (F'(X^{(k)}))^{-1} F(X^{(k)}), \quad k = 0,1,\text{K}$$

Отметим, что во всех вычислительных экспериментах было достаточно нескольких итераций для достижения точности 10^{-6}.

5. Оценка эффективности распараллеливания вычислений

Определим, какие вычисления имеет смысл распараллелить. Для этого необходимо выделить участки программы, на выполнение которых расходуется наибольшее количество времени. Полная структура временны́х затрат при решении тестовой задачи была определена с помощью профилировщика AMD CodeAnalyst [23]: 95,8 % времени работы программы занимает расчет значений A и Q внутри каждой подобласти (операция 1), а 1,2 % – вычисление неизвестных из интерфейсных условий на стыках подобластей (операция 2). Все прочие операции занимают 3 % времени выполнения расчета (рис. 3).

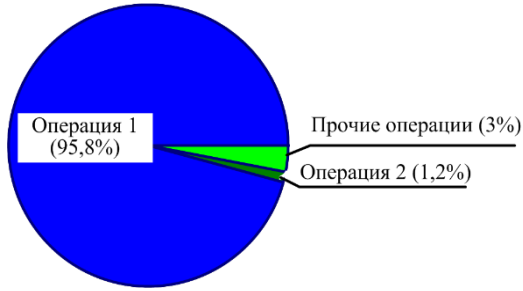


Рис. 3. Структура временны́х затрат при решении тестовой задачи
Fig. 3. The time-cost structure

Оценим максимальное ускорение, которое можно получить при распараллеливании только операции 1 или операций 1 и 2, при помощи закона Амдала [24], который гласит, что для системы из S вычислительных ялер максимально возможное ускорение программы с долей P параллельного кода и $(1-P)$ последовательного кода равно

$$\alpha = \frac{1}{\frac{P}{S} + (1-P)}.$$

Полученные оценки приведены в табл. 1. Они соответствуют случаю идеального распараллеливания, при котором параллельный код на вычислительной системе с S ядрами выполняется в S раз быстрее. Реальное ускорение будет ниже, поскольку при переходе от последовательной программы к параллельной добавятся накладные расходы на поддержку многопоточных вычислений. Далее для распараллеливания вычислений будем использовать технологию параллельного программирования OpenMP и библиотеку MPI. Технология OpenMP предполагает, что пользователь при помощи ключевых слов лишь обозначает задачи, выполняемые параллельно, а организация управления потоками и работы с ними определяются самой технологией. Таким образом, вышеперечисленные накладные расходы на поддержку многопоточности, а, значит, и реальное ускорение, зависят от используемой технологии параллельного программирования.

Табл. 1. Максимально возможное ускорение при распараллеливании операций 1 и 2

Table 1. The maximum possible speed up at operations 1 and 2 in parallel mode

Распараллеливаемые операции	P	Максимальное ускорение, раз			
		2 ядра	4 ядра	8 ядер	12 ядер
1	0,958	1,92	3,55	6,18	8,21
1, 2	0,970	1,94	3,67	6, 61	9,02

Вычислительные эксперименты проводились на учебно-экспериментальном вычислительном кластере кафедры ФН-2 "Прикладная математика" МГТУ им. Н.Э. Баумана [25]. Расчетная область разбивалась на 100 подобластей. Каждая подобласть состояла из 200 контрольных объемов. Моделировалось 1000 шагов по времени. Распараллеливались операции 1 и 2.

На рис. 4 представлена зависимость ускорения от числа вычислительных ядер при распараллеливания при помощи технологии OpenMP.

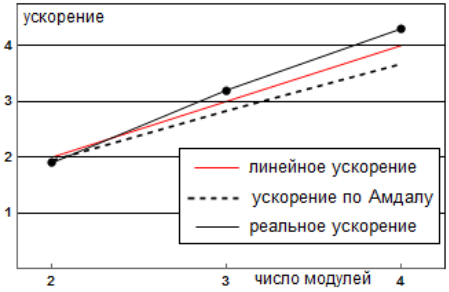


Рис. 4. Реальное ускорение при использовании технологии OpenMP и оценки ускорения
Fig. 4. The real acceleration (OpenMP technology) and acceleration estimates

Видно, что реальное ускорение превышает линейное. Это можно объяснить тем, что при решении задачи на одном процессоре данные постоянно подгружаются в кэш-память, а при распараллеливании задачи происходит деление данных по процессорам, и кэш каждого модуля смог вместить свою порцию данных.

На рис. 5 представлена зависимость ускорения от числа вычислительных ядер при распараллеливании, использующем библиотеку MPI.

