ТРУДЫ

ИНСТИТУТА СИСТЕМНОГО ПРОГРАММИРОВАНИЯ РАН

PROCEEDINGS OF THE INSTITUTE FOR SYSTEM PROGRAMMING OF THE RAS

ISSN Print 2079-8156
Том 31 Выпуск 6

ISSN Online 2220-6426
Volume 31 Issue 6

Институт системного
программирования
им. В.П. Иванникова РАН

Москва, 2019

ИСП РАН

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

Редколлегия

Главный редактор - Аветисян Арутюн Ишханович, академик РАН, доктор физико-математических наук, профессор, ИСП РАН (Москва, Российская Федерация)

Заместитель главного редактора - Кузнецов Сергей Дмитриевич, д.т.н., профессор, ИСП РАН (Москва, Российская Федерация)

Члены редколлегии

Воронков Андрей Анатольевич, доктор физико-математических наук, профессор, Университет Манчестера (Манчестер, Великобритания)
Вирбицкайте Ирина Бонавентуровна, профессор, доктор физико-математических наук, Институт систем информатики им. академика А.П. Ершова СО РАН (Новосибирск, Россия)
Коннов Игорь Владимирович, кандидат физико-математических наук, Технический университет Вены (Вена, Австрия)
Ластовецкий Алексей Леонидович, доктор физико-математических наук, профессор, Университет Дублина (Дублин, Ирландия)
Ломазова Ирина Александровна, доктор физико-математических наук, профессор, Национальный исследовательский университет «Высшая школа экономики» (Москва, Российская Федерация)
Новиков Борис Асенович, доктор физико-математических наук, профессор, Санкт-Петербургский государственный университет (Санкт-Петербург, Россия)
Петренко Александр Федорович, доктор наук, Исследовательский институт Монреаля (Монреаль, Канада)
Черных Андрей, доктор физико-математических наук, профессор, Научно-исследовательский центр CICESE (Энсенада, Баха Калифорния, Мексика)
Шустер Ассаф, доктор физико-математических наук, профессор, Технион — Израильский технологический институтTechnion (Хайфа, Израиль)

Editorial Board

Editor-in-Chief - Arutyun I. Avetisyan, Academician of RAS, Dr. Sci. (Phys.–Math.), Professor, Ivannikov Institute for System Programming of the RAS (Moscow, Russian Federation)

Deputy Editor-in-Chief - Sergey D. Kuznetsov, Dr. Sci. (Eng.), Professor, Ivannikov Institute for System Programming of the RAS (Moscow, Russian Federation)

Editorial Members

Igor Konnov, PhD (Phys.–Math.), Vienna University of Technology (Vienna, Austria)
Alexey Lastovetsky, Dr. Sci. (Phys.–Math.), Professor, UCD School of Computer Science and Informatics (Dublin, Ireland)
Irina A. Lomazova, Dr. Sci. (Phys.–Math.), Professor, National Research University Higher School of Economics (Moscow, Russian Federation)
Boris A. Novikov, Dr. Sci. (Phys.–Math.), Professor, St. Petersburg University (St. Petersburg, Russian Federation)
Alexandre F. Petrenko, PhD, Computer Research Institute of Montreal (Montreal, Canada)
Assaf Schuster, Ph.D., Professor, Technion - Israel Institute of Technology (Haifa, Israel)
Andrei Tchernykh, Dr. Sci., Professor, CICESE Research Centre (Ensenada, Baja California, Mexico).
Irina B. Virbitskaite, Dr. Sci. (Phys.–Math.), The A.P. Ershov Institute of Informatics Systems, Siberian Branch of the RAS (Novosibirsk, Russian Federation)
Andrey Voronkov, Dr. Sci. (Phys.–Math.), Professor, University of Manchester (Manchester, United Kingdom)

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

© Институт Системного Программирования им. В.П. Иванникова РАН, 2019

Содержание

Table of Contents

DOI: 10.15514/ISPRAS-2019-31(6)-1

CC BY

Анализ корректности работы с памятью с использованием расширения теории символьных графов памяти предикатами над символьными значениями

А.А. Васильев, ORCID: 0000-0002-5738-9171 <vasilyev@ispras.ru>
В.С. Мутилин, ORCID: 0000-0003-3097-8512 <mutilin@ispras.ru>

Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В работе мы рассмотрим подход статической верификации исходного кода программы на предмет корректной работы с памятью. Метод основывается на использовании символьных графов для представления памяти программы. В работе представлено расширение символьных графов памяти, позволяющее использовать предикаты над символьными значениями для повышения точности анализа. Предикаты позволяют отсекать недостижимые пути, уменьшая количество ложных сообщений об ошибках, а также находить новые ошибки за счет добавления новых проверок на символьных значениях. Метод реализован на основе инструмента CPAchecker. Практическая полезность продемонстрирована на драйверах ядра операционной системы Linux.

Ключевые слова: символьные графы памяти; верификация; модель памяти; предикатные абстракции; динамические структуры данных

Для цитирования: Васильев А.А, Мутилин В.С. Анализ корректности работы с памятью с использованием расширения теории символьных графов памяти предикатами над символьными значениями. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 7–20. DOI: 10.15514/ISPRAS–2019–31(6)–1

Благодарности: Работа поддержана грантом РФФИ 18-01-00426

Predicate extension of symbolic memory graphs for analysis of memory safety correctness

A.A. Vasilyev, ORCID: 0000-0002-5738-9171 <vasilyev@ispras.ru>
V.S. Mutilin, ORCID: 0000-0003-3097-8512 <mutilin@ispras.ru>

Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia.

Abstract. Safety-critical systems require additional effort to comply with specifications. One of the required specification is correct memory usage. The article describes an efficient method for static verification against memory safety errors as a combination of Symbolic Memory Graphs and predicate abstraction on symbolic values used in graph. In this article, we introduce an extension of Symbolic Memory Graphs. In addition to symbolic values, the graph stores predicates over symbolic values, which allow to track the relationship between symbolic values in the graph. We also expand existing vertex types to support arbitrary abstract regions, which allow us to represent such dynamic data structures as lists and trees. One of the types of abstract regions is also the ODM region, which presents a special kind of on-demand memory that occurs when analyzing incomplete programs. For this memory, the size and structure of the contents are not known in advance, but it is believed that such memory can be operated safely. The method is implemented in

CPAchecker tool. Practical usage is demonstrated on Linux kernel modules. The practical contribution of our work is to reduce false error messages by constructing more accurate abstractions using predicates over symbolic values.

Keywords: symbolic memory graphs; formal verification; predicate abstraction

For citation: Vasilyev A.A., Mutilin V.S. Predicate extension of symbolic memory graphs for analysis of memory safety correctness. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 7-20 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-1

Acknowledgements. The research was supported by RFBR grant 18-01-00426.

1. Введение

Критические ошибки безопасности и непредвиденные прекращения работы программы часто происходят из-за неправильной работы с памятью. Подобные ошибки являются сложными для обнаружения и устранения, так как время и место появления видимых последствий ошибки может быть достаточно далеко от времени и места выполнения ошибочного действия.

Существуют подходы к проектированию и созданию критического программного обеспечения, предотвращающие возникновение определенных видов ошибок за счет активного использования формальных методов на всех этапах разработки. Примером успешно реализованных проектов с формальной верификацией требований являются микроядро seL4 [1] и оптимизирующий компилятор CompCert [2] языка CLight. Но зачастую подобные методы подразумевают создание программного обеспечения с нуля и тяжело адаптируются к анализу существующего кода.

Статическая верификация исходного кода является одним из подходов для доказательства отсутствия ошибок в программах. Для этого применяются такие методы как анализ потоков данных (data-flow analysis), методы ограничиваемой верификации моделей (Bounded Model Checking, BMC) [3], k-индукция [4], метод уточнения абстракций по контрпримерам (Counter-example guided abstraction refinement, CEGAR) [5]. Наша работа основывается на последнем методе и реализующем его инструменте CPAchecker [6].

Инструмент основывается на адаптивном статическом анализе CPA [7] (Configurable Program Analysis), который представляется единообразно в виде домена и набора операторов, таких как оператор перехода, останова и слияния.

На сегодняшний день в инструменте CPAchecker представлен широкий набор CPA, например, имеется предикатный анализ [8] и анализ явных значений [9]. Унифицированное представление анализов позволяет гибко комбинировать CPA.

Одним из ключевых факторов для эффективного процесса верификации является выбор модели для представления памяти программы.

В предикатных моделях памяти [10] используется предикатная логика и, соответственно, память программы описывается с помощью логических формул, в которых могут использоваться различные теории, например, теория массивов или теория неинтерпретируемых функций [11,12,13].

Данные модели реализованы, например, в инструментах BLAST [14,15], SLAM [16], CPAchecker [8]. Границы применимости задаются решателями логических формул, которые сильно варьируются в зависимости от используемых теорий. Методы предикатных абстракций позволяют описывать структуры данных ограниченного размера, в том числе накладывать ограничения на хранящиеся в них значения, но не позволяют описывать динамические неограниченные структуры данных.

Существуют также методы, в которых динамические структуры данных представляются в структурах троичной логики (three-valued) [17]. Например, подход ленивого анализа связей (Lazy Shape Analysis) [18] реализован в инструменте BLAST.

Еще один класс моделей памяти основывается на логике разделения, являющейся расширением логики Хоара с возможностью локальных рассуждений за счет наличия в утверждениях пространственных связок [19]. Для целей верификации используются разрешимые подмножества логики разделения, но вводимые ограничения позволяют проверять только специальные классы программ (например, без использования массивов). Примеры инструментов, использующие логику разделения: SLAyer [20], VeriFast [21], SpaceInvader [22] и INFER [23]. Методы разделяемой логики позволяют описывать взаимосвязи для неограниченных данных, но для этого используются фрагменты логики, которые могут не иметь решения или иметь крайне неэффективное решение.

Наиболее перспективным [24] для верификации ошибок использования памяти является подход, основанный на символьных графах памяти (Symbolic Memory Graph, SMG) [25]. SMG – это ориентированный граф, представляющий состояние программы. Узлы этого графа хранят символьные значения и объекты. Объекты подразделяются на регионы памяти и абстракции структур данных. Дуги показывают взаимосвязи между узлами и делятся на ребра-указатели и ребра-значения. Каждая дуга и узел в SMG имеют атрибуты, представляющие размер, смещение, состояние выделения памяти.

SMG позволяет эффективно описывать ограниченные структуры данных, используя символьные имена и описание связей между ними в виде графов. Для борьбы с неограниченным ростом используются методы абстракции, с помощью которых похожие элементы структуры обобщаются, тем самым неограниченные данные представляются в виде конечного символьного графа.

В данной статье мы представим расширение SMG.

- В дополнение к символьным значениям граф будет хранить предикаты над символьными значениями, которые позволят отслеживать соотношения между символьными значениями в графе.
- Мы расширим существующие типы вершин для поддержки произвольных абстрактных регионов, что позволит представлять такие динамические структуры данных как списки и деревья. Одним из видов абстрактных регионов также является регион ODM, представляющий специального вида память по требованию (On-Demand Memory) [26], которая возникает при анализе неполных программ. Для этой памяти заранее неизвестен размер и структура содержимого, однако считается, что с такой памятью можно работать безопасно.

В следующем разделе мы дадим определение символьного графа памяти с нашими расширениями. В подразделе 2.1 определяются основные команды, позволяющие работать с SMG. В разд. 3 приводится пример интерпретации выполнения программы. Определение адаптивного статического анализа дается в разд. 4, а в разд. 5 определяется адаптивный статический анализ на основе символьных графов памяти (SMGCPA), работающий с расширенным SMG, что позволяет повысить точность анализа, используя предикаты для отсечения невыполнимых путей в операторе перехода, и поддержать неограниченные динамические структуры данных. Приводится пример работы анализа.

В разд. 6 мы представим экспериментальные данные на ядре ОС Linux. Практическим вкладом работы является сокращение ложных сообщений об ошибках за счет построения более точных абстракций, использующих предикаты над символьными значениями.

2. Символьные графы памяти

Наше определение *символьного графа памяти SMG* $\mathrm{G} = (\mathrm{O}, \mathrm{V}, \Lambda, \mathrm{H}, \mathrm{P}, \Phi, \Pi, \mathrm{E})$ опирается на определение из работы [25], со следующими отличиями:

- во множестве вершин O, вместо конкретного региона для двусвязного списка dls используется понятие абстрактного региона, которое в свою очередь может иметь тип

других структур данных, таких как деревья, а также специального вида памяти ODM;

- множества символьных значений V, функций аннотаций Λ, ребер-значений H, ребер-указателей P определяются аналогично [25];
- новый элемент Φ задает стек вызовов функций;
- Π задает новое множество предикатов над символьными значениями;
- E хранит известные явные значения для символьных значений V.

Перед формальным определением SMG обозначим множество предикатов над символьными значениями V в теории линейных неравенств над битовыми векторами как $\Xi(V)$ [10]. Для решения формул, построенных на основе данных предикатов, используются SMT решатели в соответствующих теориях.

Формально $G = (O, V, \Lambda, H, P, \Phi, \Pi, E)$ определяется следующим образом.

- O – конечное множество объектов, которые включают в себя регионы памяти R и абстракции регионов памяти A, например для сегментов списков. В регионах выделен нулевой регион $R_{NIL} \in R$, являющийся эквивалентом памяти по нулевому адресу.
- V – конечное множество символьных значений с выделенным нулевым значением NIL и неопределенным значением $undef$.
- $\Lambda = \langle kind, size, valid \rangle$ – кортеж функций-аннотаций:
  - тип объекта $kind(o): O \to K = \{region, abstract\}$;
  - размер объектов или символьных значений $size(o): O \cup V \to N$;
  - валидность объекта $valid(o): O \to B$, для нулевого региона $valid(R_{NIL}) = false$.
- H – частичное отображение $O \times Z \times N \rightharpoonup V$, задающее ребра-значения $valueEdge(obj, offs, v, sz)$ из объекта $obj \in O$ по смещению $offs \in Z$ с размером значения $sz \in N$ в символьное значение $v \in V$.
- P – частичное отображение $V \rightharpoonup Z \times O$, задающее ребра-указатели $pointsToEdge(v, offs, obj)$ из значения $v \in V$ в объект $obj \in O$ по смещению $offs \in Z$.
- Стекфрейм Φ – кортеж идентификатора функции, ее аргументов, локальных переменных и памяти на стеке, регионов памяти. В стеке вызовов хранится специальный нулевой фрейм, в котором хранятся глобальные переменные.
- Предикаты над символьными значениями $\Pi \subseteq \Xi$.
- Множество точных значений для символьных значений $E: V \rightharpoonup Z$.

Для удобства записи определим вспомогательные функции над дугами H и P.

- Объект, из которого ведет ребро $h \in H$, обозначим $object(h): H \to O$.
- Символьное значение, в которое ведет ребро $h \in H$, обозначим $value(h): H \to V$.
- Смещение относительно объекта, из которого ведет ребро $h \in H$, обозначим $offset(h): H \to N$.
- Размер значения, записанного на ребре $h \in H$, обозначим $size(h): H \to N$.
- Символьное значение, из которого ведет ребро $p \in P$, обозначим $value(p): P \to V$.
- Объект, в которое ведет ребро $p \in P$, обозначим $object(p): P \to O$.
- Смещение относительно объекта, в который ведет ребро $p \in P$, обозначим $offset(p): P \to Z$.

2.1 Операции над SMG

В этом разделе для символьного графа мы определим операции, причем для каждой операции определим как действие этой операции, так и предусловие этой операции, в котором осуществляется проверка корректности обращения к памяти.

Мы будем рассматривать следующие классы ошибок (в круглых скобках указан соответствующий идентификатор уязвимости по базе MITRE):

- BUF_OVERRUN – чтение/запись за границами буфера (CWE-119, CWE-120, CWE-121, CWE-122, CWE-124, CWE-125, CWE-126, CWE-127, CWE-129, CWE-787);
- NIL_DEREF – обращение по нулевому указателю (CWE-476, CWE-690);
- USE_AFTER_FREE – использование памяти после освобождения (CWE-416);
- UNALLOC_FREE – освобождение ранее не выделенной памяти (CWE-590, CWE-761);
- DOUBLE_FREE – повторное освобождение памяти (CWE-415);
- MEM_LEAK – утечки памяти (CWE-401).

В процессе верификации в SMG могут появляться абстрактные регионы. В дальнейшем при определении SMGCPA мы рассмотрим их построение более детально, например, как результат абстрагирования над несколькими графами SMG или интерпретации специальной функции.

Сейчас для определения операций нам потребуется вспомогательная операция материализации $materialize: SMG \times A \to 2^{SMG \times R}$ обратная к абстрагированию.

Будем писать $materialize(G, a)$, где a – это абстрактный регион, возвращает множество пар (G', r), где G' – это граф, в котором абстрактный регион a заменен на реализацию конкретным регионом r.

Пример ее задания для двусвязного списка можно найти в работе [25].

Для символьного графа памяти $G = (O, V, \Lambda, H, P, \Phi, \Pi, E)$ определены следующие операции:

- Операция чтения $read(G, obj, offs, sz)$ значения размером $sz \in N$ из объекта $obj \in O$ по смещению $offs \in Z$, возвращающая значение $v \in V$ и потенциально новый граф G'.
  - Если $kind(obj) = region$, то происходит чтение без изменения графа $G = G'$. Формально, мы возвращаем: $v = value(h)$, если существует такое $h \in H$, что $object(h) = obj \wedge offset(h) = offs \wedge size(h) = sz$, иначе $v = undef$. Заметим, что может существовать только единственное ребро h для заданных $sz, offs, obj$, так как H это отображение.
  - Если $kind(obj) = abstract$, т.е. это абстрактный регион $obj \in A$, то перед чтением происходит его материализация $(G', r) = materialize(G, obj)$ и только затем чтение $read(G', r, offs, sz)$. При этом возвращается граф G'.

  До чтения проверяется валидность объекта $valid(obj)$, отвечающая за ошибки NIL_DEREF и USE_AFTER_FREE, а также проверяется выход за границы объекта BUF_OVERRUN: $offs \geq 0 \wedge offs + sz \leq size(obj)$. Если имеется явное значение для $offs$, т.е. определено $E(offs)$, то выражение вычисляется явно. Иначе, строится формула, являющаяся конкатенацией множества предикатов Π и отрицания условия выхода за границы. Тем самым, наличие решения построенной формулы, говорит о существовании состояния программы, в котором возможен выход за границы BUF_OVERRUN.

  Заметим, что ребро по условию чтения может не существовать, но при этом чтение завершаться без ошибок.

- Операция записи $write(G, obj, offs, v, sz)$ значения $v \in V$ с размером $sz \in N$ в объект $obj \in O$ по смещению $offs \in Z$, возвращающая новый граф

$G' = (O', V', \Lambda', H', P', \Phi', \Pi', E')$.

- Если $kind(obj) = region$, т.е. это регион $obj \in R$, то из множества ребер-значений удаляются все пересекающиеся с добавляемым ребром: $H_1 = H \setminus \{h \in H \mid object(h) = obj \wedge (offs \leq offset(h) \leq offs + sz \vee offs \leq size(h) + offset(h) \leq offs + sz)\}$; добавляется ребро-значение $h_v = (obj, offs, v, sz)$, в итоге $H' = H_1 \cup h_v$; символьное значение добавляется в множество $V' = V \cup \{v\}$; остальные компоненты не меняются $O' = O, \Lambda' = \Lambda, P' = P, \Phi' = \Phi, \Pi' = \Pi, E' = E$.
- Если $kind(obj) = abstract$, т.е. это абстрактный регион $obj \in A$, то перед записью происходит его материализация $(G', r) = materialize(G, obj)$ и затем запись $write(G', r, offs, sz)$.

Аналогично операции чтения, до записи проверяется валидность объекта $valid(obj)$, отвечающая за ошибки NIL_DEREF и USE_AFTER_FREE, а также проверяется выход за границы объекта BUF_OVERRUN: $offs \geq 0 \wedge offs + sz \leq size(obj)$.

- Операция создания объекта $alloc(G, sz)$, возвращающая новый регион r и SMG $G' = (O', V', \Lambda', H', P', \Phi', \Pi', E')$ с новым регионом $O' = O \cup \{r\}$, Λ' доопределяется на новом объекте так, что $kind(r) = region$, $valid(r) = true$, $size(r) = sz$. Остальные компоненты не изменяются: $V' = V, H' = H, P' = P, \Phi' = \Phi, \Pi' = \Pi, E' = E$.
- Операция освобождения объекта $free(G, obj, offs)$, возвращающая новый SMG $G' = (O', V', \Lambda', H', P', \Phi', \Pi', E')$, где в Λ' изменяется значение $valid(obj) = false$. Остальные компоненты не изменяются: $O' = O, V' = V, H' = H, P' = P, \Phi' = \Phi, \Pi' = \Pi, E' = E$.

  Перед этим проверяется валидность объекта $valid(o)$ для информирования об ошибке DOUBLE_FREE и $offs = 0$ для UNALLOC_FREE.
- Операция добавления переменной $addVariable(G, \phi, name)$ создает новый регион размером соответствующим типу переменной $r = addRegion\big(sizeof(type)\big)$, далее в стекфрейм функции добавляется соответствие имени переменной и этого региона $\phi' = \phi \cup \{name, r\}$.
- Операция удаление переменной $delVariable(G, \phi, name)$, обратная к добавлению, удаляет соответствие имени региону из стекфрейма $\phi' = \phi \setminus \{name, r\}$.
- Добавление стекфрейма функции $addStackFrame(G, function, ret, arg)$ добавляет локальные переменные функции и выделяет память на стеке этой функции.
- Удаление стекфрейма функции $dropStackFrame$. Удаляет локальные переменные функции и память, выделенную на стеке этой функции. После этого проверяется достижимость всех валидных объектов по ребрам-значениям из значений у объектов на стеке для информирования об утечках памяти MEM_LEAK.
- $getExprAddress(G, lval)$ рекурсивно разбирает выражение $lval$ и возвращает $(obj, offs)$ – объект и смещение в объекте.
- $getExprValue(G, rval)$ рекурсивно разбирает выражение $rval$, по необходимости добавляет символьные значения в граф G и в результате возвращает (G', v, e, sz) – измененный граф G', символьное значение, явное значение и размер.

Аналогично определяются операции записи и чтения указателя $writePointer(G, v, obj, offs)$/$readPointer(G, v)$; добавление/взятие явного значения $addExplicit(G, v, e)$/$getExplicit(G, v)$; добавление/удаление предикатов $addPredicate\big(G, pr(v_i)\big)$/$delPredicates(G, v)$; добавления региона $addRegion(G, sz)$; удаления объекта $delObject(G, label)$.

2.2 Конкретизация SMG

Для определения отображения графа в конкретные образы памяти нам потребуется граф без абстрактных регионов. Для SMG $G = (O, V, \Lambda, H, P, \Phi, \Pi, E)$ с множеством абстрактных регионов памяти A в O обозначим за порождаемое им множество *графов памяти* $MG(G)$ все графы с точностью до изоморфизма, которые можно получить из G посредством материализации всех абстрактных регионов памяти, т.е. $\{G' \mid a \in A: (G', r) = materialize(G, a)\}$.

Теперь определим *конкретные образы памяти* $MI(G')$ для графов $G' = MG(G)$, которые затем будут использоваться при конкретизации абстрактного домена CPA (см. разд. 5).

Конкретные образы памяти $MI(G)$ графа памяти $G = (O, V, \Lambda, H, P, \Phi, \Pi, E)$ – множество отображений графа памяти на модель физической памяти компьютера в виде массива ячеек памяти с натуральными адресами N – $\mu: O \to N$ (заметим, что $O = R$), и значениями $val: N \times N \to N$, т.е. $val(addr, size)$ – значение, записанное в памяти начиная с $addr$ по $addr + size$, удовлетворяющее следующим свойствам.

- По нулевому адресу расположен только нулевой регион памяти, т.е. $\forall r \in R: \mu(r) = 0 \Leftrightarrow r = R_{NIL}$.
- Валидные регионы памяти не пересекаются, $\forall r_1, r_2 \in R: valid(r_1) \wedge valid(r_2) \Rightarrow \big(\mu(r_1), \mu(r_1) + size(r_1)\big) \cap \big(\mu(r_2), \mu(r_2) + size(r_2)\big) = \emptyset$.
- Указатели имеют значения адресов, по которым размещаются соответствующие регионы с учетом смещения, для каждой пары ребро-значение и ребро-указатель. Формально, для всех $h \in H, p \in P: val(h) = val(p)$ (связных друг с другом) выполнено $val\Big(\mu\big(object(h)\big) + offset(h), size(h)\Big) = \mu\big(object(p)\big) + offset(p)$.
- Поля, имеющие одинаковые значения, имеют одинаковые конкретные значения, т.е. для двух ребер-значений $h_1, h_2 \in H$ выполнено $val\Big(\mu\big(object(h_1)\big) + offset(h_1), size(h_1)\Big) = val\Big(\mu\big(object(h_2)\big) + offset(h_2), size(h_2)\Big)$.
- Поля, имеющие нулевые значения, заполнены нулями, т.е. для всех ребер-значений $h \in H: value(h) = NIL$ выполнено $val\Big(\mu\big(object(h)\big) + offset(h), size(h)\Big) = 0$.
- Значения, хранящиеся в E совпадают со значениями памяти.
- Значения, хранящиеся в памяти, удовлетворяют предикатам Π.

3. Пример интерпретации выполнения программы

Подробно рассмотрим операции над графом SMG на примере простой программы (Листинг 1).

```
void f(void) {
    int * ar;
    uint ind;
    ar = malloc(SIZE);
    ind = random();
    assume (ind < SIZE)
    ar[ind] = 1;
    free(ar);
}
```

Листинг 1. Пример программы
Listing 1. Sample program

Подобная программа разворачивается в последовательность операций, показанных в табл. 1.

Табл. 1. Развертывание программы в последовательность операций SMG
Table 1. Transforming a program into a sequence of operations of SMG

Вход в функцию f	
`f(void)`	$G_1 = addStackFrame(G, "f", void)$
Декларация переменных	
`int * ar;`	$G_2 = addVariable(G_1, "ar", sizeof(int*))$
`uint ind;`	$G_3 = addVariable(G_2, "ind", sizeof(uint))$
Вызов функции аллокации и запись значения в переменную	
`ar = malloc(SIZE);`	$\{G_4, r_{alloc}\} = alloc(G_3, SIZE)$ $G_5 = writePointer(G_4, getObject("ar"), 0, r_{alloc}, 0)$
Вызов функции random и запись значения в переменную	
`ind = random();`	$v_{ind} = read(G_5, getObject("random"), 0, sizeof(uint))$ $G_6 = write(G_5, getObject("ind"), 0, v_{ind}, sizeof(uint))$
`assume (ind < len);`	$v_{ind} = read(G_6, getObject("ind"), 0, sizeof(uint))$ $v_{len} = read(G_6, getObject("len"), 0, sizeof(uint))$ $G_7 = addPredicate(G_6, E(v_{ind}) < E(v_{len}))$
`ar[ind] = 1;`	$v_{ind} = read(G_7, getObject("ind"), 0, sizeof(uint))$ $v_{ar} = read(G_7, getObject("ar"), 0, sizeof(int*))$ $r_{alloc} = readPointer(G_7, v_{ar})$ $G_8 = write(G_7, E(1), sizeof(int), r_{alloc}, v_{ind} * sizeof(int))$
`free(ar);`	$v_{ar} = read(G_8, getObject("ar"), 0, sizeof(int*))$ $r_{alloc} = readPointer(G_8, v_{ar})$ $G_9 = free(G_8, r_{alloc})$
Выход из функции f	
	$G_{10} = dropStackFrame(G_9, "f")$

4. Адаптивный статический анализ

Адаптивный статический анализ реализован в инструменте CPAchecker и подробно описан в статье [7], где приводится следующее формальное описание:

Адаптивный статический анализ $D = (D, \Pi, merge, stop, prec, \rightsquigarrow)$ состоит из абстрактного домена D, множества точностей Π, оператора объединения абстрактных состояний $merge$, оператора проверки остановки анализа $stop$, функции изменения точности анализа $prec$ и оператора перехода $\rightsquigarrow$.

1. *Абстрактный домен* $D = (\mathcal{C}, \mathcal{E}, [\![\cdot]\!])$, где $\mathcal{C}$ – множество конкретных состояний; $\mathcal{E} = (E, \top, \bot, \sqsubseteq, \sqcup)$ – решетка, состоящая из множества абстрактных состояний E, частичного порядка $\sqsubseteq \subseteq E \times E$, верхнего элемента $\top \in E$, нижнего элемента $\bot \in E$ и оператора объединения состояний $\sqcup : E \times E \to E$; функция конкретизации $[\![\cdot]\!] : E \to 2^{\mathcal{C}}$, задающая значение абстрактного состояния как множество конкретных состояний.
2. *Множество точностей* Π определяет возможные точности абстрактного домена.

3. *Оператор перехода* $\rightsquigarrow \subseteq E \times G \times E \times \Pi$ строит для абстрактного состояния e и ребра графа потока управления g множество новых абстрактных состояний e' с точностями π.
4. *Оператор объединения* $merge: E \times E \times \Pi \to E$ ослабляет второе состояние на основе первого состояния и возвращает новое абстрактное состояние заданной точности.
5. *Оператор останова* $stop: E \times 2^E \times \Pi \to B$ определяет, покрывается ли абстрактное состояние с заданным уровнем точности множеством абстрактных состояний.
6. *Функция изменения точности анализа* $prec: E \times \Pi \times 2^{E \times \Pi} \to E \times \Pi$ вычисляет новое абстрактное состояние и уточнение по заданному абстрактному состоянию с уточнением и множества абстрактных состояний с уточнениями.

4.1 Комбинация адаптивных анализов

Комбинация адаптивных анализов $D_1 = (D_1, \Pi_1, merge_1, stop_1, prec_1, \rightsquigarrow_1)$ и $D_2 = (D_2, \Pi_2, merge_2, stop_2, prec_2, \rightsquigarrow_2)$ также является адаптивным анализом $D = (D_1 \times D_2, \Pi_1 \times \Pi_2, merge_1 \times merge_2, stop_1 \times stop_2, prec_1 \times prec_2, \rightsquigarrow_1 \times \rightsquigarrow_2)$.

В статье [6] показано, что метод верификации, основанный на комбинации адаптивных анализов D, является полным, если отдельные методы, соответствующие анализам D_1 и D_2, являются полными. Подобный подход позволяет использовать существующие статические анализы из проекта CPAchecker.

5. Адаптивный статический анализ на основе символьных графов памяти (SMGCPA)

В концепции адаптивного статического анализа он задается следующим образом.

1. *Абстрактный домен* $D = (\mathcal{C}, \mathcal{E}, [\![\cdot]\!])$, где $\mathcal{C}$ – множество конкретных образов памяти, $\mathcal{E}$ – множество символьных графов памяти SMG и $[\![\cdot]\!] = MI\big(MG(G)\big)$ – соответствие символьного графа памяти G его множеству конкретных образов памяти.
2. *Множество точностей* $\Pi = \emptyset$.
3. *Оператор перехода* $\rightsquigarrow (G, op)\$$, на графе $G = (O, V, \Lambda, H, P, \Phi, \Pi, E)$ и ребре графа потока управления op. Результирующие графы будем обозначать как $G' = (O', V', \Lambda', H', P', \Phi', \Pi', E')$.
   a. Присваивание (assignment) $op = assign(lvalexpr, rvalexpr)$.
      i. Если $lvalexpr$ не является указателем, то:

         Пусть $(obj, offs) = getExprAddress(G, lvalexpr)$,

         $\big(\check{G}, v, e, sz\big) = getExprValue(G, rvalexpr)$,

         $\hat{G} = write\big(\check{G}, obj, offs, v, sz\big)$.

         Тогда результирующее E' совпадает с E за исключением значения на v, где $E'(v) = e$, а остальные компоненты G' берутся из $\hat{G}$.
      ii. Если $lvalexpr$ указатель, то:

         Пусть $(obj, offs) = getExprAddress(G, lvalexpr)$,

         $\big(\check{G}, v, e, sz\big) = getExprValue(G, rvalexpr)$,

         $(obj_1, offs_1) = getExprAddress\big(\check{G}, rvalexpr\big)$,

         $\hat{G} = write\big(\check{G}, obj, offs, v, sz\big)$.

         Тогда $\tilde{G} = writePointer\big(\hat{G}, v, obj_1, offs_1\big)$ и результирующее E' совпадает с E за исключением значения на v, где $E'(v) = e$, а остальные компоненты G' берутся из $\tilde{G}$.

b. Условный переход $op = assume(expr)$ либо $op = if(expr)$ добавляет предикат перехода в П. Проверка существования перехода осуществляется следующим образом.
   i. Если выражение можно вычислить с помощью явных значений и оно ложно, то данный переход отсекается и возвращается состояние ⊥.
   ii. При наличии предикатов над символьными значениями П осуществляются дополнительные проверки. По выражению $expr$ условия вычисляется сильнейшее постусловие. В результате получается формула, которая конкатенируется с предикатами из П. Если формула оказывается неразрешимой, то переход отсекается и возвращается состояние ⊥.

c. Выделение памяти на стеке $op = alloca(expr)$.
   Пусть $(\check{G}, v, e, sz) = getExprValue(G, expr)$.
   Тогда на SMG выполняется $(\tilde{G}, r) = addRegion(\check{G}, e)$,
   а получившийся регион добавляется в стекфрейм текущей функции $\Phi' = \widetilde{\Phi} \cup \{r, funcName\}$.

d. Выделение памяти в куче $op = malloc(expr)$.
   Пусть $(\check{G}, v, e, sz) = getExprValue(G, expr)$.
   Тогда $G' = alloc(\check{G}, e)$.

e. Освобождение памяти (free) $op = free(lvalexpr)$.
   Пусть $(obj, offs) = getExprAddress(G, lvalexpr)$.
   Тогда $G' = free(G, obj, offs)$.

f. Выход из функции $funcName$ (return).
   $G' = dropStackFrame(G, funcName)$.

g. Остальные операторы программы op описываются аналогично.

4. *Оператор объединения* $merge$ соответствует $join$ символьных графов SMG.

5. *Оператор останова* $stop$ – сравнение вложенности символьных графов памяти.

6. *Функция изменения точности анализа* $prec$ не используется.

6. Эксперименты

Описанный в разделе 5 анализ SMGCPA на основе расширенных символьных графов памяти был реализован в инструменте CPAchecker, ревизия 32467[1].

Для экспериментов использовались 3484 задания, подготовленных на основе драйверов ядра ОС Linux версии 4.18-rc5 (см. набор linux-4.18-rc5-memsafety[2]) с помощью системы Klever 2719].

Запуски производились с ограничением процессорного времени исполнения в 60 секунд на процессоре Intel Core i5-4590 в следующих конфигурациях:

- SMGCPA с выключенными предикатами – (стандартный SMG);
- SMGCPA с включенными предикатами – (расширенный SMG).

Результаты приведены в табл. 2. Вердикт *true* означает, что нарушений корректного использования памяти не выявлено. Вердикт *false* означает, что выявлено потенциальное нарушение корректного использования памяти. В скобках указано конкретное нарушение:

- *valid-deref* – BUF_OVERRUN или NIL_DEREF;

[1] https://github.com/mutilin/cpachecker/commit/0f486a996
[2] https://gitlab.com/sosy-lab/software/ldv-benchmarks

- *valid-memtrack* – MEM_LEAK;
- *valid-free* – USE_AFTER_FREE, DOUBLE_FREE или UNALLOC_FREE.

Табл.2. Изменение вердиктов стандартный SMG → расширенный SMG
Table 2. Change verdicts standard SMG → extended SMG

Стандартный SMG	Расширенный SMG	Количество
True	false(valid-deref)	4
false(valid-deref)	True	1
True	TIMEOUT	1
True	false(valid-free)	1
TIMEOUT	false(valid-deref)	19
TIMEOUT	false(valid-free)	1
TIMEOUT	True	5
false(valid-free)	false(valid-deref)	2
false(valid-memtrack)	false(valid-deref)	2
false(valid-memtrack)	TIMEOUT	1
false(valid-deref)	TIMEOUT	6

TIMEOUT означает, что не удалось получить вердикт в заданное время (60 сек.).

Переходы вердиктов в табл. 2 объясняются двумя эффектами:

i. отсечение недопустимых путей (см. описание *оператора перехода* в разд. 5).

ii. уточненная проверка выхода за границы памяти (см. описание операций $read$ и $write$ в подразделе 2.1).

В табл. 2 мы видим, что один переход из *false* в *true* был ложным срабатыванием (i).

Еще 4 перехода из *true* в *false(valid-deref)* были упущенными ошибками (ii). Остальные переходы связаны с тем, что (i) влияет на возможность получения вердикта в заданное время TIMEOUT. Суммарное количество позитивных переходов из TIMEOUT – 25, превышает количество негативных – 7.

Сокращение вердиктов TIMEOUT связано с сокращением количества состояний, построенных в процессе анализа за счет отсечения по условию (i). Увеличение состояний также возможно, так как для состояний с несовместными наборами предикатов не производится слияние в операторе $merge$.

Всего количество состояний уменьшилось в 836 заданиях, увеличилось в 240 и не поменялось в 2354.

При верификации предикаты использовались в 3225 модулях из 3484. С их помощью было отсечено 14163 недостижимых путей, что составляет в среднем 4.39 на задание, а максимально в одном задании отсекается 1271 путь.

Эффект (ii) позволяет обнаружить 25 новых ошибок.

7. Заключение

В работе представлено расширение символьных графов памяти SMG, позволяющее использовать предикаты над символьными значениями для повышения точности анализа.

Во-первых, продемонстрирована возможность отсечения недостижимых путей, уменьшая количество ложных сообщений об ошибках.

Во-вторых, использование новых проверок на основе предикатов над символьными значениями позволяет находить новые ошибки. Так стало возможным проверять выход за границу объекта в операциях чтения и записи SMG не только на явных значениях, но и на символьных.

Практическая реализация выполнена на основе инструмента CPAchecker и проведены эксперименты на драйверах ядра ОС Linux версии 4.18-rc5, по которым получено 3484 заданий.

При верификации предикаты использовались в 3225 заданиях из 3484 с помощью них было отсечено 14163 недостижимых путей и обнаружено 25 новых ошибок.

Список литературы / References

[1]. Klein G., Elphinstone K. et al. sel4: Formal verification of an os kernel. In Proc. of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, 2009, pp. 207–220.

[2]. Stewart G., Beringer L., Cuellar S., Appel A.W. Compositional compcert. In Proc. of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2005, pp. 275–287.

[3]. Beyer D., Keremoglu M.E.: CPAchecker: A tool for configurable software verification. Lecture Notes in Computer Science, vol. 6806, 2011, pp. 184-190.

[4]. Donaldson A.F., Haller L., Kroening D., R¨ummer P. Software verification using k-induction. Lecture Notes in Computer Science, vol. 6887, 2011, pp. 351-368.

[5]. Clarke E., Grumberg O., Jha S., Lu Y., Veith H. Counterexample-guided abstraction refinement. Lecture Notes in Computer Science, vol. 1855, 2000, pp. 154–169.

[6]. Beyer D., Keremoglu, M.E., Wendler, P. Predicate abstraction with adjustable-block encoding. In Proc. of the 10th International Conference on Formal Methods in Computer-Aided Design, 2010, pp. 189–197.

[7]. Beyer D., Henzinger T., Theoduloz G. Program analysis with dynamic precision adjustment. In Proc. of the 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008. pp. 29–38.

[8]. Beyer D., Löwe S. Explicit-state software model checking based on CEGAR and interpolation. Lecture Notes in Computer Science, vol. 7793, 2013, pp. 146–162.

[9]. Biere A., Cimatti A., Clarke E.M., Zhu Y. Symbolic model checking without bdds. Lecture Notes in Computer Science, vol. 1579, 1999, pp. 193–207.

[10]. Graf S., Saidi H. Construction of abstract state graphs with PVS. Lecture Notes in Computer Science, vol. 1254, 1997, pp. 72–83.

[11]. Andrianov P., Friedberger K., Mandrykin M., Mutilin V., Volkov A. CPA-BAM-BnB: Block-abstraction memoization and region-based memory models for predicate abstractions. Lecture Notes in Computer Science, vol. 10206, 2017, pp. 355–359.

[12]. Yang H., Lee O., Berdine J., Calcagno C., Cook B., Distefano D., O'Hearn P. Scalable shape analysis for systems code. Lecture Notes in Computer Science, vol. 5123, 2008, pp. 385–398.

[13]. Volkov A., Mandrykin M. Predicate Abstractions Memory Modeling Method with Separation into Disjoint Regions. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 203-216. DOI: 10.15514/ISPRAS-2017-29(4)-13.

[14]. Beyer D., Henzinger T., Jhala R., Majumdar R. The software model checker BLAST. International Journal on Software Tools for Technology Transfer, vol. 9, issue 5-6, 2007, 505–525.

[15]. Shved P., Mandrykin M., Mutilin V. Predicate analysis with BLAST 2.7. Lecture Notes in Computer Science, vol. 7214, 2012, pp. 525–527.

[16]. Ball T., Bounimova E., Kumar R., Levin V. SLAM2: Static driver verification with under 4% false alarms. In Proc. Of the 10th International Conference on Formal Methods in Computer-Aided Design, 2010. pp. 35–42.

[17]. Sagiv M., Reps T.W., Wilhelm R. Parametric shape analysis via 3-valued logic. ACM Transactions on Programming Languages and Systems, vol. 24, issue 3, 2002, pp. 217–298.

[18]. Beyer D., Henzinger T.A., Théoduloz G. Lazy shape analysis. Lecture Notes in Computer Science, vol. 4144, 2006, pp. 532–546.

[19]. Reynolds J.C. Separation logic: A logic for shared mutable data structures. In Proc. of the 17th Annual IEEE Symposium on Logic in Computer Science, 2002, pp. 55–74.

[20]. Berdine J., Cook B., Ishtiaq S. Slayer Memory safety for systems-level code. Lecture Notes in Computer Science, vol. 6806, 2011, pp. 178-183.

[21]. Jacobs B., Smans J., Piessens F. A quick tour of the verifast program verifier. Lecture Notes in Computer Science, vol. 6461, 2010, pp. 304–311.

[22]. Volkov A., Mandrykin M. Predicate Abstractions Memory Modeling Method with Separation into Disjoint Regions. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 4, 2017, pp. 203-216. DOI: 10.15514/ISPRAS-2017-29(4)-13.

[23]. Calcagno C., Distefano D. et al. Moving fast with software verification. Lecture Notes in Computer Science, vol. 9058, 2015, pp. 3-11.
[24]. Beyer D. Automatic verification of C and Java Programs: SV-COMP 2019. Lecture Notes in Computer Science, vol. 11429, 2019, pp. 133–155.
[25]. Dudka K., Peringer P., Vojnar T.: Byte-precise verification of low-level list manipulation. Lecture Notes in Computer Science, vol. 7935, 2013, pp. 215–237.
[26]. Vasilyev A.A. Static verification for memory safety of Linux kernel drivers. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018. pp. 143-160. DOI: 10.15514/ISPRAS-2018-30(6)-8.
[27]. Novikov E., Zakharov I. Towards automated static verification of GNU C programs. Lecture Notes in Computer Science, vol. 10742, 2018, pp. 402–416.

Информация об авторах / Information about authors

Антон Александрович ВАСИЛЬЕВ – стажер-исследователь, аспирант ИСП РАН. Сфера научных интересов: верификация программ, теория графов, математическая логика.

Anton Aleksandrovich VASILIEV – intern researcher, PhD student of ISP RAS. Research interests: program verification, graph theory, mathematical logic.

Вадим Сергеевич МУТИЛИН – кандидат физико-математических наук, старший научный сотрудник ИСП РАН. Сфера научных интересов: верификация программ, статический анализ, операционные системы.

Vadim Sergeevich MUTILIN – PhD in physical and mathematical sciences, Senior Researcher at ISP RAS. Research interests: program verification, static analysis, operating systems.

CXnT6PTr5DOI: 10.15514/ISPRAS-2019-31(6)-2

CC BY

Формальная модель обнаружения программных ошибок с помощью символьного исполнения программ

А.Ю. Герасимов, ORCID: 0000-0001-9964-5850 <agerasimov@ispras.ru>
Д.О. Куц, ORCID: 0000-0002-0060-8062 <kutz@ispras.ru>
А.А. Новиков, ORCID: 0000-0001-7567-0998 <a.novikov@ispras.ru>

Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Автоматическое обнаружение ошибок в программах является крайне востребованным направлением современных исследований и разработок в области обеспечения безопасности и устойчивости программного обеспечения. В рамках проекта 17-07-00702 Российского фонда фундаментальных исследований исследовались направления применения комбинированных методов анализа программ, совмещающих динамическое символьное исполнение, рандомизированное тестирование и статический анализ программ. Разработаны методы направленного анализа программ, основанные на совмещении статического анализа и динамического символьного исполнения, совмещения рандомизированного тестирования и динамического символьного исполнения программы. В данной статье рассматривается формальная модель обнаружения ошибок в программах методом символьного исполнения программ и её реализация для обнаружения ошибок выхода за границы буфера в памяти. Приводится формальная модель символьного исполнения программ, формулируется и доказывается теорема об обнаружении ошибки в программе, основанная на нарушении области определения операции вычислительной системы. Приводится описание реализации анализатора нарушения границ буфера в памяти в процессе динамического символьного исполнения программы и результаты применения реализованного прототипа анализатора на наборе программ из поставки Debian Linux, подтверждающих применимость предложенного метода обнаружения ошибок.

Ключевые слова: комбинированный анализ программ; динамическое символьное исполнение программ; обнаружение программных ошибок

Для цитирования: Герасимов А.Ю., Куц Д.О., Новиков А.А. Формальная модель обнаружения программных ошибок с помощью символьного исполнения программ. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 21–32. DOI: 10.15514/ISPRAS–2019–31(6)–2

Благодарности: Исследование проведено при поддержке Российского фонда фундаментальных исследований. Проект 17-07-00702.

A formal model for program defect detection using symbolic program execution

A.Y. Gerasimov, ORCID: 0000-0001-9964-5850 <agerasimov@ispras.ru>
D.O. Kutz, ORCID: 0000-0002-0060-8062 <kutz@ispras.ru>
A.A. Novikov, ORCID: 0000-0001-7567-0998 <a.novikov@ispras.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. An automatic program defect detection is extremely important direction of current research and development in the field of program reliability and security assurance. There were performed research of different ways of application for combined analysis methods which mix static source code analysis and dynamic symbolic execution, fuzz testing and dynamic symbolic execution as part of previous period of two years for project 17-07-00702 of the Russian Foundation for Basic Research. This paper presents elaboration of previously presented methods in form of formal model of program symbolic execution applied for program defect detection and implementation of analyzer of memory buffer bounds violation based on this model. The common theorem for program defect detection based on model of symbolic program execution and violation of definitional domain for computation system operation is formulated and proved. A special case theorem for buffer bounds violation detection is formulated and proved basing on common theorem and shadow memory model. As a practical application for theoretical basis an implementation of the analysis tool prototype description provided. Experimental results are received on the set of command line utilities of Debian Linux distribution, which shows applicability of proposed theoretical basis for solving practical tasks in the field of program reliability and security assurance.

Keywords: hybrid program analysis; dynamic symbolic execution; program defect detection

For citation: Gerasimov A.Y., Kutz D.O., Novikov A.A. A formal model for defect detection using symbolic program execution. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 21-32 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-2

Acknowledgements. The research was supported by RFBR, grant 17-07-00702.

1. Введение

Первоочередной задачей для демонстрации ошибки в программе, найденной аналитиком или инструментами автоматической инспекции кода программ, является генерация внешних данных программы, проявляющих её ошибочное поведение. Наиболее популярным методом обнаружения ошибок во времена исполнения программы является метод рандомизированного тестирования [1], который тестирует программу на псевдослучайных внешних данных и проверяет устойчивость программы к неожидаемым внешним данным [2]. Метод рандомизированного тестирования программ обладает высокой производительностью в связи с тем, что наборы внешних данных программы генерируются с высокой скоростью, а программа подвергается легковесной инструментации (в случае рандомизированного тестирования с подкреплением), которая не приводит к значительному замедлению исполнения программы. При этом, обнаружение программных ошибок методом рандомизированного тестирования обладает случайным характером и не может быть использовано для достоверного подтверждения ошибки.

Альтернативным методом генерации тестового набора внешних данных программы является динамическое символьное исполнение [3]. Данный метод основывается на тяжеловесной инструментации кода программы с целью сбора трассы исполнившихся инструкций для последующего построения формулы, описывающей модель исполнения программы. На основе этой формулы вычисляются новые наборы внешних данных программы, которые приводят к исполнению программы по альтернативному пути или воспроизводят имеющийся дефект. В связи с тяжеловесной инструментацией кода

программы и высокой вычислительной сложностью решения формулы данный метод на современном уровне развития вычислительной техники не может быть применен для исчерпывающего тестирования программы.

Наиболее перспективным способом применения метода динамического символьного исполнения является применение направленного анализа программы [4], использующего информацию о положении потенциальных ошибок в программе для целенаправленного исполнения программы по путям, приводящим к местам потенциальных ошибок [5], с последующей проверкой реализации ошибочной ситуации в программе.

В данной статье приводится формальная модель символьного исполнения программы с целью проявления ошибок в программе, а также даётся пример реализации проверки ошибочной ситуации для ошибок выхода за границы буфера в памяти.

2. Модель символьного исполнения программы

Определим программу как четверку:

$$P = < F, S, s_I, S_T >, \#(1)$$

где: S – множество состояний программы; s_I – начальное состояние программы; S_T – множество конечных состояний программы; F – множество операций, каждая из которых переводит программу из одного состояния в другое:

$$f : S \to S, f \in F, \#(2)$$

Тогда исполнение программы можно определить как последовательность переходов между состояниями программы $s_i \in S$, осуществляемых операциями $f_i \in F$:

$$\{f_I(s_I) \to s_1, f_1(s_1) \to s_2, \dots, f_i(s_i) \to s_T\}, \#(3)$$

где: $s_I \in S$ – начальное состояние программы; $s_T \in S_T$ – одно из конечных состояний программы; $f_i \in F$ – операция, принадлежащая множеству операций программы, переводящих одно состояние программы в другое.

Определим состояние программы как

$$s = < d, f > | d \in D, f \in F, \quad (4)$$

где d – подмножество множества данных, обрабатываемых программой; D – множество данных, обрабатываемых программой; f – следующая операция программы.

Тогда исполнение операции программы можно представить в виде

$$f_i(d_i) \to < d_j, f_j > | d_i, d_j \in D, f_i, f_j \in F. \quad (5)$$

Определение 1. *Ограничением пути исполнения в программе PC* (path condition) будем называть множество ограничений на значения данных программы, полученное путём преобразования операций, выполненных над данными программы на пути исполнения, в элементы множества ограничений и однозначно описывающее исполнение программы по пути, достигающем состояния s.

Не ограничивая общности рассуждений, всё множество операций в программе можно разделить на три вида:

$$f_i(d_i) \to < d_j, f' > \left| \begin{cases} d_i \neq d_j, \quad f_i \to f_j & \text{вычислительная операция} \\ d_i = d_j, f_i \to f_j & \text{безусловный переход} \\ d_i = d_j, \begin{cases} f_i \to f_j \mid p_i \\ f_i \to f_k, \mid \neg p_i \end{cases} p_i \in PC & \text{условный переход} \end{cases} \right. \#(6)$$

где:

- вычислительная операция изменяет состояние программы путём изменения множества данных программы $d_i \to d_j$ и переводит исполнение программы на следующую операцию $f_i \to f_j$;

- безусловный переход изменяет состояние программы путём перевода исполнения на следующую операцию $f_i \to f_j$;
- условный переход изменяет состояние программы путём перевода исполнения на операцию f_j, если подмножество предусловий состояния, являющееся условием перехода, вычисляется в истину, и на операцию f_k, если подмножество предусловий состояния, являющееся условием перехода, вычисляется в ложь.

Разделим множество данных программы D, на котором определено ограничение пути исполнения в программе PC, на два множества: V – множество внутренних данных программы; W – множество внешних данных программы. Тогда предусловие состояния s_i можно описать как функцию:

$$p(w, v) \mid w \subseteq W, v \subseteq V. \quad (7)$$

Заменим элементы множества W на элементы множества переменных X, где позиции каждого элемента множества внешних данных в потоке внешних данных программы соответствует переменная $x \in X$.

Определение 2. *Множеством свободных переменных* будем называть множество переменных, значения которых соответствуют значениям элементов множества внешних данных программы.

Определение 3. *Множество зависимых переменных* формируется из переменных, являющихся результатом исполнения операций, множество аргументов которых содержит хотя бы одну свободную или хотя бы одну зависимую переменную.

Определение 4. *Символьное ограничение пути* SPC, или *символьное предусловие* состояния s в программе, является множеством ограничений на значения зависимых и свободных переменных и внутренних данных программы, полученных путём преобразования операций над ними на пути исполнения программы, предшествующем состоянию s.

Множество значений данных программы удовлетворяющих символьному предусловию пути для состояния s_i описывается функцией $\pi(x_i, v_i)$:

$$D_{\pi_i} = \pi(x_i, v_i) \mid D_{\pi_i} \subseteq D. \quad (8)$$

Определение 5. *Состояние ошибки* это такое состояние в программы $s_{err} \in S$, при котором дальнейшее исполнение программы ошибочно.

Определение 6. *Множество определения операции* f – это такое подмножество данных программы $D' \subseteq D$, на котором выполнение операции f не приводит к достижению состояния ошибки.

Утверждение 1. Для каждой операции f, входящей в множество операции программы F, существует множество определения данной операции D':

$$\forall f \in F \; \exists \; D' \subseteq D, \#(9)$$

Утверждение 2. Если множество значений аргументов операции f не входит в множество определения операции D' и множество D' не пусто, то исполнение операции приводит программу в состояние ошибки

$$f(d) \to s_{err} \mid f \in F, \forall \; d \notin D' \; \& \; D' \neq \varnothing, \quad (10)$$

Если множество определения операции пусто, то выполнение операции не зависит от данных программы, что соответствует определению операции на всём множестве данных программы, и, следовательно, либо всякое исполнение операции гарантированно приводит к достижению состояния ошибки в программе s_{err}, либо выполнение данной операции не может привести исполнение программы в состояние ошибки s_{err}.

Теорема 1. Если множество определения D' операции f_i не пусто и дополнение множества D' и множества значений данных программы D_{π_i}, ограниченных функцией

$\pi(x_i, v_i) = D_{\pi_i}$, не пусто, то существуют такие значения внешних данных, исполнение программы на которых приведёт к достижению ошибочного состояния s_{err} в программе:

$$f_i(d_i) \rightarrow s_{err} \mid \forall\, d_i \in D_{err},\ D_{err} = D_{\pi_i}\ D' \,\&\, D' \neq \varnothing. \quad (11)$$

Доказательство. Исходя из формулировки *утверждения 2*, достижение состояния ошибки в программе возможно при выходе значения аргументов операции f_i за пределы множества области определения операции D'. Если символьные ограничения пути исполнения SPC наложенные на значения внешних данных программы, описываемые функцией $\pi(x_i, v_i)$, формируют такое множество D_{π_i}, что значения аргументов f_i попадают в множество допустимых значений данных программы D_{π_i}, но не попадают в множество определения D' операции f_i, то, исходя из *утверждения 2*, программа достигнет состояния ошибки s_{err}, что и требовалось доказать.

Следствие теоремы 1. Для того чтобы вычислить внешние данные программы, приводящие исполнение программы в состояние ошибки, необходимо и достаточно для каждого типа ошибки определить множество операций, исполнение которых может приводить к ошибке, и сформулировать условие выхода значений аргументов операции за пределы множества определения этих операций и при этом входящих в множество значений данных программы.

Определение 7. *Буфер в памяти* – область памяти, ограниченная адресом начала A_{begin} и адресом конца A_{end} блока данных программы в памяти.

Определение 8. Операцией *разыменования при доступе к буферу в памяти* будем называть операцию разыменования указателя, область определения которой D'_{buffer} ограничена адресом начала A_{begin} и адресом конца A_{end} буфера в памяти.

Подсистема управления динамической памятью операционной системы при выделении блока оперативной памяти программе производит выделение таким образом, что каждый новый выделенный блок памяти не пересекается по интервалу адресов начала и конца блока $< A_{begin},\ A_{end} >$ с другими выделенными блоками памяти на момент его выделения. Также менеджер может выделить блок памяти, включающий адреса, попадающие в интервал адресов ранее освобождённых блоков.

Для контроля доступа к выделенным и освобождённым блокам памяти заведём два множества: множество выделенных блоков памяти D_{alloc} и множество освобождённых блоков памяти D_{freed}, каждое из которых в качестве элементов содержит описание блоков памяти в виде интервала адресов $< A_{begin},\ A_{end} >$, где A_{begin} – адрес начала блока памяти, A_{end} – адрес конца блока памяти. При выполнении операции выделения блока в памяти интервал адресов $< A_{begin},\ A_{end} >$ добавляется в множество выделенных блоков памяти D_{alloc}. При выполнении операции освобождения блока в памяти интервал адресов $< A_{begin},\ A_{end} >$ удаляется из множества выделенных блоков памяти D_{alloc} и добавляется в множество освобожденных блоков памяти D_{freed}. Если возникает пересечение адресов нового выделенного блока с уже освобожденными блоками в множестве D_{freed}, то в множестве D_{freed} происходит преобразование интервалов адресов, которые пересекаются с выделенным блоком таким образом, чтобы интервал адресов выделенного блока не пересекался ни с одним из интервалов в множестве освобождённых блоков памяти D_{freed}.

Теорема 2. Для определения ошибки доступа к буферу в памяти по указателю f_{dbuf} необходимо и достаточно в символьное предусловие операции добавить условие проверки значения переменной указателя меньше адреса начала буфера A_{begin} и больше адреса конца буфера A_{end}:

$$f_{dbuf}\,(d) \rightarrow s_{err} \mid (SPC \,\&\, (d < A_{begin} \,||\, d > A_{end})) \neq \varnothing. \quad (12)$$

Доказательство. Множество определения операции доступа к буферу f_{dbuf} по *определению 8* ограничено адресом начала A_{begin} и адресом конца буфера A_{end}. Если символьное предусловие SPC операции разыменования f_{dbuf} позволяет вычисление значений множества внешних данных программы, таких, что аргумент операции разыменования принимает значение меньше адреса начала буфера A_{begin} или больше адреса конца буфера A_{end}, то дополнение разности множеств $(D_{\pi_i} \setminus D'_{buffer})$ не пусто и содержит адрес, выходящий за границы буфера. В соответствии с *определением 8* происходит нарушение множества определения операции разыменования указателя при доступе к буферу, что в соответствии с *теоремой 1* приводит к достижению состояния ошибки в программе s_{err}, что и требовалось доказать.

3. Реализация проверки нарушения предикатов безопасности для ошибок выхода за границы буфера в памяти

Количество программных ошибок, обнаруженных инструментами автоматического анализа программ и связанных с ошибками доступа к памяти, достаточно велико [6]. Данные ошибки связаны как с обращением к памяти за границами выделенных блоков, так и с нарушением прав доступа в пределах выделенных блоков памяти (на чтение, на запись, на исполнение). В случае нарушения прав доступа к выделенным блокам памяти или обращению к адресам за границами страниц памяти выделенных исполняющемуся процессу, происходит ошибка сегментации (segmentation fault) в операционной системе семейства Linux или ошибка нарушения доступа к блоку памяти (access violation) в операционной системе семейства Windows.

Некорректный доступ к памяти является причиной таких ошибок, как разыменование нулевого указателя (CWE-476 [7]), переполнение буфера на стеке или куче (CWE-121 [8], CWE-122 [9]), запись значения по произвольному адресу (CWE-123[10]), запись и чтение за пределами выделенного буфера (CWE-125 [11], CWE-787 [12]). В наборе инструкций архитектуры Intel x86 и x86-64 количество инструкций доступа к памяти достаточно велико, поэтому для подтверждения предложенного подхода было введено ограничение на набор инструкций, используемых в прототипе инструмента анализа программ. Учитывались инструкции помещения или чтения значения из блока памяти *mov* (*movzx, movsx* и др.), инструкции работы со строками (*cmps, movs, scas, lods, ins, outs*), инструкции сравнения *cmp* и другие.

Перечисленные виды инструкций работают с несколькими операндами, но у всех них есть операнд-источник данных для работы и операнд-приемник, куда помещается результат работы. В качестве операндов для инструкций могут выступать регистры, ячейки памяти и константы (immediate operand) – значения, закодированные в самой инструкции. С точки зрения проверки свойств безопасности программы, наиболее интересными являются случаи, когда инструкция работает с памятью – если одним из двух перечисленных операндов инструкции (то есть осуществления чтения или записи) является память, то потенциально эта инструкция может стать причиной ошибки доступа. Важным условием для возможности появления ошибки доступа к памяти является возможность использования внешних данных программы при вычислении адреса доступа. В связи с этим при построении символьной модели вычислений программы учитывались только инструкции, операнды которых явно или косвенно зависели от внешних данных программы.

Реализация алгоритмов обнаружения ошибок доступа к памяти проводилась на базе инструмента символьного исполнения программ Anxiety [13], разрабатываемого в ИСП

РАН. Инструмент Anxiety производит построение символьной формулы, описывающей исполнение программы, только для помеченных инструкций, то есть таких инструкций, значения операндов которых явно или косвенно зависят от внешних данных программы. Для этого, во-первых, в инструменте Anxiety производится анализ помеченных данных (dataflow taint analysis), в рамках которого реализованы алгоритмы распространения помеченности операндов инструкций. Для операций доступа к памяти в формулу попадают предикаты только для тех инструкций, операнды которых помечены в результате распространения пометок. Если как минимум один операнд инструкции помечен, то это означает, что адрес операции доступа к памяти вычисляется с использованием внешних данных программы явно или косвенно.

Во-вторых, для проверки наличия дефекта на заданных на предыдущем шаге инструкциях необходимо описать состояние безопасности программы при выполнении этих инструкций и сформулировать ограничения, накладываемые на данные программы, для обеспечения этого безопасного состояния. Безопасным состоянием программы будем считать корректное исполнение инструкций – доступ к памяти осуществляется по разрешенным адресам и с разрешённым уровнем доступа. Условие корректного исполнения инструкций будем называть предикатом безопасности [14]. Инвертирование предиката безопасности в процессе символьного анализа исполнения программы если ограничения совместны должно привести к генерации входных данных, приводящих программу в небезопасное состояние, что означает возможность реализации ошибки в программе. При реализации прототипа инструмента были разработаны предикаты безопасности для обнаружения нарушения доступа к памяти.

Адрес доступа к памяти может составляться в инструкции различными способами:

- хранение адреса в регистре;
- формирование адреса из значений базы, индекса и смещения.

```
mov eax -> (ecx)                 : address = %ecx
mov ecx -> 0xa4(ebp)             : address = 0xa4 + %ebp
mov edx -> 0xc8(ebp, eax, 0x8)   : address = 0xc8 + %ebp + %eax * 0x8
```

Рис. 1. Варианты формирования адреса в архитектуре x86
Fig. 1. Options for address generation in x86 architecture

Для того чтобы адрес считался помеченным, нужно чтобы была помеченной хотя бы одна из его составных частей. В результате адрес в функции предикатов должен быть представлен в виде выражения над символьными переменными, которое установит взаимосвязь символьной модели программы с конкретными значениями входных данных программы. Предикат безопасности, нацеленный на выявление нарушения доступа к памяти, осуществляет проверку корректности построения значения адреса, по которому производится операция. Функция предиката безопасности принимает на вход адрес в виде выражения над символьными переменными и составляет условие принадлежности этого адреса тем регионам памяти в программе, для которых производимая в инструкции операция разрешена. Следовательно, если для инвертированного предиката возможен адрес, выходящий за границы этих регионов, то на полученных входных данных произойдёт выход за границы разрешенного региона и сгенерируется прерывание ошибки доступа или произойдёт «тихая» ошибка, не приводящая к аварийному завершению программы.

Все адресное пространство программы разделено на области с различными правами доступа – чтение (R), запись (W), исполнение (X) и их комбинации. Про некоторые области памяти неизвестно ничего, в таком случае можно считать их неразмеченными. В прототипе реализован подход, который собирает информацию об известных

размеченных областях памяти и проверяющий попадание вычисленного адреса в неразмеченные области памяти или в размеченные области памяти с ограничением доступа, не позволяющим проведение определённой операции. Для составления предиката безопасности доступа к памяти нужна информация о разметке адресного пространства программы. Получить эту информацию можно несколькими способами:

- отслеживания загрузки модулей в адресное пространство;
- перехват системных вызовов;
- специальные файлы операционной системы.

В связи с тем, что информация из этих источников пересекается, то для корректной реализации проверки предиката безопасности операций реализован механизм управления информацией о разметке адресного пространства в виде теневой памяти, который позволяет отслеживать информацию о выделенных буферах в памяти, разрешать конфликты при пересечении буферов и прав доступа объединять смежные области и т.п.

Реализация модели теневой памяти была осуществлена на основе инструмента динамической бинарной инструментации DynamoRIO [15], в частности, использовались возможности по отслеживанию загрузки исполняемых модулей и перехвату системных вызовов mmap, munmap и mprotect. Системный вызов mmap осуществляет отображение файла в адресное пространство процесса с указанными правами на доступ. Как правило, в вызове уже передаются начальное расположение сегмента в памяти и размер, но в некоторых случаях определение стартового адреса возложено на операционную систему. В таких случаях требуется осуществлять перехват не только вызова, но и возвращаемого значения, которое содержит начальный адрес нового сегмента и его размер. Системный вызов munmap используется для выгрузки региона и освобождения памяти, информацию из данного вызова следует использовать для удаления сегмента. Вызов mprotect осуществляется для изменения прав доступа уже существующего региона памяти.

Ещё одним способом получить информацию о разметке адресного пространства является обращение к специальным файлам в операционной системе. В операционных системах Unix информация об адресном пространстве процесса хранится и постоянно обновляется в специальном файле /proc/[pid]/maps, где [pid] – уникальный идентификатор процесса. Этот файл содержит наиболее полную информацию обо всех сегментах памяти, включая неразмеченные области. Однако из-за большого количества таких сегментов это увеличивает размер самого предиката безопасности. С другой стороны, такой наиболее полный предикат безопасности позволяет наиболее точно задать область запрещенных адресов для конкретной операции.

Таким образом, в анализ был добавлен предикат безопасности, позволяющий проверить инструкции доступа к памяти на возможность обращения к адресам, расположенным в недоступных регионах памяти, тем самым вызывая исключение нарушения доступа к памяти. Если в ходе символьного исполнения при инвертировании данного предиката SMT-решатель смог найти решение для всего предиката пути и построить входные данные, то во время запуска на них программа должна аварийно завершиться или проявить «тихую» ошибку. Если этого не произошло, значит предикат был составлен не точно, и подобранное решение для предиката на самом деле относится к валидной области памяти.

4. Экспериментальная проверка предложенного подхода

Испытания разработанного метода проводились на наборе программ с открытым исходным кодом из комплекта поставки Debian Linux. Результаты тестирования приведены в табл. 1.

Табл. 1. Результаты испытаний прототипа
Tab. 1. The results of the tests of the prototype

Приложение	Итерации	Дефекты без предикатов	Дефекты с предикатами	Время (с)
faad	73	1	1	5400
pnmhistmap	236	1	2	5402
i686-w64-mingw32-objdump	71	1	2	5401
m17n-dump	65	0	1	5402
vde_autoilnk	25	2	4	5401
swig2.0	116	1	1	5403
eperl	70	0	0	5403
sg_unmap	64	1	2	5402
xapian-chert-update	158	2	2	5403
yodlpost	76	0	0	5401
hdp	46	0	2	5403

Каждая программа анализировалась на протяжении полутора часов. В качестве входных данных были подобраны файлы и опции в ожидаемом для каждой программы формате, так как это способствует значительному ускорению анализа – инструменту не нужно тратить время на обход начальных условий, обрабатывающих корректность вводимых данных. Для проверки корректности работы предикатов безопасности набор программ был проанализирован за тот же промежуток времени без их использования.

В табл. 1 значения обнаруженных дефектов указаны за два запуска: первое значение – запуск без предикатов безопасности, второй – с предикатами. В результате запусков проектов с предикатами безопасности общее количество обнаруженных дефектов возросло, более детальный анализ показал, что в ряде программ действительно удалось обнаружить новые уязвимости. Но в большинстве случаев это оказываются известные ранее уязвимости, для которых с помощью предикатов безопасности подобрались новые входные данные, расположенные на альтернативных уязвимых путях исполнения программы.

5. Заключение

В данной работе предложена символьная модель и прототипная реализация метода вычисления внешних данных программы на основе символьного исполнения программ, которые позволяют демонстрировать ошибки в программах. Реализованный прототип подтверждает применимость теоретической модели на практике для реализации инструментов анализа программ с целью обнаружения ошибок в программах при помощи динамического символьного исполнения программ.

Дальнейшее развитие работы может проводиться в направлении реализации инструментов для обнаружения ошибок различного типа, таких как утечка ресурсов операционной системы, использования неинициализированных переменных и др.

Список литературы / References

[1]. B. P. Miller, L. Fredriksen, B. So. An empirical study of the reliability of UNIX utilities. Communications of the ACM, vol. 33, issue 12, 1990, pp. 32- 44.

[2]. M. Zalewski. Symbolic execution in vuln research. Available at: https://lcamtuf.blogspot.com/2015/02/symbolic-execution-in-vuln-research.html, 23.12.2019.
[3]. R.S. Boyer, B. Elspas, K.N. Levitt. SELECT – F Formal System for Testing and Debugging Programs by Symbolic Execution. In Proc. of the International Conference on Reliable software, 1975, pp. 234-245.
[4]. А.Ю. Герасимов, Л.В. Круглов. Вычисление входных данных для достижения определенной функции в программе методом итеративного динамического анализа. Труды ИСП РАН, том 28, вып. 5, 2016, стр. 159-174 / A.Y. Gerasimov, L.V. Kruglov. Input data generation for reaching specific function in program by iterative dynamic analysis method. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 5, 2016, pp. 159-174 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-10.
[5]. Герасимов А.Ю., Круглов Л.В., Ермаков М.К., Вартанов С.П. Подход определения достижимости программных дефектов, обнаруженных методом статического анализа программ, при помощи динамического анализа. Труды ИСПРАН, том 29, вып. 5, 2017 г., стр. 111-134 / Gerasimov A.Y., Kruglov L.V., Ermakov M.K., Vartanov S.P. An approach of reachability confirmation for static analysis defects with help of dynamic symbolic execution. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 5, 2017. pp. 111-134 (in Russian). DOI: 10.15514/ISPRAS-2017-29(5)-7.
[6]. Godefroid P., Levin M. Y., Molnar D. SAGE: whitebox fuzzing for security testing. Communications of the ACM. vol. 55, issue 3, 2012, pp. 40-44.
[7]. CWE-476: NULL Pointer Dereference. Available at: https://cwe.mitre.org/data/definitions/476.html, accessed: 23.12.2019.
[8]. CWE-121: Stack-based Buffer Overflow. Available at: https://cwe.mitre.org/data/definitions/121.html, accessed: 23.12.2019.
[9]. CWE-122: Heap-based Buffer Overflow. Available at: https://cwe.mitre.org/data/definitions/122.html, accessed: 23.12.2019.
[10]. CWE-123: Write-what-where Condition. Available at: https://cwe.mitre.org/data/definitions/123.html, accessed: 23.12.2019.
[11]. CWE-125: Out-of-bounds Read. Available at: https://cwe.mitre.org/data/definitions/125.html, accessed: 23.12.2019.
[12]. CWE-787: Out-of-bounds Write. Available at: https://cwe.mitre.org/data/definitions/787.html, accessed: 23.12.2019.
[13]. A. Gerasimov, S.Vartanov, M. Ermakov, L. Kruglov, D. Kutz, A. Novikov, S. Asryan. Anxiety: a dynamic symbolic execution framework. In Proc. of the 2017 Ivannikov ISPRAS Open Conference, 2017, pp. 16-21. DOI: 10.1109/ISPRAS.2017.00010
[14]. Федотов А.Н., Каушан В.В., Гайсарян С.С., Курмангалеев Ш.Ф. Построение предикатов безопасности для некоторых типов программных дефектов. Труды ИСП РАН, том 29, вып. 6, 2017 г., стр. 151-162 / Fedotov A.N., Kaushan V.V., Gaissaryan S.S., Kurmangaleev Sh.F. Building security predicates for some types of vulnerabilities. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017. pp. 151-162 (in Russian). DOI: 10.15514/ISPRAS-2017-29(6)-8.
[15]. D. Bruening, T. Garnett, S. Amarasinghe. An infrastructure for adaptive dynamic optimization. In Proc. of the International Symposium on Code Generation and Optimization, 2003, pp. 265-275.

Информация об авторах / Information about authors

Александр Юрьевич Герасимов – кандидат физико-математических наук, старший научный сотрудник ИСП РАН. Сфера научных интересов: автоматический анализ программ, статический анализ программ, динамический анализ программ, комбинированные методы анализа программ, управление научными исследованиями и разработками, жизненный цикл разработки безопасного ПО.

Alexander Yurievich GERASIMOV – PhD in Computer Sciences, senior researcher. Research interests: automatic program analysis, static program analysis, dynamic program analysis, combined methods for program analysis, R&D management, SSDLC.

Александр Андреевич НОВИКОВ – аспирант ИСП РАН. Его научные интересы включают динамическое символьное исполнение программ, символьный анализ многопоточных программ, методы обнаружения программных дефектов.

Alexander Andreevich NOVIKOV – PhD Student at ISP RAS. Research interests: dynamic symbolic execution, symbolic execution of multithreaded programs, program defect detection.

Даниил Олегович КУЦ – аспирант ИСП РАН. Научные интересы включают информационную безопасность, динамический анализ программ, задача решения формул в теориях.

Daniil Olegovich Kuts – PhD Student at ISP RAS. Research interests: information security, dynamic program analysis, SMT.

DOI: 10.15514/ISPRAS-2019-31(6)-3

CC BY

Программный комплекс для выявления недекларированных возможностей в условиях отсутствия исходного кода

[1,2] *А.Б. Бугеря, ORCID: 0000-0002-9698-458X <shurabug@yandex.ru>*
[2] *В.Ю. Ефимов, ORCID: 0000-0003-3433-6787 <real@ispras.ru>*
[2] *И.И. Кулагин, ORCID: 0000-0003-2191-1578 <i.kulagin@ispras.ru>*
[2,3] *В.А. Падарян, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*
[2,3] *М.А. Соловьев, ORCID: 0000-0002-0530-6442 <icee@ispras.ru>*
[4] *А.Ю. Тихонов, ORCID: 0000-0003-1705-5166 <fireboo@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, Москва, улица Солженицына, д. 25.*
[3] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*
[4] *Московский государственный технический университет имени Н.Э. Баумана, 105005, Москва, 2-я Бауманская ул., д. 5, стр. 1*

Аннотация. Обнаружение недекларированных возможностей программного обеспечения является одной из основных задач анализа безопасности бинарного кода. Автоматизация решения этой задачи затруднена и требует участия эксперта информационной безопасности. Существующие решения ориентированы на ручную работу аналитика, автоматизация его действий не несет в себе системный характер. В случае отсутствия необходимого инструментария аналитик лишается необходимой поддержки и вынужден самостоятельно заниматься разработкой инструментов, что сильно отдаляет его от получения необходимых практических результатов. В данной работе представлен программный комплекс, решающий задачу выявления недекларированных возможностей в целом: от создания контролируемой среды выполнения до подготовки высокоуровневого описания интересующего алгоритма. Представлен пакет инструментов разработчика QEMU QDT, предлагающий поддержку жизненного цикла разработки виртуальных машин, включая вопросы специализированного тестирования и отладки. Представлено высокоуровневое иерархическое представление алгоритма программы на основе блок-схем, а также алгоритм его построения. Предложенное представление основано на гиперграфе и позволяет реализовать ручной анализ потока данных на различных уровнях детализации. В будущем разработанное представление может использоваться для реализации алгоритмов автоматического анализа. Предложен подход к повышению качества полученного представления алгоритма с помощью объединения отдельных потоков данных в один, связывающий логические модули алгоритма. Для оценки результата построения высокоуровневого представления алгоритма разработан набор тестов на основе реальных программ и модельных примеров.

Ключевые слова: анализ бинарного кода; блок-схемы алгоритмов; анализ потока данных; контролируемое выполнение; специализированные среды разработки

Для цитирования: Бугеря А.Б., Ефимов В.Ю., Кулагин И.И., Падарян В.А., Соловьев М.А., А.Ю. Тихонов. Программный комплекс для выявления недекларированных возможностей в условиях отсутствия исходного кода. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 33–64. DOI: 10.15514/ISPRAS–2019–31(6)–3

Благодарности: Работа поддержана грантом РФФИ № 16-29-09632.

A software complex for revealing malicious behavior in untrusted binary code

[1,2] *A.B. Bugerya, ORCID: 0000-0002-9698-458X <shurabug@yandex.ru>*
[2] *V.Yu Efimov, ORCID: 0000-0003-3433-6787 <real@ispras.ru>*
[2] *I.I. Kulagin, ORCID: 0000-0003-2191-1578 <i.kulagin@ispras.ru>*
[2,3] *V.A. Padaryan, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*
[2,3] *M.A. Solovev, ORCID: 0000-0002-0530-6442 <icee@ispras.ru>*
[4] *A.Yu. Tikhonov, ORCID: 0000-0003-1705-5166 <fireboo@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 RAS Alexander Solzhenitsyn st., 25, Moscow, 109004, Russia*
[3] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[4] *Bauman Moscow State Technical Univarsity, ul. Baumanskaya 2-ya, 5/1, Moscow, 105005, Russia*

Abstract. One of the main problem of a binary code security analysis is a revealing of malicious behavior in an untrusted program. This task is hard to automate, and it requires a participation of a cybersecurity expert. Existing solutions are aimed on the analyst manual work; automation they provide does not demonstrate a system approach. In case where needed analysis tools are absent, the analyst loses the proper support and he is forced to develop tools on one's own. This greatly slows down him from obtaining the practical results. The paper presents a software complex to solve a revealing of malicious behavior problem as a whole: from creating a controlled execution environment to man guided preparing a high-level description of an analyzed algorithm. A QEMU Developer Toolkit (QDT) is introduced, offering support for the domain specific development life cycle. QDT is especially suited for QEMU virtual machine development, including specialized testing and debugging technologies and tools. A high-level hierarchical flowchart-based representation of a program algorithm is presented, as well as an algorithm for its construction. The proposed representation is based on a hypergraph and it allows both automatic and manual data flow analysis at various detail levels. The developed representation is suitable for automatic analysis algorithms implementation. An approach to improve the quality of the resulting representation of the algorithm is proposed. The approach combines individual data streams into the one that links separate logical modules of the algorithm. A test set based on real programs and model examples has been developed to evaluate the result of constructing the proposed high-level algorithm representation.

Keywords: binary code analysis; flowcharts; data flow analysis; controlled execution; domain specific development environment.

For citation*:* Bugerya A.B., Efimov V.Yu. Kulagin I.I., Padaryan V.A., Solovev M.A., Tikhonov A.Yu. A software complex for revealing malicious behavior in untrusted binary code. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 33-64 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-3

Acknowledgements. The work is supported by RFBR grant # 16-29-09632.

1. Введение

Развитие информационных технологий во многом определяется запросами рынка, требующего от разработчиков программного обеспечения (ПО) новых функций и применения уже существующих технологий в новых областях. Высокий темп развития в условиях временны*х* и ресурсных ограничений привел к тому, что вопросы

безопасности ПО рассматривались как второстепенные, и только последние годы они начали получать должный приоритет у передовых разработчиков.

Угрозы безопасности исходят как от деструктивного функционала, целенаправленно заложенного в программу, так и от случайных программных дефектов, эксплуатация которых приводит к аналогичным последствиям – отказам в обслуживании, порче данных, утечке конфиденциальной информации.

На стороне разработчика уже сформировался технологический инструментарий для разработки безопасного ПО. Он позволяет сокращать число ошибок, своевременно выявляя их на ранних этапах жизненного цикла, помогает прослеживать реализуемые функции на этапах разработки – от формулирования требований до поставки пользователям исполняемого кода (дистрибутива).

Принципиально сложнее положение в области аудита, когда требуется оценить безопасность ПО в отсутствии исходных текстов и документации. Такого рода ситуации возникают, когда программные системы включают в свой состав сторонние библиотеки, доступные только в виде исполняемого бинарного кода, когда анализируются проприетарные приложения, системное ПО или встроенное ПО программно-аппаратных платформ, исходные коды которых и документация не доступны из-за экспортных или каких-то других ограничений. Практика такова, что из-за высокой трудоемкости ручного, плохо автоматизированного анализа бинарного кода аудиторы вынужденно изучают только отдельные фрагменты средних и крупных программ, основываясь в своем выборе на экспертном опыте: знаниях, какие компоненты ПО наиболее критичны для безопасности, истории ранее найденных ошибок и т.п.

Крайне востребованы методы и программные средства, способные качественно изменить работу аудитора в задачах оценки безопасности бинарного кода сложных программных систем: автоматизировать решение типовых задач, позволить формализовано выразить экспертные знания об исследуемой программе для улучшения работы автоматических средств анализа, повторного использования результатов обратной инженерии, обучения новых специалистов.

На данный момент существует ряд программных систем, включая системы промышленного уровня, предлагающих набор готовых средств анализа и возможности по расширению этих средств. К ним относятся итеративный дизассемблер IDA Pro [1], средство обратной инженерии Ghidra [2], платформы анализа бинарного кода BAP [3], angr [4], Radare2 [5], среда анализа бинарного кода ТРАЛ [6] и др. В состав этих расширяемых систем изначально или силами сторонних разработчиков включаются алгоритмы распознавания высокоуровневых конструкций (программных модулей, отдельных функций, операторов и выражений), механизм символьного выполнения, анализ помеченных данных и др.

Тем не менее, применение даже таких развитых систем сталкивается с еще не решенными проблемами, затягивающими получение целевого результата: восстановленного алгоритма, оценки критичности программного дефекта, восстановления структуры программы или форматов обрабатываемых данных. Стоит выделить две причины, растягивающие работу аудитора: отсутствие возможности провести динамический анализ и непосредственно увидеть фактические данные, с которыми работает программа, и этап ручного восстановления интересующего алгоритма в виде высокоуровневой блок-схемы.

В данной работе представлены результаты, позволяющие качественно ускорить проведение обратной инженерии бинарного кода, включая этап подготовки инструментальных средств, необходимых для эффективного анализа. Во разд. 2 описывается программный комплекс, позволяющий эффективно проводить обратную инженерию ПО в условиях отсутствия исходного кода. В разд. 3 описывается

инструментарий быстрой разработки контролируемых сред выполнения, которые затем будут использоваться при проведении динамического анализа. В разд. 4 рассматривается задача автоматизированного ручного анализа бинарного кода с целью высокоуровневого описания свойств заданного алгоритма и последующего экспертного анализа свойств алгоритма, в первую очередь – в части наблюдаемых потоков данных. Формулируется обобщённая постановка задачи поиска недекларированных возможностей (НДВ) и приводится обзор соответствующих результатов мирового уровня. В следующем, пятом, разделе предлагается иерархическое высокоуровневое представление алгоритма, описаны разработанные алгоритмы построения этого представления. Раздел 6 посвящен набору тестовых программ, использовавшихся в качестве репрезентативных примеров в ходе разработки высокоуровневого представления и оценки удобства ручной работы. В седьмом, последнем разделе даются итоговые заключения.

2. Архитектура программного комплекса анализа бинарного кода

Состав комплекса технологий обусловлен различными требованиями: по расширяемости, скорости адаптации под новые программно-аппаратные платформы, по возможностям интеграции с другими средствами анализа, причем расширяемость должна обеспечиваться в двух различных направлениях: (1) возможностей комплекса должно быть достаточно для решения различных практических задач информационной безопасности, (2) должна обеспечиваться применимость комплекса к разным классам ПО (различных процессорных архитектур, различных ОС, различный уровень ПО: встроенное, системное и прикладное).

Упрощенная схема комплекса представлена на Рис. 1.

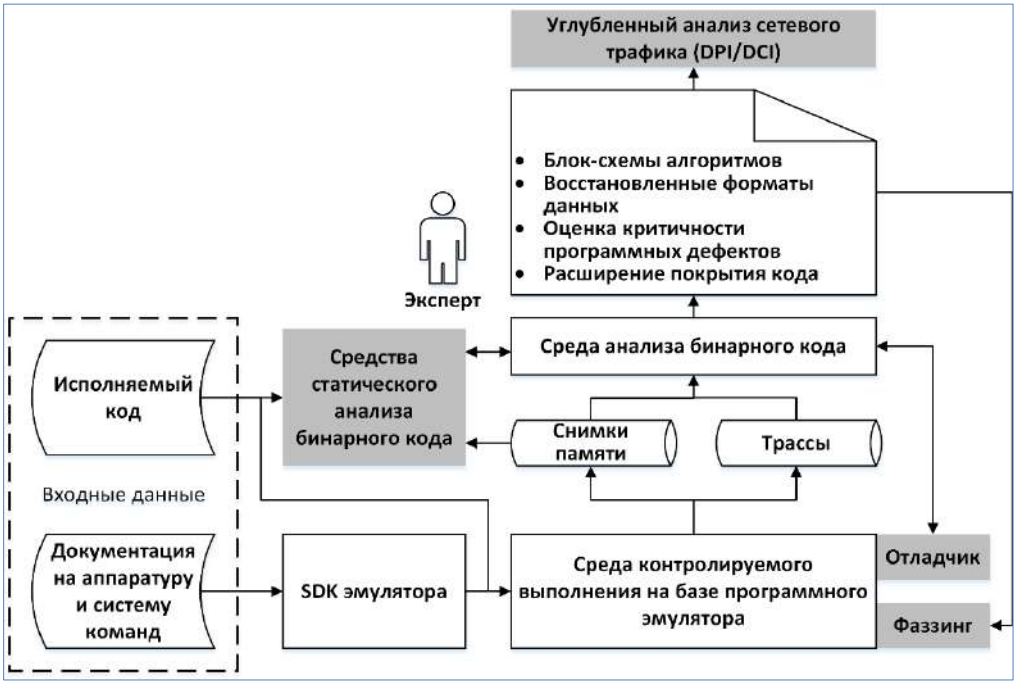

Рис. 1. Состав программного комплекса анализа бинарного кода и его связь со сторонними средствами анализа

Fig. 1. The structure of the binary code analysis software package and its relationship with third-party analysis tools

Комплекс состоит из трех основных компонент: среды анализа бинарного кода, среды контролируемого выполнения на базе программного эмулятора, где собирается информация о работе исследуемой программы, и средств быстрой разработки новых виртуальных машин.

Исследуемое ПО разворачивается в виртуальной машине (Рис. 2), эмулирующей необходимую аппаратуру, что обеспечивает возможности полносистемного анализа ПО для различного уровня. При выполнении ПО собираются трассы и снимки состояния памяти. Помимо post mortem анализа эмулятор позволяет непосредственно контролировать текущее состояние виртуальной машины, как через интерфейс отладчика, так и непосредственно встроившись в эмулятор, как это делают современные фаззеры.

Основными входными данными для среды анализа выступают как трассы выполнения уровня машинных команд, поскольку они отражают всю фактическую информацию о выполнившимся коде, так и снимки физической памяти эмулируемой системы, полученные в различные моменты времени. Снимки используются для дополнения восстанавливаемого по трассам статико-динамического представления в тех местах, где не удалось покрыть трассами код. Помимо того, снимки памяти непосредственно передаются в сторонние средства статического анализа бинарного кода, например, BINSIDE [7].

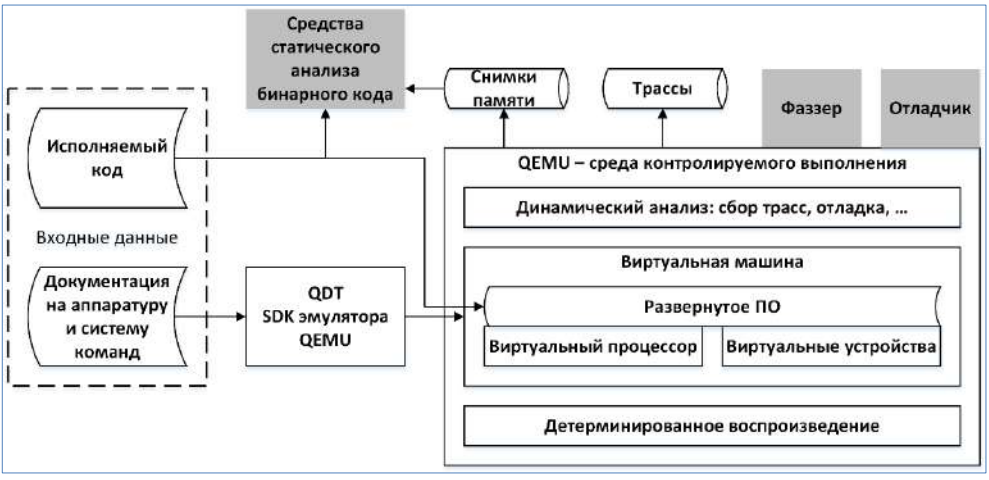

Рис. 2. Обеспечение возможностей динамического анализа средствами программной эмуляции
Fig. 2. Providing dynamic analysis capabilities with software emulation

Возможность провести динамический анализ критична для работы всего комплекса, поскольку трассы выполнения выступают «отправной точкой» для среды анализа. Для работы ОС общего назначения, таких как Windows и Linux, и их приложений на платформе x86 достаточно штатных виртуальных машин, входящих в состав эмулятора QEMU. Для некоторых аппаратных платформ удаётся подобрать либо уже готовую виртуальную машину, либо самостоятельно ее скомпоновать из штатно поддерживаемых процессоров и устройств. В остальных случаях работоспособности исследуемого кода приходится добиваться, разрабатывая необходимую виртуальную аппаратуру.

В настоящее время процессы разработки эмулятора QEMU опираются на классические инструменты, а сама разработка сопряжена с написанием большого объема шаблонного кода и времязатратным исправлением ошибок в виртуальных устройствах. Чтоб ускорить создание виртуальной машины, позволяющей выполниться объекту исследования, был разработан и реализован пакет инструментов разработчика QEMU [8]

(QEMU Development Toolkit, QDT) QDT, который меняет существующие процессы разработки и автоматизирует деятельность разработчиков. QDT поддерживает разработку новых виртуальных процессоров, отдельных устройств (контроллеров, шин, периферии и т.п.), интеграцию компонент полноценной виртуальной машины.

Среда анализа ТРАЛ позволяет проводить обратную инженерию бинарного кода, восстанавливая алгоритмы в виде блок-схем, для обрабатываемых этими алгоритмами данных – спецификацию форматов, оценивает критичность сработавших в трассах выполнения программных дефектов. Более детальное устройство среды ТРАЛ показано на Рис. 3.

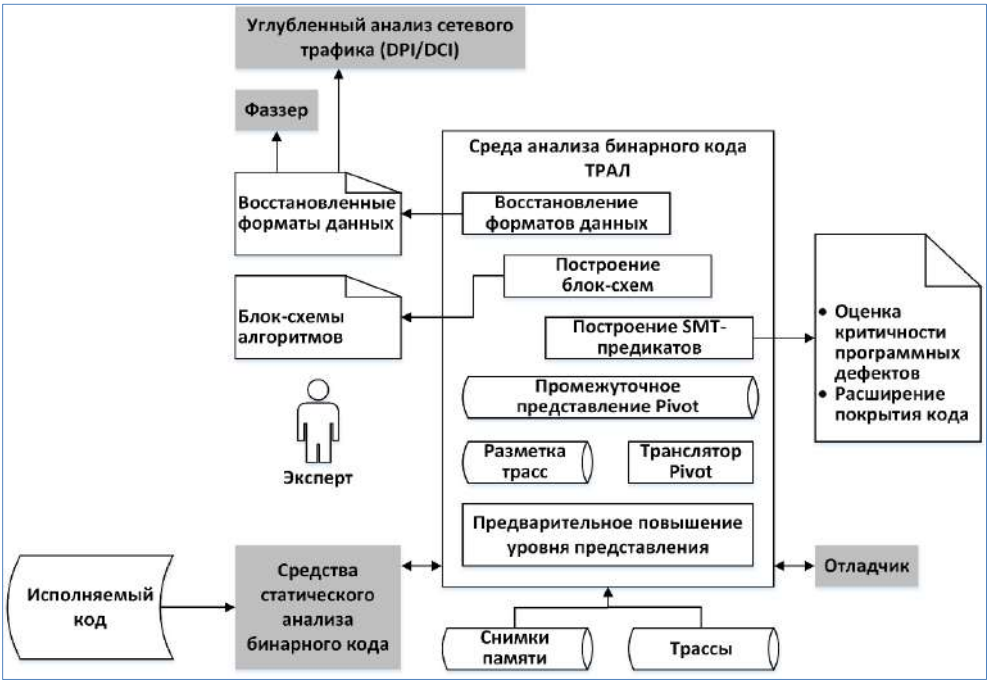

Рис. 3. Анализ трасс выполнения в среде ТРАЛ
Fig. 3. Analysis of execution traces in the TRAL environment

Работа с трассами в среде ТРАЛ начинается с повышения уровня представления, которое проводится автоматически. В определённой последовательности трассы просматривают алгоритмы анализа, строящие разметку и статико-динамическое представление программы. В трассах размечаются участки, относящиеся к различным процессам и тредам, выделяются обработчики прерываний, вызовы функций. По командам, покрытым трассами, формируется последовательность карт памяти, учитывающая модификацию кода по ходу выполнения. Восстанавливаются функции, графы потоков данных и управления, граф вызовов функций. Анализ потоков данных и управления выполняется поверх унифицированного представления декодированных инструкций, которое вырабатывают модули поддержки процессорных архитектур. Такого представления достаточно для построения блок-схем алгоритмов и восстановления форматов данных.

Если внутреннее устройство исследуемой программы частично или полностью неизвестно, нет исчерпывающего описания ее интерфейсов и форматов входных/выходных данных, то эффективность фаззинга как метода поиска ошибок падает. Становится затруднительно подготовить конфигурацию фаззинга, обеспечивающую хорошее покрытие. Некоторые интерфейсы, через которые в

программу поступают входные данные, могут остаться незамеченными. Более того, для определенных классов ПО, например, встроенного, необходимо определить критерии, позволяющие трактовать некоторое состояние программы как состояние сбоя. Такие критерии предварительно вырабатываются путем ручного анализа отдельных алгоритмов, представленных в виде блок-схем. Восстановленные форматы данных транслируются в вид, пригодный для использования в генерационном фаззинге (PeachPit). Помимо того, на основе восстановленных форматов открывается возможность уточнять разборщики протоколов, в системах анализа трафика, например, ProtoSphere [9].

Во многих практических задачах исследования бинарного кода недостаточно знать только потоки данных и управления, необходимо уметь унифицировано анализировать, как именно работают с данными команды различных процессорных архитектур. Для этого операционная семантика кода транслируется в промежуточное представление Pivot. Наиболее важные сценарии его применения предполагают динамическое символьное выполнение через дальнейшую трансляцию Pivot-кода в SMT-предикаты. Первый сценарий – поиск входных данных, позволяющий активировать ни разу не сработавший в рассматриваемых трассах условный переход, второй – оценка критичности программного дефекта.

В первом сценарии ищутся входные данные, позволяющие инвертировать в заданной точке трассы условный переход. Для этого формируется предикат пути, в котором входные данные выступают в качестве свободных символьных переменных, операции над ними записываются в виде символьных выражений, а каждый сработавший условный переход добавляет в общий предикат пути свое условие. Условие перехода в заданной точке трассы записывается в общую систему ограничений в инвертированном виде. Собранный предикат пути передается в SMT-решатель, в случае нахождения его решения открывается возможность расширить покрытие кода.

Второй сценарий предполагает совмещение предиката пути с предикатом безопасности, который описывает реализацию уязвимости некоторого типа.

Возможности среды анализа не исчерпываются описанными выше сценариями, среда содержит несколько десятков модулей с различными алгоритмами анализа. Имеется API, позволяющее пользователям среды независимо разрабатывать свои алгоритмы и оформлять их в виде подключаемых модулей.

3 Процесс разработки модели вычислительной машины для эмулятора QEMU

Виртуальная машина (ВМ) в эмуляторе QEMU состоит из относительно самостоятельных программных компонентов, процесс ее разработки можно разделить на подпроцессы, схожие между собой и с объемлющим процессом.

Состав ВМ можно условно разделить на модуль поддержки системы команд процессора (транслятор команд целевой процессорной архитектуры) и комплект модулей, реализующих шины, контроллеры, устройства ввода-вывода. На практике разработка как транслятора, так и сложных устройств оказывается весьма трудозатратной, требует высокой квалификации разработчика, отладка ошибок плохо автоматизируется и, как следствие, длится неприемлемо долго.

Каждый программный компонент ВМ взаимодействует с эмулятором через внутренний API. Процесс разработки любого компонента начинается с реализации базовой версии работоспособного модуля, имеющего минимальный функционал. Затем начинается цикл итеративной разработки, состоящий из внесения необходимого функционала, тестирования, отладки и доработки. На практике цикл приостанавливается, когда все

существенные ошибки исправлены, а модель обеспечивает достаточную точность эмуляции для работы исследуемого ПО.

В теории все «вложенные» подпроцессы разработки модулей могут выполняться параллельно и независимо. Но, например, тестировать реализацию системы команд или отдельное устройство значительно легче, если все компоненты ВМ собраны воедино. Т.е. часть этапов разных процессов может выполняться параллельно, а часть выгодней выполнять совместно и последовательно.

Целью работ, проводимых в ИСП РАН, являлось выделение в процессе разработки этапов, поддающихся автоматизации, и непосредственно сама автоматизация. Для разработки всех обозначенных выше компонентов были созданы специализированные программные инструменты. Большинство из них входит в QDT.

Ниже рассматриваются процессы классической ручной разработки компонентов ВМ и возникающие при этом трудности. Приводятся способы автоматизации процессов, в конце раздела описан предлагаемый процесс разработки ВМ с использованием инструментов автоматизации.

3.1. Система команд

Работа с системой команд целевого процессора ведется фронтендом динамического двоичного транслятора TCG [10]. Фронтенд принимает на вход данные, являющиеся машинными командами, и возвращает код на промежуточном представлении TCG. Этот код изменяет состояние виртуальной машины так, как изменилось бы состояние настоящей вычислительной машины, если бы на ней был выполнен соответствующий машинный код. Поскольку для QEMU приоритетна скорость эмуляции, то некоторые, допустимые особенности работы команд не эмулируются.

Представление семантики машинной команды – программа на промежуточном представлении TCG, выполняющаяся вместо команды и совершающая над некоторой моделью вычислительной машины действия, эквивалентные действиям, выполняемым командой при выполнении на настоящей машине.

Разработчик фронтенда должен:

- реализовать парсер (декодер) машинного кода;
- выделить важные подробности работы команд;
- разработать на языке TCG семантику каждой команды;
- реализовать на языке Си программу, которая вернёт разработанную семантику;
- протестировать работу всего переднего плана.

Наконец, для того, чтобы поддержку системы команд можно было использовать, нужно создать модель процессора.

Декодер машинного кода совмещает в себе лексический и синтаксический анализ, где токенами являются последовательности битов (коды операций, операнды и т.д.), из которых собираются машинные команды. Ручное кодирование разбора двоичного представления – задача длительная, рутинная и чревата неочевидными опечатками, поиск которых может потребовать много времени.

Однако по совокупности описаний кодировок машинных команд можно автоматически сгенерировать анализатор [11]. Задать кодировки машинных команд значительно проще, чем вручную разработать анализатор. На практике эта задача сводится к переписыванию таблиц из документации.

Сгенерированный анализатор состоит из вложенных блоков `switch-case`, где выбор каждого следующего блока case происходит путём выделения подпоследовательности битов в машинном слове. В конце выполняется вызов функции, соответствующей распознанной машинной команде. В течение вызова этой функции должна быть создана

программа на промежуточном представлении TCG, выполняющая действия, подразумеваемые соответствующей командой, – семантика команды. После работы генератора требуемые функции создаются с пустыми телами, куда разработчику следует вписать код, выражающий семантику команд.

Вместе с анализатором по специальному описанию генерируется модель процессора. Описание процессора включает перечень регистров, длину адреса, порядок байтов в машинном слове и др. Эти параметры также учитываются при генерации анализатора.

Описание семантики команд предоставляется производителем процессора на естественном языке, единственное исключение – формальная спецификация ISA ARM v8-A и v8-M, разработанная компанией ARM [12, 13] При этом не существует единого формата описания, которого бы придерживались различные производители. Кроме того, в описаниях часто бывают неточности и ошибки, выявление которых требует значительных усилий человека.

Составление программы на промежуточном представлении TCG выполняется путём вызова специальных функций из API QEMU. Другими словами, разработчик пишет не непосредственно на промежуточном представлении, а на языке Си, составляя программу, которая вернёт требуемую программу.

Промежуточное представление является низкоуровневым, напоминающим язык ассемблера, поэтому даже простая программа, например, реализация команды сложения с выставлением флагов в регистре состояния процессора, получается длинной и трудночитаемой.

Для упрощения написания семантики был разработан инструмент I3S [14] – Instruction Set Semantics Specification (Описание семантики системы команд). I3S принимает описание семантики машинных команд на одноимённом языке, основывающемся на языке Си, и генерирует программу на языке Си, которая возвращает программу в промежуточном представлении TCG.

Наиболее надёжным способом тестирования транслятора является сравнение с эталоном – настоящим процессором, однако доступ к такому эталону не всегда возможен. В таких случаях вместо эталона может быть использован произвольный процессор [15], в частности – процессор на рабочем месте разработчика. В основе такой замены лежит тот факт, что избегая неопределенного и реализационно-зависимого поведения, можно составить тестовую программу на языке Си, которая будет одинаково выполняться на процессорах различной архитектуры. Выбор относительно низкоуровневого языка Си обусловлен тем, что если для некоторой, малораспространённой процессорной архитектуры доступны средства разработки, то они гарантированно будут содержать компилятор языка Си. Поиск несоответствий выполнения эталона и транслированного кода происходит на уровне терминов языка Си: значение переменной, номер строки в пути выполнения. При обнаружении несоответствия разработчик самостоятельно разбирается, чем вызвана проблема: ошибкой в машинной команде или различиями в системах команд. Поскольку тесты изначально разрабатываются с учётом возможных различий, несоответствия второго вида на практике не встречаются. Хотя у данного подхода есть множество ограничений, он позволяет покрыть значительную (более 60%) часть реализации. Кроме того, тесты обладают высокой универсальностью, и, как следствие, могут быть повторно использованы при разработке других ВМ без изменений. Подход реализован в инструменте c2t из QDT.

3.2. Периферийные устройства

Периферийные устройства в эмуляторе QEMU задаются в событийно-ориентированном стиле. Виртуальное устройство – набор функций на языке Си (обработчиков событий) и набор структур данных, описывающих состояние времени выполнения.

В число типовых событий входят:

- создание (инициализация) устройства без привязки к контексту объемлющей ВМ;
- создание экземпляра устройства (*realization*) в составе ВМ;
- сброс в начальное состояние;
- чтение и запись в регистр.

Кроме того, устройства могут издавать и принимать запросы прерываний.

Многие устройства поддерживают настройку через интерфейс командной строки. Например, можно указать файл образ для ПЗУ, реальный сетевой интерфейс для виртуального сетевого адаптера и др.

Разработчик виртуального устройства должен:

- описать состояние устройства в терминах языка Си;
- реализовать настройку устройства пользователем;
- выбрать события, нуждающиеся в обработке;
- реализовать на языке Си реакцию виртуального устройства на каждое событие;
- протестировать работу устройства.

Положение с документацией устройств аналогично схоже с описанием систем команд – если документация доступна, она подготовлена на естественном языке и не поддается автоматической формализации. Наибольшие затруднение вызывает ситуация, когда отсутствует какое-либо описание, имеется только доступ к исполняемому коду драйвера устройства в составе образа исследуемого ПО. В таких случаях в цикле итеративной разработки появляется задача обратной инженерии драйвера [16]. На каждой итерации разработчик виртуального устройства пытается продвинуться дальше по ходу выполнения драйвера до следующего сбоя в его работе. Обнаружение настоящей причины сбоя – нетривиальная, во многом творческая, задача. Затем происходит исследование причины и корректировка поведения виртуального устройства.

Реализация виртуального устройства сводится к написанию функций и структур данных на языке Си. API QEMU накладывает множество формальных требований к тому, как должен быть написан этот код. Эти требования обусловлены как языком Си, так и стилем кодирования, внутренними интерфейсами эмулятора. Например, в API определены структуры данных, куда должны быть занесены указатели на функции-обработчики событий. И эти функции должны обладать конкретной сигнатурой. Другими словами, определённая часть кода виртуального устройства изначально определена интерфейсом прикладного программирования (API) QEMU. Эту часть будем называть интерфейсной. Остальную же часть кода, реализующую индивидуальные особенности работы конкретного устройства, будем называть индивидуальной.

Создание интерфейсной части кода хорошо поддаётся автоматизации. Достаточно составить список событий и сформировать высокоуровневые данные об устройстве, например, имя устройства, чтобы стало возможно сгенерировать начальную заготовку устройства.

Заготовка устройства состоит из различных конструкций языка Си, многие из которых взаимосвязаны. Например, обработчик записи в банк регистров регистрируется в структуре данных, описывающих этот регистр. Эта структура данных, в свою очередь, регистрируется функцией инициализации устройства в служебной структуре, сообщая эмулятору, что у данного устройства есть такой-то банк регистров. Эту цепочку можно продолжать и дальше.

В общем случае зависимости между конструкциями образуют ациклический граф. Чтобы код устройства был синтаксически корректным, нужно правильно расставить идентификаторы, расположить конструкции языка в синтаксически корректном порядке,

подключить нужные заголовки (причём желательно минимизировать их количество, учитывая включения одних другими) и т.д. Следует соблюсти требования к стилю написания кода, принятые в сообществе разработчиков QEMU, и учесть многие другие особенности.

При классической разработке это всё приходилось учитывать разработчику вручную. Существующая в сообществе разработчиков QEMU практика такова, что новые устройства компонуются из похожих по функционалу кусков кода готовых виртуальных устройств. В QDT реализован инструмент, позволяющий по описанию устройства сгенерировать заготовку для устройства с соблюдением всех требований и зарегистрировать её в инфраструктуре эмулятора. Описание формализуется с помощью API QDT на языке Python, оно как правило на порядок меньше, чем получаемая из него заготовка.

Наибольшая часть индивидуальной части устройства находится в обработчиках чтения и записи в банк регистров. Принцип разбиения банков регистров на отрезки, выровненные по границе байта, т.е. регистры с индивидуальным назначением, был использован при автоматизированной генерации заготовок. Кроме как в коде обработчика доступа, регистр присутствует в состоянии времени выполнения устройства, а также упоминается в других частях кода модели. Это создаёт определённые трудности для разработчика – необходимо согласованно редактировать код в разных местах.

Регистры могут быть описаны в одном месте файла конфигурации для генератора заготовок с использованием специально разработанного расширения для API QDT [17].

В течение разработки разнообразных устройств удалось выделить некоторые часто используемые виды регистров устройств:

- регистры, доступные только для чтения;
- регистры для чтения и записи;
- регистры-пустышки;
- регистры, доступные только для записи.

Аналогичным образом может быть описан каждый бит в регистре. Указание вида регистра и его битов позволяет сгенерировать в заготовке устройства дополнительный код.

Использование формального описания регистров сокращает объём кода, написанного вручную, примерно в два раза. Однако основное преимущество в удобстве, т.к. на практике описание регистров сводится к переписыванию таблицы из документации, без необходимости редактировать код в разных местах модели устройства.

3.3. Виртуальная машина

Виртуальная машина – совокупность процессора, включая фронтенд TCG, и устройств. Внутреннее устройство ВМ в значительной степени представлено своей функцией инициализации на языке Си, которая создаёт все компоненты и связывает их между собой.

Если ВМ достаточно большая, то функция инициализации может содержать сотни строк, а при расстановке связей между устройствами вручную легко допустить ошибку. Чтобы упростить создание ВМ, в QDT был разработан графический интерфейс пользователя (ГИП), отображающий ВМ в виде схемы из связанных блоков. ГИП реализован с использованием модели ВМ на языке Python. Эта модель ВМ аналогична модели ВМ из API QEMU, но из-за языковых различий между Си и Python работать с ней проще. Вместе с ней разработан генератор, позволяющий создавать заготовку кода машины с необходимой функцией инициализации. Полученная заготовка в большинстве случаев

требует минимальной доработки в отличие от заготовок для устройств, с генерации которых разработка только начинается.

3.4. Автоматизированный процесс разработки ВМ

Созданные средства автоматизации встраиваются в процесс разработки ВМ, ускоряя и/или облегчая некоторые его этапы. Рассмотрим основные этапы предложенного автоматизированного процесса и отличия от классического процесса разработки.

1. Изучение документации на систему команд и модель процессора.
2. Составление описаний кодировок команд (без семантики) и свойств процессора.
3. Автоматизированная генерация анализатора машинных команд, заготовки виртуального процессора и пустых обработчиков машинных команд.
4. Написание на языке I3S семантики машинных команд в обработчиках.
5. Автоматизированная трансляция описания семантики с языка I3S в программу на языке Си, генерирующую промежуточное представление TCG.
6. Доработка заготовки виртуального процессора.
7. Итеративная разработка:
   a) автоматизированное тестирование с использованием c2t;
   b) отладка;
   c) доработка.

В цикле итеративной разработки тестирование используется для приблизительного поиска ошибки, а отладка – для её локализации.

Доработку можно производить классическим способом, но рекомендуется вносить правки в описания из п. 2 и/или 4 с последующей повторной генерацией кода согласно п. 3 и 5, соответственно. Чтобы не затереть правки, внесённые на промежуточных шагах, рекомендуется использовать систему контроля версий. В QDT реализован вспомогательный инструмент для Git, который автоматически переносит правки на новую версию, используя механизм «rebase». Вмешательство разработчика требуется только в случае конфликтов изменений.

При классической разработке вместо этапов со 2 по 5 происходило ручное написание на языке Си как декодера команд, так и транслятора TCG; код виртуального процессора приходилось писать с нуля. Тестирование зачастую выполнялось вручную с помощью отладчика, поскольку в большинстве случаев системы тестирования, имеющиеся в открытом доступе, не были рассчитаны на требуемую систему команд. Тесты писались на языке ассемблера.

Автоматизированная разработка устройств состоит из следующих этапов.

1. Изучение документации на устройство.
2. Составление перечня событий и элементов API QEMU, которые потребуются для работы модели. Опционально подготавливается описание регистров.
3. Автоматизированная генерация заготовки устройства.
4. Реализация на языке Си индивидуальной части кода устройства.
5. Итеративная разработка: тестирование, отладка, доработка.

В случае ошибки в формальном описании устройства (п. 2) можно вносить правки в заготовку, как это происходит всегда при классическом процессе. Однако часто бывает быстрее внести изменения в описание и повторить генерацию (п. 3). При повторной генерации текущая реализация устройства затирается, поэтому важно использовать систему контроля версий. В состав QDT включен инструмент для работы с Git, позволяющий автоматизированно сгенерировать новую версию и перенести наработки

(п. 4 и ниже) на новую версию. Задача разработчика в этом случае – только разрешение конфликтов.

Методика тестирования устройства определяется видом этого устройства. Например, для тестирования сетевого адаптера можно подключить его к виртуальному адаптеру основной машины и попытаться обмениваться пакетами с гостевой ОС. Наиболее универсальным способом является загрузка гостевого ПО. Во время загрузки ОС драйверы обычно проводят проверку и инициализацию своих устройств. По сообщениям об ошибках от драйверов можно судить о наличии ошибки в модели устройства. Хорошей практикой является работа двух разработчиков: один реализует виртуальное устройство, другой разрабатывает гостевое ПО, которое должно тестировать работу устройства. Участие второго разработчика крайне желательно, т.к. это минимизирует риск тиражирования ошибки, вызванной неправильной трактовкой текста документации.

Процесс автоматизированной разработки ВМ включает следующие этапы.

1. Изучение документации с целью определения системы команд и версии процессора, перечня устройств и взаимосвязей между ними.
2. Составление формального описания состава ВМ на языке Python вручную и/или с использованием GUI.
3. Автоматизированная генерация заготовки ВМ.
4. Реализация на языке Си не формализованных особенностей работы.
5. Итеративная разработка: тестирование, отладка, доработка.

Формальное описание состава ВМ изначально разрабатывалось для ручного написания. Позже появился ГИП с возможностью отображения состава ВМ в виде схемы. Он позволяет изменять состав и сохранять описание в Python с использованием удобного для ручной доработки форматирования. Возможности ГИП отстают от возможностей ручного способа. Поэтому при составлении описания рекомендует попеременно использовать оба способа, т.к. некоторые манипуляции удобнее производить над текстовым описанием, а некоторые – через ГИП.

После генерации ВМ (п. 3) возможно сразу получить работоспособную ВМ. Однако часто разработка ВМ начинается одновременно с разработкой виртуальных устройств. Это вызвано тем, что тестировать устройства в составе ВМ легче. В результате к моменту завершения разработки последнего устройства, ВМ начинает работать сама собой. В некоторых случаях ВМ может требовать реализации нестандартного поведения, согласно документации (п. 4).

Доработка ВМ производится по тому же принципу, что и доработка устройств.

4. Автоматизированное восстановление алгоритма в виде блок-схемы с целью анализа его прикладных свойств и характеристик

В условиях отсутствия исходных текстов и документации выявление НДВ в бинарном коде программ вынужденно рассматривается как проверка выполнения базовых положений информационной безопасности в восстановленном представлении алгоритмов. При отсутствии «декларации», к нарушениям того, что декларируется следует относить: (1) любые программные дефекты, приводящие к сбоям, порче и утечке данных, (2) особенности реализации алгоритмов, приводящие к выдаче в открытый канал связи чувствительных данных в исходном виде либо после таких преобразований, когда эти данные могут быть полностью или частично восстановлены, иными словами – приводящие к утечке чувствительных данных. Программные дефекты в бинарном коде выявляют либо ручным анализом, либо применяя различные средства статического и динамического анализа [7, 18, 19]. В данной работе рассматривается второй вид нарушений, требующий активного участия эксперта. В этом случае человеком

проводится проверка отдельных алгоритмов программы, т.к. тотальный ручной анализ современного ПО невозможен из-за его размера и сложности. Ставится задача автоматизации анализа свойств отдельного алгоритма экспертом.

В общем случае ручная проверка разбивается на следующие этапы: 1) локализация алгоритма в коде программы; 2) представление алгоритма в удобной для анализа форме; 3) исследование свойств алгоритма. К сожалению, полная автоматизация этого процесса невозможна, так как первый и третий этап требуют выполнения нетривиальных действий от эксперта-аналитика. Облегчить труд аналитика на этапе локализации алгоритма позволяет построение статико-динамического представления программы [20], выявление вызовов известных библиотечных функций и событий уровня операционной системы, которые служат ориентирами в общем коде. Трудозатраты аналитика на выполнение третьего этапа, а также качество полученного результата во многом определяются способом представления анализируемого алгоритма. Получение такого представления алгоритма осложнено из-за отсутствия исходного текста программы, а следовательно – ее высокоуровневой семантики.

Разработка выразительного промежуточного представления (*Intermediate Representation – IR*) программы была и остается актуальной задачей. Существует большое число различных промежуточных представлений, имеющих свои преимущества и недостатки. Каждое из них разрабатывалось для конкретной области применения. Далее рассматриваются промежуточные представления, успешно решающие задачи анализа и преобразования кода программы в областях оптимизирующей компиляции, декомпиляции и высокоуровневого представления алгоритма программы.

4.1. Промежуточные представления программ для задач компиляции

Для задач компиляции наиболее известными являются IR, используемые в компиляторах, негласно признанных промышленным стандартом. А именно, промежуточные представления GENERIC, GIMPLE и RTL, реализованные в наборе компиляторов GCC, и LLVM IR, используемый в LLVM. Существуют и другие известные IR, например, основанное на графе представление программы PDG – program dependence graph (граф зависимостей программы) [21, 22], которое объединяет в себе информацию о потоке данных и потоке управления. Вершинами в этом графе являются команды, а ребра разделены на два типа: 1) выражающие зависимость по данным и 2) выражающие зависимость по управлению. Несмотря на то, что подобные IR хорошо зарекомендовали себя в задачах компиляции (анализ кода для выполнения оптимизирующих преобразований программы), они не пригодны для эффективного анализа бинарного кода и построения высокоуровневого представления алгоритма, так как анализируемый код лишен информации о высокоуровневой семантике программы.

4.2. Промежуточные представления программ для моделирования процессорных команд и анализа бинарного кода

Для задач анализа бинарного кода в общем и задач декомпиляции в частности развиваются специализированные промежуточные представления. Среди множества существующих можно выделить следующие: Pivot/Pivot 2 [23, 24], B2R2 IR [25], REIL [26], MAIL [27], BAP (BIL) [3, 28], BitBlaze [29], ESIL [5].

Промежуточный язык REIL – Reverse Engineering Intermediate Language [26] позволяет выполнять статический анализ кода программы в контексте задач обратной инженерии. Для моделирования машинных команд целевой архитектуры (или описания операционной семантики машинных команд целевой архитектуры) этот язык имеет в своем составе 17 команд, реализующих абстрактную машину с бесконечной памятью и

бесконечным числом регистров. В последствии на основе языка REIL появился RREIL [30].

В платформе BAP [3] анализа бинарного кода для описания семантики команд целевой архитектуры используется один уровень промежуточного представления, реализуемый языком BIL [28]. В отличие от BAP аналогичная ей система BitBlaze [29] поддерживает два уровня промежуточного представления: VEX IL [31], реализованное в средстве динамической инструментации Valgrind, и Vine IL [29]. Первое позволяет описать максимально точно семантику машинных команд с учетом побочных эффектов, а второе IR позволяет сократить избыточную для выполнения анализа детализацию. Точно так же, как и REIL, язык Vine IL содержит команды для абстрактной машины с неограниченным объемом памяти и числом виртуальных регистров.

Еще одним схожим по назначению с вышеупомянутыми языками является язык ESIL – Evaluable String Intermediate Language [5], который используется для описания семантики машинных команд набором инструментов для обратной инженерии radare2. Данный язык реализует абстрактную машину (как и в предыдущих случаях) и позволяет детально описать побочные эффекты машинных команд. Выражения на данном языке записываются в обратной польской нотации.

Также к вышеупомянутому множеству языков можно отнести специализированный промежуточный язык MAIL – Malware Analysis Intermediate Language [27], который содержит команды для описания структуры и поведения бинарной программы. Этот язык позволяет выполнять статический анализ программ, ориентированный на обнаружение полиморфного вредоносного кода.

Стоит отметить, что перечисленные выше промежуточные языки, используемые для анализа бинарного кода, в первую очередь ориентированы на детальное описание операционной семантики команд целевой архитектуры. Поэтому они не подходят для высокоуровневого представления алгоритма программы. Однако, они могут быть использованы в качестве начального IR, из которого итерационно будет удаляться избыточная детализация с последующим повышением уровня представления.

4.3. Высокоуровневые представления алгоритма программ

Если задачу построения высокоуровневого представления алгоритма программы свести к задаче декомпиляции, то результатом будет восстановленный из бинарного кода исходный код на высокоуровневом языке. Для решения данной задачи известные методы декомпиляции [1, 32, 33, 34] выполняют анализ потока управления программы для восстановления ее структуры. Основная идея анализа заключается в сопоставлении сегментов графа потока управления программы с шаблонами из заранее подготовленного экспертами множества, для которых определена соответствующая высокоуровневая языковая конструкция (например, `if-then-else`, `switch-case` и др.). В результате выполненных компилятором оптимизирующих преобразований в коде программы могут появиться несводимые регионы, для которых отсутствует соответствующая языковая конструкция, и тогда используется оператор безусловного перехода для повторения потока управления из бинарного кода. На практике такое случается часто. Кроме того, нерешенной остается задача восстановления структур данных. Это приводит к тому, что результирующий код фактически является копией машинного на высокоуровневом языке. Такой код не пригоден не только для высокоуровневого представления алгоритма, но и для ручного анализа экспертом. В некоторых случаях он может быть полезен для автоматического анализа либо для перекомпиляции программы под другую целевую архитектуру.

Существуют подходы к декомпиляции бинарного кода программы с привлечением методов машинного обучения [35, 36]. Такие подходы сосредоточены на двух задачах:

1) автоматизировать трудоемкий процесс создания шаблонных фрагментов графа потока управления с известной языковой структурой высокого уровня при помощи возможностей аппарата машинного обучения и 2) использовать возможности машинного обучения для установки соответствия операндов машинного кода и переменных высокоуровневых языковых конструкций. На сегодняшний день такие инструменты представляют только академический интерес, и говорить об их применении для решения практических задач еще рано.

Существуют промежуточные представления более высокого уровня, чем те, которые используются для решения задачи декомпиляции. Например, в работе [37] предложено IR для задач поиска криптографических функций.

В работе [38] предложено гибридное высокоуровневое представление программы, объединяющее в себе граф потока данных и потока управления (HI-CFG – Hybrid Information- and Control-Flow Graph). Граф HI-CFG состоит из двух типов вершин: 1) соответствующих фрагментам кода программы (они могут представлять код модуля или функции) и 2) соответствующих структурам данных программы. Вершины в графе связаны тремя типами ребер:

1) ребра между вершинами-фрагментами соответствуют ребрам графа потока управления и показывают последовательность передачи управления;

2) вершины второго типа соединены между собой ребрами, отражающими зависимость по данным и показывающими направление передачи информации от одной структуры данных к другой;

3) третий, последний, тип ребер соединяет между собой вершины обоих типов в порядке отношения «производитель-потребитель». Данный тип ребер показывает, какие структуры данных являются входными аргументами в фрагмент кода, а какие – выходными.

HI-CFG представляет сложную программу в виде ее декомпозиции на основные логические компоненты, интерфейсы взаимодействия между которыми определяются используемыми структурами данных. Такое высокоуровневое представление алгоритма не только позволяет выполнять анализ программы в автоматическом режиме, но и является удобным для восприятия и ручной обработки аналитиком.

На Рис. 4 показан пример высокоуровневого представления HI-CFG программы, которая выполняет разбор (parser) двух типов команд (decoder1/decoder2), поступающих в качестве входного аргумента (input_buffer), декодирует их, используя таблицу поиска (lookup_tbl), и выполняет соответствующие вычисления (compute1/compute2).

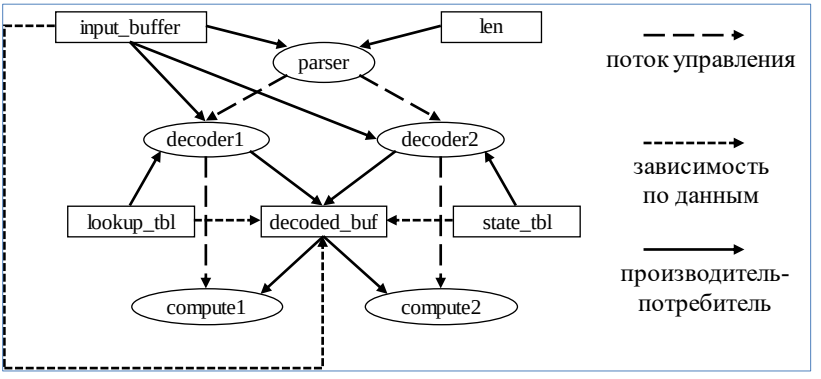

Рис. 4. Пример высокоуровневого представления HI-CFG программы
Fig. 4. An example of a high-level presentation of an HI-CFG program

Восстановление структур данных и определение буферов программы играют особую роль в получении точного и корректного высокоуровневого представления HI-CFG, так как они определяют интерфейсы, с помощью которых связываются логические компоненты программы. Отсутствие отладочной информации в бинарном коде существенно усложняет задачу восстановления структур данных. Для решения этой задачи авторы HI-CFG при построении представления используют известные методы, основанные на динамическом анализе помеченных данных, анализе шаблонов обращения к памяти, распространении типов данных от входных и выходных аргументов хорошо известных библиотечных и системных вызовов.

Более конкретно, формирование буферов выполняется алгоритмом, аналогичным тому, что используется в инструменте Howard [39] для обнаружения массивов. Этот алгоритм динамического анализа бинарного кода основан на эвристическом предположении, что доступ к массивам осуществляется в циклах. Таким образом, алгоритм группирует в массивы используемые области памяти в циклах, доступ к которым осуществлялся с одинаковым смещением относительно предыдущего элемента или относительного базового адреса массива. Стоит отметить, что данный алгоритм в реализации авторов HI-CFG не распознает многомерные массивы.

Для вывода типов используется инструмент REWARD [40], выполняющий динамический анализ потока данных на основе алгоритма Aggregate Structure Identification – ASI (алгоритм идентификации агрегированных структур данных) [41], который, в свою очередь, базируется на алгоритме Хиндли-Милнера [42]. Идея алгоритма заключается в следующем: при выполнении вызова библиотечной функции, системного вызова или выполнении команды, аргументы или операнды которых имеют достоверно известный тип, осуществляется пометка соответствующих аргументам или операндам областей памяти их типом данных. После этого информация о выведенном типе данных распространяется по зависимым операндам в процессе выполнения программы.

Информация о программе, содержащаяся в HI-CFG, может быть по отдельности получена существующими инструментами анализа бинарного кода. Авторы этого высокоуровневого промежуточного представления программы попытались объединить в нем одном результаты работы существующих алгоритмов анализа. Например, наличие ребер трех типов позволяет охарактеризовать модификацию информационного потока в терминах потока данных и потока управления. Представление PDG [21, 22] также имеет ребра, описывающие поток данных и поток управления, однако он не имеет вершин, являющихся структурами данных программы.

Таким образом, среди существующих подходов к промежуточному представлению программы HI-CFG является наиболее подходящим для ручного анализа и высокоуровневого представления алгоритма (на уровне блок-схем). Также, HI-CFG поддерживает различные уровни детализации: вершины, являющиеся фрагментами кода, могут представлять как базовые блоки кода, так и функции, а вершины, соответствующие структурам данных – как отдельные ячейки памяти, так и крупные области (типизированные или нетипизированные). Кроме того, представление HI-CFG содержит большой набор примитивов (ребра и вершины различных типов), что предоставляет возможность для реализации алгоритмов автоматического анализа различных схем потоков данных и выполнения.

5. Иерархическое высокоуровневое представление алгоритма

В контексте обнаружения НДВ, как правило, необходимо выполнить анализ алгоритмов, оперирующих чувствительными данными. Высокоуровневое представление анализируемого алгоритма должно содержать команды, участвующие в формировании

результата – *результирующего буфера*, тем самым, позволяя определить, какие входные данные использовались для выполнения алгоритма, и где они размещались, происходили ли утечки чувствительных данных при выполнении шагов алгоритма.

При проектировании промежуточного представления необходимо учесть, что оно будет использоваться по двум основным сценариям: 1) построение компактного аннотированного представления в виде иерархической блок-схемы, пригодной для ручного анализа и 2) изучение свойств алгоритма в автоматическом режиме для определения декларированных и недекларированных потоков данных. Представление должно быть применимо для динамического анализа бинарного кода в автономном режиме, то есть его построение следует реализовать по трассе выполнения программы.

Трасса выполнения программы представляет собой последовательность выполненных процессором команд, а также значения регистров на каждом шаге. Такая трасса может быть получена в результате исполнения программы полносистемным эмулятором, например, QEMU [43], либо в результате выполнения программы под управлением средств динамического инструментирования бинарного кода, например, Pin [44], DynamoRIO [45], Valgrind [31] и др.

Предлагаемое представление алгоритма программы основано на гиперграфе с иерархической организацией и состоит из двух типов вершин: 1) представляющих команды, выполняющиеся на определенном шаге трассы (обозначим их термином *точка*); 2) представляющих ячейку абстрактной памяти (это может быть диапазон адресов в виртуальном адресном пространстве, регистр или его часть) на определенном шаге выполнения трассы (обозначим их термином *буфер*). Ребра в гиперграфе отображают зависимости по данным. Вершины гиперграфа типа точка могут быть объединены в подмножества, называемые *фрагментами*, а фрагменты – в *суперблоки*. Логически связанные между собой вершины типа буфер могут быть объединены в подмножества типа *супербуфер*.

Вершины-фрагменты соответствуют фрагментам кода в трассе, то есть линейной последовательности шагов в трассе, на которых не выполнялись команды вызова функций и возврата из них. Например, если функция *A* вызывает только одну функцию *B*, которая не содержит вызовов функций, то в этом случае будет сформировано три фрагмента кода: 1) F_{A1}, содержащий последовательность команд функции *A* до вызова функции *B*, 2) F_B – последовательность команд функции *B*, 3) F_{A2} – последовательность команд функции *A*, которые выполнялись после возврата из функции *B*. Суперблоки соответствуют экземплярам вызова функций. Они состоят из фрагментов, принадлежащих данному экземпляру вызова функции, и других суперблоков, которые, в свою очередь соответствуют экземплярам функций, вызванных из данного.

Супербуферы и буферы соответствуют структурам данных программы и определяют интерфейсы взаимодействия между фрагментами и суперблоками. Также они характеризуют значение потока данных на момент входа в фрагменты или суперблоки или выхода из них. Для суперблоков входные и выходные аргументы определяются буферами и супербуферами.

Предложенное высокоуровневое представление позволяет описать алгоритм формирования результирующих буферов в пригодном для ручного и автоматического анализа виде. Ниже представлен алгоритм построения предложенного иерархического представления.

5.1. Алгоритм построения иерархического высокоуровневого представления

В основе построения высокоуровневого представления лежит алгоритм отслеживания потока данных в обратном направлении (обратный слайс трассы). Поэтому

представление алгоритма строится только по тем точкам (шагам в трассе), которые участвуют в формировании результирующего буфера, остальные считаются избыточными, не относящимися к искомому алгоритму, и поэтому не рассматриваются. Точки внутри фрагментов и сами фрагменты между собой связываются ребрами, формируя таким образом граф потока данных.

На вход алгоритма поступает *начальный буфер*, являющийся результатом работы интересующего алгоритма, и диапазон шагов в трассе, на которых выполнялись его команды. Алгоритм выполняет 2 основных шага: 1) выделение в трассе точек, относящихся к алгоритму формирования начального буфера, и объединение их в подмножества фрагментов (Рис. 5) и 2) объединение фрагментов в подмножества суперблоков (Рис. 6). Предполагается, что по трассе выполнения предварительно построено статическое представление программы, содержащее информацию о связях между командами и шагами в трассе и о вызовах функций, восстановлен стек вызовов функций.

```
    Вход: t          – трасса выполнения
                     программы
          C          – информация о вызовах
                     функций
          b:< a_i, l_i > – начальный буфер
    Выход: F         – множество фрагментов
          P          – множество точек

    createPointAndFragments(t, C, b:< a_i, l_i >)
1     F = ∅, f = ∅, P = ∅, p = ∅
2     while (p = getNextPoint(t, b:< a_i, l_i >))
3       P ∪= {p}
4       if (f == ∅)
5         [b, e] = getFragmentBound(C, p)
6         f = newFragment([b, e])
7         addPoint(f, p)
8       else if (p ∈ [b, e])
9         addPoint(f, p)
10      else
11        F ∪= {f}
12        [b, e] = getFragmentBound(C, p)
13        f = newFragment([b, e])
14        addPoint(f, p)
15    if (P ≠ ∅)
16      F ∪= {f}
```

Рис. 5. Псевдокод алгоритма построения множества точек и фрагментов представления
Fig. 5. Pseudocode of the algorithm for constructing a set of points and presentation fragments

На Рис. 5 показан псевдокод алгоритма, реализующего первый шаг построения высокоуровневого представления. Для его выполнения требуется трасса t выполнения программы, информация C о содержащихся в трассе вызовах функций (номер шага, на котором выполнялись команды вызова функций и возврата из них) и начальный буфер b: $< a_i, l_i >$, область в адресном пространстве по адресу a_i и имеющая длину l_i. В результате работы данного шага будут построены множества точек P и фрагментов F. Алгоритм осуществляет восходящий проход по точкам в трассе, относящимся к искомому алгоритму программы (точки из множества, полученного обратным слайсом по начальному буферу). Функция `getNextPoint` возвращает очередную точку p из трассы в порядке ее обратного исполнения. При помощи функции `newFragment`

создаются фрагменты. Для их создания необходимо указать номера шагов начала фрагмента и его конца – $[b, e]$. Границы фрагментов получаются при помощи функции `getFragmentBound` и содержатся в информации о вызовах функций, которая формируется на этапе построения статического представления программы. Добавление точки во фрагмент выполняется функцией `addPoint`.

На Рис. 6 приведен псевдокод алгоритма формирования суперблоков иерархического высокоуровневого представления. Алгоритм представляет собой запуск рекурсивной процедуры `superBlockCover` создания многоуровневых, вложенных суперблоков. Уровень вложенности определяется глубиной стека вызовов функций, то есть каждый суперблок является экземпляром функции, а вложенные в него суперблоки – экземплярами вызываемых функций. На нижнем уровне хранятся фрагменты, наполненные точками. В качестве входных аргументов процедура `createSuperBlocks` принимает множество F фрагментов, полученных в результате работы предыдущего шага, а также стек C вызовов функций трассы, который строится на этапе предварительной обработки трассы. Результатом работы алгоритма является множество S суперблоков. В псевдокоде используются следующие функции:

1) `getCall(`$C, x.b$`)`, используя стек вызовов функций C и номер шага в трассе, возвращает экземпляр функции c, содержащей этот шаг. Экземпляры функций имеют начальные $c.b$ и конечные $c.e$ шаги в трассе;
2) `newSuperBlock(`$[c.b, c.e]$`)` создает новый суперблок s. Фрагменты и суперблоки имеют начальный шаг в трассе (b) и конечный (e);
3) `createEdges(`s, I`)` связывает ребрами фрагменты и суперблоки из множества I и добавляет получившийся граф в суперблок s;
4) `getPrevStackCall(`C, c`)`, используя стек вызовов C, возвращает экземпляр функции предыдущего уровня относительно вызова экземпляра функции c текущего уровня.

```
    Вход: F - множество фрагментов
          C - информация о вызовах функций,
              содержащая стек вызовов функций трассы
    Выход: S - множество суперблоков

    createSuperBlocks(F, C)
1      S = ∅, B = F
2      superBlockCover(B, C)

3   superBlockCover(B, C)
4      foreach (x in B)
5         B \= {x}
6         c = getCall(C, x.b)
7         while (c ≠ ∅)
8            if (x.b ≤ c.b)
9               break
10           s = newSuperBlock([c.b, c.e])
11           S ∪= {s}
12           I = {y_i}: y_i ∈ B ∧ y_i.b ≥ c.b
13           B \= I
14           superBlockCover(I, C)
15           createEdges(s, I)
16           B ∪= {s}
17           c = getPrevStackCall(C, c)
```

Рис. 6. Псевдокод алгоритма построения суперблоков высокоуровневого представления
Fig. 6. Pseudo-code of a high-level representation superblock construction algorithm

Процедура `superBlockCover` выполняет следующие шаги.

1. Из множества B фрагментов и суперблоков извлекается очередной элемент x. По его начальной позиции в трассе $x.b$, используя стек вызовов функций C, получается экземпляр c вызова функции (Рис. 6, строки 4-6), и запускается процесс построения суперблоков для каждого уровня стека вызовов, начиная с вызова функции c (Рис. 6, строка 7). Уровень вызова экземпляра функции c считается текущим.
2. На каждом уровне стека вызовов функций выполняется проверка, находится ли текущий фрагмент или суперблок на уровне не ниже текущего (Рис. 6, строка 8), и если нет, то оставшиеся уровни вызова не рассматриваются. В противном случае создается новый суперблок s при помощи функции `newSuperBlock`, который добавляется в результирующее множество S, и начинается выполнение следующего шага.
3. Из множества суперблоков и фрагментов выбираются те, которые в стеке вызовов функций расположены на уровне не ниже текущего уровня. Выбранные элементы формируют множество I и исключаются из множества B. После чего процедура `superBlockCover` повторяется рекурсивно для вновь сформированного множества I.
4. После возвращения управления из рекурсивного вызова процедуры `superBlockCover` полученные фрагменты и суперблоки связываются между собой ребрами. Затем сформированный граф добавляется в текущий суперблок s, а сам суперблок помещается в результирующее множество B (Рис. 6, строки 15-16).
5. По стеку вызовов получается экземпляр функции предыдущего уровня вызова, который становится текущим. Далее процедура повторяется, начиная с шага 2.

5.2. Пример построения высокоуровневого иерархического представления алгоритма

Ниже рассматривается пример предлагаемого представления с различными уровнями детализации для программы, приведенной на Рис. 7. Программа генерирует ключ `key` при помощи функции `GenerateKey`, получает текст `text` из файла при помощи функции `ReadText` и выполняет его шифрование алгоритмом AES, используя сгенерированный ключ. Результат шифрования хранится в строке `cipher`. Детали реализации функций для демонстрации высокоуровневого представления не важны.

```
key = GenerateKey()
text = ReadText()
cipher = aes(key, text)
```

Рис. 7. Псевдокод программы, выполняющей шифрование текста алгоритмом AES
Fig. 7. Pseudocode for a program encrypting text using AES

Для приведенной программы имеется трасса ее выполнения, а также известен номер шага *x*, на котором завершена работа алгоритма AES. Трасса представлена на Рис. 8а. Для простоты в ней намеренно опускаются конкретные команды, а указаны только точки $p_0, \ldots, p_{27}$ – шаги выполнения. Для наглядности фрагменты кода в трассе выделены серым фоном. Зашифрованный текст хранится в строке `cipher`, которая размещена в области памяти с адресом a_1 и длиной l_1. Эта область памяти формирует отслеживаемый начальный буфер b_1. Стартовой позицией в трассе, с которой начинается построение высокоуровневого представления алгоритма, является шаг *x*.

Как уже упоминалось выше, высокоуровневое иерархическое представление содержит только те точки, которые относятся к искомому алгоритму, то есть точки, принадлежащие отслеживаемому потоку данных. После выполнения первого этапа

алгоритма построения представления, реализованного функцией `createPointAndFragments`, будет сформировано множество точек $P = \{p_1, p_4, p_5, p_7, p_{10}, p_{11}, p_{13}, p_{16}, p_{19}, p_{22}, p_{23}, p_{25}\}$, а также множество фрагментов $F = \{f_0, \dots, f_8\}$. Полученное множество точек соответствует обратному слайсу трассы. На Рис. 8б показаны полученные множества точек и фрагментов на трассе. В построенном высокоуровневом представлении точки и связанные между собой ребрами фрагменты будут формировать граф потока данных. Рис. 8в демонстрирует направления потоков данных для искомого алгоритма, вычисляющего результирующий буфер b_1.

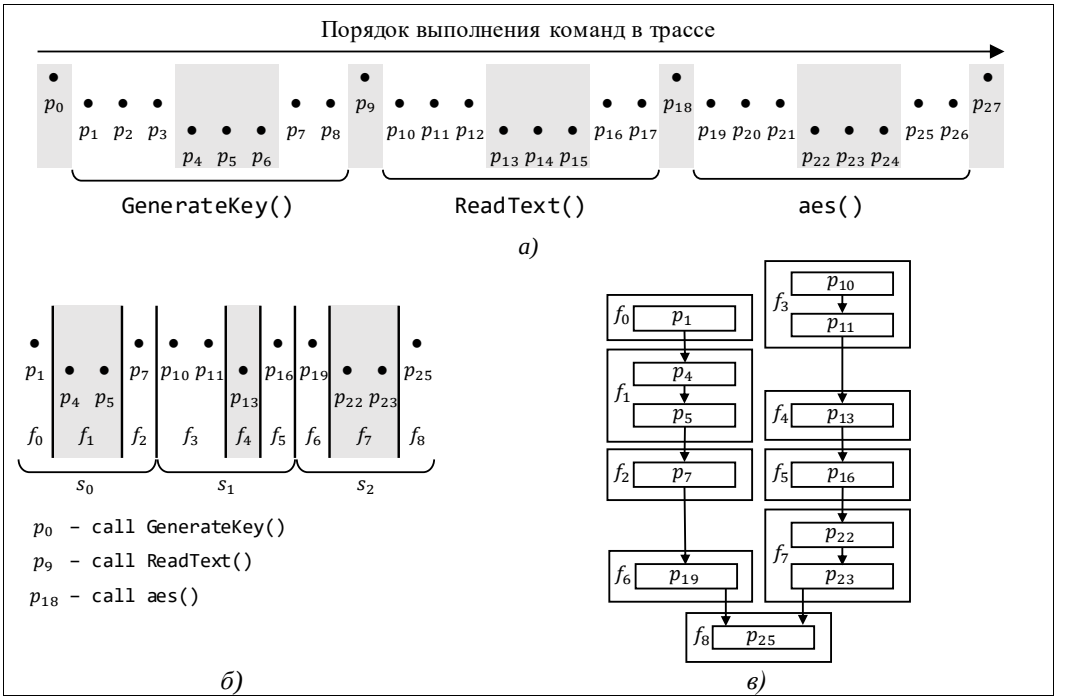

Рис. 8. Трасса выполнения и граф потока данных программы, псевдокод которой представлен на Рис. 7:
а) исходная трасса; б) результат обратного слайса для буфера <a_1, l_1>;
в) направление потоков данных для искомого алгоритма, вычисляющего результирующий буфер b_1
Fig. 8. The execution trace and the graph of the program data flow, whose pseudocode is shown in Fig. 7:
a) source track; б) the result of the inverse slice for the buffer <a_1, l_1>;
в) the direction of the data flows for the desired algorithm that computes the resulting buffer b_1

Полученное множество фрагментов F поступает на вход второго шага алгоритма построения высокоуровневого представления, реализованного процедурой `createSuperBlocks`. Алгоритм восстановления стека вызовов функций C в данной работе не рассматривается. Предполагается, что он уже восстановлен на этапе построения статического представления программы [6, 20, 46]. На втором, завершающем, этапе будет сформировано множество S суперблоков, соединенных между собой ребрами, представляя таким образом *гиперграф*. На Рис. 9 показано результирующее высокоуровневое представление алгоритма с разным уровнем детализации. В результате для примера с Рис. 7 будут сформированы следующие фрагменты:

$$f_0 = \{p_1\}, f_1 = \{p_4, p_5\}, f_2 = \{p_7\}, f_3 = \{p_{10}, p_{11}\}, f_4 = \{p_{13}\},$$

$$f_5 = \{p_{16}\}, f_6 = \{p_{19}\}, f_7 = \{p_{22}, p_{23}\}, f_8 = \{p_{25}\}$$

и суперблоки:

$$s_0 = \{f_0, f_1, f_2\}, s_1 = \{f_3, f_4, f_5\}, s_2 = \{f_6, f_7, f_8\}.$$

Суперблоку s_0 соответствует функция `GenerateKey`, s_1 – `ReadText`, s_2 – `aes`. В случае необходимости, для минимизации представления, суперблоки s_0, s_1, s_2 могут быть свернуты в суперблок s_3. Это позволяет сделать иерархическая организация предложенного представления.

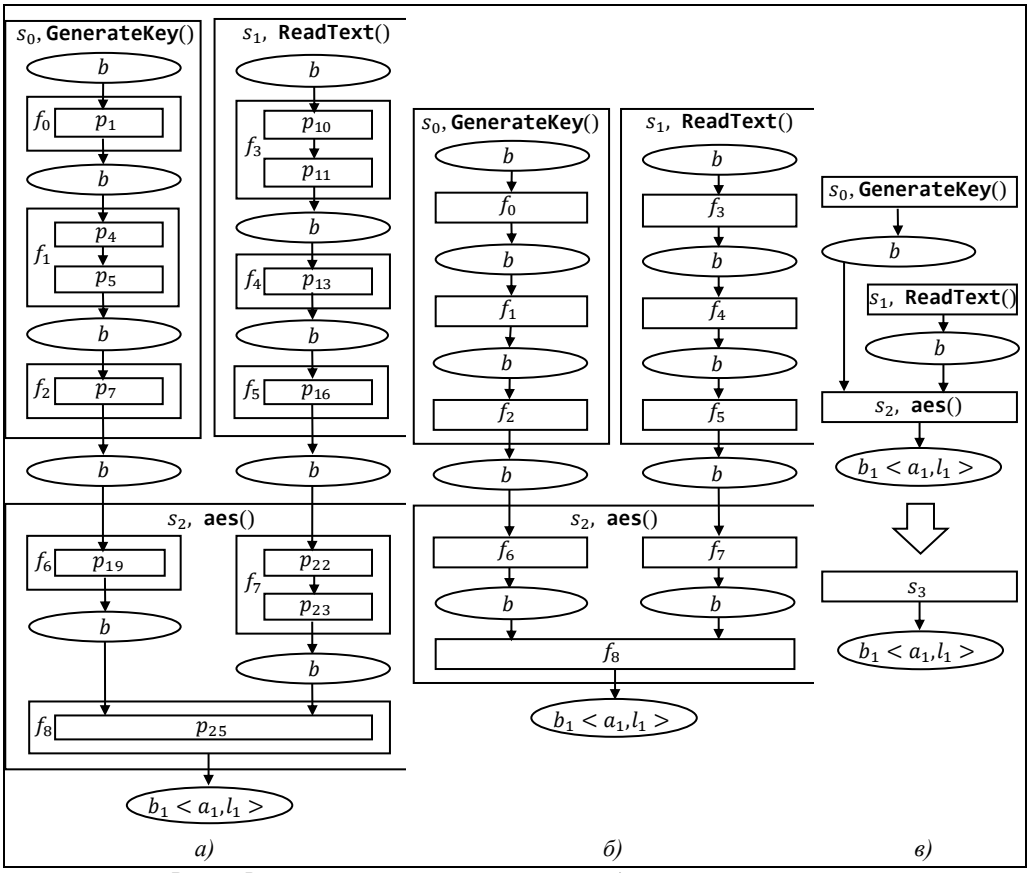

Рис. 9. Высокоуровневое иерархическое представление программы, псевдокод которой представлен на Рис. 7, с различными уровнями детализации: а) уровень точек; б) уровень фрагментов; в) уровень суперблоков

Fig. 9. High-level hierarchical representation of the program, whose pseudocode is shown in fig. 7, with various levels of detail: a) the level of points; б) the level of fragments; в) the level of superblocks

Высокоуровневое представление основано на гиперграфе, содержащем точки (шаги в трассе), на которых выполняются команды искомого алгоритма, формирующие отслеживаемый поток данных. Буферы позволяют получить значение потока данных в различные моменты времени выполнения алгоритма, а фрагменты, суперблоки и супербуферы – выполнить его масштабирование.

Иерархическая организация представления является пригодной как для выполнения анализа алгоритма в ручном режиме, скрыв избыточные подробности алгоритма путем сворачивания некоторых элементов (фрагментов или суперблоков), так и для автоматического анализа.

5.3 Объединение буферов в супербуферы

Используемые структуры данных в программе определяют интерфейсы взаимодействия между модулями программы. Интерфейсы взаимодействия являются одним из ключевых пунктов в вопросе восстановления представления алгоритма, так как позволяют установить логические связи между его компонентами. В контексте анализа бинарного кода установление логической связи затруднено из-за отсутствия в программе информации о высокоуровневой семантике.

В предлагаемом иерархическом высокоуровневом представлении алгоритма программы структуры данных определяются буферами и супербуферами. Ввиду отсутствия информации о высокоуровневой семантике, а именно об используемых типах структур данных, в процессе построения представления искомого алгоритма нельзя однозначно ответить на вопрос, чем является та или иная часть отслеживаемого потока данных на конкретном шаге выполнения: самостоятельной переменной, элементом массива или полем агрегированной структуры данных. Избежать подобной неопределенности позволяют буферы, которые в простейшем случае представляют собой абстрактные ячейки памяти. Однако такой подход затрудняет восприятие представления алгоритма аналитиком, а именно, не позволяет определить интерфейсы взаимодействия между его логическими компонентами. Например, если функция A модифицировала n элементов массива, а после этого функция B использовала их в вычислениях результирующего буфера, то в худшем случае может быть получено представление, в котором будет указано, что эти два модуля связаны между собой по n буферам. В случае больших значений n анализ подобного представления человеком не возможен. Безусловно, если все n элементов образуют смежную непрерывную область, то они будут объединены в один буфер, но такое встречается не всегда. На практике чаще всего шаблоны доступа к памяти не формируют непрерывную область, либо используется нетривиальное размещение структур данных в памяти, что приводит точно к такому же результату.

Избежать избыточную детализацию зависимостей между модулями алгоритма позволяет группировка буферов в супербуферы. В работе предложено две схемы объединения:

1) объединение множества буферов, формирующих непрерывную область, в один буфер;
2) объединение буферов в супербуферы на основе разбиения множества входных и выходных буферов суперблоков на подмножества, содержащие «эквивалентные» буферы.

Первый случай является тривиальным и не нуждается в пояснении в отличие от второго. Понятие «эквивалентность» в данном случае означает, что два различных буфера были выработаны одним общим суперблоком и используются затем в одних и тех же суперблоках, что позволяет их логически объединить. Для пояснения схемы объединения введем следующие операции для суперблоков и буферов:

- $out(s)$ – возвращает множество выходных буферов суперблока s;
- $in(s)$ – возвращает множество входных буферов суперблока s;
- $consumer(b)$ – возвращает множество суперблоков, которые используют буфер b в качестве входного, например, для представления с Рис. 10в $consumer(b_2) = \{s_2, s_3\}$, а $consumer(b_4) = \{s_3\}$;
- $producer(b)$ – возвращает суперблок, который создает буфер b, иначе говоря, $producer(b) = s \mid b \in out(s)$. Например, $producer(b_5) = s_1$ (Рис. 10в).

Таким образом, объединение выходных буферов $out(s)$ во множество супербуферов $B = \{B_i\}$ для суперблока s выполняется так, чтобы каждый супербуфер B_i содержал все буферы b с одинаковым множеством $consumer(b)$.

Ниже приводится пояснение второй схемы объединения буферов в супербуферы для примера с рис. 10. Для демонстрации рассматриваются два варианта представления.

Вариант 1. В результате построения высокоуровневого представления алгоритма было сформировано два суперблока s_1 и s_2 (Рис. 10а), множество выходных буферов $out(s_1) = \{b_1, b_2, b_3\}$ и множество входных буферов $in(s_2) = \{b_1, b_2, b_3\}$. Предполагается, что множества входных и выходных буферов не формируют непрерывную область абстрактной памяти (например, из виртуального адресного пространства или регистрового файла), поэтому на этапе построения не были объединены в один выходной и один входной буферы. На Рис. 10б изображен результат объединения буферов в супербуферы по второй схеме для данного варианта представления. Так как все множество $out(s_1)$ – выход суперблока s_1 – поэлементно соотносится со множеством $in(s_2)$ – входом суперблока s_2, то в результате будет сформирован супербуфер $B_1 = \{b_1, b_2, b_3\}$.

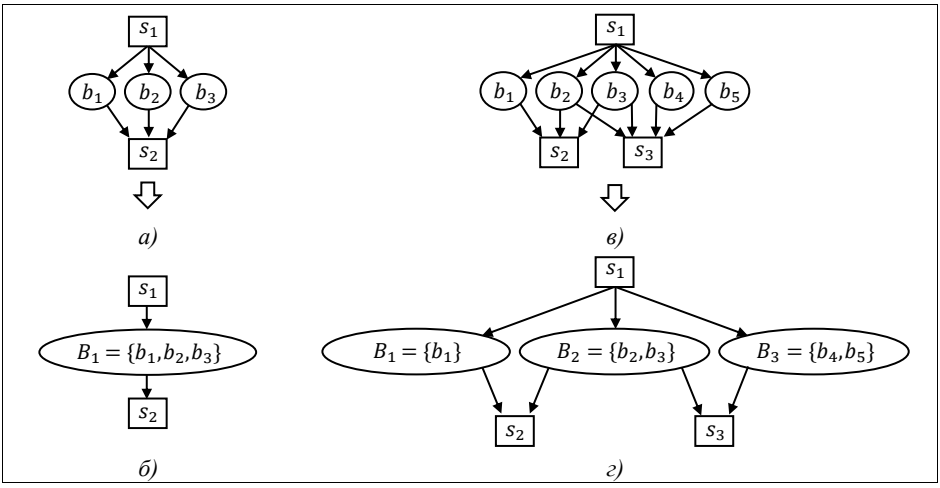

Рис. 10. Пример объединения буферов в супербуферы
Fig. 10. Example of combining buffers into superbuffers

Вариант 2. Фрагмент высокоуровневого представления алгоритма, соответствующий данному варианту, изображен на рис. 10в и содержит три суперблока s_1, s_2, s_3 со множествами $out(s_1) = \{b_1, b_2, b_3, b_4, b_5\}$ выходных буферов из s_1, $in(s_2) = \{b_1, b_2, b_3\}$ и $in(s_3) = \{b_2, b_3, b_4, b_5\}$ входных буферов в суперблоки s_2 и s_3. В этом случае выходные буферы b_2 и b_3 суперблока s_1 используются в качестве входных в суперблоки s_2 и s_3, то есть $consumer(b_2) = consumer(b_3) = \{s_2, s_3\}$. Поэтому выходные буферы суперблока s_1 будут сгруппированы в супербуферы $B_1 = \{b_1\}, B_2 = \{b_2, b_3\}, B_3 = \{b_4, b_5\}$, а не в $B_1 = \{b_1, b_2, b_3\}$ и $B_2 = \{b_3, b_4, b_5\}$, как может показаться на первый взгляд.

6. Набор тестовых программ для оценки результата построения высокоуровневого представления алгоритма

Для оценки возможностей средств восстановления высокоуровневого представления алгоритма по бинарному коду в работе предложен набор программ, содержащий как синтетические тесты, так и реально используемые приложения. При составлении тестового набора учитывался характер задач, которые требуется решить в процессе анализа исследуемых программ. К числу таких задач относится выявление ошибок логики приложения, потенциально приводящих к возникновению утечек чувствительных данных.

Набор тестов, состоящий из реальных приложений и синтетических тестов, должен покрывать значительный класс программ из различных прикладных областей и содержать разнообразные сценарии выполнения, которые либо приводят к возникновению утечек чувствительных данных, либо нуждаются в ручном анализе для доказательства их отсутствия. Для оценки результата построения высокоуровневого представления алгоритма предлагается использовать следующие реальные программы и модельные примеры:

1) модуль установки SSL-соединения браузеров Google Chrome и Mozilla Firefox;
2) программа депонирования криптографических ключей, т.е. реализующая их утечку в открытый канал в виде следующих сценариев:
   a) получение ключа и его утечка происходят в рамках одного потока выполнения;
   b) получение ключа, работа с ним и утечка разнесены по различным потокам выполнения;
3) синтетический тест, содержащий утечку чувствительных данных, оставшихся в стеке после выполнения функции, оперирующей ими;
4) программа, выполняющая «повреждение» секретного ключа перед его использованием по следующей схеме:
   a) перезапись части ключа константными значениями;
   b) перезапись ключа пользовательскими данными в результате срабатывания уязвимости CWE-123 (Write-what-where Condition, запись произвольных данных в произвольную область);
5) программа, реализующая клавиатурный фильтр (сниффер, «клавиатурный шпион»);
6) программа, использующая одинаковый генератор случайных чисел без повторной инициализации для генерации ключа и случайных данных с последующей отправкой в открытый канал текста (зашифрованного при помощи сгенерированного ключа) и сгенерированных случайных данных.

Далее подробно рассматривается тест 6 как наиболее репрезентативный.

Тест 6 представляет собой программу, которая выполняет шифрование текста алгоритмом AES и записывает зашифрованный текст в файл. Для того чтобы расшифровать содержащийся в файле текст, необходим ключ, который в зашифрованном виде также записан в файле. В свою очередь, для того чтобы выполнить расшифровку ключа, необходимо ввести пароль, с помощью которого был зашифрован ключ перед записью в файл.

Блок-схема алгоритма формирования результирующего буфера, который будет записан в файл и который содержит зашифрованные ключ, текст и случайно сгенерированные данные, представлена на рисунке Рис. 11. Алгоритм содержит следующие основные шаги:

1) получение пароля (Password) и входного текста (Input Text);
2) инициализация генератора случайных чисел текущим временем;
3) генерирование ключа для шифрования входного текста с помощью генератора случайных чисел из шага 2;
4) генерирование двух раундовых ключей алгоритмом расширения ключа (Key Expansion) на основе введенного пароля (Password) и сгенерированного на шаге 3 ключа. Первый раундовый ключ будет использоваться для шифрования ключа, сгенерированного на шаге 3, а второй – введенного текста;
5) шифрование алгоритмом AES второго ключа, используя раундовый ключ, полученный по введенному паролю;
6) шифрование алгоритмом AES введенного текста (Input Text), используя раундовый ключ, полученный по случайно сгенерированному на шаге 3 ключу;

7) объединение шифров, полученных на шагах 5 и 6, в одном буфере (Result Buffer);
8) дописывание случайных данных (Random Data) в конец результирующего буфера, используя для этого проинициализированный на шаге 2 генератор случайных чисел.

После этого результирующий буфер сохраняется в файл. Так как для заполнения результирующего буфера случайными данными используется тот же самый генератор случайных чисел, что и для создания ключа шифрования, этот ключ может быть восстановлен, а текст расшифрован.

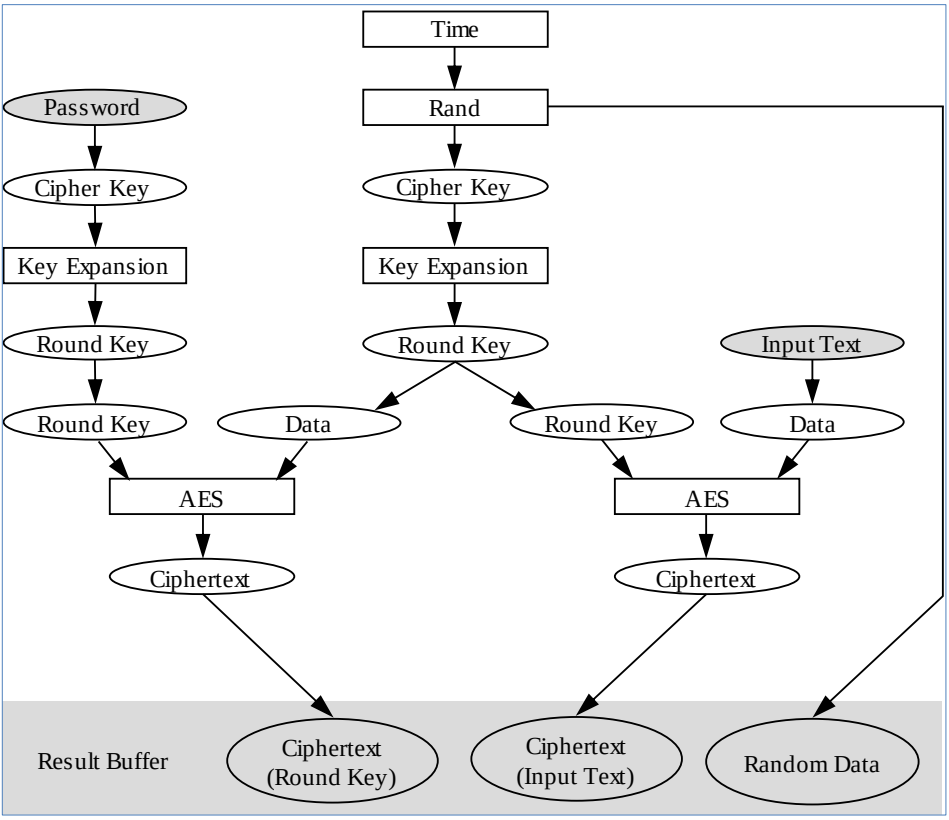

Рис. 11. Блок-схема алгоритма формирования результирующего буфера
Fig. 11. The block diagram of the algorithm for the constructing of the resulting buffer

7. Заключение

В работе представлен программный комплекс для выявления НДВ в условиях отсутствия исходного кода. Комплекс применим для ПО различных классов и различных процессорных архитектур. В его состав входят среда контролируемого выполнения на базе эмулятора QEMU, среда анализа бинарного кода ТРАЛ, пакет инструментов разработчика QEMU QDT. В совокупности эти средства обеспечивают ускоренное получение практических результатов анализа бинарного кода даже в тех ситуациях, когда изначально отсутствует возможность проведения динамического анализа.

Набор инструментов автоматизации QDT позволяет ускорить процесс разработки ВМ с помощью автоматической генерации анализатора машинных команд, заготовок моделей процессора, устройств и ВМ. Язык I3S и одноимённый инструмент облегчают написание семантики машинных команд. Инструмент i2c позволяет автоматически протестировать

наибольшую часть семантики и оценить покрытие. Графический интерфейс пользователя облегчает восприятие и редактирование связей между компонентами большой ВМ.

Для ускорения этапа ручного анализа разработано высокоуровневое иерархическое представление алгоритма программы на основе блок-схем и алгоритм его построения. Разработанное представление основывается на гиперграфе и позволяет выполнять анализ потока данных в автоматическом и ручном режимах. Его иерархическая организация делает возможным исследование свойств алгоритма программы на различных уровнях детализации и решение задач обнаружения НДВ, например, утечек конфиденциальных данных. Также предложен подход к повышению качества полученного представления алгоритма с помощью объединения отдельных потоков данных в один, связывающий логические модули алгоритма.

Основным направлением развития представленного комплекса станет его интеграция с другими средствами анализа, такими как фаззер, статический анализатор бинарного кода, анализатор сетевого трафика и др. Такая интеграция позволит в перспективе выстраивать сложные технологические цепочки из различных средств анализа, адаптируясь под требования процессов разработки, сертификации, подтверждения доверия во время эксплуатации ПО.

Список литературы / References

[1]. The IDA Pro disassembler and debugger. URL http://www.hex-rays.com/idapro/, accessed 20.11.2019.

[2]. NSA, Ghidra is a software reverse engineering (SRE) framework. NSA. URL https://github.com/NationalSecurityAgency/ghidra, accessed 20.11.2019.

[3]. D. Brumley, I. Jager, T. Avgerinos, E. J. Schwartz. BAP: A Binary Analysis Platform. Lecture Notes in Computer Science, vol. 6806, 2011, pp. 463-469.

[4]. Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel,G. Vigna. SoK: (State of) The Art of War: Offensive Techniques in Binary Analysis. IEEE Symposium on Security and Privacy, 2016, pp. 138-157.

[5]. ESIL: Radare2 book. URL https://radare.gitbooks.io/radare2book/content/disassembling/esil.html, accessed 20.11.2019.

[6]. 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 / 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, vol. 26, issue 1, 2014, pp. 251-276. DOI: 10.15514/ISPRAS-2014-26(1)-8.

[7]. Aslanyan H.K. Plarform for interprocedural static analysis of binary code. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 89-100. DOI: 10.15514/ISPRAS-2018-30(5)-5.

[8]. Ефимов В.Ю., Беззубиков А.А., Богомолов Д.А., Горемыкин О.В., Падарян В.А. Автоматизация разработки моделей устройств и вычислительных машин для QEMU. Труды ИСПРАН, том 29, вып. 6, 2017 г., стр. 77-104. DOI: 10.15514/ISPRAS-2017-29(6)-4 / Efimov V.Yu., Bezzubikov A.A., Bogomolov D.A., Goremykin O.V., Padaryan V.A. Automation of device and machine development for QEMU. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 6, 2017, pp. 77-104 (In Russian). DOI: 10.15514/ISPRAS-2017-29(6)-4.

[9]. А.И. Гетьман, Ю.В. Маркин, Д.О. Обыденков, В. А. Падарян. Архитектура системы глубокого разбора сетевого трафика. Системный администратор, том 1, вып. 2, 2018 г., стр. 83-87 / A.I. Get'man, Yu.V. Markin, D.O. Obidenkov, V. A. Padaryan. An architecture of deep packet inspection system. Sistemnyj administrator, vol. 1, issue 2, 2018, pp. 83-87 (in Russian).

[10]. Bellard F. QEMU, a fast and portable dynamic translator. In Proc. of the USENIX Annual Technical Conference, 2005, pp. 41-46.

[11]. Bezzubikov, N. Belov, K. Batuzov. Automatic dynamic binary translator generation from instruction set description. In Proc. of the 2017 Ivannikov ISPRAS Open Conference, 2017, pp. 27-33.

[12]. ARM Architecture Reference Manual (ARMv8, for ARMv8-A architecture profile). URL https://static.docs.arm.com/ddi0487/ea/DDI0487E_a_armv8_arm.pdf, accessed 20.11.2019.

[13]. Reid. Trustworthy specifications of ARM v8-A and v8-M system level architecture. In Proc. of Formal Methods in Computer-Aided Design, 2016, pp. 161-168.

[14]. I3S (Instruction Set Semantics Specification) Translator. URL https://github.com/ispras/I3S, accessed 20.11.2019.

[15]. Колтунов Д.С., Ефимов В.Ю., Падарян В.А. Автоматизированное тестирование фронтенда транслятора TCG для QEMU. Труды ИСП РАН, том 31, вып. 5, 2019 г., стр. 7–24 / Koltunov D.S., Efimov V.Y., Padaryan V.A. Automated testing of a TCG frontend for QEMU. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 5, 2019 г., pp. 7-24 (in Russian). DOI: 10.15514/ISPRAS–2019–31(5)–1.

[16]. А.И. Аветисян, К.А. Батузов, В.Ю. Ефимов, В.А. Падарян, А.Ю. Тихонов. Применение программных эмуляторов для полносистемного анализа бинарного кода мобильных платформ. Проблемы информационной безопасности. Компьютерные системы, №4, 2015 г., стр. 187-194 / Avetisyan A.I., Batuzov K.A., Efimov V.Y., Padaryan V.A., Tikhonov A.Y. Whole system emulators for mobile platform binary code analysis. Information Security Problems. Computer Systems, №4, 2015, pp. 187-194 (in Russian).

[17]. V. Efimov, V. Padaryan. Peripheral Device Register Support for Source Code Boilerplate Generator of QEMU Development Toolkit. In Proc. of the 2018 Ivannikov Memorial Workshop (IVMEM), 2018, pp. 36-39.

[18]. S. Sargsyan, J. Hakobyan, M. Mehrabyan, M. Mishechkin, V. Akozin, S. Kurmangaleev. ISP-Fuzzer: Extendable Fuzzing Framework. In Proc. of the of 2019 Ivannikov Memorial Workshop (IVMEM), 2019, pp. 68-71.

[19]. А.Н. Федотов. Метод оценки эксплуатирумости программных дефектов. Труды ИСП РАН, том 28, вып. 4, 2016 г., стр. 137-148 / A.N. Fedotov. Method for exploitability estimation of program bugs. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 4, 2016. pp. 137-148 (in Russian). DOI: 10.15514/ISPRAS-2016-28(4)-8.

[20]. Падарян В.А. О представлении результатов обратной инженерии бинарного кода. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 31-42 / Padaryan V.A. Automated vulnerabilities exploitation in presence of modern defense mechanisms. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 3, 2017. pp. 31-42 (in Russian). DOI: 10.15514/ISPRAS-2017-29(3)-3

[21]. S. Horwitz, T. Reps, D. Binkley. Interprocedural Slicing Using Dependence Graphs. ACM Transactions on Programming Languages and Systems, vol. 12, no. 1, 1990, pp. 26-60.

[22]. J. Ferrante, K. J. Ottenstein, J. D. Warren. The Program Dependence Graph and Its Use in Optimization. ACM Transactions on Programming Languages and Systems, vol. 9, no. 3, 1987, pp. 319-349.

[23]. В. А. Падарян, М. А. Соловьев, А. И. Кононов. Моделирование операционной семантики машинных инструкций. Программирование, том 37, № 3, 2011 г., стр. 50-64 / V. A. Padaryan, M. A. Solov'ev, A. I. Kononov. Simulation of operational semantics of machine instructions. Programming and Computer Software, vol. 37, Issue 3, 2011, pp 161–170.

[24]. Соловьев М.А., Бакулин М.Г., Горбачев М.С., Манушин Д.В., Падарян В.А., Панасенко С.С. О новом поколении промежуточных представлений, применяемом для анализа бинарного кода. Труды ИСП РАН, том 30, вып. 6, 2018 г., стр. 39-68 / Solovev M.A., Bakulin M.G., Gorbachev M.S., Manushin D.V., Padaryan V.A., Panasenko S.S. Next generation intermediate representations for binary code analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 39-68 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-3.

[25]. M. Jung, S. Kim, H. Han, J. Choi, S. Kil Cha. B2R2: Building an Efficient Front-End for Binary Analysis. In Proc. of the NDSS Workshop on Binary Analysis Research, 2019.

[26]. T. Dullien, S. Porst. REIL: A platform-independent intermediate representation of disassembled code for static code analysis. In Proc. of the CanSecWest Applied Security Conference, 2009, 7 p.

[27]. S. Alam, R.N. Horspool, I. Traore. MAIL: Malware Analysis Intermediate Language: A Step Towards Automating and Optimizing Malware Detection. In Proc. of the 6th International Conference on Security of Information and Networks, 2013, pp. 233–240.

[28]. A formal specification for BIL: BIL Instruction Language, 2015. URL https://github.com/BinaryAnalysisPlatform/bil/blob/master/bil.tex, accessed 20.11.2019.

[29]. D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, P. Saxena. BitBlaze: A New Approach to Computer Security via Binary Analysis. Lecture Notes in Computer Science, vol. 5352, 2008, pp. 1-25.
[30]. Sepp, B. Mihaila, A. Simon. Precise Static Analysis of Binaries by Extracting Relational Information. In Proc. of the 18th Working Conference on Reverse Engineering, 2011, pp. 357-366.
[31]. N. Nethercote, J. Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentatio. SIGPLAN Notices, vol. 42, no. 6, 2007, pp. 89-100.
[32]. E. J. Schwartz, J. Lee, M. Woo, D. Brumley. Native x86 Decompilation Using Semantics-preserving Structural Analysis and Iterative Control-flow Structuring. In Proc. of the 22Nd USENIX Conference on Security, 2013, pp. 353-368.
[33]. K. Yakdan, S. Dechand, E. Gerhards-Padilla, M. Smith. Helping Johnny to Analyze Malware: A Usability-Optimized Decompiler and Malware Analysis User Study. In Proc. of the IEEE Symposium on Security and Privacy (SP), 2016, pp. 158-177.
[34]. Retargetable Decompiler. URL https://retdec.com/, accessed 20.11.2019.
[35]. T. Ben-Nun, A. S. Jakobovits, T. Hoefler. Neural Code Comprehension: A Learnable Representation of Code Semantics. In Proc. of the 32Nd International Conference on Neural Information Processing Systems, 2018, pp. 1-13.
[36]. O. Katz, Y. Olshaker, Y. Goldberg, E. Yahav. Towards Neural Decompilation. arXiv preprint arXiv:1905.08325, 2019.
[37]. P. Lestringant, F. Guihéry, P.-A. Fouque. Automated Identification of Cryptographic Primitives in Binary Code with Data Flow Graph Isomorphism. In Proc. of the 10th ACM Symposium on Information, Computer and Communications Security, 2015, pp. 203–214.
[38]. D. Caselden, A. Bazhanyuk, M. Payer, S. McCamant, D. Song. HI-CFG: Construction by Binary Analysis and Application to Attack Polymorphism. Lecture Notes in Computer Science, vol. 81-34, 2013, pp. 164-181.
[39]. Slowinska, T. Stancescu, H. Bos. Howard: A Dynamic Excavator for Reverse Engineering Data Structures. In Proc. of the NDSS Symposium, 2011, 20 p.
[40]. Z. Lin, X. Zhang, D. Xu. Automatic Reverse Engineering of Data Structures from Binary Execution. In Proc. of the 11th Annual Information Security Symposium, 2010, Article no. 5.
[41]. G. Ramalingam, J. Field, F. Tip. Aggregate Structure Identification and Its Application to Program Analysis. In Proc. of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 1999. Pp. 119–132.
[42]. R. Milner. A theory of type polymorphism in programming. Journal of Computer and System Sciences, vol. 17, 1978, pp. 348-375.
[43]. Довгалюк П.М., Макаров В.А., Падарян В.А., Романеев М.С., Фурсова Н.И. Применение программных эмуляторов в задачах анализа бинарного кода. Труды ИСП РАН, том 26, вып. 1, 2014 г., стр. 277-296 / Dovgalyuk P.M., Makarov V.A., Padaryan V.A., Romaneev M.S., Fursova N.I. Application of software emulators for thebinary code analysis. Trudy ISP RAN/Proc. ISP RAS, vol, 26, issue 1, 2014, pp. 277-296 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-9.
[44]. C. K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, K. Hazelwood. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. ACM SIGPLAN Notices, vol. 40, no. 6, 2005, pp. 190-200.
[45]. D. Bruening, S. Amarasinghe. Efficient, transparent, and comprehensive runtime code manipulation. PhD thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, 2004.
[46]. V.A. Padaryan, I.N. Ledovskikh. On the Representation of Results of Binary Code Reverse Engineering. Programming and Computer Software, vol. 44, issue 3, 2018, pp. 200–206.

Информация об авторах / Information about authors

Александр Борисович БУГЕРЯ – кандидат физико-математических наук, старший научный сотрудник ИПМ им. М.В. Келдыша РАН, программист в ИСП РАН. Сфера научных интересов: методы и средства анализа бинарного кода программ, восстановление алгоритмов по бинарному коду, параллельное программирование: модели, языки, алгоритмы, инструментарий; распределенные системы, построение компиляторов, системное программирование.

Alexander Borisovich BUGERYA – PhD in Physical and Mathematical Sciences, Senior Researcher in Keldysh Institute of Applied Mathematics; Programmer at ISP RAS. His research interests include methods and tools for binary code analysis, recovery of program algorithm by its binary code, parallel programming: models, languages, algorithms and tools; distributed computing systems, compiler construction, system programming.

Василий Юрьевич ЕФИМОВ – младший научный сотрудник ИСП РАН. Область научных интересов: двоичная трансляция, программная эмуляция, генерация кода, инструментальные средства разработки программного обеспечения.

Vasilii Yur'evich EFIMOV – Junior Researcher at ISP RAS. Research interests: binary translation, software emulation, code generation, software development tools.

Иван Иванович КУЛАГИН – кандидат технических наук, инженер ИСП РАН. Область научных интересов: построение компиляторов, анализ бинарного кода, оптимизирующие компиляторы, полиэдральная компиляция, генерация кода, модели параллельного программирования.

Ivan Ivanovich KULAGIN – Candidate of Technical Science (PhD), Software Engineer at ISP RAS. Research interests: compiler construction, binary code analysis, compiler optimizations, polyhedral compilation, code generation, parallel programming models.

Вартан Андроникович ПАДАРЯН – кандидат физико-математических наук, ведущий научный сотрудник ИСП РАН, доцент кафедры системного программирования ВМК МГУ. Сфера научных интересов: компиляторные технологии, анализ бинарного кода, компьютерная безопасность, высокопроизводительные вычисления.

Vartan Andronikovich PADARYAN – Candidate of Physical and Mathematical Sciences, Leading Researcher at ISP RAS, Associate Professor of the Department of system programming of CMC of Lomonosov Moscow State University. Research Interests: compiler technologies, binary code analysis, cybersecurity, high performance calculating.

Михаил Александрович СОЛОВЬЕВ – кандидат физико-математических наук, старший научный сотрудник ИСП РАН, старший преподаватель кафедры системного программирования ВМК МГУ. Сфера научных интересов: абстрактная интерпретация, промежуточные представления, анализ бинарного кода, архитектура микропроцессоров.

Mikhail Aleksandrovich SOLOVEV – Candidate of Physical and Mathematical Sciences, senior researcher at ISP RAS, senior lecturer of the Department of system programming of CMC of Lomonosov Moscow State University. Research Interests: abstract interpretation, intermediate representations, binary code analysis, microprocessor architecture.

Андрей Юрьевич ТИХОНОВ – преподаватель МГТУ им. Н.Э. Баумана. Сфера научных интересов: компиляторные технологии, программная эмуляция, сетевые технологии.

Andrei Yur'evich TIKHONOV – lecturer at BMSTU. Research Interests: compiler technologies, software emulation, network technologies.

DOI: 10.15514/ISPRAS-2019-31(6)-4

CC BY

Декодирование машинных команд в задаче абстрактной интерпретации бинарного кода

[1,2] *М.А. Соловьев, ORCID: 0000-0002-0530-6442 <icee@ispras.ru>*
[1] *М.Г. Бакулин, ORCID: 0000-0002-8569-7382 <bakulinm@ispras.ru>*
[2] *С.С. Макаров, ORCID: 0000-0003-0077-237X <smakarov@ispras.ru>*
[2] *Д.В. Манушин, ORCID: 0000-0001-8985-4114 <dman95@ispras.ru>*
[1,2] *В.А. Падарян, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*

Аннотация. Программный инструментарий для работы с бинарным кодом востребован не только разработчиками: невозможно добиться достаточной безопасности современных программ без изучения свойств исполняемого кода. Базовым компонентом такого инструментария является декодер машинных команд. У разных процессорных архитектур реализации декодеров разнородны, результаты разбора команд несовместимы, а сопровождение затруднительно из-за повсеместной практики реализовывать декодеры в виде каскадов ветвлений. Дальнейший анализ бинарного кода (на уровне потоков данных и управления, символьная интерпретация и др.) оказывается непереносимым между различными процессорными архитектурами из-за ограничений и особенностей реализации декодеров. В статье предлагается подход к декодированию машинных команд, основанный на внешних спецификациях. Отличительной особенностью подхода является оригинальный способ представления декодированной команды в универсальном (т.е. не отличающемся от одной архитектуры к другой) виде. Декодирование осуществляется при помощи разработанной авторами абстрактной стековой машины. Несмотря на неизбежные накладные расходы, обусловленные большей универсальностью подхода, прототип реализации показывает скорость разбора лишь в 1,5-2,5 раза уступающую традиционным декодерам, с учетом времени разбора спецификации и формирования служебных структур данных. Предлагаемый подход к организации декодирования позволит в перспективе развернуть единый стек программных средств анализа бинарного кода, применимый к различным процессорным архитектурам. В статье обсуждаются вопросы дальнейшей трансляции декодированных команд в машинно-независимое промежуточное представление для анализа их операционной семантики и проведения абстрактной интерпретации. Приведены практически полезные примеры интерпретации: конкретная интерпретация для эмуляции бинарного кода и «направляющая» интерпретация, позволяющая увязать идею абстрактной интерпретации с задачей углубленного автоматического анализа отдельных путей в программе.

Ключевые слова: абстрактная интерпретация; анализ бинарного кода; динамический анализ; компиляторные технологии; обратная инженерия программного обеспечения; символьное выполнение; статический анализ

Для цитирования: Соловьев М.А., Бакулин М.Г., Макаров С.С., Манушин Д.В., Падарян В.А. Декодирование машинных команд в задаче абстрактной интерпретации бинарного кода. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 65–88. DOI: 10.15514/ISPRAS–2019–31(6)–4

Благодарности: Работа поддержана грантом РФФИ № 18-07-01256.

Decoding of machine instructions for abstract interpretation of binary code

[1,2] *M.A. Solovev, ORCID: 0000-0002-0530-6442 <icee@ispras.ru>*
[1] *M.G. Bakulin, ORCID: 0000-0002-8569-7382 <bakulinm@ispras.ru>*
[2] *S.S. Makarov, ORCID: 0000-0003-0077-237X <smakarov@ispras.ru>*
[2] *D.V. Manushin, ORCID: 0000-0001-8985-4114 <dman95@ispras.ru>*
[1,2] *V.A. Padaryan, ORCID: 0000-0001-7962-9677 <vartan@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*

Abstract. Not only developers require tools that work with binary code: it is impossible to achieve sufficient security in contemporary software without inspecting its properties at this level. A key component of binary code analysis toolset is the instruction decoder. Different instruction set architectures give rise to decoders that are differently structured, the decoding results are incompatible, and maintenance is hindered because of the ubiquitous practice of implementing decoders as cascades of conditional operators. Further binary code analysis (control and data flows, symbolic interpretation, etc.) cannot easily be ported from one target architecture to another because of limitations and peculiarities of decoder implementations. In this paper, we propose an approach to decoding machine instructions that is based on external specifications. The main distinction is an original way of representing the decoder instruction universally, i.e. in a way that does not differ from one architecture to another. The decoding process is handled by an abstract stack machine we have developed. Despite the inevitable overhead stemming from the approach's universality, an implementation prototype displays only 1.5-2.5 times slowdown compared to traditional decoders; the measurements include time required to parse the specification and build the required data structures. The proposed approach to organizing decoding would allow, in the long run, to establish a unified stack of binary code analysis tools that would be applicable to different instruction set architectures. The paper further discusses questions of translating the decoded instructions into a machine-neutral internal representation for analyzing their operational semantics and carrying out abstract interpretation. We give examples of practically useful interpretations: the concrete interpretation and a "directing" interpretation that allows to tie the idea of abstract interpretation with the problem of deeper automatic analysis of individual paths in a program.

Keywords: abstract interpretation; binary code analysis; dynamic analysis; compiler technologies; software reverse engineering; static analysis; symbolic execution

For citation: Solovev M.A., Bakulin M.G., Makarov S.S., Manushin D.V., Padaryan V.A. Decoding of machine instructions for abstract interpretation of binary code. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019, pp. 65-88 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-4

Acknowledgements: This work was supported by RFBR grant no. 18-07-01256.

1. Введение

В исторической ретроспективе задачи, связанные с бинарным кодом, решались в основном в контексте разработки низкоуровневого системного программного обеспечения (ПО), такого как ядра операционных систем, драйверы, компоненты встраиваемого ПО микроконтроллеров и т.п. С развитием языков программирования высокого уровня, улучшением качества компиляторов и наращиванием производительности вычислительных систем интерес к бинарному коду постепенно сокращался. Однако в последние десятилетие тема анализа и преобразования бинарного кода вновь стала крайне актуальной. Не в последнюю очередь это связано с увеличением степени внимания к безопасности ПО и качеству кода.

Одна из важнейших задач – оценка критичности дефектов в ПО – в большинстве случаев может быть решена только на уровне бинарного кода. Так, вопрос о том, приводит ли

неопределенное поведение в программе на языке Си к эксплуатируемой уязвимости нельзя решить, оставаясь в рамках модели абстрактной машины этого языка: например, оценка последствий переполнения буфера требует учитывать то, каким образом компилятор разместил этот буфер и другие переменные, какой бинарный код он сгенерировал для соответствующих обращений. Из-за этих факторов можно наблюдать взрывное появление систем анализа бинарного кода с целью обнаружения и оценки критичности уязвимостей, в т.ч. основанных на идеях символьного выполнения.

Кроме того, оценка безопасности ПО, поставляемого без исходных кодов, однозначно требует наличия развитого инструментария анализа бинарного кода. В силу того, что бинарный код более объемен и неудобен для понимания человеком, чем соответствующий исходный код на языке программирования высокого уровня, к этому инструментарию предъявляются повышенные требования с точки зрения структуризации бинарного кода и его представления в виде, пригодном для экспертной оценки человеком.

В соответствии с обзором типовых сценариев анализа бинарного кода (как связанных с обеспечением безопасности, так и иных сценариев), проведенным авторами ранее в [1], можно выделить три их группы, основываясь на требованиях к уровню представления анализируемого кода:

- сценарии, в которых требуется определять границы машинных команд в сегменте кода, проводить их декодирование и поверхностный анализ, например, итеративное дизассемблирование [2], некоторые разновидности двоичного динамического инструментирования [3], онлайн-анализ с применением аппаратной виртуализации [4];
- сценарии, в которых требуется отслеживать зависимости по данным и управлению, индуцируемые машинными командами, например, построение срезов (слайсинг) [5], анализ помеченных данных [6], интерактивное восстановление схемы работы алгоритма [7];
- сценарии, в которых требуется анализировать операционную семантику машинных команд и более крупных функциональных блоков, например, задачи поиска и оценки программных дефектов и активации условных переходов [8], а также многие оптимизационные преобразования, востребованные в классической и JIT-компиляции, а также в динамической двоичной трансляции [9].

Нетрудно заметить, что каждая следующая группа сценариев в этом списке предъявляет нарастающие требования к уровню представления анализируемого кода. Например, для того чтобы описать операционную семантику команды, необходимо предварительно ее декодировать. С другой стороны, описанная операционная семантика команды содержит информацию о возникающих при ее выполнении зависимостях по данным и управлению.

Таким образом, отдельный интерес представляет задача декодирования машинных команд, а именно определение границ команд и представление каждой из них в виде, пригодном для дальнейшего анализа. Такой вид, как минимум, должен включать код операции и описание операндов. Традиционный подход к декодированию машинных команд предполагает для каждой целевой процессорной архитектуры реализовывать отдельный программный модуль декодирования. Результатом работы такого модуля будет являться структура, описывающая декодированную команду, причем конкретный вид структуры будет зависеть от целевой процессорной архитектуры. Краткий обзор устройства декодеров, входящих в состав известных инструментов, работающих на уровне бинарного кода, дан во разд. 2 статьи.

В разд. 3 предлагается подход к декодированию, где не только кодировки команд, но и их структура (возможные коды операций, модификаторы, режимы операндов) задаются

внешними спецификациями, что позволяет единообразно осуществлять декодирование машинных команд разнородных процессорных архитектур.

Для решения задач, относящихся ко второй и третьей группе сценариев, как правило требуется трансляция полученных команд в архитектурно-независимое промежуточное представление. То, каким образом осуществляется такая трансляция, описано в разд. 4.

В разд. 5 изложен разработанный способ и кратко описана спроектированная программная инфраструктура для проведения абстрактной интерпретации на базе архитектурно-независимого промежуточного представления.

Разд. 6 содержит заключение и перечисление направлений дальнейших работ.

2. Декодирование машинных команд в существующих инструментах

В подавляющем большинстве инструментов, которые работают с бинарным кодом, декодирование команд реализовано в виде отдельных программных модулей. Эти модули разбирают кодировки команд и представляют результат в виде структур, вид которых определяется целевой процессорной архитектурой.

Среди библиотек, которые предоставляют возможность декодирования команд различных процессорных архитектур, наиболее известными являются библиотека libopcodes, входящая в состав пакета binutils [10] (и соответствующий инструмент печати декодированных команд objdump), и декодер Capstone [11].

В binutils результатом декодирования команды является структура *disassemble_info*, содержащая признак успешного разбора, класс команды с точки зрения наличия передачи управления и доступа к памяти, и непрозрачную часть, которая не описывается в доступном пользователю программном интерфейсе и различается от одной целевой архитектуры к другой. После успешного декодирования команды структуру *disassemble_info* можно передать в одну из функций печати команд (они существуют по одной для каждой поддерживаемой архитектуры) для получения текстового ассемблерного представления. Отдельно получить информацию о мнемонике и операндах команды нельзя. Разные модули декодирования написаны на языке Си и организованы по-разному. Так, модуль поддержки RISC-V управляется таблицей, содержащей маски битовых полей, на основании которых определяется тип команды и ее операнды. Модуль поддержки x86 состоит из перемежающихся таблиц и отдельных функций (как правило, выглядящих как каскады ветвлений), осуществляющих рекурсивный разбор отдельных частей команды. В текущей версии binutils (2.33.1) он состоит из 16583 строк кода без учета подключаемых заголовков.

Capstone применяет подход с внешними таблицами, элементы которых описывают изменение состояния декодера. Вид этих таблиц отличается между модулями поддержки разных архитектур. Частично эти таблицы и код обхода позаимствованы из проекта LLVM [12], однако в них внесены изменения для нужд задачи декодирования (основной сценарий использования таблиц кодировок в LLVM – для кодогенерации в компиляторном тракте). Например, комплект таблиц для архитектуры в версии Capstone 4.0.1 состоит из 4007 строк сгенерированного Си-кода. Скрипты для генерации табличного кода не входят в состав Capstone, внесение изменений возможно либо вручную, либо через взаимодействие с автором библиотеки. Результат декодирования представляется в виде структуры *cs_insn*, которая содержит идентификатор мнемоники команды, ее размер в байтах, текстовый вид мнемоники и операндов (причем не каждого отдельно, а в виде единой строки). Более детальная информация доступна в виде дополнительных структур, по одной для каждой поддерживаемой процессорной архитектуры. Набор сведений в этих структурах отличается между архитектурами и

позволяет, в частности, выяснить, какие модификаторы (префиксы, суффиксы и т.п.) присутствовали в команде, а также, как именно были закодированы ее операнды.

Некоторые инструменты, работающие с бинарным кодом, вовсе не поддерживают в явном виде структуру декодированной команды. Например, эмулятор QEMU [13] сразу в процессе декодирования команды строит промежуточное представление TCG. Опять-таки, разные модули реализованы по-разному. В модуле для RISC-V имеется таблица декодирования, которая при помощи скрипта на языке Python переводится в Си-код. Модуль для x86 реализован вручную в виде набора функций, каждая из которых, по сути, представляет собой каскад операторов *switch*.

Аналогично устроен процесс как декодирования, так и обратного кодирования машинного кода в системе динамического двоичного инструментирования Valgrind [14]. Другая система инструментирования, Pin [3], использует библиотеку Intel XED [15], которая осуществляет кодирование и декодирование x86-команд. В этой библиотеке используются внешние спецификации для описания кодировок команд, однако их вид специфичен для системы кодирования x86.

Разрабатываемая в ИСП РАН среда анализа бинарного кода ТРАЛ [16] имеет отдельные модули для декодирования команд различных архитектур, но при этом результат может быть представлен в виде единой структуры *Instruction*. Эта структура содержит признак успешного декодирования, размер декодированной команды в байтах, идентификатор мнемоники, набор флагов (является ли команда передачей управления, вызовом, возвратом и т.п.), количество слотов задержки, строковый вид команды (отдельно для префиксов, мнемоники и каждого из операндов). Кроме того, каждый операнд описывается структурой, в которой содержится: значение операнда (если это константа) или элемент какого-либо адресного пространства, в т.ч. регистрового файла (если это регистр или операнд в памяти). Модули декодирования реализованы вручную на языках Си/Си++ и в конечном счете также сводятся к разбору отдельных полей кодировки по таблицам, описанным явно или в виде каскада ветвлений.

Таким образом, практически все описанные системы не используют внешние спецификации для декодирования команд, либо используют их только в рамках отдельной процессорной архитектуры. В сочетании с тем, что результаты декодирования команд представляются в разнородном виде, для углубленного исследования (например, анализ помеченных данных, поиск ошибок, символьное выполнение, инструментирование) бинарного кода с использованием этих инструментов потребуется дополнительная прослойка, обеспечивающая еще один уровень трансляции в общий вид промежуточного представления. В рамках данной работы, в сущности, исследуется вопрос возможности обобщения вида такого рода спецификаций для того, чтобы исключить или сократить объем кодовой обвязки для такого рода трансляции.

В завершение обзора отметим, что данная работа не является первой в этом направлении. Например, в статье [17] решается похожая задача, однако для DSP-процессоров, которые, как правило, имеют существенно более регулярные и простые по структуре наборы команд. В разработанной в АНБ и выложенной в открытый доступ в апреле 2019 г. системе Ghidra [18] применяются внешние спецификации, описывающие как процесс декодирования, так и последующее построение промежуточного представления. Декодирование управляется правилами, которые описывают предикат над полями кодировки. Если предикат истинный, то применяется соответствующее правило. Правило описывает текстовый вид команды и ее семантику в виде последовательности операторов промежуточного представления.

Отметим, что на момент начала работы над методами декодирования и трансляции в промежуточное представление, излагаемыми в данной статье, система Ghidra еще не

была доступна для ознакомления (в частности, наша работа [19] была опубликована за несколько месяцев до выпуска Ghidra).