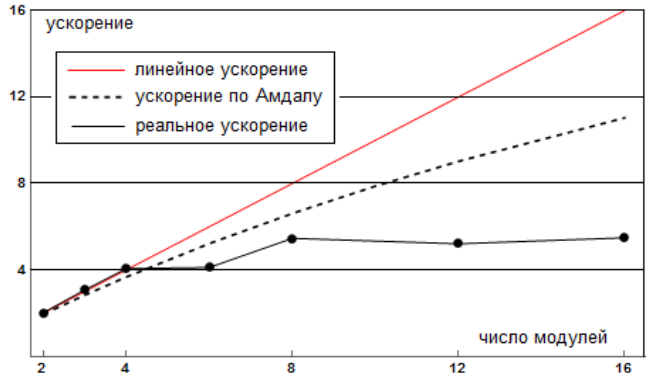


Рис. 5. Реальное ускорение при использовании библиотеки MPI и оценки ускорения
Fig. 5. The real acceleration (MPI library) and acceleration estimates

На рассмотренной тестовой задаче при числе модулей, превышающем четыре, наблюдается существенное расхождение реального ускорения с оценкой по закону Амдала. Это объясняется главным образом влиянием межмодульных обменов. При меньшем числе вычислительных ядер выигрыш по времени, достигаемый за счет использования библиотеки MPI, не превышает нескольких процентов по сравнению с применением технологии OpenMP.

6. Заключение

Исследована эффективность распараллеливания вычислений при численном моделировании кровотока в квазиодномерном приближении. Рассмотрена задача о моделировании течения крови в отдельном кровеносном сосуде. Построение параллельного алгоритма произведено при помощи метода декомпозиции области. В каждой подобласти задача на каждом шаге по времени расщепляется на гиперболическую и параболическую подзадачи при помощи расщепления Годунова. Для решения гиперболической подзадачи используется интегро-интерполяционный метод, основанный на схеме MUSCL. Для интегрирования по времени применяются методы Рунге-Кутты и Адамса-Башфорта второго порядка. Для решения параболической подзадачи используется метод Кранка-Николсона. На стыках подобластей интерфейсные условия образуют нелинейные системы с тремя неизвестными. Эти системы

решаются при помощи метода Ньютона. С помощью профилировщика AMD CodeAnalyst была определена структура временных затрат при решении тестовой задачи в последовательном режиме. При помощи закона Амдала получены оценки максимально возможного ускорения при распараллеливании наиболее дорогостоящих с вычислительной точки зрения операций. При реализации полученного алгоритма в разработанном авторами настоящей работы программном комплексе использовались технология OpenMP и библиотека MPI. Расчеты проводились на учебно-вычислительном кластере кафедры ФН-2 "Прикладная математика" МГТУ им. Н.Э. Баумана.

Результаты вычислительных экспериментов показали, что выигрыш по времени, достигаемый за счет использования библиотеки MPI, не превышает нескольких процентов по сравнению с применением технологии OpenMP. В связи с этим, принимая во внимания простоту распараллеливания алгоритмов посредством OpenMP, можно остановить выбор на данной технологии, однако использование MPI позволяет сделать программный комплекс универсальным –работающим как на системах с общей памятью, так и на системах с распределенной памятью.

Благодарности

Работа выполнена при частичной финансовой поддержке гранта Президента Российской Федерации для молодых российских ученых-кандидатов наук (проект МК-743.2018.8).

Список литературы

[1]. Quarteroni A., Formaggia L. Mathematical Modelling and Numerical Simulation of the Cardiovascular System. Handbook on numerical analysis, Ed. By P. G. Ciarlet, J. L. Lions. Amsterdam: Elsevier, 2004, 101 p. DOI: 10.1016/S1570-8659(03)12001-7

[2]. Quarteroni A., Formaggia L., Veneziani A. Cardiovascular Mathematics: Modeling and Simulation of the Circulatory System. Milano: Springer, 2011, 526 p.

[3]. Goals – euHeart. URL: https http://www.euheart.eu/index_id_27.html (дата обращения: 09.05.2018).

[4]. Formaggia L., Lamponi D., Quarteroni A. One dimensional models for blood flow in arteries. Journal of Engineering Mathematics, vol. 47, 2003, pp. 251-276. DOI: 10.1023/B:ENGI.0000007980.01347.29.