3. Декодирование машинных команд по внешним спецификациям

Для того чтобы обеспечить декодирование машинных команд по внешним спецификациям, необходимо спроектировать два компонента:

- модель архитектуры набора команд (ISA), описывающую структуру команд и задающую возможные кодировки для выбранной целевой машины;
- программный компонент, осуществляющий декодирование отдельных команд по такому описанию.

При таком подходе поддержка новой архитектуры набора команд будет сводиться к подготовке ее спецификации в соответствии с разработанной моделью (т.е. пополнению некоторой базы знаний), а изменения в программный код декодера вносить не потребуется.

3.1 Модель архитектуры набора команд

При описании архитектуры набора команд необходимо учесть следующие ключевые особенности, сформулированные ранее в работе [1]:

- могут существовать несовместимые версии архитектуры набора команд (например, в MIPS), а также расширения, влияющие на разбор определенных кодировок (например, расширение “C” в RISC-V);
- кодировки команд могут иметь переменную длину, которая не известна до начала декодирования и определяется самой кодировкой команды (например, в x86 – от 1 до 15 байтов, в RISC-V с расширением “C” – 2 или 4 байта, в наборе команд T32 архитектуры ARM);
- одна и та же последовательность битов может быть декодирована по-разному в зависимости от значений управляющих битов (например, признак разрядности кода в x86, активный набор команд в ARM).

Сформулируем первую из перечисленных особенностей в более общем виде: в архитектуре набора команд присутствуют параметры, влияющие на то, как конкретная реализация этой архитектуры (т.е. конкретный процессор) воспринимает определенные кодировки. Для того чтобы поддержать эту особенность необходимо в явном виде указать эти параметры для каждой архитектуры набора команд. Таким образом, первой составляющей модели будет являться перечень таких параметров. Каждый параметр назовем *характеристикой* (*feature*) и зафиксируем его идентификатор и длину в битах. При задании значения характеристики для конкретного процессора потребуется указать битовый вектор соответствующей длины. Например, характеристиками в рамках такого определения являются:

- поддержка семейства команд AVX в x86 – логическое значение (1 бит), также определяет возможность кодирования команд при помощи VEX-префиксов;
- поддержка расширения “M” в RISC-V – логическое значение (1 бит), определяет, доступны ли в данной реализации архитектуры RISC-V команды умножения, деления и взятия остатка;
- номер версии архитектуры MIPS (release) – целое число (3 бита), влияющее на наличие или отсутствие отдельных групп кодировок команд.

Теперь перейдем к вопросу о представлении результата декодирования, что позволит сформулировать еще две составляющие модели. Напомним, что основным отличием и целью предлагаемого подхода является единообразный процесс декодирования и, как

следствие, единообразное представление его результатов (т.е. декодированных команд). В качестве такого представления предлагается структура, описывающая команду как набор *морфем* и последовательность операндов.

Под морфемой здесь понимается свойство, дающее вклад в описание поведения команды. Наиболее характерным примером морфемы является мнемоника, соответствующая коду операции. Однако во многих процессорных архитектурах мнемоника – не единственное такое свойство команды. Например, в наборе команд A32 архитектуры ARM большая часть команд может иметь поле “cond”, определяющее условие над флагами регистра состояния, при котором команда должна быть исполнена. Если это условие не выполняется, то команда отбрасывается. В ассемблерном виде это поле соответствует двухбуквенному суффиксу после мнемоники команды, например команда сложения “addal” выполняется всегда (при этом суффикс “al” может быть опущен – “add”), а команда “addcs” – только если флаг переноса содержит единицу.

Другим примером является суффикс «точка» в PowerPC – команды “add” и “add.” отличаются тем, будут ли при выполнении сложения обновляться флаги регистра состояния. Морфемами являются также суффиксы “.aq” и “.rl”, указывающие возможный порядок операций доступа к памяти в командах расширения “A” архитектуры RISC-V. Наконец, морфемой является префикс “lock”, меняющий свойства некоторых команд доступа к памяти в x86. Таким образом, набор морфем описывает операцию, которую выполняет команда, с учетом модификаторов. Понятно, что перечень морфем для каждой целевой архитектуры свой и является второй составляющей модели.

Помимо набора морфем для задания структуры декодированной команды необходим также формализм для операндов. Возможные режимы адресации существенно отличаются от одной архитектуры набора команд к другой. Например, в архитектуре RISC-V в качестве операндов могут выступать:

- регистр общего назначения (РОН) x0…x31;
- регистр для вычислений с плавающей точкой f0…f31;
- системный CSR-регистр, заданный 12-разрядным номером;
- непосредственно закодированная константа длиной 5, 6, 12 либо 20 битов;
- операнд в памяти, адресуемый при помощи одного из регистров общего назначения и 12-разрядного знакового смещения.

Видно, что с каждым из этих режимов операндов можно сопоставить набор *атрибутов*, конкретные значения которых и будут задавать операнд соответствующего режима. Так, для регистров общего назначения и регистров для вычислений с плавающей точкой единственным атрибутом будет 5-разрядный номер регистра, для CSR-регистра – 12-разрядный номер, для каждой разновидности констант – значение соответствующей разрядности. Наконец, операнду в памяти соответствуют два атрибута: номер базового регистра и значение смещения. Перечислив все возможные режимы операндов и задав для каждого из них перечень атрибутов, мы получим третью составляющую модели, а именно «выразительные средства» для описания операндов декодированных команд. В рамках зафиксированных для каждой целевой архитектуры режимов операндов декодированный операнд описывается: во-первых, идентификатором режима; во-вторых, кортежем битовых векторов, задающих конкретные значения атрибутов, соответствующих данному режиму.

Приведем несколько примеров представления декодированных команд для различных архитектур (для записи использована JSON-нотация):

- [["lwu"], ["reg", { "rid": 4 }], ["mem", { "rid": 13, "offset": 10 }]]:
  - архитектура RISC-V, ассемблерный вид: “lwu x4, 10(x13)”;

- множество морфем состоит из единственного элемента;
- имеется два операнда: с режимом “reg” (регистр общего назначения) и с режимом “mem” (операнд в памяти);

- [["ori"], ["reg", { "rid": 6 }], ["reg", { "rid": 8 }], ["imm16", { "value": 127 }]]:
  - архитектура MIPS, ассемблерный вид: “ori $6, $8, 127”;
  - множество морфем состоит из единственного элемента;
  - имеется три операнда: первый и второй с режимом “reg” (регистр общего назначения), третий с режимом “imm16” (16-разрядная непосредственно закодированная константа);
- [["subfze", "o", "."], ["reg", { "rid": 4 }], ["reg", { "rid": 2 }]]:
  - архитектура PowerPC, ассемблерный вид “subfzeo. r4, r2”;
  - множество морфем состоит из мнемоники “subfze” и модификаторов “o” и “.”;
  - имеется два операнда с режимом “reg” (регистр общего назначения);
- [["lock", "xadd"], ["mem16a32", { "sri": 3, "bri": 2, "bri_p": 1, "iri": 0, "iri_p": 0, "scale": 0, "disp": 0, "disp_sz": 0 }], ["reg16", { "rid": 6 }]]:
  - архитектура x86, ассемблерный вид “lock xadd word [edx], si”;
  - множество морфем состоит из мнемоники “xadd” и префикса “lock”;
  - имеется два операнда: операнд в памяти с режимом “mem16a32” (размер операнда 16 битов, размер адреса 32 бита) и регистр общего назначения с режимом “reg16”.

Как можно видеть, предложенный способ описания декодированной команды подходит для большого набора разнородных архитектур набора команд. Теперь мы имеем возможность более конкретно поставить задачу декодирования команд: декодер должен принимать на вход поток битов (соответствующий одной или нескольким командам, заданным полностью или частично) и текущие значения управляющих битов, влияющих на декодирование, а на выходе формировать либо структуру команды как набор морфем и последовательность операндов, а также возвращать длину кодировки в битах, либо выдавать признак ошибки, если команда не может быть декодирована.

Эту задачу можно решить отдельно для каждой целевой архитектуры набора команд (что будет соответствовать традиционному подходу к декодированию) или задать правила, по которым формируются кодировки команд, в рамках модели архитектуры. Предлагаемое решение соответствует выбору второго пути; основная мотивация такого выбора связана с тем, что поддержка и пополнение базы знаний (спецификаций целевых процессорных архитектур) представляется менее затратным процессом, чем доработка и отладка существующих и реализация новых программных компонентов.

3.2 Декодирование машинных команд

Для описания правил, по которым в рамках заданной архитектуры формируются кодировки команд, разработан формализм, основанный на абстрактной стековой машине. Состояние стековой машины характеризуется:

- содержимым стека, каждый элемент которого является битовым вектором, векторы могут иметь различную длину;
- текущим набором выданных морфем;
- текущей последовательностью выданных операндов.

Состояние стековой машины меняется в ответ на применение *правил*, которые задаются в рамках модели архитектуры набора команд. Каждое правило относится к одному из двух типов: командное или вспомогательное правило. Совокупность всех командных правил описывает допустимые кодировки команд. Вне зависимости от своего типа,

правило характеризуется количеством входных и выходных аргументов, каждый из которых – битовый вектор. Предполагается, что на момент применения правила на стеке абстрактной машины расположены значения входных аргументов правила (глубже по стеку могут располагаться произвольные дополнительные элементы), а при его успешном применении на момент выхода входные аргументы будут сняты со стека, а вместо них размещены значения выходных аргументов правила.

С точки зрения выполняемых абстрактной машиной шагов каждое правило является последовательностью *случаев*, а каждый случай, в свою очередь, последовательностью *действий*. Перечислим возможные действия и то, как их выполнение влияет на состояние абстрактной машины, а далее перейдем к случаям и целым правилам.

- Действие *APPLY-BIT-VEC-OPERATION* применяет одну из простых операций над битовыми векторами (из словаря BTOR[1], определенного в работе [20]) к операндам, расположенным на стеке. Действие выполняется успешно, если операция может быть применена к такому сочетанию операндов. В этом случае операнды снимаются со стека, а вместо них на стек помещается результат применения операции. В противном случае абстрактная машина переходит в особое состояние *REJECT,* соответствующее невозможности продолжения работы.
- Действие *APPLY-RULE* применяет указанное вспомогательное правило. Как было указано выше, входные аргументы правила находятся на стеке; если правило может быть успешно применено, то после его отработки входные аргументы будут сняты со стека, а вместо них положены выходные. В противном случае абстрактная машина переходит в состояние *REJECT*.
- Действие *CHECK-FEATURE* вычисляет логическое выражение над конкретными значениями характеристик (features) процессора. Если результат вычисления – ложь, то абстрактная машина переходит в состояние *REJECT*. Иными словами, последующие действия могут выполняться только в том случае, если описанная комбинация характеристик либо истинна, либо может оказаться истинной (в случае, когда значения каких-либо характеристик неизвестны).
- Действие *DUP* копирует элемент с указанном индексом в стеке и добавляет эту копию в качестве новой вершины стека. Если стек короче, чем указанный индекс, то абстрактная машина переходит в состояние *REJECT*.
- Действие *EMIT-MORPHEME* добавляет в набор выданных морфем указанную морфему. Данное действие всегда выполняется успешно.
- Действие *EMIT-OPERAND* добавляет в набор выданных операндов операнд с указанным режимом, значения атрибутов которого последовательно снимаются со стека. Если длины или количество атрибутов на стеке не совпадают с заданными в модели для данного режима, абстрактная машина переходит в состояние *REJECT*.
- Действие *GATHER* снимает со стека один битовый вектор и по указанной маске собирает его отдельные биты в виде нового битового вектора, который кладется на стек. Если стек пуст или вектор на его вершине имеет недостаточную длину для применения указанной маски, абстрактная машина переходит в состояние *REJECT*.
- Действие *MATCH-PATTERN* проверяет соответствие битового вектора на вершине стека с указанным шаблоном. Шаблон представляет собой последовательность битов, где каждый бит имеет либо конкретное значение (0 или 1), либо допускает любое

[1] BTOR представляет собой способ описания SMT-формул на основе словаря базовых арифметических и логических операций. Операции были отобраны для описания поведения дискретной аппаратуры в рамках задачи проверки моделей и работают над битовыми векторами различной длины. В частности, SMT-решатель Boolector использует BTOR как основное представление формул.

значение бита в данной позиции. Если стек пуст либо битовый вектор на вершине стека не соответствует шаблону, абстрактная машина переходит в состояние *REJECT*.

- Действие *POP* снимает со стека указанное количество элементов. Если на стеке меньше указанного числа элементов, абстрактная машина переходит в состояние *REJECT*.
- Действие *PUSH* кладет на вершину стека указанный константный битовый вектор. Данное действие всегда выполняется успешно.
- Действие *SCATTER* снимает со стека один битовый вектор и по указанной маске распределяет его отдельные биты в виде нового битового вектора, который кладется на стек. Если стек пуст или вектор на его вершине имеет недостаточную длину для применения указанной маски, абстрактная машина переходит в состояние *REJECT*.

Как было сказано выше, каждый случай представляет собой последовательность действий. Абстрактная машина меняет свое состояние при выполнении случая, последовательно выполняя эти действия. Если после выполнения очередного действия абстрактная машина переходит в состояние *REJECT*, то и выполнению всего случая будет соответствовать состояние *REJECT*. Иными словами, случай выполняется успешно тогда и только когда, когда все действия в нем последовательно выполняются успешно.

Наконец, при выполнении правила последовательно рассматривается каждый его случай. Абстрактная машина пытается выполнить очередной случай, и если это успешно удается (т.е. результатом не является состояние *REJECT*), то правило в целом считается успешно выполненным и полученное состояние становится состоянием после выполнения правила в целом. Иначе восстанавливается состояние на момент начала выполнения правила и происходит рассмотрение следующего случая. Если ни один случай не выполнился успешно, то и все правило в целом считается не выполнившимся успешно, и на выходе из выполнения правила абстрактная машина переходит в состояние *REJECT*. Иными словами, правило выполняется успешно тогда и только тогда, когда существует хотя бы один случай в нем, который выполняется успешно, причем тогда выполнение правила эквивалентно выполнению первого из таких случаев.

Следует обратить внимание, что упорядоченность случаев в правиле важна для более компактного задания правил, и является заимствованием из PEG-грамматик [21]. Например, зачастую встречаются кодировки команд, где какое-либо поле имеет несколько значений, интерпретируемых особым образом, и остальные значения, которые соответствуют общему случаю. Размещая частные случаи перед общими, можно решить проблему пересечения описываемых значений без дополнительных проверок.

Таким образом, наивная реализация декодера по описанным правилам предполагает перебор всех возможных случаев. В подразделе 3.4 мы покажем, каким образом можно ускорить работу декодера за счет подготовки вспомогательных структур данных.

Все командные правила в рамках одной спецификации должны иметь один и тот же набор входных и выходных аргументов. Первым входным аргументом всегда является кодировка, разбор которой производится. Последующие аргументы (которых может и не быть) соответствуют отдельным управляющим битам, которые могут влиять на декодирование. Единственным выходным аргументом является длина в битах разобранной кодировки. Все командные правила могут быть объединены в одно общее правило путем конкатенации наборов случаев. Это общее командное правило и будет выполняться при декодировании очередной команды.

Таким образом, начальное состояние абстрактной машины при декодировании очередной команды следующее:

- на вершине стека расположен битовый вектор с кодировкой одной или нескольких команд, первую из которых необходимо декодировать (поскольку в общем случае заранее известна только максимальная длина команды для заданной архитектуры, этот вектор зачастую будет иметь именно такую длину);
- глубже по стеку расположены значения отдельных управляющих битов (полей системных регистров), которые могут влиять на декодирование;
- текущий набор выданных морфем пуст;
- текущая последовательность выданных операндов пуста.

Если при выполнении общего командного правила абстрактная машина не переходит в состояние *REJECT*, то:

- на вершине стека расположен битовый вектор, значение которого – битовая длина кодировки декодированной команды;
- текущий набор выданных морфем соответствует полному набору морфем команды;
- текущая последовательность выданных операторов соответствует полной последовательности операторов команды.

Рассмотрим два примера командных правил, соответствующих кодировкам команды “andi” архитектуры RISC-V.

- Команда “andi” кодируется вектором вида “*iiiiiiiiiii*_*mmmmm*_111_*ddddd*_0010011”, где поле *ddddd* задает номер целевого РОН, поле *mmmmm* – номер исходного РОН, а поле *iiiiiiiiiii* – значение операнда-константы. Соответствующее командное правило будет принимать на входе единственный вектор *input* (входную кодировку) и выдавать на выходе длину при успешном разборе команды. Правило состоит из единственного случая со следующей последовательностью действий:
  - *MATCH-PATTERN*(“*xxxxxxxxxxx*_*xxxxx*_111_*xxxxx*_0010011”) – проверка того, что фиксированные биты кодировки соответствуют требуемым – стек имеет вид [*input*];
  - *EMIT-MORPHEME*(“andi”) – выдача морфемы – стек имеет вид [*input*];
  - *DUP*(0) – копирование входной кодировки – стек имеет вид [*input, input*];
  - *GATHER*(“000000000000_00000_000_11111_0000000”) – выделение поля *ddddd* – стек имеет вид [*input, ddddd*];
  - *EMIT-OPERAND*(“reg”) – выдача первого регистрового операнда, номер регистра снимается с вершины стека, т.е. берется из поля *ddddd* – стек имеет вид [*input*];
  - *DUP*(0) – копирование входной кодировки – стек имеет вид [*input, input*];
  - *GATHER*(“000000000000_11111_000_00000_0000000”) – выделение поля *mmmmm* – стек имеет вид [*input, mmmmm*];
  - *EMIT-OPERAND*(“reg”) – выдача второго регистрового операнда, номер регистра снимается с вершины стека, т.е. берется из поля *mmmmm* – стек имеет вид [*input*];
  - *GATHER*(“111111111111_00000_000_00000_0000000”) – выделение поля *iiiiiiiiiii* – стек имеет вид [*iiiiiiiiiii*];
  - *EMIT-OPERAND*(“imm12”) – выдача последнего операнда-константы, значение снимается с вершины стека, т.е. берется из поля *iiiiiiiiiii* – стек становится пустым;
  - *PUSH*(32) – на стек заносится длина кодировки успешно декодированной команды – стек имеет вид [32].
- При поддержке процессором расширения “C” команда “andi” может также быть закодирована в 16-разрядном виде “100_*i*_10_*rrr*_*iiiii*_01”. Эта кодировка требует,

чтобы в качестве целевого и исходного регистра фигурировал один и тот же регистр, причем его номер должен быть в диапазоне от 8 до 15 включительно, а операнд-константа не может превышать 6 битов с учетом последующего знакового расширения. Правило для этой кодировки также будет состоять из единственного случая со следующей последовательностью действий (после тире указано состояние стека при успешном выполнении действия):

- *CHECK-FEATURE*(“C”) – [*input*];
- *MATCH-PATTERN*(“100_*x*_10_*xxx_xxxxx*_01”) – [*input*];
- *EMIT-MORPHEME*(“andi”) – [*input*];
- *DUP*(0) – [*input*, *input*];
- *GATHER*(“000_0_00_111_00000_00”) – [*input*, *rrr*];
- *APPLY-BIT-VEC-OPERATOR*(extu.2) – [*input*, 00*rrr*];
- *PUSH*(8) – [*input*, 00*rrr*, 8];
- *APPLY-BIT-VEC-OPERATOR*(or) – [*input*, 01*rrr*];
- *DUP*(0) – [*input*, 01*rrr*, 01*rrr*];
- *EMIT-OPERAND*(“reg”) – [*input*, 01*rrr*];
- *EMIT-OPERAND*(“reg”) – [*input*];
- *GATHER*(“000_1_00_000_11111_00”) – [*i_iiiii*];
- *APPLY-BIT-VEC-OPERATOR*(exts.6) – [(*iiiiii*)_*i_iiiii*];
- *EMIT-OPERAND*(“imm12”) – [];
- *PUSH*(16) – [16].

3.3 Язык внешних спецификаций

Предложенный вид задания правил является достаточно удобным для реализации абстрактной машины декодирования, но явно неудобен для описания человеком. В связи с этим был разработан более язык более высоко уровня для спецификации архитектур набора команд, а также транслятор, который преобразует спецификации на этом языке в описанные в предыдущем подразделы сущности (правила, случаи, действия и т.п.).

Не приводя полную грамматику и правила интерпретации языковых конструкций, покажем общий вид языка на рис. 1. Здесь с его помощью описаны рассмотренные выше кодировки команды “andi” архитектуры RISC-V вместе с определениями морфем, режимов операндов и характеристик.

Ключевые слова *rule* и *insn* определяют, соответственно, вспомогательные и командные правила. Перед любым из них может быть указано ключевое слово *match*, которое меняет синтаксис задания правила на табличный. В приведенном примере в табличном синтаксисе заданы оба командных правила, а вспомогательное правило “RVC_REG” задано обычным способом как набор случаев и (высокоуровневых) действий. Конкретно в этом правиле случай всего один, поэтому в фигурных скобках сразу перечисляются действия. Помимо измененного синтаксиса, в табличные правила автоматически дописывается действие, возвращающее на стеке длину считанной кодировки.

```
feature C: '1;

morpheme andi;

opmode reg { rid: '5 }
opmode imm12 { imm: '12 }

match insn RV32I(input) -> (isz) [
    |iiiiiii_iiiii_mmmmm_111_ddddd_0010011| => {
        andi;
        reg { rid: d };
        reg { rid: m };
        imm12 { imm: i };
    },
]

match insn RVC64_Q1(input) -> (isz) feature C [
    |100_i_10rrr_iiiii_01| => {
        andi;
        RVC_REG(r);
        RVC_REG(r);
        imm12 { imm: 'exts.6(i) };
    },
]

rule RVC_REG(crid) -> () {
    reg { rid: 'or('exts.2(crid), 8'5)) };
}
```

Рис. 1. Фрагмент спецификации архитектуры набора команд RISC-V
Fig. 1. RISC-V ISA specification fragment

3.4 Реализация декодера

В рамках данной работы был реализован программный компонент, осуществляющий декодирование команд по внешним спецификациям. Компонент, в сущности, реализует описанную выше абстрактную стековую машину, однако имеет некоторые особенности, которые позволяют ему работать несколько быстрее:

- перед тем, как осуществлять декодирование команд, требуется задать значения для всех характеристик, т.е. перейти от архитектуры набора команд вообще к частной ее реализации (процессору);
- в рамках этого перехода статически вычисляются все значения предикатов в действиях *CHECK-FEATURE*; действия, где предикат истинный, удаляются, а действия, где предикат ложный, удаляются вместе с объемлющим случаем;
- таким образом, в полученной модификации модели описаны только те кодировки, которые соответствуют заданному набору значений характеристик, а действия *CHECK-FEATURE* не встречаются;
- если все случаи какого-либо правила начинаются с действия *MATCH-PATTERN* (характерная ситуация для табличного сравнения), то для этого правила создается специальная структура, ускоряющая выбор нужного случая.

Структура для ускорения логически представляет собой таблицу шаблонов, с каждым из которых сопоставлено число – номер случая. Структура отвечает на вопрос «какому номеру случая соответствует данный входной битовый вектор?», причем в качестве положительного ответа возвращается номер случая, а в качестве отрицательного ответа

– признак отсутствия подходящего случая. Физически эта таблица представлена в виде дерева со следующими типами вершин:

- *ACCEPT* – принимает любой вход и возвращает указанный номер случая;
- *REJECT* – не принимает никакой вход и возвращает отрицательный ответ;
- *FORK* – по очереди применяет вход к своим потомкам до тех пор пока не получит положительный ответ или потомки не закончатся (что соответствует отрицательному ответу);
- *DENSE* – выделяет по маске битовое поле из входа, интерпретирует его как номер потомка и применяет вход к нему;
- *SPARSE* – выделяет по маске битовое поле из входа, интерпретирует его как ключ в хеш-таблице, значения которой ссылаются на потомков, и применяет вход к соответствующему потомку, либо возвращает отрицательный ответ, если хеш-таблица не содержит искомый ключ.

Структура дерева строится при помощи вспомогательного компонента, который принимает на вход пары вида (битовый шаблон, номер случая), а также выбранную стратегию построения.

Линейная стратегия построения соответствует поведению абстрактной стековой машины декодирования без оптимизаций: вершина дерева имеет тип *FORK*, а каждому случаю соответствует вершина типа *ACCEPT* с соответствующим номером.

Жадная стратегия построения работает следующим образом. Сначала для каждой битовой позиции подсчитывается, в скольких шаблонах этот бит определен (т.е. его значение влияет на выбор соответствующего случая). На основании этих подсчетов формируется очередь битовых последовательностей в порядке невозрастания указанной метрики. Далее начинает применяться следующая рекурсивная процедура.

- Если множество шаблонов пусто, то формируется дерево из единственной вершины типа *REJECT*.
- Если множество шаблонов состоит из единственного тривиального шаблона (т.е. такого, в котором любой бит может иметь любое значение), то формируется дерево из единственной вершины типа *ACCEPT*, а номер соответствующего случая берется из единственного тривиального шаблона.
- В остальных случаях из очереди извлекается следующая рассматриваемая битовая позиция. Множество шаблонов разбивается на три подмножества: P_0 – те шаблоны, где на рассматриваемой битовой позиции нулевой бит, P_1 – те шаблоны, где на рассматриваемой битовой позиции единичный бит, и P_x – те шаблоны, где на рассматриваемой битовой позиции произвольный бит. При этом разбиении во множествах P_0 и P_1 бит на рассматриваемой позиции заменяется на произвольный. Процедура вызывается рекурсивно для каждого из трех множеств с оставшейся частью очереди, в результате чего формируются три поддерева T_0, T_1 и T_x. Они, в свою очередь, объединяются в общее дерево по следующим правилам.
  - Сначала объединяются поддеревья T_0 и T_1. Если оба корня – *REJECT*, то и результат объединения тоже *REJECT*.
  - Если один из корней – *DENSE*, а второй – *DENSE* с такой же маской или *REJECT* и можно объединить случаи в общую *DENSE*-вершину, у которой будет не более, чем некоторое пороговое число потомков (на настоящий момент – 256), то это объединение происходит. При этом логически (но не физически) *REJECT*-корень (если он есть) заменяется на *DENSE*-поддерево с соответствующим количеством *REJECT*-потомков. Далее два *DENSE*-поддерева объединяются путем выбора потомков то одного, то второго из них в правильном порядке в зависимости от того, где находится текущая рассматриваемая битовая позиция относительно

маски потомков.

- В остальных случаях формируется *DENSE*-поддерево с поддеревьями T_0 и T_1, а маска формируется из текущей битовой позиции.
- После того, как поддеревья T_0 и T_1 объединены в поддерево T_{01}, оно объединяется с T_x. Если одно из этих поддеревьев имеет *REJECT*-корень, то результатом объединения становится второе. В противном случае создается *FORK*-поддерево с потомками T_{01} и T_x (в данном порядке).

Наконец, *жадная разреженная* стратегия отличается от жадной тем, что при построении дерева *DENSE*-вершины, в которых более чем половина поддеревьев является *REJECT*-поддеревьями, заменяются на эквивалентные *SPARSE*-вершины (т.е. массив с «пустыми» элементами заменяется на хеш-таблицу). Эта стратегия имеет такую же амортизированную вычислительную сложность, что и жадная стратегия, однако позволяет несколько сократить объем используемой памяти.

Тестирование производительности реализованного декодера было проведено для двух наборов команд: 64-разрядная модель RISC-V и 64-разрядная модель PowerPC. В качестве входных файлов выступали «сырые» (без заголовков) секции кода исполняемых файлов и библиотек из образов Debian Linux. Для RISC-V использовался образ от 18.04.2018 г. [22], а для PowerPC – от 16.11.2019 г. [23]. Чтобы обеспечить объективность тестирования, вокруг разработанного декодера была реализована обвязка, которая выводит декодированные команды в текстовом виде в соответствии с ассемблерным синтаксисом, который принят для каждой из двух рассматриваемых архитектур. В тестировании также участвовали две сборки утилиты objdump из набора binutils [10] версии 2.33 (для RISC-V и для PowerPC), а также декодер Capstone [11] версии 4.0.1 для PowerPC и декодер RV8 [24] версии от 23.09.2018 г. для RISC-V. Каждый тестируемый инструмент запускался на каждом входном файле 10 раз подряд, после чего высчитывалось среднее время запуска. Тестирование проводилось на компьютере с процессором AMD Ryzen 5 1400 3.20ГГц и 32ГиБ оперативной памяти под управлением 64-разрядной ОС Ubuntu Linux 19.10. Результаты тестирования приведены на рис. 2 (RISC-V) и рис. 3 (PowerPC). Горизонтальная ось графиков – среднее время одного запуска в миллисекундах (логарифмическая шкала).

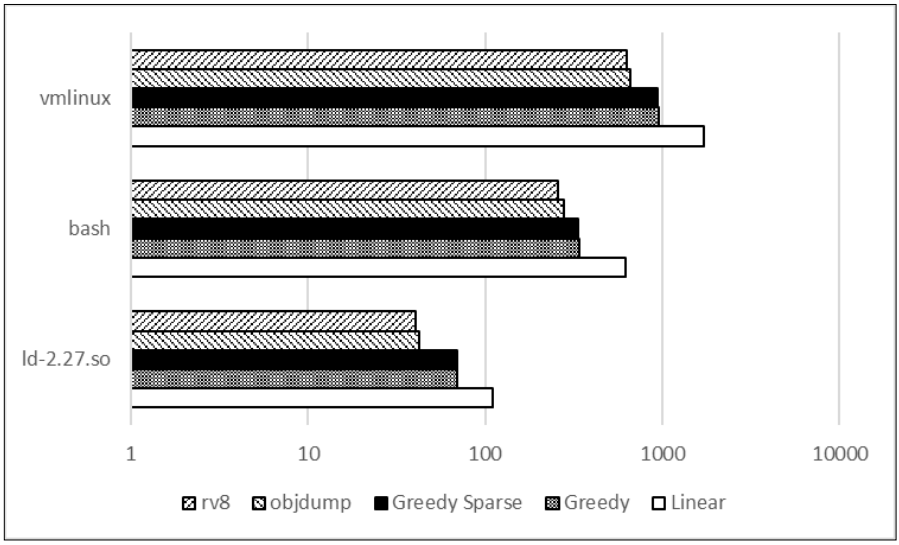

Рис. 2. Сравнение производительности декодирования на наборе команд RISC-V
Fig. 2. Decode benchmark for RISC-V ISA

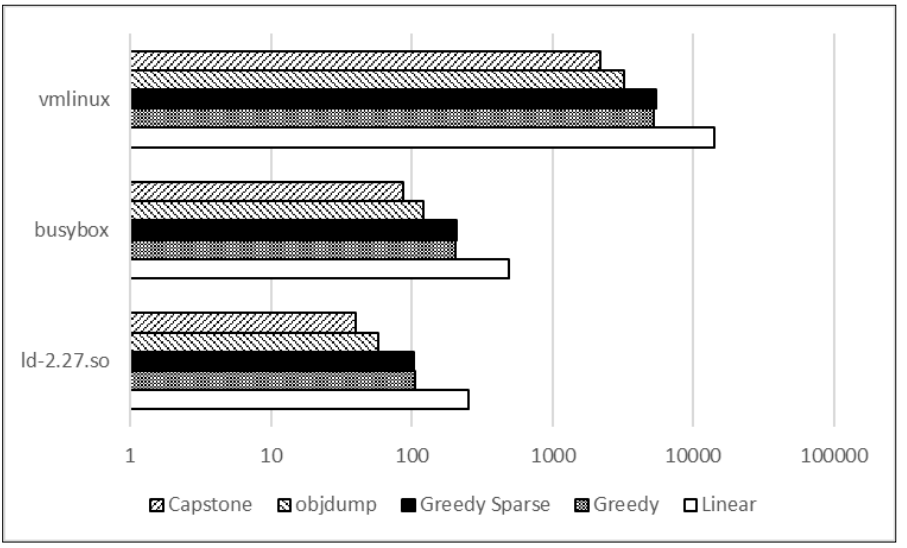

Рис. 3. Сравнение производительности декодирования на наборе команд PowerPC
Fig. 3. Decode benchmark for PowerPC ISA

Видно, что во всех случаях скорость разработанного декодера уступает реализованным вручную и оптимизированным под конкретные наборы команд традиционным декодерам. В особенности это касается случаев, когда использовалась линейная стратегия. Однако при применении жадной или жадной разреженной стратегии отставание по скорости от objdump в среднем находится в диапазоне 1,5-2,5 раза. Следует учесть, что для разработанного декодера тестировался вариант с «полным циклом»: чтение внешней спецификации на исходном языке, ее трансляция в правила для абстрактной стековой машины и дальнейшее декодирование команд по этим правилам.

Проведенное профилирование показывает, что в реализации есть пространство для дальнейшей оптимизации за счет уменьшения количества копирований и динамического выделения памяти, что дает надежду на достижение времен, близких к показываемым реализованными вручную традиционными декодерами. Кроме того, заметный вклад во время работы разработанного декодера вносит также отдельный этап формирования строкового представления команды, в то время как в традиционных декодерах этот этап совмещен с самим декодированием.

4. Трансляция в промежуточное представление

Углубленный анализ поведения бинарного кода предполагает его трансляцию в машинно-независимое промежуточное представление. В предыдущей работе [1] нами составлены требования к такому представлению и предложено реализующее их представление Pivot 2. Кратко напомним, что Pivot 2 является представлением в форме статического единичного присваивания, однако вместо φ-функции применяется подстановка значений переменных на ребрах. Все переменные представляют собой битовые векторы различной длины. Наиболее крупная единица представления – модуль – состоит из фрагментов, соответствующих функциональным блокам с единственной входной и единственной выходной вершиной.

Фрагменты могут иметь входные и выходные параметры-битовые векторы. С точки зрения внутренней структуры фрагмент представляет собой компоненту связности графа потока управления. Каждый базовый блок в нем состоит из последовательности

операторов: *CALL* (вызов фрагмента), *INIT* (инициализация переменной константным битовым вектором), *INVOKE* (применение операции над переменными-битовыми векторами), *LOAD.L* (загрузка из локального адресного пространства), *LOAD.R* (загрузка из удаленного адресного пространства), *MIX* (переименование переменных), *SLICE* (конкатенация и выделение полей битовых векторов[2]), *STORE.L* (выгрузка в локальное адресное пространство) и *STORE.R* (выгрузка в удаленное адресное пространство). Адресными пространствами в терминах Pivot 2 являются любые адресуемые области памяти, включая регистровые файлы, порты ввода-вывода и т.д. Под локальным адресным пространством понимается такое, которое может быть корректно промоделировано как буфер памяти (например, регистры общего назначения в большинстве процессоров), причем любой доступ к такому пространству завершается успешно. Удаленное адресное пространство имеет произвольную семантику доступов, некоторые из которых могут завершаться с ошибкой.

Трансляция отдельно взятой машинной команды в промежуточное представление сводится к следующему. Необходимо описать фрагмент, который моделирует операционную семантику этой команды. При этом входными параметрами фрагмента становятся значения атрибутов декодированной и представленной в описанном выше виде команды. Выбор требуемого фрагмента осуществляется на основе аннотаций, в которых для фрагментов, описывающих поведение машинных команд, указывается:

- набор морфем: все перечисленные морфемы должны входить в множество, которое сформировал декодер в результате разбора;
- режимы адресации операндов: для каждого операнда режим адресации должен совпасть с тем, который получен в результате разбора.

В качестве примера рассмотрим уже упоминавшуюся выше команду “andi” процессорной архитектуры RISC-V. Эта команда осуществляет операцию «побитовое И», входными операндами которой являются 32-, 64- или 128-разрядный РОН (в зависимости от варианта архитектуры) и 12-разрядная непосредственно закодированная константа, которая подлежит знаковому расширению до размера РОН. Результат помещается в выходной РОН (номер которого может совпадать или не совпадать с входным). На рис. 4 приведен представленный на высокоуровневом языке[3] фрагмент для этой команды для случая 64-разрядных РОН.

Аннотация *isa::insn* перед фрагментом указывает форму команды, к которой применим этот фрагмент: множество морфем должно включать морфему “addi”, команда должна иметь три операнда, первые два из которых относятся к режиму “reg”, а третий – “imm12”. При аргументах функции присутствуют аннотации *isa::subst*, указывающие, значением какого атрибута декодированной команды должен быть инициализирован соответствующий аргумент.

В процессорной архитектуре RISC-V РОН с номером 0 имеет особенный смысл: записи в него не происходят, а чтения всегда возвращают 0. Описанный фрагмент учитывает это поведение: если номер выходного регистра *rd* – нулевой, то команда игнорируется (1). В противном случае во временной переменной *a* формируется значение первого входного операнда: если это нулевой РОН, то нулевое значение, а иначе значение, считанное из адресного пространства регистров по смещению, соответствующему номеру регистра (2). Переменная *b* получает значение второго входного операнда как знаковое расширение *imm* (3). Наконец, результат операции «побитовое И» записывается в адресное пространство регистров (4).

[2] На момент публикации работы [1] для этих действий использовались отдельные операторы *CONCAT* и *EXTRACT*, которые позже были объединены в оператор *SLICE*.

[3] Синтаксис языка не является окончательным; на момент написания статьи над ним ведется активная работа.

```
#[isa::insn "andi(reg, reg, imm12)"]
fn andi(
    #[isa::subst "0.rid"]
    rd: '5,
    #[isa::subst "1.rid"]
    rs1: '5,
    #[isa::subst "2.imm"]
    imm: '12,
) {
    if zero(rd) {
        // (1)
    } else little {
        let a = if zero(rs1) { 0'64 }
                  else { regs[mul(rs1, 8'5)] };  // (2)
        let b = exts(imm);  // (3)
        regs[mul(rd, 8'5)] = and(a, b);  // (4)
    }
```

Рис. 4. Спецификация операционной семантики команды “addi” набора команд RISC-V
Fig. 4. Operational semantics specification for the “addi” instruction of the RISC-V ISA

Поскольку в разные моменты доступ даже к одним и тем же адресным пространствам может предполагать разный порядок байтов, любое обращение должно быть аннотировано конкретным порядком. В данном случае это происходит при помощи объемлющего блока *little*, предписывающего порядок little endian.

Имея фрагмент, описывающий семантику команды, транслировать ее в промежуточное представление легко: достаточно вызвать этот фрагмент с нужными параметрами. Однако такой подход не очень эффективен: видно, что для конкретного экземпляра команды “addi” большая часть вычислений может быть сделана статически (во время трансляции): могут быть удалены ветвления, константа может быть заранее расширена, а адреса в пространстве регистров заранее вычислены. В связи с этими соображениями более эффективно осуществить специализацию фрагмента путем подстановки его содержимого в объемлющий фрагмент вместе с инициализацией значений входных переменных, а затем провести оптимизационные преобразования:

- удаление общих подвыражений, в т.ч. избыточных доступов к локальным адресным пространствам;
- свертку и продвижение констант;
- удаление мертвого кода и пустых базовых блоков.

Следует отметить, что особенно заметен выигрыш от оптимизационных преобразований будет при трансляции блоков машинных команд в виде одного фрагмента: в этом случае общие подвыражения будут удаляться в рамках всего транслируемого блока. Трансляция в этом случае сводится к последовательной подстановке фрагментов, соответствующих отдельным командам, и оптимизации результирующего фрагмента. При этом, конечно, транслируемый блок должен иметь единственный вход. Если у блока несколько выходов, то потребуется возвращать из фрагмента номер выхода для дальнейшей диспетчеризации.

5. Абстрактная интерпретация промежуточного представления

Напомним, что в классическом определении [25] абстрактная интерпретация состоит из:

- полной полурешетки абстрактных состояний *A-Cont* с операцией ◦ и индуцированным ей отношением частичного порядка ≤, нижним и верхним элементами;
- отображения *Int*, переводящего набор абстрактных состояний на входных ребрах базового блока в абстрактные состояния на выходных ребрах.

В классическом определении предполагается, что отображение *Int* сохраняет порядок (или, в терминах задач потока данных, является монотонной передаточной функцией). Тогда по теореме Кнастера-Тарского [26] при итеративном применении отображения *Int* ко всем точкам программы соответствующее «глобальное» отображение для состояния всей программы имеет неподвижную точку. При этом предполагается, что решается задача оценки поведения программы в целом, т.е. задача статического анализа.

В рамках разрабатываемой инфраструктуры анализа бинарного кода предлагается в части случаев использовать более слабое понимание абстрактной интерпретации, разбив его на два.

Под *интерпретацией* в контексте данной работы будем понимать:

- состояние – произвольный тип данных, описывающий некоторые аспекты состояния анализируемой программы, и поддерживающий создание копий;
- набор передаточных функций, отображающих входное состояние либо в выходное, либо в особое «невозможное» состояние:
  - функция *edge*, соответствующая проходу по ребру;
  - функция *call_entry*, соответствующая подготовительным действиям перед вызовом фрагмента (формированию значений входных аргументов перед началом интерпретации вызванного фрагмента);
  - функция *call_exit*, соответствующая завершающим действиям после вызова фрагмента (формированию значений выходных аргументов по окончании интерпретации вызванного фрагмента);
  - функции *init*, *invoke*, *load_local*, *load_remote*, *mix*, *slice*, *store_local*, *store_remote*, соответствующие выполнению операторов *INIT*, *INVOKE*, *LOAD.L*, *LOAD.R*, *MIX*, *SLICE*, *STORE.L*, *STORE.R*.

Интерпретация также характеризуется *направлением* – прямым или обратным. Это влияет, в частности, на то, в каком порядке будут рассматриваться операторы в базовом блоке. Отметим, что декартово произведение любого набора интерпретаций также будет являться интерпретацией, причем ее состояние будет соответствовать декартову произведению состояний отдельных интерпретаций. В этом случае определим, что общее состояние будет считаться «невозможным» если хотя бы одно из состояний отдельных интерпретаций выродилось в «невозможное».

Под *монотонной интерпретацией* будем понимать интерпретацию, для которой выполнено дополнительно следующее:

- ее состояние является полной решеткой;
- все передаточные функции являются монотонными относительно частичного порядка, определяемого данной решеткой.

Можно видеть, что определение монотонной интерпретации совпадает с классическим определением абстрактной интерпретации (поскольку утверждения, что множество является полной решеткой и полной полурешёткой, эквивалентны). В монотонных интерпретациях «невозможное» состояние по определению соответствует нижнему элементу решетки состояний. Декартово произведение монотонных интерпретаций является монотонной интерпретацией.

5.1 Конкретная интерпретация

Построим в описанных терминах *конкретную интерпретацию*, т.е. такую интерпретацию, которая соответствует поведению бинарного кода при его реальном запуске на некоторой машине. В сущности, решаемая задача является задачей эмуляции бинарного кода, однако в рассматриваемом случае она решается в рамках терминологии абстрактной интерпретации.

Состояние конкретной интерпретации состоит из двух аспектов: стека, в свою очередь состоящего из отдельных кадров, и состояний каждого адресного пространства. Каждый кадр стека содержит отображение из номера локальной переменной в ее значение. Значение переменной – это либо конкретный битовый вектор, либо признак «неизвестное значение» определенной битовой длины. Состояние адресного пространства – это непрозрачный объект, который изменяется в ответ на загрузку и выгрузку.

- Метод *load* данного объекта соответствует загрузке из пространства данных с указанным размером и порядком байтов, размещенных по указанному адресу. В случае успеха возвращается полученное значение переменной, в случае ошибки – дескриптор ошибки, также в виде значения переменной. В обоих случаях значение может быть неопределенным, аналогично значениям переменных в кадрах стека.
- Метод *store* соответствует выгрузке в пространство данных с указанным значением (определенным или нет) и порядком байтов по указанному адресу. В случае ошибки возвращается дескриптор ошибки.

Если метод *load* или *store* возвращает ошибку при доступе к локальному пространству, это интерпретируется как «невозможное» состояние. Поскольку все локальные пространства должны демонстрировать единообразное поведение, существует одна реализация такого объекта для поддержания состояния локального пространства. В основе реализации лежит разреженный массив байтов, выделяемый по необходимости отдельными страницами. Каждый бит содержит либо 0, либо 1, либо неопределенное значение.

Если для какого-либо пространства на момент начала интерпретации не задан объект состояния, то записи в это пространство игнорируются, а чтения всегда возвращают неопределенные значения.

Сама конкретная интерпретация сводится к реализации описанных выше передаточных функций и является, очевидно, прямой интерпретацией. Следует отметить, что функция *invoke* реализуется путем вычисления SMT-выражений, описывающих значения выходных аргументов в виде формул над значениями входных аргументов операций.

Конкретная интерпретация не является монотонной: переменная в рамках какого-либо кадра, равно как и участок адресного пространства, могут произвольным образом менять свои значения произвольное число раз (в случае переменной для этого необходимо, чтобы место ее статического единичного присваивания находилось внутри цикла). Таким образом, для ее применения требуется некоторый компонент, который будет отслеживать текущее состояние интерпретации и положение в интерпретируемом коде. Этот компонент описан в следующем подразделе.

5.2 Конкретный исполнитель

Конкретный исполнитель осуществляет применение выбранной прямой интерпретации вдоль одного пути в анализируемом коде. В начале работы конкретному исполнителю должны быть переданы:

- Pivot-модуль, в рамках которого будет проводиться интерпретация;
- начальная точка в модуле;

- выбранная прямая интерпретация;
- состояние интерпретации, соответствующее начальной точке.

После инициализации конкретный исполнитель ведет себя как *итератор*: он может продвигаться к следующему состоянию до тех пор, пока это состояние существует (т.е. не является «невозможным»). Каждый шаг происходит следующим образом.

- Если текущая точка соответствует какому-либо оператору, то происходит вызов соответствующей передаточной функции над текущим состоянием и полученное состояние становится текущим. Если рассматриваемый оператор – *CALL*, то его выполнение распадается на три части – подготовку вызова, интерпретацию вызванного фрагмента и завершение вызова.
- Если текущая точка соответствует концу базового блока, то для каждого ребра, исходящего из этого блока, вызывается передаточная функция *edge*. Если оба состояния «невозможные», то состояние исполнителя становится «невозможным». Если «невозможное» только одно из этих состояний, то состоянием исполнителя объявляется второе. Наконец, если оба состояния оказались возможными, то выбранная интерпретация не может быть продолжена в рамках конкретного исполнителя, поскольку появился второй возможный путь. В этом случае состояние исполнителя также становится «невозможным».

Конкретный исполнитель может применяться не только с конкретной интерпретацией. Он может также использоваться для проведения произвольного анализа в рамках некоторого фиксированного пути. В этом случае требуется таким образом задать интерпретацию, чтобы только вдоль интересующего пути она имела «возможные» состояния.

Например, если проводится динамический анализ по трассе и пройденный в рамках данного выполнения путь известен, направить произвольную интерпретацию вдоль этого пути можно следующим образом. Зададим еще одну прямую интерпретацию, состояние которой представляет собой единичный тип (т.е. тип с единственным значением). Будем считать, что это единственное значение соответствует «возможному» состоянию (напомним, что «невозможное» состояние существует отдельно от выбранного типа для состояния). Все передаточные функции задаваемой интерпретации, кроме функции *edge*, будут возвращать входное состояние (они будут вызываться только на «возможном» входном состоянии и, таким образом, возвращать «возможное» выходное состояние). Функция *edge*, рассматривая трассу, должна вернуть «возможное» состояние для пройденного исходящего ребра, и «невозможное» состояние для не пройденного исходящего ребра. Объединив построенную таким образом интерпретацию с произвольной заданной прямой интерпретацией через декартово произведение, мы получаем возможность проведения прямого динамического анализа. Симметричным образом можно организовать и обратный динамический анализ вдоль заданного пути. Отметим, что построенная «направляющая» интерпретация является монотонной.

Наконец, отметим, что если имеется какая-либо монотонная интерпретация, и она при помощи конкретного исполнителя применяется в связке с «направляющей» интерпретацией вдоль каждого возможного пути, то применяя операцию «сбор» к полученным в конечной точке состояниям, мы (по определению) построим MOP-решение соответствующей задачи потока данных.

6. Заключение

В настоящий момент все изложенные в данной статье методы и подходы в той или иной степени реализованы в экспериментальной инфраструктуре для абстрактной интерпретации бинарного кода Glassfrog, разрабатываемой в ИСП РАН. Дальнейшие

работы связаны с улучшением существующих компонентов и разработкой и реализацией новых.

В части улучшения существующих компонентов планируется провести окончательную стабилизацию программного интерфейса и улучшить производительность компонента декодирования машинных команд, выполнить его интеграцию с компонентом абстрактной интерпретации.

Помимо существующего конкретного исполнителя, планируется реализовать исполнитель для проведения статического анализа, позволяющий вычислить наибольшую или наименьшую неподвижную точку для заданной монотонной интерпретации. Также планируется реализация исполнителя для символьного выполнения, поддерживающего абстрактные состояния на множестве активных путей с внешней диспетчеризацией.

Список литературы / References

[1]. Solovev M.A., Bakulin M.G., Gorbachev M.S., Manushin D.V., Padaryan V.A., Panasenko S.S. Next-generation intermediate representations for binary code analysis. Programming and Computer Software, vol. 45, issue 7, 2019, pp. 424-437. DOI: 10.1134/S0361768819070107.

[2]. Ben Khadra M.A., Stoffel D., Kunz W. Speculative disassembly of binary code. In Proc. of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems, 2016, Article No. 16.

[3]. Luk C.K., Cohn R., Muth R., Patil H., Klauser A., Lowney G., Wallace S., Reddi V.J., Hazelwood K. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. ACM SIGPLAN Notices, vol. 40, no. 6, 2005, pp. 190-200.

[4]. Ren S, Tan L, Li C, Xiao Z, Song W. Samsara: Efficient deterministic replay in multiprocessor environments with hardware virtualization extensions. In Proc. of the USENIX Annual Technical Conference, 2016, pp. 551-564.

[5]. Weiser M. Program slicing. IEEE Transactions on software engineering, vol. 10, issue 4, 1984, pp. 352-357.

[6]. Bakulin M, Klimushenkova M, Egorov D. Dynamic Diluted Taint Analysis for Evaluating Detected Policy Violations. Ivannikov ISPRAS Open Conference, 2017, pp. 22-26. DOI: 10.1109/ISPRAS.2017.00011.

[7]. Bugerya A.B., Kulagin I.I., Padaryan V.A., Solovev M.A., Tikhonov A.Yu. Recovery of High-Level Intermediate Representations of Algorithms from Binary Code. Ivannikov Memorial Workshop (IVMEM), 2019, pp. 57-63. DOI: 10.1109/IVMEM.2019.00015.

[8]. Федотов А.Н., Падарян В.А., Каушан В.В., Курмангалеев Ш.Ф., Вишняков А.В., Нурмухаметов А.Р. Оценка критичности программных дефектов в рамках работы современных защитных механизмов. Труды ИСП РАН, том 28, вып. 5, 2016, стр. 73-92 / Fedotov A.N., Padaryan V.A., Kaushan V.V., Kurmangaleev Sh.F., Vishnyakov A.V., Nurmukhametov A.R. Software defect severity estimation in presence of modern defense mechanisms. Trudy ISP RAN/Proc. ISP RAN, vol. 28, issue 5, 2016, pp. 73-92 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-4.

[9]. Muchnick S.S. Advanced Compiler Design & Implementation. Morgan Kaufmann Publishers, 1997, 856 p.

[10]. GNU binutils. URL: https://www.gnu.org/software/binutils/, accessed 25.11.2019.

[11]. Capstone. URL: http://www.capstone-engine.org/, accessed 25.11.2019.

[12]. Lattner C., Adve V. LLVM: A compilation framework for lifelong program analysis & transformation. In Proc. of the International Symposium on Code Generation and Optimization: feedback-directed and runtime optimization, 2004, pp. 75-86.

[13]. Bellard F. QEMU, a fast and portable dynamic translator. In Proc. of the USENIX Annual Technical Conference, 2005, pp. 41-46.

[14]. Nethercote N., Seward J. Valgrind: a framework for heavyweight dynamic binary instrumentation. ACM SIGPLAN Notices, vol. 42, no. 6, 2007, pp. 89-100.

[15]. Intel XED. URL: https://intelxed.github.io/, accessed 25.11.2019.

[16]. Padaryan V.A., Getman A.I., Solovyev M.A., Bakulin M.G., Borzilov A.I., Kaushan V.V., Ledovskikh I.N., Markin Yu.V., Panasensko S.S. Methods and software tools to support combined

binary code analysis. Programming and Computer Software, vol. 40, no. 5, 2014, pp. 276-287. DOI: 10.1134/
S0361768814050077.

[17]. Рубанов В.В., Михеев А.С. Интегрированная среда описания системы команд для сигнальных процессоров. Труды ИСП РАН, том 9, 2006, стр. 143-158 / Rubanov V.V, Mikheev A.S. Integrated environment for describing instruction systems of signal processors. Trudy ISP RAN/Proc. ISP RAN, vol. 8, 2006, pp. 143-158 (in Russian).

[18]. Ghidra. URL: https://www.nsa.gov/resources/everyone/ghidra/, accessed 25.11.2019.

[19]. Соловьев М.А., Бакулин М.Г., Горбачев М.С., Манушин Д.В., Падарян В.А., Панасенко С.С. О новом поколении промежуточных представлений, применяемом для анализа бинарного кода. Труды ИСП РАН, том 30, вып. 6, 2018, стр. 39-68 / Solovev M.A., Bakulin M.G., Gorbachev M.S., Manushin D.V., Padaryan V.A, Panasenko S.S. Next-generation intermediate representations for binary code analysis. Trudy ISP RAN/Proc. ISP RAN, vol. 30, issue 6, 2018, pp. 39-68. DOI: 10.15514/ISPRAS-2018-30(6)-3.

[20]. Brummayer R, Biere A, Lonsing F. BTOR: bit-precise modelling of word-level problems for model checking. In Proc. of the Joint Workshops of the 6th International Workshop on Satisfiability Modulo Theories and 1st International Workshop on Bit-Precise Reasoning, 2008, pp. 33-38.

[21]. Ford B. Parsing expression grammars: a recognition-based syntactic foundation. In Proc. of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2004, pp. 111-122.

[22]. Debian Linux RISC-V OS image. URL: https://people.debian.org/~mafm/debian-riscv64-tarball-20180418.tar.xz, accessed 25.11.2019.

[23]. Debian Linux PowerPC OS image. URL: https://cdimage.debian.org/debian-cd/current/ppc64el/iso-cd/debian-10.2.0-ppc64el-netinst.iso, accessed 25.11.2019.

[24]. RV8. URL: https://rv8.io/, accessed 25.11.2019.

[25]. Cousot P, Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In Proc. of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, 1977, pp. 238-252.

[26]. Tarski A. A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics, vol. 5, issue 2, 1955, pp. 285-309.

Информация об авторах / Information about authors

Михаил Александрович СОЛОВЬЕВ – кандидат физико-математических наук, старший научный сотрудник отдела компиляторных технологий ИСП РАН; старший преподаватель кафедры системного программирования факультета ВМК МГУ. Его научные интересы включают анализ бинарного и исходного кода, обратную инженерию ПО, операционные системы.

Mikhail Aleksandrovich SOLOVEV is a candidate of physical and mathematical sciences, senior researcher at the compiler technologies department of ISP RAS; senior lecturer at the system programming department of the faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University. His research interests include binary and source code analysis, software reverse engineering, and operating systems.

Максим Геннадьевич БАКУЛИН – младший научный сотрудник отдела компиляторных технологий ИСП РАН. Его научные интересы включают анализ бинарного и исходного кода, динамический анализ помеченных данных, символьное выполнение, эмуляцию и виртуализацию.

Maksim Gennadevich BAKULIN is a junior researcher at the compiler technologies department of ISP RAS. His research interests include binary and source code analysis, dynamic taint analysis, symbolic execution, emulation and virtualization.

Сергей Сергеевич МАКАРОВ – студент кафедры системного программирования ВМК МГУ. Его научные интересы включают анализ бинарного кода, эмуляцию и виртуализацию, операционные системы, обратную инженерию ПО.

Sergei Sergeevich MAKAROV is a student at the system programming department of the faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University. His research interests include binary code analysis, emulation and virtualization, operating systems and software reverse engineering.

Дмитрий Валерьевич МАНУШИН –аспирант кафедры системного программирования факультета ВМК МГУ. Его научные интересы включают анализ бинарного кода, анализ исходного кода, безопасность ПО.

Dmitrii Valerevich MANUSHIN is a postgraduate at the system programming department of the faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University. His research interests include binary code analysis, source code analysis and software security.

Вартан Андроникович ПАДАРЯН – кандидат физико-математических наук, ведущий научный сотрудник отдела технологий ИСП РАН; доцент кафедры системного программирования факультета ВМК МГУ. Его научные интересы включают компиляторные технологии, безопасность ПО, анализ бинарного кода, параллельное программирование, эмуляция и виртуализация.

Vartan Andronikovich PADARYAN is a candidate of physical and mathematical sciences, leading researcher at the compiler technologies department of ISP RAS; associate professor of the system programming department of the faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University. His research interests include compiler technologies, software security, binary code analysis, parallel programming, emulation and virtualization.

DOI: 10.15514/ISPRAS-2019-31(6)-5

CC BY

Исследование и разработка межпроцедурных алгоритмов поиска дефектов в исполняемом коде программ

[1,2] *Г.С. Иванов, ORCID: 0000-0002-0544-6816 <gregory@ispras.ru>*
[3] *П.М. Пальчиков, ORCID: 0000-0003-0449-2447 <pavel98@ispras.ru>*
[3] *А.Ю. Тарасов, ORCID: 0000-0002-6212-8144 <tematarasov48rus@ispras.ru>*
[3] *Г.С. Акимов, ORCID: 0000-0002-6601-4312 <akimg@ispras.ru>*
[4] *А.К. Асланян, ORCID: 0000-0002-7320-4835 <hayk@ispras.ru***Ошибка! Недопустимый объект гиперссылки.***>*
[4] *В.Г. Варданян, ORCID: 0000-0002-4899-2999 <vaag@ispras.ru>*
[4] *А.С. Арутюнян, ORCID: 0000-0002-2420-7968 <arutunian@ispras.ru>*
[4] *Г.С. Керопян, ORCID: 0000-0003-2150-0461 <goqor@ispras.ru>*

[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский авиационный институт (национальный исследовательский университет) 125993, Россия, г. Москва, Волоколамское шоссе, д. 4*
[3] *Московский Государственный Технический Университет имени Н.Э.Баумана, 105005, Россия, г. Москва, 2-я Бауманская ул., д. 5, стр. 1*
[4] *Российско-Армянский Университет, 0051, Республика Армения, г. Ереван, ул. Овсепа Эмина 123*

Аннотация. В последнее время всё больше компаний, производящих программное обеспечение, заинтересованы в инструментах повышения стабильности и безопасности их продукта. Используемые разработчиками закрытые библиотеки и сторонние приложения могут содержать дефекты, использование которых злоумышленником или пользователем может привести к нарушению стабильности и безопасности работы приложения. В ряде случаев исходный код проблемных участков может отсутствовать. Приобретают популярность статические методы поиска дефектов в коде, позволяющие находить дефекты, недостижимые для динамических методов. Статические методы представляют собой алгоритмы исследования статической модели программы, в том числе графа вызовов, потока управления, потока данных. Исследование бинарного кода предполагает восстановление статической модели программы из бинарного файла путём дизассемблирования, восстановления границ функций, трансляцию в промежуточное представление и восстановление графа вызовов. Дефекты в современных кодовых базах, как правило, проявляются лишь на определённом множестве путей в графе вызовов, что требует межпроцедурных алгоритмов поиска дефектов. Целью данной работы является разработка методов межпроцедурных алгоритмов поиска дефектов в бинарном коде, обладающих хорошей масштабируемостью, набором поддерживаемых архитектур и приемлемой точностью. Алгоритмы построены на базе инструмента ИСП РАН Binside.

Ключевые слова: статический анализ кода; поиск ошибок; анализ исполняемого кода

Для цитирования: Иванов Г.С., Пальчиков П.М., Тарасов А.Ю., Акимов Г.С., Асланян А.К., Варданян В.Г., Арутюнян А.С., Керопян Г.С. Исследование и разработка межпроцедурных алгоритмов поиска дефектов в исполняемом коде программ. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 89–98. DOI: 10.15514/ISPRAS–2019–31(6)–5

Благодарности: Работа поддержана грантом РФФИ 18-07-01154А

Research and development of interprocedural algorithms for defect searching in executable program code

[1,2] *G.S. Ivanov, ORCID: 0000-0002-0544-6816 <gregory@ispras.ru>*
[3] *P.M. Palchickov, ORCID: 0000-0003-0449-2447<pavel98@ispras.ru>*
[3] *A.Y. Tarasov, ORCID: 0000-0002-6212-8144 <tematarasov48rus@ispras.ru>*
[3] *G.S. Akimov, ORCID: 0000-0002-6601-4312 <akimg@ispras.ru>*
[4] *A.K. Aslanyan, ORCID: 0000-0002-7320-4835 <hayk@ispras.ru>*
[4] *V.G. Vardanyan, ORCID: 0000-0002-4899-2999 <vaag@ispras.ru>*
[4] *M.S. Arutunian, ORCID: 0000-0002-2420-7968 <arutunian@ispras.ru>*
[4] *G.S. Keropyan, ORCID: 0000-0003-2150-0461 <goqor@ispras.ru>*

[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Moscow Aviation Institute (National Research University) 4, Volokolamskoe shosse, Moscow,125993, Russia*
[3] *Bauman Moscow State Technical University, 5/1, 2nd Baumanskaya, Moscow, 10500, Russia*
[4] *Russian-Armenian University, 123 Hovsep Emin str., Yerevan, 0051, Armenia*

Abstract: Recently, more and more software companies are interested in tools to improve the stability and security of their product. The closed libraries and third-party applications used by developers may contain defects, the use of which by an attacker or by a user may lead to a violation of the stability and security of the application. In some cases, the source code of the problem areas may be missing. At the moment, static methods for finding defects in code are gaining popularity, which allow finding defects that are unattainable for dynamic methods. Static methods are algorithms for studying a static model of a program, including a call graph, control flow, data flow. Studying binary code involves restoring a static model of a program from a binary file by disassembling, restoring function boundaries, translating it into an intermediate representation, and restoring a call graph. Defects in modern code bases, as a rule, appear only on a certain set of paths in the call graph, which requires interprocedural algorithms for finding defects. The aim of this work is to develop methods of interprocedural algorithms for finding defects in binary code that have good scalability, a set of supported architectures, and acceptable accuracy. Algorithms are developed based on ISP RAS Binside tool.

Keywords: staic code analysis; defect searching; executable code analysis

For citation: Ivanov G.S., Palchickov P.M., Tarasov A.Y., Akimov G.S., Aslanyan A.K., Vardanyan V.G., Arutunian M.S., Keropyan G.S. Research and development of interprocedural algorithms for defect searching in executable program code. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 89-98 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-5

Acknowledgements. The work is supported by RFBR, grant 18-07-01154.

1. Введение

В последнее время доля информационных технологий во всех сферах жизнедеятельности человека стремительно увеличивается. Становится всё больше мест, где внедряются автоматизированные системы, начиная с критических информационных инфраструктур и заканчивая повседневной жизнью человека.

Закономерно, что производители программного обеспечения стараются защитить свои продукты от воздействия злоумышленников. Пользователи же хотят быть уверенными, что их персональные данные не попадут к третьим лицам, а приложение будут стабильно

работать. Таким образом, встает вопрос обеспечения безопасности и стабильности приложений.

В целях обеспечения компьютерной безопасности и повышения стабильности, программу исследуют и проводят различные виды анализов, для обнаружения сбоев, дефектов, либо утечек данных. В настоящее время используют два подхода к анализу бинарного кода: статический и динамический анализ. Динамический анализ позволяет проводить анализ во время исполнения программы. Каждый дефект, найденный в процессе динамического анализа, соответствует реальному дефекту в коде программы. Кроме того, для динамического анализа не требуется наличие исходного кода исследуемой программы. Сложности возникают при решении задачи, касающейся генерации входных данных, которые могут покрыть все ошибочные точки. Статический анализ проводится без исполнения кода программы и включает в себя методы анализа потока управления, анализа потока данных, а также методы, использующие символьное выполнение. Статические методы поиска дефектов позволяют найти дефекты на всех путях исполнения программы, в то время как динамические – только на выполнившихся путях.

Статический анализ может быть выполнен как на уровне исходного, так и бинарного кода. Анализ исходного кода будет более полным, поскольку опирается на наиболее полную информацию о программе, имеющуюся в исходном коде. Компиляция отбрасывает часть информации, содержащейся в исходном коде, например, имена переменных и функций. Однако, анализ бинарного кода также имеет смысл. Например, компиляция дефектным компилятором, наличие неоднозначных конструкций в исходном коде, экспериментальных возможностей языка и ошибок в компиляторных оптимизациях могут создавать такие дефекты в бинарном коде, которые отсутствуют в исходном тексте программы. Дефекты на уровне исходного кода, которые содержатся в закрытых библиотеках, используемых при сборке программы, переносятся и в собранную программу.

К сожалению, в наше время открытые промышленные анализаторы бинарного кода не обладают хорошей масштабируемостью, набором поддерживаемых архитектур и достаточной точностью работы. В ИСП РАН разрабатывается инструмент, способный решать поставленные задачи, – Binside. В данной статье будет рассмотрены принципы работы инструмента и достигнутые на текущий момент результаты.

Дальнейшее изложение построено следующим образом. В разд. 2 рассмотрены существующие методы и инструменты поиска дефектов на бинарном коде. В разд. 3 описывается общая архитектура межпроцедурного поиска дефектов в исполняемом коде. Разд. 4 посвящён промежуточному представлению и выполняемому над ним анализу. Межпроцедурный анализ помеченных данных рассматривается в разд. 5. Разд. 6 посвящён учёту вызовов функций в анализе потока данных. Разд. 7 содержит описание детекторов ошибок и пример их работы. Разд. 8 описывает результаты работы инструмента, и разд. 9 завершает статью.

2. Исследование существующих подходов

BAP – статический анализатор бинарных файлов (x86 и ARM) [1]. На первом этапе бинарный код дизассемблируется на основе линейного алгоритма. Следующий этап – это транслирование бинарного кода в промежуточное представление (без побочных эффектов). Промежуточное представление приводится в форму статического единственного присваивания. На полученном представлении проводится анализ и оптимизация, восстанавливаются зависимости по данным на базе цепочек def-use и use-def, удаляется мертвый код. BAP выполняет задачу транслирования кода в промежуточное представление и восстановления графа потока управления и графа

вызовов. В текущей версии [2] пользователям доступны API для написания собственных модулей детекторов. Использование в процессе анализа SMT-решателей, позволяет получать хорошую точность анализов, но при этом уменьшается масштабируемость инструмента.

BARF [3] - статический анализатор бинарных файлов с открытым исходным кодом. Архитектура фреймворка состоит из трех модулей: ядро, архитектура и анализатор. Первый разделён на следующие модули: REIL, SMT и BL, которые не зависят от платформы. Кроме того, он реализует эмулятор, парсер и трансляцию во внутреннее представление. Модуль SMT используется для создания переменных и установки допущений, после проверяет выполнимость бинарного кода на различных наборах формул. Модуль архитектуры описывает архитектуру, т.е. регистры, размер адреса памяти, наборы команд, операндов и другие данные. Модуль архитектуры содержит модуль синтаксического анализа, который получает строку с разобранными инструкциями и создает серию аннотированных объектов, которые их описывают, далее каждая инструкция преобразовывается в семантически эквивалентную последовательность команд REIL [4, 5].

BARF выполняет задачу транслирования кода в REIL и восстановления графа потока управления и графа вызовов. Анализ проводится за счет других инструментов, к примеру, в качестве SMT-решателей выступают инструменты Z3 и CVC4, которые проверяют заданные выражения и находят программные дефекты. Пользователями доступны API для написания собственных модулей детекторов.

BAP и BARF, хотя и предоставляют программный доступ к своим инструментам, требуют значительной доработки. В частности, BARF предоставляет в списке инструментов кроме алгоритмов статического представления программы SMT-решатель. SMT-решатели часто используются в инструментах статического исследования программ, в том числе и в целях поиска дефектов. Однако, как показывает практика, одних SMT-решателей недостаточно для масштабируемого поиска дефектов в коде программы.

3. *Архитектура межпроцедурного поиска дефектов*

Дефекты бинарного кода классифицированы в базе CWE [5]. Каждый элемент этой базы определяет тип дефекта, имеет свой идентификационный номер и описание. С помощью описания можно составить правило, определяющее наличие дефекта в бинарном коде. Для поиска сразу нескольких типов ошибок в коде целесообразно реализовать платформу, на основе которой можно было бы создавать правила поиска дефектов. Большинство дефектов в базе CWE могут быть вызваны последовательностью вызовов различных функций, что требует межпроцедурности платформы [6]. Платформа межпроцедурного поиска дефектов определяется обходом графа вызовов [6], внутрипроцедурным и межпроцедурным анализом потока данных, а также детекторами ошибок (так называемые чекеры), каждый из которых реализует правило поиска дефекта из базы CWE. В инструменте Binside [7], разработанном в ИСП РАН, были реализованы две платформы межпроцедурного поиска дефектов, одна обеспечивает обход графа вызовов снизу-вверх по слоям, вторая – сверху-вниз.

Для межпроцедурного анализа в Binside используется архитектура обхода графа вызовов снизу-вверх, что предполагает снижение контекстной чувствительности, но позволяет учитывать влияние вызовов функций на состояние программы с помощью аннотаций. Аннотации представляют собой структуру данных ограниченного размера, в которую записываются наиболее важные для анализа эффекты от вызова анализируемой функции в произвольном контексте. Аннотации являются структурой, достаточной для создания передаточной функции вызова процедуры. Обход функций производится в соответствии

с графом вызовов таким образом, чтобы вызываемая функция анализировалась перед вызывающей. То есть анализ функций начинается с листьев графа вызова функций; при этом сам граф вызова функций разбивается на слои и анализируется послойно.

Слой представляет собой набор функций, представленный таким образом, чтобы каждая функция в графе вызовов анализировалась один раз, и сохранялся порядок снизу-вверх. Анализ функций проводится независимо друг от друга, и функции в одном слое могут анализироваться параллельно, что позволяет увеличивать масштабируемость инструмента и снижать время анализа бинарного файла. Обход графа вызовов снизу-вверх позволяет найти больше дефектов, однако в силу своей природы теряет точность из-за замены функций в аннотации на их представления.

Качество работы детекторов ошибок можно описать двумя признаками: точностью и полнотой. Точность характеризует отношение количества корректно найденных точек к количеству всех найденных точек, а полнота – отношение корректно найденных точек к количеству всех действительных дефектов в программе. При реализации алгоритма может возникнуть неполнота и избыточность. У хорошего промышленного анализатора кода неполнота и избыточность должны быть сведены к установленному минимуму.

4. Промежуточное представление и выполняемые над ним анализы

Первичный разбор бинарного файла, дизассемблирование, восстановление графа потока управления и восстановление графа вызовов выполняются с помощью IDA Pro [8, 9]. Дизассемблерное представление транслируется в промежуточное представление REIL [4, 5]. Так, множество поддерживаемых для анализа архитектур определяется множеством трансляторов. На данный момент поддерживаются следующие архитектуры: ARM, x86, x64 и MIPS.

REIL – это промежуточное представление программного кода, которое является независимым от платформы. Это язык сходный с ассемблерным представлением, однако содержит всего 17 инструкций (против сотен и тысяч инструкций популярных архитектур) и каждая инструкция не имеет побочных эффектов. Инструкции REIL можно сгруппировать в пять категорий:

1) арифметические;
2) побитовые;
3) работа с памятью;
4) передача управления;
5) служебные (не нужны для анализа в Binside).

Каждая инструкция имеет ровно три операнда, первые два являются источниками и третий – операнд назначения. Существуют три типа операндов: целочисленные литералы, регистры и адреса. Адреса REIL состоят из двух целочисленных частей. Первое число представляет адрес инструкции из исходной архитектуры, второе – нумерация REIL-инструкций.

В Binside реализован анализ значений (Value Analysis, VA) [10]. VA восстанавливает аппроксимированную карту памяти и регистров в каждой точке программы. Все анализы потока данных, включая VA выполняются на промежуточном представлении REIL. VA - это комбинированный алгоритм числового анализа и анализа указателей, Ключевой особенностью VA является то, что он обрабатывает числовые и адресные значения однообразно [11], что имеет решающее значение для анализа исполняемых файлов, так как числовые значения и адреса неразличимы во время выполнения. VA имеет чувствительность к потоку управления.

Восстановление множества инструкций в программе, результаты которых зависят от входных данных, называется анализом помеченных значений [12]. Анализ помеченных

значений имеет высокое значение для составления детекторов ошибок в коде программы и трассы, полученные с его помощью, могут характеризоваться тремя видами зависимости: функциональные адресные и по управлению [13]. Функциональная зависимость возникает, когда результат вычисления на одном шаге программы напрямую зависит от результата вычисления на другом. Адресная зависимость возникает, когда выбор ячейки, от которой функционально зависит результат вычисления на одном шаге, зависит от результата вычисления на другом шаге. А зависимость по управлению определяет возможный порядок исполнения инструкций. Внутрипроцедурный анализ помеченных данных в Binside построен, аналогично VA, на монотонных решётках, что обеспечивает функциональную зависимость и зависимость по управлению в полученных трассах [14]. Схема работы инструмента представлена на рис. 1.

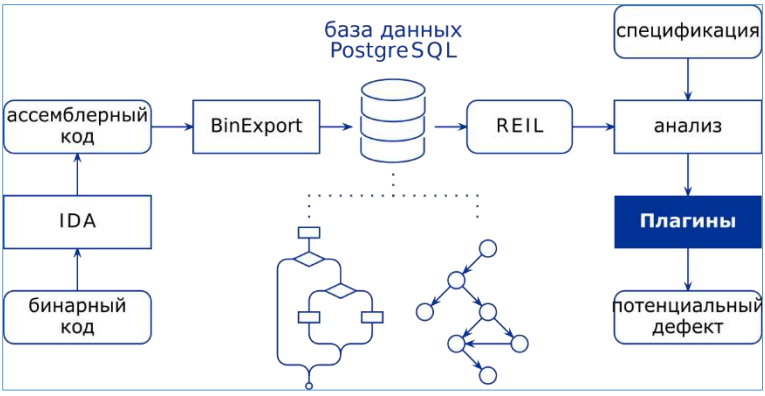

Рис. 1. Схема работы инструмента Binside
Fig. 1. Binside tool operation diagram

5. Межпроцедурный анализ помеченных данных

Платформа Binside поддерживает поиск дефектов сверху-вниз, реализованный в виде межпроцедурного анализа помеченных данных. Начиная с точки входа помеченных данных, помеченные данные распространяются вниз по графу вызовов через аргументы вызываемых функций. При таком подходе в функции ниже по графу вызовов передаются конкретные помеченные аргументы, обеспечивая контекстную чувствительность и повышая полноту анализа. Пользователь имеет возможность задавать вручную функцию, с которой начинается распространение помеченных данных (source), что позволяет дополнить анализ результатами обратной разработки программного обеспечения аналитиком.

6. Учёт вызовов функций в анализах потока данных

Полнота анализа потока данных требует учёта влияния вызываемых функций. В результате динамической линковки программ код в результирующем бинарном файле вызывает функции из других бинарных файлов – динамически подключаемых библиотек. Учёт влияния вызовов библиотечных функций на поток данных можно реализовать двумя способами. Первый способ состоит в использовании в процессе анализа бинарного файла все требуемые для его работы динамически подключаемые библиотеки [15], что требует их наличия в системе. Второй метод использует базу аннотаций для каждой неизвестной функции [16]. Аннотация определяет влияние конкретной функции на контекст вызывающей функции. Использование аннотации при анализе инструкции вызова функции позволяет учесть влияние вызываемой функции на контекст анализа потока данных не имея кода вызываемой функции.

Для аннотаций в инструменте Binside составлена база функций из наиболее часто используемых библиотек. Функции управления памятью, используемые в операторах C++ “new” и ”delete” [17], также включены в базу. Аннотация записывается в виде набора правил учёта вызова функции на контекста анализа потока данных. К примеру, аннотация “allocatesMemory” имеет семантику выделения памяти на куче и при анализе потока данных помещает в возвращаемое значение функции символический адрес новой выделенной памяти на куче. Аннотации могут использоваться в детекторах ошибок, для поиска дефектов некорректного вызова функций.

7. Детекторы ошибок и пример их работы

Детекторы ошибок в Binside запускаются платформой в процессе межпроцедурного анализа как сверху вниз, так и снизу вверх после выполнения всех внутрипроцедурных анализов потока данных. Детекторы принимают на вход промежуточное представление программы и результаты внутрипроцедурных анализов потока данных и потока управления, в частности: восстановление карты памяти и регистров, построение цепочек определение-использование (def-use) и восстановление трассы помеченных данных. Если правило, реализованное в детекторе сработало, с учётом результатов всех выполненных анализов, платформа генерирует сообщение об ошибке.

Примером такого детектора ошибок может служить детектор переполнения буфера, основанного на некорректном вызове функции `strcpy`. Анализ помеченных данных, поддержанный анализом значений, помечает все инструкции и операнды, содержащие данные зависимые от входных. Если доказывается, что первый аргумент `strcpy`, представляющий собой указатель на буфер-источник, строится из данных пользователя – генерируется ошибка. Схожим детектором является детектор инъекции команд, также основанного на результатах анализа помеченных данных. Если доказывается, что в аргумент вызова system, передающего команду в командную оболочку, попадают данные, формируемые пользователем – платформа генерирует предупреждение.

В качестве информации, дополняющей анализ, могут служить функции, полученные вследствие ручного анализа и потенциально содержащие ошибку в работе с памятью, например, выполняющие неаккуратное копирование в ходе своей работы. При успешном доказательстве контроля пользователем данных поступающих в аргументы такой функции соответствующим детектором генерируется предупреждение об ошибке.

Рассмотрим действие алгоритма и правил на простом примере (листинг 1).

```
int main(int argc, char *argv[]) {
   char buf[10];
   strcpy(buf, argv[1]);
}
```

Листинг 1. Пример утечки помеченных данных
Listing 1. Example of tagged data leakage

На листинге 1 приведен пример программы, написанной на языке C++, содержащей переполнение буфера. В первой строке программа получает аргументы функции main которые будут являться источником помеченных данных. В третьей строке вызывается небезопасная функция strcpy, которой в качестве второго аргумента попадает помеченное значение `argv[1]`. Так как это значение полностью контролируется пользователем, а функция `strcpy` не проверяет длину исходной строки, то, если подать на вход строку, содержащую более 10 символов, произойдёт переполнение буфера. Функция `strcpy` присутствует в базе аннотаций Binside с аннотацией неаккуратного копирования данных. А условие, при котором второй аргумент этой функции помечен, генерирует предупреждение о дефекте в программе.

8. Результаты

В табл. 1 приведены результаты работы детекторов при использовании обхода графов вызовов сверху вниз.

Запуск межпроцедурной платформы поиска дефектов сверху вниз также показал неплохие результаты на бинарных файлах с дефектом переполнением буфера при специально оформленных аргументах командной строки. Для таких бинарных файлов точкой входа помеченных данных является функция «main». Результаты работы представлены в табл. 2.

Табл. 1. Результаты работы детекторов ошибок с обходом графа вызовов сверху вниз
Table 1. The results of the operation of error detectors with the call graph top-down traversal

Имя бинарного файла	Версия	Размер	Время анализа	Количество найденных точек	Количество корректно найденных точек
accell-ppd	1.10.0	244KB	1m40sec	6	4
giflib	5.1.2	116KB	15 sec	2	2
open-slp	1.2.1	336KB	50 sec	4	3
tiff2pdf	4.0.3	124KB	3m 31sec	4	2
jasper	<1.900.2	200KB	5m 36 sec	4	2
Процент корректных срабатываний:					65%

Табл. 2. Результаты работы детекторов ошибок с обходом графа вызовов снизу вверх
Table 2. The results of the operation of error detectors with the call graph bottom-up traversal

Имя бинарного файла	Версия	Размер	Время анализа	Количество найденных точек	Количество корректно найденных точек
hsolinkcontrol	1.0.118	28KB	3s	23	23
shar	4.2.1	42.5KB	31s	2	1

Как видно, в обоих таблицах присутствуют срабатывания на точках, не являющихся действительными дефектами программного обеспечения. В общем случае невозможно разработать абсолютно строгий алгоритм поиска дефектов для применения на современных программах, поэтому в современных статических анализаторах делается ряд предположений, вносящих нестрогость в общий алгоритм и понижающих точность анализа.

9. Заключение

В ходе работы была реализована платформа поиска дефектов в исполняемом коде с поддержкой анализа значений на основе монотонных решеток. На её основе была выбрана архитектура обхода графа вызовов функций снизу-вверх с использованием аннотаций и совместно реализована архитектура межпроцедурного анализа помеченных данных сверху-вниз. Была составлена база аннотаций, которые могут использоваться в детекторах ошибок для поиска дефектов некорректного вызова функций. Также были разработаны детекторы для обнаружения дефектов в программе на основе результатов полученных анализов. Используемые решения обладают хорошей масштабируемостью, поддерживают различные архитектуры и обладают приемлемой точностью.

Список литературы / References

[1]. David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J. Schwartz. BAP: a binary analysis platform. In Proc. of the 23rd International Conference on Computer Aided Verification, 2011, pp. 463-469.

[2]. BinaryAnalysisPlatform / bap. Available at https://github.com/BinaryAnalysisPlatform/bap, accessed 10.12.2019.

[3]. programa-stic / barf-project. Available at https://github.com/programa-stic/barf-project, accessed 10.12.2019.

[4]. Thomas Dullien, Sebastian Porst. REIL: A platform-independent intermediate representationof disassembled code for static code analysis. In Proc. of the CanSecWest Conference, 2009, 7 p.

[5]. Mitre. Available at https://cwe.mitre.org/, accessed 10.12.2019.

[6]. V. P. Ivannikov, A. A. Belevantsev, A. E. Borodin, V. N. Ignatiev, D. M. Zhurikhin, A. I. Avetisyan. Static analyzer Svace for finding defects in a source program code. Programming and Computer Software, vol. 40, issue 5. 2014, pp. 265-275. DOI: 10.1134/S0361768814050041.

[7]. BINSIDE. Инструмент обнаружения дефектов в программе методами статического анализа исполняемого кода. Режим доступа: https://www.ispras.ru/technologies/binside/, дата обращения 10.12.2019 / BINSIDE. Static binary code analysis tool. Available at https://www.ispras.ru/en/technologies/binside/, accessed 10.12.2019.

[8]. IDA Pro. Available at https://www.hex-rays.com/, accessed 10.12.2019.

[9]. Chris Eagle. The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler. 2nd edition. No Starch Press, 2011, 672 p.

[10]. G. Balakrishnan and T. Reps. WYSINWYX: What You See Is Not What You Execute. ACM Transactions on Programming Languages and Systems, 2010, Article No. 23.

[11]. Balakrishnan G., Reps T. Analyzing Memory Accesses in x86 Executables. Lecture Notes in Computer Science, vol. 2985, 2004, pp. 5-23.

[12]. Кошелев В.К., Избышев А.О, Дудина И.А. Межпроцедурный анализ помеченных данных на базе инфраструктуры LLVM. Труды ИСП РАН, том 26, вып. 2, 2014 г., стр. 97-118 / Koshelev V.K., Izbyshev A.O., Dudina I.A. Interprocedural taint analysis for LLVM-bitcode. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 2, 2014, pp. 97-118 (in Russian). DOI: 10.15514/ISPRAS-2014-26(2)-4.

[13]. Тихонов А.Ю., Аветисян А.И. Развитие taint-анализа для решения задачи поиска программных закладок. Труды ИСП РАН, том 20, 2011 г., стр. 9-24 / Tichonov A.Y., Avetisyan A.I. Development of taint-analysis methods to solve the problem of searching of undeclared features. Trudy ISP RAN/Proc. ISP RAS, vol. 20, 2011, pp. 9-24 (in Russian).

[14]. Беляев М.В., Шимчик Н.В., Игнатьев В.Н., Белеванцев А.А. Сравнительный анализ двух подходов к статическому анализу помеченных данных. Труды ИСП РАН, том 29, вып. 3, 2017 г., стр. 99-116 / M.V. Belyaev, N.V. Shimchik, V.N. Ignatyev, A.A. Belevantsev Comparative analysis of two approaches to the static taint analysis. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 3, 2017, pp. 99-116. DOI: 10.15514/ISPRAS-2017-29(3)-7.

[15]. Angr. Available at https://github.com/angr/angr, accessed 10.12.2019

[16]. Статический анализатор Svace. Промышленный поиск критических ошибок в безопасном цикле разработки программ. Режим доступа: https://www.ispras.ru/technologies/svace /, дата обращения 10.12.2019 / Svace Static Analyzer. Finding critical program errors in production within secure development lifecycles. Available at https://www.ispras.ru/en/technologies/svace/, accessed 10.12.2019.

[17]. Bjarne Stroustrup. The C++ Programming Language: Special Edition (3rd Edition). Addison-Wesley Professional, 2000, 1030 p.

Информация об авторах / Information about authors

Григорий Сергеевич ИВАНОВ – старший лаборант ИСП РАН. Студент специалитета МАИ. Научные интересы: анализ бинарного кода, компиляторные технологии, параллельное программирование.

Grigoriy Sergeevich IVANOV – senior assistant of ISP RAS, MAI student. Research interests: binary code analysis, compiler technologies, parallel programming.

Павел Михайлович ПАЛЬЧИКОВ – студент специалитета МГТУ им. Баумана. Научные интересы: анализ бинарного кода, компиляторные технологии, параллельное программирование.

Pavel Michaylovich PALCHIKOV – BMSTU student. Research interests: binary code analysis, compiler technologies, parallel programming.

Артём Юрьевич ТАРАСОВ – студент специалитета МГТУ им. Баумана. Научные интересы: анализ бинарного кода, компиляторные технологии, параллельное программирование.

Artem Yuryevich TARASOV – BMSTU student. Research interests: binary code analysis, compiler technologies, parallel programming.

Глеб Станиславович АКИМОВ – студент специалитета МГТУ им. Баумана. Научные интересы: анализ бинарного кода, компиляторные технологии, параллельное программирование.

Gleb Stanislavovich AKIMOV – BMSTU student. Research interests: binary code analysis, compiler technologies, parallel programming.

Айк Каренович АСЛАНЯН – научный сотрудник, преподаватель, кандидат физико-математических наук. Сфера научных интересов: анализ программ, статический анализ кода, поиск клонов кода.

Hayk Karenovich ASLANYAN – researcher, lecturer, Ph.D in physical and mathematical sciences. Research interests: program analysis, code static analysis, code clone detection.

Ваагн Геворгович ВАРДАНЯН – научный сотрудник, преподаватель, кандидат технических наук. Сфера научных интересов: анализ программ, статический и динамический анализ кода.

Vahagn Gevorgovich VARDANYAN – researcher, lecturer, Ph.D in technical sciences. Research interests: program analysis, code static and dynamic analysis.

Мариам Сероповна АРУТЮНЯН – научный сотрудник, преподаватель, аспирант. Сфера научных интересов: анализ программ, статический анализ кода, поиск клонов кода.

Mariam Seropovna ARUTUNIAN – researcher, lecturer, Ph.D student. Research interests: program analysis, code static analysis, code clone detection.

Григор Сосович КЕРОПЯН – научный сотрудник, магистрант. Сфера научных интересов: анализ программ, статический анализ кода.

Grigor Sosovich KEROPYAN – researcher, master student. Research interests: program analysis, code static analysis.

DOI: 10.15514/ISPRAS-2019-31(6)-6

CC BY

Обзор методов автоматизированной генерации эксплойтов повторного использования кода

[1,2] *А.В. Вишняков, ORCID: 0000-0003-1819-220X <vishnya@ispras.ru>*
[1] *А.Р. Нурмухаметов, ORCID: 0000-0003-1681-1580 <nurmukhametov@ispras.ru>*
[1] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[2] *Московский государственный университет имени М.В. Ломоносова, 119991, Россия, Москва, Ленинские горы, д. 1*

Аннотация. В работе приводится обзор существующих методов и инструментов автоматизированной генерации эксплойтов повторного использования кода. Такие эксплойты используют код, уже содержащийся в уязвимом приложении. Подход повторного использования кода (например, возвратно-ориентированное программирование) позволяет эксплуатировать уязвимости программного обеспечения при наличии защитного механизма операционной системы, который запрещает исполнение кода страниц памяти, помеченных в качестве данных. В статье дается определение базовых понятий таких, как гаджет, фрейм гаджета, каталог гаджетов. Кроме того, показывается, что гаджет по своей сути является инструкцией, а их набор задает некоторую виртуальную машину. Задача создания эксплойта сводится к задаче генерации кода для такой виртуальной машины. Набор команд виртуальной машины задается исполняемым кодом конкретной программы. В работе приводится обзор методов поиска гаджетов и определения их семантики (формирования каталога гаджетов). Они позволяют получить набор команд виртуальной машины. Если набор гаджетов в каталоге полон по Тьюрингу, то гаджеты из каталога можно использовать в качестве набора команд целевой архитектуры компилятора. Однако в каталоге гаджетов для конкретного приложения могут отсутствовать некоторые инструкции, поэтому в литературе было предложено несколько способов для замены отсутствующих инструкций несколькими гаджетами. Связывание гаджетов в цепочки может происходить как поиском гаджетов по шаблонам, задаваемым регулярными выражениями, так и с учетом семантики гаджета. Более того, существуют подходы конструирования ROP цепочек с использованием генетических алгоритмов, а также методы с использованием SMT-решателей. В статье проводится сравнение инструментов с открытым исходным кодом. Мы предлагаем тестовую систему rop-benchmark, с помощью которой была проведена экспериментальная проверка работоспособности генерируемых инструментами цепочек на специально сформированном наборе тестов.

Ключевые слова: атаки повторного использования кода; возвратно-ориентированное программирование; ROP; предотвращение выполнения данных; DEP; рандомизация размещения адресного пространства; ASLR; гаджет; фрейм гаджета; каталог гаджетов; ROP цепочка; поиск гаджетов; классификация гаджетов; ROP компилятор; символьная интерпретация; ROP benchmark

Для цитирования: Вишняков А.В., Нурмухаметов А.Р. Обзор методов автоматизированной генерации эксплойтов повторного использования кода. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 99–124. DOI: 10.15514/ISPRAS–2019–31(6)–6

Благодарности: Работа поддержана грантом РФФИ № 17-01-00600

Survey of methods for automated code-reuse exploit generation

[1,2] *A.V. Vishnyakov, ORCID: 0000-0003-1819-220X <vishnya@ispras.ru>*
[1] *A.R. Nurmukhametov, ORCID: 0000-0003-1681-1580 <nurmukhametov@ispras.ru>*
[1] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[2] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation*

Abstract. This paper provides a survey of methods and tools for automated code-reuse exploit generation. Such exploits use code that already contains in a vulnerable program. The code-reuse approach, e.g., return-oriented programming, allows one to exploit vulnerabilities in the presence of operating system protection that prohibits execution of code in memory pages marked as data. We define fundamental terms such as gadget, gadget frame, gadget catalog. Moreover, we show that a gadget is, in fact, an instruction, and a set of gadgets defines a virtual machine. We can reduce an exploit creation problem to code generation for this virtual machine. Each particular executable file defines a virtual machine instruction set. We provide a survey of methods for gadgets searching and determining their semantics (creating a gadget catalog). These methods allow one to get the virtual machine instruction set. If a set of gadgets is Turing-complete, then a compiler can use a gadget catalog as a target architecture. However, some instructions can be absent. Hence we discuss several approaches to replace missing instructions with multiple gadgets. An exploit generation tool can chain gadgets by pattern searching (regular expressions) or taking gadgets semantics into consideration. Furthermore, some chaining methods use genetic algorithms, while others use SMTsolvers. We compare existing open source tools and propose a testing system rop-benchmark that can be used to verify whether a generated chain successfully opens a shell.

Keywords: code-reuse attacks; return-oriented programming; ROP; data execution prevention; DEP; address space layout randomization; ASLR; gadget; gadget frame; gadget catalog; ROP chain; gadget search; gadget classification; ROP compiler; symbolic execution; ROP benchmark

For citation: Vishnyakov A.V., Nurmukhametov A.R. Survey of methods for automated code-reuse exploit generation. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 99-124 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-6

Acknowledgements. This work was supported by the Russian Foundation for Basic Research, project no. 17-01-00600

1. Введение

Современное программное обеспечение содержит ошибки. Их наличие рассматривается некоторыми исследователями как неизбежность. Однако далеко не все ошибки возможно использовать (*эксплуатировать*) в злонамеренных целях. Эксплуатируемые ошибки называются *уязвимостями*. Эксплуатация уязвимостей приводит к серьезным последствиям: денежным убыткам, деградации средств коммуникации, компрометации криптографических ключей [1] и др. С развитием технологий интернета вещей окружающие нас повседневно предметы (чайники, холодильники, душевые системы и др.) могут быть подвержены эксплуатации. Особенно критичны вопросы безопасности медицинского оборудования. Гальперин (Halperin) и др. [2] показали возможность эксплуатации имлантируемых сердечных дефибрилляторов.

Вместе с развитием безопасного цикла разработки программного обеспечения улучшаются также методы по обнаружению разнообразных программных дефектов. В ответ на усовершенствование методов защиты от эксплуатации уязвимостей разрабатываются новые методы их обхода и эксплуатации. Поэтому необходимо знать и понимать, как устроены методы как защиты, так и атаки на программное обеспечение. Более того, для приоритетного исправления уязвимостей вендоры и разработчики программного обеспечения требуют подтверждающие примеры – *эксплойты*.

Переполнение буфера на стеке является, пожалуй, одним из самых эксплуатируемых программных дефектов [3]. Это объясняется сравнительной легкостью в использовании этого дефекта для перехвата потока управления под контроль атакующего. В простейшем случае полного отсутствия защит эксплуатация происходит следующим образом. Адрес возврата, расположенный на стеке выше локального буфера, перезаписывается контролируемым значением. Данное значение указывает обратно внутрь буфера, где располагается код, который хочет исполнить атакующий.

Для противодействия исполнению кода, расположенного в буфере на стеке, появилась защита DEP. Она позволила запретить исполнять определенные регионы памяти процесса, такие как стек и куча. И, в свою очередь, запретить писать в регионы памяти, помеченные для исполнения. Данная защита положила конец эпохи инъекции кода в память процесса. Атакующие оказались ограничены в своих возможностях исполнять только код, имеющийся в памяти процесса.

В ответ на повсеместное распространение DEP начали активно развиваться атаки повторного использования кода. Первым таким типом атаки являлась атака возврата в библиотеку [4]. На стек помещается адрес функций для вызова на место адреса возврата, а за ним следом размещаются аргументы функции. Обобщением данного метода атаки стало развитие методов возвратно-ориентированного программирования [5–7]. При возвратно-ориентированном программировании вместо функций выступают короткие последовательности инструкций, заканчивающиеся инструкцией возврата и называемые *гаджетами*. Гаджеты связываются в цепочки так, чтобы они последовательно передавали друг другу управление и осуществляли в совокупности некоторую вредоносную нагрузку. Шахем (Shacham) [5] дал определение понятию гаджет и привел первый каталог гаджетов, для которого была показана полнота по Тьюрингу для архитектуры набора команд x86. В дальнейшем была показана применимость возвратно-ориентированного программирования и для других архитектур набора команд: ARM [8–12], SPARC [13], Atmel AVR [14], PowerPC [15], Z80 [16], MIPS [15]. В работах [9, 17–19] было показано, что можно использовать гаджеты, которые оканчиваются не только инструкциями возврата.

Вместе с развитием методов повторного использования кода происходило и развитие инструментов, с помощью которых атакующий конструировал атаки данного типа. Вначале это процесс был практически ручным, но со временем он постепенно автоматизировался. В данный момент в литературе представлен набор подходов к автоматизированному построению эксплойтов повторного использования кода [6, 7, 11–13, 18, 20–26]. Кроме того, для некоторых из них даже доступны инструменты [27–36].

Данная работа ставит своей целью детальное изучение имеющихся методов и инструментов автоматизированной генерации эксплойтов повторного использования кода с целью определения сильных и слабых сторон каждого из них, а также выявления перспективных направлений исследования в данной области.

Кроме практической применимости, рассматриваемые в статье методы и инструменты могут иметь и научный интерес. Задача автоматизированной генерации эксплойтов для атак повторного использования кода является задачей трансляции некоторого описания эксплойта в архитектуру набора команд виртуальной машины, неявно задаваемой состоянием памяти эксплуатируемого процесса. В качестве инструкций набора команд выступают гаджеты, расположенные в памяти процесса. Причем заранее неизвестно, какой набор инструкций предоставляет эксплуатируемый исполняемый файл. Для его определения необходимо найти все гаджеты, а затем произвести процедуру определения их функциональности (семантики). В результате, формируется каталог гаджетов, где описана их семантика. Каталог гаджетов является входными данными для инструмента, который генерирует эксплойт. Генерирующий эксплойт инструмент должен учитывать тот факт, что в наборе гаджетов, в отличии от инструкций процессора, могут

отсутствовать некоторые инструкции, а другие – иметь нетривиальные побочные эффекты. Все это усложняет построение инструментов автоматической генерации эксплойтов возвратно-ориентированного программирования.

Данная работа устроена следующим образом. В разд. 2 приводится определение возвратно-ориентированного программирования. В разд. 3 описывается общая схема генерации эксплойтов повторного использования кода. В разд. 4 вводится определение *каталога гаджетов*. В разд. 5 описываются подходы к поиску гаджетов. В разд. 6 приводятся методы определения семантики гаджетов. В разд. 7 проводится обзор методов генерации цепочек гаджетов. В разд. 8 освещается проблема учета запрещенных символов в цепочках. Экспериментальное сравнение инструментов с открытым исходным кодом, выполненное с использованием разработанной нами тестовой системы rop-benchmark [37], описывается в разд. 9.

2. Возвратно-ориентированное программирование

Шахем [5] предложил термин возвратно-ориентированное программирование (return-oriented programming, ROP). ROP является эффективным средством обхода предотвращения выполнения данных (DEP, W^X). В некотором смысле данный подход является обобщением атаки возврата в библиотеку (return-to-libc) [4]. Однако вредоносная нагрузка осуществляется не вызовом одной функции, а формируется из нескольких уже присутствующих в программе кусочков кода, которые называются *гаджетами*. Гаджет – это последовательность инструкций, заканчивающаяся инструкцией передачи управления. Каждый гаджет изменяет состояние регистров и памяти вычислительной машины. Например, складывает значения двух регистров и записывает результат в третий. Атакующий, изучив все имеющиеся в программе гаджеты, связывает их в *цепочки*, в которых гаджеты последовательно передают управление друг другу. Суммарная вредоносная нагрузка реализуется такой цепочкой гаджетов. При достаточном количестве гаджетов атакующим может быть сформирован полный по Тьюрингу набор, который позволит реализовывать произвольные вычисления [5]. Следует отметить, что ROP может также применяться при частичной рандомизации адресного пространства. Тогда используются гаджеты из нерандомизированных областей памяти.

Для наглядности приводится пример нескольких гаджетов для x86 в таблице 1, где представлен ассемблерный код[1] инструкций трех гаджетов. Каждый из гаджетов заканчивается инструкцией передачи управления `ret`, которая позволяет передавать управление следующему гаджету через адрес, размещаемый на стеке.

Табл. 1. Пример гаджетов для x86
Table 1. Example of x86 gadgets

`mov eax, ebx ; ret`	Копирование значения регистра `ebx` в регистр `eax`
`pop ecx ; ret`	Загрузка на регистр `ecx` значения со стека
`add eax, ebx ; ret`	Прибавление к регистру `eax` значения регистра `ebx`

Архитектура x86 является CISC. Инструкции x86 имеют нефиксированную длину, и каждая инструкция может выполнять несколько других низкоуровневых команд. Количество команд настолько велико, и они закодированы так плотно, что практически любая последовательность байтов декодируется в корректную инструкцию. Кроме того, из-за различных длин команд (от 1 байта до 15) архитектура x86 не требует выравнивания инструкций. С точки зрения ROP это означает следующее. Набор гаджетов в программе не ограничивается только инструкциями, которые были сгенерированы компилятором. Этот набор расширяется за счет инструкций, не

[1] Здесь и далее мы будем использовать синтаксис Intel для ассемблера x86.

присутствовавших в исходной программе и полученных при доступе в середину других команд. Пример, иллюстрирующий это, приводится ниже [38]:

```
f7c7070000000f9545c3 →    test edi, 0x7 ; setnz BYTE PTR [ebp-0x3d]
  c7070000000f9545c3 →    mov DWORD PTR [edi], 0xf000000 ;
                        xchg ebp, eax ; inc ebp ; ret
```

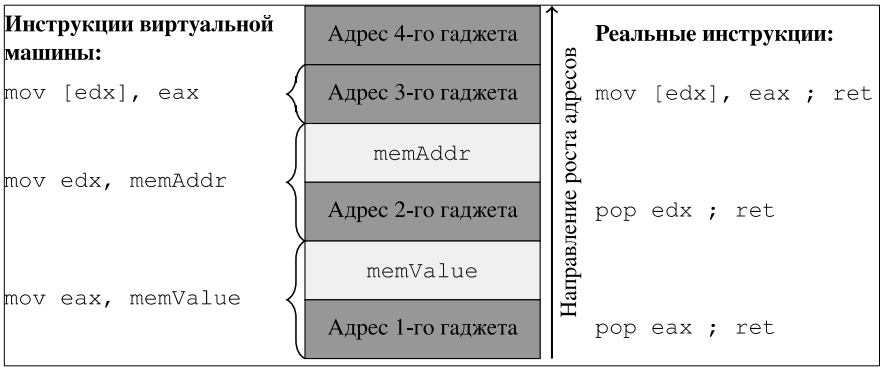

Рис. 1. Цепочка гаджетов, осуществляющая запись значения memValue по адресу memAddr
Fig. 1. A ROP Chain, storing memValue to memAddr

Набор гаджетов, который можно использовать при составлении ROP цепочки по сути задается исполняемым файлом. И для другого исполняемого файла ROP цепочку придется собирать заново. ROP цепочку можно рассматривать как программу для некоторой виртуальной машины, задаваемой исполняемым файлом [39]. Указатель стека выполняет роль счетчика инструкций этой виртуальной машины. Коды операций (адреса гаджетов) и их операнды размещаются на стеке. Грациано (Graziano) и др. [40] даже предложили инструмент для трансляции ROP цепочек в обычную программу x86. На рис. 1 приводится пример размещенной на стеке цепочки гаджетов, осуществляющей запись значения `memValue` по адресу `memAddr`. Коды операций (адреса гаджетов) размещаются от адреса возврата на стеке и закрашены темно-серым. Операнды `memValue` и `memAddr` закрашены светло-серым. Фигурными скобками обозначены инструкции виртуальной машины (код команды и ее операнды). Реальные инструкции гаджета на машине x86 приводятся справа. В начало цепочки, где при нормальном исполнении размещается адрес возврата из функции, помещается код команды – адрес первого гаджета. Затем располагается операнд `memValue`, который первый гаджет загрузит на регистр `eax`. Затем следует адрес второго гаджета, которому передаст управление первый гаджет на инструкции `ret`, и так далее. Даже бинарные файлы (x86) сравнительно небольшого размера содержат практически применимые с точки зрения атакующих наборы гаджетов. Шварц (Schwartz) и др. [6, 41] приводят статистику, что среди программ размером больше 100KB около 80 % содержат наборы гаджетов, которые позволяют вызывать любую функцию из динамически скомпонованной с уязвимым приложением библиотеки [42].

Рис. 2. Фрейм гаджета pop eax ; ret 8
Fig. 2. pop eax ; ret 8 gadget frame

В дальнейшем применение ROP было успешно продемонстрировано для других архитектур набора команд: SPARC [13], Z80 [16], ARM [8–12, 43]. В перечисленных работах было показано, что на RISC архитектурах возможно сконструировать как применимый для эксплуатации, так и полный по Тьюрингу набор гаджетов. RISC архитектуры часто характеризуются фиксированной длиной команд, требованием к выравниванию инструкций по их размеру и упрощенным доступом к памяти (к памяти как правило обращаются только инструкции сохранения и загрузки). Требование к выравниванию инструкций приводит к тому, что по сравнению с x86 остаются только гаджеты, оканчивающимися инструкциями возврата, которые изначально содержались в программе.

2.1 Фрейм гаджета

Для размещения ROP цепочки на стеке удобно ввести понятие *фрейма гаджета* [44, 45] аналогично стековому кадру x86. Цепочка гаджетов собирается из фреймов. Фрейм гаджета содержит в себе значения параметров гаджета (например, значение, загружаемое на регистр со стека) и адрес следующего гаджета. Начало фрейма определяется значением указателя стека перед выполнением первой инструкции гаджета. На рис. 2 фигурной скобкой обозначены границы фрейма гаджета `pop eax ; ret 8`. Гаджет загружает значение со стека в `eax` по смещению 0 от начала фрейма. Гаджет имеет размер фрейма `FrameSize = 16`, а адрес следующего гаджета располагается по смещению 4 от начала фрейма (`NextAddr = [esp + 4]`).

3. Общая схема генерации эксплойтов повторного использования кода

Схематично процесс генерации эксплойтов повторного использования кода делится на четыре этапа.

1. Поиск гаджетов в нерандомизованных исполняемых областях образа памяти процесса (разд. 5).
2. Определение семантики гаджетов (в некоторых методах данный этап может быть пропущен). На этом этапе определяется полезная нагрузка, которую выполняет каждый гаджет (разд. 6).
3. Комбинация гаджетов и их параметров для получения цепочки гаджетов, выполняющей заданную последовательность действий (разд. 7).
4. Автоматизированная генерация эксплойта [46–50] – входных данных, приводящих к эксплуатации программы путем размещения и выполнения ROP цепочки. На этом этапе в результате символьной интерпретации [51–53] машинных инструкций на трассе выполнения программы от точки получения входных данных до точки проявления уязвимости происходит построение предиката пути. Предикат пути объединяется с предикатом безопасности, описывающим размещение ROP цепочки и передачу на нее управления. Решением полученной системы уравнений будет эксплойт. Предикат пути обеспечит выполнение программы по тому же пути до уязвимости. А предикат безопасности – перехват потока управления.

4. Каталог гаджетов

Определим *каталог гаджетов* как список записей со следующим содержанием.

1. **Семантическое описание** последовательности инструкций машинного кода. Каждое описание соответствует, как правило, некоторой базовой вычислительной операции или операции работы с памятью (сложение, вычитание, запись в память,

чтение из памяти, инициализация регистра непосредственным значением, передача управления и т.д.).

2. **Виртуальный адрес** гаджета, обнаруженного в адресном пространстве приложения. Является некоторой аналогией кода операции для архитектуры набора команд, которая задана каталогом гаджетов.
3. **Машинные инструкции** гаджета – конкретная последовательность инструкций, реализующих заданное семантическое описание. Может задаваться вручную при составлении каталога или заполняться при автоматическом анализе двоичного образа приложения.
4. **Параметры гаджета** – параметры семантического описания, а именно: конкретные регистры, константы и т.д.
5. **Побочные эффекты** выполнения гаджета с соответствующей семантикой.

Табл. 2. Заполненный каталог гаджетов
Table 2. Complete gadget catalogue

Семантическое описание	Виртуальный адрес	Машинные инструкции	Параметры гаджета	Побочные эффекты
r1 += r2	0xdeadbeef	add eax, ebx pop edx ret	r1 = eax r2 = ebx	edx ✗
	0xcafecafe	add eax, ebx pop ecx ret	r1 = eax r2 = ebx	ecx ✗
	0xcafebabe	add edx, ecx ret	r1 = edx r2 = ecx	—
r = M[ESP + Offset]	0x12345678	pop eax ret	r = eax Offset = 0	—
r1 = M[ESP + Off1] r2 = M[ESP + Off2] r3 = M[ESP + Off3]	0x10203040	pop eax pop ebx pop ecx ret	r1 = eax, Off1 = 0 r2 = ebx, Off2 = 4 r3 = ecx, Off3 = 8	—

Побочным эффектом является любое изменение памяти и регистров, не описываемое семантикой гаджета. Побочные эффекты могут задаваться при построении каталога гаджетов вручную или автоматически вычисляться в процессе классификации гаджетов.

Для пояснения данного определения приведем пример. В табл. 2 представлен заполненный каталог гаджетов, состоящий из нескольких семантических описаний. Первое семантическое описание соответствует операции сложения значений двух регистров *r*1 += *r*2. Второе семантическое описание соответствует инструкции загрузки значения со стека в регистр. Последнее семантическое описание определяет гаджет загрузки трех регистров со стека. Первые два гаджета, найденные по адресам 0xdeadbeef и 0xcafecafe, обладают побочными эффектами относительно основного семантического описания, потому что они изменяют значения регистров `edx`, `ecx` соответственно.

4.1 Полный по Тьюрингу каталог гаджетов

Авторы многих работ [5, 7, 18, 19, 54, 55] составляют каталог гаджетов таким образом, чтобы набор семантических описаний являлся полным по Тьюрингу. Такой каталог гаджетов задает некоторую новую вычислительную машину, способную выполнять произвольные вычисления.

В процессе поиска и определения семантики гаджетов происходит заполнение каталога адресами найденных гаджетов. После этого возможны две ситуации:

1. для каждого семантического описания удалось найти конкретный гаджет;

2. для некоторых семантических описаний отсутствуют конкретные гаджеты.

В первом случае получается, что каталог гаджетов реализует на конкретном исполняемом файле вычислительную машину, способную производить произвольные вычисления. Более того, можно использовать этот каталог в качестве описания набора команд целевой архитектуры для компилятора языка Си (llvm [7]).

Во втором случае, когда отсутствуют гаджеты для некоторых семантических описаний полного по Тьюрингу каталога гаджетов, произвольные вычисления уже не выполнить. Нужно пытаться построить эксплойт из имеющихся гаджетов. В худшем случае, задача генерации может свестись к перебору всех возможных комбинаций гаджетов.

Для составления полных по Тьюрингу ROP цепочек необходимо уметь условно изменять указатель стека, который выступает в роли счетчика команд. Ремер (Roemer) и др. [7] предлагают следующий способ реализации условного ветвления для архитектуры x86:

1. выполнить некоторую операцию, которая обновит интересующий флаг;
2. скопировать интересующий флаг из регистра флагов в регистр общего назначения;
3. использовать этот флаг для условного изменения указателя стека на желаемое смещение (например, путем умножения смещения на значение флага 0 или 1).

5. Поиск гаджетов

Независимо от способа построения эксплойта необходимо в первую очередь найти в двоичном образе приложения все доступные гаджеты. К задаче поиска гаджетов существует два принципиальных подхода. Первый из них предлагает осуществлять поиск гаджетов по списку шаблонов. Шаблоны, как правило, задаются регулярными выражениями над бинарными кодами команд гаджетов. Изначально каталог гаджетов содержит семантические описания гаджетов. Для каждого семантического описания производится поиск гаджетов по некоторому шаблону. В результате в каталог гаджетов для семантических описаний будут добавлены конкретные гаджеты: виртуальные адреса, машинные инструкции и параметры гаджетов. Побочные эффекты (например, испорченные регистры [29, 33]) могут быть получены после анализа машинных инструкций найденных гаджетов.

Второй подход заключается в автоматическом поиске всевозможных последовательностей инструкций, заканчивающихся инструкцией передачи управления. Классическим алгоритмом, реализующим поиск всех гаджетов, является алгоритм Галилео [5]. Алгоритм сначала ищет инструкции передачи управления в исполняемых секциях программы. Для каждой найденной инструкции пробует дизассемблировать несколько байтов, предшествующих инструкции. Все корректно дизассемблированные последовательности инструкций добавляются в каталог гаджетов. Таким образом, каталог будет содержать виртуальные адреса и машинные инструкции гаджетов. Данный алгоритм используется во многих инструментах поиска гаджетов с открытым исходным кодом [27–33, 56–58].

6. Определение семантики гаджетов

Не все найденные гаджеты пригодны для составления ROP цепочек. Для использования гаджета при составлении ROP цепочки необходимо понимать, какую полезную нагрузку выполняет этот гаджет. Определение семантики гаджета может производится вручную [5]. При шаблонном поиске гаджетов семантика содержится в описании шаблона [7, 9, 13, 17–20, 54, 59].

6.1 Типы гаджетов

Шварц и др. [6] предложили определять функциональность гаджета его принадлежностью к некоторым параметризованным типам, которые задают новую архитектуру набора команд (ISA). Параметрами типов выступают регистры, константы и бинарные операции. Тип гаджета описывается семантически с помощью постусловия – булева предиката $\mathcal{B}$, который должен быть всегда истинным после выполнения гаджета. Следует отметить, что один гаджет может принадлежать сразу нескольким типам. Например, гаджет `push eax ; pop ebx ; pop ecx ; ret` одновременно перемещает `eax` в `ebx` и загружает значение со стека в `ecx`, что соответствует типам `MoveRegG: ebx ← eax` и `LoadConstG: ecx ← [esp + 0]`.

Для того чтобы определить, удовлетворяет ли последовательность инструкций гаджета $\mathcal{I}$ постусловию $\mathcal{B}$, Шварц и др. [6] используют известную технику из формальной верификации – вычисление слабейшего предусловия [60]. Проще говоря, слабейшее предусловие $wp(\mathcal{I}, \mathcal{B})$ для последовательности инструкций для последовательности инструкций и постусловия $\mathcal{B}$ – это булево предусловие, которое описывает, когда $\mathcal{I}$ завершается в состоянии, удовлетворяющем $\mathcal{B}$. Слабейшее предусловие используется, чтобы убедиться, что определение семантики гаджета всегда выполняется после выполнения последовательности инструкций $\mathcal{I}$. Для этого достаточно проверить: $wp(\mathcal{I}, \mathcal{B}) \equiv true$. Если формула верна, то $\mathcal{B}$ всегда истинно после выполнения $\mathcal{I}$, а значит, $\mathcal{I}$ – гаджет с семантическим типом $\mathcal{B}$.

Однако формальная верификация семантики гаджетов на практике показала себя очень медленной. Для ускорения процесса авторы предложили комбинированный подход. Инструкции гаджета предварительно несколько раз выполняются с использованием случайных входных данных, а затем проверяется истинность $\mathcal{B}$. Если $\mathcal{B}$ окажется ложным хотя бы для одного выполнения, то последовательность инструкций не может быть гаджетом этого типа. Таким образом, более сложное вычисление слабейшего предусловия производится, только если $\mathcal{B}$ истинно для каждого выполнения.

Комбинированный подход можно условно разделить на два этапа: классификацию гаджетов и верификацию гаджетов. На этапе классификации делаются гипотезы о принадлежности гаджетов к некоторым типам и о значениях параметров этих типов. Гипотезы по сути задаются булевыми постусловиями. А на этапе верификации для каждого постусловия формально доказывается его истинность или ложность, и гипотеза принимается или отвергается соответственно.

6.1.1 Классификация гаджетов

В настоящее время существует множество процессорных архитектур с различными инструкциями. Промежуточное представление машинных инструкций (VEX [61], REIL [62], Pivot [63] и др.) позволяет абстрагироваться от конкретной архитектуры и писать универсальные алгоритмы.

В работах [44, 64] классификация гаджета производится на основе интерпретации промежуточного представления инструкций гаджета. Во время интерпретации отслеживаются обращения к регистрам и памяти. Если происходит первое чтение регистра или области памяти, считанное значение генерируется случайным образом. В результате интерпретации будут получены начальные и конечные значения регистров и памяти. На основе этой информации делается вывод о возможной принадлежности гаджета тому или иному типу. Например, для принадлежности типу `MoveRegG` [6] должна существовать такая пара регистров, что начальное значение первого регистра равно конечному значению второго. В результате анализа составляется список всех удовлетворяющих гаджету типов и их параметров (список кандидатов). Затем

производится еще несколько запусков процесса интерпретации с отличными входными данными, в результате которых из списка кандидатов удаляются ошибочно определенные типы.

Более того, в результате классификации гаджета могут быть получены [44]:

- список «испорченных» регистров, значения которых изменились в результате выполнения гаджета;
- информация о фрейме гаджета (разд. 2.1): размер фрейма и смещение ячейки с адресом следующего гаджета относительно начала фрейма.

Следует отметить, что число неверно классифицированных гаджетов можно уменьшить, если добавить запуски процесса интерпретации с граничными входными данными 0 и -1. Доля неверно классифицированных гаджетов в таком случае незначительна и составляет 0.7 % [65].

6.1.2 Верификация гаджетов

Классификация гаджета предоставляет набор постусловий, описывающих возможную семантику гаджета. Верификация гаджета позволяет формально доказать истинность этих постусловий для произвольных входных данных. Верификация гаджета может производиться как на основе построения слабейшего предусловия [6, 23], как было описано выше, так и с использованием символьной интерпретации инструкций гаджета [64, 65].

Табл. 3. Пример верификации гаджета `ArithmeticLoadG: ebx ← ebx + [eax]`
Table 3. Verification of gadget `ArithmeticLoadG: ebx ← ebx + [eax]`

Шаг	**Символьное состояние**	**Инструкция**	**Множество формул**
initial	$M, eax = \phi_1, ebx = \phi_2,$ $ecx = \phi_3, esp = \phi_4,$ $eip = \phi_5$	—	$S_0 = \emptyset$
1	$ecx = \phi_6$	`mov ecx, [eax]`	$S_1 = S_0 \cup \{\phi_6 = (concat$ $(select\ M\ \phi_1)$ $(select\ M\ \phi_1 + 1)$ $(select\ M\ \phi_1 + 2)$ $(select\ M\ \phi_1 + 3))\}$
2	$ebx = \phi_7$	`add ebx, ecx`	$S_2 = S_1 \cup \{\phi_7 = \phi_2 + \phi_6\}$
final	$eip = \phi_8, esp = \phi_9$	`ret`	$S_3 = S_2 \cup \{\phi_8 = (concat$ $(select\ M\ \phi_4),$ $(select\ M\ \phi_4 + 1),$ $(select\ M\ \phi_4 + 2),$ $(select\ M\ \phi_4 + 3)),$ $\phi_9 = \phi_4 + 4\}$
	Определение семантики		**Верификация**
verify	$final(ebx) = initial(ebx) + initial(M[eax])$		$\phi_7 \neq \phi_2 + (concat$ $(select\ M\ \phi_1)$ $(select\ M\ \phi_1 + 1)$ $(select\ M\ \phi_1 + 2)$ $(select\ M\ \phi_1 + 3))$ is **UNSAT**

Рассмотрим подробнее способ классификации гаджета с использованием символьной интерпретации [51–53]. Во время символьной интерпретации моделируется семантика гаджета с использованием SMT выражений. Изначально всем регистрам присваиваются свободные символьные переменные. Символьная память в начале представляет из себя пустой байтовый массив *M* битовых векторов:

$$M = (Array\ (\_\ BitVec\ \langle addrSize \rangle)\ (\_\ BitVec\ 8)),$$

где ⟨*addrSize*⟩ – размерность адресного слова архитектуры. Символьное состояние содержит отображение регистров в символьные переменные и текущее состояние символьной памяти. Символьная интерпретация инструкций гаджета порождает SMT формулы над переменными и константами, а также обновляет символьное состояние регистров и памяти в соответствии с операционной семантикой инструкции. Работа с символьной памятью реализуется через операции *select* и *store* над *Array*. Функция (*select M i*) возвращает *i*-ый элемент массива *M* и моделирует чтение байта по адресу *i*. Функция (*store M i b*) возвращает массив, полученный из массива *M* сохранением элемента *b* по индексу *i*, что моделирует запись байта *b* по адресу *i*.

Хейтман (Heitman) и др. [64] сначала транслируют инструкции гаджета в промежуточное представление REIL [62]. А после уже производится символьная интерпретация REIL инструкций.

Постусловие для верификации семантики гаджета представляет из себя булевый предикат над начальными и конечными значениями регистров и памяти. В предикат подставляются регистры и память из соответствующих символьных состояний. Общезначимость формулы постусловия проверяется через невыполнимость ее отрицания с использованием SMT-решателя.

В табл. 3 приводится пример верификации гаджета `ArithmeticLoadG: ebx ← ebx + [eax]`. Изначально регистрам сопоставляются свободные символьные переменные ϕ_i, а память представляется массивом *M*. Множество формул пусто. Новые формулы добавляются в соответствии с операционной семантикой интерпретируемой инструкции. Для множества формул поддерживается SSA форма – при добавлении формулы создается новая символьная переменная, которой присваивается эта формула. На первом шаге создается новая символьная переменная ϕ_6, которая равна загруженному из памяти значению второго операнда инструкции `[eax]`. В символьном состоянии регистру *ecx* ставится в соответствие символьная переменная ϕ_6. Аналогично интерпретируются оставшиеся шаги. В постусловие, описывающее тип гаджета, подставляются символьные переменные из начального и конечного символьных состояний. При помощи SMT-решателя проверяется выполнимость отрицания формулы. Отрицание формулы невыполнимо, значит, гаджет удовлетворяет заявленному типу с параметрами.

Например, гаджет `neg eax ; sbb eax, eax ; and eax, ecx ; pop ebp ; ret` может быть неверно классифицирован, а верификация позволит устранить эту ошибку. Во время классификации гаджет был отнесен к типу `MoveRegG: EAX ← ECX`. Для отличного от нуля начального значения регистра `eax` гаджет действительно скопирует значение регистра `ecx` в `eax`. Однако, если начальное значение `eax` будет нулевым, то и его конечное значение будет нулевым, что не является копией значения регистра `ecx`, отличного от нуля.

6.2 Резюме гаджетов

Резюме гаджета [8, 24] представляет собой описание семантики гаджета в виде компактной спецификации. Резюме гаджета содержит предусловия и постусловия над значениями регистров и памяти. В частности, резюме гаджета может содержать:

- регистры, загруженные со стека (`eax = [esp + 4]`);
- регистры, считанные из памяти (`ecx = [edx + 2]`);
- регистры, значение которых было изменено (`ecx = eax + ebx`);
- диапазоны адресов памяти, по которым производились чтение или запись (`[rsp] <-> [rsp + 0x20]`).

Фоллнер (Follner) и др. [24] предложили следующий метод составления резюме гаджета. Сначала инструкции гаджета поднимаются до уровня промежуточного представления VEX [61]. Потом продвигаются все присваивания, таким образом, чтобы сформировать в результате одно выражение, называемое постусловием, которое описывает все операции, с помощью которых получилось конечное значение в рассматриваемом регистре. Анализ поддерживает модель памяти, которая позволяет корректно моделировать ситуацию передачи значения через стек. Также этот анализ позволяет получить предусловия, которые описывают диапазоны доступа к памяти по регистру со смещением (`[rax] <-> [rax + 0x20]`). Предусловия указывают на то, что регистры из этих диапазонов памяти должны указывать на память, доступную для чтения/записи.

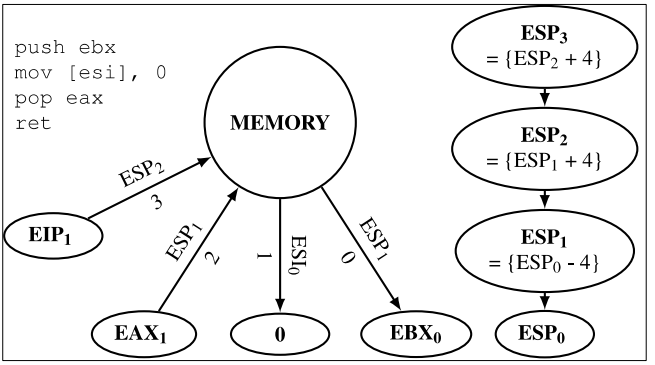

Рис. 3. Граф зависимостей гаджета

Fig. 3. Gadget dependency graph

6.3 Граф зависимостей гаджета

Миланов (Milanov) [25] предложил представлять гаджет в виде ориентированного графа зависимостей (рис. 3). Вершины соответствуют регистрам, памяти и константам. Вся память представляется единственной вершиной. Регистр же может соответствовать нескольким вершинам: каждая модификация регистра порождает новую вершину (reg_0, reg_1, reg_2 и т.д.). Направленные ребра отражают зависимости по данным (присваивание регистров, доступ к памяти и др.). Инструкции гаджета порождают новые ребра на графе. Ребра, соединенные с памятью, также содержат метки с адресом доступа к памяти и пронумерованы в хронологическом порядке.

Каталог гаджетов (разд. 4) заполняется следующим образом. Изначально каталог содержит виртуальные адреса и инструкции гаджетов. Инструкции каждого гаджета транслируются в промежуточное представление REIL [62], для которого после строится граф зависимостей. В результате обхода графа вычисляется семантическое описание гаджета: конечные значения регистров и памяти выражаются через начальные. Выражение для некоторого конечного значения может иметь условие, при котором это выражение истинно. Далее гаджеты классифицируются по типам (6.1). Автор мотивировал такой метод определения семантики гаджета меньшим временем работы по сравнению с методами, использующими SMT-решатели. Для примера на рис. 3 будет получено следующее семантическое описание:

$$EIP_1 = MEM[ESP_0]$$
$$ESP_3 = ESP_0 + 4$$
$$EAX_1 = \begin{cases} 0 & \text{if } ESP_0 - 4 = ESI_0 \\ EBX_0 & \text{if } ESP_0 - 4 \neq ESI_0 \end{cases}$$
$$MEM[ESP_0 - 4] = \begin{cases} 0 & \text{if } ESP_0 - 4 = ESI_0 \\ EBX_0 & \text{if } ESP_0 - 4 \neq ESI_0 \end{cases}$$
$$MEM[ESI_0] = 0$$

7. Генерация цепочек гаджетов

В данном разделе описываются различные методы генерации ROP цепочек. Следует отметить, что комбинирование гаджетов в цепочки является переборной задачей, поэтому для уменьшения числа итераций перебора можно предварительно отфильтровывать ненужные гаджеты и сортировать их по качеству [66]. Процесс генерации ROP цепочек отличается от обычной компиляции следующим.

- Чаще всего у ROP цепочки нет возможности сохранять значения регистров в память для их последующего восстановления из-за нехватки соответствующих гаджетов.
- У ROP гаджетов могут быть побочные эффекты. Например, гаджет может «портить» регистры. Значения «испорченных» регистров не сохранятся после выполнения гаджета. Побочные эффекты необходимо учитывать при составлении расписания гаджетов из цепочки [6].
- Некоторые типы гаджетов (6.1), которые выступают в качестве инструкций виртуальной машины, могут отсутствовать. В таком случае необходимо заменять недостающие гаджеты последовательностью других [6].

Во время генерации следует учитывать запрещенные символы, которые нельзя использовать в ROP цепочке. Такая потребность возникает, когда, например, переполнение происходит при помощи функции `strcpy`, что не позволяет цепочке содержать нулевые байты. Однако только немногие [22, 31] полноценно решают проблему запрещенных символов. Большинство просто удаляют гаджеты, чьи адреса содержат запрещенные символы, но не следят за значениями параметров гаджетов на стеке.

ROP нагрузка часто может быть разбита на установку значений регистров в заданные значения и выполнение еще одного гаджета [32]. Таким образом, метод генерации ROP цепочек можно базировать на установке регистров, а остальные нагрузки осуществлять путем добавления к полученной цепочке одного гаджета записи в память, вызова функции, системного вызова и др.

7.1 Полная по Тьюрингу компиляция с фиксированным каталогом гаджетов

Рассмотрим построение компилятора на основе фиксированного каталога гаджетов. Бьюкенен (Buchanan) и др. [7, 13] вручную сформировали полный по Тьюрингу каталог гаджетов из машинного кода стандартной библиотеки `libc` ОС Solaris для архитектуры набора команд SPARC. Каждому семантическому описанию сопоставляется единственная последовательность инструкций из машинного кода библиотеки. Архитектура SPARC допускает только выровненный доступ к инструкциям, поэтому все примеры гаджетов являются легитимными эпилогами функций библиотеки.

Одна из особенностей архитектуры SPARC – использование регистровых окон. Регистровое окно состоит из регистров, предназначенных для входных параметров, возвращаемых значений, временных значений внутри процедуры. При вызове функции

происходит сдвиг регистрового окна вперед, а при возврате – в обратную сторону. При большой вложенности стека вызовов в программе происходит нехватка регистровых окон, что приводит к необходимости их сохранения на стеке. В таком случае, при возврате из функции значения регистров восстанавливаются из сохраненных на стеке значений, что приводит к нежелательному изменению значений регистров при передаче управления от одного гаджета к другому. Таким образом, архитектура SPARC и ее соглашение о вызовах накладывает ограничения на способ передачи вычисленных значений между гаджетами – только через память. Каталог гаджетов Бьюкенена и др. [13] реализует набор гаджетов, использующих модель память-память, которая позволяет использовать регистры только внутри гаджетов, а хранение и передача значений от одного гаджета к другому происходит через память. Каждой переменной в ROP цепочке сопоставляется адрес ячейки памяти, который используется как операнд гаджета.

После полного заполнения каталога гаджетов существует две опции для автоматического создания ROP цепочек. Во-первых, у каталога гаджетов существует программный интерфейс на языке Си. В нем содержатся 13 функций, которые позволяют создавать переменные, присваивать им значения и вызывать функции (или делать системные вызовы). С помощью данного программного интерфейса можно написать программу, которая будет автоматически генерировать ROP цепочку по заполненному каталогу гаджетов. Во-вторых, Бьюкенен и др. [13] написали транслятор из некоторого псевдо-языка описания эксплойтов (урезанного Си) в последовательность вызовов функций программного интерфейса каталога гаджетов на языке Си. Компилятор реализует большую часть базовой арифметики, логических операций, операций работы с указателями и памятью, и операций условной и безусловной передачи управления. Авторами ряда работ [13, 23] отмечается возможность написания для компиляторной инфраструктуры LLVM расширения, позволяющего генерировать код для виртуальной машины, задаваемой каталогом гаджетов.

Инструмент, представленный Мосьером (Mosier) [26, 57], опирается на ROPC-IR, ассемблерный язык описания эксплойтов, который задает полную по Тьюрингу архитектуру набора команд. Он содержит три регистра: `ACC(eax)`, `SP(rbp)`, `PC(rsp)`, и набор базовых команд: арифметические операции, инструкции передачи управления, инструкции для работы с памятью и стеком. В качестве счетчика команд `PC` выступает регистр `rsp`. Кроме того, выделяется отдельный стек для функций ROP цепочки, указателем на который `SP` выступает `rpb`. Поддержка второго стека позволяет реализовывать вызовы функций внутри ROP цепочки, которые реализуют полноценные подпрограммы.

Каталог гаджетов в данном инструменте представлен описанием языка ROPC-IR. Процесс поиска гаджетов выводит всевозможные гаджеты из целевой программы. Затем необходимо вручную найти и сопоставить каждому семантическому описанию конкретный найденный в целевой программе гаджет и вручную заполнить следующие поля каталога гаджетов: виртуальный адрес, параметры гаджета. По определению языка ROPC-IR гаджеты не имеют побочных эффектов (не принимая во внимание выходные регистры). Теоретически ассемблерный язык ROPC-IR может выступать в качестве целевого языка компилятора Си. Однако практическое применение для построения эксплойтов для реальных программ может быть существенно ограничено наличием необходимых гаджетов в программе и размером генерируемых ROP цепочек. Размер цепочек из-за неоптимальности процесса трансляции языка ROPC-IR значительно больше типичных размеров эксплойтов.

Подход к построению автоматизированного инструмента генерации ROP цепочек, предложенный в работах [7, 13, 26], опирается на фиксированный каталог гаджетов. Он единожды сформирован авторами и не изменяется. Кроме того, семантические описания жестко привязаны к конкретным регистрам, используемым в гаджетах. В случае, если в

какой-то версии стандартной библиотеки `libc` отсутствует какой-то из гаджетов, то компиляция ROP цепочки может завершиться неудачно. При этом можно было бы использовать другие гаджеты, имеющие другие операнды, но схожий функционал, и добиться успешной компиляции. Другими словами, данный подход обладает ограниченной практической применимостью, особенно в ситуациях небольшого количества гаджетов в исследуемом коде библиотеки.

7.2 Генерация на основе шаблонов гаджетов

Генерация на основе шаблонов гаджетов заключается в поиске по регулярным выражениям определенной последовательности гаджетов, выполняющей некоторую вредоносную нагрузку: системный вызов `execve` [30, 33], вызов функции `VirtualProtect` с последующим выполнением обычного шелл-кода на стеке [29] и др. Запрещенные символы при таком подходе могут учитываться путем отрицания загруженного со стека значения, многочисленным повторным инкрементом до желаемого значения или же другими арифметическими операциями. Следует отметить, что Ropper [33] поддерживает поиск с использованием SMT-решателей гаджетов, удовлетворяющих семантике, которая задается постусловием над регистрами, памятью и константами. Однако на момент написания статьи для генерации ROP цепочек инструмент использует только регулярные выражения.

В работе Хуана (Huang) и др. [11] для архитектуры набора команд ARM применяется подход, основанный на использовании специального гаджета, который одновременно устанавливает значения всех регистровсо стека. Алгоритм поиска и одновременной проверки гаджета на соответствие заданной семантике производится путем анализа инструкций ассемблерного кода. Генерация цепочки из одного гаджета является тривиальной задачей и требует только правильного расположения значений регистров на стеке.

Другой подход к построению каталога гаджетов и последующей компиляции представлен Хундом (Hund) и др. [20]. На этапе поиска ищутся только гаджеты, состоящие из одной инструкции, не считая саму инструкцию возврата. Скорее всего, так сделано ради упрощения алгоритмов анализа параметров гаджета и побочных эффектов. Такие гаджеты добавляются в каталог гаджетов. На следующем шаге каталог гаджетов дополняется гаджетами, которые можно скомбинировать из имеющихся. Это можно проиллюстрировать следующим примером:

1. `pop eax ; ret` – гаджет загрузки значения со стека в регистр `eax`,
2. `mov ebx, eax ; ret` – гаджет перемещения значения из регистра `eax` в регистр `ebx`.

Эти два гаджета, вызванные последовательно, образуют гаджет загрузки значения со стека в регистр `ebx`. Хунд и др. [20] приводят алгоритм поиска всех возможных комбинаций гаджетов, перемещающих значение из одного регистра в другой регистр. Данная задача сводится к поиску пути от одной вершины к другой в специальном графе. В качестве вершин в нем выступают регистры, а в качестве ребер выступают первичные гаджеты, осуществляющие перемещение одного значения регистра в другой.

Данный подход позволяет расширить каталог гаджетов, что особенно полезно для эксплуатируемых программ небольшого размера. Однако использование комбинированных гаджетов требует обязательного учета побочных эффектов на стадии объединения гаджетов в ROP цепочку.

Нгуен Ань Куинь (Nguyen Anh Quynh) [21] предложил похожую идею объединения нескольких гаджетов в один, выполняющий желаемое поведение. Например, последовательность гаджетов `push 0x1234 ; pop ebp ; ret ; xchg ebp,`

`eax ; ret` может рассматриваться как один гаджет загрузки константы 0x1234 в регистр `eax`.

7.3 Связывание гаджетов в цепочки с использованием семантических запросов

Миланов [25] получает семантическое описание каждого гаджета путем построения его графа зависимостей (разд. 6.3). Связывание гаджетов в цепочки осуществляется последовательными семантическими запросами к каталогу гаджетов. Данный метод реализован в виде инструмента с открытым исходным кодом ROPGenerator [31].

Семантический запрос по сути является выражением над константами, конечными/начальными значениями регистров и памяти. Сначала в каталоге гаджетов ищутся гаджеты с семантическим описанием, удовлетворяющим семантическому запросу. Если такие гаджеты отсутствуют, то семантический запрос разбивается на несколько по некоторым стратегиям. Например, первый регистр можно переместить во второй через некоторый промежуточный третий регистр.

Примечательной особенностью инструмента ROPGenerator является поддержка использования при построении ROP цепочки гаджетов, оканчивающихся на инструкции `call Reg` и `jmp Reg`. Для этого перед такими гаджетами добавляется гаджет загрузки регистра `Reg`, который загружает значение адреса гаджета, которому необходимо передать управление после выполнения гаджета с `call` или `jmp`. В случае с `call` может также потребоваться передача управления на специальный гаджет, убирающий со стека адрес возврата, размещаемый `call`.

7.4 Генетический алгоритм

Фрейзер (Fraser) и др. [12] предлагают иной подход к конструированию ROP цепочки. Авторы предлагают использовать генетические алгоритмы для этого. Инструмент ROPER [28] позволяет генерировать для ARM архитектуры ROP цепочку, устанавливающую значения регистров в заданные значения.

Вначале в исполняемом файле находятся гаджеты. Для каждого из них вычисляется размер фрейма гаджета и смещение адреса следующего гаджета в нем (2.1). Затем исполняемый целевой файл загружается в адресное пространство виртуальной машины для повторяемого выполнения ROP цепочек-кандидатов. Виртуальная машина предоставляет удобный интерфейс для выполнения инструкций гостевой архитектуры.

В процессе генетических мутаций роль генов выполняют адреса гаджетов и случайные значения, размещаемые на стеке в качестве данных и адреса следующего гаджета. Функцией приспособленности является разность текущего и желаемого вектора значений регистров виртуальной машины. Каждый элемент популяции изменяется методами генетических мутаций. Среди всех кандидатов отбирается набор потенциально лучших, для которых процесс мутации повторяется вновь.

Следует отметить, что сформированные генетическим алгоритмом ROP цепочки сильно отличаются от тех, что создаются людьми. Например, они могут писать значения себе на стек или передавать управление на гаджеты, которые не были изначально обнаружены в процессе поиска. Кроме того, размер цепочки из-за неоптимального выбора гаджетов может быть большим. Описанные недостатки могут быть следствием отсутствия у генетического алгоритма информации о семантике инструкций гаджета. Возможно, если в каком-то виде ее учитывать и использовать более современные методы машинного обучения, то можно развить концепцию данного подхода в практически применимый инструмент.

Алгоритм 1. Алгоритм поиска кратчайших цепочек инициализации регистров
Algorithm 1. Search algorithm of shortest chain initializing registers

```
regset_to_chain ← empty register set mapping to shortest chains
queue ← empty queue
queue.push(empty chain)
while queue is not empty do
    chain ← queue.pop()
    for all gadget ∈ gadgets do
        new_chain ← chain + gadget
        regset ← controlled_registers(new_chain)
        if regset not in regset_to_chain or new_chain is shorter than regset_to_chain[regset] then
            regset_to_chain[regset] ← new_chain
            queue.push(new_chain)
        end if
    end for
end while
```

7.5 Генерация цепочек с использованием SMT-решателей

Фоллнер и др. [24] предложили метод генерации на основе резюме гаджетов (6.2), который имеет доступный исходный код [27]. Метод позволяет получать последовательность гаджетов, которая запишет в m запрошенных регистров заданные значения. Следует отметить, что метод не вычисляет загружаемые со стека параметры гаджетов, а только предоставляет последовательность адресов гаджетов. Изначально для всех гаджетов составляется резюме. Для каждого запрошенного регистра выбираются n наиболее подходящих гаджетов [66], которые загружают его значение со стека или из памяти, контролируемой атакующим. Далее для всевозможных цепочек гаджетов ($n^m * m!$ комбинаций) вычисляются предусловия и постусловия, но уже для всей цепочки. Если постусловия удовлетворяют ситуации, когда атакующий контролирует значения всех запрошенных регистров, то метод переходит к финальной стадии – разрешению предусловий. К цепочке дополнительно в начало добавляются гаджеты, которые проинициализируют регистры из предусловий, так чтобы они указывали на доступную для чтения и записи память.

Солс (Salls) [32] развил описанный выше метод. Ниже будет описан метод генерации цепочки, устанавливающей значения регистров в заданные значения. Остальные цепочки, такие как запись в память и вызов функции, могут быть получены добавлением всего лишь одного гаджета к цепочке, инициализирующей регистры. Метод можно разделить на три шага:

1. **Составление резюме гаджетов.** Происходит символьная интерпретация [51–53] инструкций каждого гаджета. Резюме гаджетов составляется с использованием статического анализа полученных в результате символьной интерпретации SMT выражений и запросов к SMT-решателю.
2. **Связывание гаджетов в цепочку.** На этом шаге происходит поиск кратчайших цепочек для инициализации произвольных наборов регистров. Предложенный алгоритм 1 был вдохновлен алгоритмом Дейкстры [67] поиска кратчайших путей от одной из вершин графа до всех остальных. Создается пустое отображение из наборов регистров в кратчайшие цепочки, инициализирующие эти регистры. В очередь добавляется пустая цепочка. Алгоритм достает цепочки из очереди. Для каждого гаджета создается новая цепочка, полученная добавлением этого гаджета к взятой из очереди цепочки. Вычисляется набор инициализируемых регистров (*controlled_registers*) новой цепочкой. Если такого набора нет в отображении, или полученная цепочка короче той, что в отображении, то в отображение для этого набора добавляется новая цепочка. Также эта же цепочка добавляется в очередь.

Таким образом, будет получено отображение из наборов регистров в кратчайшие цепочки, которые эти регистры инициализируют.

3. **Размещение ROP цепочки на стеке.** Запускается процесс символьной интерпретации инструкций всей ROP цепочки. Для значений, загружаемых со стека, создаются свободные символьные переменные. В конце процесса интерпретации составляется конъюнкция равенств запрошенных регистров заданным значениям. В результате решения этой конъюнкции SMT-решателем будут получены байты, которые необходимо разместить на стеке.

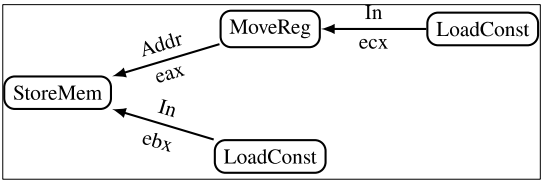

Рис. 4. Дерево гаджетов, которое записывает произвольное значение по произвольному адресу памяти

Fig. 4. Gadget tree that stores arbitrary value at arbitrary memory address

Описанный метод, в отличие от предыдущего, позволяет использовать в цепочках гаджеты, которые инициализируют сразу несколько регистров, а также гаджеты, которые выполняют арифметическую операцию над регистрами, загруженными другими гаджетами (правильные значение на стеке при этом вычислит SMT-решатель). Более того, данный метод позволяет выбирать самые короткие цепочки.

7.6 Генерация на основе семантических деревьев

Шварц и др. [6] предлагают подход к генерации ROP цепочек на основе семантических деревьев. Авторы создали свой язык QooL для написания ROP цепочек, который не обладает полнотой по Тьюрингу, но позволяет выражать применяемые на практике ROP цепочки (вызов библиотечной функции, системный вызов и запись в память). Процесс трансляции программы на языке QooL в ROP цепочку состоит из следующих этапов.

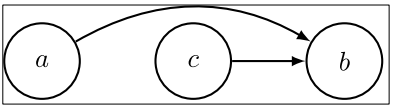

Рис. 5. Построение расписания для дерева гаджетов

Fig. 5. Scheduling gadget tree

1. Генерация семантических деревьев путем замощения [68] абстрактного синтаксического дерева исходной программы на языке QooL. Семантическое дерево состоит из абстрактных гаджетов (типов гаджетов), которые задают архитектуру набора команд и описаны в разделе 6.1.
2. Присвоение абстрактным гаджетам из семантического дерева реальных гаджетов, найденных в программе. Пример дерева реальных гаджетов приводится на рисунке 4. В вершинах дерева записаны типы гаджетов. На ребрах – имена параметров типов и их значения (конкретные регистры). Дерево гаджетов производит запись произвольного значения по произвольному адресу памяти. Записываемое значение и адрес загружаются со стека в регистры *ebx* и *ecx* соответственно. Адрес из регистра *ecx* перемещается в регистр *eax*. После чего уже происходит запись значения регистра *ebx* по адресу *eax*.
3. Построение расписания по дереву гаджетов и генерация ROP цепочки.

На первом шаге происходит ленивая генерация всех возможных семантических деревьев из абстрактных гаджетов. Это необходимо делать поскольку некоторые гаджеты могут отсутствовать в конкретной программе. На втором шаге для каждого семантического дерева применяется присвоение гаджетов. В случае, если не удается присвоить каждому абстрактному гаджету конкретный, то семантическое дерево отбрасывается и берется следующее. В случае успешного присваивания на третий шаг передается дерево реальных гаджетов. Для генерации ROP цепочки его необходимо линеаризовать, т.е. построить расписание. Построение расписания для дерева гаджетов должно учитывать: зависимости между регистрами гаджетов по данным и «испорченные» регистры. Это означает следующее (рис. 5):

1. расписание должно удовлетворять топологической сортировке дерева;
2. если выходной регистр гаджета a используется гаджетом b, то этот регистр не должен быть «испорчен» ни одним гаджетом в расписании между a и b.

При генерации семантических деревьев учитывается возможность отсутствия некоторых типов гаджетов и применяются последовательно все имеющиеся правила выражения вершины абстрактного синтаксического дерева через семантические деревья из абстрактных гаджетов. Например, авторы заметили, что успешность генерации ROP цепочки возрастает, если добавить следующее правило выражения вершины сохранения значения в память:

1. `mov [eax], 0 ; ret`
2. `pop ebx ; ret`
3. `add [eax], ebx ; ret`

Оуян (Ouyang) и др. [23] расширили набор инструкций языка QooL до полного по Тьюрингу набора. В целом они повторяют подход Шварца и др. [6] с построением семантических деревьев, используя при учете побочных эффектов анализ жизни значений. Следует отметить, что существуют попытки реализации метода Шварца и др., имеющие открытый исходный код [34, 36, 69].

8. Учет запрещенных символов

Автоматические инструменты генерации ROP цепочек должны учитывать особенности санитизации входных данных для конкретного эксплойта. Например, данные, копируемые через функцию `strcpy`, не могут содержать нулевые байты.

Байты, требующие санитизации, могут содержаться как в адресах гаждетов, так и в данных, загружаемых этими гаджетами на регистры. В простейшем случае санитизация адресов гаджетов производится путем отбрасывания гаджетов, содержащих запрещенные символы в адресе. Так поступают многие инструменты. Однако данный подход неизбежно приходит к уменьшению каталога гаджетов, что приводит к нехватке гаджетов и необходимости их комбинирования для моделирования недостающих гаджетов.

Гораздо сложнее обстоит ситуация, когда запрещенные символы содержатся в данных, предназначенных для загрузки на регистры (значения аргументов функций и необходимые для записи в память значения). Для решения данной проблемы можно использовать всевозможные арифметические операции для получения значений, содержащих запрещенные символы.

Подробное описание способов борьбы с запрещенными символами описано в статье Дина (Ding) и др. [22]. Стоит отметить, что авторы статьи борются со всеми непечатаемыми символами в цепочках, что может быть избыточно в некоторых случаях. Однако их методы применимы и в более общем случае произвольного заданного

множества запрещенных символов. Для каждого найденного гаджета авторы строят семантическое дерево, описывающее функциональность гаджета и содержащее явные зависимости между регистрами и памятью относительно арифметических операций и операций взаимодействия с памятью.

Построенные семантические деревья используются при построении конечного автомата (рис. 6), используемого для поиска инструкций, загружающих значение на регистр. Вершинами в конечном автомате являются следующие состояния, отвечающие разным загружаемым значениям:

- Z – ноль,
- SI – небольшое число,
- GC – число, не содержащее запрещенных символов,
- BC – число, содержащее запрещенные символы,
- T – конечное состояние.

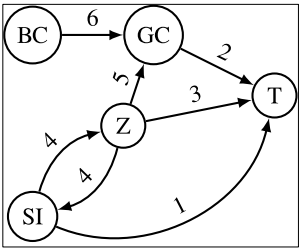

Рис. 6. Конечный автомат, описывающий алгоритм санитизации значений загрузки на регистр
Fig. 6. State machine describing the input sanitizing algorithm

Между этими вершинами проводятся ребра, соответствующие определенным гаджетам, при условии их наличия в каталоге гаджетов. Возможные варианты перехода между состояниями конечного автомата:

1. $SI \longrightarrow T$, из вершины с небольшим числом в конечное состояние ведет ребро, соответствующее гаджету с инструкцией, непосредственно устанавливающей это значение в регистре.
2. $GC \longrightarrow T$, из вершины с числом, не содержащим запрещенных символов, в конечное состояние ведет ребро с гаджетом `pop`.
3. $Z \longrightarrow T$, из вершины с нулем в конечное состояние ведет ребро с инструкцией `xor`.
4. $SI \longleftrightarrow Z$, из вершины с небольшим числом в нуль и обратно ведут ребра с инструкциями `inc, dec`.
5. $Z \longrightarrow GC$, из вершины с нулем в состояние с числом, не содержащим запрещенных символов, ведет ребро с арифметическими инструкциями `and, or, sal, shl, shr, sar`.
6. $BC \longrightarrow GC$, из вершины с числом, содержащим запрещенные символы, в вершину, не содержащую запрещенных символов, ведут ребра, состоящие из комбинации двух арифметических операций, например, $a + b - c$.

Работа алгоритма начинается из состояния, соответствующего значению, которое нужно установить в определенном регистре. Путем обхода состояний данного автомата решается вопрос возможной санитизации данных ROP цепочки. Алгоритм прерывается, если достигнуто конечное состояние, что соответствует успешному нахождению комбинации гаджетов, решающих поставленную задачу, или в случае отсутствия переходов в другие состояния из текущего.

9. Экспериментальное сравнение инструментов

Экспериментальная проверка инструментов с доступным исходным кодом проводилась с помощью тестовой системы rop-benchmark [37]. Данная система позволяет проверять работоспособность ROP цепочек, генерируемых инструментами. Система предоставляет воспроизводимое окружение для проверки факта успешной генерации и работоспособности ROP цепочек, осуществляющих системный вызов `execve("/bin/sh", 0, 0)`. Система тестирования поддерживает платформу Linux x86-64. В качестве тестовых наборов взяты исполняемые файлы и библиотеки минимальных установок нескольких популярных дистрибутивов: CentOS 7, Debian 10, OpenBSD 6.2, OpenBSD 6.4. Дистрибутивы OpenBSD 6.2 и 6.4 взяты по причине того, что авторы этой операционной системы ведут целенаправленную борьбу с ROP гаджетами [70].

Табл. 4. Экспериментальное сравнение инструментов автоматической генерации ROP цепочек
Table 4. The experimental evaluation of automatic ROP chain generating tools

Тестовый набор	Debian 10			CentOS 7			OpenBSD 6.2			OpenBSD 6.4		
Кол-во файлов	689			649			397			410		
Инструмент	OK	F	TL	OK	F	TL	OK	F	TL	OK	F	TL
ROPgadget [1]	7	0	0	8	0	0	4	0	0	2	0	0
angrop [2]	87	8	4	53	6	1	24	1	5	7	1	5
ROPGenerator [3]	83	4	20	66	5	3	24	3	13	1	0	12
Ropper [4]	53	33	0	31	37	0	14	14	1	1	0	2

Результаты экспериментальной проверки приводятся в таблице 4. Четыре столбца соответствуют четырем наборам тестовых файлов. В первой строке указано общее количество тестовых файлов в каждом из наборов. Ниже находятся строки с инструментами, и напротив каждого инструмента указана следующая информация:

- OK – количество тестовых файлов, для которых созданная ROP цепочка работоспособна, т.е. приводит к открытию оболочки системного интерпретатора.
- F – количество тестовых файлов, для которых созданная ROP цепочка не является работоспособной, т.е. по каким-то причинам не приводит к открытию оболочки системного интерпретатора.
- TL – количество тестовых файлов, на которых время работы инструмента превысило установленный лимит в 300 секунд.

В экспериментальное сравнение попали только находящиеся в открытом доступе инструменты, которые способны в полностью автоматическом режиме генерировать ROP цепочку, осуществляющую системный вызов для архитектуры x86-64 с операционной системой семейства Linux. Из-за операционной системы не попал в рассмотрение инструмент mona.py [29]. Другие могут работать только с архитектурой x86 (32-битной) [58], ARM [28]. Некоторые доступные инструменты не удалось успешно встроить в автоматизированную систему запуска [34].

В набор тестовой системы не вошли тесты с запрещенными символами (например, 0x00 в случае переполнения при копировании с `strcpy`) по причине того, что только ROPGenerator [31] поддерживает их в полном объеме, т.е. проверяет на наличие запрещенных символов не только адреса гаджетов, но и значения параметров гаджетов на стеке.

10. Заключение

В данной статье был проведен подробный обзор методов автоматизированной генерации эксплойтов для атак повторного использования кода. Атаки повторного использования кода предполагают использование кусочков кода из адресного пространства программы,

называемых *гаджетами*. Гаджеты связываются в цепочку, выполняющую вредоносную нагрузку. Схематично процесс генерации эксплойтов повторного использования кода делится на четыре этапа: поиск гаджетов в эксплуатируемой программе, определение семантики гаджетов, комбинация гаджетов в цепочки и генерация входных данных, эксплуатирующих уязвимость. Найденные в программе гаджеты добавляются в каталог гаджетов. После этого происходит определение семантики гаджетов: классификация гаджетов по параметризованным семантическим типам, составление резюме гаджетов или построение графов зависимостей гаджетов. Если набор гаджетов в каталоге полон по Тьюрингу, то гаджеты из каталога можно использовать в качестве целевой архитектуры набора команд компилятора. Связывание гаджетов в цепочки может происходить как поиском гаджетов по шаблонам, задаваемых регулярными выражениями, так и с учетом семантики гаджета. Также существуют подходы конструирования ROP цепочек с использованием генетических алгоритмов, а также методы с использованием SMT-решателей.

Мы предложили набор тестов ROP Benchmark [37] для экспериментального сравнения инструментов генерации ROP цепочек. С помощью него было проведено сравнение инструментов генерации ROP цепочек с открытым исходным кодом для платформы Linux x86-64. В том числе сравнение производилось на дистрибутивах операционной системы OpenBSD, авторы которой целенаправленно ведут борьбу с ROP гаджетами [70].

Список литературы / References

[1]. The Heartbleed bug. URL: http://heartbleed.com.

[2]. D. Halperin, T. S. Heydt-Benjamin, B. Ransford, S. S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. H. Maisel. Pacemakers and implantable cardiac defibrillators: software radio attacks and zero-power defenses. In Proc. of the IEEE Symposium on Security and Privacy, 2008, pp. 129–142.

[3]. 2019 CWE top 25 most dangerous software errors. URL: https:// cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html.

[4]. A. Peslyak. Getting around non-executable stack (and fix). Bugtraq mailing list archives, Aug. 1997. URL: https://seclists.org/bugtraq/1997/Aug/63.

[5]. H. Shacham. The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86). In Proc. of the 14th ACM Conference on Computer and Communications Security, 2007, pp. 552–561.

[6]. E.J. Schwartz, T. Avgerinos, and D. Brumley. Q: exploit hardening made easy. In Proc, of the 20th USENIX Conference on Security, 2011, 16 p.

[7]. R. Roemer, E. Buchanan, H. Shacham, and S. Savage. Return-oriented programming: systems, languages, and applications. ACM Transactions on Information and System Security, vol. 15, no. 1, 2012, pp. 2:1–2:34.

[8]. T. Kornau. Return oriented programming for the ARM Architecture. Master's thesis, Ruhr-University, Bochum, Germany, 2009.

[9]. S. Checkoway, L. Davi, A. Dmitrienko, A.-R. Sadeghi, H. Shacham, and M. Winandy. Return-oriented programming without returns. In Proc. of the 17th ACM Conference on Computer and Communications Security, 2010, pp. 559–572.

[10]. L. Davi, A. Dmitrienko, A.-R. Sadeghi, and M. Winandy. Return-oriented programming without returns on ARM. Technical Report HGI-TR2010-002, Ruhr-University, Bochum, Germany, 2010.

[11]. Z.-S. Huang and I. G. Harris. Return-oriented vulnerabilities in ARM executables. In Proc. of the IEEE Conference on Technologies for Homeland Security, 2012, pp. 1–6.

[12]. O.L. Fraser, N. Zincir-Heywood, M. Heywood, and J.T. Jacobs. Return-oriented programme evolution with ROPER: a proof of concept. In Proc. of the Genetic and Evolutionary Computation Conference Companion, 2017, pp. 1447–1454.

[13]. E. Buchanan, R. Roemer, H. Shacham, and S. Savage. When good instructions go bad: generalizing return-oriented programming to RISC. In Proc. of the 15th ACM Conference on Computer and Communications Security, 2008, pp. 27–38.

[14]. A. Francillon and C. Castelluccia. Code injection attacks on harvard-architecture devices. In Proc. of the 15th ACM Conference on Computer and Communications Security, 2008, pp. 15–26.
[15]. F. Lindner. Cisco IOS router exploitation. In Black Hat USA, 2009. URL: https://www.blackhat.com/presentations/bh-usa-09/LINDNER/BHUSA09-Lindner-RouterExploit-PAPER.pdf.
[16]. S. Checkoway, A. J. Feldman, B. Kantor, J. A. Halderman, E. W. Felten, and H. Shacham. Can DREs provide long-lasting security? The case of return-oriented programming and the AVC advantage. In Proc. of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections, 2009, 16 p.
[17]. T. Bletsch, X. Jiang, V. W. Freeh, and Z. Liang. Jump-oriented programming: a new class of code-reuse attack. In Proc. of the 6th ACM Symposium on Information, Computer and Communications Security, 2011, pp. 30–40.
[18]. P. Chen, X. Xing, B. Mao, L. Xie, X. Shen, and X. Yin. Automatic construction of jump-oriented programming shellcode (on the x86). In Proc. of the 6th ACM Symposium on Information, Computer and Communications Security, 2011, pp. 20–29.
[19]. A. Sadeghi, S. Niksefat, and M. Rostamipour. Pure-call oriented programming (PCOP): chaining the gadgets using call instructions. Journal of Computer Virology and Hacking Techniques, vol. 14, no. 2, pp. 139–156.
[20]. R. Hund, T. Holz, and F. C. Freiling. Return-oriented rootkits: bypassing kernel code integrity protection mechanisms. In Proc. of the 18th Conference on USENIX Security Symposium, 2009, pp. 383–398.
[21]. N.A. Quynh. OptiROP: hunting for ROP gadgets in style. URL: https://media.blackhat.com/us-13/US-13-Quynh-OptiROP-Hunting-for-ROP-Gadgets-in-Style-Slides.pdf.
[22]. W. Ding, X. Xing, P. Chen, Z. Xin, and B. Mao. Automatic construction of printable return-oriented programming payload. In Proc. of the 9th International Conference on Malicious and Unwanted Software: The Americas, 2014, pp. 18–25.
[23]. Y. Ouyang, Q. Wang, J. Peng, and J. Zeng. An advanced automatic construction method of ROP. Wuhan University Journal of Natural Sciences, vol. 20, no. 2, 2015, pp. 119–128.
[24]. A. Follner, A. Bartel, H. Peng, Y.-C. Chang, K. Ispoglou, M. Payer, and E. Bodden. PSHAPE: automatically combining gadgets for arbitrary method execution. Lecture Notes in Computer Science, vol. 9871, 2016, pp. 212–228.
[25]. B. Milanov. ROPGenerator: practical automated ROP-chain generation, 2018. URL: https://youtu.be/rz7Z9fBLVs0.
[26]. N. Mosier and P. Johnson. ROP with a 2nd stack, 2019. URL: http://www.cs.middlebury.edu/~nmosier/portfolio/rsrc/ropc-slides.pdf.
[27]. A. Follner, A. Bartel, H. Peng, Y.-C. Chang, K. Ispoglou, M. Payer, and E. Bodden. PSHAPE - practical support for half-automated program exploitation. URL: https://github.com/Alexandre-Bartel/inspector-gadget.
[28]. O.L. Fraser. ROPER: a genetic ROP-chain development tool. URL: https: //github.com/oblivia-simplex/roper.
[29]. mona.py, Corelan Consulting BVBA. URL: https://github.com/corelan/mona.
[30]. J.Salwan. ROPgadgettool. URL: https://github.com/JonathanSalwan/ROPgadget.
[31]. B. Milanov. ROPGenerator. URL: https://github.com/Boyan-MILANOV/ropgenerator.
[32]. C. Salls. angrop. URL: https://github.com/salls/angrop.
[33]. S. Schirra. Ropper. URL: https://github.com/sashs/ropper.
[34]. Paul. ROPC. URL: https://github.com/pakt/ropc.
[35]. ropc-llvm, Programa STIC. URL: https://github.com/programa-stic/ropc-llvm.
[36]. J. Stewart. An open source, multi-architecture ROP compiler. URL: https:// github.com/jeffball55/rop_compiler.
[37]. A. Nurmukhametov. ROP Benchmark. URL: https://github.com/ispras/rop-benchmark.
[38]. J. Salwan. An introduction to the return oriented programming and ROP-chain generation, 2014. URL: http://shell-storm.org/talks/ROP_ course_lecture_jonathan_salwan_2014.pdf.
[39]. T. F. Dullien. Weird machines, exploitability, and provable unexploitability. IEEE Transactions on Emerging Topics in Computing (Early Access), 2017, 15 p.
[40]. M. Graziano, D. Balzarotti, and A. Zidouemba. ROPMEMU: a framework for the analysis of complex code-reuse attacks. In Proc. of the 11th ACM on Asia Conference on Computer and Communications Security, 2016, pp. 47–58.

[41]. E.J. Schwartz, T. Avgerinos, and D. Brumley. Update on Q: exploit hardening made easy, 2012. URL: https://edmcman.github.io/papers/usenix11-update.pdf.

[42]. G. F. Roglia, L. Martignoni, R. Paleari, and D. Bruschi. Surgically returning to randomized lib(c). In Proc. of the Annual Computer Security Applications Conference, 2009, pp. 60–69.

[43]. N.R. Weidler, D. Brown, S.A. Mitchell, J. Anderson, J.R. Williams, A. Costley, C. Kunz, C. Wilkinson, R. Wehbe, and R. Gerdes. Return-oriented programming on a resource constrained device. Sustainable Computing: Informatics and Systems, vol. 22, 2019, pp. 244-256.

[44]. Вишняков А.В. Классификация ROP гаджетов. Труды ИСП РАН, том 28, вып. 6, 2016, стр. 27-36 / Vishnyakov A.V. Classification of ROP gadgets. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 6, 2016, pp. 27-36 (in Russian). DOI: 10.15514/ISPRAS-2016-28(6)-2.

[45]. Вишняков А.В, Нурмухаметов А.Р., Курмангалеев Ш.Ф., Гайсарян С.С. Метод анализа атак повторного использования кода. Труды ИСП РАН, том 30, вып. 5, 2018 г., стр. 31-54 / Vishnyakov A.V., Nurmukhametov A.R., Kurmangaleev Sh.F., Gaisaryan S.S. Method for analysis of code-reuse attacks. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018, pp. 31-54 (in Russian). DOI: 10.15514/ISPRAS-2018-30(5)-2.

[46]. T. Avgerinos, S.K. Cha, B.L.T. Hao, and D. Brumley. AEG: automatic exploit generation. In Proc. of the Network and Distributed System Security Symposium, 2011, pp. 283– 300.

[47]. S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley. Unleashing Mayhem on binary code. In Proc. of the IEEE Symposium on Security and Privacy, 2012, pp. 380–394.

[48]. V.A. Padaryan, V.V. Kaushan, and A.N. Fedotov. Automated exploit generation for stack buffer overflow vulnerabilities. Programming and Computer Software, vol. 41, no. 6, 2015, pp. 373–380.

[49]. Федотов А.Н., Падарян В.А., Каушан В.В., Курмангалеев Ш.Ф., Вишняков А.В., Нурмухаметов А.Р. Оценка критичности программных дефектов в рамках работы современных защитных механизмов. Труды ИСП РАН, том 28, вып. 5, 2016 г., стр. 73-92 / Fedotov A.N., Padaryan V.A., Kaushan V.V., Kurmangaleev Sh.F., Vishnyakov A.V., Nurmukhametov A.R. Software defect severity estimation in presence of modern defense mechanisms. Trudy ISP RAN/Proc. ISP RAS, vol. 28, issue 5, 2016. pp. 73-92 (in Russian). DOI: 10.15514/ISPRAS-2016-28(5)-4.

[50]. Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna. SOK: (state of) the art of war: offensive techniques in binary analysis. In Proc. of the IEEE Symposium on Security and Privacy, 2016, pp. 138–157.

[51]. J.C. King. Symbolic execution and program testing. Communications of the ACM, vol. 19, no. 7, 1976, pp. 385– 394.

[52]. E. J. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In Proc. of the IEEE Symposium on Security and Privacy, 2019, pp. 317– 331.

[53]. P. Godefroid, M. Y. Levin, and D. A. Molnar. Automated whitebox fuzz testing. In Proc. of the 16th Annual Network & Distributed *System* Security Symposium, 2008, pp. 151–166.

[54]. A. Homescu, M. Stewart, P. Larsen, S. Brunthaler, and M. Franz. Microgadgets: size does matter in turing-complete return-oriented programming. In Proc. of the 6th USENIX Workshop on Offensive Technologies, 2012, 13 p.

[55]. M. Tran, M. Etheridge, T. Bletsch, X. Jiang, V. Freeh, and P. Ning. On the expressiveness of return-into-libc attacks. Lecture Notes in Computer Science, vol. 6961, 2011, pp. 121–141.

[56]. BARF: binary analysis and reverse engineering framework. URL: https://github.com/programa-stic/barf-project.

[57]. N. Mosier. A pair of return-oriented programming utilities: a gadget finder and ROP compiler. URL: https://github.com/nmosier/rop-tools.

[58]. SQLab ROP payload generation. URL: https://github.com/SQLab/ropchain.

[59]. E. Göktas, B. Kollenda, P. Koppe, E. Bosman, G. Portokalidis, T. Holz, H. Bos, and C. Giuffrida. Position-independent code reuse: on the effectiveness of ASLR in the absence of information disclosure. In Proc. of the IEEE European Symposium on Security and Privacy, *2018*, pp. 227–242.

[60]. I. Jager and D. Brumley. Efficient Directionless Weakest Preconditions. Technical Report CMU-CyLab-10-002, Carnegie Mellon University, CyLab, 2010.

[61]. N. Nethercote and J. Seward. Valgrind: a framework for heavyweight dynamic binary instrumentation. SIGPLAN Notice, vol. 42, no. 6, 2007, pp. 89–100.

[62]. T. Dullien and S. Porst. REIL: a platform-independent intermediate representation of disassembled code for static code analysis, 2009.

[63]. В.А. Падарян, М.А. Соловьев, А.И. Кононов. Моделирование операционной семантики машинных инструкций. Труды ИСП РАН, том 19, 2010 г., стр. 165–186 / V.A. Padaryan, M.A. Soloviev, and A.I. Kononov. Modeling operational semantics of machine instructions. Trudy ISP RAN/Proc. ISP RAS, vol. 19, 2010, pp. 165–186 (in Russian).

[64]. C. Heitman and I. Arce. BARF: a multiplatform open source binary analysis and reverse engineering framework. En los Materiales del XX Congreso Argentino de Ciencias de la Computación, 2014, 10 p.

[65]. А.В. Вишняков. Верификация семантики линейной последовательности машинных инструкций. Курсовая работа, МГУ им. М.В. Ломоносова, ф-т ВМК, 2019 г. / A. V. Vishnyakov. Semantic verification of linear machine instruction sequence. Course Paper, M.V. Lomonosov Moscow State University, faculty of CMC, 2019 (in Russian).

[66]. A. Follner, A. Bartel, and E. Bodden. Analyzing the gadgets: towards a metric to measure gadget quality. Lecture Notes in Computer Science, vol. 9639, 2016, pp. 155– 172.

[67]. E.W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, vol. 1, no. 1, 1959, pp. 269–271.

[68]. A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. Code Generation by Tiling an Input Tree. In Compilers: principles, technologies, and tools. Addison Wesley, 2th edition, 2006, pp. 560–563.

[69]. J. Stewart and V. Dedhia. ROP compiler. URL: https://css.csail.mit.edu/6.858/2015/projects/je25365-ve25411.pdf.

[70]. T. Mortimer. Removing ROP gadgets from OpenBSD. In Proc. of the AsiaBSDCon, 2019, pp.13–21.

Информация об авторах / Information about authors

Алексей Вадимович ВИШНЯКОВ работает в отделе компиляторных технологий ИСП РАН, закончил бакалавриат ВМК МГУ, сейчас там же получает степень магистра. Сфера научных интересов: компьютерная безопасность, возвратно-ориентированное программирование, анализ бинарного кода, символьная интерпретация, обратная инженерия и компиляторы.

Alexey Vadimovich VISHNYAKOV works for the Compiler Technology Department at ISP RAS, obtained BSc degree in the Faculty of Computational Mathematics and Cybernetics at Lomonosov Moscow State University. He is a M.D. student in the same faculty now. Research interests: computer security, return-oriented programming, binary analysis, symbolic execution, reverse engineering, and compilers.

Алексей Раисович НУРМУХАМЕТОВ – младший научный сотрудник отдела компиляторных технологий ИСП РАН, закончил МФТИ по специальности прикладные математика и физика в 2013 году. Сферой научных интересов являются: компиляторы, компьютерная безопасность, возвратно-ориентированное программирование.

Aleksei Raisovich NURMUKHAMETOV is a research fellow at the ISP RAS, the Compiler Technology Department. He obtained both his BSc and MSc degrees in applied mathematics and physics at the Moscow Institute of Physics and Technology in 2011 and 2013, respectively. He has a strong interest in compilers, computer security, return-oriented programming.

DOI: 10.15514/ISPRAS-2019-31(6)-7

CC BY

Кэширование данных в мультиконтейнерных системах

[1] *Д.А. Грушин, ORCID: 0000-0002-6789-5473 <grushin@ispras.ru>*
[1,2] *Д.О. Лазарев, ORCID: 0000-0002-6253-6447 <denis.lazarev@phystech.edu>*
[1,2] *С.А. Фомин, ORCID: 0000-0002-1151-2189 <fomin@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, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*

Аннотация. Сегодня виртуализация – это ключевая технология облачных вычислений и современных центров обработки данных, обеспечивающая масштабируемость и безопасность, управление глобальной ИТ-инфраструктурой и снижение затрат. Среди методов виртуализации, наиболее популярной стала контейнеризация – изоляция связанных групп процессов, разделяющих общее ядро операционной системы. Эффективность контейнеризации в сравнении с классической аппаратной виртуализацией, проявляется в компактности контейнеров и меньших накладных затратах вычислительных ресурсов – памяти, диска, ЦПУ. Однако в сравнении с классическими архитектурами без изоляции процессов контейнеры могут обходиться дороже, и в любом случае, индустрия ждет дополнительной оптимизации – скорости запуска, экономии памяти и дискового пространства и других ресурсов. В этом может помочь различное кеширование – старейший механизм повышения производительности программ без радикальной модификации алгоритма и оборудования. Однако при этом возникают различные архитектурно-инженерные дилеммы вида «безопасность или эффективность» и здесь мы рассмотрим современные научно-технические подходы к их решению в разных аспектах – ускорение запуска, оптимизация совместного использования, ускорение сборки образов, а также некоторые проблемы безопасности, возникшие из-за агрессивного кэширования в процессорных архитектурах. А в некоторых сценариях использования мультиконтейнерных систем наоборот, скорость и задержки не важны, важно обеспечить максимальную загрузку физических серверов – в этом случае актуальны алгоритмы планирования и размещения контейнеров, и нами приведен обзор теоретических работ на эту тему.

Ключевые слова: контейнеры; кеширование; облачные вычисления

Для цитирования: Грушин Д.А., Лазарев Д.О., Фомин С.А. Кэширование данных в мультиконтейнерных системах. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 125–144. DOI: 10.15514/ISPRAS–2019–31(6)–7

Благодарности: Исследования, результаты которых представлены в этой статье, выполнены при поддержке Российского фонда фундаментальных исследований (проект 17-07-01006).

Data caching in multi-container systems

[1] *D.A. Grushin, ORCID: 0000-0002-6789-5473 <grushin@ispras.ru>*
[1,2] *D.O. Lazarev, ORCID: 0000-0002-6253-6447 <denis.lazarev@phystech.edu>*
[1,2] *S.A. Fomin, ORCID: 0000-0002-1151-2189 <fomin@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, 141700, Russia*

Abstract. Today, virtualization is a key technology for cloud computing and modern data centers, providing scalability and security, managing the global IT infrastructure and reducing costs. Among the methods of virtualization, the most popular was containerization, that is the isolation of related groups of linux processes that share a common Linux kernel. Containerization is more profitable that classical hardware virtualization because of compactness of containers and lower overhead costs of memory, disk, CPU. However, in comparison with classical architectures without process isolation containers can cost more, and in any case, the industry is waiting for additional optimization – the speed of launch, saving memory and disk space and other resources. Different caching techniques can help in this, because Caching is the oldest mechanism of increasing software productivity without radical modification of algorithms and hardware. However, there are a lot of architectural and engineering tradeoffs. Here we will consider modern scientific and technical approaches to their solution in different aspects – acceleration of launch, optimization of shared usage, acceleration of building container images, as well as some security problems caused by aggressive caching in modern processor architectures. And in some use cases for multi-container systems performance and latency are not important, but we have to ensure the maximum load of physical servers. In these cases, the algorithms of planning and placement of containers are relevant, and we give an overview of theoretical work on this topic.

Keywords: containers; caching; cloud computing

For citation: Grushin D.A., Lazarev D.O., Fomin S.A. Data caching in multi-container systems. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 125-144 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-7

Acknowledgements: The studies, the results of which are presented in this article, were supported by the Russian Foundation for Basic Research (project 17-07-01006).

1. Введение

Кеширование – один из важнейших механизмов повышения производительности, без радикальной модификации алгоритма и оборудования. К сожалению, в реальности не существует компьютеров, реализующих теоретическую модель RAM – Random Access Machine, машину с произвольным доступом к памяти, способную за один условных такт дотянутся до любой ячейки однородной памяти. В реальных вычислениях существует очень строгая иерархия доступа к памяти, от наносекунд к кэшам первого уровня процессора, до секунд доступа к удаленным сетевых хранилищам на шпиндельных жестких дисках.

Как пример, можно привести ориентировочные времена доступа к различным устройствам (иерархия – кэши процессора, память, ssd-диски, сеть, … [1], табл. 1).

Видно, что каждый уровень «попадания в кеш» (не важно, речь идет о диске при кэшировании сети, или про кэш первого уровня при кэшировании второго уровня), может увеличить производительность на порядки. Кроме того, кеширование бывает и вычислительным, т.е. обеспечивается быстрый доступ не только к элементам, хранящимся на более «низком» уровне систем хранения, но и сохраненным результатам вычисления, требующим серьезных временных затрат. Кеширование – старая

технология, термин «cache» появился в 1967 году, но практически этот подход – ровесник «электронно-вычислительных машин».

Табл. 1. Иерархия типов памяти и задержки доступа
Table 1. Hierarchy of memory types and access delays

Операция	Задержка (в наносек)
Доступ к кэшу первого уровня	0.5
Доступ к кэшу второго уровня	7
Доступ к DRAM	100
3D XPoint на основе чтения NVMe SSD	10000
NAND NVMe SSD R/W	20000
NAND SATA SSD R/W	50000
Случайное чтение блоков 4K с SSD	150000
Задержка P2P TCP/IP (физика на физику)	150000
Задержка P2P TCP/IP (ВМ на ВМ)	250000
Последовательное чтение 1MB из памяти	250000
Сетевая задержка внутри ЦОД	500000
Последовательное чтение 1 MB с SSD	1000000
Поиск диска	10000000
Последовательное чтение 1 MB с диска	20000000
Отправка пакетов США → Европа → США	150000000

С другой стороны, контейнеры – недавно появившаяся технология обеспечения дешевой изоляции групп процессов, давшая огромный развитие IT-индустрии, родившая такие практики как DevOps и микросервисная архитектура. Конечно, сама идея некоторой изоляции процессов на уровне операционной системы не нова – «chroot» был введен в 1982 году, но реальная изоляция процессов появилась только в начале нулевых с FreeBSD Jail, эволюционировала к концу нулевых уже к промышленным технологиям LXC, Solaris Containers и OpenVZ/Virtuozzo, но только с появлением Docker в середине десятых годов нашего века стала поистине массовой, настолько, что большинство ITшников при слове «контейнеры» вспоминают только технологии от «докер», что безусловно обидно для первопроходцев контейнеризации.

Контейнеры практически окончили войну между разработчиками и системными администраторами, дав возможность первым практически никак не зависеть от тонкостей целевых платформ, и их особенностей – будь то операционная система и ее дистрибутив, или оборудование с его глюками. Теперь, для каждой выкатки новой версии не требуется многомесячное согласование «отдела разработки» и «отдела администрирования», появилась возможность выкатывать в «production» новые версии ежечасно, более того, появились новые архитектуры оркестрации контейнеров, позволяющие экономить на долгом цикле тестирования и поддерживать в работе несколько версий сервиса, автоматически переключаясь на проверенную версию, в случае проблем с новой. Все это, а также тенденция к глобализации вычислительных ресурсов, обозначаемых сильно заезженным маркетинговым термином «облака», сильно уронило престиж и ценность «бородатых сисадминов», хранителей знаний о взаимодействии зоопарка различного оборудования и операционных систем. Большинству из них пришлось переквалифицироваться в DevOps инженеров,

специалистов по сборке контейнеров, развертывании вычислительной инфраструктуры, и они стали близки с разработчиками как по навыкам – «программирование инфраструктуры как кода» вместо беготни с оборудованием, так и по взаимодействию – DevOps специалистов принято размещать вместе с командой разработки.

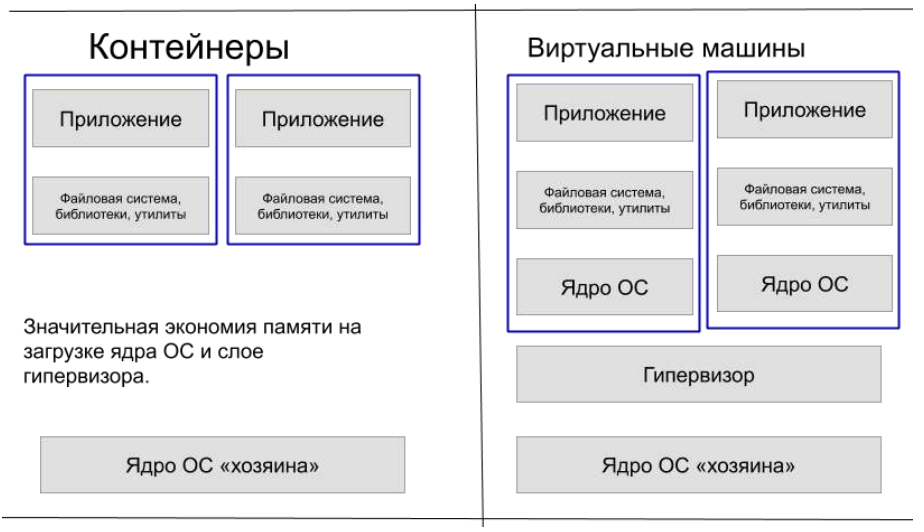

Рис. 1. Архитектурная выгода контейнеров в сравнении с классической виртуализацией
Fig. 1. The architectural benefits of containers versus classic virtualization

Даже если не используются микросервисные архитектуры и используется классический подход «виртуальная машина с сервисами», зачастую предоставляемая провайдерами виртуальная машина – это тоже контейнер (LXC или OpenVZ), ведь таким образом достигается серьезная экономии памяти (все контейнеры используют только одно ядро ОС), и достигается их большая плотность размещения на физических серверах (рис. 1).

Но разумеется, весь этот тренд на изоляцию и дублирование процессов, обеспеченный контейнеризацией находится в некоторой оппозиции к эффективности использования памяти и ресурсов, примерно в такой же, в которой «конфликтуют» архитектурные аспекты «эффективности» и «безопасности». Соответственно, возникают инженерные и научные проблемы, как решить эти технологические дилеммы, как сохранить эффективность при использовании контейнеров, какого рода кеширование, и в каком случае, может тут помочь.

Далее мы приведем несколько разделов с обзором различных аспектов темы «контейнеры и кеши», рассмотрев современные научно-практические подходы к их решению в разных аспектах – ускорение запуска, оптимизация совместного использования, ускорение сборки образов, а также некоторые проблемам безопасности, возникшим из-за агрессивного кэширования в процессорных архитектурах.

В некоторых сценариях использования мультиконтейнерных систем наоборот, скорость и задержки не важны, важно обеспечить максимальную загрузку физических серверов – в этом случае актуальны алгоритмы планирования и размещения контейнеров – и мы привели обзор некоторых теоретических алгоритмов, еще не реализованных в системах виртуализации или контейнерной оркестрации.

2. Межконтейнерное совместное использование файлов

Несмотря на то, что контейнеры сильно выигрывают в плотности размещения по сравнению с виртуальными машинами, есть множество областей, где сложились практики обеспечивающие еще большую плотность сервисов. Одна из таких областей – так называемый shared-хостинг, т.е. хостинг, в котором все сайты живут внутри одной операционной системы (Linux), пользуясь общим сервисом СУБД (обычно MySQL) под разными аккаунтами, общим веб-сервером (обычно связка Apache+PHP), так называемым классическим веб-стеком LAMP, под которым до сих пор работают большинство популярных CMS, Content Management Systems, систем управления контентом – таких как Wordpress, Drupal, Joomla, Bitrix, Mediawiki …, т.е. все те же фреймворки, на которых до сих пор держится большинство сайтов – от блогов и сайтов визиток, до порталов сообществ и интернет магазинов (рис. 2).

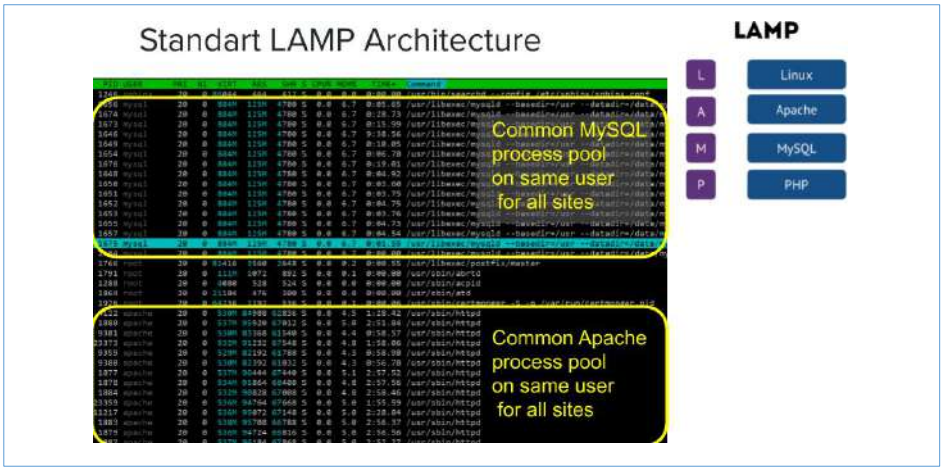

Рис. 2. Архитектура LAMP – Linux, Apache, MySQL, PHP
Fig. 2. LAMP architecture – Linux, Apache, MySQL, PHP

Плотность размещения сайтов внутри при разделяемом хостинге (*шаред* хостинге, до сих пор нет устоявшегося русского перевода для термина *shared hosting*) может быть на порядок больше, чем при хостинге отдельных сайтов внутри виртуальных машин и даже контейнеров, и по сути, ограничена только совестью провайдеров. Обычно, провайдеры пользуются тем, что такие дешевые услуги покупают владельцы слабо посещаемых сайтов – непопулярные блоги, сайты-визитки малоизвестных компаний, хобби-проекты, и часто им удается размещать до десятков тысяч сайтов внутри одного физического сервера.

Однако эта дешевизна имеет и оборотную сторону. В виду отсутствия настоящей изоляции процессов, один сайт получивший внезапную нагрузку, «выедает» все вычислительные ресурсы физического сервера (процессор, память, дисковые IOPSы, сеть) и «кладет», т.е. вызывает отказ в обслуживании у всех остальных сайтов на том же сервере. Причем эта нагрузка может быть не обязательно в результате DDOSa, возможно просто внезапный всплеск посещаемости, когда ссылка на сайт попадает на популярных ресурс («reddit-эффект» или «хабра-эффект», по названию известных коллективных блогов, по ссылке с которых могут пойти одновременно десятки тысяч пользователей). Контейнеризация, т.е. изоляция в отдельных контейнерах хотя бы процессов вебсервера могла бы решить эту проблему, т.к. для отдельного контейнера можно поставить ограничения по потреблению CPU и памяти.

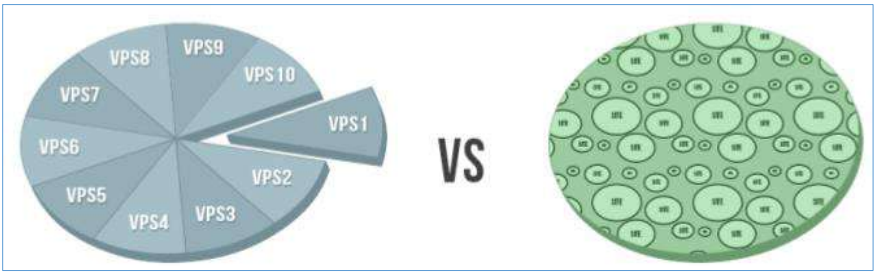

Рис. 3 Преимущество разделяемого хостинга – в плотности
Fig. 3 The advantage of shared hosting is density

Да, контейнеры были бы лучше, если бы их удалось сделать более экономичными, не дублирующими ни общие файлы библиотек и системных процессов (glibc, apache-nginx-php-python-mysql-postgres), ни развесистые файлы прикладных фреймворков на PHP и других языках, которые могут занимать сотни мегабайтов дискового пространства, и, что более вредно, одни и те же файлы в разных контейнерах, занимают свое собственное место в страничном кэше файловой системы в оперативной памяти (рис. 3)! Аналогичная проблема возникает и вне шаредхостинга, в микросервисной архитектуре, когда приложение разбито на десятки микросервисов, каждый из которых тянет с собой как минимум ограниченную версию дистрибутива и легкий веб-сервер (обычно nginx), а максимум… не ограничен, и многогигабайтные образы контейнеров – вполне не редкость.

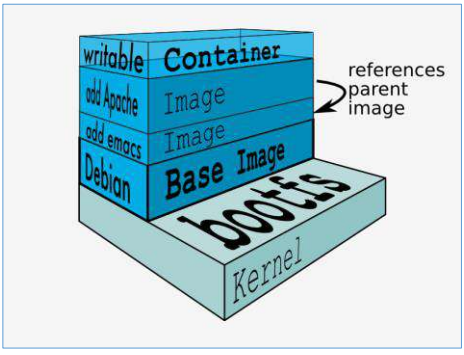

Рис. 4. Слои overlayfs в docker-контейнерах
Fig. 4. Layers of overlayfs in docker containers

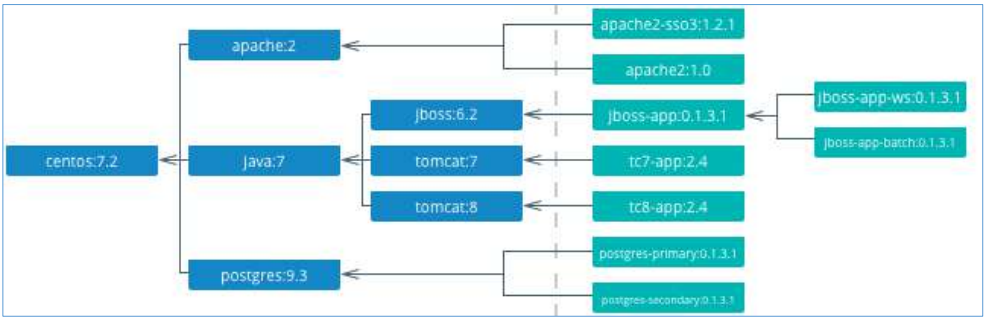

Рис. 5. Иерархия и наследование docker-образов
Fig. 5. Hierarchy and inheritance of docker images

С проблемой экономии дискового пространства удалось справиться технологии Docker-образов, использующих с одной стороны, технологию «слоистой» файловой системы overlay-fs (рис. 4), когда можно добавлять свои «дельта-слои», добавляющие (или удаляющие) какие-то файлы к уже известным и неизменным образам, с другой – выстраивая эти образы в иерархию зависимостей (рис. 5), плюс инфраструктура построения и распространения таких образов (docker-реестр).

Однако эта технология, overlayfs, не подходит для классического шаредхостинга, где не очень продвинутый пользователь может только «положить и поправить» PHP-файлы (несмотря на то, что в сотнях соседних контейнеров они уже есть, и есть общий докер-образ, включающих нужный PHP-фреймворк), либо где такие общие файлы образуются множеством других образов в обход технологии докер-образов. Или докер вовсе не используется, а используется OpenVZ, LVX или Virtuozzo. Ну и в целом, нельзя заранее выстроить целостную иерархию файловых слоев, каждый контейнер управляется непредсказуемо и независимо, но в целом, очень много одинаковых файлов и очень хотелось бы что-то с этим сделать.

В таком случае, можно использовать технологию PFCache, VZFs, или аналогичную, позволяющую дедуплицировать операции над файлами. Идея этого подхода состоит в следующем. Файловые системы для процессов внутри контейнера создавать не напрямую на блочных устройствах хоста, а на специальном промежуточном блочном устройстве (ПБУ).

Это ПБУ можно создать (рис. 6)

- на обычных дисках хоста с помощью Union FS;
- в файлах хостовой системы, с помощью Loop device или Ploop, ZFS ZVol или подтомов BTRFS;
- LVM-разделов на блочном устройстве хоста;
- на сетевом кластере, с помощью, например, Ceph RBD.

В любом случае, у нас возникает потенциальная возможность контроля за системными файловыми вызовами внутри контейнера.

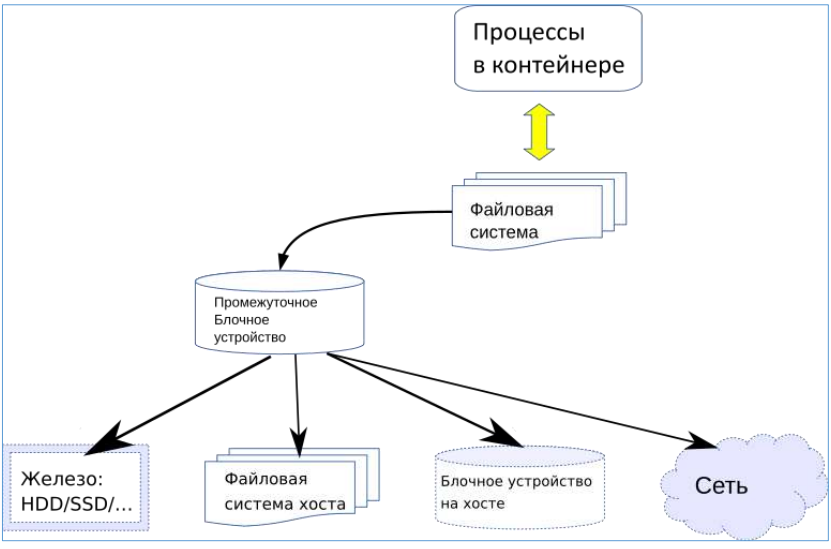

Рис. 6. Промежуточное блочное устройство для перехвата обращений к файлам
Fig. 6. Intermediate block device for intercepting file access

Далее заводится дисковый кеш на отдельном ПБУ, выделенном под эту задачу (на быстром устройстве, например, SSD NMVE). Этим ПБУ будут пользоваться все контейнеры/VPS внутри узла – физического или виртуального сервера. Затем, для процессов в контейнерах используется файловая система Ext4, где каждый файл, кроме имени, может нести дополнительные атрибуты в метаинформации (xattr) (рис. 7).