[5]. Azer K., Peskin C. S. A one-dimensional model of blood flow in arteries with friction and convection based on the Womersley velocity profile. Cardiovasc. Eng., vol. 7, 2007, pp. 51–73. DOI: 10.1007/s10558-007-9031-y

[6]. Sherwin S. J., Franke V., Peiro J., Parker K. One-dimensional modeling of vascular network in space-time variables. Journal of Engineering Mathematics. vol. 47, 2003, pp. 217–250. DOI: 10.1023/B:ENGI.0000007979.32871.e2

[7]. Wang X., Delestre O., Fullana J.-M., Saito M., Ikenaga Y., Matsukawa M., Lagree P.-Y. Comparing different numerical methods for solving arterial 1D flows in networks. Comput. Methods Appl. Mech. Eng., vol. 15, 2012, pp. 61–62. DOI: 10.1080/10255842.2012.713677

[8]. Wang X., Fullana J.-M., Lagrée P.-Y. Verification and comparison of four numerical schemes for a 1D viscoelastic blood flow model. Computer Methods in Biomechanics and Biomedical Engineering, 2014, pp.1-22. DOI: 10.1080/10255842.2014.948428

[9]. Аветисян А.И., Бабкова В.В., Гайсарян С.С., Губарь А.Ю. Разработка параллельного программного обеспечения для решения трехмерной задачи о рождении торнадо по теории Николаевского. Матем. моделирование, том. 20, № 8, . 2008 г., стр. 28–40.

[10]. Марчевский И.К., Токарева С.А. Сравнение эффективности параллельных алгоритмов решения задач газовой динамики на разных вычислительных комплексах. Вестник МГТУ им. Н.Э. Баумана. Серия "Естественные науки", № 1 2009 г., стр. 90–97.

[11]. Марчевский И.К., Щеглов Г.А. Применение параллельных алгоритмов при решении задач гидродинамики методом вихревых элементов. Вычислительные методы и программирование, том 11, стр. 105–110.

[12]. Морева В.С. Способы ускорения вычислений при решении плоских задач аэродинамики методов вихревых элементов. Вестник МГТУ им. Н.Э. Баумана. Серия "Естественные науки", № S, 2011 г., стр. 83–95.

[13]. Лукин В.В., Марчевский И.К., Морева В.С., Попов А.Ю., Шаповалов К.Л., Щеглов Г.А. Учебно-экспериментальный вычислительный кластер. Ч. 2. Примеры решения задач. Вестник МГТУ им. Н.Э. Баумана. Серия "Естественные науки". 2012. № 4. С. 82–102.

[14]. Марчевский И.К., Пузикова В.В. Исследование эффективности распараллеливания вычислений при моделировании течений вязкой несжимаемой среды методом LS-STAG на системах с общей памятью. Вычислительные методы и программирование, том 16, 2015 г., стр. 595–606.

[15]. Пузикова В.В. Реализация параллельных вычислений в программном комплексе «LS-STAG_turb» для моделирования течений вязкой несжимаемой среды на системах с общей памятью. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 221-242. DOI: 10.15514/ISPRAS-2016-28(1)-13

[16]. MPICH Overview | MPICH. URL: https://www.mpich.org/about/overview/ (дата обращения: 09.05.2018).

[17]. Avetisyan A.I., Gaisaryan S.S., Ivannikov V.P., Padaryan V.A. Productivity prediction of MPI programs based on models. Autom. Remote Control., vol. 68, 2007, pp. 750-759. DOI: 10.1134/S0005117907050037

[18]. Intel (R) Cillk (TM) Plus | Intel® Software. URL: https://software.intel.com/ru-ru/node/522579 (дата обращения 09.05.2018).

[19]. Reinders J. Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. Sebastopol: O'Reilly, 2007, 336 p.

[20]. OpenMP FAQ – OpenMP. URL: https://www.openmp.org/about/openmp-faq/ (дата обращения: 09.05.2018).

[21]. Quarteroni A., Valli A. Domain decomposition methods for partial differential equations. Oxford: Clarendon Press, 1999, 360 p.

[22]. Годунов С.К., Забродин А.В., Иванов М.Я., Крайко А.Н., Прокопов Г.П. Численное решение многомерных задач газовой динамики. М.: Изд-во "Наука", 1976. 400 с.

[23]. Drongowski P., Lei Yu, Swehosky F., Suthikulpanit S., Richter R., Incorporating Instruction-Based Sampling into AMD CodeAnalyst. 2010 IEEE International

Symposium on Performance Analysis of Systems & Software (ISPASS 2010), White Plains, NY, 2010, pp. 119-120. DOI: 10.1109/ISPASS.2010.5452049.

[24]. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем. М.: Изд-во Моск. ун-та, 2010, 544 с.