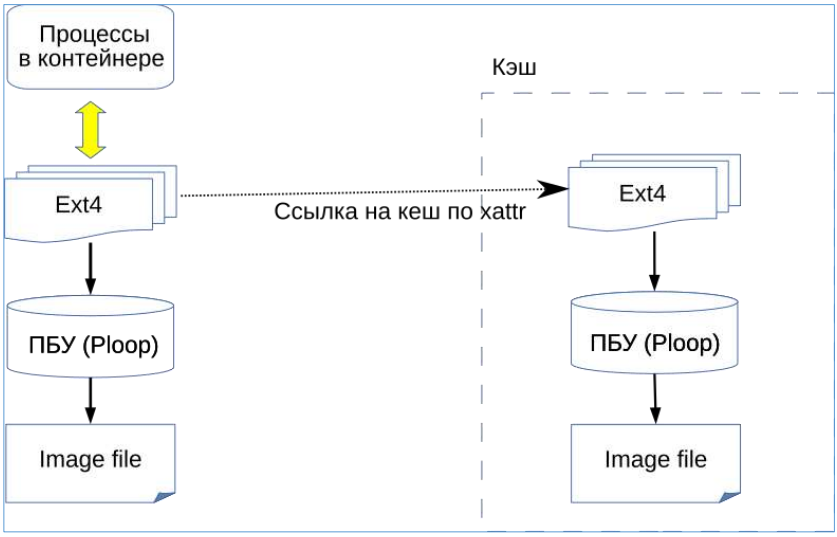

Рис. 7. Кэширование обращений к файлам с использованием файловых атрибутов ext4
Fig. 7. Caching file access using ext4 file attributes

В процессе эксплуатации происходит следующее.

1. На уровне контейнера, специальный пользовательский сервис прописывает SHA1 хеши файлов в их мета-атрибут *xattr*.
2. На уровне ядра:
   a. нужно периодически собирать статистику чтения файлов из ядра, анализирует ее, и добавлять файлы в кэш, если их использование частое;
   b. при чтении файла проверяется, содержит ли он указанный хеш в расширенных атрибутах xattr; если содержит – открывается «общий» файл, вместо файла контейнера;
   c. при записи в файл хеш инвалидируется; таким образом, при последующем открытии будет открываться уже непосредственно файл контейнера, а не его кеш.

В результате, держа в страничном кэше в основном «общие файлы» для всех контейнеров, получается значительная экономия как самого кэша и, следовательно, оперативной памяти, а также существенной экономии на дисковых операциях, ведь вместо чтения десяти файлов с диска, читается один, который сразу идет в страничный кэш.

Более детально, для технологии Pfcache с конкретными командами администрирования, этот процесс рассмотрен в статье [2], для технологии VZFS в [3], но аналогичный подход вполне может быть реализован и самостоятельно.

3. Кеширование старта контейнера

Контейнеры, автоматически запускаемые системой оркестрации, такой как Kubernetes, дали «путевку в жизнь» не только микросервисной архитектуре, но и такому модному

тренду как «serverless», когда разработчики практически ничего не знают о целевой платформе, и реализуют только отдельные, максимально атомарные функции на удобных для них языках программирования. Всю остальную работу – запуск этих функций в изолированных контейнерах, балансировка, и самое главное – автоматическое масштабирование – делает за них система оркестрации. Т.е. больше нет необходимости в редких специалистах по «highload»-архитектуре, снобах-архитекторах, умеющих рисовать диаграммы балансировки, или слишком дорогих разработчиках, изобретающих свои велосипеды по динамическому масштабированию числа сервисов при росте нагрузке. Все это может взять на себя система оркестрации, и скорее всего, это будет самый популярный за последние годы, проект Kubernetes.

Но есть еще один момент на переднем крае «технологических проблем» – проблема задержки при холодном старте. Действительно, при внезапном росте нагрузки (приход новых пользователей, неожиданный всплеск), система оркестрации автоматически поднимет дополнительные контейнеры для обслуживания… но сам старт этих контейнеров – дело не быстрое. Тут и старт самого контейнера (что, например, в docker, далеко не мгновенно), и самое главное – старт собственно сервисов внутри контейнера. Не важно, написаны они на «медленном» python, или «быстрых» NodeJS или Java, старт их может занимать сотни миллисекунд, а то и секунды/минуты. Если для обычного сайта, и даже интернет-магазина это наверное не является критической проблемой, то есть множество сервисов, обычно имеющих в своих названиях «real time», требующих реакцию в считанные миллисекунды (обычно порядка сотни мс). Это могут быть разные разновидности роботизированных торговых агентов, причем не обязательно для понятных «человеческих» бирж, где торгуют валютами и ценными бумагами, а например, real time bidding (RTB) для рекламных систем. В тот момент, когда пользователь открывает какую-нибудь страницу в интернете, пока еще идет загрузка, во множестве рекламных систем RTB-систем происходит соревновательная торговля ботов-агентов, за то, какую рекламу показать этому пользователю. Ну и разумеется, быстрая реакция будет нужна и для растущего рынка IoT устройств, особенно медицинских.

Тут как раз очень востребовано упомянутое в «введении» вычислительное кеширование. Как бы сделать так, чтобы запускаемые контейнеры уже оживали в рабочем состоянии, а не мучительно читали конфигурационные файлы и собирали в памяти нужные библиотеки и кеши?

Хорошие новости в том, что для этой задачи сейчас удается применить интересный проект CRIU – *Checkpoint/Restore In Userspace*. Рожденный русскими разработчиками Virtuozzo в 2014 (см. [4]), сейчас он стал уже стабильным и надежным международным продуктом с открытыми исходными кодами, которыми пользуются в своих решениях такие гиганты индустрии, как IBM/RedHat и Google. Изначально этот проект был создан для решения проблемы «живой» миграции контейнера, без остановки, с одного физического узла на другой. Подобное решение для виртуальных машин уже было, и несмотря на сложность живой миграции виртуальных машин, стоит признать, что оно алгоритмически и системно проще – работает на уровне гипервизора, перемещает постепенно блоки диска и оперативной памяти, пока процесс не сойдется и виртуальная машина не оживет на другом сервере. В случае с контейнерами, надо помнить что контейнер – просто группа процессов и ресурсов (сокеты, блокировки и т.п.) внутри одной ОС, их очень непросто даже собрать вместе с клубком их зависимостей (там все сложно, см., например, свежие теоретические модели и алгоритмы [5]), не говоря уже о перемещении на другой сервер, и выполнять большую часть работы надо не в гипервизоре, в пользовательском пространстве, наравне с перемещаемыми процессами.

CRIU безусловно является «швейцарским ножом» для целого ряда разнородных задач (есть даже использование для «паузы в играх»), но именно возможность «заморозить» и «восстановить» группу процессов, оказалась той самой «серебряной пулей» для задачи быстрого «холодного старта», которую и стали пытаться использовать практически только год назад, добиваясь отличных результатов.

Так, в докладе [6] приведены приведенные в табл. 2 цифры обычного времени старта против восстановления через CRUI для тестовых приложений на разных современных фреймворках – видно, что стабильно получается выигрыш практически на порядок, сводя задержку старта к уровню сетевой между географически удаленными датацентрами.

Табл. 1: Выигрыш в скорости запуска контейнера при использовании CRIU
Table 2: Gain in container launch speed when using CRIU

Время до старта (мс)	python 3	nodejs 6	java 8
Обычный старт	919	743	566
Восстановление	78	91	89

Более того, аналогичный подход можно применять не только для того чтобы запускать миниатюрные микросервисы, но и чтобы реплицировать готовые базы или кэширующие сервисы (т.к. решать проблему «прогрева кэшей» для веб-сервисов).

Также в [6] (кстати, это исследование от IBM и Red Hat) приводится эксперимент с наполнением кэширующих NoSQL баз данных 100K короткими записями в сравнении с миграцией уже подготовленного контейнера (табл. 3).

Табл. 2: Выигрыш в скорости репликации хранилища при использовании CRIU
Tab. 3: Gain in storage replication speed when using CRIU

	memcached	redis
Наполнение	1.238	6.254
Миграция	0.806	1.671

Прямо сейчас еще нет проверенных, стабильных решений, реализующих этот подход, есть попытки использовать его в стартапе SwiftyCloud[1] и в проекте gVisor[2], но эта технология уже в плане развития Kubernetes, можно ожидать, что появится в ближайшем году.

Отдельный вопрос при запуске контейнеров системой оркестрации на распределенном ЦОД – обеспечение быстрой доступности самих образов на физических нодах. При быстром интерконнекте (10GB Ethernet, или Infiniband, с низкими задержками) эта проблема может быть решена использованием кластерной файловой системой, CEPH/GlusterFS/ZFS, или использованием специализированных сетевых хранилищ. Но если быстрый интерконнект отсутствует, или невозможен, например, в случае географически распределенных ЦОД, возникает задача предварительной планировки размещения образов контейнеров на всех физических узлах. Да, можно заранее реплицировать реестр всех контейнеров на все узлы, но это может привести и к существенному расходу дискового пространства и сетевой ширины канала, поэтому сейчас есть исследования, как размещать на узлах только часть докер-образов, основываясь на статистике использования или различных алгоритмах, таких как многомерная упаковка контейнеров [7].

[1] https://github.com/swiftycloud
[2] https://gvisor.dev

4. Кэширование, контейнеры и безопасность

Наверное самой большой иллюстрацией инженерного конфликта между «эффективностью» и «безопасностью», а именно, между «кешированием» и «изоляцией», явилась открытая в 2018 году серия уязвимостей, известная теперь под «брендами» MELTDOWN и SPECTRE. Смешные картинки-логотипы (рис. 8), должны всем донести разницу – если «Meltdown» – «утечка» памяти ядра в юзерспейс, вещь неприятная, но которую уже придумали, как починить, то «Spectre» – страшное привидение, проникающее через любые закрытые двери («изолирующие песочницы», проверки в совершенно корректных программах).

Рис. 8: Логотипы уязвимостей Meltdown и Spectre
Fig. 8: Vulnerability logos for Meltdown and Specter

Для тех, кто не в курсе, напомним основные идеи этих уязвимостей. Гонка последних десятилетий за производительностью процессоров, чтобы соответствовать маркетинговым ожиданиям так называемого «закона Мура», привела не только к росту параллелизма и ядер, но и хитрым техникам реализации микроархитектуры процессоров, основанных на аппаратных кешах и спекулятивных вычислениях. Вспомните таблицу иерархии задержек доступа к памяти из введения – доступ к оперативной памяти по сравнению с кешами процессора в сотни раз дольше, и при любой операции, где операнд не попадает в кеш, возникает «голодание по данным» – процессор бессмысленно простаивает в ожидании данных, и часто в операциях ветвления в зависимости от значения в памяти, выгодно, не дожидаясь выяснения результата в условии, выполнить «упреждающие вычисления» (*instruction speculation, speculative execution*) наиболее вероятной ветки, параллельно дождавшись «приехавшего условия», ну и если «не повезло», то отбросить вычисленные результаты, восстановив регистры (до оперативной памяти результаты все равно не успеют доехать, это еще дольше, чем читать RAM). Кроме ветвлений так ускоряют и некоторые циклы, и в целом, это, кроме многоядерности, и было основным методом роста производительности процессоров основных производителей популярных архитектур (Intel, AMD, ARM, Power), с тех пор, как из-за инженерных ограничений практически остановилась миниатюризация и рост частоты. К сожалению, это открыло возможность к еще одной «атаке по побочным каналам» (*side-channel attack*), используя в качестве такого канала время чтения данных.

Если кратко, то заставив процессор упреждающе выполнить операцию с косвенной адресацией по атакуемому участку памяти, разумеется, мы не получаем ни значение этой ячейки напрямую, ни то, куда она указывает – это даже не нужно атакующему, и процессор, выполнив упреждающе эту операцию, поймет, что сделал это зря, спокойно сбросит регистры, но значение атакуемой ячейки осядет в одном из кэшей процессора, в подсистеме TLB, Translation Lookaside Buffers, который не сбрасывается вообще никогда. Затем, можно устроить перебор выборок значений из собственной памяти, с замером времени выполнения, и статистически достоверно «поймать» тот момент, когда выборка будет быстрее – и тем самым понять, какое значение осталось в кеше TLB. В целом, это комбинация достаточно известных подходов «исследование состояния

аппаратуры методом временного прозвона» и «получение данных, осевших в кэше» примененный конкретно к архитектуре процессоров. Атаки Spectre (их две, первого и второго рода) еще более хитрые, там происходит обход проверки границ (*bounds check bypass*) и манипуляция целевым кешем адресов ветвлений (*branch target injection*). За подробными деталями этих атак отошлем к [8, 9], а также десяткам популярных статей и презентаций, которых легко найти по «брендам этих уязвимостей», заметим, что многим специалистам по ИБ эти уязвимости были известна, и скорее всего, они активно эксплуатировалась и раньше.

Эти уязвимости безусловно были опасны во многих сценариях и вовсе не связанных с виртуализацией и контейнерами, везде, куда каким-нибудь образом, можно было доставить атакующий код. Так, например, Javascript мог атаковать память браузера и вытащить логины-пароли. Но если с Javascript в браузере удалось быстро разобраться – загрубление таймеров доступных для JS-кода, модификация JIT компилятора с использованием хитрых программных конструкций *retpoline*, сохраняющих большую часть выгод упреждающих вычислений, но сбивающих выполнение этих атак, то с областью виртуализации, особенно со стороны облачных провайдеров, предоставляющих публичные услуги, или крупных частных корпоративных центров данных с виртуальными контурами доступа, и автоматической миграцией виртуальных контейнеров между физическими серверами, все хуже, причем именно для контейнерной виртуализации. С помощью Meltdown и Spectre пока не удалось преодолеть барьер между гостевой системой и гипервизором при физической виртуализации, а вот попасть из контейнера в адресное пространство ядра – да, удалось.

Да, производители оборудования, операционных систем, и систем виртуализации уже выпустили множество патчей на всех уровнях:

1. обновление микрокода для процессоров (особенно Intel и AMD);
2. обновление ядра ОС Linux и Windows – патчи KPTI (*Kernel Page Table Isolation,* изоляция таблицы страниц ядра), KAISER (*Kernel Address Isolation to have Side-channels Efficiently Removed* – изоляция пространства адресов ядра во избежание атак по стороннему каналу);
3. обновление компиляторов, использующихся для сборки критических бинарников – ядра ОС, гипервизоров, для реализации в них конструкций *retpoline* (новорожденное слово из «return» и «trampoline») [10]

Но, во-первых, каждое из таких исправлений может уронить производительность на 10-20%, что сильно снижает эффективность контейнерной виртуализации по сравнению с аппаратной. Во-вторых, специалисты признают, что все это полумеры, рассчитанные на затруднение эксплуатации уже реализованных по этим уязвимостям эксплоитов, и надежно избавится от этих проблем можно будет только сменив оборудование, или хотя бы процессоры на самые современные. Основные выводы тут такие – сейчас, до смены поколения серверов, лучше избегать контейнерной виртуализации при предоставлении публичных услуг хостинга, когда недоверенный атакующий код злоумышленника может внезапно оказаться среди контейнеров других клиентов, если этого нельзя избежать – то обязательно поставить все необходимые патчи на всех уровнях, проверив нагрузочными тестами, что это не вызовет критического падения производительности в тех задачах, для которых применяются контейнеры.

5. Кеширование при сборке образов контейнеров

В отличие от контейнеров LXC и OpenVZ, контейнеры Docker обрели огромную популярность среди разработчиков, именно благодаря эффективной реализации «шаблонов», «образов» контейнеров, которые, благодаря «слоистой структуре» overlayfs

и иерархическим ссылкам обеспечивают дедупликацию данных, и экономию диска и памяти. С другой стороны – инфраструктура так называемых докер-реестров, публичных и приватных, решает проблему эффективного совместного использования этих образов и доставку и выкатку их как на тестовые среды, так и на «боевые» системы. В результате докер-образы стали практически самым универсальным и популярным методом упаковки приложения, вытесняя линукс-пакеты, которых надо было раньше мучительно собирать под каждый целевой дистрибутив, и разные другие инсталляционные форматы – ведь теперь, приложение в виде docker-контейнера можно запустить не только на Linux, но даже и на Windows-системах. В результате расцвели практики непрерывной интеграции, с сборкой и тестированием приложения после каждого минимального изменения.

Но тут же возникли проблемы максимального ускорения этого процесса – если раньше разработчики выпадали из «потока разработки», оправдываясь, что оно «компилируется», то сейчас, простую «компиляцию» заменила сборка новых докер-образов (установка зависимостей, размещение артефактов, быстрое автоматическое тестирование). Даже если простая сборка образа в чистом окружении системы сборки будет занимать, скажем, минут 10, то при росте команды и числа коммитов в день ожидание готового результата будет расти нелинейно, даже в пуассоновской модели теории массового обслуживания [11] (рис. 9).

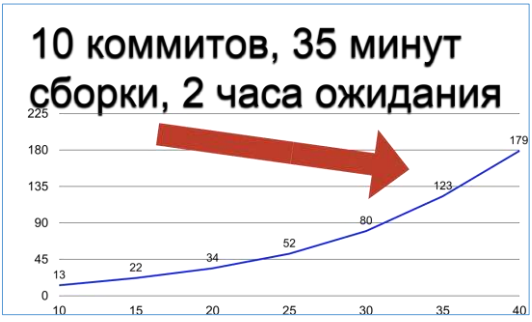

Рис.9: Нелинейный рост времени ожидания сборки при росте числа коммитов
Fig. 9: Non-linear increase in build waiting time with an increase in the number of commits

Решения есть – нужно обеспечить максимальную атомарность всех изменений в докер-образах, чтобы каждое изменение изменяло только один, небольшой «слой» докер-образа, ссылаясь на уже неизменные существующие в докер-реестре образы. Это можно сделать на «человеческом уровне», обязав разработчиков и докер инженеров, максимально атомизировать собираемые докер-образы при сборке, путем специальных соглашений в формировании Dockerfile (см. [11]). Но как любое решение, зависящее от людей («стандарты кодирования», «архитектурные политики» и т.п.), это трудно соблюдать, и в индустрии есть желание это автоматизировать.

Так, появляются проекты, такие как Kaniko Cache[3], которые автоматически кэшируют в докер-реестре все промежуточные слои, вне зависимости от того, как разработчики оформляли докер-файлы. Да, это приводит к дополнительным затратам дискового пространства, но существенно ускоряет скорость сборки.

[3] https://cloud.google.com/cloud-build/docs/kaniko-cache

6. Исследования оптимального размещения мультиконтейнерных конфигураций на физических серверах

Кроме рассмотренных ранее сценариев использования, где требовалось экономить время запуска или сборки, часто возникают и альтернативные требования – когда скорость запуска не важна, а требуется максимальная энергоэффективность оборудования. Опуская возможности инженерной оптимизации датацентров – эффективные технологии охлаждения и электропитания, на программном уровне, для максимальной энергоэффективности требуется обеспечить максимальную загрузку имеющихся физических серверов, чтобы исключить использование в холостую. Еще недавно [12] ситуация была совершенно неудовлетворительной; условно говоря, средний сервер был загружен всего лишь на 10%, при этом его совокупное удельное энергопотребление падало не больше чем в два раза, от такового при максимальной нагрузке.

Сейчас, с ростом популярности виртуализации и глобализации вычислительных ресурсов в ЦОД крупных провайдеров, таких как Google и Amazon, ситуация значительно улучшилась. Впрочем, крупные провайдеры обычно не ставят задачу 100% утилизации каждого сервера, оставляя резерв для низколатентного маневрирования мощностью под внезапные заказы дополнительной мощности от клиентов, но эта задача актуальна для небольших и средних специализированных ЦОД, где задержки адаптации к меняющейся нагрузке не так важны, а важна именно максимальная загрузка. Как пример, можно привести различные системы обработки пакетных заданий – научных расчетов, сборок сложного ПО (линукс-дистрибутивов), рендер-фермы и т.п. Для таких задач еще нет реализованного в ПО виртуализации оптимальных решений, но идут исследования ученых, как из университетов, так и крупных компаний (IBM, Redhat, VMWare), предлагающих различные алгоритмы эффективного планирования.

6.1 Консолидация виртуальных машин на основе решения декомпозированной многоцелевой задачи об упаковке

В статье [13] специалистов IBM рассмотрена задача размещения виртуальных машин (ВМ) на физических серверах, известная как *Virtual Machine Consolidation* или *Virtual Machine Packing Problem*. Решение использует тот факт, что обычно провайдеры виртуальных машин предоставляют виртуальные машины с конечным множеством наборов параметров. Задача сведена к 2-шаговому варианту многомерной задачи упаковки в контейнеры *multi-dimensional bin packing problem*. На первом шаге виртуальные машины объединяются в кластеры методом k-средних. ВМ из одного кластера изменяются так, что все становятся одинаковыми. Для полученных ВМ проводится метод динамического программирования. На втором шаге из полученных наборов виртуальных машин строится их расположение на физических серверах. Существенно уменьшено время работы алгоритма по сравнению с ранее известными алгоритмами, такими как одномерный First Fit и многомерный Bin Packing.

Метод First Fit для одномерной задачи максимально прост: «Очередной объект упаковывается в последний созданный контейнер, в который он помещается. Если объект не помещается ни в один контейнер, то для его упаковки создаётся новый, куда он и размещается». Зато он может работать в режиме онлайн, когда объекты на вход поступают последовательно в некотором неизвестном порядке и при упаковке очередного объекта не известны размеры следующих за ним по порядку объектов. В этом случае алгоритм гарантирует, что асимптотически отношение суммы размеров созданных контейнеров к сумме размеров объектов будет не более, чем *17/10*, как показано в [14]. Однако оффлайновый алгоритм First Fit, на вход которому подаются объекты с размерами, упорядоченными по убыванию, как показано в [15], гарантирует

уменьшение асимптотики данной величины до *11/9*. Данный алгоритм известен как *First Fit Decreasing* и в [13] применяется его обобщение на многомерный случай – «Иерархический многомерный алгоритм Bin Packing для виртуальных машин».

Задача размещения виртуальных машин сводится к многомерной задаче упаковке в контейнеры, где измерения соответствуют ресурсам различных типов, которые необходимы для создания ВМ, а именно: CPU, RAM, IOPS.

Задача ставится следующим образом: даны параметры серверов: $C = \{c_1, c_2, c_3, \dots, c_n\}$ такие, что $\forall c_i \in C, c_i \geq 0$ для каждого сервера C. Каждый параметр соответствует количеству ресурса определённого типа на физическом сервере. Также известны параметры ВМ для каждой машины V: $V = \{v_1, v_2, v_3, \dots, v_n\}$.

Требуется расположить ВМ на физических серверах так, чтобы каждая ВМ находилась на сервере и сумма каждого из параметров по ВМ, находящихся на сервере, не превосходила значение этого параметра по серверу. При размещении требуется минимизировать число используемых физических серверов.

Задача в данной постановке сводится к многомерной задаче упаковки объектов в контейнеры *Multi-dimensional Bin Packing*, где V, или ВМ – упаковываемые объекты, а $C,$ или серверы – контейнеры. Поэтому задача далее рассматривается в терминах многомерной задачи упаковки в контейнеры.

Рассмотрим построение шаблонов для оптимального решения.

Фиксируется натуральное число K. Применяется метод k-средних кластеризации к набору объектов или параметров ВМ, в результате применения которого ВМ разобьются на K кластеров, и в каждом кластере будут ВМ с близкими параметрами.

Объекты из каждого кластера преобразуются следующим образом: пусть в кластере Cl находятся объекты $V_1, \dots, V_{last}$ с параметрами $v_1^1, \dots, v_n^1, \dots, v_1^{last}, \dots, v_n^{last}$. Тогда после преобразования, параметр с номером i для каждой новой ВМ будет равен $v_i^{k\ new} = v_i^{new} = \max_{1 \leq j \leq last} v_i^j$, и все объекты из одного кластера будут одинаковыми.

Далее для этих изменённых объектов методом динамического программирования строятся все возможные наборы объектов, такие, что число объектов из каждого кластера в наборе не превышает общего числа объектов, принадлежащего данному кластеру и такие, что для каждого параметра сумма данных параметров в наборе не превосходит количества параметра данного типа в самом вместительном по параметру данного типа контейнере.

Считая любые 2 объекта из одного кластера идентичными, данные наборы можно построить за $O(n^k)$ операций, где K – число кластеров, а n – число ВМ, которые нужно упаковать.

Аллокация ВМ производится алгоритмом, похожим на First Fit Decreasing. Сначала наборы ВМ упорядочиваются с помощью метрики из оценки МакКарти (short-circuit evaluation), а именно, если у наборов Set_1 и Set_2 сумма параметров по всем ВМ равна $\{c_1^{set1}, \dots, c_n^{set1}\}$ и $\{c_1^{set2}, \dots, c_n^{set2}\}$ соответственно, то $Set_1 > Set_2$ тогда и только тогда, когда $\exists i \in \{1, \dots, n\}$: если $j < i$, то $c_j^{set1} = c_j^{set1}$, а $c_i^{set1} > c_i^{set1}$. При оценке по данной метрике сначала сравниваются наборы по сумме первых параметров всех элементов. Если для одного из наборов данная величина больше, чем для другого набора, то данный набор больше другого. Если же суммы первых параметров равны, рассматриваются вторые параметры и т.д.

Алгоритм аллокации ВМ работает следующим образом.

Выбирается самый большой по метрике МакКарти набор ВМ из оставшихся наборов и сервер с самой большой оценкой оставшихся параметров. Если набор ВМ можно расположить на сервере так, что сумма каждого параметра не превышается, а ВМ из

каждого кластера осталось достаточно, чтобы набрать набор, то выбираются ВМ наибольшего размера из оставшихся из каждого кластера и на сервере размещаются данные ВМ. Если этого нельзя сделать, то набор удаляется из рассмотрения.

6.2 Консолидация виртуальных машин с учетом производительности

В работе [16] был предложен алгоритм «PACMan» для задачи размещения виртуальных машин. В отличии от [13], алгоритм строился с учётом того, что в результате уплотнения ВМ происходит размещение нескольких ВМ на одном сервере. Как следствие, происходит конфликт за использование разделяемых ресурсов, таких как CPU или RAM, в результате чего происходит ухудшение качества и скорости работы ВМ. Алгоритм позволяет сохранить ухудшение качества в заранее указанных пределах, при этом добиваясь производительности, близкой к лучшей возможной.

Предполагается возможной оценка числа раз, в которое ухудшается работа каждой ВМ относительно изолированного размещения на отдельном сервере, при размещении набора ВМ на одном сервере для каждой ВМ каждого набора, исполняемого на том сервере.

Для размещаемого набора S ВМ вводится мера удельной стоимости совместного размещения

$$V(S) = \frac{w(S)}{|S|},$$

где за $w(S)$ обозначено использование ресурсов каждой ВМ из набора. Качество работы алгоритма будем оценивать суммой затрат $E(ALG)$, равной сумме стоимостей всех размещенных наборов ВМ:

$$E(ALG) = \sum_{S \in phisical\ server} |S|\ V(S).$$

Все наборы ВМ с допустимым ухудшением работы каждой размещаются в порядке неубывания $V(S)$. Обозначим за $\mathfrak{F}$ множество всех этих наборов.

Алгоритм размещения ВМ работает следующим образом:

1. выбираем набор S из $\mathfrak{F}$ с наименьшей стоимостью $V(S)$;
2. если все ВМ из него можно разместить на сервере, то делаем это;
3. если это сделать нельзя, удаляем набор S из множества наборов.

Пусть на сервере нельзя разместить больше k ВМ. Была доказана следующая:

Теорема. Для всех входных данных, алгоритм имеет сумму затрат $E(ALG) = O(\ln k \cdot E(OPT))$,

где $E(OPT))$ – сумма затрат оптимального алгоритма размещения ВМ. Однако так как максимальное число k ВМ, которые можно разместить на сервере может быть велико, то и расчёт ухудшения для каждого набора ВМ занимает $O(k \cdot n^k \cdot \ln n)$ операций, где n – число размещаемых ВМ и может быть вычислительно сложным. Однако, если, аналогично работе Dow, разбить множества ВМ на l кластеров методом k-средних, то, возможно, за счет уменьшения $E(ALG)$, вычислительная сложность алгоритма может быть уменьшена до $O(l \cdot n^l \cdot \ln n)$ для любого $l \in \mathbb{N}$.

6.3 Оптимальные онлайновые детерминированные алгоритмы и эвристики для динамической консолидации виртуальных машин

В работе [17] был предложен динамический алгоритм уплотнения расположения виртуальных машин в дата-центрах. Задача размещения ВМ представлена в виде задачи

bin packing упаковки в контейнеры с разными значениями размеров и стоимости контейнеров. Размеры контейнеров здесь – число доступных ЦП узлов дата-центра, а стоимости соответствуют потреблению энергии узла.

Был предложен алгоритм, являющийся обобщением эвристики *Best Fit Decreasing Height* для задачи *bin packing*. Данная эвристика упаковывает объекты, расположенные в порядке убывания их веса в тот контейнер, при упаковке в который остаётся наименьшее число незаполненного пространства. Как показано в [18], данный алгоритм использует не более, чем 11/9·OPT+1 контейнеров, где OPT – это число решений, используемых оптимальным решением задачи.

Алгоритм *Power Aware Best Fit Decreasing* для размещения ВМ работает следующим образом: Сначала все ВМ размещаются по убыванию числа используемых ЦП, затем очередная ВМ размещается в хост, на котором из-за добавления ВМ происходит наименьшее среди всех хостов увеличение энергопотребления.

6.4 Консолидация контейнеров в облачных центрах данных с обеспечением эффективного энергопотребления

В работе [19] предложен фреймворк, уплотняющий расположение контейнеров на виртуальных машинах и сокращающий потребление энергии на хосте.

В последнее время наряду с такими облачным сервисами, как IaaS (инфрастуктура как сервис), PaaS (платформа как сервис) и SaaS (программное обеспечение как сервис), всё шире и шире входит в употребление новый вид сервиса – CaaS, или контейнеры как сервис. Благодаря ему получается уменьшить время запуска системы.

Слой CaaS находится между слоем IaaS, предоставляющим виртуальные вычислительные ресурсы и слоем PaaS, предоставляет приложениям среду выполнения и соединяет эти два слоя вместе.

Наиболее широко используемым инструментом контейнеризации на сегодняшний день является ПО Docker. Однако в [20] показано, что запуск контейнеров Docker-а на ВМ позволяет достичь близкой, а, в некоторых случаях, даже лучшей производительности, чем запуск в “родной” среде, благодаря изолированности ВМ. Подход разворачивания контейнеров внутри ВМ был рассмотрен в работе.

Серверы, элементы сети и системы охлаждения являются основными потребителями энергии в современном дата-центре. В работе с помощью алгоритмов миграции контейнеров уменьшается количество работающих серверов, благодаря чему уменьшается суммарное по серверам потребление энергии.

Однако, уменьшая потребление энергии, следует удовлетворять SLA-соглашениям (Service Level Agreement). В случае развёртывания контейнеров на ВМ, SLA-соглашения нарушаются тогда и только тогда, когда ВМ, на которой запущен контейнер не имеет ЦП, необходимых ему для работы. Таким образом, за SLA сумму по всем ВМ, не получивших требующегося числа ядер, отношений разности чисел требуемых и полученных чисел ЦП к требуемому числу ЦП, или

$$SLA = \sum_{i=1}^{N_s} \sum_{j=1}^{N_{vm}} \sum_{p=1}^{N_v} \frac{CPU_r(vm_{j,i}, t_p) - CPU_\alpha(vm_{j,i}, t_p)}{CPU_r(vm_{j,i}, t_p)}.$$

Здесь N_s – число серверов, N_{vm} – число ВМ, N_v – число нарушений SLA-соглашений, $vm_{j,i}$ – ВМ j на сервере i , t_p – время, в которое произошло нарушение SLA-соглашений с номером p, $CPU_r(vm_{j,i}, t_p)$ – число ЦП, требующихся ВМ j для развертывания контейнеров, расположенной на сервере i в момент времени t_p, и $CPU_\alpha(vm_{j,i}, t_p)$ – число ЦП, выделенных ВМ j в момент времени t_p.

Алгоритмы, отвечающие за миграцию контейнеров, работают в три этапа:

1. определение ситуаций, при которых контейнеры должны мигрировать;
2. определение, какие контейнеры мигрируют;
3. определение, куда контейнеры мигрируют.

Данные этапы в фреймворке реализованы следующим образом.

1. Для каждого хоста статически заданы пороги недогрузки UL (*under-load*) и перегрузки OL (*over-load*), определяемые специальными алгоритмами. Если хост перегружен или недогружен, происходит миграция контейнеров
2. Рассмотрены алгоритмы Mcor, выбирающий контейнеры с наиболее коррелированной нагрузкой с сервером, на котором контейнеры развёрнуты, и алгоритм MU, выбирающий контейнеры, которые используют больше всего ЦП.
3. Были рассмотрен три алгоритма для выбора хоста, работающие по принципам алгоритмов для классической задачи *Bin Packing*: *First Fit Host Select*, выбирающий последний созданный подходящий хост для размещения контейнера; *Random HS*, выбирающий случайный подходящий хост и *Least Full HS*, выбирающий хост с наименьшим числом использованных ЦП из имеющихся.

При увеличении порога UL и уменьшении порога OL, число миграций контейнеров увеличивается, и, следовательно, создаётся большее число ВМ и увеличивается использование ресурсов. С другой стороны, при увеличении OL и уменьшении UL, риск нарушения SLA-соглашений увеличивается. Оптимальные значения OL и UL для алгоритма Mcor выбора мигрирующих контейнеров и алгоритма First Fit HS выбора хоста назначения миграции равны OL=0.8, UL=0.7 соответственно. По сравнению с другими методами выбора мигрирующего контейнера и назначения миграции и значениями OL, UL, достигается экономия ресурсов на 7.4% при средних нарушениях SLA меньше 5%

7. Заключение

В данной работе рассмотрено множество аспектов технологий кэширования в мультиконтейнерных системах облачных вычислений.

Рассмотрены практические технологии, позволяющие улучшить производительность мультиконтейнерных систем за счет различного кэширования — ускорение скорости старта за счет кэширования образов и состояний контейнеров, улучшение экономичности за счет разделения файлов, кэширование слоев образов при сборке и т.п. При этом рассмотрены и проблемы безопасности, которые приносят в контейнерную виртуализацию технологии аппаратного кэширования данных в процессорах.

Отдельно рассмотрены теоретические подходы улучшения производительности мультиконтейнерных систем виртуализации за счет различных алгоритмов оптимального планирования.

Список литературы / References

[1]. Latency numbers every programmer should know. URL https://gist.github.com/hellerbarde/2843375, accessed 15.11.2019.

[2]. Увеличение плотности контейнеров на ноде с помощью технологии PFCACHE. Блог компании Rusonyx / Increase container density on a node using PFCACHE technology. Rusonyx company blog. URL https://habr.com/ru/company/rusonyx/blog/444696/, accessed 15.11.2019.

[3]. Virtuozzo User's Guide. URL https://docs.virtuozzo.com/pdf/virtuozzo_7_users_guide.pdf, accessed 15.12.2019.

[4]. Павел Емельянов. CRIU – как маленький open-source проект меняет жизнь большой компании. Материалы 11-й международная конференция Linux Vacation/Eastern Europe, 2015, http://0x1.tv/20150627H / Pavel Emelyanov. CRIU – how a small open-source project changes the

life of a large company. In Proc. of the 11th International Linux Vacation / Eastern Europe Conference, 2015, http://0x1.tv/20150627H (in Russian).

[5]. Николай Ефанов. Единая теория восстановления деревьев процессов Linux – огибая подводные камни checkpoint-restore. Материалы 15-й Центрально-восточноевропейской конференция в России по разработке программного обеспечения, 2019, http://0x1.tv/20191115BN / Nikolay Efanov. The unified theory of Linux process tree restoration - enveloping the pitfalls of checkpoint-restore. In Proc. of the 15th Central & Eastern European Software Engineering Conference in Russia, 2019, http://0x1.tv/20191115BN (in Russian).

[6]. Mike Rapoport, Adrian Reber. To Kill or to Checkpoint – That is the Question. In Proc. of the Open Source Summit + Embedded Linux Conference & OpenIoT Summit Europe, 2018. https://static.sched.com/hosted_files/osseu18/2a/kill-or-checkpoint.pdf.

[7]. Grushin D.A., Kuzyurin N.N. On Effective Scheduling in Computing Clusters. Programming and Computer Software, vol. 45, issue 7, 2019, pp. 398–404. doi:10.1134/S0361768819070077.

[8]. Moritz Lipp, Michael Schwarz et al. Meltdown: Reading Kernel Memory from User Space. In Proc. of the 27th USENIX Security Symposium, 2018, pp. 973-990.

[9]. Paul Kocher, Jann Horn et al. Spectre Attacks: Exploiting Speculative Execution. In Proc. of the IEEE Symposium on Security and Privacy, 2019, pp. 1-19.

[10]. Paul Turner. Retpoline: a software construct for preventing branch-target-injection. URL https://support.google.com/faqs/answer/7625886, accessed 15.12.2019.

[11]. Николай Пасынков. Слои Docker для ускорения сборки проекта. Материалы 15-й Центрально-восточноевропейской конференция в России по разработке программного обеспечения, 2019, http://0x1.tv/20191114DJ / Nikolay Pasynkov. Docker layers to speed up project builds. In Proc. of the 15th Central & Eastern European Software Engineering Conference in Russia, 2019, http://0x1.tv/20191114DJ (in Russian).

[12]. Report to Congress on Server and Data Center Energy Efficiency: Public Law 109-431. U.S. Department of Energy, Office of Scientific and Technical Information. URL https://www.osti.gov/servlets/purl/929723, accessed 15.12.2019.

[13]. Dow Eli M. Decomposed Multi-Objective Bin-Packing for Virtual Machine Consolidation. PeerJ Computer Science, vol. 2, 2016, article e47.

[14]. Xia Binzhou, Zhiyi Tan. Tighter Bounds of the First Fit Algorithm for the Bin-Packing Problem. Discrete Applied Mathematics, vol. 158, issue 15, 2010, pp. 1668-1675.

[15]. György Dósa. The Tight Bound of First Fit Decreasing Bin-Packing Algorithm Is $FFD(I) \leq 11/9OPT(I) + 6/9$. Combinatorics, Algorithms, Probabilistic and Experimental Methodologies. Lecture Notes in Computer Science, vol. 4614, 2007, pp. 1-11/

[16]. Roytman Alan, Aman Kansal et al. PACMan: Performance aware virtual machine consolidation. In Proc. of the 10th International Conference on Autonomic Computing, 2013, pp. 83-94.

[17]. Beloglazov Anton, and Rajkumar Buyya. Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers. Concurrency and Computation: Practice and Experience, vol. 24, issue 3, 2012, pp. 1393-1550.

[18]. A Simple Proof of the Inequality FFD (L) ≤ 11/9 OPT (L) 1, ∀L for the FFD Bin-Packing Algorithm. Acta Mathematicae Applicatae Sinica, vol. 7, issue 4, 1991, pp, 321–331.

[19]. Piraghaj, Sareh Fotuhi, Amir Vahid Dastjerdi, Rodrigo N. Calheiros, and Rajkumar Buyya. A Framework and Algorithm for Energy Efficient Container Consolidation in Cloud Data Centers. In Proc. of the IEEE International Conference on Data Science and Data Intensive Systems, 2015, pp. 368 - 375.

[20]. Ali Qasim. Scaling web 2.0 applications using docker containers on vsphere 6.0. VMware VROOM! Blog. URL https://blogs.vmware.com/performance/2015/04/scaling-web-2-0-applications-using-docker-containers-vsphere-6-0.html, accessed 15.11.2019.

Информация об авторах / Information about authors

Денис Олегович ЛАЗАРЕВ – стажер-исследователь ИСП РАН, аспирант МФТИ. Научные интересы: задачи упаковки в полосы и контейнеры, случайные графы и дискретная оптимизация.

Denis Olegovich LAZAREV – a research trainee at ISP RAS and a postgraduate student of MIPT. Research interests: bin and strip packing problems, random graphs and discrete optimisation.

Дмитрий Андреевич ГРУШИН является научным сотрудником ИСП РАН. Его научные интересы включают оптимизацию размещения задач в распределенных вычислительных системах, виртуализацию, контейнеризацию, облачные технологии.

Dmitry Andreevich GRUSHIN is a researcher at ISP RAS. His research interests include scheduling optimization in distributed computing systems, virtualization and containerization.

Станислав Александрович ФОМИН – программист ИСП РАН, преподаватель МФТИ. Научные интересы: теория сложности, дискретная оптимизация.

Stanislav Aleksandrovich FOMIN – ISPRAS programmer, lecturer at MIPT. Research interests: theory of complexity and discrete optimisation.

DOI: 10.15514/ISPRAS-2019-31(6)-8

О возможности стойкой обфускации программ в одной модели облачных вычислений

[1,2] *А.В. Шокуров, ORCID: 0000-0002-6801-7728 <shok@ispras.ru>*
[3] *И.В. Абрамова, ORCID: 0000-0002-8421-4617<abramovairinacmcmsu@gmail.com>*
[1,2,3] *Н.П. Варновский, ORCID: 0000-0002-2363-0254 <barnaba.np@gmail.com>*
[1,2,3,4] *В.А. Захаров, ORCID: 0000-0002-3794-9565 <zakh@cs.msu.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, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
[3] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1
[4] *НИУ Высшая школа экономики,*
101978, Россия, г. Москва, ул. Мясницкая, д. 20

Аннотация. В данной статье проведено исследование возможности применения одной модели облачных вычислений, использующей криптосерверы, для обфускации программ. Ранее эта модель облачных вычислений была предложена нами в связи с изучением задачи обеспечения информационной безопасности мультиклиентских распределенных вычислений над зашифрованными данными. На основе этой модели нами предложен новый подход, предусматривающий использование пороговых гомоморфных криптосистем для обфускации программ. Основным результатом статьи являются новое определение стойкости обфускации программ в модели облачных вычислений и теорема, доказывающая криптографическую стойкость предложенного алгоритма обфускации программ в предположении существования криптографически стойких пороговых гомоморфных систем шифрования.

Ключевые слова: обфускация программ; гомоморфное шифрование; стойкость; облачные вычисления

Для цитирования: Шокуров А.В., Абрамова И.В., Варновский Н.П., Захаров В.А. О возможности стойкой обфускации программ в одной модели облачных вычислений. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 145–162. DOI: 10.15514/ISPRAS–2019–31(6)–8

Благодарности: Исследования, результаты которых представлены в этой статье, выполнены при поддержке Российского Фонда Фундаментальных Исследований (проект 19-01-00702).

On the possibility of secure program obfuscation in some model of cloud computing

[1,2] *A.V. Shokurov, ORCID: 0000-0002-6801-7728 <shok@ispras.ru>*
[3] *I.V. Abramova* ORCID: *0000-0002-8421-4617 <abramovairinacmcmsu@gmail.com>*
[1,2,3] *N.P. Varnovsky, ORCID: 0000-0002-2363-0254 <barnaba.np@gmail.com>*
[1,2,3,4] *V.A. Zakharov, ORCID: 0000-0002-3794-9565 <zakh@cs.msu.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, 141700, Russia*
[3] *Lomonosov Moscow State University, GSP-1, Leninskie Gory, Moscow, 119991, Russia*
[4] *National Research University, Higher School of Economics 20, Myasnitskaya Ulitsa, Moscow, 101978, Russia*

Abstract. In this paper, we study the possibility of using a certain cloud computing model supplied with cryptoservers to obfuscate software programs. Earlier, we proposed this cloud computing model in our study of some information security problems for multi-client distributed computing over encrypted data. Based on this model, we proposed a new approach involving the use of threshold homomorphic cryptosystems for program obfuscation. The main result of this paper is a new definition of the resistance of obfuscation of programs in the cloud computing model and a theorem proving the cryptographic strength of the proposed algorithm of obfuscation of programs under the assumption of the existence of cryptographically strong threshold homomorphic encryption systems. The paper is organized as follows. In the introducing section we discuss the main aspects of the information security problems for cloud computing systems. Section 2 provides a description of the obfuscation program objectives, as well as a brief overview of the main achievements in its study. The next section provides general information about homomorphic cryptosystems. Section 4 describes a more special class of homomorphic cryptosystems - threshold homomorphic encryption systems. Section 5 introduces the cloud computing model, which is used as framework for our program obfuscation techniques. For this computing environment, in Section 6, the definition of the cryptographic strength of program obfuscation is formulated, a new method of program obfuscation using threshold homomorphic cryptosystems is described, and the cryptographic strength of the proposed obfuscation algorithm is proved.

Keywords: program obfuscation; homomorphic encryption; security; cloud computing

For citation: Shokurov A.V., Abramova I.V., Varnovsky N.P., Zakharov V.A. On the possibility of secure program obfuscation in some model of cloud computing. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 145-162 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-8

Acknowledgements. The studies, the results of which are presented in this article, were supported by the Russian Foundation for Basic Research (project 19-01-00702).

1. Введение

Современные прикладные информационные системы должны удовлетворять большому числу разнообразных требований, предъявляемых к их корректности, отказоустойчивости, совместимости с системным обеспечением, адаптируемости к вычислительному оборудованию и др. С появлением и развитием концепции облачных вычислений к числу этих требований было добавлено также и требование информационной безопасности; до недавнего времени оно предъявлялось лишь к телекоммуникационным протоколам и сравнительно узкому классу специализированных программ, реализующих криптографические алгоритмы и протоколы. Средства криптографии позволяли хорошо защищать данные при их хранении на общедоступных накопителях и передаче по открытым каналам связи. Но при вычислении (преобразовании) данных прикладными программами защите подлежали не сами

обрабатываемые данные или программы, работающие с этими данными, а вычислительные устройства (компьютеры, серверы, процессоры и пр.), на которых выполняются эти программы. Как правило, такое вычислительное устройство находилось в полном распоряжении пользователя, и он имел возможность применять для его защиты административные меры и физические средства.

Но при использовании систем облачных вычислений эти способы информационной защиты уже непригодны: в облачных системах вычислений данные (а также и программы) находятся в общедоступной среде, подобно тому, как это происходит при их хранении и передаче. Но тогда можно предполагать, что для их информационной защиты также можно использовать криптографические средства, например, шифрование. Главная трудность здесь состоит в том, что при хранении и передаче данные не подвергаются существенному изменению; они могут изменять форму их представления, но их семантическое содержание остается неизменным. Поэтому при хранении и передаче данные можно представлять в зашифрованном виде. Напротив, при проведении вычислений семантическое содержание данных очень важно, и поэтому для их защиты могут быть пригодны лишь такие системы шифрования, в которых эти изменения можно адекватно представить при помощи подходящих операций над шифрами данных.

Системы шифрования, обладающие указанным свойством, называются гомоморфными криптосистемами (гомоморфизм в алгебре означает свойство согласованности операций в двух различных алгебраических системах – в нашем случае, в пространстве данных и в пространстве их шифртекстов). Построение гомоморфных систем шифрования позволило бы значительно расширить область применения криптографических средств защиты информации. Поэтому задача построения гомоморфных криптосистем остается одной из центральных проблем современной криптографии.

Однако построение гомоморфной системы шифрование является лишь необходимым условием обеспечения надежной информационной защиты данных в процессе вычисления. Как известно, стойкость криптографических протоколов зависит не только от стойкости используемых в них криптографических примитивов (т.е. базовых функций, наподобие процедуры генерации ключей, шифрования, цифровой подписи и пр.), но также и от особенностей устройства самих протоколов и задач, для решения которых они предназначены. В связи с этим не меньшее значение имеют также и вопросы о том, для каких вычислительных схем и при каких условиях стойкость гомоморфного шифрования является достаточным условием стойкости информационной защиты этих схем.

В данной статье проведено исследование возможности применения одной модели облачных вычислений, использующей криптосерверы, для обфускации программ. Ранее эта модель облачных вычислений была предложена нами в связи с изучением задачи обеспечения информационной безопасности мультиклиентских распределенных вычислений над зашифрованными данными. На основе этой модели нами предложен новый подход, предусматривающий использование пороговых гомоморфных криптосистем для обфускации программ. Основным результатом статьи являются новое определение стойкости обфускации программ в модели облачных вычислений и теорема, доказывающая криптографическую стойкость предложенного алгоритма обфускации программ в предположении существования криптографически стойких пороговых гомоморфных систем шифрования. Подобный подход к обфускации программ восходит к статьям [1-3], в которых было показано, что информационная безопасность вычислений может быть гарантирована в том случае, если один из компонентов вычислительной системы (например, память компьютера) оказывается недоступным для противника.

Статья устроена следующим образом. В разд. 2 приводится описание задачи обфускации программ, а также краткий обзор основных достижений в ее исследовании. В следующем разделе приведены общие сведения о гомоморфных криптосистемах. В разд. 4 рассказано о более специальном классе гомоморфных криптосистем – пороговых гомоморфных системах шифрования. В разд. 5 приведено описание модели облачных вычислений, в которой решается задача обфускации программ. Для этой вычислительной среды в разд. 6 сформулировано определение криптографической стойкости обфускации программ, описан новый метод обфускации программ с использованием пороговых гомоморфных криптосистем, и доказана криптографическая стойкость предложенного алгоритма обфускации.

2. Задача обфускации программ

Обфускацией программы называется любое ее преобразование, которое сохраняет функциональность программы, но при этом приводит программу в такую форму, из которой очень трудно извлечь полезную информацию об алгоритмах и структурах данных, реализованных в программе. Согласно определению, предложенному в статье [4], обфускатор программ – это такой вероятностный алгоритм O, который, получив на входе программу π, преобразует ее в программу $O(\pi)$, удовлетворяющую следующим трем требованиям.

- функциональность: программы π и $O(\pi)$ вычисляют одну и ту же функцию;
- эффективность: размер и быстродействие программы $O(\pi)$ ухудшаются незначительно по сравнению с программой π;
- стойкость: программа $O(\pi)$ трудна для понимания.

Задача обфускации программ была впервые упомянута в основополагающей работе [5]. В явном виде понятие обфускации программ было введено в статье [6]. Более точное определение требований эффективности и стойкости обфускации зависит от тех приложений, в которых обфускация программ планируется использоваться. С описанием различных применений обфускации программ для решения задач системного программирования и компьютерной безопасности можно ознакомиться в статьях [6-10]. Обфускирующие преобразования могут быть использованы для защиты интеллектуальной собственности в качестве средства, препятствующего восстановлению исходных алгоритмов на основе открытого программного кода и удалению из программ водяных знаков (*watermarkings*) и «отпечатков пальцев», для защиты программного обеспечения от атак со стороны вредоносных программ (компьютерных вирусов) и обеспечения безопасности мобильных агентов в информационных сетях, для проведения безопасного поиска в потоках данных, защиты баз данных, защиты проектных решений при проектировании микроэлектронных схем. Оборотной стороной полезных достоинств обфускации является возможность ее использования для затруднения обнаружения вредоносных программ, а также создания уязвимостей в системах защиты компьютеров.

Помимо системного программирования задача обфускации программ также исследовалась в криптографии. Уже в ранних работах [4,11-14] было отмечено, что обфускация программ позволяет преобразовывать криптосистемы с секретным ключом в криптосистемы с открытым ключом: в качестве открытого ключа выступает обфускированная процедура шифрования с вставленным в нее секретным ключом. При помощи обфускации программ можно также конструировать гомоморфные системы шифрования, функциональные системы шифрования, доверенные схемы перешифрования и электронно-цифровой подписи, избавляться от модели случайного оракула при доказательстве стойкости криптографических протоколов, осуществлять случайную перестановку зашифрованных сообщений в схемах тайного голосования,

создавать схемы дезавуируемого (двусмысленного) шифрования и односторонние функции с секретом. Однако для того, чтобы каждое из перечисленных приложений обладало определенной криптографической стойкостью, используемая для его построения обфускация программ также должна удовлетворять некоторым требованиям стойкости. Поэтому при исследовании проблемы обфускации программ с позиции математической криптографии требование стойкости выдвигается на первый план.

В 2001 г. строгое математическое определение стойкости обфускации было предложено в статье [4]: обфускация считается стойкой, если всякий противник может эффективно извлечь из анализа текста обфускированной программы не больше информации, нежели при проведении тестовых испытаний, имея к ней доступ как к «черному ящику». В этой же статье было показано, что существуют такие программы, для которых такая стойкость обфускации в принципе недостижима. Впоследствии в ряде работ [15-20] были предложены и другие, менее требовательные определения стойкости обфускации; однако и для этих определений была показана невозможность построения трансляторов, гарантирующих стойкую обфускацию произвольных программ. Обзор различных определений стойкости обфускации программ представлен в статье [21].

Вместе с тем, в работах [11,12,22-24] было показано, что для отдельных классов функций стойкая обфускация программ, вычисляющих эти функции, возможна при тех или иных криптографических предположениях. Наиболее значительное продвижение было достигнуто в работе [13]; ее авторы доказали осуществимость стойкой обфускации процедуры перешифрования сообщений. В целом, однако результаты исследований в этом направлении не дают больших оснований для оптимизма: до сих пор не удалось обнаружить достаточно сложной криптографической функции, процедуры вычисления которой допускают доказуемо стойкую обфускацию.

Существенное продвижение в изучении возможности построения криптографически стойких обфускаторов программ произошло с открытием методов построения стойких систем гомоморфного шифрования. Если в начале века бытовало мнение о том, что стойкие обфускаторы программ позволят создать гомоморфные системы шифрования, то после основополагающих результатов К. Джентри (C. Gentry) [25,26] исследователи задались противоположным вопросом: насколько полезным могут быть гомоморфные криптосистемы для построения стойких обфускаторов программ?

3. Гомоморфные криптосистемы

Система шифрования с открытым ключом состоит из следующих шести компонентов:

- пространство *сообщений* (открытых текстов) M; элементы множества M играют роль данных подлежащих информационной защите;
- пространство *шифртекстов* C; элементы множества C – это зашифрованные данные;
- пространство ключей шифрования (открытых ключей) K_p и расшифрования (секретных ключей) K_s;
- процедура *генерации ключей* Gen; это эффективный (полиномиальный по времени) вероятностный алгоритм (машина Тьюринга), который по заданному параметру стойкости λ вычисляет пару ключей (sk, pk) из множества $K_s \times K_p$ (секретный ключ расшифрования и соответствующий ему открытый ключ шифрования);
- процедура *шифрования* Enc; это эффективный вероятностный алгоритм, который по заданному открытому ключу pk и сообщению m вычисляет шифртекст $c = Enc(pk, m)$ этого сообщения;

- процедура *расшифрования* Dec; это эффективный детерминированный алгоритм, который по заданному секретному ключу sk и шифртексту c вычисляет сообщение $m = Dec(sk, c)$.

Обычно пространства сообщений, шифртекстов и ключей – это множества двоичных наборов. Параметр стойкости λ определяет длину ключей n, и $K = \{0,1\}^n$. В зависимости от выбранной длины ключей определяются размерности пространства сообщений $M = \{0,1\}^{q(n)}$ и пространства шифртекстов $C = \{0,1\}^{r(n)}$, где $q(n), r(n)$ – некоторые полиномы, зависящие от n. Для удобства описания и анализа некоторых систем шифрования двоичные наборы могут рассматриваться как записи натуральных чисел в двоичной системе счисления или представления элементов специальных групп в некоторой кодировке.

Три указанные процедуры системы шифрования – генерации ключей, шифрования и расшифрования – должны удовлетворять следующему требованию *корректности*:

- для любой пары ключей (sk,pk), порожденного процедурой генерации ключей Gen, и для любого сообщения m верно равенство $Dec\big(sk, Enc(pk, m)\big) = m$.

Пусть имеются некоторые множества S_1 и S_2, на которых определены функции $f_i^{(n_i)}(x_1, \dots, x_{n_i})$ и $g_i^{(n_i)}(y_1, \dots, y_{n_i}), 1 \le i \le m,$ соответственно. Отображение $\varphi: S_1 \to S_2$ называется гомоморфизмом из алгебраической системы $\left(S_1, f_1^{(n_1)}, \dots, f_m^{(n_m)}\right)$ в алгебраическую систему $\left(S_2, g_1^{(n_1)}, \dots, g_m^{(n_m)}\right)$, если для любого $i, 1 \le i \le m$, и для любого набора $(d_1, \dots, d_{n_i})$ элементов из множества S_1 справедливо равенство

$$\varphi\left(f_i^{(n_i)}(d_1, \dots, d_{n_i})\right) = g_i^{(n_i)}$$

Предположим, что в пространстве сообщений M системы шифрования (M, C, K, Gen, Enc, Dec) определены эффективно вычислимые функции $f_i^{(n_i)}(x_1, \dots, x_{n_i})$ $, 1 \le i \le m$. Эта система шифрования считается *гомоморфной* относительно перечисленного множества функций, если в пространстве шифртекстов существуют эффективно вычислимые функции $g_i^{(n_i)}(y_1, \dots, y_{n_i}), 1 \le i \le m$, для которых при любом ключе, порождаемом процедурой генерации ключей Gen, отображение шифрования Enc является гомоморфизмом из алгебраической системы $\left(M, f_1^{(n_1)}, \dots, f_m^{(n_m)}\right)$ в алгебраическую систему $\left(C, g_1^{(n_1)}, \dots, g_m^{(n_m)}\right)$. Система шифрования считается *вполне гомоморфной*, если множество функций $f_1^{(n_1)}, \dots, f_m^{(n_m)}$ является полным относительно операции суперпозиции множеством функций на множестве M, т.е. любую эффективно вычислимую функцию на множестве M можно получить при помощи операции суперпозиции из функций указанного множества. Если $M = \{0,1\}$ (т.е. рассматриваются только булевы функции), то система шифрования является вполне гомоморфной в том и только том случае, когда она гомоморфна относительно полной системы операций булевой алгебры (например, относительно сложения и умножения). В общем случае система вполне гомоморфного шифрования снабжена эффективно вычислимой процедурой $Eval$, которая преобразует произвольное описание функции F на множестве сообщений (например, представленное схемой из функциональных элементов) в описание функции G на множестве шифртекстов, удовлетворяющей равенству

$$Enc(F(d_1, \dots, d_n), sk) = G$$

для любых наборов сообщений $(d_1, \dots, d_n)$и любых ключей sk, порождаемых процедурой генерации ключей Gen.

Понятие гомоморфного шифрования возникло в криптографии в 1978 году [27] сразу же после создания первой криптосистемы с открытым ключом. Однако все криптосистемы с открытым ключом, разработанные на рубеже веков, оказались гомоморфными относительно лишь одной из двух операций сложения и умножения битов (см., например, [28]). Наибольшим достижением в ранних работах по гомоморфному шифрованию является результат статьи [29], авторы которой предложили криптосистему, которая на основе билинейных спариваний на эллиптических кривых была способна выполнять неограниченное число сложений и одно умножение над зашифрованными данными.

Проблема построения доказуемо стойких вполне гомоморфных криптосистем оставалась открытой до 2009 года, Джентри (C. Gentry) теоретически обосновал возможность построения такой системы в статьях [25,26]. Описанная система шифрования Джентри является семантически стойкой в предположении о неразрешимости за полиномиальное время некоторых вычислительных задач, наподобие наилучшей аппроксимации заданного вектора в векторном метрическом пространстве элементом заданной решетки, вложенной в это пространство или вычисления приближенного наибольшего общего кратного целых чисел, и она способна проводить гомоморфные вычисления над зашифрованными данными. Поэтому теоретически она может быть использована для информационной защиты данных в облачных вычислениях.

Однако возможности практического использования этой системы шифрования до сих пор остаются неочевидными. В 2010 г. Смарт (N.P. Smart) и Веркотерен (F. Vercauteren) модифицировали схему Джентри, уменьшив размер ключа, но ценой усложнения процедуры генерации ключей [30]. В том же году Джентри и Халеви (S. Halevi) представили реализацию системы шифрования [31,32], а Штеле (Stehle) и Штейн (Stein) упростили эту схему и одновременно с этим повысили ее эффективность [33]. В серии работ, начиная с 2011 г., Бракерски (Z. Brakerski), Джентри, Халеви, Смарт и Вайкунтанатхан (V. Vaikuntanathan) продолжали совершенствовать схемы гомоморфного шифрования с целью повышения их эффективности [34-42]. Им удалось разработать систему гомоморфного шифрования без использования процедуры скрытного перешифрования. Эта система основана на задаче выведывания на основе примеров с ошибками (LWE, learning with errors) [38]. В ней существенно улучшена эффективность гомоморфных вычислений. В 2012 г. Бракерски доказал, что стойкая система шифрования не может иметь простую процедуру расшифрования и быть при этом вполне гомоморфной [39]. Этот результат устанавливает определенные пределы эффективного практического применения гомоморфных систем шифрования в некоторых приложениях. В 2012-13 г. Халеви, Джентри и др. исследовали вопросы организации гомоморфных вычислений над зашифрованными данными и использования систем гомоморфного шифрования в базах данных [43,32].

Возможность использования гомоморфных криптосистем для обфускации программ впервые была отмечена в статье [44]. Было показано, что при наличии вполне гомоморфной криптосистемы с открытым ключом для построения стойкой обфускации произвольной программы достаточно добиться стойкой обфускации всего лишь одной процедуры расшифрования. Дальнейшие усилия были сосредоточены на решении этой задачи [45,46,33]. Однако окончательного решения эта проблема все еще не получила.

Долгое время бытовала уверенность в том, что с изобретением систем вполне гомоморфного шифрования, будут решены многие задачи криптографии. Однако после того как криптосистема Джентри была предложена, неожиданно было обнаружено, что в некоторых случаях безопасность вычислений все равно не может быть обеспечена,

причем этот результат имеет абсолютный характер, т.е. он не зависит от стойкости систем шифрования. Впервые этот эффект был обнаружен в работе Ван Дайка (Van Dijk) и Джулса (Jules) [47]. В этой работе они рассмотрели три класса вычислений с секретными данными и сформулировали строгое определение стойкости схем вычислений над зашифрованными данными. Опираясь на это определение стойкости, они показали, что все вычисления первого класса можно сделать безопасными при помощи гомоморфных криптосистем. Однако для двух других классов это уже не так. Как было показано, вычислительные схемы второго класса можно использовать для того, чтобы построить стойкий обфускатор программ в модели «черного ящика». Поскольку ранее было доказано, что такой обфускатор не существует, отсюда следует, что стойкой информационной защиты схем вычислений второго класса построить также невозможно.

Таким образом, для защиты данных в облачных системах вычислений необходимы дополнительные средства. В статье [48] был предложен один подход к организации безопасных вычислений в облачной среде с использованием пороговых гомоморфных систем шифрования.

4. Пороговые системы гомоморфного шифрования

Пороговая система гомоморфного шифрования с открытым ключом состоит из множества криптосерверов $S_1, S_2, \dots, S_l$, которые можно рассматривать как интерактивные вероятностные машины Тьюринга. Все криптосерверы попарно соединены друг с другом каналами связи и предназначены для выполнения следующих алгоритмов.

1. Протокол генерации ключей. При выполнении этого протокола на вход каждого криптосервера поступает натуральное число n в унарной форме записи, которое служит параметром стойкости системы шифрования. Получив это число, каждый криптосервер S_i вычисляет свою долю секретного ключа s_i, а затем все криптосерверы совместно в интерактивном режиме вычисляют открытый ключ pk, соответствующий сгенерированным долям $s_1, \dots, s_l$ секретного ключа.
2. Алгоритм шифрования Enc. На вход алгоритма поступает открытый ключ pk и открытый текст m; на выходе алгоритма вычисляется криптограмма $c = Enc(pk, m)$. Алгоритм шифрования может быть выполнен на любой вероятностной машине Тьюринга за полиномиальное время.
3. Протокол расшифрования, который включает процедуры расшифрования, выполняемые криптосерверами, и процедуру интеграции, которая выполняется пользователем криптосистемы. Каждый криптосервер S_i, получив на входе криптограмму c, вычисляет, используя свою долю секретного ключа s_i, фрагмент расшифрованного сообщения $Dec(s_i, c)$. Пользователь, располагая криптограммой c и всеми фрагментами расшифрованного сообщения $Dec(s_1, c), \dots, Dec(s_l, c)$, восстанавливает зашифрованное сообщение m при помощи процедуры интеграции. Для выполнения процедуры интеграции секретные ключи не нужны.

Примером пороговой системы гомоморфного шифрования может служить криптосистема, описанная в статье [7]. Организация вычислений над зашифрованными данными с применением подобной криптосистемы описана в статье [49].

Обычно рассматриваются два сценария использования пороговых криптосистем, в зависимости от того, кто является получателем информации после расшифрования.

В первом из них получателем открытого текста является каждый из криптосерверов. Основное требование к криптосистеме состоит в существовании такого порога $t, t < l$, что любая криптограмма может быть расшифрована только при согласии не менее чем t криптосерверов. Именно этот вариант обычно рассматривается в литературе.

Во втором сценарии получателем открытого текста является внешний участник, а функции криптосерверов полностью соответствуют их названию. В этой статье мы будем рассматривать пороговые криптосистемы с неинтерактивным протоколом расшифрования: крпитосервер S_i, получив криптограмму c, вычисляет некоторый фрагмент, обозначаемый записью $Dec(s_i, c)$, и посылает его получателю открытого текста по защищенному каналу связи. Существует эффективный алгоритм интеграции, который, получив на вход $c, Dec(s_1, c), \dots, Dec(s_l, c)$, вычисляет открытый текст m. В этом сценарии основное требование к криптосистеме формулируется так: существует такой порог $t, t < l$, что знание любых t долей секретного ключа не позволяет расшифровать криптограмму.

Мы приведем определение стойкости пороговой системы гомоморфного шифрования для второго варианта применения криптосистемы. Пороговая система шифрования с открытым ключом является пороговой не вполне гомоморфной криптосистемой (Threshold Somewhat Homomorphic Encryption, TSHE), если существуют такой эффективный алгоритм $Eval$ и такой параметр d, что для любой булевой схемы F глубины не более d с k входами, для любых $m_1, \dots, m_k \in \{0,1\}$ результат $Eval(pk, c_1, \dots, c_k, f)$ - это криптограмма открытого текста $F(m_1, \dots, m_k)$. Здесь $c_j = Enc(pk, m_j)$, а f – это битовая строка, описывающая схему F. Формально алгоритм $Eval$ определяется как полиномиальная вероятностная машина Тьюринга. Параметр d, вообще говоря, является функцией параметра стойкости n и других параметров TSHE.

В отличие от вполне гомоморфных, не вполне гомоморфные криптосистемы не используют процедуру перешифрования (*bootstrapping*) и потому могут проводить лишь ограниченные вычисления.

Для определения стойкости TSHE воспользуемся моделью противника, который представляет собой полиномиальную вероятностную машину Тьюринга Adv. Противнику доступна атака с известным открытым ключом pk и известными t долями секретного ключа. Все доли секретного ключа равноправны, и поэтому без ограничения общности далее мы будем считать, что это доли $s_1, \dots, s_t$. Угроза различения открытых текстов по их криптограммам определяется следующим образом: противник выбирает пару открытых текстов m^0, m^1 одинаковой длины, получает криптограмму $c = Enc(pk, m^\sigma)$, где $\sigma \in_R \{0,1\}$, и угадывает σ с вероятностью, существенно отличающейся от 1/2. Поскольку гомоморфные криптосистемы работают с однобитовыми открытыми текстами, здесь предполагается, что строки m^0и m^1 шифруются побитово и c - конкатенация соответствующих криптограмм.

Для формализации угрозы машине Adv предоставляется доступ к оракулу O, который, получив от противника Adv пару (m^0, m^1), выбирает случайный бит σ, вычисляет криптограмму $c = Enc(pk, m^\sigma)$ и возвращает c в качестве ответа на запрос.

Определение 1. TSHE называется семантически стойкой, если для любой полиномиальной вероятностной машины Тьюринга Adv имеет место равенство

$$Pr[Adv^O(pk, s_1, \dots, s_t) = \sigma] - 1/2 = \nu(n),$$

где $\nu(n)$ пренебрежимо малая функция, т.е. функция, удовлетворяющая соотношению $\nu(n) = o(1/P(n))$ для любого полинома P.

5. Модель облачных вычислений

В исследовании задачи построения стойкой системы обфускации программ в облачных вычислениях рассматривается следующая модель облачной среды с использованием криптосерверов и гомоморфного шифрования. Эта модель включает следующие компоненты.

1. **Участники (агенты)**: облачный сервер, пользователи системы системы облачных вычислений $U_1, \dots, U_k$, клиенты системы облачных вычислений $C_1, \dots, C_r$, криптосерверы $S_1, \dots, S_l$.
2. **Телекоммуникационная сеть**. Каждый из пользователей, а также каждый из клиентов имеет канал связи с облачным сервером. Каналы связи с криптосерверами образуют полный граф с l вершинами. Каждый криптосервер, помимо этого, имеет канал связи с облачным сервером.
3. **Данные**. У каждого пользователя $U_i, i = 1, \dots, k$ имеется конфиденциальная информация m_i, которые должны храниться и обрабатываться на облачном сервере/
4. **Программа**. На облачном сервере также размещается код программы F, который может быть зашифрован $Enc(pk, F)$ (например, при помощи гомоморфной системы шифрования). Каждый клиент $C_i, i = 1, \dots, r$ имеет право передать облачному серверу запрос на применение программ к данным пользователей этой системы. Этот запрос включает в себя идентификатор клиента C_i, открытой ключ p_i некоторой системы шифрования с открытым ключом.
5. **Контроль доступа**. Облако передает запрос клиента C_i в центр аутентификации, который проверяет полномочия клиента на применение программы F, и при наличии таковых, санкционирует выполнение запроса.
6. **Параметр стойкости**: натуральное число n; каждый из основных криптографических параметров модели (длина ключей и пр.) ограничен некоторым полиномом, зависящим от n.
7. **Вычислительные ресурсы**. Все участники рассматриваемой модели (включая противника) представляют собой вероятностные машины Тьюринга, работающие за полиномиальное от n время.

Основные допущения, в рамках которых рассматривается данная модель вычислений, таковы.

- Пользователи не доверяют облачному серверу, который может рассматриваться в качестве пассивного противника. Это означает, что все функции по хранению и обработке конфиденциальных данных выполняются облачным сервером корректно, но все сведения, доступные облачному серверу также считаются доступными противнику и могут быть использованы для компрометации как хранимых пользовательских данных, так и обрабатывающих эти данные программ.
- Криптосерверы связаны между собой защищенными каналами связи, и память каждого криптосервера также защищена. Это может быть достигнуто с применением аппаратных или программных средств информационной защиты, включающей криптосистемы шифрования, электронно-цифровой подписи и пр.
- Существует такое натуральное число t, которое называется порогом и ограничивает сверху количество скомпрометированных криптосерверов. Криптосервер считается скомпрометированным, если он проводит возложенные на него вычисления некорректно или допускает утечку доступных ему данных.

6. Обфускация программ в модели облачных вычислений

Интерес представляет следующая задача построения эффективной и стойкой системы обфускации программ в описанной модели облачных вычислений. Владелец программы зашифровывает ее посредством гомоморфной системы шифрования и передает для хранения на облачный сервер. Любой клиент системы, желающий воспользоваться программой, обращается с запросом. Если запрос удовлетворен, то клиент отправляет данные на криптосерверы, которые, используя пороговую гомоморфную систему шифрования, шифруют данные и передают их на облачный сервер. Облачный сервер проводит вычисление зашифрованной программы над зашифрованными данными и

отправляет зашифрованный результат вычисления. Получив этот зашифрованный результат, криптосерверы, используя доли секретного ключа гомоморфной системы шифрования, расшифровывают его и отправляют клиенту. Обфускацией программы в данном случае является ее шифрование посредством гомоморфной криптосистемы, реализованной на криптосерверах.

Основной задачей здесь является исследование стойкости предложенного метода обфускации программ в рассматриваемой модели облачных вычислений.

Без ограничения общности можно предполагать, что в системе облачных вычислений имеется единственный клиент. Пусть p – открытый ключ некоторой системы шифрования РКС с открытым ключом и функцией шифрования E. Можно считать, что эта система шифрования удовлетворяет стандартному определению семантической стойкости.

Предполагается, что о противнике (атаке) известно следующее:

- открытые ключи пользователя и системы гомоморфного шифрования, которая реализована на криптосерверах;
- обфускированная программа $Enc(pk, F)$;
- доли секретного ключа $s_1, \dots, s_t$ гомоморфной системы шифрования, которыми владеют скомпрометированные криптосерверы.

Противник также имеет доступ к специальному оракулу S. Запросом к оракулу служит произвольная криптограмма c. В ответ оракул возвращает набор фрагментов, $Dec(s_1, c), \dots, Dec(s_t, c)$, расшифрованных при помощи тех долей секретного ключа, которыми располагают скомпрометированные криптосерверы. Оракул S моделирует возможность противника контролировать не более t криптосерверов.

Угроза состоит в различении обфускации в данной модели и идеальной обфускации, которая определяется как модель, в которой противнику доступны лишь открытый ключ pk и значения $F(m_1, \dots, m_k)$ всех возможных результатов применения программы к клиентским данным. Эти значения становятся доступны при обращении к оракулу F.

Определение 2. Обфускация в модели облачных вычислений с вспомогательными криптосерверами называется стойкой, если для любой полиномиальной вероятностной машины Тьюринга A существует такая полиномиальная вероятностная машина Тьюринга B, что для любой программы F, размер которой не зависит от параметра стойкости, и для любого набора пользовательский данных $m_1, \dots, m_k$ справедливо неравенство

$$|Pr[A^S(pk, p, Enc(pk, F), s_1, \dots, s_t) = 1] - Pr[B^F(pk) = 1]| \leq \nu(n),$$

где $\nu(n)$ – пренебрежимо малая функция (величина), и вероятность вычисляется относительно случайных величин, используемых в алгоритмах шифрования, а также относительно открытых ключей шифрования p, pk.

Нами была доказана следующая

Теорема 1. Если существует стойкая пороговая криптосистема гомоморфного шифрования, то существует стойкая обфускация в модели облачных вычислений с использованием криптосерверов.

Доказательство. Прежде всего, заметим, что из предположения о существовании стойкой пороговой системы гомоморфного шифрования следует существование семантически стойкой системы шифрования с открытым ключом $PKE(G, E, D)$.

Рассмотрим систему облачных вычислений с криптосерверами, на которых реализована пороговая гомоморфная криптосистема с открытым ключом $THSE(Gen, Enc, Dec, Eval)$. Обфускация программ в такой системе проводится следующим образом.

1. Вначале криптосерверы выполняют протокол генерации ключей Gen, в результате работы которого каждый криптосервер $S_i, 1 \le i \le l$, формирует свою долю секретного ключа s_i, и, кроме того, вычисляется и публикуется открытый ключ pk, соответствующий этим долям секретного ключа.
2. Владелец программы выполняет протокол загрузки программы, в результате которого вычисляется шифртекст $Ob(F) = Enc(pk, F)$ программы Fи размещается в памяти облачной системы вычислений.
3. Клиент системы облачных вычислений выполняет протокол подготовки данных для вычисления; в результате выполнения этого протокола вычисляются шифртексты $c_i = Enc(pk, m_i)$ данных $m_i, 1 \le i \le k$,и размещаются в памяти облачной системы вычислений. Кроме того, клиент, используя протокол генерации ключей G криптосистемы с открытым ключом, формирует секретный ключ s и открытый ключ p, вычисляет шифр открытого ключа $p' = Enc(pk, p)$ и размещает его в памяти облачной системы.
4. Облачная система, используя алгоритм вычислений над зашифрованными данными $Calc$, применяет зашифрованную программу $Ob(F)$ к зашифрованным данным $c_1, \dots, c_k, p'$, вычисляет дважды зашифрованный результат $c = Enc\left(pk, E\left(p, F(m_1, \dots, m_k)\right)\right)$, и отправляет его криптосерверам.
5. Криптосерверы $S_i, 1 \le i \le l$ выполняют протокол расшифрования Dec, используя доли секретного ключа s_i, и вычисляют, используя открытый ключ клиента p, зашифрованные фрагменты $d_i = E\left(p, Dec(s_i, c)\right)$ зашифрованного результата c и отправляют их клиенту.
6. Клиент получает от облачной системы и криптосерверов строки $c, d_1, \dots, d_l$. Используя секретный ключ s, клиент вначале расшифровывает криптограммы $d_1, \dots, d_l$ и извлекает фрагменты $Dec(s_1, c), \dots, Dec(s_l, c)$. Затем, применяя алгоритм интеграции расшифрованных фрагментов, клиент на основе строк $c, Dec(s_1, c), \dots, Dec(s_l, c)$ вычисляет зашифрованный результатат вычисления $E\left(p, F(m_1, \dots, m_k)\right)$. И, наконец, используя еще раз секретный ключ s, клиент извлекает результат вычисления $F(m_1, \dots, m_k)$.

Покажем, что предложенная обфускация программ является стойкой.

Предположим противное. Тогда существует такие полиномиальная вероятностная машина Тьюринга A, набор входных данных $m_1, \dots, m_k$, и программа F, что для любой полиномиальной вероятностной машины Тьюринга B неравенство

$$|Pr[A^S(pk, p, Enc(pk, F), s_1, \dots, s_t) = 1] - Pr[B^F(pk) = 1]| > \varepsilon(n),$$

выполняется для бесконечно многих значений n. Здесь $\varepsilon(n)$ – функция, не являющаяся пренебрежимо малой, т.е. удовлетворяющая неравенству $\varepsilon(n) \ge 1/P\,(n)$ для некоторого полинома P.

Рассмотрим два сценария выполнения алгоритма A. В первом сценарии оракул S отвечает на запросы алгоритма в соответствии со своим предназначением, т.е. в ответ на запрос c возвращает набор криптограмм $Dec(s_1, c), \dots, Dec(s_t, c)$. Пусть

$$\delta_1^A(n) = Pr[A^S(pk, p, Enc(pk, F), s_1, \dots, s_t) = 1].$$

Во втором сценарии вместо оракула S противник использует оракул Q, который в ответ на каждый запрос c длины $|c| = r$ возвращает набор строк $Dec\left(s_1, Enc\left(pk, E(p, 0^r)\right)\right), \dots, Dec\left(s_t, Enc\left(pk, E(p, 0^r)\right)\right)$. Пусть

$$\delta_2^A(n) = Pr[A^Q(pk, p, Enc(pk, F), s_1, \dots, s_t) = 1].$$

Лемма 1. Для любой полиномиальной вероятностной машины Тьюринга верно соотношение

$$\delta_1^A(n) - \delta_2^A(n) = \nu(n)$$

Доказательство леммы.

Предположим противное: существует такая полиномиальная вероятностная машина Тьюринга A, для которой для бесконечно многих значений n неравенство $\delta_1^A(n) - \delta_2^A(n) \geq 1/Q\,(n)$ выполняется для некоторого полинома $Q(n)$. Покажем, что из этого предположения следует нестойкость системы шифрования РКС, которую использует клиент.

Построим полиномиальную вероятностную машину Тьюринга Adv_E, которая будет выступать в роли противника по отношению к РКС. Эта машина имеет возможность обращаться к случайному оракулу O. На входе p машина Adv_E создает описанную выше систему облачных вычислений и обращается к машине A, подавая ей на вход $pk, p, Enc(pk, F)$. Когда машина обращается к оракулу с запросом c, противник Adv_Eформирует пару $\Big(\big(Dec(s_1, c), \dots, Dec(s_l, c)\big), 0^r\Big)$, второй компонентой которой является нулевая строка той же длины, что и первый элемент пары, и передает эту пару оракулу O. Оракул возвращает в ответ криптограмму c^*, и противник Adv_E передает эту криптограмму машине A. Выходом машины Adv_E служит выход машины A.

В том случае, когда c^* - это криптограмма первого элемента пары, все параметры и случайные величины, с которыми работает противник Adv_E, соответствуют работе машины A с оракулом S. Поэтому $Pr[Adv_E^O(p) = 1] = \delta_1^A(n)$. А если c^* – это криптограмма второго элемента пары, то в этом случае вычисление противника Adv_E, соответствует работе машины A с оракулом Q. Поэтому $Pr[Adv_E^O(p) = 0] = \delta_2^A(n)$. Коль скоро согласно предположению $\delta_1^A(n) - \delta_2^A(n) \geq 1/Q\,(n)$ для бесконечно многих значений n, получаем неравенство $|Pr[Adv_E^O(p) = 1] - Pr[Adv_E^O(p) = 0]| \geq 1/Q\,(n)$, противоречащее условию семантической стойкости системы шифрования РКС, которая используется в модели облачных вычислений.

Лемма доказана.

Располагая доказанной леммой, построим противника Adv для криптосистемы TSHE, используемой в модели облачных вычислений. Получив на входе открытый ключ pk, машина Adv формирует пару (m^0, m^1), где $m^1 = (m_1, \dots, m_k), m^0 = 0^r$, где $r = m^1 \vee$, и передает эту пару оракулу O, который возвращает в ответ криптограмму $c = Enc(pk, m^i)$, где $i \in \{0,1\}$. После этого противник Adv запускает машину A и подает на ее вход следующие данные: $pk, p, Enc(pk, F), c$.

Пусть $\sigma_1(n)$ – это вероятность совместного осуществления следующих двух событий: 1) оракул O вычислил криптограмму $c = Enc(pk, m^1)$ и 2) $Adv^O(pk) = 1$, а $\sigma_0(n)$ – это вероятность совместного осуществления следующих двух событий: 1) оракул O вычислил криптограмму $c = Enc(pk, m^0)$ и 2) $Adv^O(pk) = 0$.

Оценим величину $\sigma_1(n) + \sigma_0(n)$, которая равна вероятности того, что противник Adv принял правильное решение. Из описание противника Adv и определения вероятностей $\sigma_1(n)$ и $\sigma_0(n)$ следует, что их сумма равна величине

$$1/2\left(1 - Pr\left[A^Q\left(pk; p; Enc(pk, F); Enc(pk, m^0)\right) = 1\right]\right) +$$
$$1/2\left(Pr\left[A^Q\left(pk; p; Enc(pk, F); Enc(pk, m^1)\right) = 1\right]\right),$$

или, что то же самое

$$1/2 + 1/2(Pr\left[A^Q\left(pk; p; Enc(pk, F); Enc(pk, m^1)\right) = 1\right] -$$
$$Pr\left[A^Q\left(pk; p; Enc(pk, F); Enc(pk, m^0)\right) = 1\right].$$

Согласно лемме 1 верно, что

$$Pr\left[A^Q\left(pk; p; Enc(pk, F); Enc(pk, m^1)\right) = 1\right] -$$

$$Pr\left[A^S\left(pk; p; Enc(pk, F); Enc(pk, m^1)\right) = 1\right] = \nu(n).$$

Учитывая, что $m^0 = 0^r$, нетрудно заметить, что существует такая полиномиальная вероятностная машина Тьюринга B, для которой верно соотношение

$$Pr\left[A^Q\left(pk; p; Enc(pk, F); Enc(pk, m^0)\right) = 1\right] - Pr[B^F(pk)] = \nu(n).$$

И, наконец, как следует из предположения о нестойкости предложенной системы обфускации программ, соотношение

$$|Pr[A^S(pk, p, Enc(pk, F), s_1, \dots, s_t) = 1] - Pr[B^F(pk) = 1]| > \varepsilon(n)$$

верно для некоторой функции $\varepsilon(n)$, не являющейся пренебрежимо малой, для бесконечно многих значений n.

Поэтому на основании приведенных неравенств приходим к заключению о том, что

$$Pr\left[A^Q\left(pk; p; Enc(pk, F); Enc(pk, m^1)\right) = 1\right] -$$
$$Pr\left[A^Q\left(pk; p; Enc(pk, F); Enc(pk, m^0)\right) = 1\right] > \varepsilon(n) - \nu(n),$$

а это означает, что $\sigma_1(n) + \sigma_0(n) > 1/2 + \varepsilon(n)/2$. Последнее неравенство означает, в частности, что противник Adv компрометирует криптосистему TSHE вопреки условию о том, что TSHE является семантически стойкой.

Полученное противоречие означает, что гипотеза о том, что предложенная система обфускации программ не является стойкой, несостоятельна.

Доказательство теоремы закончено.

7. Заключение

Основные выводы, которые могут быть извлечены из проведенных в настоящее время исследований применимости методов криптографии для информационной защиты облачных вычислений таковы.

1. Существует большое число эффективных и стойких систем шифрования, являющихся гомоморфными относительно отдельных алгебраических операций (сложения, умножения).
2. Разработано несколько систем вполне гомоморфного шифрования, стойкость которых доказана. Однако все эти разработки пока носят исключительно теоретический характер, и вычисления, которые можно проводить с использованием этих криптосистем, все еще чрезвычайно неэффективны.
3. Для разработки криптостойких схем облачных вычислений необходимы формальные математические модели. Отдельные модели были разработаны для этой цели. Однако разнообразие этих моделей все еще невелико, и они не охватывают многие аспекты облачных вычислений.
4. При помощи предложенных формальных моделей облачных вычислений удалось показать, что существование систем вполне гомоморфного шифрования еще не является достаточным условием решения задачи информационной защиты облачных вычислений. Более того, было показано, что некоторые схемы облачных вычислений принципиально не могут иметь стойкой информационной защиты.

Перспективным для дальнейших исследований представляется подход к разработке пороговых систем гомоморфного шифрования и использования системы распределенных доверенных серверов для выполнения критических по требованиям безопасности криптографических процедур.

Список литературы / References

[1]. Goldwasser S., Micali S. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Proc. of the 14th ACM Symposium on the Theory of Computing, 1982, pp. 365–377.

[2]. Ostrovsky R. Efficient computation on oblivious RAMs. Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, 1990, p. 514-523.

[3]. Ostrovsky R., Skeith III W.E. Private searching on streaming data. Lecture Notes in Computer Science, vol. 3621, 2005, p. 223-240.

[4]. Barak B., Goldreich O., Impagliazzo R., Rudich S., Sahai A., Vedhan S., Yang K. On the (Im)possibility of obfuscating programs. Journal of the ACM, vol, 59, no, 2, 2012, article no. 6

[5]. Collberg C., Thomborson C., Low D. A taxonomy of obfuscating transformations. Technical Report, no. 148, Department of Computer Science, University. of Auckland, 1997.

[6]. Collberg C, Thomborson C. Watermarking, tamper-proofing, and obfuscation - tools for software protection. IEEE Transactions on Software Engineering, vol. 28, no. 6, 2002, pp. 735 - 746.

[7]. D'Anna L., Matt B., Reisse A., Van Vleck T., Schwab S., LeBlanc P. Self-protecting mobile agents obfuscation report. Report 03-015, Network Associates Laboratories, 2003.

[8]. Diffie W., Hellman M. New directions in cryptography. IEEE Transactions on Information Theory, vol. 22, issue 6, 1976, pp. 644–654.

[9]. Sahai A., Waters B. How to Use Indistinguishability Obfuscation: Deniable Encryption, and More. In Proc. of the 46th Annual ACM Symposium on Theory of Computing, 2014, pp. 475–484.

[10]. Varnovsky N.P. A note on the concept of obfuscation. Trudy ISP RAN/Proc. ISP RAS, vol. 6, 2004, pp. 127-136.

[11]. Garg S., Gentry C., Halevi S., Raykova M., Sahai M., Waters B. Candidate Indistinguishability Obfuscation and Functional Encryption for all circuits. SIAM Journal on Computing, vol. 45, no. 3, 2016, pp. 882-929.

[12]. Hofheinz D., Malone-Lee J., Stam M. Obfuscation for cryptographic purposes. Lecture Notes in Computer Science, vol. 4392, 2007, pp. 214-232.

[13]. Hohenberger S., Rothblum G. N., Shelat A., Vaikuntanathan V. Securely obfuscating reencryption. Lecture Notes in Computer Science, vol 4392, 2007, pp. 233-252.

[14]. Pass K., Seth K., Telang S. Indistinguishability Obfuscation from Semantically-Secure Multilinear Encodings. Lecture Notes in Computer Science, vol. 8616, 2014, p. 500-517.

[15]. Goldwasser S., Rothblum G.N. On best possible obfuscation. Lecture Notes in Computer Science, vol 4392, 2007, pp. 194-213.

[16]. Hada S. Secure obfuscation for encrypted signatures. Lecture Notes in Computer Science, vol. 6110, 2010, pp. 92-112..

[17]. Lynn B., Prabhakaran M., Sahai A. Positive results and techniques for obfuscation. Lecture Notes in Computer Science, vol. 3027, 2004, pp. 20-39.

[18]. Varnovsky N.P., Zakharov V.A. On the possibility of provably secure obfuscating programs. Lecture Notes in Computer Science, vol. 2890, 2003, pp. 91-102.

[19]. Goldwasser S., Micali S. Probabilistic encryption and how to play mental poker keeping secret all partial information. In Proc. of the 14th ACM Symposium on the Theory of Computing, 1982, pp. 365–377.

[20]. Goldwasser S., Tauman Kalai Y. On the impossibility of obfuscation with auxiliary input. In Proc. of the 46th IEEE Symposium on Foundations of Computer Science, 2005, pp. 553-562..

[21]. Варновский Н.П., Захаров В.А., Кузюрин Н.Н., Шокуров А.В. Современное состояние исследований в области обфускации программ: определения стойкости обфускации. Труды ИСП РАН, том 26, вып. 3, 2014 г., стр. 167-198 / Varnovsky N.P., Zakharov V.A.., Kuzurin N.N., Shokurov A.V. The current state of art in program obfuscations:definitions of obfuscation security. . Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 3, 2014, pp. 167-198 (in Russian). DOI: 10.15514/ISPRAS-2014-26(3)-9.

[22]. Kuzurin N.N., Shokurov A.V., Varnovsky N.P., Zakharov V.A. On the concept of software obfuscation in computer security. Lecture Notes in Computer Science, vol.4779, 2008, pp. 281-298.

[23]. Min. Zao E, Yang Ceng. Homomorphic Encryption Technology for Cloud Computing. Procedia Computer Science, vol. 154, 2019, pp. 73-83.

[24]. Wee H. On obfuscating point functions. In Proc. of 37th ACM Symposium on Theory of Computing, 2005, p. 523-532.
[25]. Gentry C. Computing Arbitrary Functions of Encrypted Data. Communication of the ACM, vol. 53, no. 3, 2010. pp. 97-105.
[26]. Gentry C., Halevi S. Implementing Gentry's Fully-Homomorphic Encryption Scheme. Lecture Notes in Computer Science, vol. 6632, 2011, pp. 129-148.
[27]. Rivest R.L., Adleman L., Dertouzos M.L. On data banks and privacy homomorphisms. In DeMillo R.A., ed., Foundations on Secure *C*omputation, Academia Press, 1978, pp. 169-179.
[28]. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. Lecture Notes in Computer Science, vol. 1592, 1999, pp. 223–238.
[29]. Boneh D., Goh Eu-Jin, Nissim K. Evaluating 2-DNF formulas on ciphertexts. Lecture Notes in Computer Science, vol. 3378, 2005, pp. 325-341.
[30]. Smart N., Verkauteren F. Fully homomorphic encryption with relatively small ciphertext and key size. Lecture Notes in Computer Science, vol. 6056, 2010, pp. 420-443.
[31]. Gentry C., Halevi S., Smart N. Homomorphic Evaluation of the AES Circuit. Lecture Notes in Computer Science, vol. 7417. 2012, pp. 850-867.
[32]. Gentry C., Halevi S., Smart N. Better Bootstrapping in Fully Homomorphic Encryption. Lecture Notes in Computer Science, vol. 7293, 2012, pp. 1-16.
[33]. Stehle D., Steinfeld R. Faster Fully Homomorphic Encryption. Lecture Notes in Computer Science, vol. 6477, 2010, pp. 377-394.
[34]. Brakerski Z., Vaikuntanathan V. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. Lecture Notes in Computer Science, vol. 6841, 2011, pp. 505-524.
[35]. Brakerski Z. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. Lecture Notes in Computer Science, vol. 7417, 2012, pp. 868-886.
[36]. Brakerski Z., Gentry C., Vaikuntanathan V. (Leveled) Fully Homomorphic Encryption without Bootstrapping. ACM Transactions on Computation Theory, vol. 18, no. 3, 2014, article no. 13.
[37]. Brakerski Z., Gentry C., Halevi S. Packed Ciphertexts in LWE-based Homomorphic Encryption.. Lecture Notes in Computer Science, vol 7778, 2013, pp. 1-13.
[38]. Brakerski Z. When Homomorphism Becomes a Liability. Lecture Notes in Computer Science, vol. 7785, 2013, pp. 143-161.
[39]. Brakerski Z., Vaikuntanathan V. Lattice-Based FHE as Secure as PKE. In Proc. of the 5th Conference on Innovations in Theoretical Computer Science, 2014, pp.1-12.
[40]. Brakerski Z., Rothblum G.N. Virtual Black-Box Obfuscation for All Circuits via Generic Graded Encoding. Lecture Notes in Computer Science, vol. 8349, 2014, pp. 1-25.
[41]. Gentry C., Halevi S., Smart N. Fully Homomorphic Encryption with Polylog Overhead. Lecture Notes in Computer Science, vol. 7237, 2012, pp. 465-482.
[42]. Gentry C., Sahai A., Waters B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. Lecture Notes in Computer Science, vol. 8042, 2013, pp. 75-92. DOI: 10.1007/978-3-642-40041-4_5.
[43]. Boneh B., Gentry C., Gorbunov S., Halevi S., Nikolaenko V., Segev G., Vaikuntanathan V., Vinayagamurthy D. Lecture Notes in Computer Science, vol. 8441, 2014, pp. 533-556.
[44]. Gentry C. Fully homomorphic encryption using ideal lattices. In Proc. of the 41st ACM Symposium on Theory of Computing, 2009, pp. 169-178.
[45]. Barak B., Garg S., Tauman Kalai Y., Paneth O., Sahai A. Protecting Obfuscation against Algebraic Attacks. Lecture Notes in Computer Science, vol. 8441, 2014, pp. 221–238.
[46]. Canetti R., Dwork C., Naor M., Ostrovsky R. Deniable encryption. Lecture Notes in Computer Science, vol. 1294, 1997, p. 90-104.
[47]. Van Dijk M., Juels A. On the impossibility of cryptography alone for privacy preserving cloud computing. On Proc. of the 5th USENIX Conference on Hot Topics in Security, 2010, pp. 1–8.
[48]. Варновский Н.П., Мартишин С.А., Храпченко М.В., Шокуров А.В. Методы пороговой криптографии для защиты облачных вычислений. Труды ИСП РАН, том 26, вып. 2, 2014, стр. 269-274 / Varnovskij N.P., Martishin S.A., Khrapchenko M.V., Shokurov A.V. A Threshold Cryptosystem in Secure Cloud Computations. Trudy ISP RAN/Proc. ISP RAS, vol. 26, issue 2, 2014, pp. 269-274 (in Russian). DOI: 10.15514/ISPRAS-2014-26(2)-12.
[49]. Варновский Н.П., Захаров В.А., Шокуров А.В. К вопросу о существовании доказуемо стойких систем облачных вычислений. Вестник Московского университета. Серия 15: Вычислительная математика и кибернетика, 2016, no. 2, стр. 32-37 / Varnovsky N.P., Zakharov

V.A., Shokurov A.V. On the Existence of Provably Secure Cloud Computing Systems. Moscow University Computational Mathematics and Cybernetics, vol. 40, no. 2, 2016, pp. 83–88.

Информация об авторах / Information about authors

Александр Владимирович ШОКУРОВ – кандидат физико-математических наук, доцент, ведущий научный сотрудник отдела прикладной математики и информатики ИСП РАН, доцент кафедры системного программирования МФТИ. Сфера научных интересов: гомоморфное шифрование, облачные вычисления, криптография на решетках, алгебра.

Alexander Vladimirovich SHOKUROV – Candidate of Physics and Mathematics, Associate Professor, Leading Researcher of the Department of Applied Mathematics and Computer Science, ISP RAS, Associate Professor of the Department of System Programming at MIPT. Research interests: homomorphic encryption, cloud computing, lattice cryptography, algebra.

Ирина Валерьевна АБРАМОВА – студентка магистратуры факультета вычислительной математики и кибернетики МГУ. Сфера научных интересов: математические методы криптоанализа, криптографические методы защиты информации.

Irina Valerievna ABRAMOVA – Master of Science, faculty of computational mathematics and cybernetics, Lomonosov Moscow State University. Research interests: mathematical methods of cryptoanalysis, cryptographic methods for information protection.

Николай Павлович ВАРНОВСКИЙ – старший научный сотрудник отдела прикладной математики и информатики ИСП РАН, старший научный сотрудник Института проблем информационной безопасности МГУ, преподаватель кафедры системного программирования МФТИ. Сфера научных интересов: теория сложности вычислений, математические методы криптоанализа, криптографические методы защиты информации.

Nikolay Pavlovich VARNOVSKY – Senior Researcher, Department of Applied Mathematics and Informatics, ISP RAS; Senior Researcher, Institute of Information Security Problems, Moscow State University; Lecturer, Department of System Programming, MIPT. Research interests: theory of computational complexity, mathematical methods of cryptanalysis, cryptographic methods for information protection.

Владимир Анатольевич ЗАХАРОВ – доктор физико-математических наук, профессор, старший научный сотрудник отдела прикладной математики и информатики ИСП РАН; профессор кафедры математической кибернетики МГУ; ведущий научный сотрудник лаборатории процессно-ориентированных информационных ВШЭ; доцент кафедры системного программирования МФТИ. Сфера научных интересов: теория сложности вычислений, математические методы криптоанализа, математическая логика, теория автоматов, методы верификации программ.

Vladimir Anatolyevich ZAKHAROV – Doctor of Physics and Mathematics, Professor, Senior Researcher, Department of Applied Mathematics and Computer Science, ISP RAS; Professor, Department of Mathematical Cybernetics, Moscow State University; Leading Researcher, HSE Laboratory of Process-Oriented Information Systems; Associate Professor, Department of System Programming, MIPT. Research interests: theory of computational complexity, mathematical methods of cryptanalysis, mathematical logic, theory of automata, methods of program verification.

DOI: 10.15514/ISPRAS-2019-31(6)-9

CC BY

Моделирование метеоусловий в районе порта и в прибрежной зоне залива Тикси

[1] *А.В. Иванов, ORCID: 0000-0002-2052-3912 <av.ivanov@ispras.ru>*
[2] *С.В. Стрижак, ORCID: 0000-0001-5525-5180 <s.strijhak@ispras.ru>*
[3] *М. И. Захаров, ORCID: 0000-0002-8916-2166 <mplusz@inbox.ru>*
[1] *Институт прикладной математики им. М.В. Келдыша РАН, 125047, Россия, г. Москва, Миусская пл., 4*
[2] *Институт системного программирования им. В.П. Иванникова РАН, 109004, Россия, г. Москва, ул. А. Солженицына, д. 25*
[3] *ФГАОУ ВО Северо-Восточный федеральный университет им. М.К. Аммосова, 677000, Россия, г. Якутск, ул. Белинского, д. 58*

Аннотация. Ветроэнергетика – одно из важнейших направлений в развитии возобновляемых источников энергии в РФ. Наиболее важными направлениями в ветроэнергетике являются задачи проектирования новых ветропарков, задачи эксплуатации и мониторинга ветропарков, задачи проектирования ветроэлектрических установок (ВЭУ), разработка тематического российского программного обеспечения и т.п. В связи со строительством новых ветропарков на территории РФ возникает ряд актуальных фундаментальных и прикладных задач. В данной работе рассматривается одна из таких задач: исследование ветровой обстановки в районе поселка Тикси, ввиду установки там ветроэлектростанции. Прогнозирование ветровой обстановки производится на основе мезомасштабной модели Advanced Research WRF (Weather Research and Forecasting) с использованием данных Global Forecast System (GFS) модели. Представленные расчеты производились на серии вложенных сеток с пространственным разрешением 1, 3 и 9 км. Предлагается методика для исследования и мониторинга ветропарка вблизи поселка Тикси, которая основана на интерполяции результатов, полученных из Advanced Research WRF, на модель меньшего масштаба в рамках библиотеки Simulator fOr Wind Farm Applications (SOWFA) открытого пакета OpenFOAM.

Ключевые слова: ветроэнергетика; прогнозирование; моделирование ветровой обстановки; мониторинг ветропарков; ветропарк в Тикси; WRF-ARW; GFS; SOWFA

Для цитирования: Иванов А.В., Стрижак С.В., Захаров М. И. Моделирование метеоусловий в районе порта и в прибрежной зоне залива Тикси. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 163–176. DOI: 10.15514/ISPRAS–2019–31(6)–9

Modeling weather conditions in the port area and coastal zone of Tiksi Bay

[1] *A.V. Ivanov, ORCID: 0000-0002-2052-3912 <av.ivanov@ispras.ru>*
[2] *S.V. Strijhak, ORCID: 0000-0001-5525-5180 <s.strijhak@ispras.ru>*
[3] *M. I. Zakharov, ORCID: 0000-0002-8916-2166 <mplusz@inbox.ru>*

[1] *M. V. Keldysh Institute of Applied Mathematic of the Russian Academy of Sciences, 4, Miusskaya sq., Moscow, 125047, Russia*
[2] *Ivannikov Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., Moscow, 109004, Russia*
[3] *M.K. Ammosov North-Eastern federal university, 58 Belinsky st., Yakutsk, 677000, Russia*

Abstract. Wind energy is one of the most important directions in the development of renewable energy sources in the Russian Federation. The most important trends in the wind industry are objectives for the design of new wind farms, the tasks of maintenance and monitoring of wind farm design challenges of wind turbines, the development of Russian software, etc. A number of topical basic and applied problems are appeared in connection with the construction of new wind farms on the territory of the Russian Federation. In this paper, one of these tasks is considered: the study of the wind situation in the area of Tiksi village, due to the installation of a new wind farm there. The wind forecast is based on the Advanced Research WRF (Weather Research and Forecasting) mesoscale model using the Global Forecast System data. The presented results are obtained on a series of nested grids with spatial resolution of 1, 3 and 9 km. A method for wind farm study and monitoring of a near the village of Tiksi is proposed, which is based on interpolation the results obtained from WRF-ARW to a smaller scale model within the future use of Simulator fOr Wind Farm Applications (SOWFA) library in OpenFOAM platform.

Keywords: wind energy; forecasting; modeling of weather conditions; monitoring of wind farms; Tiksi wind farm; WRF-ARW; GFS; SOWFA

For citation: Ivanov A.V., Strijhak S.V., Zakharov M. I. Modeling weather conditions in the port area and coastal zone of Tiksi Bay. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 163-176 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-9

1. Введение

Задачи моделирования погоды, изучения физических процессов в атмосфере Земли и в планетарном пограничном слое являются по-прежнему актуальными [1-3]. Ежедневно в Гидрометцентр России поступает большое количество числовых данных, которые совместно с множеством спутниковых снимков характеризуют состояние атмосферы Земли. Дважды в сутки проводится обработка информации, а затем на современном суперкомпьютере считается оперативный прогноз погоды. Приближенно решается система дифференциальных уравнений в частных производных и уравнений, описывающих свойства вещества. Данная система описывает эволюцию метеорологических полей давления, скорости ветра, температуры, концентрации веществ. Данные расчета погоды по каждому региону России доступны в интернете.

Начиная с 2017 года в РФ (в Ульяновской области, Республика Адыгея, Республика Саха, в Ставропольском крае, в Ростовской и Мурманской областях) ведется строительство новых современных ветропарков. В ноябре 2018 года в арктическом поселке городского типа Тикси, который является важным транспортным узлом для Северного Морского пути, были введены в эксплуатацию 3 японских ветроэлектрические установки (ВЭУ) мощностью 0.3 МВт каждая. Успешная эксплуатация 3-х ВЭУ совместно с дизельгенератором в зимний период 2019 года показала эффективность выработки электроэнергии в сложных метеоусловиях и

целесообразность дальнейшего развития этого направления. В 2020 году в Булунском районе ветропарк альтернативных источников энергии пополнится тремя новыми японскими ветрогенераторами.

Ветроустановки дают большую экономию «Сахаэнерго». Учредители ветропарка — компании «Якутскэнерго» и «Русгидро» — стали пионерами в привлечении инвестиций в отдаленный арктический район – поселок Тикси, в котором ветровые нагрузки позволяют работать электростанции.

Пространственно организация поселка городского типа Тикси дисперсная и разделена на три микрорайона Тикси-1, Тикси-2 и Тикси-3. Всего в поселке проживает около 4600 человек. Тикси является одним из стратегически важных транспортных узлов Российского Севера, там имеется морской порт основная узловая часть инфраструктуры Северного морского пути и аэропорт федерального значения. Транспортно-экономическая значимость территории делает необходимым разработку новых подходов к рациональному природопользованию и к вопросу энергоснабжения в условиях Арктики.

Несмотря на высокую устойчивость оборудования к климатическим изменениям, для обеспечения безопасного функционирования ветроустановок и для увеличения эффективности их работы важным фактором является наличие системы мониторинга и прогноза метеорологических условий в районе ветропарка Тикси. Помимо того не изучены вопросы надежной работы ВЭУ и оценки влияния ветропарка, работающего в сложных климатических условиях, на местный микроклимат. В настоящее время по данному ветропарку отсутствуют какие-либо научные публикации.

Ранее проводились исследования по моделированию метеоусловий в ветропарках с использование модели WRF и данных с метеостанций. Одна из первых работ в этом направлении была связана с исследованием потерь мощности на ВЭУ ветропарка Horns Rev (Дания), располагающегося на морском побережье [4]. Исследование основано на наблюдениях и результатах моделирования, выполненного с использованием WRF-ARW. Конфигурация модели имела высокое горизонтальное разрешение – 333 м. В результате было показано, что мезомасштабные модели, такие как WRF-ARW, способны качественно моделировать метеоусловия, связанные с морскими ветряными электростанциями. Более подробное исследование, рассматривающее тот же ветропарк, было приведено в работе [5]. Там приводится детальное обсуждение вопросов совмещения и связи мезо- и микромасштабных моделей для моделирования ветропарков, а также обзор имеющихся исследований в этом направлении. В работе [6] рассматривается новая модель VWiS для моделирования ветропарков, которая использует данные мезомасштабной модели WRF в качестве начальных и граничных условий. В работе [7] выполняется моделирование физических параметров в атмосферном пограничном слое с помощью вихреразрешающего моделирования в сравнении с данными, полученными с 200-метровой метеовышки Cabauw в Нидерландах, с применением программы WRF.

2. Постановка задачи

Ветропарк включает в себя 3 ВЭУ, их координаты: 71°39'25.8"N 128°46'18.7"E, 71°39'19.2"N 128°45'52.6"E, 71°39'11.9"N 128°45'46.8"E. Ветротурбины ветропарка Тикси расположены на высоте порядка 120 м над уровнем моря в 4 км от поселка Тикси и 2.7 км от моря Лаптевых, рис. 1. Тип климата в данном регионе – арктический, средняя температура самого холодного месяца -39.2, абсолютный минимум -50.5 С. Перепад между среднемесячной температурой самого холодного и самого тёплого месяца составляет 42.8 градусов, [8].

Распределение направлений и скорости ветра обусловливается барическим режимом над побережьем моря Лаптевых и Арктикой в целом, рис. 2. Направление и скорость ветра у

поверхности земли также зависят от физико-географических особенностей, таких как рельеф местности и высота над уровнем моря, Прилегающая территория Тикси разделена на две основные части: Хараулахских хребет на западной прибрежной части и аккумулятивная равнина Быковского полуострова на востоке [9].

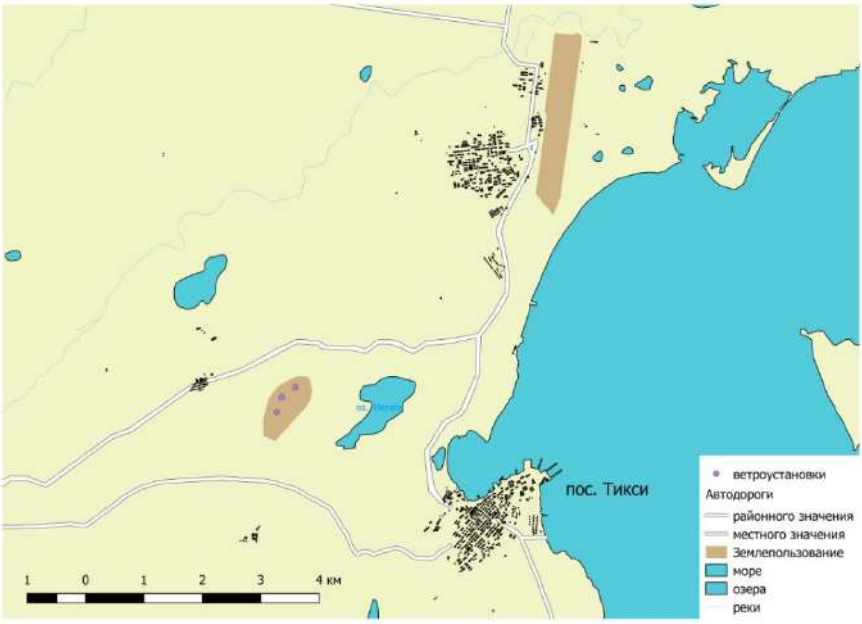

Рис. 1. Карта поселка Тикси. Open Street Map
Fig. 1. Map of Tiksi village. Open Street Map

Рис. 2. Исследуемая область поселка Тикси, снимок со спутника, Sentinel 2A, Август 2019
Fig. 2. Study zone of Tiksi, Sentinel 2A RGB image August 2019

3. Математическая модель

Предлагаемая модель, как и во многих других подобных работах, например [6] и [7], включает в себя работу с 3 различными программными продуктами. Это связано с тем, что необходимо учитывать разные характерные масштабы при моделировании. Это «макромасштаб», в котором работает глобальная модель, являющаяся первичным источником метеоданных. В качестве такой модели, как правило, выбирается GFS. Далее следует «мезомасштаб», в рамках которого производится интерполяция глобальных данных и расчет региональных областей. В роли мезомасштабной модели выступает WRF-ARW. Результатом всего моделирования является «микромасштаб», рассматривающий и исследующий область вблизи ветроустановок. В качестве такой модели предлагается использовать библиотеку SOWFA в составе открытого пакета OpenFOAM 2.4.0. Расскажем подробнее о каждой из моделей.

3.1 Модель GFS

Global Forecast System (GFS) – это глобальная система численного прогноза погоды, содержащая глобальную компьютерную модель и модели вариационного анализа, разрабатываемая в Национальном центре прогнозов окружающей среды (NCEP, США). Модель GFS представляет собой сопряженную модель, состоящую из четырех отдельных моделей (модель атмосферы, океана, суши/почвы и модель морского льда), которые работают вместе, чтобы построить точную картину погодных условий. GFS – это спектральная полулагранжева модель с полунеявной схемой интегрирования по времени, имеющая горизонтальное разрешением 13 км в течение первых 10 дней прогнозирования и 28 км с 240 до 384 часов (16 дней). По вертикали модель разделена на 64 слоя и по времени выдает прогнозируемый результат каждый час в течение первых 120 часов. Математическая модель запускается четыре раза в день и выдает прогноз на срок до 16 дней, но с уменьшенным пространственным разрешением может выдавать результат через 10 дней. Точность прогноза, как правило, уменьшается со временем (как и в любой численной модели прогноза погоды), а для более долгосрочных прогнозов значительную точность сохраняют только более крупные масштабы. Это одна из преобладающих моделей средней дальности синоптического масштаба общего назначения, также в неё регулярно вносятся изменения для повышения ее производительности и точности прогноза.

Результаты данных GFS модели доступны с сайта[1]. Файлы с данными GFS на данный момент предоставляются в открытом доступе по FTP[2]. Поля анализа GFS, а также прогностические поля этой модели могут быть использованы в качестве начальных данных для мезомасштабных моделей океана и атмосферы.

Метеоданные для текущих расчётов были взяты из Глобальной системы усвоения данных (GDAS, [10]) на основе модели GFS, подготавливаемые каждые шесть часов на сетке с разрешением в 1 географический градус.

3.2 Модель WRF-ARW

Пакет WRF-ARW разрабатывается в National Center for Atmospheric Research (NCAR, США) с середины 1990-х годов. Модель WRF-ARW базируется на негидростатических уравнениях для сжимаемой жидкости, записанных в декартовых координатах по горизонтали и с использованием орографической координаты η по вертикали, которая напоминает сигма-координату, но отличается от нее тем, что она определяется не через

[1] https://www.nco.ncep.noaa.gov/pmb/products/gfs/
[2] ftp://ftp.ncep.noaa.gov/pub/data/nccf/com/gfs/prod

полное давление p, а через его гидростатическую составляющую p_h. Пакет WRF-ARW может быть использован в целях краткосрочного прогноза опасных явлений погоды (сильных осадков и ветров, града), связанных с интенсивными мезомасштабными системами циркуляции: мезомасштабными конвективными комплексами, линиями шквалов и атмосферными фронтами.

Расчетная область модели WRF-ARW рассматривается в декартовой системе координат, сетка представляет собой параллелепипед, стороны которого ориентированы вдоль меридианов и параллелей, а основание касается сферической Земли. Для увеличения точности расчётов, вычисления проводятся на вложенных сетках, однако вложения выполняются только в горизонтальной плоскости, поэтому по вертикали область расчетов неизменна на всех сетках. Вложенные сетки ориентированы точно так же, как и основная (материнская) сетка и, наконец, шаг каждой дочерней сетки должен быть в целое число раз меньше, чем шаг родительской сетки. Имеется также возможность решения задачи на подвижной сетке.

Для корректного моделирования микро- и макропроцессов численное решение уравнений в WRF-ARW происходит с расщеплением по времени: медленные или низкочастотные моды, связанные с процессом переноса, интегрируются с использованием схемы Рунге-Кутта третьего порядка по времени (РК3), а высокочастотные акустические моды, связанные с процессом адаптации полей давления и скорости, интегрируются с меньшим шагом по времени для сохранения вычислительной устойчивости.

Программа WRF-ARW используется в качестве системы прогноза метеоусловий. В частности в Гидрометцентре России эксперименты по составлению прогнозов с помощью модели WRF-ARW начались в 2007 г. Опубликовано много работ по этой тематике, например [11] и [12]. Значительное количество работ имеется об использовании WRF-ARW за рубежом, например, [13].

Детальное описание физической модели используемой системы уравнений, а также их численного решения и параметризации физических процессов приводится в основном документе для последней, на данный момент, версии ARW v.4, [14]. Краткий обзор модели также приведен в работах Гидрометцентра России, например, в [6]

WRF-ARW – сложная вычислительная система, состоящая из трёх основных модулей: препроцессинговой системы (WPS), модели ARW и постпроцессинга (WPP). На рис. 3 изображена схема работы WRF.

Модуль WPS производит инициализацию статических геофизических данных (рельефа, типа почв, типа подстилающей поверхности и т.д.) в соответствии с параметрами, заданными в файле namelist.wps, также под эти параметры извлекаются метеоданные из базы GFS, а затем производится горизонтальная интерполяция всех полей в узлы вычислительной сетки модели ARW. Помимо статических геофизических, а также метеорологических данных, в ARW есть возможность использовать данные анализов, наблюдений и прогнозов, для этого существует система усвоения данных (WRFDA).

В модуле WRF модели производится инициализация необходимых входных параметров и параметризация физических процессов по описанным в namelist.input переменным. После этого происходит запуск ядра ARW, т.е. численное решение уравнений, описывающих заданную физическую модель.

Обработкой полученного результата занимается модуль постпроцессинга. Данный модуль является опциональным и имеет в себе множество вариантов для обработки, визуализации и интерпретации полученных расчетов.

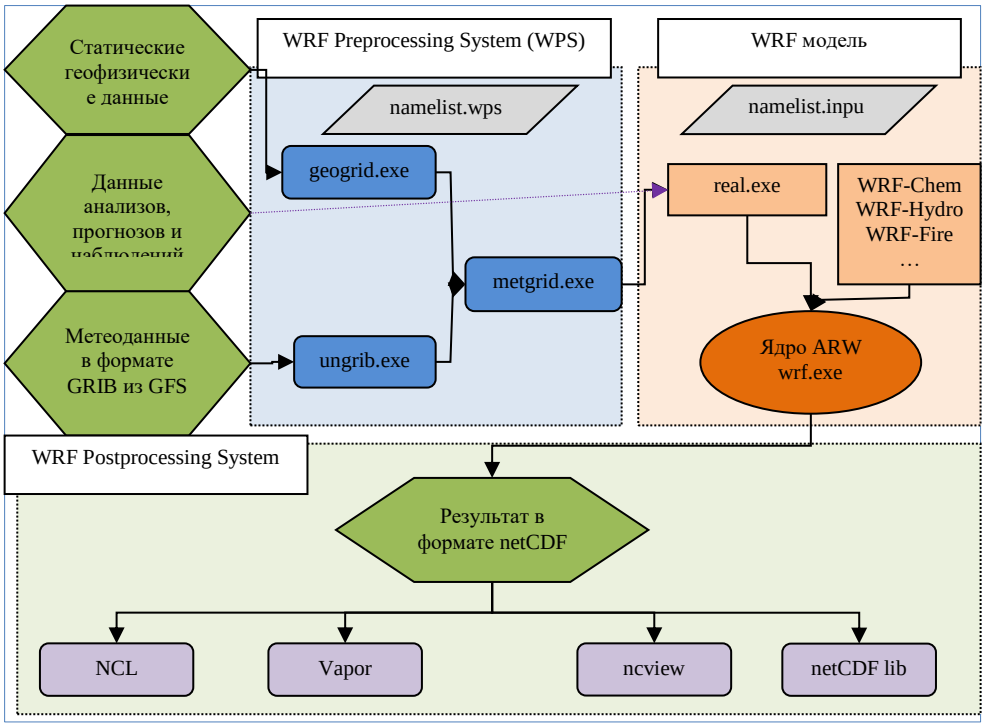

Рис. 3.Схема работы WRF-ARW
Fig. 3.WRF-ARW operation scheme

3.3 Библиотека SOWFA

Библиотека SOWFA, разработанная в Национальной лаборатория по изучению возобновляемой энергии (NREL) на базе математической модели для несжимаемых течений в открытом пакете OpenFOAM 2.4.0, является реализацией алгоритмов и численных инструментов для комплексного моделирования ветропарков. В составе библиотеки SOWFA имеется несколько решателей, в том числе для расчёта атмосферного пограничного слоя, решатель ABLSolver, и физических параметров в ветропарке, решатель windPlantSolver, с использованием метода крупных вихрей (LES). В библиотеке имеется несколько моделей для подсеточной турбулентной вязкости, специальные граничные условия для задания скорости и температуры. Модель Actuator Line Model, или модель плоских сечений аэродинамических профилей, как и модель Actuator Disk Model, могут быть использованы для расчёта течения вблизи вращающихся лопастных турбин на фиксированной расчётной сетке, что значительно экономит вычислительные ресурсы и упрощает процесс счёта. Обзор открытой библиотеки SOWFA приведен в работе [15].

3.4 Расчет метеоусловий

ФГБУ «Гидрометцентр России» использует систему прогноза ветрового волнения в Черном море с учетом данных GFS для шельфовых зон (Керченский пролив, Цемесская бухта, район Сочи) в качестве вспомогательного метода. Данные модели GFS, возможности региональной модели прогноза погоды WRF-ARW и библиотеки SOWFA могут быть использованы для разработки цифровой модели ветропарка города Тикси. Для демонстрации возможностей пакета WRF-ARW далее приводится пример модельного расчёта.

4. Использование WRF-ARW

4.1 Моделируемая область

Для модельного расчета была выбрана ограниченная область в пределах поселка Тикси. В качестве способа отображения горизонтальной плоскости использовалась проекция Меркатора. Для более точного прогнозирования внутри основной рассматриваемой области (d01) были выбраны подобласти d02 и d03, охватывающие посёлок Тикси и прилегающую акваторию, схема расположения областей изображена на рис. 4. Количество уровней по вертикали равно 33. Параметры областей приведены в табл. 1.

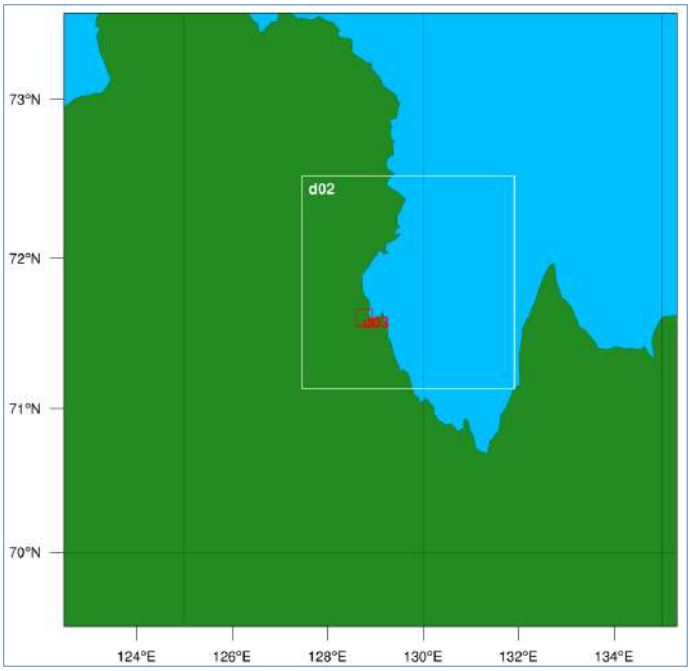

Рис. 4. Схема расположения основной области с изображением вложенных подобластей d02 и d03, выполненная с помощью NCL, [24]
Fig. 4. The layout of the main area with the nested subdomains d02 and d03, performed using NCL, [24]

Табл. 1. Параметры областей
Table 1. Zones parameters

Область	Координаты центра	Размер расчетной сетки по горизонтали	Шаг по горизонтали
Основная (d01)	71°36'N 128°54'E	40×40	9 км
Вложенная (d02)	71°50'47"N 129°41'9"E	52×52	3 км
Вложенная (d03)	71°36'50"N 128°46'8"E	13×13	1 км

Расчет проводился для 42 часов в период 29.09.2019 00:00 – 30.09.2019 18:00. В качестве параметризации физических процессов в модели была выбрана стандартная схема (CONUS), подходящая для небольших областей и имеющая следующие параметры:

- для параметризации микрофизики – схема Томпсона, [16];
- для параметризации конвекции – модифицированная схема Tiedtke, [17];
- для коротковолновой и длинноволновой радиации – модель RRTMG, [18];
- для параметризации планетарного пограничного слоя – параметризация Меллора-

Ямады-Янича, [19 - 21];

- для параметризации приземного слоя – схема Монина-Обухова-Янича, [22];
- для параметризации процессов на поверхности и почве – унифицированная модель Noah, [23].

В модели принят следующий перечень вертикальных η-уровней: 1.000, 0.997, 0.989, 0.981, 0.969, 0.956, 0.939, 0.918, 0.893, 0.863, 0.829, 0.791, 0.749, 0.705, 0.658, 0.610, 0.561, 0.512, 0.463, 0.412, 0.363, 0.314, 0.268, 0.223, 0.182, 0.144, 0.113, 0.086, 0.064, 0.045, 0.029, 0.016, 0.005, 0.000. Шаг по времени для родительской сетки – 1 минута.

4.2 Результаты расчета

Модель WRF-ARW позволяет получить большой спектр физических величин, описывающих погодные условия. В качестве основных параметров, как правило, выделяют распределение скорости ветра, температуры и давления. В качестве второстепенных параметров могут выступать влажность воздуха, распределение осадков, тип осадков и многое другое.

Для сравнения результатов была выбрана метеостанция Тикси: 71°34'48"N 128°54'E, высота над уровнем моря 7 м, метеоданные брались с сайта http://www.pogodaiklimat.ru, они продублированы в табл. 2. В ближайшей к месту расположения метеостанции точке области d03 выводились данные о температуре на уровне 2 м над поверхностью, давлении и скорости ветра на поверхности (на уровне метеостанции). Графики сравнения данных приведены на рис. 5-7.

Табл. 2. Данные метеостанции Тикси
Table 2. Tiksi weather station data

Время (UTC)	Дата	Ветер		T, °C	P_o, гПа
		Направление	Скорость, м/с		
0	29.09	ЮЗ	3	0.6	1010.2
3	29.09	З	4	1.6	1009.7
6	29.09	З	3	1.3	1009.9
9	29.09	З	3	0.1	1010
12	29.09	ЮЗ	2	-0.6	1009.9
15	29.09	З	2	-1	1009.4
18	29.09	З	3	-1	1008.8
21	29.09	ЮЗ	1	-1.8	1007.9
0	30.09	З	2	-0.7	1007.7
3	30.09	СЗ	4	0.2	1007.3
6	30.09	С	4	0.2	1007.1
9	30.09	СЗ	3	-0.8	1007.2
12	30.09	СЗ	5	-1.8	1007
15	30.09	СЗ	5	-3	1006.4
18	30.09	СЗ	8	-1.6	1006
21	30.09	СЗ	7	-2.6	1005.7

Наблюдается занижение температуры на поверхности, рис. 5. Это свойственно для модели, основной причиной тому могло стать наличие снега, а также некорректное задание статических данных местности. В целом эта проблема свойственна для WRF-ARW, её неоднократно упоминали в публикациях по моделированию метеоусловий. Одними из таких работ являются [25] и [26], в них также наблюдалось занижение температуры вплоть до 7-9 градусов Цельсия.

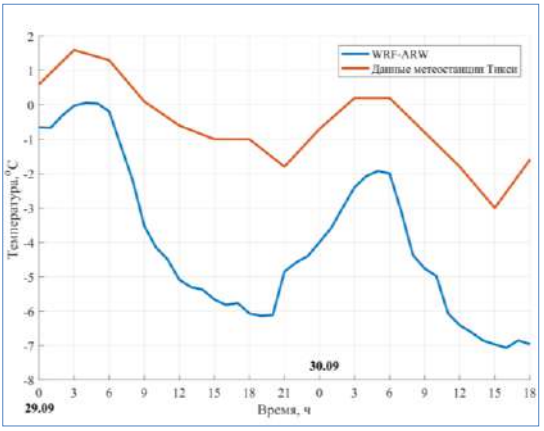

Рис. 5. Сравнение температуры в точке расположения метеостанции Тикси для модельных и реальных данных

Fig. 5. Comparison of temperature at the location of Tiksi weather station for model and real data

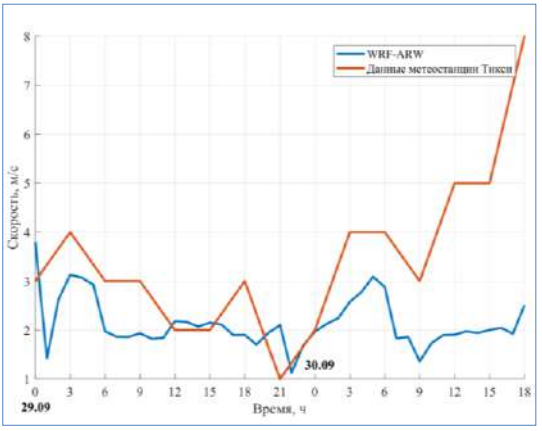

Рис. 6. Сравнение скорости ветра в точке расположения метеостанции Тикси для модельных и реальных данных

Fig. 6. Comparison of wind speed at Tiksi weather station location for model and real data

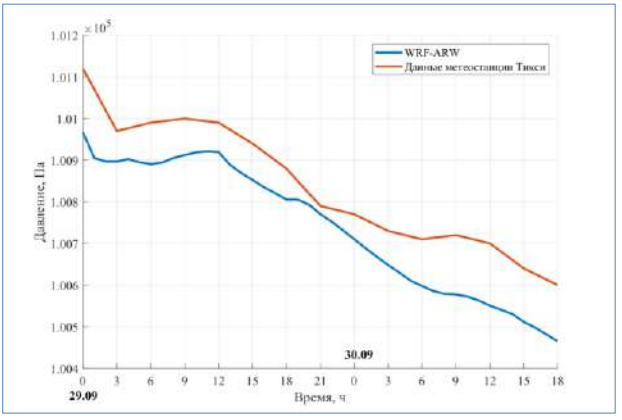

Рис. 7. Сравнение атмосферного давления в точке расположения метеостанции Тикси для модельных и реальных данных

Fig.7. Comparison of atmospheric pressure at the location of Tiksi weather station for model and real data

Помимо этого, как уже отмечалось в [6], WRF-ARW плохо воспроизводит сильные ветра, что хорошо заметно на рис. 6. Однако, направление ветра практически полностью повторяет реальные данные.

Давление также имеет незначительные отклонения от наблюдаемого, рис. 7. В целом эти отклонения не превышают 100-150 Па.

На рис. 8 и 9 приведены карты распределения ветра над рельефом местности. Опираясь на данные из табл. 2, можно заметить, что направление ветра на изображениях в точке расположения метеостанции совпадает реальным направлением ветра (направление ветра учитывается с точки зрения того, откуда дует ветер), т.е это ЮЗ для рис. 8 и СЗ для рис. 10.

В целом, результаты модели соответствуют реальным данным, во всяком случае они повторяют профиль изменения показателей.

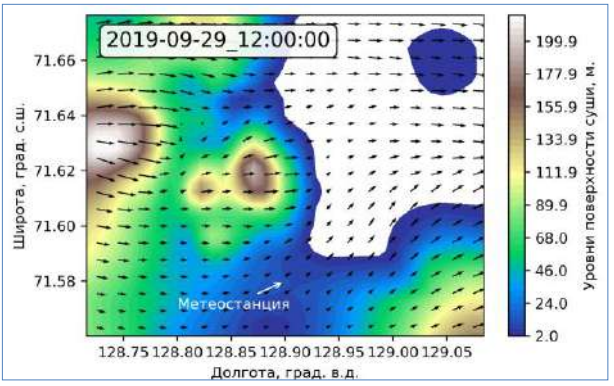

Рис. 8. Карта распределения скорости ветра над рельефом местности для 29.09 12:00
Fig. 8. A map of the wind speed distribution over terrain for 29.09 12 p.m.

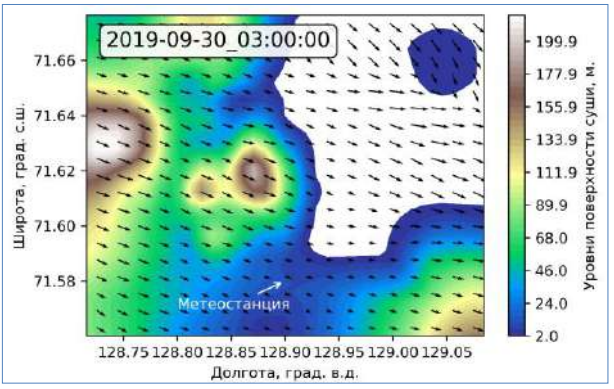

Рис. 9. Карта распределения скорости ветра над рельефом местности для 30.09 3:00
Fig. 9. A map of the wind speed distribution over terrain for 30.09 3 a.m.

5. Заключение

С применением модели ARW был получен прогноз метеоусловий в районе поселка Тикси, а именно: картина распределения скорости ветра, давления и температуры на поверхности. Данные показатели являются основными для построения модели ветропарка. Следующий этап – это интерполяция полученных значений на сетку библиотеки SOWFA пакета OpenFOAM в качестве начальных и граничных условий.

Полученные результаты не отличаются высокой точностью, однако передают приближенную картину метеоусловий в исследуемой области.

Среднеквадратичная ошибка в расчете температуры составила порядка 3-4 градуса.

Причиной расхождения данных, полученных из наблюдений на метеостанции, и результатов моделирования может быть несколько. Во-первых, остаётся под вопросом точность статических геофизических данных для северных регионов России. Именно поэтому в перспективе предполагается улучшить их качество, поскольку текущие данные являются стандартными (предоставляются путем спектрорадиометрии) и поставляются вместе с пакетом WRF-ARW.

Во-вторых, планируется более точная калибровка модели WRF-ARW, т.е. задание корректной физической параметризации для включения тех или иных методов ядра ARW. Данные значения должны подбираться в соответствии с метеоусловиями и геофизической обстановкой исследуемого района поселка Тикси.

В-третьих, необходимо добавление в модель самих ветроустановок для изучения их влияния на погоду и климат. Это может быть произведено с помощью встроенной модели, работа которой проанализирована в [27].

Помимо этого, потребуется создание сетки и моделей ветротурбин в библиотеке SOWFA и объединение мезомасштабных данных с этой моделью.

Список литератур / References

[1]. Гордин В.А. Математика, компьютер, прогноз погоды и другие сценарии математической физики. М., Физматлит, 2010 г., 736 стр. / Gordin V.A. Mathematics, computer, weather forecast and other scenarios of mathematical physics. M., Fizmatlit, 2010, 736 p. (in Russian).

[2]. Лыкосов В.Н., Глазунов А.В., Кулямин Д.В., Мортиков Е.В., Степаненко В.М. Суперкомпьютерное моделирование в физике климатической системы: учебное пособие. М., Изд.-во Московского университета, 2012 г., 408 стр. / Lykosov V.N., Glazunov A.V., Kulyamin D.V., Mortikov E.V., Stepanenko V.M. Supercomputer modeling in the physics of the climate system: a training manual. Moscow, Publishing House of Moscow University, 2012, 408 p. (in Russian).

[3]. Зилитинкевич С.С. Атмосферная турбулентность и планетарные пограничные слои. М., Физматлит, 2014 г., 252 стр. / Zilitinkevich S.S. Atmospheric turbulence and planetary boundary layers. M., Fizmatlit, 2014, 252 p. (in Russian).

[4]. Jiménez P.A., Navarro J., Palomares A.M., Dudhia J. Mesoscale modeling of offshore wind turbine wakes at the wind farm resolving scale: a composite-based analysis with the weather research and forecasting model over Horns Rev. Wind Energy, vol. 18, issue 3, 2014, pp. 559–566.

[5]. Javier Sanz Rodrigo, Roberto Aurelio Chávez Arroyo et al. Mesoscale to microscale wind farm flow modeling and evaluation. WIREs Energy Environ, vol. 6, issue 2, 2016, article no. e214

[6]. Yang Xiaolei, Sotiropoulos Fotis, Conzemius Robert, Wachtler John, Strong Mike. Large-eddy simulation of turbulent flow past wind turbines/farms: The Virtual Wind Simulator (VWiS). Wind Energy, vol. 18, issue 12, 2014, pp. 2025-2045.

[7]. Rodrigo J., Allaerts Dries et al. JMLM & Tomaszewski, Jessica & Troldborg, N & van der Laan, M. Paul & Veiga Rodrigues, Carlos. Results of the GABLS3 diurnal-cycle benchmark for wind energy applications. Journal of Physics: Conference Series, vol. 854, issue 1, 2017, article no. 012037.

[8]. Федоров А.Н., Торговкин Я.И. и др. Мерзлотно-ландшафтная карта Республики Саха (Якутия). Масштаб 1: 1 500 000. Якутск, ИМЗ СО РАН, 2018 г. / Fedorov A.N., Torgovkin Y.I. et al. The permafrost-landscape map of the Republic of Sakha (Yakutia). Scale 1: 1,500,000. Yakutsk, IIM SB RAS, 2018 (in Russian).

[9]. Periglacial features around Tiksi. Russian-German Cooperation SYSTEM LAPTEV SEA The Expedition LENA 2002. Berichte zur Polar- und Meeresforschung, vol. 466, 2003, 195 p.

[10]. NCEP FNL Operational Model Global Tropospheric Analyses, continuing from July 1999. Research Data Archive at the National Center for Atmospheric Research, Computational and Information Systems Laboratory.

[11]. Вельтищев Н. Ф., Жупанов В. Д. Численные прогнозы погоды по негидростатическим моделям общего пользования WRF-ARW и WRF-NMM. В сб. 80 лет Гидрометцентру России.1939 - 2010, 2010 г., стр. 94-135. / Vel'tishchev N.F., Zhupanov V.D., Numerical weather forecasts by non-hydrostatic open-source models WRF-ARW and WRF-NMM, In 80 years to the Hydrometeorological Center of Russia, 2010, pp. 94-135.

[12]. Лаппо П.О., Шакур В.Н., Прохареня М. Результаты верификации модели WRF-ARW в Гидромете Республики Беларусь. Труды Гидрометцентра России, вып. 358, 2015 г., стр. 67–77. / Lappo P.O., Shakur V.N, Prakharenia M. The results of verification of the WRF-ARW model in the Hydromet of the Republic of Belarus. Proceedings of Hydrometcentre of Russia, issue 258, 2015, pp. 67–77.

[13]. Yijia Zheng, Yucong Miao, Shuhua Liu, Bicheng Chen, Hui Zheng, and Shu Wang. Simulating Flow and Dispersion by Using WRF-CFD Coupled Model in a Built-Up Area of Shenyang, China. Advances in Meteorology, vol. 2015, 2015, 15 p.

[14]. Skamarock W.C., Klemp J.B. et al. A Description of the Advanced Research WRF Model Version 4. Ncar Technical Notes, No. NCAR/TN-556+STR, 2019.

[15]. Крапошин М.В., Стрижак С.В. Проблемно-ориентированная библиотека SOWFA для решения прикладных задач ветроэнергетики. Труды ИСП РАН, том 30, вып. 6, 2018 г., стр. 259–274 / Kraposhin M.V., Strijhak S.V. The problem-oriented library SOWFA for solving the applied tasks of wind energy. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 259–274 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-14.

[16]. Thompson G., Rasmussen R. M. and Manning K. Explicit forecasts of winter precipitation using an improved bulk microphysics scheme. Part I: Description and sensitivity analysis. Monthly Weather Review, vol. 132, no. 2, 2004, pp. 519-642.

[17]. Tiedtke, M. A comprehensive mass flux scheme for cumulus parameterization in largescale models. Monthly Weather Review, vol. 117, no. 8, 1989, pp. 1779-1800.

[18]. Michael J. Iacono, Jennifer S. Delamere et al. Radiative forcing by long-lived greenhouse gases: Calculations with the AER radiative transfer models. Journal of Geophysical Research, vol. 113, issue D13103. 2008.

[19]. Janjic Z.I. The step-mountain Eta coordinate model: Future developments of convection, viscous sublayer, and turbulence closure schemes. Monthly Weather Review, vol. 122, no. 5, 1990, pp. 927-945.

[20]. Janjic Z.I. The surface layer in the NCEP Eta model. In Proc. of the 11th Conference on Numerical Weather Prediction, Norfolk, 1996, pp. 354-355.

[21]. Janjic Z.I. Nonsingular implementation of the Mellor-Yamada level 2.5 scheme. In the NCEP Meso model, NCEP Office Note, no. 437, 2002, 61 p.

[22]. Монин А.С., Обухов А.М. Основные закономерности турбулентного перемешивания в приземном слое атмосферы. Труды Геофизического института АН СССР, том 151, вып. 24, 1954, стр. 163–187. // Monin, A.S. and A.M. Obukhov. Basic laws of turbulent mixing in the surface layer of the atmosphere. Contributions of the Geophysical Institute of the USSR's Academy of Sciences, vol. 24, no. 151, pp. 163–187, 1954.

[23]. Chen F. and J. Dudhia, Coupling an advanced land-surface/ hydrology model with the Penn State/ NCAR MM5 modeling system. Part I: Model description and implementation. Monthly Weather Review, vol. 129, no. 4, 2001, pp. 569–585.

[24]. The NCAR Command Language (Version 6.4.0) [Software]. Boulder, Colorado, UCAR/NCAR/CISL/VETS, 2017.

[25]. Шихов А.Н., Свиязов Е.М. Прогнозирование динамики процесса снеготаяния на Западном Урале с применением мезомасштабной модели WRF/ARW. Современные проблемы науки и образования, 2013 г., вып. 4. / Shikhov A.N., Sviyazov E.M. forecasting of the dynamics of snow melting in the Western Ural region, using WRW/ARW mesoscale model, issue 4, 2013.

[26]. Набокова Е.В. Опыт применения модели WRF с учетом двух методов параметризации городского подслоя для прогноза температуры воздуха и скорости ветра. Труды Гидрометцентра России, 2010 г., вып. 344, стр. 180-195. / Nabokova E. V. The experience of using the WRF model in respect 2 parameterization methods of urban sub-layer for wind velocity

and temperature prediction. Proceedings of the Hydrometcentre of Russia, Atmospheric physics and weather forecast, issue 344, 2010, pp. 180-195 (in Russian).

[27]. Fitch, A. C. et al. Local and Mesoscale Impacts of Wind Farms as Parameterized in a Mesoscale NWP Model. Monthly Weather Review, vol. 140, no. 9, 2012, pp. 3017-3038.

Информация об авторах / Information about authors

Александр Владимирович ИВАНОВ – аспирант ИПМ имени М. В. Келдыша с 2019 года. Сфера научных интересов: численное моделирование и решение прикладных задач в области газо- и гидродинамики.

Aleksandr Vladimirovich IVANOV – PhD student of the Keldysh Institute of Applied Mathematics since 2019. Research interests: numerical modeling and solving applied problems in the field of gas- and hydrodynamics.

Сергей Владимирович СТРИЖАК – кандидат технических наук, ведущий инженер ИСП РАН с 2009 года. Сфера научных интересов: вычислительная гидродинамика, ветроэнергетика, теория турбулентности.

Sergei Vladimirovich STRIJHAK – candidate of technical sciences, leading engineer of the ISP RAS since 2009. Research interests: computational fluid dynamics, wind energy, theory of turbulence.

Моисей Иванович ЗАХАРОВ – ассистент, аспирант СВФУ. Сфера научных интересов: геоинформационное моделирование ландшафтов, разработка мобильных ГИС-приложений.

Moisey Ivanovich ZAKHAROV – assistant, PhD student of the NEFU in Yakutsk. Research interests: geoinformation modeling of landscapes, development of mobile GIS applications.

DOI: 10.15514/ISPRAS-2019-31(6)-10

CC BY

Моделирование динамики частиц в планетарном пограничном слое и в модельном ветропарке

К.Б. Кошелев, ORCID: 0000-0002-7124-3945 <koshelevkb@mail.ru>
С.В. Стрижак, ORCID: 0000-0001-5525-5180 <s.strijhak@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В настоящее время в РФ активно ведется строительство новых ветропарков. Вопросы изучения физических процессов являются актуальными, так как действующие ветропарки оказывают влияние на микроклимат и экологию. В ветропарках возможно появление и движение жидких и твердых частиц. При исследовании двухфазных потоков, содержащих взвесь аэрозольных частиц (дисперсная фаза) в несущей среде (дисперсионная среда) в атмосфере важно правильно выбирать основные параметры, определяющие систему, и адекватно описать реальный процесс при помощи сформулированной математической модели. Работа посвящена разработке новых решателей на базе библиотеки SOWFA в составе открытого пакета OpenFOAM 2.4.0 для изучения моделирования динамики частиц в планетарном (атмосферном) пограничном слое и в модельном ветропарке. Для описания динамики частиц используется эйлер-лагранжев подход. Разработаны два новых решателя на базе ABLSolver и pisoTurbineFoam.ALM для моделирования динамики частиц. Представлены результаты расчета для случая нейтрального пограничного слоя и модельного ветропарка с 14 модельными ветроустановками. Приведены графики для распределения частиц с разным диаметром по высоте. Для расчета одного примера было использовано от 72 до 96 вычислительных ядер.

Ключевые слова: SOWFA; библиотека**;** течение; моделирование; решатель; частицы; атмосферный пограничный слой; ветропарк; скорость; вязкость; турбулентность; высота

Для цитирования: Кошелев К.Б., Стрижак С.В. Моделирование динамики частиц в планетарном пограничном слое и в модельном ветропарке. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 177–186. DOI: 10.15514/ISPRAS–2019–31(6)–10

Благодарности: Работа выполнена при финансовой поддержке РФФИ (грант № 17-07-01391).

Simulation of particle dynamics in planetary boundary layer and in a model wind farm

K.B. Koshelev, ORCID: 0000-0002-7124-3945 <koshelevkb@mail.ru>
S.V. Strijhak, ORCID: 0000-0001-5525-5180 <s.strijhak@ispras.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Currently, Russia is actively building new wind farms. The issues of studying physical processes are relevant, as the existing wind farms have an impact on the microclimate and ecology. In wind farms, the appearance and movement of liquid and solid particles is possible. In the study of two-phase flows containing a suspension of aerosol particles (dispersed phase) in the carrier medium (dispersion medium) in the atmosphere, it is important to choose the main parameters that determine the system correctly and adequately describe the real process using the formulated mathematical model. The

work is devoted to the development of new solvers based on the SOWFA library as part of the open-source OpenFOAM 2.4.0 package for the study of particle dynamics modeling in the atmospheric boundary layer and in the model wind farm. The Euler-Lagrangian approach is used to describe particle dynamics. Two new solvers based on ABLSolver and pisoTurbineFoam.ALM have been developed to simulate particle dynamics. The calculation results for the case of a neutral boundary layer and a model wind farm with 14 model wind turbines are presented. Graphs for the distribution of particles with different diameters in height are given. Between 72 and 96 cores were used to calculate one example.

Keywords: SOWFA, library, flow, simulation, solver, particles, atmospheric boundary layer, wind farm, velocity, viscosity, turbulence, altitude

For citation: Koshelev K.B., Sttrijhak S.V. Simulation of particle dynamics in planetary boundary layer and in a model wind farm. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 177-186 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-10

Acknowledgements. This work was supported by the Russian Foundation for Basic Research (Grant No. 17-07-01391).

1. Введение

В настоящее время в РФ активно ведется строительство новых ветропарков. Вопросы изучения физических процессов являются актуальными, так как действующие ветропарки оказывают влияние на микроклимат и экологию.

Изучение движения вихревых структур в атмосферном пограничном слое (АПС) с учетом влияния стратификации среды, орографии местности, вращения Земли, изменения тепловых потоков и градиента давления важно с точки зрения задач моделирования климата, изучения физических процессов при проектировании ветропарков и оценки их эффективности работы на территории РФ [1,2]. Для моделирования процессов в атмосферном пограничном слое характерно использование вихреразрешающего моделирования с использованием различных моделей для подсеточной турбулентной вязкости. Известно, что для работы ветропарков характерно условия существования нейтральной и устойчивой стратификации. Полученные результаты расчета для АПС могут быть использованы в качестве начальных данных для моделирования параметров ветропарка.

В ветропарках возможно появление и движение жидких и твердых частиц. При исследовании двухфазных потоков, содержащих взвесь аэрозольных частиц (дисперсная фаза) в несущей среде (дисперсионная среда) в атмосфере важно правильно выбирать основные параметры, определяющие систему, и адекватно описать реальный процесс при помощи сформулированной математической модели. Дисперсионной средой может быть жидкость (Ж) или газ (Г), дисперсной фазой могут быть твердые частицы (Т), капли жидкости (Ж) или газовые пузырьки (Г) [3-5].

Различные частицы, переносимые воздухом в атмосферном пограничном слое, могут быть причиной образования наростов и повреждений на горизонтальных ветроэлектрических установках (ВЭУ) в действующих ветропарках. Такие частицы как капли дождя, песчинки пыли, ледяные снежинки, градины, различные насекомые являются главными причинами загрязнения поверхности лопасти ВЭУ и снижения производительности ВЭУ во время эксплуатации [6-8].

В 2019 году на территории Республика Адыгея и в Ставропольском крае ведется строительство двух новых ветропарков. Данные регионы расположены в южных широтах, поэтому в летнее время возможно появление различных насекомых (стрекозы, жуки, бабочки, плодовые мошки и другие). Как показывает исследование, проведенное в немецком центре DLR, максимальная плотность насекомых составляет 9.0 кг/км3 на высотах 0-400 метров [9]. Скорость движения плодовых мошек может достигать 12-14 метров. В дневное время конвективные шлейфы с вертикальной скоростью ветра

(плюмы) до 5 м/с помогают всем видам насекомых достичь высоких уровней планетарного пограничного слоя. Радиолокационный анализ выявил до тысячи раз более высокую концентрацию насекомых в таких шлейфах по сравнению с окружающей средой. Большинство мелких, пассивно летающих насекомых остается в пределах таких шлейфов и добирается высот в 1000 метров или более над уровнем Земли, прежде чем они начинают перемещаться с ветром в горизонтальном направлении.

Более крупные насекомые, особенно когда они движутся ночью без влияния конвективных шлейфов, могут выбирать нижние слои для своей миграции. Для экономии энергии и поддержания сбалансированной температуры тела во время полета они могут выбрать слой с максимальной скоростью ветра и адекватной температурой воздуха. Общеизвестно, что насекомые обычно избегают летать, когда есть сильный ветер и дождь, так как при высокой турбулентности вблизи Земли полет и навигация затруднены. В этом случае насекомые предпочитают прятаться и оставаться внизу. Миграция начинается в спокойные погодные условия, когда скорость ветра на уровне Земли достаточно низкая. Однако, как только насекомые достигают поверхностного слоя, скорость перемещения может варьироваться от 20 до 100 км/ч (5 - 25 м/с). К сожалению, такое поведение приводит их непосредственно в критический диапазон высот работы роторов ветроэлектрических установок (ВЭУ).

Пространственная поверхность лопасти ВЭУ может аккумулировать грязь вокруг передней кромки. Более того, соударение частиц, резкие скачки температуры, циклы замораживания-оттаивания могут вызывать образование трещин в покрытии, приводящие к эрозии поверхности, расслоению ядра материала и коррозионному разрушению внутренней структуры композиционного материала. Гладкая первоначальная поверхность лопасти может значительно измениться по сравнению с начальным состоянием, и образовавшаяся шероховатость может значительно снизить величину вырабатываемой мощности ВЭУ.

На поверхности лопасти в дозвуковом пограничном слое могут происходить существенные изменения и образовываться локальный отрыв потока, если толщина “нароста” сравнима с толщиной пограничного слоя. Коэффициент трения существенно увеличивается в связи с ламинарно-турбулентным переход в пограничном слое. С аэродинамической точки зрения снижение аэродинамической эффективности ВЭУ связано с коэффициентом сопротивления как результат взаимодействия поверхности лопасти с окружающимися частицами. Другая важная проблема связана с увеличением уровня шума, измеряемого в Дб. Изменение структуры поверхности может повысить уровень шума в пограничном слое.

В связи с этим операторы ветропарков вынуждены проводить проверку и обслуживание лопастей ВЭУ. С технологической точки зрения компаниями разработаны специальные покрытия на основе полиэритана.

Изучение движения частиц в ветропарке также важно с точки зрения оценки дефицита скорости для ВЭУ, расположенных в разных рядах, и оценки эффекта инжекции воздушных масс [10].

Поэтому изучение динамики частиц в ветропарках и их влияния на поверхность лопастей ВЭУ с помощью математического моделирования является актуальной задачей. Нами было выбрано программное обеспечение, в основе которого используется открытая библиотека SOWFA для моделирования физических процессов в АПС и расчета параметров турбулентного течения в ветропарках [11,12].

2. Программное обеспечение

Расчёт описанной задачи произведём в пакете OpenFOAM версии 2.4.0. Используется библиотека SOWFA, предназначенная для решения моделирования течений в

атмосферном пограничном слое и в ветропарке. Данная библиотека была разработана в 2010 году в лаборатории NREL. Авторы данной статьи приняли решение продолжить разработку новых решателей в библиотеке SOWFA в OpenFOAM версии 2.4.0. Если бы использовалась последняя версии OpenFOAM 7.0, то потребовалось бы переписать значительную часть исходного кода библиотеки SOWFA. Численный метод в данной библиотеке основан на методе конечных объёмов. Математическая модель включает в себя законы сохранения массы, количества движения и энергии.

В составе библиотеки SOWFA имеется несколько проблемно-ориентированных решателей, в том числе для расчёта атмосферного пограничного слоя (решатель ABLSolver) и физических параметров в ветропарке (решатель windPlantSolver) с использованием метода крупных вихрей и динамической лагранжевой модели Смагоринского [13,14]. В составе библиотеки имеется несколько моделей для подсеточной турбулентной вязкости и специальные граничные условия для задания скорости и температуры. Модель плоских сечений профиля (Actuator Line Model - ALM) может быть использована для расчёта течения вблизи вращающихся лопастных турбин на фиксированной расчётной сетке, что значительно экономит вычислительные ресурсы и упрощает процесс счёта. Параллельные вычисления могут выполняться с использованием стандартного метода декомпозиции расчётной области.

3. Математическая модель

Решатель ABLSolver предназначен для моделирования атмосферного пограничного слоя (АПС) в случае плоской поверхности земли при наличии температурного градиента (стабильный, нестабильный, нейтральный АПС). Математическая модель включает в себя уравнения (1) – (8).

$$\frac{\partial \bar{u}_j}{\partial x_j} = 0 \#(1)$$

$$\frac{\partial \bar{u}_i}{\partial t} = -\frac{\partial}{\partial x_j}\left(\bar{u}_j \bar{u}_i\right) - \frac{\partial R_{ij}^D}{\partial x_j} - \frac{\partial \tilde{p}}{\partial x_i} - \left(\frac{\partial \tilde{p}}{\partial x_i}\right)^d + \left(1 - \frac{\bar{\theta}}{\bar{\theta}^0}\right) g_i + \epsilon_{ij} f^c \bar{u}_j \#(2)$$

$$\frac{\partial \bar{\theta}}{\partial t} = -\frac{\partial}{\partial x_j}\left(\bar{u}_j \bar{\theta}\right) - \frac{\partial R_{\theta j}}{\partial x_j} \#(3)$$

$$R_{ij}^D = -2\nu^{SGS} \bar{S}_{ij} \#(4)$$

$$\bar{S}_{ij} = \frac{1}{2}\left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i}\right) \#(5)$$

$$\nu^{SGS} = (C_s \Delta)^2 \left(2\bar{S}_{ij}\bar{S}_{ij}\right)^{\frac{1}{2}} \#(6)$$

$$R_{\theta j} = -\frac{\nu^{SGS}}{Pr_t}\frac{\partial \bar{\theta}}{\partial x_j} \#(7)$$

$$\frac{\partial R_{ij}^D}{\partial x_j} = -\frac{\partial}{\partial x_j}\left(\nu^{SGS}\frac{\partial \bar{u}_i}{\partial x_j}\right) - \frac{\partial}{\partial x_j}\left[\nu^{SGS}\left(\frac{\partial \bar{u}_j}{\partial x_i} - \frac{2}{3}\frac{\partial \bar{u}_k}{\partial x_k}\delta_{ij}\right)\right], \#(8)$$

где $\bar{u}_j$ – скорость, $\tilde{p}$ - давление, $R_{ij}^D = R_{ij} - R_{kk}\delta/3$ – тензор напряжения, $\left(\frac{\partial \tilde{p}}{\partial x_i}\right)^d$ – градиент давления, $\bar{\theta}$ – потенциальная температура, f^c – параметр Кориолиса.

Для описания динамики частиц используется эйлер-лагранжев подход. Модификация решателя ABLSolver заключалась в добавлении модели кинематического облака частиц [15]. В данной реализации решателя были добавлены слагаемые, учитывающих движение частиц и их взаимодействие с воздушным потоком. Тепловое взаимодействие пока не учитывалось.

Фактически решатель ABLSolverP был разработан путем добавления к решателю ABLSolver следующих строк (выделены жирным шрифтом):

```
Модуль UEqn.H:
// Solve the momentum equation
    #include "computeCoriolisForce.H"
    #include "computeBuoyancyTerm.H"
fvVectorMatrixUEqn
    (
fvm::ddt(U)                  // time derivative
        + fvm::div(phi, U)   // convection
        + turbulence->divDevReff(U)   // momentum flux
        + fvc::div(Rwall)
        - fCoriolis                  // Coriolis force
        - SourceU                     // mesoscale source terms
- prho1 * kinematicCloud.SU(U)        // momentum form particles
    );
UEqn.relax();
```

Также были внесены ссылки на необходимые модули в файлы проекта.

Решатель pisoTurbineFoam.ALM предназначен для моделирования течения в ветропарке с использованием метода ActuatorLineModel. Его модификация также заключалась в добавлении кинематического облака частиц.

Дополнительно был разработан новый решатель partpisoTurbineFoam.ALM для моделирования облака частиц в ветропарке. Решатель partpisoTurbineFoam.ALM был разработан путем добавления к решателю pisoTurbineFoam.ALM следующих строк (выделены жирным шрифтом):

```
        // PISO algorithm
        {
            // Momentum predictor
fvVectorMatrixUEqn
            (
fvm::ddt(U)
        + fvm::div(phi, U)
        + turbulence->divDevReff(U)
        - turbines.force()
        - rrho1*kinematicCloud.SU(U)    // momentum form particles
            );
UEqn.relax();
```

4. Объекты исследования

4.1 Атмосферный пограничный слой

Известно, что для атмосферного пограничного слоя характерно наличие крупных вихревых структур, которые оказывают влияние на развитие турбулентности у поверхности [16-18]. Для изучения физических процессов с помощью вихреразрешающего моделирования обычно используют прямоугольную расчетную область [18].

Задача об изучение образования кластеров аэрозолей с учетом влияния влажности, тепловых потоков в атмосферном пограничном слое с использованием данных метеонаблюдений является актуальной задачей в связи изучением микроклимата и оценки уровня солнечной радиации, достигающей поверхности Земли [19, 20].

Будем рассматривать следующую постановку задачи: имеется область в форме параллелепипеда с заданными размерами 3000 х 1000 х 3000 метров. Исследуются

физические параметры в нейтральном пограничном слое [18, 21,22]. На входе в расчетную область задается профиль скорости и вводится облако твердых частиц.

Для задания тестового примера теперь надо иметь, по крайней мере один дополнительный файл в каталоге constant – Cloud1properties для описания характеристик частиц. Секция interpolationSchemes должна содержать схемы для переменных rhok, U, nuSGS. В качестве частиц рассматриваются частицы песка. Типичная плотность песка равна 2550 кг/м3. В нижеприведенном примере модель ввода частиц (парселей) - модель сечения. В расчетную область вводится 100 кг частиц за 10 секунд со скоростью ввода 10 частиц (парселей) в секунду и собственной скоростью 8 м/с. Диаметр частиц задается фиксированным $2.5 \cdot 10^{-4}$ метра. Ввод частиц осуществлялся через входное сечение для скорости газа. На выходной и боковых границах задавалось циклические граничные условия. Таким образом введенные частицы не покидали расчетную область.

Шаг по времени для данной задачи в эйлерово-лагранжевой постановке пришлось уменьшить в 5 раз по сравнению с аналогичной задачей без частиц. Положение частиц (парселей) при «установившемся» режиме течения приведено на рис. 1. Оказалось, что частицы под влиянием конкурирующих сил тяжести и турбулентной диффузии занимают примерно нижнюю треть пространственной области.

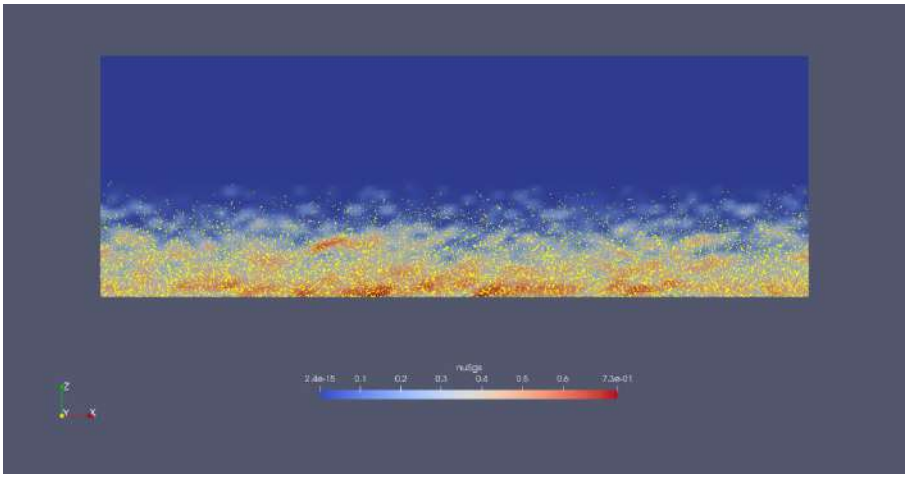

Рис.1. Положение частиц при «установившемся» течении на фоне турбулентной вязкости
Fig. 1. Parcel position in a "steady" flow against a background of turbulent viscosity

4.2 Модельный Ветропарк

Рассматривается модельный ветропарк с движением твердых частиц [23]. Пространственная область в данном примере имеет размеры 1м х 5м х 9м. В модельном ветропарке расположено 14 модельных ветроэлектрических установок (ВЭУ). Расположение имитаторов ветроустановок соответствует расположению 14 ВЭУ в ветропарке, расположенном в Ульяновской области РФ. Геометрические данные для имитаторов ВЭУ взяты из работ [21,23]. Расчет течения проводился с использованием URANS модели и k-ε модели турбулентности.

Валидация решателя без частиц pisoTurbineFoam.ALM ранее проводилась на модельных задачах с 2, 12, 14 имитаторами ВЭУ в нейтральном атмосферном пограничном слое [24-28]. Было получено хорошее совпадение с данными эксперимента в климатических трубах, ошибка при вычислении значений коэффициента мощности Cp для ВЭУ составила менее 5 %.

Для задания тестового примера надо иметь, по крайней мере два дополнительных файла в каталоге constant – стандартный g и kinematicCloud1properties. Секция interpolationSchemes должна содержать схемы для переменных rhok, U, nut (при использовании RANS k-e модели турбулентности). В нижеприведенном примере модель ввода частиц (парселей) - модель сечения. Вводится масса частиц m=0.0006 кг со скоростью ввода 1011 частиц (парселей) в секунду и собственной скоростью 1.5 м/с. Диаметр частиц фиксированный и равен 10^{-5} м. Ввод частиц осуществлялся через входное сечение для скорости газа. На боковых границах задавалось циклические граничные условия. Процесс вычислений полагался установившимся, когда за 1 с модельного времени количество введенных и покинувших область частиц совпадало.

5. Результаты расчета

Положение частиц (парселей) при «установившемся» режиме течения приведено на рис. 1. Оказалось, что частицы вод влиянием конкурирующих сил тяжести и турбулентной диффузии занимают примерно нижнюю треть пространственной области. На рис. 2 показано положение частиц в ветропарке в момент времени t=7.4 секунда.

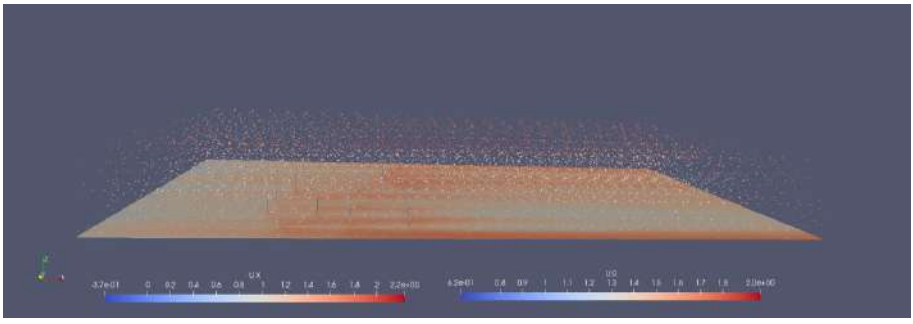

Рис. 2. Положение парселей в ветропарке в момент времени t=7.4 секунд
Fig. 2. The position of the parcels in the wind farm at time t = 7.4 seconds

На рис. 3 для примера с атмосферным пограничным слоем красным цветом выделено распределение частиц по высоте с диаметром 25 мкм, а синим цветом - с диаметром 50 мкм. Как видно особой разницы в распределение частиц по высоте нет, так как частиц относительно мало и на турбулентность они не влияют. Там, где заканчивается влияние турбулентности (выше примерно чуть более 400 м) все частицы под влиянием силы тяжести опускаются ниже. Впрочем, влияние силы тяжести наблюдается на всех высотах.

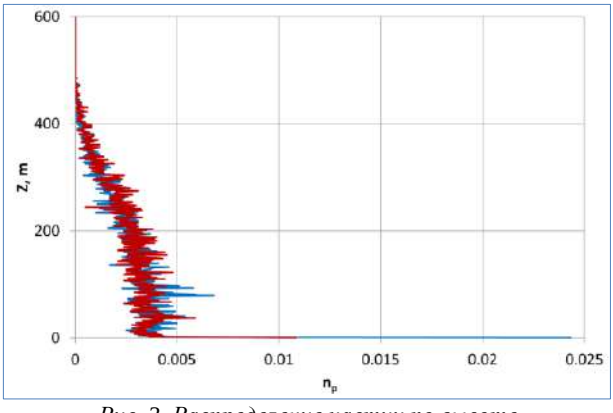

Рис. 3. Распределение частиц по высоте
Fig. 3. Parcel distribution by height

На Рисунке 4 для случая модельного ветропарка [28] показано распределение частиц по высоте с диаметром 10 мкм. Как видно, картина в корне отличается от реального примера с атмосферным пограничным слоем. Влияние ВЭУ и силы тяжести распознать достаточно сложно. В дальнейшем требуется проведение дополнительных исследований для изучения данного вопроса.

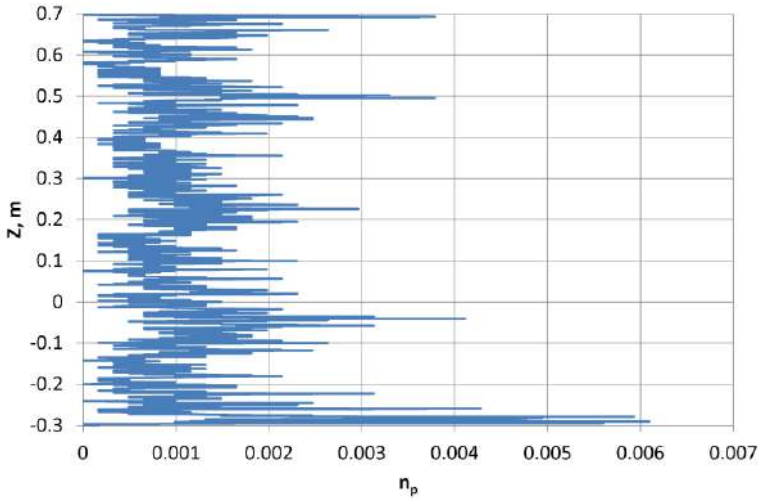

Рис. 4. Распределение частиц по высоте в ветропарке
Fig. 4. Parcel distribution by height in the wind farm

6. Заключение

В статье рассмотрены возможности библиотеки SOWFA для решения прикладных задач механики сплошной среды в области ветроэнергетики. Исследование процессов турбулентного движения в атмосферном пограничном слое и в модельном ветропарке предложено осуществлять с применением средств отслеживания облака частиц. В статье приведен пример добавления модели облака частиц в состав решателя ABLSolver и pisoTurbineFoam.ALM. Разработаны два новых решателя для моделирования динамики части в составе библиотеки SOWFA. Для демонстрации работы новых решателей представлены результаты расчета поля турбулентной вязкости, для модельного ветропарка с 14 ВЭУ. Вычисления были проведены с использованием ресурсов вычислительного кластера web-лаборатории UniCFD ИСП РАН. Для расчета одного примера было использовано от 72 до 96 вычислительных ядер.

Список литературы / References

[1]. Лыкосов В.Н., Глазунов А.В., Кулямин Д.В., Мортиков Е.В., Степаненко В.М. Суперкомпьютерное моделирование в физике климатической системы: учебное пособие. М., Изд.-во Московского университета, 2012 г., 408 стр. / Lykosov V.N., Glazunov A.V., Kulyamin D.V., Mortikov E.V., Stepanenko V.M. Supercomputer modeling in the physics of the climate system: a training manual. M., Publishing House of Moscow University, 2012, 408 p. (in Russian).

[2]. Зилитинкевич С.С. Атмосферная турбулентность и планетарные пограничные слои. М., Физматлит, 2014 г., 252 стр. / Zilitinkevich S.S. Atmospheric turbulence and planetary boundary layers. M., Fizmatlit, 2014, 252 p. (in Russian).

[3]. Алоян А.Е. Динамика и кинетика газовых примесей и аэрозолей в атмосфере. Курс лекций. М., ИВМ РАН, 2002 г., 201 стр. / Aloyan A.E. Dynamics and kinetics of gas impurities and aerosols in the atmosphere. Lecture course. M., INM RAS, 2002, 201 p. (in Russian).

[4]. Береснев С.А., Грязин В.И. Физика атмосферных аэрозолей: Курс лекций. Екатеринбург, Изд-во Урал. ун-та, 2008 г., 227 стр. / Beresnev S.A., Gryazin V.I. Physics of Atmospheric Aerosols: Lecture Course. Yekaterinburg, Publishing House of Ural University, 2008, 227 p. (in Russian).
[5]. Архипов В.А., Усанина А.С. Движение аэрозольных частиц в потоке: учебное пособие. Томск, Издательский дом Томского государственного университета, 2013 г., 92 стр. / Arkhipov V.A., Usanina A.S. The movement of aerosol particles in a stream: a training manual. Tomsk, Publishing House of Tomsk State University, 2013, 92 p. (in Russian).
[6]. Fiore G., Selig M. S. A Simulation of Operational Damage for Wind Turbines. In Proc. of the 32nd AIAA Applied Aerodynamics Conference, 2014. pp. 1-19.
[7]. Fiore G., Fujiwara G.E.C., Selig M.S. A Damage Assessment for Wind Turbine Blades from Heavy Atmospheric Particles. In Proc. of the 53rd AIAA Aerospace Sciences Meeting, 2015, pp. 1-22.
[8]. Sareen A., Sapre C.A., Selig M.S. Effects of leading edge erosion on wind turbine blade performance. Wind Energy, vol. 17, issue 10, 2014, pp. 1531–1542.
[9]. Trieb F. Interference of Flying Insects and Wind Parks. Study Report. Deutsches Zentrum für Luft- und Raumfahrt (DLR), 2018, 30 p.
[10]. Breton S-P, Sumner J., Sorensen J.N., Hansen K.S., Sarmast S., Ivanell S. A survey of modeling methods for high-fidelity wind farm simulations using large eddy simulation. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, vol. 375, issue 2091, 2017, article no. A 375:20160097.
[11]. Churchfield M.J., Moriarty P.J., Vijayakumar G., Brasseur J.G. Wind Energy-Related Atmospheric Boundary Layer Large-Eddy Simulation Using OpenFOAM. In Proc. of the 19th Symposium on Boundary Layers and Turbulence, 2010, 23 p.
[12]. Churchfield M.J., Lee. S., Michalakes J., Moriarty P.J. A numerical study of the effects of atmospheric and wake turbulence on wind turbine dynamics. Journal of Turbulence, vol. 13, 2012, article no. 13.
[13]. Sagaut P. Large eddy simulation for incompressible flows: an introduction. Springer, Berlin, 2001, 319 p.
[14]. Meneveau C, Lund T S, Cabot W H. A Lagrangian dynamic subgrid-scale model of turbulence. Journal of Fluid Mechanics, vol. 319, 1996, pp. 353–385.
[15]. Крапошин М.В., Стрижак С.В. Проблемно-ориентированная библиотека SOWFA для решения прикладных задач ветроэнергетики. Труды ИСП РАН, том 30, вып. 6, 2018 г., стр. 259-274 / M.V. Kraposhin, S.V. Strijhak The problem-oriented library SOWFA for solving the applied tasks of wind energy. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 6, 2018, pp. 259-274 (in Russian). DOI: 10.15514/ISPRAS-2018-30(6)-14.
[16]. Zilitinkevich S S, Hunt J C R., Grachev A A, Esau I N, et al. The influence of large convective eddies on the surface layer turbulence. // Quart. J. Roy. Meteorol. Soc., 2006. 132 1423- 1456.
[17]. Huang J., Cassiani M., Albertson J. D. Analysis of coherent structures within the atmospheric boundary layer. // Boundary-Layer Meteorology. 2009, 131 147–171.
[18]. Shah S., Bou-Zeid E. Very-Large-Scale Motions in the Atmospheric Boundary Layer Educed by Snapshot Proper Orthogonal Decomposition. Boundary-Layer Meteorology, vol. 153, issue 3, 2014, 355-387.
[19]. Yu H., Liu S.C., Dickinson R.E. Radiative effects of aerosols on the evolution of the atmospheric boundary layer. Journal of Geophysical Research, volume 107, issue D12, 2002, pp. AAC 3-1 – AAC 3-14.
[20]. Lauros J., Nilsson E.D., Dal Maso M., Kulmala M. Contribution of mixing in the ABL to new particle formation based on observations. Atmospheric Chemistry and Physics, vol. 7, 2007, pp. 4781–4792.
[21]. Hancock P.E., Farr T.D. Wind-tunnel simulations of wind-turbine arrays on neutral and non-neutral winds. Journal of Physics: Conference Series, vol. 524. 2014, article no. 012166.
[22]. Tellez-Alvarez J., Koshelev K., Strijhak S., Redondo J.M. Simulation of turbulence mixing in the atmosphere boundary layer and analysis of fractal dimension. Physica Scripta, vol. 94, no. 6, 2019, article no. 064004.
[23]. Hancock P.E., Pascheke F. Wind-Tunnel Simulation of the Wake of a Large Wind Turbine in a Stable Boundary Layer: Part 2, the Wake Flow. Boundary-Layer Meteorology, vol. 151, 2014, pp. 23–37.
[24]. Крючкова *А.С.,* Стрижак *С. В.* Моделирование вихревого следа для случая двух модельных ветроустановок. Научный вестник МГТУ, том 21, № 1, 2018 г., стр. 40-48 / Kryuchkova A.S.,

Strijhak S.V. Modelling of turbulent wake for two wind turbines. Civil Aviation High Technologies, vol. 21, № 1, 2018, pp. 40-48 (in Russian.)

[25]. Kryuchkova A., Tellez-Alvarez J., Strijhak S., Redondo J.M. Assessment of Turbulent Wake Behind Two Wind Turbines Using Multi-Fractal Analysis. Ivannikov ISPRAS Open Conference, 2017, pp. 110-116.

[26]. Strijhak S.V., Koshelev K.B., Kryuchkova A.S. Studying parameters of turbulent wakes for model wind turbines. AIP Conference Proceedings, vol. 2027, issue 1, 2018, article no. 030086.

[27]. Tellez-Alvarez J., Strijhak S., Kharchi R., Kryuchkova A., Redondo J.M. 3D Numerical Simulation of Wind Turbines and Fractal Dimension Analysis. In Proc. of the 2018 International Conference on Wind Energy and Applications in Algeria, 2018, 5 p.

[28]. Strijhak S.V., Koshelev K.B., Kryuchkova A.S. Simulation of turbulent wakes in model wind farm with arbitrary location for wind turbines. Journal of Physics: Conference Series, vol. 1382, 2019, article no. 012043.

Информация об авторах / Information about authors

Константин Борисович КОШЕЛЕВ – кандидат физико-математических наук, доцент, старший научный сотрудник ИСП РАН с 2016 года. Сфера научных интересов: вычислительная гидродинамика.

Konstantin Borisovich KOSHELEV is candidate of physical and mathematical sciences, associate professor, senior researcher at the ISP RAS since 2016. Research interests: computational fluid dynamics.

Сергей Владимирович СТРИЖАК – кандидат технических наук, ведущий инженер ИСП РАН с 2009 года. Сфера научных интересов: вычислительная гидродинамика.

Sergei Vladimirovich STRIJHAK - candidate of technical sciences, leading engineer of the ISP RAS since 2009. Research interests: computational fluid dynamics.

DOI: 10.15514/ISPRAS-2019-31(6)-11

CC BY

The effect of numerical dissipation on the predictive accuracy of wall-modelled large-eddy simulation

T. Mukha, ORCID: 0000-0002-2195-8408 <timofey@chalmers.se>

Chalmers University of Technology, Department of Mechanics and Maritime Sciences, Hörselgången 4, SE-412 96 Gothenburg, Sweden

Abstract. The effect of numerical dissipation on the predictive accuracy of wall-modelled large-eddy simulation is investigated via systematic simulations of fully-developed turbulent channel flow. A total of 16 simulations are conducted using the open-source computational fluid dynamics software OpenFOAM®. Four densities of the computational mesh are considered, with four simulations performed on each, in turn varying in the amount of numerical dissipation introduced by the numerical scheme used for interpolating the convective fluxes. The results are compared to publicly-available data from direct numerical simulation of the same flow. Computed error profiles of all the considered flow quantities are shown to vary monotonically with the amount of dissipation introduced by the numerical schemes. As expected, increased dissipation leads to damping of high-frequency motions, which is clearly observed in the computed energy spectra. But it also results in increased energy of the large-scale motions, and a significant over-prediction of the turbulent kinetic energy in the inner region of the boundary layer. On the other hand, dissipation benefits the accuracy of the mean velocity profile, which in turn improves the prediction of the wall shear stress given by the wall model. Thus, in the current framework, the optimal choice for the dissipation of the numerical schemes may depend on the primary quantity of interest for the conducted simulation. With respect to the resolution of the grid, the change in the accuracy is much less predictable, and the optimal resolution depends on the considered quantity and the amount of dissipation introduced by the numerical schemes.

Keywords: large-eddy simulation; turbulence; wall modelling; numerical dissipation; channel flow.

For citation: Mukha T. Effects of numerical dissipation on the predictive accuracy of wall-modelled large-eddy simulation. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 187-194. DOI: 10.15514/ISPRAS-2019-31(6)-11

Acknowledgements. The author would like to thank Saleh Rezaeiravesh from the Royal Institute of Technology in Stockholm for fruitful discussions and providing useful comments on the draft of the manuscript. The simulations were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC).

Влияние численной диссипации на расчетную точность метода моделирования крупных вихрей с пристенным моделированием

Т.Д. Муха, ORCID: 0000-0002-2195-8408 <timofey@chalmers.se>

Технологический университет Чалмерса, факультет механики и морских наук, Швеция, Гетеборг, SE-412 96, Hörselgången 4

Аннотация. В данной работе систематически исследуется влияние численной диссипации на расчетную точность метода моделирования крупных вихрей с пристенным моделированием. С этой целью в свободном программном обеспечении OpenFOAM® было проведено шестнадцать расчетов развитого турбулентного течения в канале. Расчеты проведены на сетках четырех разных плотностей, по четыре расчета на каждой сетке, в каждом из которых, в свою очередь, установлен разный уровень численной диссипации посредством изменения интерполяционной схемы для конвективного переноса. Проведено сравнение результатов расчетов с данными прямого численного моделирования, находящимися в открытом доступе. Показано, что профили ошибки всех рассмотренных величин находятся в монотонной зависимости от объема численной диссипации. Диссипация предсказуемо приводит к подавлению высокочастотных флуктуаций скорости. Кроме этого, она также приводит к увеличению энергии крупномасштабных флуктуаций и существенной переоценке уровня кинетической энергии турбулентности во внутреннем слое. Однако повышенный уровень диссипации приводит и к улучшению точности расчета средней скорости течения, что, в свою очередь, обеспечивает более точную оценку касательного напряжения на стенке пристенной моделью. Таким образом, оптимальный уровень диссипации может зависеть от основной цели расчета. Эффект плотности расчетной сетки на точность расчета трудно предсказуем, и оптимальное значение плотности зависит как от рассматриваемой физической величины, так и от уровня диссипативности интерполяционных схем.

Ключевые слова: метод моделирования крупных вихрей; турбулентность; пристенное моделирование; численная диссипация; течение в канале

Для цитирования: Муха Т.Д. Влияние численной диссипации на расчетную точность метода моделирования крупных вихрей с пристенным моделированием. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 187-194 (на английском языке). DOI: 10.15514/ISPRAS–2019–31(6)–11

Благодарности. Автор выражает благодарность Салеху Резаеиравешу из Королевского технологического института в Стокгольме за плодотворное обсуждение и полезные комментарии к черновику статьи. Вычислительные мощности для проведения расчетов были предоставлены Шведской национальной вычислительной инфраструктурой (SNIC).

1. Introduction

In turbulent boundary layers (TBLs), two fundamental length-scales can be distinguished. One is the thickness of the boundary layer, δ, which governs the size of the largest eddies in the TBL. The other is the viscous length-scale, δ_ν, defining the size of the small eddies present in the inner region of the TBL. In standard, or wall-resolving, large-eddy simulation (LES) both scales are resolved, leading to an accurate solution, but forcing the size of the computational mesh to scale as $Re_\tau^{1.85}$, where $Re_\tau = \delta/\delta_\nu$ is the friction-based Reynolds number. In wall-modelled LES (WMLES) a model for the scales $\sim\delta_\nu$ is employed, allowing to use a mesh that only resolves the large scales $\sim\delta$. This reduces the mesh size scaling with Re_τ to linear, making higher Re_τ-number simulations affordable [1]–[3].

Several approaches to WMLES exist, the reader interested in a detailed review is referred to [4]–[7]. Most commonly, so-called wall-stress modelling is used, in which the task of the wall-model is to predict the wall shear stress τ_w, given the current solution to the LES equations, typically sampled from a single point located at some distance h from the considered location on the wall.

The predictive accuracy of WMLES has been assessed in numerous works, using both in-house codes and publicly available solvers, such as OpenFOAM. Most of the studies consider a single set of modelling choices (subgrid scale model, numerical methods, etc.) and focus on evaluating or improving the performance of the wall model, see, for example, [8]–[15]. However, a recent study [16] has shown that the other parameters of the simulation affect the accuracy of WMLES at least as much as the wall modelling. In particular, the role of numerical dissipation in the simulation was highlighted, with more dissipative schemes and subgrid scale models, somewhat surprisingly, leading to more accurate results for certain flow quantities. The goal of this work is to further analyze the role of numerical dissipation via a set of simulations in which the resolution of the computational mesh and the dissipative contribution of the numerical scheme used for interpolating convective fluxes are systematically varied. Both temporal statistics of the quantities of interests and energy spectra are considered, contributing to a more holistic picture of how numerical dissipation affects the solution.

2. Methods of computational fluid dynamics

The governing equations for LES are obtained by applying a spatial filter to the Navier-Stokes equations, see [17] for details. The resulting system of PDEs must be complemented with a model for the subgrid stresses. In this work, the WALE model [18] is used to that end.

To solve the governing equations, the open-source CFD software OpenFOAM v1806 is employed. OpenFOAM uses the finite volume method to discretize the governing equations, supports arbitrary convex polyhedral cells, and offers a rich selection of numerical schemes [19]–[21]. For wall modelling, an additional publicly available library is used to enhance OpenFOAM's built-in capabilities [22]. A particularly important improvement (see [23]) is that the library allows sampling the LES solution used as the input to the wall model from an arbitrary distance from the wall, and not only from the wall-adjacent cell.

In scale-resolving simulations, it is common practice to employ at least second-order accurate numerical schemes. Here, a fully implicit second-order accurate backward differencing scheme is used to integrate the equations in time, and linear interpolation is used to compute the diffusive cell-face fluxes. The choice of the scheme for interpolating the convective fluxes is used as a controller of the amount numerical dissipation in the simulation. To that end, a linear blending of two second-order schemes is considered: second-order upwind and linear interpolation. The former scheme is dissipative, and its weight, expressed in percent, will from here-on be referred to as "the amount of upwinding" for simplicity. In OpenFOAM, the scheme using 25% upwinding is referred to as LUST (linear upwind stabilized transport). WMLES comparing LUST and linear interpolation with no upwinding have been conducted in previous studies [16], [22], with more favorable results for first-order statistics achieved using LUST. Here, the cases of 15% and 5% upwinding are additionally considered to get a clearer picture of how upwinding affects the results.

An algebraic wall model based on Spalding's law [24] is employed.

3. Case set-up

Fully-developed turbulent channel flow at $Re_b = 125000$ is considered, where $Re_b = U_b\delta/\nu$, with $U_b = 1$ m/s denoting the bulk velocity, $\delta = 1$ m the channel half-height, and ν the kinematic viscosity. These values correspond to those used in the direct numerical simulation (DNS) of Lee and Moser [25], which is used here as reference data, with respect to which the relative errors in the simulation results are computed. The computational domain is a box of size $9\delta \times 2\delta \times 4\delta$ in the streamwise, wall-normal, and spanwise direction, respectively. The value of U_b is enforced at each time-step by a time-varying source term in the streamwise momentum equation.

The domain is discretized with cubic cells of equal size. The mesh is thus fully defined by the number of cells discretizing the channel half-height, denoted n/δ. Four meshes are considered, with n/δ equal to 15, 20, 25, and 30. For each mesh, four simulations are performed, corresponding to a different amount of upwinding employed in the convective flux interpolation scheme: 25%, 15%, 5%, and 0%, respectively.

The location of the sampling point of the wall model is set to $\approx 0.1\delta$ in each simulation. The approximation sign is necessary because 0.1δ does not coincide with a cell center for all the four considered mesh resolutions. However, the accuracy of the law of the wall is quite robust with respect to change in h, therefore comparing results across different mesh resolutions should still be possible.

4. Results

An analysis of the performance of the wall model is given first. The solid lines in fig. 1 show the relative error in the predictions of the mean friction velocity, $\langle u_\tau \rangle = \sqrt{\langle \tau_w \rangle / \rho}$. Both plots present the same data, but with a different quantity used as the abscissa. In the left plot, a trend towards underpredicting $\langle u_\tau \rangle$ with decreasing amount of upwinding is clearly observed. In the right plot, a similar trend is observed with respect to increasing the resolution of the mesh. The dashed lines in Fig. 1 show the relative error in the mean velocity at the location of the sampling point. The magnitude and behavior of this error are very similar to that exhibited by $\langle u_\tau \rangle$. This confirms the observation made in a previous study [16] that in the case of attached boundary layers the accuracy of the wall model is chiefly determined by the accuracy of the velocity signal that it bases its predictions on.

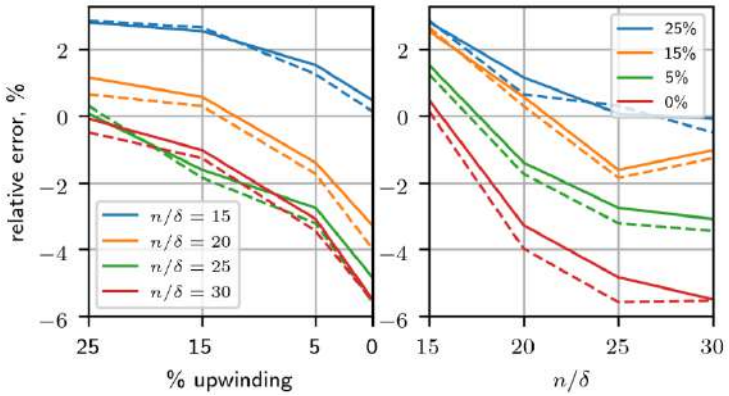

Fig. 1. Relative error in the mean friction velocity (solid lines), relative error in the mean velocity at the sampling point (dashed line). The same data are shown in both plots, with a different quantity used as the abscissa

Attention is now turned to the analysis of the mean velocity profiles in outer scaling, $\langle u \rangle / U_b$. The relative errors in the obtained values are shown in fig. 2. Only the values above the overlap region are plotted, since this is the region where we expect WMLES to give accurate results. It is observed that the amount of upwinding defines the behavior of the error. Less upwinding leads to increasingly larger under-prediction for $y \lesssim 0.5\delta$, and, consequently (due to a fixed flowrate), an over-prediction in the region above. Arguably, the best accuracy across all four considered mesh resolutions is obtained using 15% upwinding. Unfortunately, results generally do not improve with increased mesh resolution apart for the case with 25% upwinding.

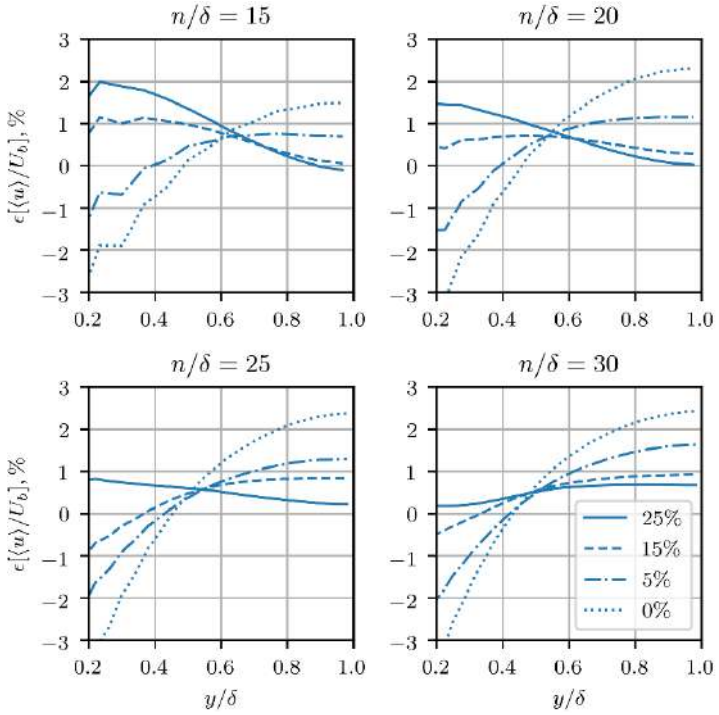

Fig. 2. Relative error in the outer-scaled mean velocity profiles as a function of y/δ

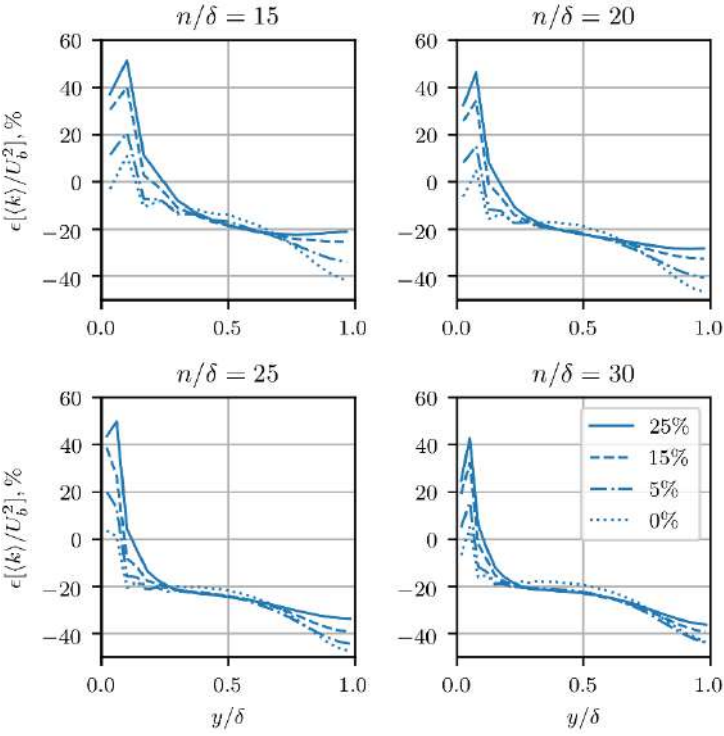

Fig. 3. Relative error in the outer-scaled mean turbulent kinetic energy profiles as a function of y/δ

Fig. 3 shows the relative errors in the profiles of the outer-scaled mean turbulent kinetic energy, $\langle k \rangle / U_b^2$. Firstly, it is evident that the magnitude of the errors is significantly higher than that observed for $\langle u \rangle$ and $\langle u_\tau \rangle$. Near the wall, a large error peak is observed, the size of which monotonously varies with the amount of upwinding. Above the overlap region, the values underpredict the reference DNS data instead. It is noted that the true values of $\langle k \rangle$ exhibit a large peak in the buffer layer, meaning that while the relative errors in the inner and outer regions are comparable in magnitude, the absolute error in the inner region is significantly larger. Interestingly, with increased resolution, the influence of upwinding appears to diminish in the outer region, leading to slightly improved accuracy for the cases of 0% and 5% upwinding. However, similar to the other analyzed flow quantities, no convergence with respect to n/δ can be observed.