[25]. Лукин В.В., Марчевский И.К. Учебно-экспериментальный вычислительный кластер. Ч.1. Инструментарий и возможности. Вестник МГТУ им. Н.Э. Баумана. Серия «Естественные науки», № 4, 2011 г., стр. 28–43.

Application of parallel algorithms for numerical simulation of quasi-one dimensional blood flow

A.N. Avdeeva <aan--bmstu@yandex.ru>
V.V. Puzikova <vvp@dms-at.ru>
Federal state budgetary institution of higher professional education «Bauman Moscow State Technical University (National research university of technology)», 5/1, 2-nd Baumanskaya st., Moscow, 105005, Russia

Abstract. The main goal of modern hemodynamic simulation is the prediction of blood pressure in the arteries, as well as the study of the various factors complex effect on the cardiovascular system characteristics. Quasi-one dimensional models of blood flow through blood vessels are the most popular. They allow to model the blood flow in the entire vascular system. Since full-scale simulation of the cardiovascular system requires large computational costs, the problem of parallelizing computation is actual. In this paper, the efficiency study was carried out for parallel algorithms in the numerical simulation of blood flow in the quasi-one-dimensional approximation. For simplicity, we consider the problem of the blood flow simulation in a separate blood vessel. When constructing a parallel algorithm, the domain decomposition method was applied. In each subdomain, the problem at each time step splits into a hyperbolic and parabolic subproblems. To solve the hyperbolic subtask, an integro-interpolation method based on the MUSCL scheme is used. To integrate over time, the second order Runge-Kutta and Adams-Bashfort methods are applied. The Crank-Nicholson method is used to solve the parabolic subproblem. At the subdomains conjunctions, the interface conditions are non-linear systems with three unknowns. These systems are solved using the Newton method. The time-cost structure was obtained for solving the test problem in a serial mode using the AMD CodeAnalyst profiler. Computations were carried out at the cluster of the BMSTU "Applied Mathematics" FN-2 department. The computational results showed that the time gain achieved by using the MPI library does not exceed a few percent in comparison with the OpenMP technology usage. Taking into account the simplicity of parallelizing algorithms through OpenMP, we can choose this technology, but MPI usage allows to make the software package universal (it can work both on shared memory systems and on distributed memory systems).

Keywords: OpenMP technology; MPI; quasi-one dimensional model; blood flow; domain decomposition method; parallel algorithm; cluster; MUSCL scheme; Godunov splitting.

DOI: 10.15514/ISPRAS-2018-30(2)-15

For citation: Avdeeva A.N., Puzikova V.V. Application of parallel algorithms for numerical simulation of quasi-one dimensional blood flow. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 2, 2018, pp. 301-316 (in Russian). DOI: 10.15514/ISPRAS-2018-30(2)-15

References

[1]. Quarteroni A., Formaggia L. Mathematical Modelling and Numerical Simulation of the Cardiovascular System. Handbook on numerical analysis, Ed. By P. G. Ciarlet, J. L. Lions. Amsterdam: Elsevier, 2004, 101 p. DOI: 10.1016/S1570-8659(03)12001-7

[2]. Quarteroni A., Formaggia L., Veneziani A. Cardiovascular Mathematics: Modeling and Simulation of the Circulatory System. Milano: Springer, 2011, 526 p.

[3]. Goals – euHeart. URL: https http://www.euheart.eu/index_id_27.html (дата обращения: 09.05.2018).

[4]. Formaggia L., Lamponi D., Quarteroni A. One dimensional models for blood flow in arteries. Journal of Engineering Mathematics, vol. 47, 2003, pp. 251-276. DOI: 10.1023/B:ENGI.0000007980.01347.29.

[5]. Azer K., Peskin C. S. A one-dimensional model of blood flow in arteries with friction and convection based on the Womersley velocity profile. Cardiovasc. Eng., vol. 7, 2007, pp. 51–73. DOI: 10.1007/s10558-007-9031-y

[6]. Sherwin S. J., Franke V., Peiro J., Parker K. One-dimensional modeling of vascular network in space-time variables. Journal of Engineering Mathematics. vol. 47, 2003, pp. 217–250. DOI: 10.1023/B:ENGI.0000007979.32871.e2

[7]. Wang X., Delestre O., Fullana J.-M., Saito M., Ikenaga Y., Matsukawa M., Lagree P.-Y. Comparing different numerical methods for solving arterial 1D flows in networks. Comput. Methods Appl. Mech. Eng., vol. 15, 2012, pp. 61–62. DOI: 10.1080/10255842.2012.713677