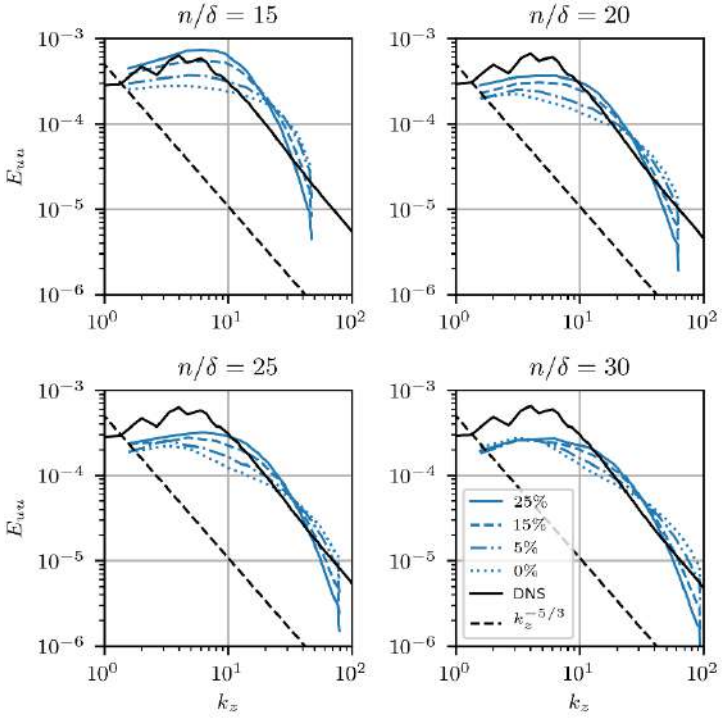

Fig. 4. Spanwise one-dimensional energy spectrum of u at $y/\delta = 0.1$ (the location of the sampling point)

Finally, the spanwise energy spectrum of the streamwise velocity is analyzed. In Fig. 4, the spectra at the location of the sampling point of the wall model, $y = 0.1\delta$, are presented. As expected, increased upwinding leads to faster damping of the high-frequency modes. A more surprising result is that it also leads to increased energy in the low-frequency motions. This effect is amplified at lower mesh resolutions. The inertial range (where the energy is expected to follow the $k_z^{-5/3}$ slope) is significantly less pronounced in the WMLES spectra compared to the DNS data. This is in part due to the limited frequency bandwidth in the WMLES and in part due to the damping of the high-frequency motions mentioned above.

5. Conclusions

Sixteen simulations of fully-developed turbulent channel flow have been conducted in order to investigate the effect of numerical dissipation on the predictive accuracy of wall-modelled LES. The amount of dissipation was varied by changing the resolution of the grid and the amount of upwinding used in the numerical scheme for interpolating convective fluxes at the

cell faces. Several trends in the error patterns of the considered flow quantities with respect to the amount of dissipation were identified. One observation, already reported in previous studies, is that results do not significantly improve with increased mesh resolution. This is unsatisfying, but it is important to keep in mind that mesh convergence is generally not well defined for LES with implicit filtering, and that the cell size plays the double role of defining the smallest eddies that can be resolved and controlling the amount of numerical dissipation. As indicated by the results, the latter affects the solution in a non-trivial way. A positive effect of higher resolution has been observed, however: The simulations using different amounts of upwinding resulted in more similar profiles. It is plausible that further improvements can be achieved by considering denser meshes than those covered in the simulation matrix, and simulations on such meshes is a subject of future works.

With respect to the dissipation coming from the linear upwind interpolation scheme, a monotonous change in the error patterns has been observed. An expected consequence of using more dissipative schemes is the damping of higher-frequency fluctuations, as reflected in the plots of the energy spectra (see fig. 4). A more interesting finding is that dissipation appears to lead to more energetic large-scale motions. These are likely connected to the observed near-wall peaks in the error in $\langle k \rangle$ (see fig. 3). These peaks were reported for the case of 25% upwinding before, and here it is shown that their size decreases monotonously with decreased upwinding. The mechanism behind the production of excessive $\langle k \rangle$ and the overly energetic large-scale motions requires further investigation and is a subject of future work. In particular, it would be interesting to analyze the budgets of k and each of the Reynolds stress components.

The mean velocity is the quantity for which increased dissipation led to increasingly more accurate results. It should be noted that the disparity among the error curves corresponding to different amounts of upwinding is only $\approx$ 1-2%, an order of magnitude less than what is observed for $\langle k \rangle$. It can be argued that the simulation parameters should be adjusted based on the more sensitive quantity. On the other hand, in practical applications the mean velocity is often seen as the primary quantity of interest, and to a large extent it also controls the error in the predictions of the wall shear stress made by the wall model (see fig. 1). Therefore, it is important to further develop the computational methodology so that an accurate mean velocity profile can be achieved without compromising second-order statistics.

References

[1]. H. Choi and P. Moin. Grid-point requirements for large eddy simulation: Chapman's estimates revisited. Physics of Fluids, vol. 24, 011702, 2012.

[2]. M. Liefvendahl and C. Fureby. Grid requirements for LES of ship hydrodynamics in model and full scale. Ocean Engineering, vol. 143, 2017, pp. 259–268.

[3]. S. Rezaeiravesh and M. Liefvendahl. Grid construction strategies for wall-resolving large eddy simulation and estimates of the resulting number of grid points. Technical report, Department of Information Technology, Uppsala University, 2017.

[4]. U. Piomelli and E. Balaras. Wall-layer models for large-eddy simulations. Annual Review of Fluid Mechanics, vol. 34, 2002, pp. 349–374.

[5]. U. Piomelli. Wall-layer models for large-eddy simulations. Progress in Aerospace Sciences, vol. 44, no. 6, 2008, pp. 437–446.

[6]. J. Larsson, S. Kawai, J. Bodart, and I. Bermejo-Moreno. Large eddy simulation with modeled wall-stress: recent progress and future directions. Mechanical Engineering Reviews, vol. 3, no. 1, 2016, pp. 1–23.

[7]. S. T. Bose and G. I. Park. Wall-modeled large-eddy simulation for complex turbulent flows. Annual Review of Fluid Mechanics, vol. 50, 2018, pp. 535–561.

[8]. M. Wang and P. Moin. Dynamic wall modeling for large-eddy simulation of complex turbulent flows. Physics of Fluids, vol. 14, 2002, pp. 2043–2051.

[9]. T.-H. Shih, L. A. Povinelli, N.-S. Liu, and K.-H. Chen. Generalized wall function for complex turbulent flows. Journal of Turbulence, vol. 4, no. 15, 2003, pp. 37–41.

[10]. S. Kawai and K. Asada. Wall-modeled large-eddy simulation of high Reynolds number flow around an airfoil near stall condition. Computers and Fluids, vol. 85, 2013, pp. 105–113.
[11]. C. Duprat, G. Balarac, O. Métais, P. M. Congedo, and O. Brugière. A wall-layer model for large-eddy simulations of turbulent flows with/out pressure gradient. Physics of Fluids, vol. 23, 015101, 2011.
[12]. G. I. Park and P. Moin. An improved dynamic non-equilibrium wall-model for large eddy simulation. Physics of Fluids, vol. 26, 015108, 2014.
[13]. W. Sidebottom, O. Cabrit, I. Marusic, C. Meneveau, A. Ooi, and D. Jones. Modelling of wall shear-stress fluctuations for large-eddy simulation. In Proc. of the 19th Australasian Fluid Mechanics Conference, 2014, pp. 8–11.
[14]. G. I. Park. Wall-modeled large-eddy simulation of a high Reynolds number separating and reattaching flow. AIAA Journal, vol. 55, no. 11, 2017, pp. 3709–3721.
[15]. W. H. Cabot and P. Moin. Approximate wall boundary conditions in the large-eddy simulation of high Reynolds number flow. Flow, Turbulence and Combustion, vol. 63, 1999, pp. 269–291.
[16]. S. Rezaeiravesh, T. Mukha, and M. Liefvendahl. Systematic study of accuracy of wall-modeled large eddy simulation using uncertainty quantification techniques. Computers and Fluids, vol. 185, 2019, pp. 34–58.
[17]. P. Sagaut. Large eddy simulation for incompressible flows: An introduction. Springer-Verlag, 2005, 558 p.
[18]. F. Nicoud and F. Ducros. Subgrid-scale stress modelling based on the square of the velocity gradient tensor. Flow, Turbulence and Combustion, vol. 62, no. 3, 1999, pp. 183–200.
[19]. H. Jasak. Error analysis and estimation for the finite volume method with applications to fluid flows. PhD Thesis, Imperial College of Science, Technology and Medicine, 1996, 396 p.
[20]. H. G. Weller, G. Tabor, H. Jasak, and C. Fureby.A tensorial approach to computational continuum mechanics using object-oriented techniques. Computers in Physics, vol. 12, no. 6, 1998, pp. 620–631.
[21]. E. De Villiers. The potential of large eddy simulation for the modeling of wall bounded flows. PhD Thesis, Imperial College of Science, Technology and Medicine, 2006.
[22]. T. Mukha, S. Rezaeiravesh, and M. Liefvendahl. A library for wall-modelled large-eddy simulation based on OpenFOAM technology. Computer Physics Communications, vol. 239, 2019.
[23]. S. Kawai and J. Larsson. Wall-modeling in large eddy simulation: Length scales, grid resolution, and accuracy. Physics of Fluids, vol. 24, no. 1, 015105, 2012.
[24]. D. B. Spalding. A single formula for the `law of the wall'. Journal of Applied Mechanics, vol. 28, no. 3, 1961, pp. 455–458.
[25]. M. Lee and R. D. Moser. Direct numerical simulation of turbulent channel flow up to $\mathrm{Re}_\tau \approx 5200$. Journal of Fluid Mechanics, vol. 774, pp. 395–415, 2015.

Информация об авторах / Information about authors

Тимофей Дмитриевич МУХА, Ph.D., является сотрудником факультета механики и морских наук технического университета Чалмерса. Его научные интересы связаны с разработкой вихреразрешающих методов моделирования турбулентности, а также моделированием многофазных течений.

Timofey MUKHA, Ph.D., is a postdoctoral researcher at the Department of Mechanics and Maritime Sciences at Chalmers University of Technology. His main research interests are scale-resolving turbulence modelling approaches and modelling of multiphase flows.

DOI: 10.15514/ISPRAS-2019-31(6)-12

CC BY

Анализ полного сопротивления корпуса судна на различных скоростях хода

К.Д. Овчинников, ORCID: 0000-0001-8753-6243 <ovchinnikov_kd@mail.ru>

Санкт-Петербургский государственный морской технический университет, 190121, Санкт-Петербург, ул. Лоцманская, 3

Аннотация. В статье предложен метод для анализа полного сопротивления движению судна на скоростях хода, соответствующих числу Фруда по длине от 0,14 до 0,41. Основная идея метода состоит в разделении корпуса судна на составляющие поверхности. Данный подход позволяет рассчитать распределение силы сопротивления по длине судна. На основании полученных данных могут быть выбраны зоны для корректировки обводов, а также может быть выполнено более качественное сравнение нескольких вариантов корпуса. Перед примером применения предложенного метода были выполнены постановка задачи расчета сопротивления в программном комплексе OpenFOAM, валидация расчетной схемы и проверка сеточной сходимости. Для выполнения расчетов использовались стандартный решатель interDyMFoam и модель корпуса DTMB 5415. Для реализации предложенного метода корпус DTMB 5415 был разделен на 22 поверхности по длине судна. По результатам расчетов были получены кривые распределения силы сопротивления по длине корпуса для рассматриваемых скоростей хода. По результатам анализа кривых были разработаны два варианта обводов носовой оконечности корпуса DTMB 5415: без бульба и с обтекаемым бульбом. Оба варианта обводов обладают меньшим сопротивлением на всех скоростях хода по сравнению с базовым корпусом DTMB 5415. По результатам анализа кривых распределения силы сопротивления трех корпусов получено, что в районе средней и кормовой частей корпуса (~3/4 длины судна) распределенная сила сопротивления практически не зависит от скорости хода и обводов носовой оконечности. Носовая оконечность при этом играет первостепенную роль в формировании значении полного сопротивления корпуса.

Ключевые слова: численное моделирование; эксперимент; сопротивление; DTMB 5415; OpenFOAM

Для цитирования: Овчинников К.Д. Анализ полного сопротивления корпуса судна на различных скоростях хода. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 195-202. DOI: 10.15514/ISPRAS–2019–31(6)–12

Analysis of total resistance for different ship speeds

K.D. Ovchinnikov ORCID: 0000-0001-8753-6243 <ovchinnikov_kd@mail.ru>

Saint-Petersburg State Marine Technical University, 3, Lotsmanskaya st., Saint-Petersburg, 190121, Russia

Abstract. The paper shows method for analyzing total resistance of ship which calculated on Froude numbers from 0.14 to 0.41. To this method the hull surface divides into its component surfaces to the ship length. This approach allows to calculate the distribution of resistance forces along the ship length. Based on this data hull zones for correcting can be selected, and a better comparison of several hull can be made. First of all mathematical scheme of total resistance calculating develops and validates in the OpenFOAM software. The standard solver interDyMFoam and DTMB 5415 hull data uses. Three grids uses for mesh convergence. To implement the proposed method the DTMB 5415 hull divides into 22 surfaces along the length. According to the results of numerical simulations the curves of distribution of resistance force

along the hull length for the different ship speeds obtain. Due to analysis of curves two versions of hull develops based on DTMB 5415 hull: without bulb and with streamlined bulb. Both hulls have lower resistance at all speeds compared to the DTMB 5415 base hull. Due to analysis of resistance force distribution curves of new hulls distributed resistance force on middle and stern parts of hull does not depend on ship speed and stem part lines. Stem part of hull has the most important influence on total resistance.

Keywords: numerical simulation; CFD; experiment; resistance; DTMB 5415; OpenFOAM

For citation: Ovchinnikov K.D. Analysis of total resistance for different ship speeds. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 195-202 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-12

1. Введение

При создании новых объектов морской техники особое внимание уделяют сопротивлению движению. Связано это в первую очередь с тем, что на преодоление сопротивления уходит наибольшая часть энергии, вырабатываемой энергетической установкой.

В рамках решения проектной задачи стараются выбрать такую форму корпуса, чтобы сопротивление движению было минимальным при удовлетворительных значениях прочих мореходных и эксплуатационных характеристик.

Применение программных продуктов вычислительной гидромеханики получает все большее распространение при проектировании судов. Это связано с относительной дешевизной отработки различных вариантов формы корпуса по сравнению с экспериментальными исследованиями.

В настоящей работе предложен и рассмотрено применение метода для анализа полного сопротивления судна, путем разделения корпуса по длине на составляющие поверхности. Численное моделирование реализуется в программном комплексе с открытым кодом OpenFOAM [1].

2. Математическая постановка задачи в программном комплексе OpenFOAM

Уравнение Навье-Стокса для несжимаемой жидкости определяется как система из уравнений сохранения импульса и уравнения неразрывности.

При высоких числах Рейнольдса вычисления прямым численным моделированием сопровождаются либо большими временными издержками, либо большими вычислительными мощностями, поэтому для моделирования турбулентных потоков обычно используются уравнения Навье-Стокса осредненные по времени, называемые уравнениями Рейнольдса. Эти уравнения получены из уравнений Навье-Стокса путем разбиения полей скорости давления на среднее значение и флуктуацию [2].

Для замыкания системы уравнений используется двухпараметрическая модель турбулентности k-ω SST [3].

Дискретизация фундаментальных уравнений в программном комплексе OpenFOAM выполняется с использованием схемы контрольного объема, свободная поверхность моделируется с помощью модифицированного метода объема жидкости.

Движение твёрдого тела, обладающего шестью степенями свободы, может быть описано динамическими уравнениями Эйлера [4]. В соответствии с описанной схемой решения поставленной задачи выбран стандартный решатель interDyMFoam.

Для создания расчетной области используются внутренние утилиты пакета OpenFOAM, такие как topoSet, refineMesh и snappyHexMesh. Размеры расчетной области определены в соответствии с [5].

3. Валидация расчетной схемы

Для выполнения контрольных расчетов выбран корпус DTMB 5415 [6, 7, 8].

Характеристики корпуса DTMB5415: длина между перпендикулярами L_{pp} = 3,048 м, ширина B = 0,409 м, осадка T = 0,132 м, водоизмещение D = 83,5 кг, аппликата центра тяжести z_g = 0,163 м, момент инерции относительно продольной оси J_{44} = 1,92 кг·м^2, момент инерции относительно поперечной оси J_{55} = 48,5 кг·м^2.

При выполнении валидации следует также провести проверку сеточной сходимости. Для этого было разработано несколько расчетных областей с осями симметрии по диаметральной плоскости корпуса судна со следующими характеристиками:

- «грубая» сетка: количество ячеек расчетной области ~936000, количество граней на поверхности корпуса ~13400, максимальный коэффициент пропорциональности ~80, максимальный коэффициент закрученности ячейки ~3,2, среднее значение неортогональности ~4,7;
- «средняя» сетка: количество ячеек расчетной области ~2867000, количество граней на поверхности корпуса ~25000, максимальный коэффициент пропорциональности ~53, максимальный коэффициент закрученности ячейки ~2,7, среднее значение неортогональности ~3,4;
- «хорошая» сетка: количество ячеек расчетной области ~7458000, количество граней на поверхности корпуса ~42000, максимальный коэффициент пропорциональности ~40, максимальный коэффициент закрученности ячейки ~2,4, среднее значение неортогональности ~2,7.

Расчеты выполнялись для трех скоростей хода:

- при скорости хода, соответствующей числу Фруда Fr = 0,14; при этой скорости хода преобладающее влияние в полном сопротивлении имеет сопротивление трения;
- при скорости хода, соответствующей числу Фруда Fr = 0,28; при этой скорости хода волновое сопротивление начинает оказывать все большую важную роль в полном сопротивлении, возникает первый горб волнового сопротивления;
- при скорости хода, соответствующей числу Фруда Fr = 0,41; при этой скорости хода волновое сопротивление играет преобладающую роль в полном сопротивлении, может возникать основной горб волнового сопротивления [9].

Численное моделирование выполнялось на вычислительном кластере ФГБОУ ВО «Санкт-Петербургский государственный морской технический университет». Кластер включает в себя восемь узлов по 20 ядер. Один расчет выполнялся на одном узле. Время численного моделирования было фиксировано и равнялось 20 секундам. За это время значения сопротивления выравнивались, равно как и волнообразование вокруг корпуса модели.

Сравнение результатов численного моделирования обтекания моделей в пакете OpenFOAM и результатов экспериментального исследования представлено в табл. 1. В табл. 1 приведены следующие условные обозначения: Fr – число Фруда по длине; R_t – полное сопротивление, Н; ξ – абсолютная погрешность, Н; ζ – относительная погрешность, %; t – время расчета, ч.

Табл. 1. Сопротивление корпуса DTMB 5415 для различных расчетных областей
Table 1. DTMB 5415 resistance for different grids

	Эксперимент			«Грубая» сетка			«Средняя» сетка			«Хорошая» сетка		
Fr	0,14	0,28	0,41	0,14	0,28	0,41	0,14	0,28	0,41	0,14	0,28	0,41
R_t, N	1,79	7,43	23,36	1,57	6,93	21,90	1,68	7,58	23,48	1,69	7,70	23,16
ξ, N	-	-	-	-0,22	-0,50	-1,46	-0,11	0,15	0,12	-0,10	0,27	-0,20
ζ	-	-	-	-12%	-7%	-6%	-6%	2%	1%	-6%	4%	-1%
t, h	-	-	-	2,6	4,0	6,0	10,0	15,9	31,0	23,0	57,9	146,7

На основе анализа данных, представленных в табл. 1, сделан вывод о том, что удовлетворительные результаты получаются при расчетах на «средней» и «хорошей» сетках. При этом в проектной практике рекомендуется использование «средних» сеток, поскольку время выполнения численного моделирования при этом существенно меньше, нежели при использовании «хорошей» сетки.

Здесь также следует отметить один факт: при малых числах Фруда, когда основную роль в полном сопротивлении играет сопротивление трения, результаты численного моделирования имеют достаточно высокие значения относительной погрешности (до 6 %). При увеличении скорости хода, когда волновое сопротивление начинает играть большую роль, значения относительной погрешности падают.

4. Описание метода анализа полного сопротивления корпуса

При экспериментальном исследовании могут быть получены только полное сопротивление для всего корпуса. Численное моделирование также позволяет получить полное сопротивление, действующее на тело. Однако при численном моделировании есть возможность получить распределение силы сопротивления по длине судна – кривой распределения силы сопротивления.

При наличии кривой распределения силы сопротивления по длине корпуса судна имеется возможность для более точного анализа характеристик обтекания тела, а при сравнении нескольких корпусов – более точного задания функции оптимума.

Для получения кривой распределения сопротивления по длине судна корпус должен быть задан как набор поверхностей, полученных путем разделения по длине единого замкнутого тела на составляющие.

Эталонный расчет кривой распределения силы сопротивления по длине корпуса для модели DTMB 5415 выполнен при представленных ранее скоростях хода. Моделирование выполнялось на «хорошей» сетке.

При подготовке численного моделирования корпус был разделен на поверхности по 21 теоретическому шпангоуту – итоговый корпус имел 22 поверхности (рис. 1).

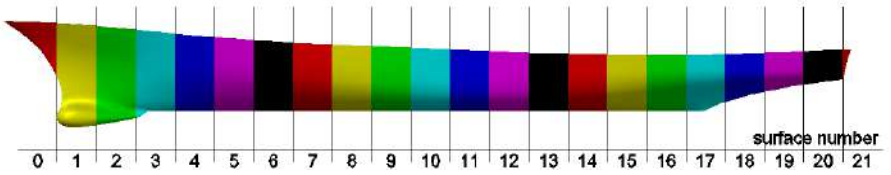

Рис. 1. Корпус DTMB 5415, разделенный на поверхности
Fig. 1. DTMB 5415 hull divided on surfaces

По результатам численного моделирования получены кривые распределения силы сопротивления по длине корпуса при различных скоростях хода, представленные на рис. 2. Если сложить все значения, представленные на рис. 2 для выбранной скорости хода, получатся значения полного сопротивления, представленные в табл. 1.

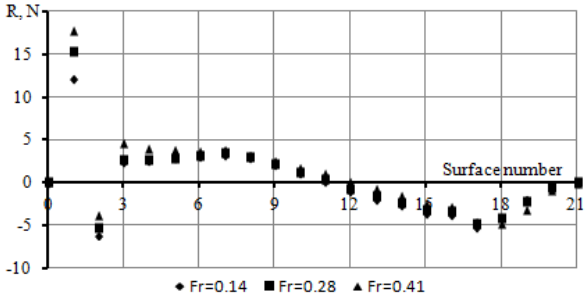

Рис. 2. Распределение силы сопротивления по длине корпуса DTMB 5415
Fig. 2. Distribution of resistance force on length of DTMB 5415 hull

На основе анализа данных, представленных на рис. 2, сделаны следующие выводы:

- кривая распределения силы сопротивления по длине корпуса имеет как положительные, так и отрицательные значения, что связано непосредственно с формой корпуса;
- наибольшие значения силы сопротивления возникают на носовой части бульба (поверхность № 1), и они тем больше, чем выше скорость хода;
- из-за обводов бульба наблюдается значительный скачок силы сопротивления в районе кормовой части бульба (поверхность № 2);
- от поверхности № 6 до самой кормы (~3/4 длины судна) значения распределенной силы сопротивления мало зависят от скорости хода, что может говорить о том, что основную роль в увеличении полного сопротивления при повышении скорости хода играет носовая оконечность;
- для уменьшения полного сопротивления корпуса судна необходимо произвести корректировку обводов носовой оконечности, в частности – бульба.

5. Анализ кривой распределения силы сопротивления для различных вариантов носовой оконечности

На базе корпуса DTMB 5415 (далее – корпус 1) были разработаны два варианта новых обводов носовой оконечности: корпус 2 – без бульба, и корпус 3 – с обтекаемым бульбом.

Все три рассматриваемых корпуса представлены на рис. 3.

Расчеты для корпусов 2 и 3 выполнялись на «средней» сетке. Поскольку обводы корпусов изменились незначительно, характеристики расчетных областей также изменились незначительно.

В табл. 2 представлены результаты расчетов полного сопротивления рассматриваемых корпусов при различных скоростях хода.

Табл. 2. Полное сопротивления для различных корпусов
Table 2. Total resistance for different hulls

Число Фруда по длине Fr	Корпус 1	Корпус 2	Корпус 3
0,14	1,68	1,19	1,47

0,28	7,70	6,14	6,73
0,41	23,16	21,27	21,30

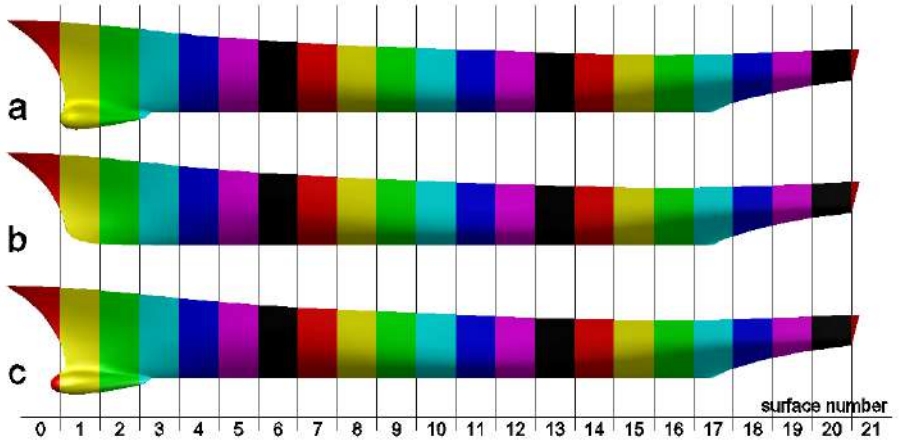

Рис. 3. Корпуса, разделенные на поверхности: a – Корпус 1 (DTMB 5415), b – Корпус 2, c – Корпус 3
Fig. 3. Hulls divided on surfaces: a – Hull 1 (DTMB 5415), b – Hull 2, c – Hull 3

В табл. 2 показано, что наименьшее сопротивление при всех скоростях хода имеет корпус 2 (без бульба). Однако здесь следует отметить одну проектную деталь: корпус DTMB 5415 имеет обводы военного корабля, которому бульб необходим для размещения гидроакустической станции. Корпус 3 имеет бульб, вместимость которого не меньше вместимости бульба корпуса 1, а сопротивление при всех рассматриваемых скоростях хода меньше.

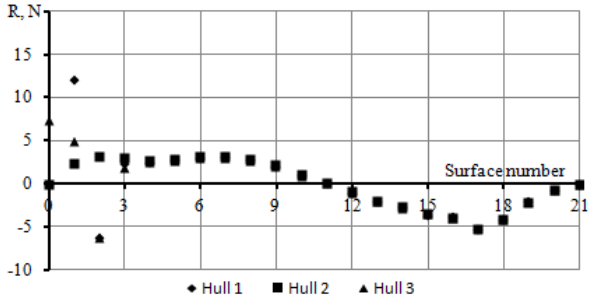

Рис. 4. Кривые распределения силы сопротивления по длине корпуса при скорости хода, соответствующей числу Фруда по длине Fr = 0,14
Fig. 4. Distribution of resistance force on hull surfaces for Froude number Fr = 0,14

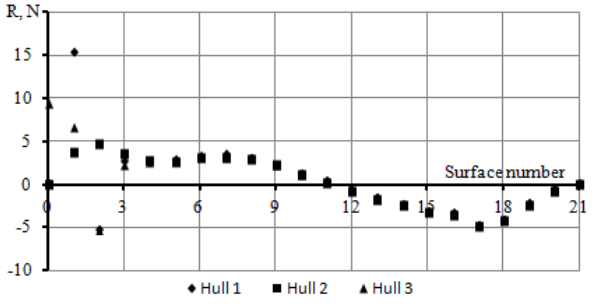

Рис. 5. Кривые распределения силы сопротивления по длине корпуса при скорости хода, соответствующей числу Фруда по длине Fr = 0,28
Fig. 5. Distribution of resistance force on hull surfaces for Froude number Fr = 0,28

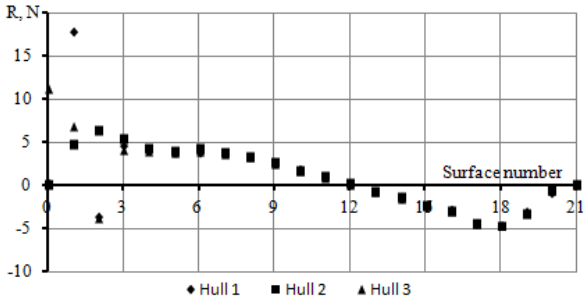

Рис. 6. Кривые распределения силы сопротивления по длине корпуса при скорости хода, соответствующей числу Фруда по длине Fr = 0,41
Fig. 6. Distribution of resistance force on hull surfaces for Froude number Fr = 0,41

На основе анализа данных, представленных на рис. 4 – 6, сделаны следующие выводы:

- при отсутствии бульба кривая распределения силы сопротивления по длине корпуса не имеет резких скачков, что благоприятно влияет на значение полного сопротивления;
- изменение обводов бульба привело к перераспределению силы сопротивления, что благоприятно сказалось на значении полного сопротивления;
- несмотря на изменение обводов бульба, отрицательные значения на поверхности № 3 практически не изменились;
- от поверхности № 6 до самой кормы (~3/4 длины судна) значения распределенной силы сопротивления не зависят от формы носовой оконечности.

6. Заключение

По результатам работы можно сделать следующие выводы.

Программный комплекс OpenFOAM рекомендуется для расчета сопротивления движению судна с использованием как минимум «средних» сеток. При этом погрешность расчета падает с увеличением скорости хода, что связано с уменьшением влияния сопротивления трения на значение полного сопротивления.

Кривая распределения силы сопротивления по длине судна имеет ярко выраженные скачки в носовой части судна при наличии бульба. При этом, начиная со средней части корпуса и заканчивая кормой (~3/4 длины судна), значения распределенной силы сопротивления практически не зависят от скорости хода.

Изменения обводов носовой оконечности судна не оказывают влияния на значения распределенной силы сопротивления в средней и кормовой частях корпуса (~3/4 длины судна). Носовая оконечность играет первостепенную роль в формировании значения полного сопротивления корпуса.

Дальнейшие направления работы включают, разработку численной схемы для оценки значений распределенной силы сопротивления носовой оконечности (или другой части) судна при серьезных уменьшениях временных затрат для решения задачи оптимизации формы корпуса с точки зрения ходовых качеств, а также разработку метода оценки влияния различных составляющих сопротивления (сопротивления трения,

сопротивлении формы и волнового сопротивления) на значение полного сопротивления корпуса.

Список литературы / References

[1] The OpenFOAM Foundation. Available at: https://openfoam.org/, accessed 01.09.2019.

[2] Henry Peter Piehl. Ship Roll Damping Analysis. Von der Fakultät für Ingenieurswissenschaften, Abteilung Maschinenbau und Verfahrenstechnik, der Universität Duisburg-Essen zur Erlangung des akademischen Grades eines. Doktors der Ingenieurswissenschaften Dr.-Ing. April 2016.

[3] Menter, F. R. Two-equation eddy-viscosity turbulence models for engineering applications. AIAA Journal, vol. 32, no. 8, 1994, pp. 1598-1605.

[4] Овчинников К.Д., Тряскин Н.В., Ткаченко И.В. Численное моделирование качки полупогружной платформы на регулярном волнении. Морские интеллектуальные технологии, № 2, том 1 2015 г., стр. 14-18 / Ovchinnikov K.D., Tryaskin N.V., Tkachenko I.V. Numerical simulation of motions of semisubmersible rig in regular waves. Marine intellectual technologies, № 2, vol. 1, 2015, pp. 14-18 (in Russian).

[5] Ovchinnikov K.D. Numerical simulation of motions of ship with moonpool in head waves. Trudy ISP RAN/Proc. ISP RAS, vol. 30, issue 5, 2018. pp. 235-248. DOI: 10.15514/ISPRAS-2016-30(5)-14.

[6] Gui L., Longo J., and Stern F. Biases of PIV Measurement of Turbulent Flow and the Masked Correlation-Based Interrogation. Experiments in Fluids, vol. 30, 2001, pp. 27-35.

[7] Gui L., Longo J., and Stern F. Towing Tank PIV Measurement System, Data and Uncertainty Assessment for DTMB Model 5512. Experiments in Fluids, vol. 31, 2001, pp. 336-346.

[8] Longo J. and Stern F. Uncertainty Assessment for Towing Tank Tests with Example for Surface Combatant DTMB Model 5415. Journal of Ship Research, vol. 49, no. 1, 2005, pg. 55-68.

[9] Справочник по теории корабля: в трех томах. Том 1. Гидромеханика. Сопротивление движению судов. Судовые движители. Под ред. Я.И. Войткунского. Л., Судостроение, 1985, 785 стр. / Ship theory handbook. Ed. by Y.I. Voitkounski. In 3 volumes. Vol. 1. Hydromechanics, Resistance of Ship, Ship Propulsion Devices. Leningrad. Sudostroenie, 1985. 768 p. (in Russian).

Информация об авторах / Information about authors

Кирилл Дмитриевич ОВЧИННИКОВ – кандидат технических наук, старший преподаватель кафедры проектирования судов Санкт-Петербургского государственного морского технического университета с 2015 года. Сфера научных интересов: проектирование судов, теория корабля, методы вычислительной гидродинамики.

Kirill D. OVCHINNIKOV – PhD, senior lecturer of ship design department in Saint-Petersburg State Marine Technical University since 2015. Research interests: ship design, ship theory, computation fluid mechanics.

DOI: 10.15514/ISPRAS-2019-31(6)-13

CC BY

Numerical study of effect of the turbulence initial conditions on transition flow over 2D airfoil

Ali Rami, ORCID: 0000-0003-0591-6221 <ramimamdouhali@gmail.com>
N.V. Tryaskin, ORCID: 0000-0002-2208-2241 <nikita.tryaskin@smtu.ru>

State Marine Technical University (SMTU),
190121, Russia, Saint-Petersburg, Lotsmanskaya st., 3

Abstract. The performance of the airfoil is strongly dependent on the development of the boundary layer on the surface and therefore an accurate prediction of the laminar to turbulent transition onset is essential. A grid independence study is performed, turbulence variables values have been changed frequently, regularly, and carefully so that they cover the entire range of acceptable values reported by previous researches. Effects of turbulent variating at far stream on turbulent boundary layer structure and on transition stage characteristics at moderate Reynolds number $\Re = 10^6$ have been studied over a full range of angles of attack of NACA0012. numerical results have been post-processed, analyzed and found that far stream turbulence variables have a significant effect on transition characteristic, their effects on skin friction is limited to small extent along wing surface where transition take place, increasing turbulence intensity or eddy viscosity ratio at far boundary shifts transition onset towards leading edge and increase transition length.

Keyword: Turbulence intensity; Eddy viscosity ratio; k-ω SST; Transition; NACA0012

For citation: Ali Rami, Tryaskin N.V. Numerical study on effect of the turbulence initial conditions on transition flow over 2D airfoil. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 203-214. DOI: 10.15514/ISPRAS-2019-31(6)-13

Численное изучение влияния начальных турбулентных параметров на переходный режим над плоским крылом

Али Рами, ORCID: 0000-0003-0591-6221 <ramimamdouhali@gmail.com>
Н.В. Тряскин, ORCID: 0000-0002-2208-2241 <nikita.tryaskin@smtu.ru>

Санкт-Петербургский государственный морской технический университет,
190121, Россия, Санкт-Петербург, ул. Лоцманская, 3

Аннотация. Эксплуатационные характеристики аэродинамического профиля сильно зависят от развития пограничного слоя на поверхности, и поэтому точный прогноз начала ламинарного перехода к турбулентному имеет важное значение. Проводится исследование сеточной сходимости, начальные значения турбулентных параметров меняются таким образом, чтобы охватить весь диапазон допустимых значений. Влияние параметров турбулентности в дальнем потоке на структуру турбулентного пограничного слоя и на характеристики зоны ламинарно-турбулентного перехода при умеренном числе Рейнольдса $\Re = 10^6$ было изучено во всем диапазоне углов атаки NACA0012. Численные результаты были проанализированы и обнаружено, что переменные турбулентности в дальнем потоке оказывают существенное влияние на характеристики перехода, их влияние на изменение коэффициента трения ограничивается областью крыла, где происходит переход. При увеличении интенсивности турбулентности или коэффициента вихревой вязкости на дальней границе сдвигается начало перехода к передней кромке и увеличивается его длина.

Ключевые слова: интенсивность турбулентности; коэффициент вихревой вязкости;, k-ω SST; ламинарно-турбулентный переход, NACA001.

Для цитирования: Али Рами, Тряскин Н.В. Численное изучение влияния начальных турбулентных параметров на переходный режим над плоским крылом. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 203–214 (на английском языке). DOI: 10.15514/ISPRAS–2019–31(6)–13

1. Introduction

Flows at moderate and high Reynolds number are characterized by complex turbulent boundary-layer effects, including leading-edge laminar-to-turbulent transition, flow reattachment, trailing-edge separation, and leading-edge separation.

Laminar-to-turbulent transition in shear layer of the flows over airfoil is of particular importance, although it has subjected to intensive research over the past decades but it still far from completely understood.

The location where transition starts, and the spatial extension occupied by it are of a great importance in the airfoil performance. Therefore, the accurate understanding of transition process allows us adjusting the flow and controlling its nature so we can delay the turbulent phase where laminar flow characteristics are desirable or to accelerate it where high mixing rates of turbulent flow are of interest.

The difficulty in modelling transition arises from its non-linearity, wide range of scales, and the fact that it can occur through different mechanisms [1]. An overview of different scenarios of laminar-turbulent transition can be found in studies [2,3,4].

The transition process is greatly influenced by pressure gradients and separation, Mach number effects, free-stream turbulence, wall roughness, streamline curvature, surface heating or cooling, suction or blowing of fluid from the wall and so on [5].

H.A. Madsen et al. researched effects of leading-edge roughness on Laminar-Turbulent Transition, their analyses show that the critical height of the leading-edge roughness is to be met in order to have a bypass transition to turbulent flow [6]. Mayle [7] ensured the importance of pressure gradient and turbulence intensity on the transition onset on airfoil. Butler et al. [8] studied effects of turbulent intensity on flow separation at low Reynolds number and found that increasing turbulence intensity to high level 10% prevent the separation. Mueller and Pohlen [9], Hoffmann [10] and Huang and Lee [11] show that the increasing in turbulent intensity has resulted increase in maximum lift coefficient for the airfoil. Jian-Ping Wang et al researched the effect of the turbulence intensity of the oncoming flow with low Reynolds numbers on the airfoil performance and found that turbulence intensity has significant impact on lift and drag coefficients [12]. Ning Cao focused in his study on the independent effects of the turbulence intensity and integral length scale on the Cl and Cd, of an asymmetric, high lift airfoil, at different $\Re$ and found that increasing Tu leads to delay stall at high angle of attack [13],

However, a literature review shows that there is a little investigation about the effects of eddy viscosity ratio at certain value of turbulent intensity on the transition character over airfoil at moderate Reynold number. Bridging this gap is the main motivation of current study. So, this paper deals with the effects of far stream turbulence variables on the onset and length of laminar to turbulent transition and its influence on flow character and airfoil performance at moderate Reynold number.

The paper starts with a description of the methodology in section 2, which includes a brief description of used turbulence model k-ω SST and wall functions, the geometrical and operational characteristics of the chosen airfoil, the grid and computational domain, the numerical settings and grid convergence analyze which include a sensitivity analyze of results to grid resolution and in addition to validation with experimental data [14].

Finally results and discussion of far stream turbulence parameter effects on transition onset and length are discussed in section 3, conclusions are presented in section 4.

2. Methodology

2.1 k-ω SST standard turbulence model

The Menter Shear Stress Transport Turbulence Model k-ω SST is a two-equation eddy-viscosity model used for many hydrodynamic and aerodynamic applications, this model combines the well-known low Reynolds turbulence model $k-\omega$ and high Reynolds turbulence model $k-\varepsilon$, the former is suitable for simulating flow in the viscous sub-layer but it suffers from a high sensitivity to the inlet free-stream turbulence properties, while the latter is ideal for predicting flow behavior in regions away from the wall but it performs poorly in adverse pressure gradient situations and cannot be used all the way down to wall.

To eliminate the disadvantages of each individual model and to maintain their strengths a blinder function F_1 is used to switch between $k-\varepsilon$ in the free stream and $k-\omega$ near the walls, which ensures that the appropriate model is utilized throughout the flow field. Therefore, k-ω SST model can be used all the way down to the wall without being influenced by free-stream turbulence [15,16].

The two transport equations that form the basis of the model are as follows:

$$\frac{\partial(\rho k)}{\partial t}+\frac{\partial(\rho u_j k)}{\partial x_j}=P-\beta^*\rho\omega k+\frac{\partial}{\partial x_j}\left[(\mu+\sigma_k\mu_t)\frac{\partial k}{\partial x_j}\right]$$

$$\frac{\partial(\rho\omega)}{\partial t}+\frac{\partial(\rho u_j\omega)}{\partial x_j}=\frac{\gamma}{\nu_t}P-\beta\rho\omega^2+\frac{\partial}{\partial x_j}\left[(\mu+\sigma_\omega\mu_t)\frac{\partial\omega}{\partial x_j}\right]+2(1-F_1)\frac{\rho\sigma_{\omega2}}{\omega}\frac{\partial k}{\partial x_j}\frac{\partial\omega}{\partial x_j}$$

In these equations, P is the sources term, F_1 is a blending function and β, β^*, $\sigma_{\omega2}$, σ_ω and σ_k are constant.

Viscous effects near walls are characterized by the non-dimensional wall distance $y^+=(y\cdot u^*)/\nu$, where u^* is the friction velocity, y is the distance to the wall, and ν is the kinematic viscosity of the fluid. Using low Reynolds turbulence model as approach to simulate flow near the wall enforces us to place the first cell center in the viscous sublayer, advised to be $y^+\approx 1$, that leads to finest mesh and thus increasing computing time, the case will be the same when using modified high Reynolds turbulence models. Launder and Spalding [17] proposed the wall function which is an empirical equation used to satisfy the physics of the flow in the near wall region so the first cell center now needs to be placed in the log-law region $y^+\geq 30$ to ensure the accuracy of the result, both approaches produce large errors if used outside of their range of validity.

It is difficult to ensure that cells adjacent to the wall will all fall within the desired layer, so in OpenFOAM the existing wall functions have been modified and improved to include the buffer layer [18] and to ensure that they can provide the accurate result so wherever the position of the first cell center [19]. In this study we will employ OpenFOAM Low Reynold wall functions to approximate the viscous quantities in the boundary layer surrounding the airfoil, this type of wall functions has improved to allow the first cell center to be placed in buffer layer, depending on internal switch developed based on the value of yPlusLam coefficient. yPlusLam is determined by an iteration process designed to divide buffer layer into two parties, one uses the liner relation as that in viscous sublayer, and the other uses logarithmic function as that in logarithmic region. So, these wall functions give appropriate values wherever the position of the first cell center, that gives us more flexibility during the mesh optimization stage, facilitate calculations and reduce simulation time, now minimum wall distance y and y^+ value could be enlarged, and the case will be properly modeled [19].

We have to indicate that the theory of wall functions used in OpenFOAM is based on the paper proposed by Georgi Kalitzin et al. [20].

2.2 Chosen Airfoil

In this paper we will use NACA 0012 airfoil, a conventional, symmetrical airfoil with maximum thickness $12\%C$ located at $30\%C$, where C is the airfoil chord length.

NACA 0012 belongs to NACA 4-digits airfoil series (NACA MPXX), a series controlled by 4 digits prepared carefully to describe the geometrical properties of an airfoil, maximum camber, position of the maximum camber and maximum airfoil thickness at 30% chord, respectively. These digits are given as percentages of the airfoil chord length C. The camber line of airfoil, in addition to the perpendicular distribution of the thicknesses along its length, is given by mathematical equations, thus facilitate the generation of the geometrical section of any airfoil.

2.3 Computational domain and grid

O-Grid domain prepared to simulate flow over the wing, based on similar studies and to avoid confinement effects and consequently to apply the far-field boundary, the circular domain surrounding the airfoil of unit chord was chosen to be of radius $50C$, fig. 1(a).

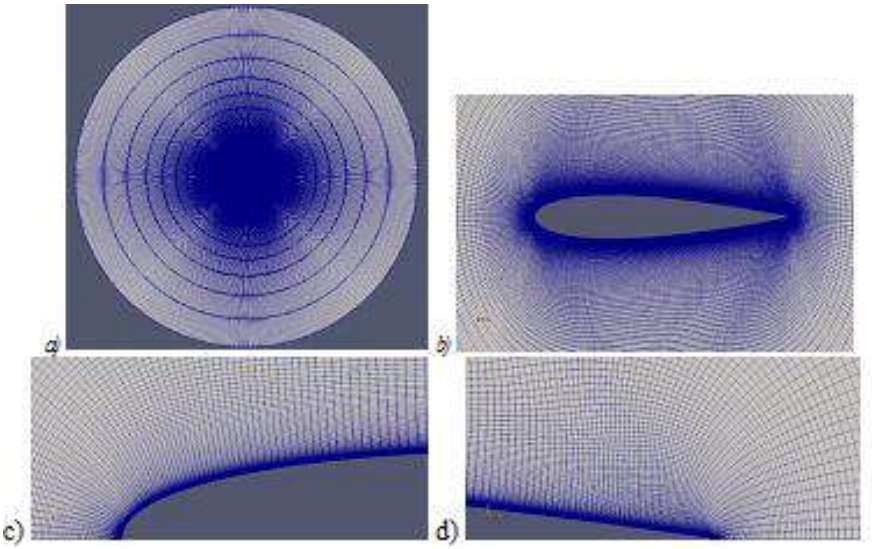

Fig. 1 (a) O-Grid computational domain mesh; (b) Internal domain mesh; (c) Leading edge spans; (d) Trailing edge spans

To improve the accuracy of the results and to reduce the consumed time, O-Grid approach is used to generate a 2-D multi zone structured mesh, the resulted mesh is divided depending on its discretization density into two different zones, fine and coarse mesh.

Fine mesh forms the internal zone of the computational domain, extends along a distance of $0.3C$ perpendicularly to airfoil connectors with a growth rate of 1.01, so the mesh is fine next to wing and relatively coarser far away fig. 1(b). Benefiting from Open Foam improved wall functions, and to achieve a balance between computation speed and accurately capturing the linear viscous sublayer, the minimum wall normal distance is maintained to be 0.0001 and therefore maximum y^+ values of the simulations at various angle of attack will be in the order of $y^+ < 10$.

Airfoil connectors were divided into 405 points. To improve aspect ratio as well as mesh distribution around the wing, taking into consideration the selected values of growth rate and wall distance, the leading and trailing edge spans were chosen to be identical as 0.0001, as shown in fig. 1(c),(d).

The second zone is a coarser mesh, encloses the internal zone with a growth rate of 1.3 normal to airfoil connectors, its minimum first cell high was chosen carefully to ensure mesh continuity and to avoid, as much as possible, solver convergence errors.

It is believed that the selected growth rate ratio, coupled with the first cell high and edges span are convenient to make mesh fine enough in vicinity of airfoil, and to solve the flow well around it. The selected values consider the conditions of the physical phenomena we simulate $\Re = 10^6$, in addition to the modeling software and boundary conditions we plan to use.

2.4 Numerical settings

The open-source CFD software package OPENFOAM was chosen to solve the incompressible Reynolds-averaged Navier-Stokes (RANS) equations. being an open-source package, having a wide variety of boundary-conditions, wall-functions, finite volume solution and schemes gives it a great advantage of managing, controlling and detecting the whole simulation processes. all the simulation settings were set corresponding to the characters and circumstances of the experiment [14].

SIMPLE, a pressure-velocity coupling algorithm based on finite volume method was used. Turbulence is modeled using the 2-equation transition turbulence model K-ω SST and the discretization of turbulence model equations was made using the upwind second order scheme, in order to obtain high accuracy, the convergence criterion is set at $1 \cdot 10^{-6}$.

according to the environment of the experimental work, the free stream temperature is $293.15K$, therefore the corresponding value of fluid kinematic viscosity ν is set to $1.516 \cdot 10^{-5}$.

The consistent boundary conditions, freestreamPressure and freestreamVelocity, were used simultaneously to give appropriate condition for pressure and speed at far stream based on the velocity orientation. At far stream, atmospheric pressure of uniform value 0 is applied, Reynolds number used in experiments has a value of $\Re = 10^6$, so the free-stream velocity (U_∞) is set to constant uniform value of (15.16,0,0) m/s and its components for different angles of attack α are calculated by simple formulas $V_x = V \cdot cos(\alpha)$, $V_y = V \cdot sin(\alpha)$. zeroGradient and no-Slip conditions are applied to pressure and velocity along the airfoil surface respectively. lowReWallFunction was used for k and ν_t while omegaWallFunction was used for ω.

The turbulence intensity Tu and eddy viscosity ratio μ_t/μ have been set at the inlet to 0.1% and 0.01 respectively, which give the kinematic turbulent energy k and turbulent dissipation ω at far-stream boundary a value of $0.0003447[J/kg]$, $2274[1/s]$ respectively, during the simulation operation their values have been systematically changed. Turbulence length scale has remained constant $136 \cdot 10^{-7}[m]$ and was not changed for all tested cases since its variation has not any considered effects on final results [21,22].

2.5 Grid convergence analysis

To investigate the discretization errors and to validate the dependency of simulation results on the computational grid, the simulation was performed on a series of three levels of successively finer grids, a coarse, medium and fine mesh.

In order to distinguish the discretization error from iterative convergence and computer rounding errors the grid refinement ratio in x direction selected to be of $r = 2$, on other hand this allows the Grid Convergence Index (GCI) to be employed [23,24,25]. hence the number of nodes along the whole airfoil surface would be 200, 400 and 800 nodes respectively, the drag coefficient C_d, the most sensitive variable [27], at $\alpha = 0°$ and $\alpha = 8°$ was set as the criterion for mesh dependency.

A grid convergence was performed, the order of grid convergence P was calculated and found to be of second-order $P = 1.82 \approx 2$ which is close to the theoretical order of convergence. The reduction of the discretization error as a function of grid size is shown in fig. 2. Richardson's extrapolation spatial convergence of CFD simulation was performed on the finest two grids to

predict the true value of the drag coefficient at zero-grid spacing based on order of convergence, and found to be 0.01 (see table 1). The grid convergence indexes GCI for fine-medium grid and medium-coarse grid were calculated using a safety factor for three grids $fs = 1.25$ and found that $GCI_{12} = 0.146$, $GCI_{23} = 0.521$ and they achieve the relation $GCI_{23}/(r^p \cdot GCI_{12}) = 1.003$ which is approximately one, and indicates that we are asymptotically approaching a converged answer, and thus our solution is definitely grid independent (see table 1).

Table 1. Studying the Computational grid convergence for drag coefficient at $\alpha = 0°, 8°$

Angle of Attack	fine mesh	medium mesh	coarse mesh	Order of Grid Convergence P
0	0.010281	0.010250	0.010142	1.824499
8	0.015261	0.015087	0.014468	1.822492
Richardson extrapolation	GCI_{12}%	GCI_{23}%	$\frac{GCI_{23}}{(r^p \bullet GCI}$	Richardson extrapolation
0.010293	0.146650	0.520965	1.002991	0.010293
0.015330	0.564287	2.018959	1.011585	0.015330

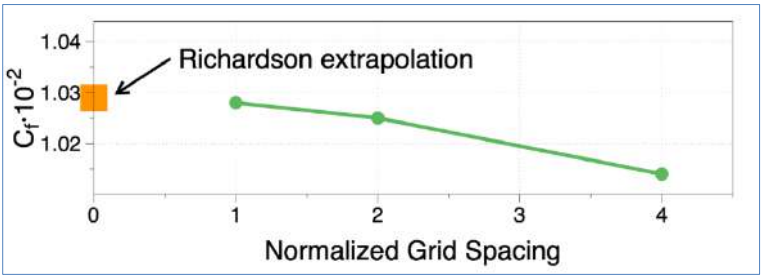

Fig. 2. Grid convergence study

Furthermore, the drag coefficient C_d, the most sensitive variable [26], and lift coefficients (C_l) of the airfoil at moderate Reynold number are compared for the three grids in order to further investigate the grid dependence of the results (see fig. 3). It can clearly be seen that the ability of predicting increases as mesh size increases:

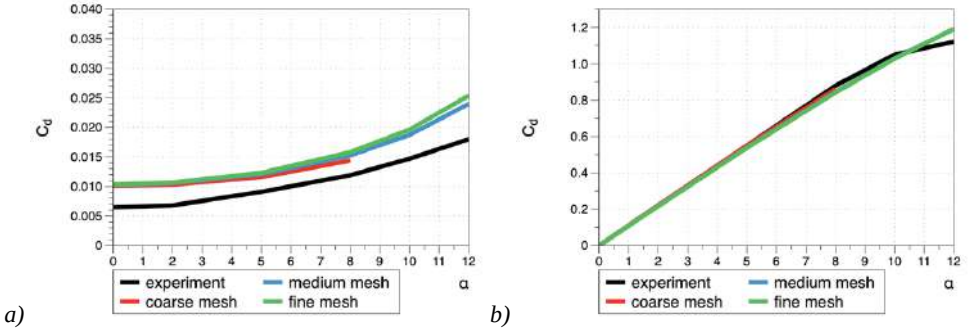

Fig. 3. Drag and lift curves for the three grids (coarse, medium, fine.

The maximum absolute difference is 0.0004 at $\alpha = 8°$ which results in less than 3 % difference in drag coefficient. The difference which can be observed can be attributed to a slight improvement in the prediction of flow separation on the airfoil which also has a very limited effect on the airfoil behavior.

Based on these results the medium grid was selected for the rest of the calculations, and it is believed that the initial 400 nodes grid was quite enough to capture the flow.

3. Results and discussion

The onset of the transition phase is evidenced by the apparent changes in the behavior of skin-friction curve C_f. The minimum local value in the friction curve indicates the start of the transition phase, the emergence of bubbles of turbulence, its growth in the laminar layer, its explosion and subsequent fusion, thus announcing manifestation of the fully developed turbulent flow.

The huge computational possibilities and vast computing resources available at laboratories of SMTU university allowed conducting a digital modeling and simulation of flow around 2-D NACA0012 profile. The simulations were done for a series of far-stream turbulence intensities and eddy viscosity ratios to indicate their effects on the onset and length of transition phase for airfoil located in viscous incompressible fluid, viscosity $\nu = 1.516$ at temperature $293.15K$, at various angle of attack and constant $\Re = 10^6$.

3.1 Effects of eddy viscosity ratio on onset and length of transition over the airfoil

Studying the effects of eddy viscosity ratio can be done by examine the behavior of skin friction coefficient curve at different value of μ_t/μ. Generally, μ_t/μ has a local effect on the skin friction coefficient (C_f), which maintains itself along the length of the wing, except for a very limited area of the wing surface along it the transition process take place. Using higher values for μ_t/μ at the far boundary increases the value of skin friction coefficient (C_f) in this area, as well as shifting the onset transition point towards the front edge of the wing.

Fig. 4(a) shows the curves of the skin friction coefficient variation along a section of a two-dimensional airfoil NACS0012 located at angle of attack $\alpha = 0°$ in a non-compressible viscous flow field at turbulence intensity $Tu = 0.1$ and various values of μ_t/μ.

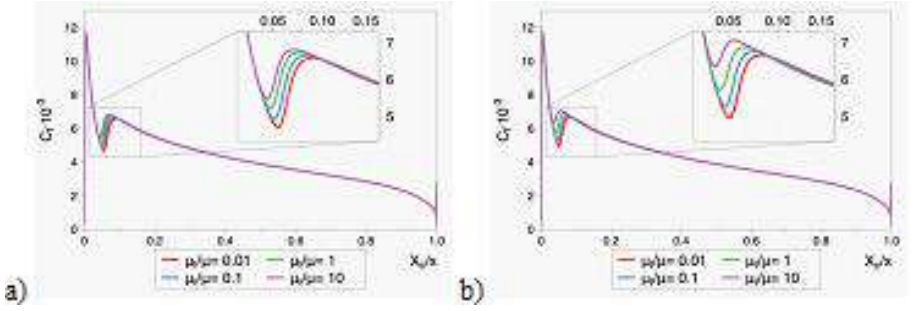

Fig. 4. Skin friction coefficient curves on NACA0012 surface at angle of attack 0 Degree, for various values of far-stream turbulence variables:
a) Tu=0.1, b) Tu=6

The curves of fig. 4(a) show that increasing the Eddy viscosity ratio at a certain value of Turbulence intensity leads to shift onset transition location towards leading edge of the airfoil. At a low value of far stream turbulence intensity $Tu = 0.1$, increasing μ_t/μ from 0.01 to 10 at moderate values of Reynolds number $\Re = 10^6$ leads to shift the transition point of flow over NASA 0012 surface towards leading edge from 0.0558 to 0.04387, which represents a change of 1.19% of chord length. This displacement is accompanied by an increase in the value of skin friction coefficient from $4.681 \cdot 10^{-3}$ to $5.457 \cdot 10^{-3}$, a relative increase of 16.57%, and a decrease in the length of the transition.

To investigate generalization of the reported conclusions, the simulation was repeated at higher value of far stream turbulence intensity, $Tu = 6$. The resulted skin friction coefficients at different values of eddy viscosity ratio were plotted on the right side. The curves of fig. 4(b) ensure that C_f curves maintain its behavior at high values of turbulence intensity too, except that the effect of μ_t/μ is more pronounced, this means that the ability of μ_t/μ to influence the skin friction coefficient is improved with the increase in the turbulence intensity, at higher disturbance values, the same increase in μ_t/μ causes a greater increase in the friction coefficient. Curves of Figure 6,b at $Tu = 6$ confirm that an increase in μ_t/μ value from 0.01 to 10 leads to shifting the transition point from 0.0556 to 0.0409694, which represents a change of 1.46% of the airfoil chord. This change in practice is 0.27% higher than using the low value of the turbulence intensity $Tu = 0.1$ at far boundary. The increase in μ_t/μ from 0.01 to 10 leads to an increase of C_f from $4.69702 \cdot 10^{-3}$ to $5.71491 \cdot 10^{-3}$ which constitutes 21.67% of the value of the friction coefficient. If compared with the value of the change at small values of turbulence intensity $Tu = 0.1$, we will find that this increase is 5.1%

3.2 Effect of turbulence intensity on onset and length of transition over the airfoil

Skin friction coefficient at certain value of eddy viscosity ratio were plotted for different values of turbulent intensity, see fig. 5.

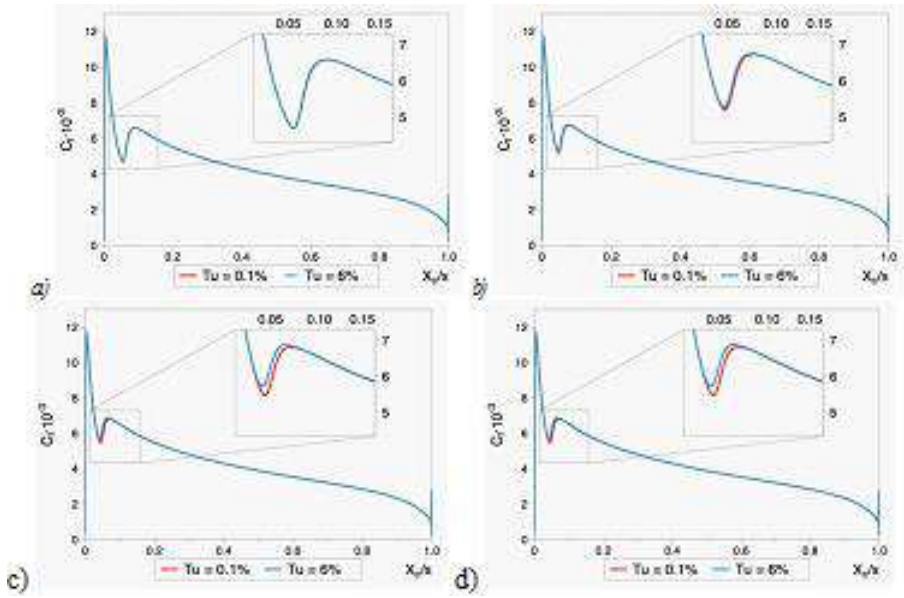

Fig. 5. Skin friction coefficient curves on NACA0012 surface at angle of attack 0 Degree, at certain value of eddy viscosity ratio and different values of far-stream turbulence intensity: a) μ_t/μ=0.01, b) μ_t/μ=0.1, c) μ_t/μ=1, d) μ_t/μ=10

Analysis of shape curves indicates that increased intensity of turbulence slightly excites the layer instability over the airfoil surface and therefore increases the rates of disturbance in the sheer layer causing upstream shift of transition onset as well as slightly increase in transition length.

Despite the values of turbulence intensity Tu vary widely at boundaries, their values next to the wing are convergent and the modeling values of the transition onset are therefore close to each other. This behavior is related to the ability of the grid to solve decay equations at different values of turbulence variables. When high values are used at boundary, the decay rates

modeled by grid are dramatically decreased while gradually decreased when low values are used.

3.3 Effects of angle of attack on onset and length of transition over the airfoil.

To study angle of attack effects on the characteristics of the transition phase, the previous simulation was repeated at the same values of far stream turbulence intensity and eddy viscosity ratios but using different values for angle of attack.

Curves of fig. 6, 7 illustrate the variation of the skin friction coefficient along a section of a two-dimensional airfoil NACS0012 located at angle of attack $\alpha = 8°$,$\alpha = 10°$, respectively, immersed in a non-compressible viscous flow field at far stream turbulence intensity $Tu = 0.1$ (left side), and $Tu = 6$ (right side) for various values of μ_t/μ.

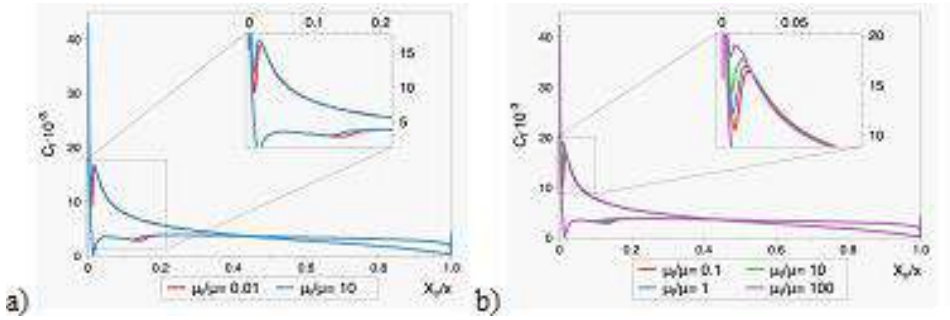

Fig. 6. Skin friction coefficient curves on NACA0012 surface at angle of attack 8 Degree, for various values of far-stream turbulence variables:

a) Tu=0.1, b) Tu=6

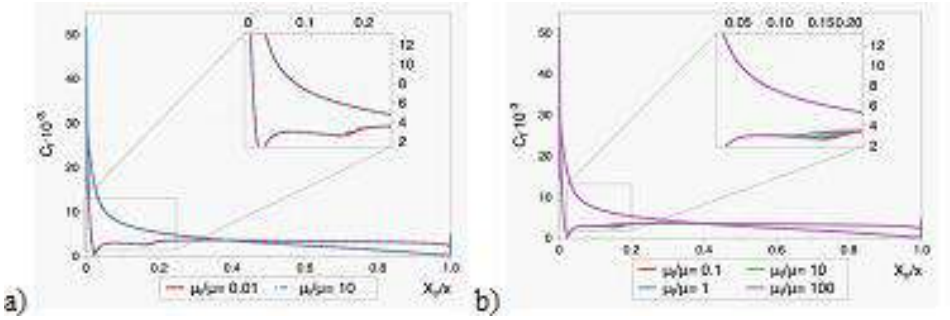

Fig. 7. Skin friction coefficient curves on NACA0012 surface at angle of attack 10 Degree, for various values of far-stream turbulence variables:
a) $Tu = 0.1$, *b)* $Tu = 6$

By comparing the curves of the fig. 4, 6, and 7 at the corresponding values of the turbulence intensity, we observe that skin friction coefficient curves maintain their behavior at the various values of angle of attack, but as angle of attack increases the effects of μ_t/μ on the transition onset point fades. Curves indicate that although the μ_t/μ is increased from 0.01 to 100, the transition start at approximately the same point, which in turn shifts towards the leading edge of the wing as angle of attack increases.

During numerical simulation process, it was observed that the simultaneous use of small values for Tu with large values for μ_t/μ causes convergence problems, which explain absence of some skin friction coefficients curves at $Tu = 0.1$ from results of different angle of attack.

Reviewing previous figures indicates that increasing angle of attack requires increase in Tu values to overcome convergence problems which occur at low values of Tu at the same angle.

Finally, we should point out that the super-fining of the grid cause losing its sensitivity to changes in the disturbance variables at far boundaries, so that the turbulent model k-w SST cannot predict the transition point. Fig. 8 shows simulated skin friction coefficient on super fine mesh at different angle of attack and various values of far stream turbulence variables.

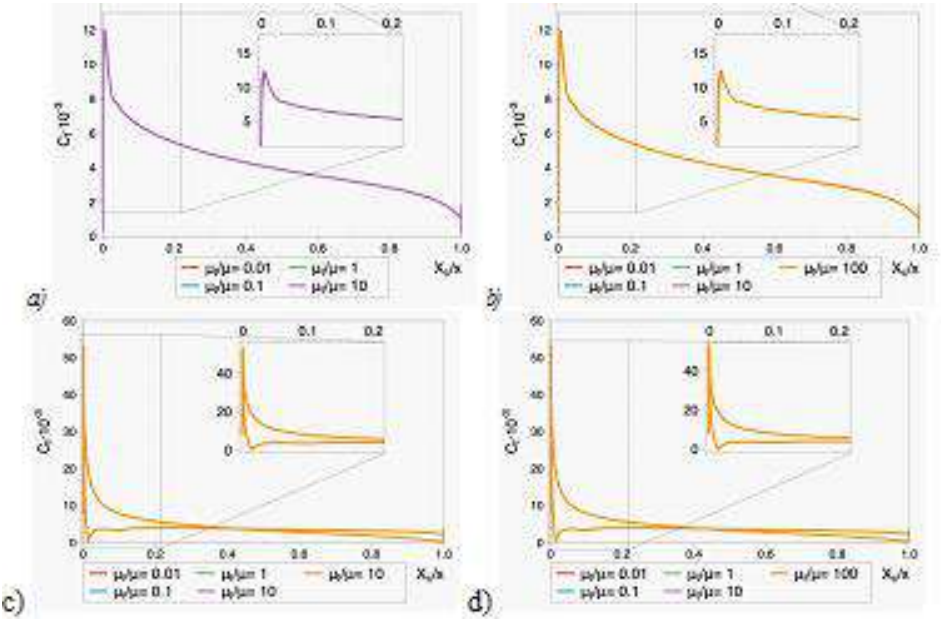

Fig. 8. Skin friction coefficient curves on NACA0012 surface simulated on super-fining grid at different angle of attack, for various values of far-stream turbulence variables:
a) $\alpha = 0°$, $Tu = 0.1$, *b)* $\alpha = 0°$, $Tu = 6$, *c)* $\alpha = 8°$, $Tu = 0.1$, *d)* $\alpha = 8°$, $Tu = 6$

4. Conclusions

RANS Simulations were carried out on a NACA0012 airfoil using the software OpenFoam and turbulent model k-ω SST in order to study effects of turbulence variables at far stream on transition flow characters over the airfoil at moderate Reynolds number of $\Re = 10^6$. Analyses and numerical investigation leads to the following conclusions.

- Far stream turbulence variables have significant effects on transition flow character over airfoil at moderate Reynolds number.
- Effect of turbulence intensity and eddy viscosity ratio at far stream on the skin friction coefficient is negligible, except for a limited area where the transition occurs.
- Using high values for Tu gives a greater ability to control the desired position of the transition phase on the wing surface, which is done by changing the μ_t/μ on a wide range from 0.01 to 100 without convergence problems.
- Using small values for Tu causes convergence problems especially if used with large values of μ_t/μ which in turn limits the controlling ability.
- Increasing Tu shifts the transition onset a little towards the leading edge and has a negligible effect on transition length and skin friction coefficient at transition phase.
- Increasing μ_t/μ shifts the transition onset towards the leading edge, as well as increases skin friction coefficient at transition phase, increases transition length.

- Increasing angle of attack reduces sensitivity to changes in far stream turbulence variables.
- Increasing angle of attack requires simultaneous increase in turbulence intensity at far stream, so solver reaches a stable solution without convergence problems over a wide range of μ_t/μ values.
- Over-fining of the computational domain causes grid to lose its sensitivity to changes in disturbance values at far boundaries.

References / Список литературы

[1]. Rui Miguel Alves Lopes. Calculation of the flow around hydrofoils at moderate Reynolds numbers. Master thesis. Instituto Superior Tecnico, Lisboa, Portugal, 2015.
[2]. Robert Edward Mayle. The Role of Laminar-Turbulent Transition in Gas Turbine Engines. ASME Journal of Turbomachinery, vol. 113, no. 4, 1991, pp. 509-537.
[3]. Emmons H.W. The laminar-turbulent transition in a boundary layer. Part I. Journal of the Aeronautical Sciences, vol. 18, 1951, pp. 490-498.
[4]. Robin Blair Langtry. A correlation-based transition model using local variables for unstructured parallelized CFD codes. PhD Thesis, Universität Stuttgart, Fakultät Maschinenbau, 2006.
[5]. F.R. Menter, R.B. Langtry, S.R. Likki, Y.B. Suzen, P.G. Huang, S. Völker. A Correlation-Based Transition Model Using Local Variables – Part I: Model Formulation. ASME Journal of Turbomachinery, vol. 128, no. 3, 2006, pp. 413-422.
[6]. Ö.S. Özçakmak, H.A. Madsen, N.N. Sørensen, J.N. Sørensen, Andreas Fischer, C. Bak. Inflow Turbulence and Leading Edge Roughness Effects on Laminar-Turbulent Transition on NACA 63-418 Airfoil. Journal of Physics: Conference Series, vol. 1037, issue 2, 2018, article no. 022005.
[7]. R.E. Mayle. The Role of Laminar-Turbulent Transition in Gas Turbine Engines. ASME Journal of Turbomachinery, vol. 113, no. 4, 1991, pp. 509-537
[8]. Butler R.J., Byerley A.R., VanTreuren K., and Baughn J.W. The Effect of Turbulence Intensity and Length scale on Low-pressure Turbine Blade Aerodynamics. International Journal of Heat and Fluid Flow, vol. 22, issue 2, 2001, pp. 123-133.
[9]. Mueller T.J., Pohlen L.J. The Influence of Free-Stream Disturbances on Low Reynolds Number Airfoil Experiments. Experiments in Fluids, vol. 1, issue 1, 1983, pp 3–14.
[10]. Hoffmann J.A. Effects of Freestream Turbulence on the Performance Characteristics of an Airfoil. AIAA Journal, vol. 29, no. 9, 1991, pp. 1353-1354.
[11]. Huang R.F. and Lee H.W. Effects of Freestream Turbulence on Wing-Surface Flow and Aerodynamic Performance. Journal of Aircraft, vol. 36, no. 6, 1999, pp. 965-972.
[12]. Shao-wu Li, Shu Wang, Jian-ping Wang, Jian-chun Mi. Effect of turbulence intensity on airfoil flow: Numerical simulations and experimental measurements. Applied Mathematics and Mechanics, vol. 32, 2011, pp. 1029-1038.
[13]. Cao Ning. Effects of turbulence intensity and integral length scale on an asymmetric airfoil at low Reynolds numbers. Master Thesis. University of Windsor, Canada, 2010.
[14]. Sheldahl R.E. and Klimas P.C. Aerodynamic characteristics of seven symmetrical airfoil sections through 180-degree angle of attack for use in aerodynamic analysis of vertical axis wind turbines. Technical Report, U.S. Department of Energy, Office of Scientific and Technical Information, 1981.
[15]. Menter F.R., Kuntz M., Langtry R. Ten Years of Industrial Experience with the SST Turbulence model. In Proc. of the Fourth International Symposium on Turbulence, Heat and Mass Transfer, 2003, pp. 624 – 632.
[16]. Daniel Lindblad. Implementation and run-time mesh refinement for the k−ω SST DES turbulence model when applied to airfoils. Project work. Chalmers University of Technology, 2014.
[17]. D.B. Spalding. A single formula for the “law of the wall”. Journal of Applied Mechanics, vol. 28, issue 3, 1961, pp. 455-458.
[18]. Liu S.N. Implementation of a Complete Wall Function for the Standard k-ε Turbulence Model in OpenFOAM 4.0. Technical report. Chalmers University of Technology, 2016.
[19]. Fangqing Liu.: A Thorough Description of How Wall Functions are Implemented in OpenFOAM. Technical report. Chalmers University of Technology, 2016.
[20]. Georgi Kalitzin, Gorazd Medic, Gianluca Iaccarino, Paul Durbin. Near-wall behavior of rans turbulence models and implications for wall functions. Journal of Computational Physics, vol. 204, issue 1, 2005, pp. 265–291.

[21]. Robert J. Butler, Aaron R. Byerley, Kenneth VanTreuren, James W. Baughn. The effect of turbulence intensity and length scale on low-pressure turbine blade aerodynamics. International Journal of Heat and Fluid Flow, vol. 22, issue 2, 2001, pp. 123-133.

[22]. Cao Ning. Effects of turbulence intensity and integral length scale on an asymmetric airfoil at low Reynolds numbers. Master thesis. University of Windsor, Canada, 2010.

[23]. Roache P.J. Perspective: A Method for Uniform Reporting of Grid Refinement Studies. Journal of Fluids Engineering. Journal of Fluids Engineering, vol. 116, no. 3, 1994, pp. 405-413.

[24]. Roache P.J. Quantification of Uncertainty in Computational Fluid Dynamics. Annual Review of Fluid Mechanics, vol. 29, 1997, pp. 123-160.

[25]. Ali Rami, Tryaskin N.V. Effects of turbulence variables on transition flow characteristics over NACA0012 airfoil. Marine intellectual technologies. № 3(45), vol. 2, 2019, pp. 39-44 (in Russian) / Али Рами, Тряскин Н.В. Влияние параметров турбулентности на характеристики переходного режима течения при обтекании профиля NACA 0012. Морские интеллектуальные технологии. № 3(45), том 2, 2019 г., стр. 39-44.

[26]. F. R. Menter, M. Kuntz, and R. Langtry. Ten Years of Industrial Experience with the SST Turbulence Model. In Proc. of the 4th International Symposium on Turbulence, Heat and Mass Transfer, 2003, pp. 625-632.

Информация об авторах / Information about authors

Рами АЛИ – аспирант кафедры гидроаэромеханики и морской акустики. Научные интересы: численное моделирование, интенсивность турбулентности, уравнения Рейнольдса.

Rami ALI is a PhD student of the Department of Hydroaeromechanics and Marine Acoustics. Research interests: numerical modeling, turbulence intensity, Reynolds equations.

Никита Владимирович ТРЯСКИН – кандидат технических наук, доцент кафедры гидроаэромеханики и морской акустики. Научные интересы: гидродинамика, турбулентность, уравнения Рейнольдса, численное моделирование.

Nikita Vladimirovich TRYASKIN – candidate of technical sciences, associate professor of the Department of Hydroaeromechanics and Marine Acoustics. Research interests: hydrodynamics, turbulence, Reynolds equations, numerical simulation.

DOI: 10.15514/ISPRAS-2019-31(6)-14

CC BY

Исследование влияния регулярных магнитных полей на течения во внешних кольцах галактик

[1] *Е.А.Михайлов, ORCID: 0000-0002-9747-4039 <ea.mikhajlov@phyics.msu.ru>*
[2, 3] *И.Н. Сибгатуллин, ORCID: 0000-0003-2265-3259 <sibgat@ocean.ru>*

[1] *Московский государственный университет имени М.В. Ломоносова,*
119991, Россия, Москва, Ленинские горы, д. 1
[2] *Институт океанологии им. П.П. Ширшова РАН,*
117218, Россия, г. Москва, Нахимовский пр., д. 36
[3] *Институт системного программирования им. В.П. Иванникова РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В настоящее время практически не вызывает сомнений наличие крупномасштабных магнитных полей в спиральных галактиках. Их эволюция описывается с помощью механизма динамо. Динамо основано на совместном действии дифференциального вращение и альфа-эффекта. В случае галактик, как правило, используется планарное приближение, связанное с тем, что галактический диск достаточно тонкий. Отдельный интерес представляют так называемые внешние кольца галактик. Исследование магнитных полей в них имеет некоторые существенные трудности. Во-первых, полагать, что внешнее кольцо является тонким по сравнению с радиальными размерами, уже нельзя. Во-вторых, необходим учет влияния магнитного поля на турбулентные движения межзвездной среды. В настоящей работе представлены результаты моделирования магнитных полей, а также продемонстрировано влияние магнитного поля на турбулентные движения межзвездной среды.

Ключевые слова: магнитные поля; турбулентность; внешние кольца галактик; динамо

Для цитирования: Михайлов Е.А., Сибгатуллин И.Н. Исследование влияния регулярных магнитных полей на течения во внешних кольцах галактик. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 215-224. DOI: 10.15514/ISPRAS–2019–31(6)–14

Благодарности: Работа была выполнена при поддержке РФФИ (проекты 18-32-00124 и 18-02-00085).

Research of influence of regular magnetic fields on flows in outer rings of galaxies

[1] *E.A.Mikhailov, ORCID: 0000-0002-9747-4039 <ea.mikhajlov@physics.msu.ru>*
[2,3] *I.N. Sibgatullin, ORCID: 0000-0003-2265-3259 <sibgat@ocean.ru>*

[1] *Lomonosov Moscow State University,*
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[2] *Shirshov Oceanology Institute of the Russian Academy of Sciences,*
36, Nakhimovsky prospect, Moscow, 117218, Russia
[3] *Ivannikov Institute for System Programming of the Russian Academy of Sciences,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Now it is no doubt that there are large-scale magnetic fields in the spiral galaxies. Their evolution is described by the dynamo mechanism which is based on joint action of differential rotation

and alpha-effect. For galaxies usually the no-z approximation is used, which takes into account that the galaxy disk is quite thin. It is necessary to study so-called outer rings of galaxies. Studying the magnetic fields there is quite difficult. Firstly, we cannot say that the outer ring is thinner than the radial lengthscale. Secondly, it is necessary to take into account the influence of the magnetic field on the turbulent motions. In this work we present the results of the magnetic field modeling, and also demonstrate the influence of the magnetic field on the turbulent motions of the interstellar medium.

Keywords: magnetic fields; turbulence; outer rings of galaxies; dynamo

For citation: Mikhailov E.A., Sibgatullin I.N. Research of influence of regular magnetic fields on flows in outer rings of galaxies. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 215-224 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-14

Acknowledgements. This work was supported by RFBR (projects 18-32-00124 and 18-02-00085).

1. Введение

В настоящее время тот факт, что большое количество спиральных галактик обладает крупномасштабными магнитными полями величиной несколько микрогаусс, является твердо установленным и практически не вызывает сомнений [1, 2, 3, 4, 5]. С точки зрения наблюдательной астрономии их наличие было впервые обнаружено с помощью исследований спектра синхротронного излучения [6, 7]. В настоящий момент основным методом исследования крупномасштабных магнитных полей является исследование фарадеевского вращения плоскости поляризации электромагнитных волн, которое измеряется с помощью современных радиотелескопов. Можно показать, что вращение плоскости поляризации пропорционально составляющей магнитного поля, параллельной лучу зрения, умноженной на квадрат длины волны [8, 9]. Исследуя плоскость поляризации на разных длинах волн, можно восстановить характерную структуру магнитного поля для различных галактических объектов [10, 11, 12, 13, 14, 15].

С теоретической точки зрения генерация магнитных полей в астрофизических объектах (как галактиках, так и звездах, планетах и т.д.), как правило, описывается с помощью механизма динамо, описывающего переход энергии движения среды в энергию магнитного поля [16, 17]. Различают мелкомасштабное магнитное поле, имеющее характерные линейные масштабы изменения порядка нескольких десятков килопарсек, а также регулярные структуры магнитного поля, являющиеся результатом усреднения по соответствующим пространственным масштабам. Если говорить о крупномасштабных структурах магнитного поля, имеющих ключевое значение в понимании эволюции различных процессов, его генерация обусловлена так называемым механизмом динамо, являющегося результатом совместного действия альфа-эффекта, характеризущего закрученность турбулентных движений, и дифференциального вращения, которое связано с нетвердотельностью вращения галактики вокруг своей оси [2]. Им противодействует турбулентная диффузия, которая стремится разрушить крупномасштабные структуры магнитного поля. Ввиду этого механизм динамо является пороговым: генерация магнитного поля возможна только в случае, если совместная интенсивность альфа-эффекта и дифференциального вращения достаточно высока, чтобы противостоять диссипативным процессам [2]. Она характеризуется так называемым динамо-числом, включающим в себя полутолщину диска, скорость турбулентных движений и угловую скорость вращения. Как правило, рост магнитного поля возможен в случае, если оно превышает определенное значение, называемое критическим [2].

В настоящее время большой интерес представляют не только сами спиральные галактики, но и так называемые внешние кольца, которые окружают некоторые из них [18, 19, 20]. Они имеют небольшую ширину, и могут лежать как в плоскости диска, так и быть перпендикулярными к нему (отметим, что промежуточные положения являются

достаточно редкими). В них имеют место аналогичные эффекты, которые могут также обуславливать действие механизма динамо и обеспечивать наличие в них магнитных полей. Нельзя также не отметить ряд наблюдательных предпосылок к наличию магнитных полей во внешних кольцах галактик. Тем не менее, процесс теоретического исследования магнитных полей во внешних кольцах наталкивается на ряд достаточно серьезных трудностей.

Дело в том, что уравнение динамо среднего поля (также называемое уравнением Штеенбека – Краузе – Рэдлера) является достаточно сложным для решения как с аналитической, так и с вычислительной точки зрения. Как правило, в случае галактик для этого применяются различные приближения, позволяющие существенно упростить процесс решения. Одним из самых популярных подходов является так называемое планарное приближение [21, 22, 23]. Оно основано на том факте, что галактический диск достаточно тонкий. По этой причине можно приближенно считать, что магнитное поле лежит в плоскости галактического диска. Кроме того, некоторые частные производные магнитного поля в вертикальном направлении можно заменить на алгебраические выражения, а другие – восстановить с помощью условия соленоидальности. Это позволяет сократить число неизвестных функций до двух, и существенно упростить процесс решения задачи. Первые оценки для магнитных полей во внешних кольцах галактик были получены с использованием планарного приближения [24]. Хотя они и дают достаточно разумную с качественной точки зрения картина, нельзя не отметить, что изначальные предпосылки, которые использовались при построении модели, оказываются неверны в случае исследования внешних колец. Так, полутолщину кольца уже нельзя считать малой на фоне его размеров в радиальном направлении. Это ставит нас перед необходимостью использования других моделей для магнитного поля. Один из возможных подходов – использование модели динамо в торе [25, 26, 27, 28, 29]. Она исходит из достаточно разумных представлений об осесимметричности решения, и использует представление магнитного поля в виде комбинации из тороидальной компоненты, а также ротора тороидальной части векторного потенциала. В таком случае возможно получить различные решения для магнитного поля, которые аккуратно учитывают вертикальную структуру, и дают возможность вычислить поля различной симметрии. Так, динамо в торе предусматривает возможность генерации не только квадрупольного, но и дипольного магнитного поля [28, 29]. Хотя оно и генерируется лишь в случае крайне интенсивных движений межзвездной среды, нельзя не отметить, что при использовании планарного приближения, популярного в случае галактического магнитного поля, принципиально (в силу построения модели) возможна генерация лишь квадрупольных, симметричных относительно экваториальной плоскости структур.

Другая сложность связана с необходимостью изучения влияния магнитного поля на турбулентные движения межвездной среды, которые должны очевидно иметь место как в основной части галактики, так и во внешних кольцах. Дело в том, что достаточно быстро магнитное поле приобретает энергию, сопоставимую или даже превышающую типичное значение энергии турбулентных движений, которые изначально присутствуют в межзвездной среде. Поэтому логично предположить, что по мере роста магнитного поля его влияние на турбулентность будет заметно возрастать. Можно даже предположить, что турбулентность в том виде, в котором она существует во внешних кольцах галактик, обусловлена именно действием магнитного поля. Дополнительным доводом в пользу этого является то, что в отличие от основной части галактики, где турбулентность связана, по-видимому, с наличием взрывов сверхновых, во внешних кольцах их количество заметно меньше, что требует поиска нового источника для турбулентных движений, одним из которых может быть магнитное поле.

Основной целью нашей работы является представление одного из возможных представлений о характере эволюции магнитного поля во внешних кольцах галактик,

связанным с использованием модели динамо в торе. После этого мы планируем обсудить характерную структуру магнитного поля, которая получается при использовании соответствующих уравнений. Затем мы проводим процесс решения уравнений для движения межзвездной среды, в которые включено действие электромагнитной силы, обусловленной влиянием магнитного поля, полученного в нашей модели. Представлены характерные картины турбулентных движений при наличии магнитного поля. Исследуется спектр турбулентности, которая связана с наличием крупномасштабного магнитного поля.

После этого обсуждается возможность применения полученных решений для изучения как внешних колец галактик, так и других объектов с качественно сходной структурой - таких как, к примеру, аккреционные диски, образующиеся около массивных объектов – белых карликов, черных дыр и т.д.

2. Генерация магнитного поля

Галактическое магнитное поле $\vec{H}$ включает в себя две основных составляющих :

$$\vec{H} = \vec{B} + \vec{b},$$

где $\vec{b}$ – мелкомасштабная часть, имеющая характерный масштаб изменения около 50 пк, а $\vec{B}$ – крупномасштабная составляющая, являющаяся результатом усреднения и меняющаяся достаточно плавно.

Для эволюции крупномасштабного магнитного поля можно записать уравнение динамо среднего поля, также называемое в астрофизической литературе уравнением Штеенбека – Краузе – Рэдлера:

$$\frac{\partial \vec{B}}{\partial t} = \mathrm{rot}\left[\vec{V}, \vec{B}\right] + \mathrm{rot}\left(\alpha \vec{B}\right) + \eta \Delta \vec{B},$$

где $\vec{V}$ - скорость крупномасштабных движений межзвездной среды,α- коэффициент, отвечающий за альфа-эффект, η - коэффициент турбулентной диффузии.

Как правило, предполагается, что крупномасштабные движения обусловлены вращением галактики (или внешнего кольца), поэтому можно считать, что:

$$\vec{V} = r\Omega \vec{e}_{\phi},$$

где Ω - угловая скорость вращения объекта.

Для альфа-эффекта часто используют следующее выражение:

$$\alpha = \frac{\Omega l^2 \mathrm{z}}{\mathrm{h}},$$

где h - полутолщина диска или кольца, l - характерная длина свободного пробега для частиц (как правило, составляющая величину от 50 до 100 пк).

При решении уравнений для эволюции крупномасштабной составляющей магнитного поля удобно представить его в виде следующей комбинации [25]:

$$\vec{B} = B\vec{e}_{\phi} + \mathrm{rot}\left(A\vec{e}_{\phi}\right),$$

где B – тороидальная часть магнитного поля, A – тороидальная часть векторного потенциала (ее ротор описывает в осесимметричном случае полоидальную часть магнитного поля объекта).

В таком случае уравнения для магнитного поля можно записать в следующей форме [25, 26, 27, 28, 29]:

$$\frac{\partial A}{\partial t} = \frac{\Omega l^2 z}{\mathrm{ha}} B + \eta \Delta A;$$
$$\frac{\partial B}{\partial t} = \Omega \frac{\partial A}{\partial z} + \eta \Delta B.$$

Достаточно удобно использовать безразмерные переменные, измеряя расстояния в радиусах кольца R, а времена в единицах $\frac{a^2}{\eta}$, где a – полуширина кольца. В таком случае уравнения можно переписать в следующем виде [27, 28]:

$$\frac{\partial A}{\partial t} = R_\alpha \mathrm{zB} + \lambda^2 \Delta A;$$
$$\frac{\partial B}{\partial t} = R_\omega \frac{\partial A}{\partial z} + \lambda^2 \Delta B;$$

где введены несколько безразмерных параметров [28, 29]:

$$R_\alpha = \frac{\Omega l^2 a^2}{\eta h^2};$$
$$R_\omega = \frac{\Omega a^2}{\eta};$$
$$\lambda = \frac{a}{R};$$

характеризующие альфа-эффект, дифференциальное вращение и диссипацию в плоскости диска.

В качестве граничных условий можно использовать следующие [25, 26, 27, 28, 29]:

$$B_{z=\pm\lambda/k} = B_{r=1\pm\lambda} = \frac{\partial A}{\partial n_{z=\pm\lambda/k}} = \frac{\partial A}{\partial n_{r=1\pm\lambda}} = 0.$$

Возможность генерации магнитного поля описывается с помощью так называемого динамо-числа, которое вводится как произведение коэффициентов, отвечающих за альфа-эффект и дифференциальное вращение [2]:

$$D = R_\alpha R_\omega.$$

Можно показать, что при $D > 30$ возможна генерация магнитных полей [29]. При больших значениях D становится возможной генерация полей не только квадрупольной, но и дипольной симметрии (рис. 1, штрихпунктирная линия показывает случай дипольного магнитного поля).

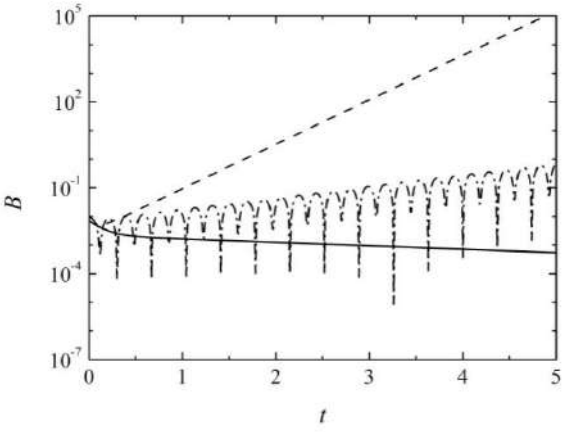

Рис. 1. Эволюция магнитного поля в линейном случае (сплошная линия показывает D=25, пунктирная – D=150, штрихпунктирная – D=900)
Fig. 1. Magnetic field evolution for linear case (solid line shows D=25, dashed – D=150, dot-dashed – D=900)

Разумно учитывать, что при приближении магнитного поля к уровню насыщения его рост будет замедляться. Это может быть учтено с помощью следующей модификации уравнений [27, 28, 29]:

$$\frac{\partial A}{\partial t} = R_\alpha \mathrm{zB}\left(1 - \frac{B^2}{B_0^2}\right) + \lambda^2 \Delta A;$$

$$\frac{\partial B}{\partial t} = R_\omega \frac{\partial A}{\partial z} + \lambda^2 \Delta B;$$

где B_0 – поле насыщения. Измеряя поле в единицах насыщения, можно записать уравнения в полностью безразмерной форме [29]:

$$\frac{\partial A}{\partial t} = R_\alpha \mathrm{zB}(1 - B^2) + \lambda^2 \Delta A;$$

$$\frac{\partial B}{\partial t} = R_\omega \frac{\partial A}{\partial z} + \lambda^2 \Delta B.$$

Характерные решения представлены на рис. 2. Структура квадрупольного магнитного поля показана на рис. 3, дипольного – на рис. 4.

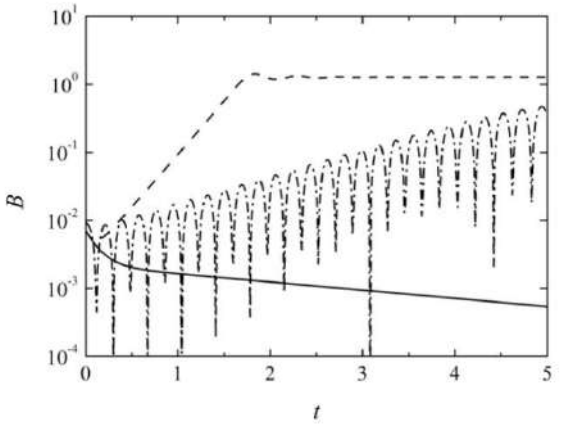

Рис. 2 Эволюция магнитного поля в нелинейном случае (сплошная линия показывает D=25, пунктирная – D=150, штрихпунктирная – D=900)
Fig. 2. Magnetic field evolution for nonlinear case (solid line shows D=25, dashed – D=150, dot-dashed – D=900)

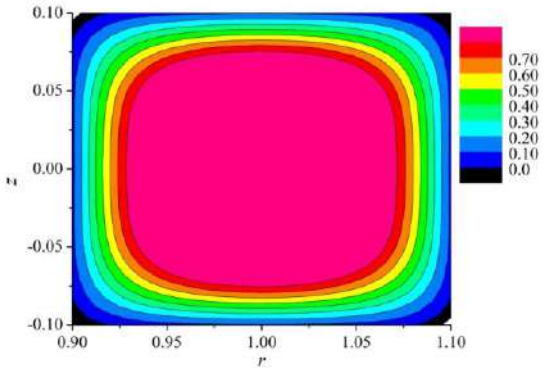

Рис. 3. Магнитное поле квадрупольной симметрии
Fig. 3. Magnetic field of quadrupolar symmetry

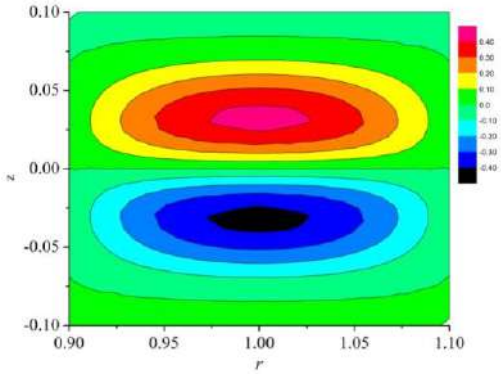

Рис. 4. Магнитное поле дипольной симметрии
Fig.4. Magnetic field of dypolar symmetry

В случае магнитного поля квадрупольной симметрии можно полагать, что его зависимость от координат будет иметь следующий вид:

$$B = B_0 \cos\left(\frac{\pi z k}{2\lambda}\right) \cos\left(\frac{\pi(r-R)}{2\lambda}\right);$$

где введено безразмерное число

$$k = \frac{a}{h}.$$

3. Влияние магнитного поля на турбулентные движения

При изучении влияния магнитного поля на турбулентные движения необходимо решать уравнение Навье – Стокса:

$$\frac{\partial \vec{v}}{\partial t} + (\vec{v}, \vec{\nabla})\vec{v} = -\frac{1}{\rho}\vec{\nabla}p + \vec{g} - \vec{f}_{\mathrm{Cor}} + \vec{f}_L + \beta \Delta \vec{v};$$

где $\vec{g}$ - сила тяжести, $\vec{f}_{\mathrm{Cor}}$ - сила Кориолиса, $\vec{f}_L$ - сила Лоренца, определяемая по правилу:

$$\vec{f}_L = \frac{1}{4\pi\rho}[\vec{B}, \mathrm{rot}\,\vec{B}],$$

где в качестве поля используются данные, представленные в предыдущем разделе.

Поскольку рассматривается сегмент внешнего кольца галактики, то расстояние от центра и, соответственно, радиус кривизны настолько велики, что в настоящей работе эффектами кривизны пренебрегается. Поэтому область течения принимается прямоугольной, с периодическими условиями в азимутальном направлении.

Известно, что сдвиговые течения могут являться линейно устойчивыми, в частности, плоскопараллельное течение Куэтта с линейным профилем скорости является устойчивым для конечных значений числа Рейнольдса. В галактических кольцах одним из дестабилизирующих факторов, способствующих формированию турбулентных режимов, может явиться последовательность взрывов сверхновых. Для моделирования локальных возмущений в качестве начального состояния был задан ряд локальных конвективных возмущений поля скорости.

В настоящей работе мы провели прямое численное моделирование турбулентных течений при дипольном и квадрупольном магнитном поле, ориентированном в азимутальном направлении, при умеренных числах Рейнольдса 10^4-10^5. Для прямого численного моделирования использовался метод спектральных элементов, представляющий из себя вариант метода конченых элементов. В каждом элементе

проводится ортогональное разложение по полиномам Лагранжа в точках Гаусса-Лобатто-Лежандра [30, 31]. Подобный подход позволяет избежать численной диффузии.

На рисунках ниже представлены представлены характерные турбулентные режимы в отсутствие магнитного поля (рис 5), дипольном (рис. 6) и квадрупольном (рис. 7) воздействии.

Рис. 5. Модуль скорости в отсутствие магнитного поля
Fig. 5. The velocity module in the absence of a magnetic field

Рис. 6. Модуль скорости при магнитном поле дипольной симметрии
Fig. 6. The velocity module in a magnetic field of dipole symmetry

Рис. 7. Модуль скорости при магнитном поле квадрупольной симметрии
Fig. 7. The velocity module in a magnetic field of quadrupole symmetry

Магнитное поле и сила Кориолиса изменяет мелкомасштабную структуру и формирование когерентных структур в турбулентном течении, их количественная оценка является предметом анализа в настоящее время.

4. Заключение

В настоящей работе был исследован процесс генерации магнитных полей во внешних кольцах галактик, а также их влияние на турбулентные движения. Полученные результаты могут представлять интерес не только для внешних колец, но и для других объектов, таких как, например, аккреционные диски.

Список литературы / References

[1]. Beck R., Brandenburg A., Moss D., Shukurov A., Sokoloff D. Galactic Magnetism: Recent Developments and Perspectives. Annual Review of Astronomy and Astrophysics, vol. 34, 1996, pp.155-206.
[2]. Beck R. Galactic and extragalactic magnetic fields. Space Science Reviews, vol. 99, 2001, pp. 243-260.
[3]. Arshakian T., Beck R., Krause M., Sokoloff D. Evolution of magnetic fields in galaxies and future observational tests with the Square Kilometre Array. Astronomy and Astrophysics, vol. 494, 2009, pp.21-32.
[4]. Beck R. Magnetic fields in galaxies. Space Science Reviews, vol. 166, 2012, pp.215-230.
[5]. Beck R. Magnetic fields in spiral galaxies. The Astronomy and Astrophysics Review, vol. 24, 2015, article id.4.
[6]. Ginzburg V.L. Radio astronomy and the origin of cosmic rays. In Paris Symposium on Radio Astronomy, IAU Symposium no. 9 and URSI Symposium no. 1, Stanford University Press, 1959, p. 589-594.
[7]. Beck R. Magnetic field structure from synchrotron polarization. European Astronomical Society Publications Series, vol. 23, 2007, pp.19-36.

[8]. Ginzburg V.L. The propagation of electromagnetic waves in plasmas. Pergamon Press, 1970, 535 p.
[9]. Зельдович Я.Б., Рузмайкин А.А., Соколов Д.Д. Магнитные поля в астрофизике. Москва–Ижевск, НИЦ «Регулярная и хаотическая динамика», Институт компьютерных исследований, 2006, 384 с. / Zeldovich, Ya.B., Ruzmaikin A.A., Sokoloff D.D. Magnetic fields in space. Moscow–Izhevsk, NIC «Regular and chaotic dynamics», Institute of computer researches, 2006, 384 p. (in Russian).
[10]. Morris D., Berge G. Direction of the magnetic field in the vicinity of the Sun. Astrophysical Journal, vol. 139, 1964, pp.1388-1392.
[11]. Manchester R.N. Pulsar rotation and dispersion measures and the Galactic magnetic field. Astrpophysical Journal, vol. 172, 1972, p.43-52.
[12]. Andreasyan R.R., Makarov A.N. Structure of the Galactic magnetic field. Astrophysics, vol. 30, 1989, pp.101-110.
[13]. Stepanov R., Arshakian T.G., Beck R., Frick P., Krause M. Magnetic field structures of galaxies derived from analysis of Faraday rotation measures, and perspectives for the SKA. Astronomy & Astrophysics, vol. 480, 2008, pp. 45-59.
[14]. Frick P., Sokoloff D., Stepanov R., Beck R. Faraday rotation measure synthesis for magnetic fields of galaxies. Monthly Notices of the Royal Astronomical Society, vol. 414, 2011, pp. 2540-2549.
[15]. Beck, R. Magnetic fields in the nearby spiral galaxy IC 342: A multi-frequency radio polarization study. Astronomy & Astrophysics, vol. 578, 2015, A93.
[16]. Molchanov S.A., Ruzmaikin A.A., Sokoloff D.D. Kinematic dynamo in random flow. Soviet Physics Uspekhi, vol. 28, no. 4, 1985, pp. 307-327.
[17]. Sokoloff D.D., Stepanov R.A., Frick P.G. Dynamo: from an astrophysical model to laboratory experiments. Physics-Uspekhi, vol. 57, 2014, pp. 292-311.
[18]. Buta R., Combes F. Galactic Rings. Fundamentals of Cosmic Physics, vol. 17, 1996, pp. 95-281.
[19]. Athanassoula E., Puerari I., Bosma A. Formation of rings in galactic discs by infalling small companions. Monthly Notices of the Royal Astronomical Society, vol. 286, 1997, pp. 284-302.
[20]. Ilyina M.A., Sil'chenko O.K., Afanasiev V.L. Nature of star-forming rings in S0 galaxies. Monthly Notices of the Royal Astronomical Society, vol. 439, 2014, pp. 334-341.
[21]. Moss D. On the generation of bisymmetric magnetic field structures in spiral galaxies by tidal interactions. Monthly Notices of the Royal Astronomical Society, vol. 275, 1995, pp.191-194.
[22]. Subramanian K., Mestel L. Galactic dynamos and density wave theory - II. an alternative treatment for strong non-axisymmetry. Monthly Notices of the Royal Astronomical Society, vol. 265, 1993, pp. 649-654.
[23]. Mikhailov E.A. Galactic dynamo with helicity fluxes. Astronomy Letters, vol. 39, 2013, pp.414-420.
[24]. Moss D., Mikhailov E., Silchenko O., Sokoloff D., Horellou C., Beck R. Magnetic fields in ring galaxies. Astronomy and Astrophysics, vol. 592, 2016, A44.
[25]. Deinzer W., Grosser H., Schmitt D. Torus dynamos for galaxies and accretion disks. I. The axisymmetric alpha omega-dynamo embedded into vacuum. Astronomy and Astrophysics, vol. 273, 1993, pp. 405-414.
[26]. Brooke J.M., Moss D. Nonlinear Dynamos in Torus Geometry - Transition to Chaos. Monthly Notices of the Royal Astronomical Society, vol. 266, 1994, pp. 733-739.
[27]. Mikhailov E.A. A dynamo in a torus as an explanation of magnetic fields in the outer rings of galaxies. Astronomy Reports, vol. 61, 2017, pp.7 39-746.
[28]. Mikhailov E.A. Torus Dynamo model for study of magnetic fields in the outer rings of galaxies. Astrophysics, vol. 61, 2018, pp. 147-159. y
[29]. Mikhailov E.A., Khokhryakova A.D. Torus dynamo in the outer rings of galaxies. Geophysical and Astrophysical Fluid Dynamics, vol. 113, 2019, pp.199-207.
[30]. Fischer P. An overlapping Schwarz method for spectral element solution of the incompressible Navier-Stokes equations. Journal of Computational Physics. vol. 133, issue 1, 1997, pp. 84-101.
[31]. Deville, M., Fischer, P., Mund, E. High-Order Methods for Incompressible Fluid Flow. Cambridge Monographs on Applied and Computational Mathematics. Cambridge University Press, Cambridge, 2002. 528 p.

Информация об авторах / Information about authors

Евгений Александрович МИХАЙЛОВ – кандидат физико-математических наук, ассистент физического факультета МГУ имени М.В.Ломоносова с 2016 г. Сфера научных интересов: теория динамо, магнитные поля галактик.

Evgeny Alexandrocih MIKHAILOV – Candidate of Physical and Mathematical Sciences, assistant of Faculty of Physics of Lomonosov Moscow State University. Research interests: dynamo theory, magnetic fields of galaxies.

Ильяс Наилевич СИБГАТУЛЛИН – кандидат физико-математических наук, старший научный сотрудник Институт океанологии им. П.П. Ширшова Российской академии наук, Москва. Сфера интересов: конвекция, стратифицированные среды, внутренние волны, инерционные волны, турбулентность, нелинейная динамика

Ilias Nailevich SIBGATULLIN – candidate of Physical and Mathematical Sciences, senior researcher at Shirshov Institute of Oceanology of Russian Academy of Sciences. Research interests: convection, internal waves, inertial waves, turbulence, nonlinear dynamics.

DOI: 10.15514/ISPRAS-2019-31(5)-15

CC BY

Исследование условий возникновения эолова микрорельефа

Е.А.Малиновская, ORCID: 0000-0003-0385-0396 <elen_am@inbox.ru>

Институт физики атмосферы им.А.М. Обухова РАН,
119017, Россия, Москва, Пыжевский пер., 3

Аннотация. Возможности численного моделирования движения гидродинамических потоков около объектов сложной формы позволяют рассматривать результаты вычислений, как один из способов понимания механизмов ветрового выноса песчаных частиц. Для исследования условий возникновения микрорельефа проводится ряд численных экспериментов с использованием открытого пакета Open OpenFOAM. Над неоднородностями поверхности, определяемыми особенностями взаимного расположения частиц, возникают области понижения давления, вблизи которых более вероятен ветровой вынос частиц. Причиной такого понижения давления являются различного рода разрежения структуры пространственного расположения элементов поверхности: изменение расстояния, изменение ориентации структуры в пространстве, изменение угла между плоскостями, содержащими частицы. За счет понижения давления происходит увеличение скорости воздушного потока у поверхности и возникновение микровихрей.

Ключевые слова: микрофизика ветрового выноса частиц; численное моделирование обтекания объектов сложной формы.

Для цитирования: Малиновская Е.А. Исследование условий возникновения эолова микрорельефа. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 225-236. DOI: 10.15514/ISPRAS–2019–31(6)–15

Благодарности: Автор выражает благодарность О.Г. Чхетиани за полезные консультации и обсуждения, М.В. Крапошину, С.В. Стрижаку за помощь в освоении принципов работы с OpenFoam. Исследование выполнено при частичной поддержке программы Президиума РАН №12 (проект КП19-270).

Study of the conditions for the occurrence of aeolian microrelief

E.A.Malinovskaya, ORCID: 0000-0003-0385-0396 <elen_am@inbox.ru>

A.M. Obukhov Institute of Atmospheric Physics RAS,
119017, Russia, Moscow, Pyzhevsky per., 3

Abstract. The possibilities of numerical modeling of the motion of hydrodynamic flows around objects of complex shape allow us to consider the results of calculations as one of the ways to understand the mechanisms of wind removal of sand particles. To study the conditions for the occurrence of microrelief, a number of numerical experiments are carried out using the open package OpenFOAM. Over inhomogeneities of the surface, determined by the features of the mutual arrangement of particles, there are areas of pressure reduction, near which the wind removal of particles is more likely. The reason for this decrease in pressure is a different kind of rarefaction of the spatial arrangement of surface elements: a change in distance, a change in the orientation of the structure in space, a change in the angle between planes containing particles. Due to the decrease in pressure, an increase in the air velocity at the surface and the occurrence of microvortices occur.

Keywords: microphysics of wind removal of particles, numerical simulation of flow around complex objects

For citation: Malinovskaya E.A. Study of the conditions for the occurrence of aeolian microrelief. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 225-236 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-15

Acknowledgements. The author is grateful to O.G. Chkhetiani for useful consultations and discussions, M.V. Kraposhin, S.V. Strizhak for help in learning the principles of working with OpenFoam. The study was partially supported by the program of the Presidium of the Russian Academy of Sciences No. 12 (project KP19-270).

1. Введение

Аридные территории являются источниками минерального аэрозоля, выносимого под воздействием ветра с подстилающей поверхности. Оценки интенсивности выноса возможны с использованием моделей переноса пыли [1]. Многие стороны механизма ветрового выноса аэрозоля, несмотря на уже довольно продолжительную историю этого вопроса, остаются недостаточно изученными и являются предметом интенсивных исследований [2-8]. Одной из важных характеристик процесса ветрового выноса пылевых и песчаных частиц является критическая или пороговая скорость ветра – та скорость воздушного потока над подстилающей поверхностью, при которой возможен отрыв частицы от поверхности [7, 8, 9, 10, 11, 12, 13]. Подвижность частиц на поверхности приводит к возмущениям воздушной среды [14, 15, 16]. Мгновенные скорости потока влияют на реализацию того или иного типа движения частицы: перекатывание, смещение, подъем [17]. Это приводит к изменению локальных характеристик поверхности и формированию на микромасштабах вторичных течений – типа струй и вихрей с вертикальной и горизонтальной осью [18]. При интенсификации турбулентности с увеличением скорости ветра и нагреве поверхности возникают вертикальные потоки и вихри, сообщающие больший импульс частицам, достаточный для их подъема на большую высоту. Наиболее активен вынос на тех участках, где происходит формирование барханов и движение песков [1]. Устойчивость частиц песка к воздействию воздушного потока определяется местом их отложения на эоловой структуре [19], что связано с особенностями обтекания и микрофизики их выноса с поверхности [17].

Песчаные частицы при различных расстояниях между ними частично или полностью погружены в ламинарный подслой [2], что влияет на шероховатость. Параметр шероховатости z_0 и коэффициент сопротивления C_f связаны между собой, так как z_0 зависит от величины динамической скорости u_* [5]**.** Для задачи обтекания кубиков, распределенных на поверхности, определено возрастание коэффициента сопротивления в зависимости от плотности покрытия, что объясняется наличием вторичных турбулентных потоков [20]. Изменение шероховатости в виде наличия частиц различного размера влияет на профили скорости ветра, потоки частиц, выносимые ветром [21]. В [22] на основе модели и спутниковых данных о рельефе нескольких эоловых структур для области наветренного склона увеличение динамической скорости при движении вверх от подножья до вершины составляет 0,015 м/с для эоловых структур высотой порядка 10 м, 0,02 м/с для 70 м, 0,17 м/с, если выше 80 м. Соответственно, параметр шероховатости и сопротивление возрастают при приближении к вершине. При этом на вершине эоловой структуры отмечается устойчивость к воздействию ветра [19]. За вершиной при изменении наклона плоскости поверхности ближе к подветренному склону расположена область струйного усиления ветра. Неоднородности на поверхности являются теми областями, в которых наблюдаются более активные изменения структуры самой поверхности при ветровом

выносе. Результаты предварительного моделирования при реализации различных условий необходимы для проведения натурных или лабораторных экспериментов.

2. Исходные данные численной модели обтекания поверхности

Область обтекающей среды представлена объемом параллелепипеда высотой 2000 мкм и длиной граней основания 5000 мкм. Высота выбрана таким образом, чтобы учесть характерную, известную из экспериментов, скорость воздушного потока над поверхностью [13]. В табл. 1 приведены оценки скорости воздушного потока на высоте ниже 1 мм [13]. Значение высоты 0,5 мм, считая профиль скорости линейным, определена скорость выше верхней кромки частиц в средней части параллелепипеда, 0,7 мм – это высота объема параллелепипеда, внутри которого будут имитироваться движения воздуха около частиц на поверхности.

Табл. 1. Оценка скорости воздушного потока около частиц
Table 1. Estimation of air velocity near particles

Высота над поверхностью, мм	Скорость воздушного потока, м/с		
	над склоном	за гребнем	на дне
Динамическая скорость, $u_*, м/с$	0,24	0,32	0,31
0,5	2,8	1,2	2,1
0,3	2,1	1,5	0,8

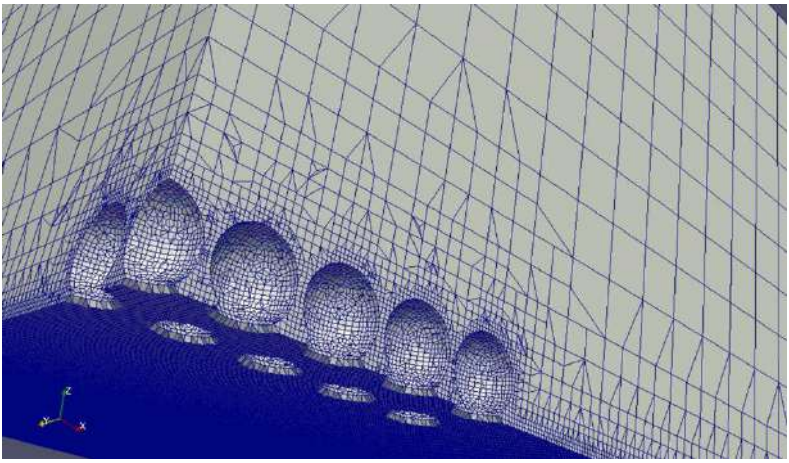

Рис. 1. Адаптивная разностная схема
Fig. 1. Adaptive difference scheme

Поперечные и вертикальные размеры расчетной области выбраны так, чтобы исключить эффекты влияния граничных условий. При этом следует вспомнить, что средний размер песчаной частицы 200 мкм. Таким образом, если в длину расположить 10 частиц такого размера в центральной области, до левой или правой границ области останется более 1000 мкм. Так как воздушный поток у поверхности движется в целом горизонтально, область разбита на два слоя по 1000 мкм. В верхнем слое задается поток с характерным для этой высоты значением скорости [13]. В нижнем слое на левой и правой стенках свободный поток с нулевым градиентом, как и на верхней стенке. На нижней стенке и для всех элементов поверхности, представляющих собой модели частиц, использованы условия прилипания.

В связи с тем, что поверхность имеет сложную форму, используется утилита snappyHexMesh, доступная в открытом пакете OpenFOAM [23], для послойного представления сетки с адаптацией под форму поверхности с минимальным размером шага сетки 5 мкм (рис. 1). Элементы, моделирующие частицы, представляют собой поверхности, состоящие из сфер, присоединенные к цилиндрам с целью минимизации проблем с расхождением ошибок в геометрии и расчетной сетки в областях контактов. Частицы объединены в блоки по 9 элементов (рис.2a), из которых строится поверхность с различной ориентацией в пространстве (рис. 2b, 2c). Проводится исследование влияния неоднородностей структуры поверхности (наличие неодинаковых расстояний между частицами) при различном взаимном расположении и наличии изменений в расстояниях между ними (рис. 2d, 2e) на характер обтекания и изменение подъемной силы. Расстояние между поверхностями взаимодействующих частиц в слое на одном уровне $D: D \approx (0.07 - 0.11)d$: ($d$ – средний размер частиц песка) определено с использованием расчетного оценочного значения порозности, близкого к наблюдаемому в природе [23].

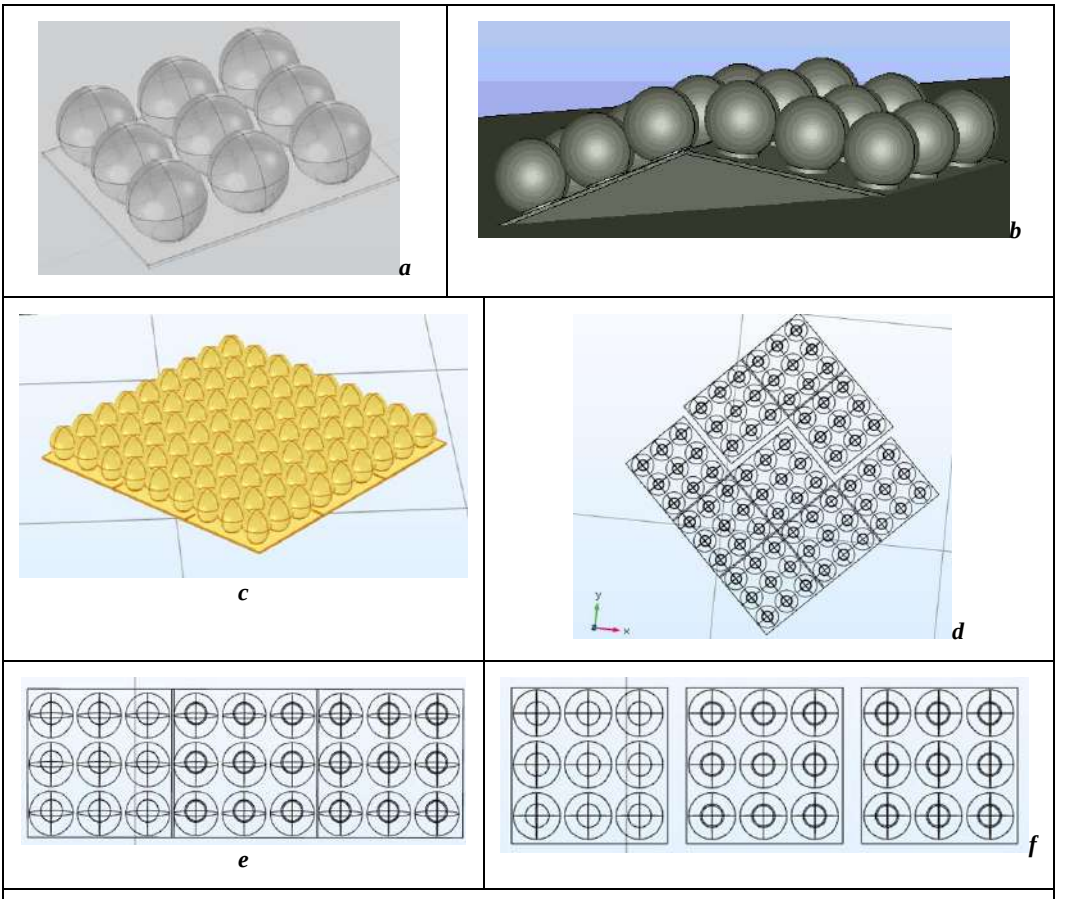

Рис. 2. 3D-моделирование геометрии обтекаемой поверхности (a – блок из 9 шарообразных частиц, b – три блока частиц, повернутых относительно друг друга, c – 9 блоков частиц, развернутых под углом 45 градусов к направлению движения воздуха, d – 9 блоков частиц под углом в 45 градусов с каналом внутри)

Fig. 2. 3D modeling of the geometry of the streamlined surface (a – a block of 9 spherical particles, b – three blocks of particles rotated relative to each other, c – 9 blocks of particles turned at an angle of 45 degrees to the direction of air movement, d – 9 blocks of particles under angle of 45 degrees with a channel inside)

Первоначальный захват сопровождается периодами покоя при скоростях воздушного потока, близких к пороговому значению [15], и активизируется при усилении турбулентного режима потока. При этом исходно ламинарный подслой становится турбулентным за счет движущихся по поверхности частиц [15]. В связи с этим в настоящем исследовании моделируется возможность ветрового выноса без участия частиц, поднятых в воздух, и учета их подвижности. Для анализа влияния особенностей структуры поверхности на характеристики ветрового выноса взята турбулентная $k-\varepsilon$ модель обтекания частиц. В качестве исходных параметров используются: кинетическая энергия турбулентного перемешивания $k=\frac{3}{2}\left(u_{ref}T_i\right)^2$, скорость потока u_{ref}, интенсивность турбулентности T_i, $\varepsilon=0.09^{3/4}\frac{k^{3/2}}{l}$, $l=0.07L$. Для указанных условий принималось, что размер обтекаемых частиц определяет параметр L=200 мкм, $u_{ref}=0.8$ м/с, так как предполагается линейный профиль скорости ветра в нижней области, а скорость в верхней области задается равной 2,8 м/с. Предложенная постановка задачи используется для исследования влияния особенностей структуры поверхности на характер обтекания и усиление турбулентного режима у поверхности на уровне микропроцессов.

2. Выбор решателей и устойчивость решений

Численное моделирование процесса обтекания частиц, лежащих на поверхности, реализовано средствами открытого пакета OpenFOAM [24]. Для турбулентного несжимаемого потока в качестве вариантов решателя выбран SimpleFOAM. В SimpleFOAM используется полуявный метод для уравнений, связанных с давлением, и заменяет PisoFOAM в качестве нестационарного несжимаемого решателя потока RANS Flow. С учетом рекомендаций в [25], было принято решение использовать решатели SimpleFOAM в связке с PimpleFOAM. SimpleFOAM применяется для того, чтобы подобрать параметры, при которых получается устойчивое решение. PimpleFOAM показывает хорошую устойчивость, однако требует много времени на одно решение, и по этому направлению еще ведутся расчеты. Поэтому далее будут обсуждаться результаты, полученные с использованием решателя SimpleFOAM.

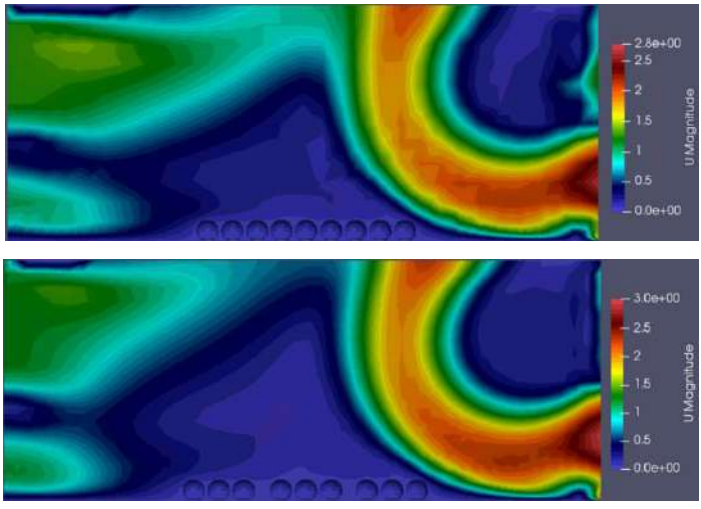

Рис. 3. Скорости обтекаемого потока вблизи поверхности для двух реализаций: без расстояний между блоками частиц (верхний рисунок) и с расстоянием 100 мкм (нижний рисунок)
Fig. 3. Flow velocities near the surface for two realizations: without distances between blocks of particles (upper figure) and with a distance of 100 μm (lower figure)

3. Влияние изменений однородности структуры поверхности на режим обтекания

Изменение расстояния между блоками частиц на 50, 100 и 150 мкм при прямой ориентации частиц по направлению к потоку не меняет структуры потока в целом. Около частиц происходит замедление потока, за областью их расположения – ускорение (рис.3). Однородности, реализованные увеличением расстояния между блоками, ускоряют поток, а также способствуют его закручиванию (рис.4), появляются области понижения давления над поверхностью (рис.5). В вблизи области неоднородности давление уменьшается в среднем на 1 $\frac{м^2}{с^2}$ на 500 мкм, над структурной неоднородностью поверхности возникает область пониженного давления, отмечается закручивание потока (рис.4).

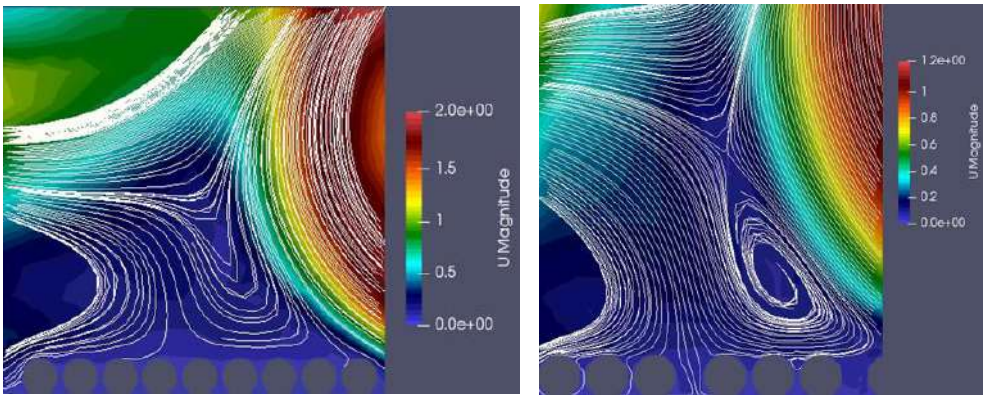

Рис. 4. Линии тока вблизи поверхности для двух реализаций: без расстояний между блоками частиц (верхний рисунок) и с расстоянием 100 мкм (нижний рисунок)
Fig. 4. Current lines near the surface for two realizations: without distances between blocks of particles (upper figure) and with a distance of 100 μm (lower figure)

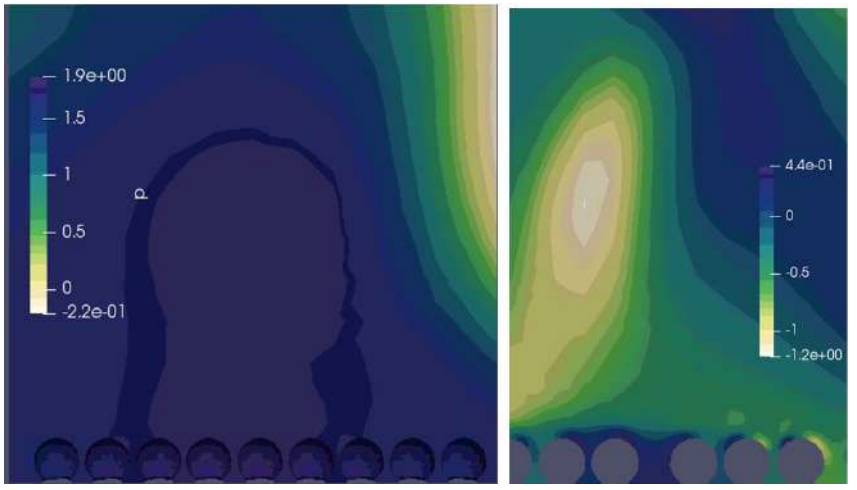

Рис. 5. Увеличение разности давления при обтекании частиц для двух реализаций: без расстояний между блоками частиц (рисунок слева) и с расстоянием 100 мкм (рисунок справа)
Fig. 5. The increase in the pressure difference in the flow around particles for two realizations: without distances between the blocks of particles (figure on the left) and with a distance of 100 μm (figure on the right)

Изменение ориентации структуры связанных частиц при повороте блока вокруг вертикальной оси (рис.2в) можно определить, как изменение расстояний между их поверхностями вдоль оси по направлению движения воздушного потока. Расстояние между частицами влияет величину разности давлений под частицей и над ней [17]. В данном случае расстояние увеличивается в 1,4 раза. Если для малых расстояний при прямой ориентации блока частицы придавливаются к поверхности (рис.5), то при развороте в 45 градусов отмечаем области с выталкивающей силой, за счет разности давления над частицей и под ней возможен ее отрыв от поверхности (рис. 6).

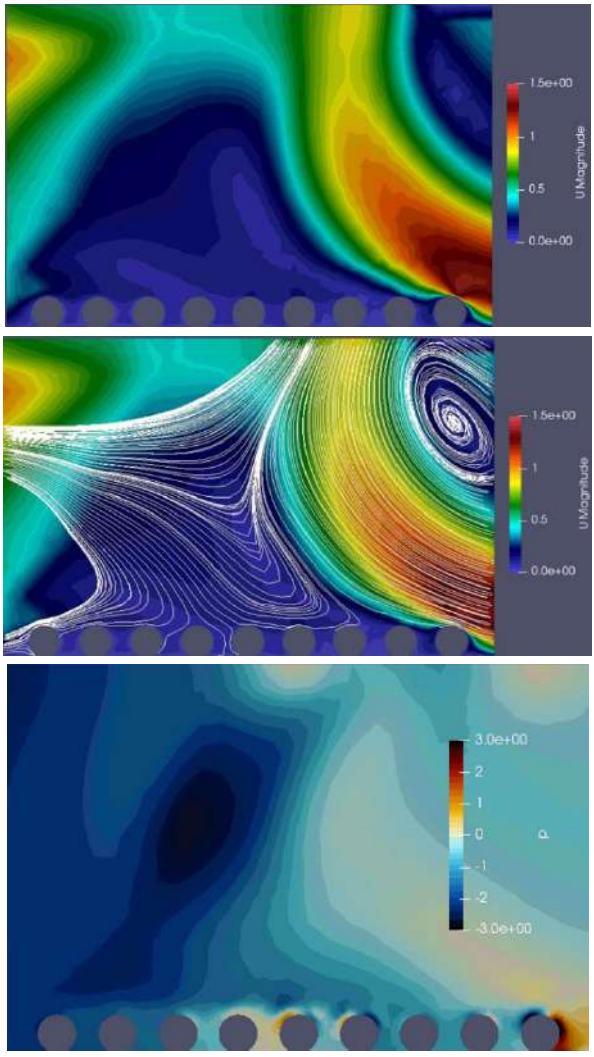

Рис. 6. Увеличение разности давления при обтекании частиц с различными типами неоднородных элементов структуры
Fig. 6. The increase in pressure difference when flowing around particles with various types of heterogeneous structural elements

При изменении расстояний между отдельными частицами наблюдается похожий эффект для центральных частиц (рис. 5). В этом также возникает область пониженного давления над поверхностью, отмечается закручивание потока.

При обтекании частиц, собранных на двух блоках, плоскости которых наклонены друг к другу под углом (рис. 7), наибольшее значение давления отмечаем на вершине структуры, что приводит к возникновению горизонтальной разности давлений, разбрасывающей частиц в разные стороны, что характерно для однородной структуры. Над подветренной частью такой структуры появляется микровихрь.

Следует отметить, что наиболее интересными оказываются области между центральными частицами однородной структуры для блоков, развернутых под углом 45 градусов, и в областях неоднородности. В связи с этим далее будет приведен краткий анализ параметров, вычисляемых для этих случаев.

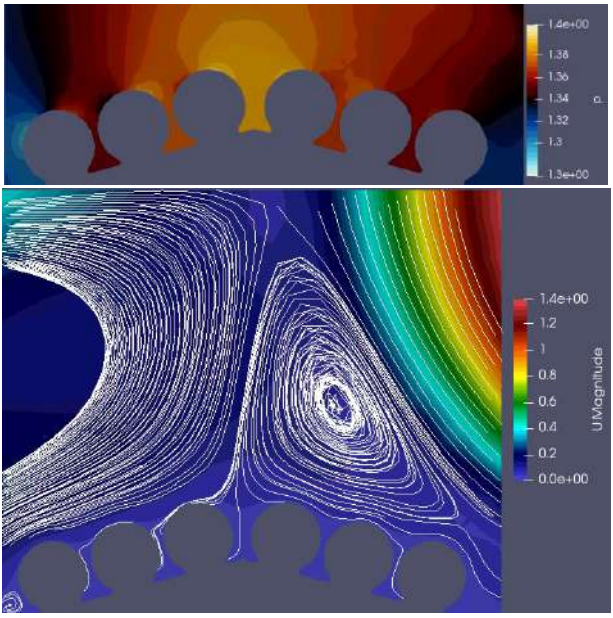

Рис 7. Изменение давления и скорости потока при обтекании частиц при взаимодействии двух наклоненных поверхностей, содержащих частицы

Fig. 7. Change in pressure and flow rate during particle flow when there is the interaction of two inclined surfaces containing particles

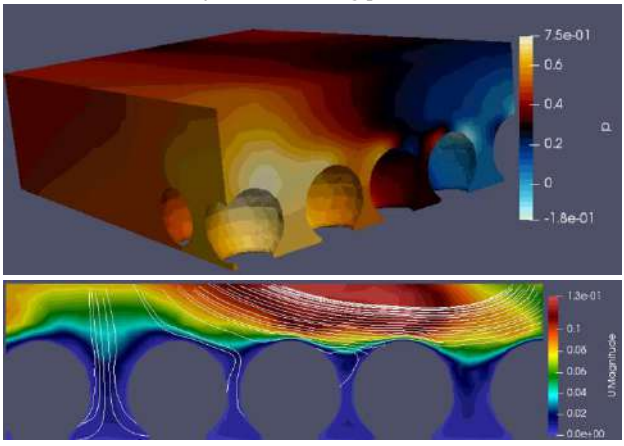

Рис. 8. Увеличение разности давления и линии тока около области неоднородности при обтекании частиц с различными типами неоднородных элементов структуры

Fig. 8. The increase in the difference in pressure and streamline near the region of inhomogeneity when flowing around particles with various types of inhomogeneous structural elements

4. Условия возникновения эолового микрорельефа при постоянном ветровом воздействии

На обтекание поверхности, состоящей из частиц, существенным образом влияют на величину выталкивающей из слоя силы неоднородности поверхности, возникающие за счет изменения расстояния между отдельными частицами, равномерности или неравномерности их распределения на поверхности, а также неровностей за счет изменений углов наклона плоскостей по отношению к направлению движения воздушного потока.

Для определения влияния неоднородностей рассматривался сценарий изменения расстояний между двумя блоками частиц. В частности, для случая, когда расстояние равно 100 мкм (рис.8) отмечаем изменение разности давлений в зависимости от положения частицы относительно области неоднородности. Для удобства пронумеруем области между частицами от 1 до 4 слева направо. Отмечаем, что в первой области скорость воздушного потока больше, за счет чего увеличивается в последующих двух областях давление.

Табл. 2. Изменение давления в областях между частицами
Table 2. The change in pressure in the areas between the particles

№ области	$P_{z=20\ мкм}$	$P_{z=100\ мкм}$	$P_{z=220\ мкм}$	Разность давлений
1	0,66	0,67	0,7	-0,03
2	0,55	0,54	0,5	0,05
3	0,31	0,31	0,255	0,05
4	0,58	0,76	0,96	-0,38

Значения давлений и скоростей приведены в табл. 2 и 3 соответственно. Для всей протяженности структуры, состоящей из 9 последовательных частиц вдоль потока, разность давления над частицей и под ней максимальна и положительна, то есть возможен отрыв частицы, для двух частиц, расположенных между областями разрежения структуры. Поэтому ветровой вынос начинается именно в таких областях, что в результате определяет квазипериодичность структуры поверхности.

Табл. 3. Изменение скорости в областях между частицами
Tab. 3. The change in velocity in the areas between the particles

№ области	$u_{z=20\ мкм}$	$u_{z=100\ мкм}$	$u_{z=220\ мкм}$
1	0,002	0,01	0,04
2	0,0001	0,0001	0,069
3	0,0014	0,0016	0,1
4	0,003	0,01	0,09

При минимальных расстояниях между поверхностями частиц в слое невозможен ветровой вынос. С увеличением расстояний в областях неоднородности или при изменении ориентации блоков по отношению к направлению движения ветра возможен ветровой вынос отдельных частиц. Для роста эоловой структуры необходимо, чтобы выносилось с поверхности определенная доля частиц [22], оставшиеся перекатывались, или оставались неподвижными. Сохранение возникшей структуры без раскатывающихся частиц от вершины определяет различие в структурах и наклонах наветренной и подветренной сторон.

5. Заключение

Изменение угла между плоскостями поверхностей блоков приводит к появлению неоднородностей в первоначальной структуре. В таких областях отмечается повышение скорости обтекающего потока от 0,05 м/с при отрыве от поверхности на стыке двух блоков и снижение давления. Увеличение при этом разностей давления дает рост начальной скорости выноса частицы из слоя. Разность значений давления для точек вокруг частицы, определяющая выталкивающую силу, увеличивается также и при движении по направлению потока над поверхностями частиц. Относительное увеличение составляет около 10% от величины разности. Неоднородности, возникшие за счет увеличения расстояния между поверхностями отдельных частиц или при изменении угла наклона плоскости, влияют на локальное изменение разности давления на 10-30%.

Все эти факторы указывают на то, что песчаная поверхность под постоянным воздействием ветра представляет собой чередование квазирегулярных областей, в которых профицит разности давления снизу и сверху частицы преобладает над силой тяжести. Как следствие, появляются области с различной вероятностью ветрового выноса, за счет чего, в частности, происходит возникновение эоловой ряби.

Список литературы / References

[1]. Закарин Э.А. и др. ГИС-технология мониторинга и моделирования пыльных бурь. Гидрометеорология и экология, №. 3, 2010 г., стр. 8-20 / Zakarin E.A. et al. GIS technology for monitoring and modeling dust storms. Hydrometeorology and ecology, no. 3, 2010, pp. 8-20 (in Russian).

[2]. Greeley R., Iversen D.J. Wind as geological process of Earth, Mars and Titan. New York, Cambridge University press, 1985. 333 p.

[3]. Ivanov V.K., Matveev A.Ya., Tsymbal V.N., Yatsevich S.Ye. Radar investigations of the aeolian sand and dust transporting manifestations in desert areas. Telecommunications and Radio Engineering, vol. 74, no. 14, 2015, pp. 1269-1283.

[4]. Почвозащитное земледелие. Под общ. ред. А.И. Бараева. М., Колос, 1975, 304 стр. / Conservation Agriculture. Under general editorship of A.I. Barayev. M., Kolos, 1975, 304 pp. (in Russian).

[5]. Chamberlain A.C. Roughness length of sea, sand, and snow. Boundary-Layer Meteorology, vol. 25, № 4, 1983, pp. 405-409.

[6]. Горчаков Г.И., Карпов А.В. и др. Экспериментальное и теоретическое исследование траекторий сальтирующих песчинок на опустыненных территориях. Оптика атмосферы и океана, том 25, № 6, 2012 г., стр. 501-506 / Gorchakov G.I., Karpov A.V. et al. Experimental and theoretical study of trajectories of salt sand grains in desert territories. Optics of the Atmosphere and the Ocean, vol. 25, № 6, 2012, pp. 501-506 (in Russian).

[7]. Гендугов В.М., Глазунов Г.П. Ветровая эрозия почвы и запыление воздуха. М., Физматлит, 2007 г., 238 стр. / Gendugov V.M., Glazunov G.P. Wind erosion of the soil and dusting of the air. M., Fizmatlit, 2007, 238 p. (in Russian).

[8]. Семенов О.Е. Экспериментальные исследования кинематики и динамики пыльных бурь и поземков. Труды Казахского научно-исследовательского гидрометеорологического института, вып. 49, 1972 г., стр. 2-31 / Semenov O.E. Experimental studies of the kinematics and dynamics of dust storms and windfalls. Proceedings of the Kazakh Research Hydrometeorological Institute, issue 49, 1972, pp. 2-31 (in Russian).

[9]. Bagnold R.A. The physics of blown sand and desert dunes. London, Chapman & Hall, 1973, 265 p.

[10]. Бютнер Э.К. Динамика приповерхностного слоя воздуха. Л., Гидрометиздат, 1978 г., 156 стр. / Butner E.K. The dynamics of the surface air layer. L., Gidrometizdat, 1978, 156 p. (in Russian).

[11]. Hau Lu. An integrated wind erosion modeling system with emphasis on dust emission and transport PhD Thesis, School of Mathematics, University of New South Wales, Sydney, Australia, 1999. 185 p.

[12]. Shao Y. Physics and modeling of wind erosion. Springer, 2008, 452 p.

[13]. Семенов О.Е. Введение в экспериментальную метеорологию и климатологию песчаных бурь. Алматы, Казахский научно-исследовательский институт экологии и климата, 2011 г., 580 стр. /

Semenov O.E. Introduction to experimental meteorology and climatology of sandstorms. Almaty, Kazakh Research Institute of Ecology and Climate, 2011, 580 p. (in Russian).
[14]. Lämmel M., Rings D., Kroy K. A two-species continuum model for aeolian sand transport. New Journal of Physics, vol. 14, №. 9, 2012, 24 p.
[15]. Pähtz T. et al. The critical role of the boundary layer thickness for the initiation of aeolian sediment transport. Geosciences, vol. 8, №. 9, 2018, 14 p.
[16]. Williams J. J., Butterfield G. R., Clark D. G. Aerodynamic entrainment threshold: effects of boundary layer flow conditions. Sedimentology, vol. 41, №. 2, 1994, pp. 309-328.
[17]. Malinovskaya E., Gorchakov G., Chkhetiani O., Karpov A. About the quasi-periodical variations of particles saltation. Geophysical Research Abstracts, vol. 21, EGU2019-3693-1, 2019.
[18]. Chou Y. J., Fringer O. B. A model for the simulation of coupled flow bed form evolution in turbulent flows. Journal of geophysical research, vol. 115, 2010, 20 p.
[19]. Малиновская Е.А. Трансформация эоловых форм рельефа при ветровом воздействии. Известия РАН. Физика атмосферы и океана, том 55, №. 1, 2019 г., стр. 54-64 / Malinovskaya E.A. Transformation of aeolian relief forms under wind influence. Izvestiya, Atmospheric and Oceanic Physics, vol. 55, issue 1, 2019, pp. 50–58.
[20]. Yang X.I.A. et al. Drag forces on sparsely packed cube arrays. Journal of Fluid Mechanics, vol. 880, 2019, pp. 992-1019.
[21]. Dupont S., Bergametti G., Simoëns S. Modeling aeolian erosion in presence of vegetation. Journal of Geophysical Research: Earth Surface, vol. 119, №. 2, 2014, pp. 168-187.
[22]. Малиновская Е. А. Модель формирования ветрового эолового склона. Известия РАН. Физика атмосферы и океана, том 55, №2, 2019 г., стр. 86-95 / Malinovskaya E.A. Windward Aeolian Slope Formation Model. Izvestiya, Atmospheric and Oceanic Physics, vol. 55, issue 2, 2019, pp. 218–228.
[23].] Кормилицына О. В., Бондаренко В. В., Палий И. М. Оценка свойств гранулометрических элементов как основа для создания почвенно-грунтовых смесей заданного качества. Лесной вестник, №7, 2007, стр. 84-89 / Kormilitsyna O.V., Bondarenko V.V., Paliy I.M. The estimation of properties of texture particles as a basis for creation of soil substrates of the required quality. Forestry Bulletin, №7, 2007, pp. 84-89 (in Russian).
[24]. Moukalled F. et al. The finite volume method in computational fluid dynamics //An advanced introduction with OpenFoam and Matlab. Springer, 2016, 135 p.
[25]. The PIMPLE algorithm in OpenFOAM. Available at: https://openfoamwiki.net/index.php/OpenFOAM_guide/The_PIMPLE_algorithm_in_OpenFOAM.

Информация об авторах / Information about authors

Елена Александровна МАЛИНОВСКАЯ – кандидат физико-математических наук, научный сотрудник лаборатории геофизической гидродинамики Института физики атмосферы им. А.М. Обухова РАН с 2018 г. Сфера научных интересов: исследование динамических и обменных процессов на границе раздела атмосфера – подстилающая поверхность, математическое и численное моделирование ветрового выноса минеральных аэрозолей.

Elena Aleksandrovna MALINOVSKAYA – Ph.D. in Physics and Mathematics, researcher at the Laboratory of Geophysical Hydrodynamics at the A.M. Obukhov Institute of Atmospheric Physics, Russian Academy of Sciences since 2018. Research interests: study of dynamic and metabolic processes at the interface between the atmosphere and the underlying surface, mathematical and numerical modeling of the wind removal of mineral aerosols.

DOI: 10.15514/ISPRAS-2019-31(6)-16

CC BY

Применение сеточно-характеристического метода для решения задач распространения динамических волновых возмущений на высокопроизводительных вычислительных системах

Н.И. Хохлов, ORCID: 0000-0002-2460-0137 <khokhlov.ni@mipt.ru>
И.Б. Петров, ORCID: 0000-0003-3978-9072 <petrov@mipt.ru>

Московский физико-технический институт
141701, Московская область, г. Долгопрудный, Институтский переулок, д.9

Аннотация. В работе рассматривается применение различных современных технологий высокопроизводительных вычислений для ускорения численного решения задач распространения динамических волновых возмущений с использованием сеточно-характеристического метода. Рассматриваются технологии как для центральных процессоров (CPU), так и для графических процессоров (GPU). Приведены сравнительные результаты применения технологий MPI, OpenMP, CUDA. В качестве примеров работы разработанного программного комплекса приводится ряд примеров расчета задач сейсмики и геофизики. Отдельно рассмотрен вопрос распараллеливания задач с наличием контактов многих сеток и топографии дневной поверхности, используя криволинейные сетки.

Ключевые слова: сеточно-характеристический метод; сейсмика; геофизика; MPI; OpenMP; CUDA

Для цитирования: Хохлов Н.И., Петров. И.Б. Применение сеточно-характеристического метода для решения задач распространения динамических волновых возмущений на высокопроизводительных вычислительных системах. Труды ИСП РАН, том 31, вып. 6, 2019 г., стр. 237-252. DOI: 10.15514/ISPRAS–2019–31(6)–16

Благодарности: Работа выполнена при финансовой поддержке РФФИ в рамках научного проекта № 18-07-00914.

Application of the grid-characteristic method for solving the problems of the propagation of dynamic wave disturbances in high-performance computing systems

N.I. Khokhlov, ORCID: 0000-0002-2460-0137 <khokhlov.ni@mipt.ru>
I.B. Petrov , ORCID: 0000-0003-3978-9072 <petrov@mipt.ru>

Moscow Institute of Physics and Technology,
9 Institutskiy per., Dolgoprudny, Moscow Region, 141701, Russian Federation

Abstract. The paper considers the application of various modern technologies of high-performance computing to accelerate the numerical solution of the problems of propagation of dynamic wave disturbances using the grid-characteristic method. Technologies are considered both for central processing units (CPUs) and for graphic processors (GPUs). Comparative results of applying MPI,

OpenMP, CUDA technologies are presented. As examples of the work of the developed software package, a number of examples of calculating the problems of seismic and geophysics are given. Separately, the issue of parallelizing problems with the presence of contacts of many grids and the topography of the day surface using curvilinear grids is considered.

Keywords: grid-characteristic method; seismic; geophysics; MPI; OpenMP; CUDA

For citation: Khokhlov N.I., Petrov I.B. Application of the grid-characteristic method for solving the problems of the propagation of dynamic wave disturbances in high-performance computing systems. Trudy ISP RAN/Proc. ISP RAS, vol. 31, issue 6, 2019. pp. 237-252 (in Russian). DOI: 10.15514/ISPRAS-2019-31(6)-16

Acknowledgments: This work was supported by the Russian Foundation for Basic Research as part of research project No. 18-07-00914.

1. Введение

Численное моделирование распространения динамических волновых возмущений в твердых телах используется при решении широкого круга задач. К таким задачам, в том числе, относятся задачи сейсморазведки и вычислительной геофизики. Роль численного моделирования в каждой из данных областей очень важно. Поиск полезных ископаемых, таких как углеводороды, является дорогостоящей задачей, для решения которой применяются различные методы петрофизики, сейсмики и геологии. В результате применения этих методов строится приблизительная модель геологической среды. Далее эта модель используется для проверки различных гипотез о положении в пространстве геологических пластов и распределении полезных ископаемых. Один из методов такого анализа – точное решение прямой задачи распространения волн упругости с использованием численного моделирования. Кроме того, решение прямой задачи является составной частью методов инверсии и миграции, цель которых заключается в определении параметров среды и положений в пространстве разделов между средами, что в конечном итоге позволяет обнаружить полезные ископаемые в толще земли. Численное моделирования распространения сейсмических волн представляет существенную часть работ при проведении геологоразведки в нефтяной отрасли. Математическое моделирование проводится в различных геологических средах, в том числе, в слоистых средах и в средах с наличием различных неоднородностей (например, трещин или каверн). Задачи такого рода представляются очень ресурсоемкими с точки зрения вычислительных ресурсов.

Характерные размеры расчетных областей в таких задачах по каждому измерению около 1000-10000 узлов, а разумные времена расчета – около суток. Производительность современных настольных компьютеров позволяет решать прямую задачу явными численными методами в двумерном случае за разумное время. А производительности кластеров уже достаточно для решения трехмерных задач. Повышенный интерес промышленности к решению прямой задачи объясняется ростом производительности современных центральных процессоров, поскольку это позволило быстро получать результаты моделирования для крупных геологических моделей. Повышению интереса также способствует интенсивное развитие технологий параллельного программирования прикладных задач общего назначения на графических процессорах. Как правило, явные численные методы могут быть переписаны для эффективного исполнения на GPU, что в зависимости от типа задачи и численного метода может повысить производительность на несколько порядков по сравнению с реализацией для центрального процессора. Таким образом, возросший интерес к проведению численного моделирования в индустрии объясняет актуальность данной работы. В работе рассматривается сеточно-характеристический метод решения уравнения распространения волн в упругой среде. Данный метод обладает рядом особенностей, из-за которых его применение в

определенных постановках более целесообразно, чем использование классических методов наподобие конечно-разностного. В работе [1] было произведено сравнение разрывного метода Галеркина с сеточно-характеристическим методом на неструктурированной сетке и на регулярной сетке [9, 24]. Авторами была продемонстрирована более высокая скорость расчета с использованием СХ метода, а точность расчета показала применимость метода к практическим задачам.

В данной работе рассматривается программный комплекс, созданный для моделирования задач распространения динамических волновых возмущений в твердых телах. Программный комплекс использует двумерные и трехмерные структурные блочные сетки с наличием неоднородностей. Для численного интегрирования используются сеточно-характеристические [1,2] и конечно-объемные [3] методы 2-4 порядка точности. Алгоритм распараллелен используя различные технологии написания параллельных приложений. В настоящее время достигнута эффективность распараллеливания до 70% используя технологию MPI при масштабировании до 16 тысяч вычислительных ядер. В системах с общей памятью алгоритм распараллелен используя технологию OpenMP. Кроме того, код распараллелен с использованием технологии CUDA, что дает ускорение до 50 раз по сравнению с одним ядром CPU. Программа может использовать несколько карточек в рамках одного хоста. В данной работе рассмотрены результаты одного и того же алгоритма используя различные технологии. Приведены тесты распараллеливания до 16 тысяч ядер CPU и 8 устройств CUDA.

2. Математическая модель

В работе рассматривается модель сплошной среды, использующая определяющие соотношения линейной теории упругости. Здесь и далее будем использовать верхний индекс для обозначения номера компоненты вектора (или тензора). Рассмотрим физическую модель среды. В каждой точке пространства $x = (x^1, x^2, x^3)$ задан вектор смещения $u(x,t) = (u^1, u^2, u^3)$ частиц среды от начального положения в результате упругого взаимодействия. Тензор деформации $\varepsilon(x,t)$ вычисляется на основе вектора смещений следующим образом:

$$\varepsilon^{ij} = \frac{1}{2}\left(u^i_j + u^j_i\right), \quad i,j \in \{1,2,3\}. \#(1)$$

Нижний индекс обозначает дифференцирование. Если там указано число, то имеется в виду дифференцирование по пространственной координате с данным номером. Например, дифференцирование некоторой переменной $a(x,t)$ будет выглядеть так:

$$a_t = \frac{\partial a}{\partial t}, \quad a_2 = \frac{\partial a}{\partial x_2}.$$

Для каждой точки среды можно записать второй закон Ньютона:

$$\rho u^i_{tt} - \sum_{j=1}^{3} \sigma^{ij}_j - f^i = 0; \;\; i \in \{1,2,3\}. \#(2)$$

Здесь $\rho(x)$ – плотность в рассматриваемой точке пространства, а $f(x,t) = (f^1, f^2, f^3)$ – плотность силы, направленной из рассматриваемой точки пространства x, а σ^{ij} – тензор упругого напряжения.

Помимо уравнения движения приведенного выше, понадобится еще закон сохранения, который задаст взаимосвязь между тензором деформации и тензором напряжения. Обобщенный закон Гука как раз и устанавливает такую связь.

$$\sigma^{ij} = \sum_{k=1}^{3} \sum_{l=1}^{3} c^{ijkl} \varepsilon^{kl}, \quad i,j \in \{1,2,3\}. \#(3)$$

Остается задать значения c_{ijkl}. Дальнейшие выкладки будут рассматриваться для модели идеальной линейно-упругой среды без поглощения волн. В программном комплексе также реализованы и другие модели, например: модель вязкоупругой среды, учитывающая затухание волн и модель упругопластичной среды, учитывающая гистерезистные свойства материала.

В случае изотропной линейно-упругой среды, равенство (3) значительно упрощается. В силу симметрии тензора напряжений, а также в силу изотропности среды, количество независимых переменных в c_{ijkl} снижается с 91 до 2. В качестве таких неизвестных удобно принять так называемые параметры Ламе λ и μ. Данные параметры выражаются через скорость продольных P-волн $c_p = \sqrt{(\lambda + 2\mu)/\rho}$ и скорость поперечных S-волн $c_s = \sqrt{\mu/\rho}$. Тогда закон сохранения примет следующий вид:

$$\sigma^{ij} = \lambda \sum_{k=1}^{3} \varepsilon^{kk} \delta^{ij} + 2\mu\varepsilon^{ij}, \quad i,j \in \{1,2,3\}. \#(4)$$

где δ^{ij} - символ Кронекера. С учетом равенства 1, закон сохранения можно записать как связь между тензором напряжений и вектором смещений:

$$\sigma^{ij} = \lambda \sum_{k=1}^{3} u_k^k \delta^{ij} + \mu\left(u_j^i + u_i^j\right), \quad i,j \in \{1,2,3\}. \#(5)$$

Таким образом, мы получили систему уравнений (2), (5), которая описывает распространение волн в линейно-упругих средах.

Для того, чтобы перейти к двумерному случаю, положим производные компонент вектора смещений и тензора напряжений по оси x_3 равными 0. Это делается из соображения, что изменения параметров вдоль одной из осей трехмерного пространства в двумерном случае нет. Уравнения (2), (5) распадутся на две независимых системы уравнений. Первая описывает распространение упругих волн в двумерном случае и представляет для нас интерес:

$$\rho u_{tt}^i - \sum_{j=1}^{2} \sigma_j^{ij} - f^i = 0, \quad i \in \{1,2\}. \#(6)$$

$$\sigma^{ij} = \lambda \sum_{k=1}^{2} u_k^k \delta^{ij} + \mu\left(u_j^i + u_i^j\right), \quad i,j \in \{1,2\}. \#(7)$$

В двумерном случае вид системы уравнений остался тем же, как и в трехмерном случае, но количество уравнений уменьшилось с 9 до 5.

3. Сеточно-характеристический метод

Для того, чтобы продемонстрировать применение численного метода, надо записать систему (6), (7) в матричной форме [9]. Для ее записи будем использовать вектор скорости $v = u_t$ вместо вектора смещения u.

$$\begin{bmatrix} v_t^1 \\ v_t^2 \\ \sigma_t^{11} \\ \sigma_t^{22} \\ \sigma_t^{12} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \frac{1}{\rho} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{\rho} \\ \lambda+2\mu & 0 & 0 & 0 & 0 \\ \lambda & 0 & 0 & 0 & 0 \\ 0 & \mu & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_1^1 \\ v_1^2 \\ \sigma_1^{11} \\ \sigma_1^{22} \\ \sigma_1^{12} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{1}{\rho} \\ 0 & 0 & 0 & \frac{1}{\rho} & 0 \\ 0 & \lambda & 0 & 0 & 0 \\ 0 & \lambda+2\mu & 0 & 0 & 0 \\ \mu & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_2^1 \\ v_2^2 \\ \sigma_2^{11} \\ \sigma_2^{22} \\ \sigma_2^{12} \end{bmatrix} + \begin{bmatrix} \frac{f^1}{\rho} \\ \frac{f^2}{\rho} \\ 0 \\ 0 \\ 0 \end{bmatrix}. \#(8)$$

Введем вектор $\varphi(x,t) = (v^1, v^2, \sigma^{11}, \sigma^{22}, \sigma^{12})$. Обозначим матрицы в уравнении (8) как A^1 и A^2. Вектор с функциями, определяющими источник упругих колебаний, обозначим как $s = (f^1/\rho, f^2/\rho, 0,0,0)$. Тогда уравнение (10) можно будет записать в более короткой форме:

$$\varphi_t = A^1\varphi_1 + A^2\varphi_2 + s. \#(9)$$

Далее, используя метод расщепления (или метод дробных шагов) по пространству, описанный например в [5], перейдем к решению одномерных уравнений переноса:

$$\varphi_t = A^1\varphi_1, \quad \varphi_t = A^2\varphi_2. \#(10)$$

В дальнейшем пересчет с одного временного слоя на следующий согласно формулам (10) будем называть соответственно шагом по X и шагом по Y. Обозначим их в виде операторов: $\varphi^{t+\Delta t} = \mathrm{S}_x(\Delta t, \varphi^t)$, $\varphi^{t+\Delta t} = \mathrm{S}_y(\Delta t, \varphi^t)$. Последовательность вычислений этих шагов накладывает ограничение на порядок схемы по пространству. В работе [6] описано так называемое расщепление Странга (Strang splitting), определяющее последовательность вычисления одномерных уравнений переноса, которое позволяет получить второй порядок точности при условии, что каждый шаг по отдельности имеет порядок не меньше второго. Расщепление Странга заключается в применении следующего порядка вычислений отдельных шагов для пересчета с момента времени t в $t + \Delta t$:

$$\varphi^{t+\frac{\Delta t}{4}} = \mathrm{S}_x\left(\frac{\Delta t}{4}, \varphi^t\right), \#(11)$$

$$\varphi^{t+\frac{3\Delta t}{4}} = \mathrm{S}_y\left(\frac{\Delta t}{2}, \varphi^{t+\frac{\Delta t}{4}}\right), \#(12)$$

$$\varphi^{t+\Delta t} = \mathrm{S}_x\left(\frac{\Delta t}{4}, \varphi^{t+\frac{3\Delta t}{4}}\right). \#(13)$$

Как показала практика, применение более простого способа расчета шагов по X и Y хоть и имеет первый порядок точности, но позволяет получить результат, близкий к расщеплению Странга:

$$\varphi^{t+\frac{\Delta t}{2}} = \mathrm{S}_x\left(\frac{\Delta t}{2}, \varphi^t\right), \#(14)$$

$$\varphi^{t+\Delta t} = \mathrm{S}_y\left(\frac{\Delta t}{2}, \varphi^{t+\frac{\Delta t}{2}}\right). \#(15)$$

Поскольку выбор способа расщепления не вносит существенных изменений в пригодность к повышению производительности и распараллеливанию метода, было принято решение в тестовой программной реализации воспользоваться вторым, более простым вариантом.

Уравнения (10) решаются при помощи приведения матриц к диагональному виду [4]: $A^1 = L^1\Lambda^1 R^1$ и $A^2 = L^2\Lambda^2 R^2$, где $L^1R^1 = E$, $L^2R^2 = E$, а матрицы Λ^1 и Λ^2 – диагональные. Такое приведение возможно, поскольку матрицы A^1 и A^2 имеют действительные собственные числа, которые и стоят на диагоналях матриц Λ^1, Λ^2:

$$\Lambda^1 = \Lambda^2 = \mathrm{diag}(c_p, -c_p, c_s, -c_s, 0). \#(16)$$

Поскольку матрицы рассматриваемой системы уравнений могут быть приведены к диагональному виду, система уравнений является гиперболической. Матрицы L^1 и L^2 содержат в столбцах собственные векторы A^1 и A^2 соответственно:

$$L^1 = \begin{bmatrix} c_p & -c_p & 0 & 0 & 0 \\ 0 & 0 & c_s & -c_s & 0 \\ \lambda+2\mu & \lambda+2\mu & 0 & 0 & 0 \\ \lambda & \lambda & 0 & 0 & 1 \\ 0 & 0 & \mu & \mu & 0 \end{bmatrix}, \quad L^2 = \begin{bmatrix} 0 & 0 & c_s & -c_s & 0 \\ c_p & -c_p & 0 & 0 & 0 \\ \lambda & \lambda & 0 & 0 & 1 \\ \lambda+2\mu & \lambda+2\mu & 0 & 0 & 0 \\ 0 & 0 & \mu & \mu & 0 \end{bmatrix}. \#(17)$$

Матрицы R^1 и R^2 получаются обращением матриц L^1 и L^2:

$$R^1 = \begin{bmatrix} +\frac{1}{2c_p} & 0 & \frac{1}{2(\lambda+2\mu)} & 0 & 0 \\ -\frac{1}{2c_p} & 0 & \frac{1}{2(\lambda+2\mu)} & 0 & 0 \\ 0 & +\frac{1}{2c_s} & 0 & 0 & \frac{1}{2\mu} \\ 0 & -\frac{1}{2c_s} & 0 & 0 & \frac{1}{2\mu} \\ 0 & 0 & -\frac{\lambda}{\lambda+2\mu} & 1 & 0 \end{bmatrix}, \quad R^2 = \begin{bmatrix} 0 & +\frac{1}{2c_p} & 0 & \frac{1}{2(\lambda+2\mu)} & 0 \\ 0 & -\frac{1}{2c_p} & 0 & \frac{1}{2(\lambda+2\mu)} & 0 \\ +\frac{1}{2c_s} & 0 & 0 & 0 & \frac{1}{2\mu} \\ -\frac{1}{2c_s} & 0 & 0 & 0 & \frac{1}{2\mu} \\ 0 & 0 & 1 & -\frac{\lambda}{\lambda+2\mu} & 0 \end{bmatrix}. \#(18)$$

Умножим слева уравнения (10) на R^1 и R^2 соответственно. Тогда получим:

$$R^1\varphi_t = R^1(L^1\Lambda^1R^1)\varphi_1, \quad R^2\varphi_t = R^2(L^2\Lambda^2R^2)\varphi_2. \#(19)$$

Перейдем от исходных неизвестных φ к новым: $\omega^1 = R^1\varphi$ и $\omega^2 = R^2\varphi$, а также вспомним, что $R^1L^1 = E, R^2L^2 = E$. Тогда придем к следующим системам уравнений:

$$\omega_t^i = \Lambda^i\omega_i^i, \quad i \in 1,2. \#(20)$$

В каждом окажется по 5 скалярных независимых уравнений переноса, так как Λ^i – диагональные матрицы.

Известно большое количество методов решения скалярных одномерных уравнений переноса $u_t + Cu_x = 0$. Самым простым примером является разностная схема первого порядка CIR, предложенная в работе [7]. Она записывается следующим образом:

$$\frac{u^{m,n+1} - u^{m,n}}{\Delta t} + C\frac{u^{m+1,n} - u^{n,m}}{\Delta x} = 0. \#(21)$$

Данную схему можно интерпретировать как результат линейной интерполяции значения в точке $u^{m^\star,n}$ на основе значений в точках $u^{m,n}$ и $u^{m+1,n}$ для $C < 0$ (или между $u^{m-1,n}$ и $u^{m,n}$ для $C > 0$). Поскольку решение вдоль характеристики постоянно, $u^{m,n+1} = u^{m^\star,n}$.

Для практических расчетов используются более сложные методы решения уравнений переноса, учитывающие дополнительно законы сохранения и позволяющие получать точные решения даже при наличии разрывов в решении и параметрах среды, которые могут возникать на границах пластов земли с разными скоростями.

Для расчетов представленных в работепримениласьсхема Рузанова третьего порядка [8]. После того, как решения ω^i будут пересчитаны с временного слоя t на слой $t + \Delta t$, значения исходных неизвестных находится по формуле $\varphi = L^i\omega^i, i \in 1,2$. Можно переходить к следующей итерации расчета, для вычисления значений φ в момент времени $t + 2\Delta t$, а потом $t + 3\Delta t$ и т.д., пока не будет достигнуто желаемое количество шагов по времени.

4. Распараллеливание MPI

Для работы в системах с распределенной памятью программный комплекс распараллелен с использованием технологии MPI [16]. При распараллеливании использовались стандартные алгоритмы декомпозиции расчетной области и обмена приграничными ячейками, широко применяемые для явных сеточных методов [17-20]. Поскольку в данной реализации сеточно-характеристического решателя используются регулярные сетки, узлы могут храниться в 2D/3D массивах (в памяти они хранятся в виде непрерывных одномерных массивов). На вход программе поступают номера процессов для каждой сетки, которые будут пересчитывать узлы внутри этих сеток. Алгоритм определяет наилучшее разбиение массивов на равные по размеру блоки для распределения работы между процессами путем вызова MPI_Dims_create. Кроме этого возможно вручную задать желаемое распределение процессов. Количество блоков, на которые разбита сетка, такое же, как и число процессов, ассоциированное с сеткой. Это значит, что каждый процесс пересчитывает узлы на следующий временной шаг не более чем для одного блока сетки. В то время как процессы могут быть ответственны не более чем за один блок сетки, они могут одновременно обрабатывать несколько блоков с разных сеток.

Для упрощения синхронизации узлов на границах блоков внутри одной сетки, создается специальный коммуникатор (MPI_Comm) и специальную группу (MPI_Group) для этой сетки. Число процессов, которые даны на входе, определяют список процессов из коммуникатора MPI_COMM_WORLD, на основе которых создается новый коммуникатор. Синхронизация внутри сетки производится между шагами по времени, и каждый процесс может получить информацию о процессах, обрабатывающих соседние блоки. Зная точные номера процессов, которые готовы обмениваться пересчитанными узлами на границах блоков, процессы могут выполнять асинхронные операции приема и передачи (MPI_Isend, MPI_Irecv).

В противоположность синхронизации на границе блоков, при синхронизации на границах контактных поверхностей между двумя сетками, процессы не знают номеров процессов, с которыми обмениваться данными, так как геометрия контактов может быть произвольной. В первоначальной реализации использовалась функция MPI_Alltoall для того, чтобы все процессы коммуникатора MPI_COMM_WORLD содержали одинаковую информацию о декомпозиции. Основной причиной неэффективности такой стратегии являлось то, что процессы, которые не нуждаются в синхронизациях для продолжения расчета, все равно были обязаны участвовать в этом взаимодействии. Другими словами была создана точка глобальной синхронизации на каждом временно шаге, которую можно было устранить.

Тогда были введены доступные всем процессам списки со свойствами всех сеток. Это свойства содержат информацию о размерах сеток, номерах процессов, связанных с сетками, точные положения блоков и их размеры для каждого из этих процессов. Списки свойств создаются в начале инициализации расчетного алгоритма синхронизируются между всеми процессами и далее остаются неизменными (это возможно благодаря статичной структуре сеток и связей между ними). Для каждой сетки возможно получить номера процессов из MPI_COMM_WORLD, которые обрабатывают каждый блок. Следовательно, когда некоторый процесс должен обновить узлы на контактной границе для синхронизации с другим процессом, он знает какой именно процесс содержит данные из другой сетки с этим контактом, поэтому он может установить обмен данными только с этим процессом. Более того, данный подход позволяет учитывать случай, в котором несколько процессов на границе одной сетки обменивается данными с независимо распределенными процессами на границе другой сетки. Это значит, что декомпозиции сеток могут быть независимы друг от друга.

Информация о контактах на границах сеток задается обобщенным способом. Положение области в одной сетке, которая должна быть отправлена в заданную область другой сетки, поступает на вход решателю. Список таких указаний содержит полную информацию о контактах в модели с несколькими сетками. Данная информация используется на каждом шаге по времени, когда происходит синхронизация. Если положения узлов сетки не совпадают в точности, то производится переинтерполяция.

Рассмотрим пример декомпозиции блочных сеток. Разбиение выполнено для модели слоистой среды с криволинейными контактными поверхностями между геологическими слоями. Модель имеет 8 слоев, каждый из которых обозначен отдельным цветом на рис. 1.

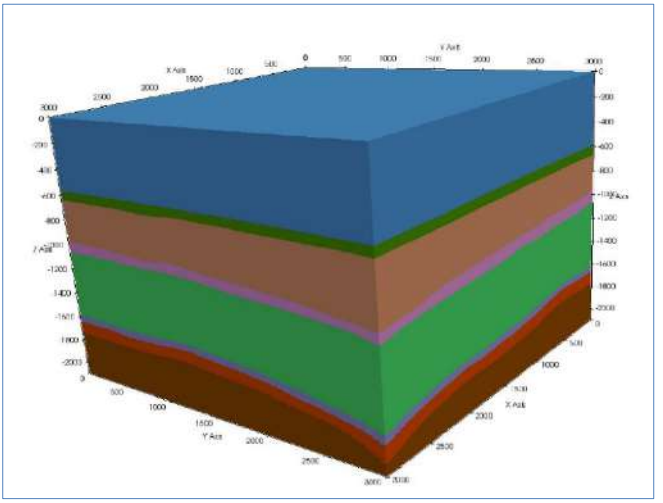

Рис. 1. Структура геологических слоев
Fig. 1. The structure of geological layers

Количество узлов во всей расчетной области 601 x 601 x 421. На рис. 2 разными цветами изображены блоки с узлами, пересчитываемые разными процессами. Это упрощенный вариант с 16 процессами, каждый цвет обозначает один процесс. Для наглядности слои также были разделены по оси Z, хотя в реальном расчете для сеток с небольшим количество узлов по оси Z такое разбиение не производилось, поскольку значительно ухудшало производительность.

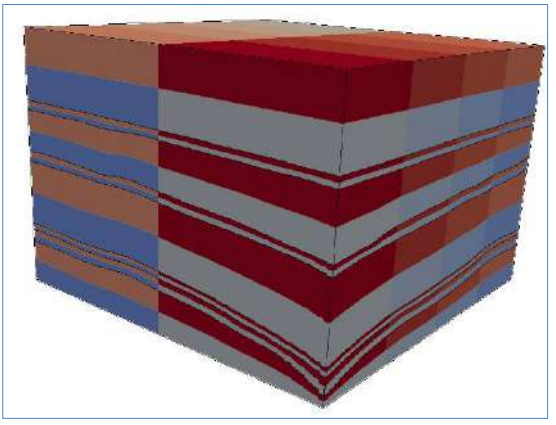

Рис. 2. Распределение 16 процессов по сеткам. Каждая сетка разделена на 4 × 2 × 2 блоков
Fig. 2. Distribution of 16 processes on grids. Each grid is divided into 4 × 2 × 2 blocks

Была измерена производительность нашей параллельной реализации сеточно-характеристического метода. Результаты изображены на рис. 3. В тесте использовалась расчетная сетка размером 1000x1000x1000 узлов. Тестирование проводилось на кластере НЕСТоR. Данный суперкомпьютер состоит из 2816 вычислительных узлов. Каждый из узлов оснащен двумя процессорами 16-core AMD Opteron 2.3GHz Interlagos. Оперативная память составляет 32 ГБ на узел. Отсчет идет от 128 процессов MPI, для того, чтобы получить корректные результаты, так и при малом числе процессов времена расчетов могут быть неприемлемо долгими. Можно заметить, что представленная реализация масштабируется до 4096 процессов практически без потери эффективности. При увеличении числа процессов наблюдается ожидаемое снижение эффективности, однако алгоритм все еще демонстрирует хорошие показатели даже на 16384 процессах.

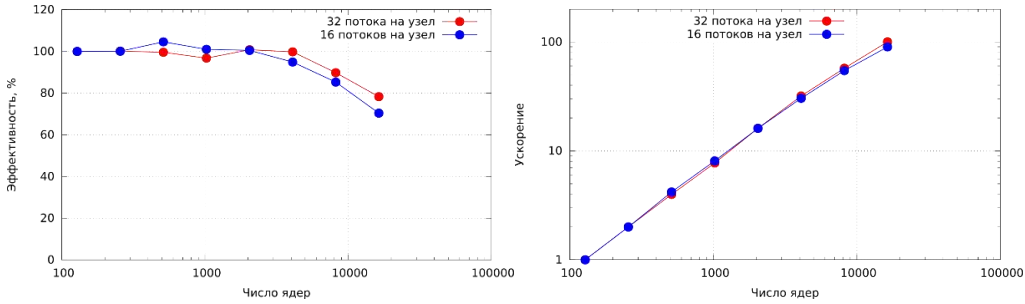

Рис. 3. Эффективность (слева) и масштабируемость (справа) параллельного алгоритма в среде MPI
Fig. 3. Efficiency (left) and scalability (right) of the parallel algorithm in the MPI environment

5. Распараллеливание в средах с общей памятью

Было проведено распараллеливание для систем с общей памятью с использованием технологий OpenMP [10, 21] и POSIX Threads. Полученное ускорение приведено на рис. 4. В силу несущественных различий в ускорении и простоты внедрения параллельного кода с применением OpenMP из них было решено использовать в расчетах только технологию OpenMP. Код распараллеливается согласно принципу геометрического параллелизма. Производится разбиение расчетной сетки на прямоугольные области, каждая из которых пересчитывается одним потоком OpenMP. Таким же образом разбиваются наложенные сетки, используемые для задания трещин. Глобальная синхронизация потоков происходит между шагами по времени. При этом происходит обмен значениями в узлах, которые находятся на границе разбиений и должны быть доступны одновременно нескольким потокам.

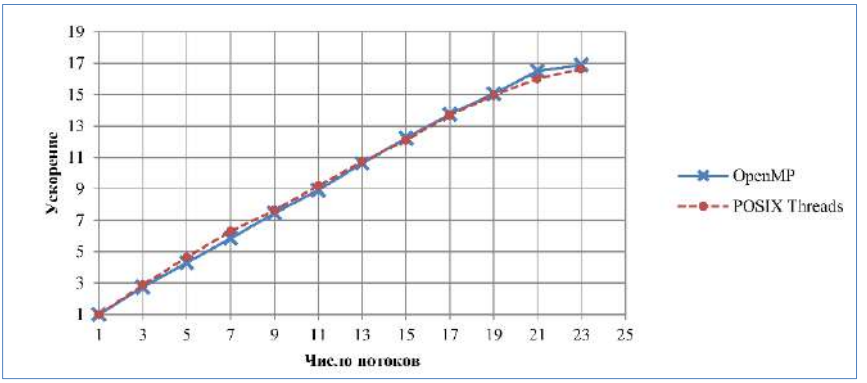

Рис. 4. Зависимость ускорения от числа потоков для систем с общей памятью
Fig. 4. Dependence of acceleration on the number of threads for systems with shared memory

Достигнута 74% эффективность ускорения полученного алгоритма на машине с процессорами Intel Xeon E5-2697 (на 24 ядрах).

6. Распараллеливание на графических процессорах

Алгоритм также был распараллелен на графических GPGPU процессорах NVidia используя технологию CUDA. Данная технология широко применяется для распараллеливания, в том числе явных, вычислительных алгоритмов [11-15]. Потребовалось полное переписывание части расчетного модуля под данную архитектуру [21]. За основу для реализации алгоритма на графических процессорах была взята оптимизированная для выполнения на центральном процессоре версия данной программы. Оптимизациям была подвергнута наиболее затратная в вычислительном плане часть алгоритма. Так как использовалось расщепление по пространственной координате, для пересчета всей сетки требовалось два шага: по оси X и по оси Y. В первоначальной версии (cuda1 на рис. 5) на графическом устройстве выделяется в 2 раза больше памяти, чем требуется для хранения расчетной сетки. В памяти хранятся две ее копии.

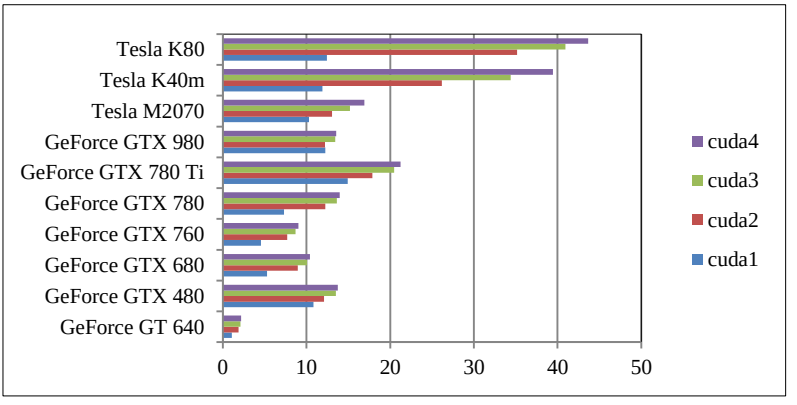

Рис. 5. Ускорение по сравнению с CPU
Fig. 5. Acceleration over CPU

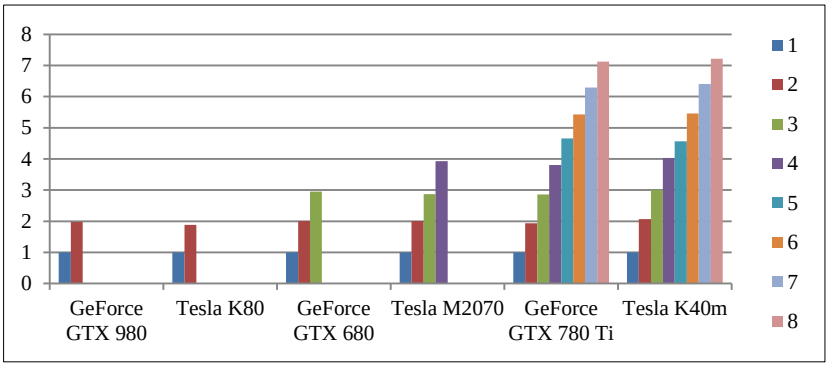

Рис. 6. Ускорение в зависимости от количества графических устройств
Fig. 6. Acceleration depending on the number of graphics devices

На каждом шаге происходит пересчет значений узлов сетки, хранящихся в одной из копий. При этом результат вычислений записывается в другую копию расчетной сетки. В результате происходит синхронизация только между вызовами kernel'ов CUDA. Это было сделано по причине того, что глобальная синхронизация всего устройства создает большие временные задержки. Таким образом, удалось уменьшить количество глобальных синхронизаций до двух раз на один шаг по времени. Следующая оптимизация (cuda2) состояла в использовании общей памяти (shared memory) устройства. Чтение из глобальной памяти происходит только один раз на каждом шаге, причем обращения становятся последовательными (coalesced), что также ведет к повышению производительности. Далее используются оптимизации, дающие незначительный прирост производительности. В версии cuda3 дополнительные данные вычисляются на CPU и передаются через параметры вызова kernel'а. Таким образом, ожидается, что у каждого потока будет создана локальная копия этих значений. В версии cuda4 подбираются размеры блоков таким образом, чтобы минимизировать количество узлов, для которых требуются обмены памятью между блоками.

Для выполнения на нескольких GPU использовался наиболее оптимизированный вариант. Также была применена технология GPUDirect, позволяющая передавать данные через шину PCI Express в обход центрального процессора. Максимальное полученное ускорение по сравнению с CPU на одном графическом устройстве – в 55 раз на GeForce GTX 780 Ti в вычислениях с одинарной точностью и в 44 раза на Tesla K80 в вычислениях с двойной точностью (рис. 5). Максимальное достигнутое ускорение от количества графических устройств - в 7.1 раз на 8 графических устройствах (рис. 6) для двойной точности. Технология GPUDirect повысила ускорение на 10% от того, что было достигнуто без нее при вычислениях с одинарной точностью. При вычислениях с двойной точностью ускорение составило 2.4%.

7. Примеры расчетов

Используется модель слоистой среды с криволинейными контактами между слоями [22]. Модель имеет 8 слоев, каждый из которых обозначен отдельным цветом на рис. 1. Источник волн задан в виде взрыва с помощью применения вертикальной силы в объеме верхнего слоя. Изменение этой силы во времени задано с помощью импульса Рикера с частотой 30 Гц. Поверхность верхнего слоя считается плоской. На дневной поверхности задано условие свободной границы, а на всех остальных границах – поглощающее граничное условие. На рис. 7 приведена сейсмограмма с удаленным импульсом с источника, приведен вертикальный компонент скорости.

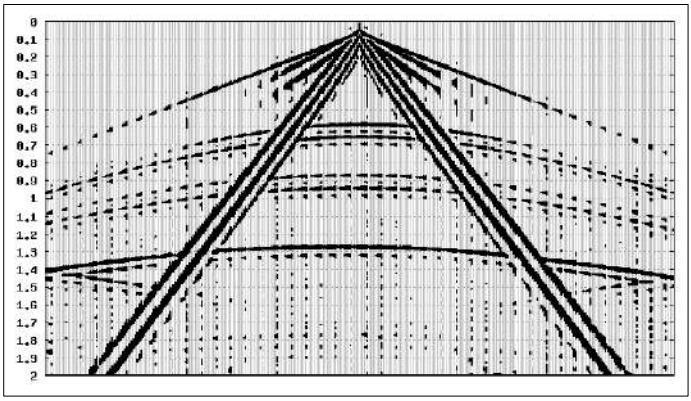

Рис. 7. Расчетная сейсмограмма, вертикальный компонент скорости6
Fig. 7. Design seismogram, vertical velocity component

Еще один пример работы алгоритма проводился для стандартной модели SEG C3 NA [23]. На рис. 8 приведено распределение скорости продольных волн в пространстве.

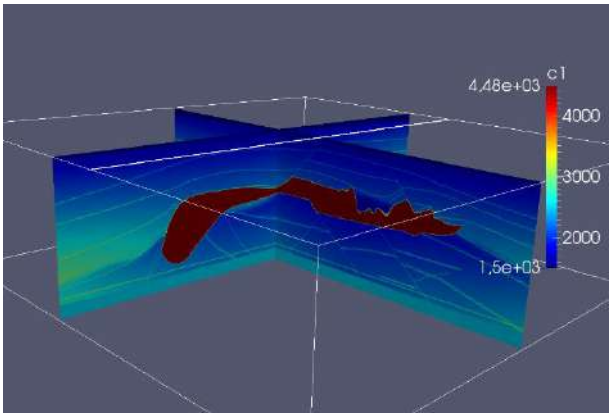

Рис. 8. Модель SEG C3 NA, цветом отображено распределение скорости продольных волн в среде
Fig. 8. Model SEG C3 NA, color shows the distribution of the velocity of longitudinal waves in the medium

Расчет проводился на сетке размером 676x676x201 узел. Источник – импульс Риккера с частотой 30.0 Гц. На рис. 9 приведена сейсмограмма, полученная от источника. Отображен вертикальный компонент скорости.

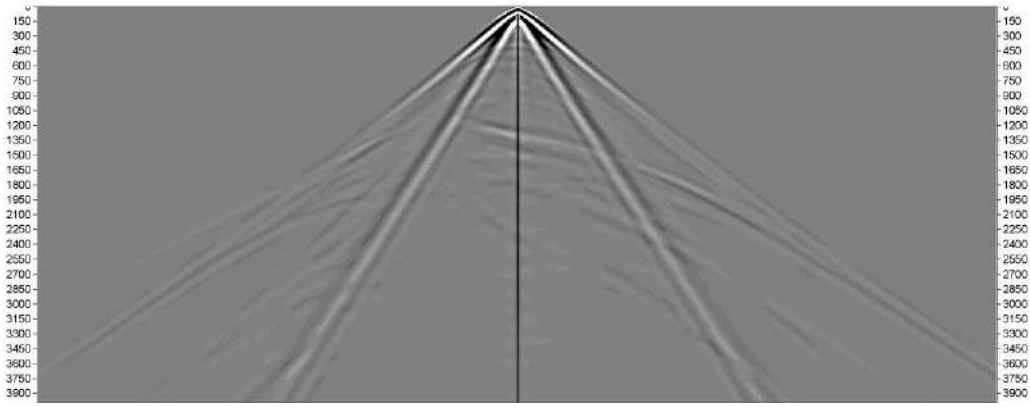

Рис. 9. Сейсмограмма вертикальной составляющей скорости для модели SEG C3 NA
Fig. 9. Seismogram of the vertical velocity component for the SEG C3 NA model

На рис. 10 приведены волновые картины в среде в последовательные моменты времени.

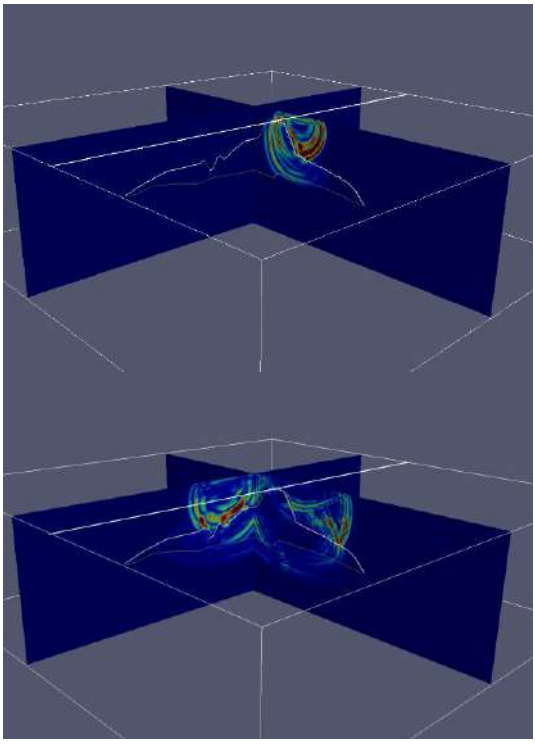

Рис. 10. Волновые картины для модели SEG C3 NA в последовательные моменты времени. Линией обозначен контур соляного купола
Fig. 10. Wave patterns for the SEG C3 NA model at successive times. The line indicates the outline of the salt dome

8. Выводы и результаты

В работе рассмотрен программный комплекс, предназначенный для моделирования распространения динамических волновых возмущений в твердых телах. В основе расчетного алгоритма лежит сеточно-характеристических метод решения систем гиперболических уравнений в частных производных. Данный метод позволяет учитывать волновую структуру уравнения и производить корректную постановку граничных и контактных условий. Реализован алгоритм, позволяющий производить расчет используя блочно-структурные сетки. Алгоритм распараллелен используя технологии MPI, OpenMP, CUDA. В настоящее время достигнута эффективность распараллеливания до 70% с использованием технологии MPI при масштабировании до 16 тысяч вычислительных ядер. В системах с общей памятью алгоритм распараллелен с использованием технологии OpenMP. Кроме того, код распараллелен с использованием технологии CUDA, что дает ускорение до 50 раз по сравнению с одним ядром CPU. Программа может использовать несколько карточек в рамках одного хоста. В данной работе рассмотрены результаты одного и того же алгоритма используя различные технологии. Приведены тесты распараллеливания до 16 тысяч ядер CPU и 8 устройств CUDA. Показана работоспособность алгоритма на тестовых примерах.

Список литературы / References

[1]. V.A. Biryukov, V.A. Miryakha, I.B. Petrov, and N.I. Khokhlov. Simulation of elastic wave propagation in geological media: Intercomparison of three numerical methods. Computational Mathematics and Mathematical Physics, vol. 56, no. 6, 2016, pp. 1086–1095.

[2]. Голубев В.И., Петров И.Б., Хохлов Н.И. Численное моделирование сейсмической активности сеточно-характеристическим методом. Журнал вычислительной математики и математической физики, том 53, № 10, 2013 г., стр. 1709 – 1720 / Golubev V.I., Petrov I.B., Khokhlov N.I. Numerical simulation of seismic activity by the grid-characteristic method. Computational Mathematics and Mathematical Physics, vol. 53, № 10, 2013, pp. 1523–1533.

[3]. P.L. Roe. Characteristic-Based Schemes for the Euler Equations. Annual Review of Fluid Mechanics, № 18, 1986, pp. 337-365.

[4]. LeVeque R.J. Finite volume methods for hyperbolic problems. Cambridge Texts in Applied Mathematics, vol. 31. Cambridge university press, 2002, 552 p.

[5]. LeVeque R.J. Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems. Other Titles in Applied Mathematics, vol. 98. Siam, 2007, 328 p.

[6]. Strang G. On the construction and comparison of difference schemes. SIAM Journal on Numerical Analysis, vol. 5, no. 3, 1968, pp. 506–517.

[7]. Courant R., Isaacson E., Rees M. On the solution of nonlinear hyperbolic differential equations by finite differences. Communications on Pure and Applied Mathematics, vol. 5, no. 3, 1952, pp. 243–255.

[8]. Русанов В.В. Разностные схемы третьего порядка точности для сквозного счета разрывных решений. Доклады АН СССР, том 180, no. 6, 1968 г., стр. 1303–1305 / Rusanov V.V. Difference schemes of the third order of accuracy for the forward calculation of discontinuous solutions. Doklady Akademii Nauk SSSR, vol. 180, no. 6, pp. 1303–1305 (in Russian).

[9]. Favorskaya A.V., Petrov I.B. Grid-characteristic method. Innovations in Wave Processes Modelling and Decision Making. In Innovations in Wave Processes Modelling and Decision Making, Springer, 2018, pp. 117–160.

[10]. Dagum L., Menon R. OpenMP: an industry standard API for shared-memory programming. IEEE computational science and engineering, vol. 5, no. 1, 1998, pp. 46–55.

[11]. Nakata N., Tsuji T., Matsuoka T. Acceleration of computation speed for elastic wave simulation using a Graphic Processing Unit. Exploration Geophysics, vol. 42, no. 1, 2011, pp. 98–104.

[12]. Weiss R. M., Shragge J. Solving 3D anisotropic elastic wave equations on parallel GPU devices. Geophysics, vol. 78, no. 2, 2013, 22 p.

[13]. F. Rubio et al. Finite-difference staggered grids in GPUs for anisotropic elastic wave propagation simulation. Computers & geosciences, vol. 70, issue C, 2014, pp. 181–189.

[14]. Komatitsch D., Michéa D., Erlebacher G. Porting a high-order finite-element earthquake modeling application to NVIDIA graphics cards using CUDA. Journal of Parallel and Distributed Computing, vol. 69, no. 5, 2009, pp. 451–460.

[15]. D. Komatitsch et al. Modeling the propagation of elastic waves using spectral elements on a cluster of 192 GPUs. Computer Science – Research and Development, vol. 25, no. 1/2, 2010, pp. 75–82.

[16]. Message Passing Interface Forum. MPI: a Message-Passing Interface Standard. Technical Report. University of Tennessee, 1994.

[17]. Якобовский М.В. Введение в параллельные методы решения задач: Учебное пособие. М., Издательство Московского университета, 2013, 328 стр. / Yakobovsky M.V. An Introduction to Parallel Problem Solving Methods: A Training Manual. M., Moscow University Publishers, 2013, 328 p.

[18]. Ivanov A.M., Khokhlov N.I. Efficient Inter-process Communication in Parallel Implementation of Grid-Characteristic Method. In Smart Innovation, Systems and Technologies, vol. 133, 2019, pp. 91–102.

[19]. Иванов А.М., Хохлов Н.И. Параллельная реализация сеточно-характеристического метода в случае явного выделения контактных границ. Компьютерные исследования и моделирование, том 10, № 5, 2018 г., стр. 667-678 / Ivanov A.M., Khokhlov N.I. Parallel implementation of the grid-characteristic method in the case of explicit contact boundaries. Computer Research and Modeling, vol. 10. № 5, 2018, pp. 667–678 (in Russian).

[20]. Khokhlov N. et al. Solution of Large-scale Seismic Modeling Problems. Procedia Computer Science, vol. 66, 2015, pp. 191–199.

[21]. Khokhlov N. et al. Applying OpenCL Technology for Modelling Seismic Processes Using Grid-Characteristic Methods. Communications in Computer and Information Science, vol. 678, 2016, pp. 577–588.

[22]. Golubev V.I. et al. Simulation of dynamic processes in three-dimensional layered fractured media with the use of the grid-characteristic numerical method. Journal of Applied Mechanics and Technical Physics, vol. 58. № 3, 2017, 539–545.

[23]. Aminzadeh F., Brac J., and Kunz T. 3D Salt and Overthrust models. In Distribution CD of Salt and Overthrust models, SEG/EAGE Modeling Series, No. 1, 1997.

[24]. Фаворская А.В., Петров И.Б. Численное моделирование волновых процессов в скальных массивах сеточно-характеристическим методом. Математическое моделирование, том 30, № 3, 2018 г., стр. 37–51 / Favorskaya A. V., Petrov I.B. Numerical Modeling of Wave Processes in Rocks by the Grid-Characteristic Method. Mathematical Models and Computer Simulations, vol. 10. issue 5, 2018, pp. 639–647.

Информация об авторах / Information about authors

Николай Игоревич ХОХЛОВ, кандидат физико-математических наук, работает в МФТИ в должности старшего научного сотрудника, заместителя заведующего лабораторией прикладной вычислительной геофизики. Научные интересы включают численное моделирование, параллельные алгоритмы, высокопроизводительные вычислительные системи и технологии написания параллельных приложений.

Nikolai Igorevich KHOKHLOV, candidate of physical and mathematical sciences, works at MIPT as a senior researcher, deputy head of the laboratory of applied computational geophysics. His research interests include numerical modeling, parallel algorithms, high-performance computing systems, and parallel application writing technologies.

Игорь Борисович ПЕТРОВ является доктором физико-математических наук, профессором, членом-корреспондентом РАН, заведующим кафедры информатики МФТИ. В число научных интересов входят сеточно-характеристический метод, математическое моделирование, высокопроизводительные вычислительные системы

Igor Borisovich PETROV is a doctor of physical and mathematical sciences, professor, corresponding member of the Russian Academy of Sciences, head of the department of computer science at the Moscow Institute of Physics and Technology. His research interests include the grid-characteristic method, mathematical modeling, and high-performance computing systems.