[8]. Wang X., Fullana J.-M., Lagrée P.-Y. Verification and comparison of four numerical schemes for a 1D viscoelastic blood flow model. Computer Methods in Biomechanics and Biomedical Engineering, 2014, pp.1-22. DOI: 10.1080/10255842.2014.948428

[9]. Avetisyan A.I., Babkova V.V., Gaisaryan S.S., Gubar' A.Yu. Development of parallel software for 3D tornado arising modeling by Nikolaevskiy theory. *Matematicheskoe modelirovanie* [*Mathematical Models and Computer Simulations*], vol. 20, № 8, 2008, pp. 28–40 (in Russian)

[10]. Marchevskii I.K., Tokareva S.A. Comparison of the parallel algorithms effectiveness for solving gas dynamics problems on different computational complexes. *Vestnik MGTU im. N.E. Baumana. Ser. Estestvennye nauki* [*Herald of the Bauman Moscow State Technical University. Series Natural Sciences*], № 1, 2009, pp. 90–97 (in Russian)

[11]. Marchevskii I.K., Shcheglov G.A. Application of parallel algorithms for solving hydrodynamic problems by the vortex element method. *Vychislitel'nye metody i programmirovanie* [*Computational methods and programming*], vol. 11, 2010, pp. 105–110 (in Russian)

[12]. Moreva V.S. Ways for calculation speed-up in solving 2D aerodynamics problems by vortex element method. *Vestnik MGTU im. N.E. Baumana. Ser. Estestvennye nauki* [*Herald of the Bauman Moscow State Technical University. Series Natural Sciences*], № S, 2011, pp. 83–95 (in Russian)

[13]. Lukin V.V., Marchevskii I.K., Moreva V.S., Popov A.Yu, Shapovalov K.L., Shcheglov G.A. Educational-experimental computing cluster. Part 2. Examples of problem solving. *Vestnik MGTU im. N.E. Baumana. Ser. Estestvennye nauki* [*Herald of the Bauman Moscow State Technical University. Series Natural Sciences*], № 4, 2012, pp. 82–102 (in Russian)

[14]. Marchevsky I.K., Puzikova V.V. Efficiency investigation of computation parallelization for viscous incompressible flow simulation on systems with shared memory.

Vychislitel'nye metody i programmirovanie [*Computational methods and programming*], vol. 16, 2015, pp. 595–606 (in Russian)

[15]. Puzikova V.V. Realization of parallel computations in the software package «LS-STAG_turb» for viscous incompressible flow simulation on systems with shared memory. *Trudy ISP RAN / Proc. ISP RAS*, vol. 28, issue 1. 2016, pp. 221-242. DOI: 10.15514/ISPRAS-2016-28(1)-13 (in Russian)

[16]. MPICH Overview | MPICH. URL: https://www.mpich.org/about/overview/ (accessed: 09.05.2018).

[17]. Avetisyan A.I., Gaisaryan S.S., Ivannikov V.P., Padaryan V.A. Productivity prediction of MPI programs based on models. Autom. Remote Control., vol. 68, 2007, pp. 750-759. DOI: 10.1134/S0005117907050037

[18]. Intel (R) Cillk (TM) Plus | Intel® Software. URL: https://software.intel.com/ru-ru/node/522579 (accessed: 09.05.2018).

[19]. Reinders J. Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. Sebastopol: O'Reilly, 2007, 336 p.

[20]. OpenMP FAQ – OpenMP. URL: https://www.openmp.org/about/openmp-faq/ (дата обращения: 09.05.2018).

[21]. Quarteroni A., Valli A. Domain decomposition methods for partial differential equations. Oxford: Clarendon Press, 1999, 360 p.

[22]. Godunov S.K., Zabrodin A.V., Ivanov M.Ya., Kraiko A.N., Prokopov G.P. Numerical solution of gas dynamics multidimensional problems. *Moscow: Science Publ.*, 1976. 400 p. (in Russian)

[23]. Drongowski P., Lei Yu, Swehosky F., Suthikulpanit S., Richter R., Incorporating Instruction-Based Sampling into AMD CodeAnalyst. 2010 IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS 2010), White Plains, NY, 2010, pp. 119-120. DOI: 10.1109/ISPASS.2010.5452049.

[24]. Gergel' V.P. High-performance computing for multi-core systems. *Moscow: Moscow University* Publ., 2010, 544 p. (in Russian)

[25]. Lukin V.V., Marchevskii I.K. Educational-experimental computing cluster. Part 1. Tools and capabilities. Vestnik MGTU im. N.E. Baumana. Ser. Estestvennye nauki [Herald of the Bauman Moscow State Technical University. Series Natural Sciences], № 4, 2011, pp. 28–43 (in Russian)