ИСП Институт Системного Программирования Российской Академии наук

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

Труды Института Системного Программирования РАН

Proceedings of the Institute for System Programming of the RAS

Том 29, выпуск 1

Volume 29, issue 1

Москва 2017

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

Труды Института системного программирования РАН

Proceedings of the Institute for System Programming of the RAS

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

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

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

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

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

Труды ИСП РАН реферируются и/или индексируются в:

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

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

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

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

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

УДК004.45

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

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

Содержание

Table of Contents

Численное решение задачи обтекания клина потоком стратифицированной жидкости с использованием OpenFOAM

Н.Ф. Димитриева <dimitrieva@list.ru>
Институт гидромеханики НАНУ,
03680, Украина, г. Киев, ул. Желябова, д. 8/4

Аннотация. Представлены результаты численного моделирования течений устойчиво стратифицированной жидкости на примере задачи обтекания клина в двумерной нестационарной постановке в широком диапазоне скоростей. Стратифицированные течения характеризуются широким диапазоном значений внутренних масштабов, отсутствующих в однородной жидкости. Все элементы течений (вихри, волны, тонкоструктурные прослойки) существуют одновременно и активно взаимодействуют между собой. Предложена система балансных уравнений, которая дает возможность одновременного изучения всех элементов течений в рамках единого описания в естественных физических переменных без привлечения дополнительных констант и связей. Поставленная задача решалась методом конечных объемов в открытом пакете OpenFOAM. Особое внимание уделялось созданию качественной высокоразрешающей расчетной сетки, которая учитывает многомасштабность поставленной задачи. Тестовые расчеты подтвердили необходимость разрешения минимальных диффузионных микромасштабов. Обсуждаются вопросы использования стандартных и расширенных утилит пакета OpenFOAM с целью реализации сложных граничных условий и разработки собственных решателей. В качестве начальных условий задачи обтекания клина внешним потоком стратифицированной среды использовались ранее рассчитанные поля течений, индуцированных прерыванием диффузионного переноса неподвижным клином, которые качественно согласуются с данными лабораторных опытов. Расчеты проводились в параллельном режиме с использованием сервисов платформы UniHUB. Единая система уравнений и общий алгоритм были использованы во всем диапазоне параметров задачи. Результаты расчетов показали сложную нестационарную структуру стратифицированных течений около клина. Определены механизмы формирования вихрей в областях больших градиентов возмущения солености в окрестности кромок препятствия. Во всех режимах течение характеризуется сложной внутренней структурой, в которой вначале выражены диссипативно-гравитационные волны, затем группа присоединенных волн, которые образуются в противофазе у кромок клина. Далее основным компонентом течения становятся вихри, которые начинают формироваться около передней кромки и становятся выраженными в следе. С увеличением скорости вся картина течения становится более нестационарной, подвижные вихри заполняют все поле вблизи тела и в следе за ним.

Ключевые слова: численное моделирование; открытые вычислительные пакеты; стратифицированные течения

DOI: 10.15514/ISPRAS-2017-29(1)-1

Для цитирования: Димитриева Н.Ф. Численное решение задачи обтекания клина потоком стратифицированной жидкости с использованием OpenFOAM. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 7-20. DOI: 10.15514/ISPRAS-2017-29(1)-1

1. Введение

В настоящей работе изучаются механизмы формирования, развития, распада волновых и вихревых компонент течений в стратифицированной среде, где волны и вихри существуют одновременно и активно взаимодействуют между собой. Сложность подхода – внутренняя многомасштабность, отражающая присутствие крупных компонент течений (вихри, внутренние волны в толще жидкости) и тонких высокоградиентных прослоек, разделяющих структурно отличающиеся области течения [1].

Гравитационные (внутренние) волны являются важным элементом динамики морской среды и атмосферы, они переносят на большие расстояния энергию и импульс, интенсифицируют перенос вещества [2, 3]. Научный интерес к данной проблеме обусловлен необходимостью изучения ряда явлений в окружающей среде, таких как интенсивные долинные или горные ветры в атмосфере и склоновые потоки в океане [4, 5], а также самодвижение объектов [6].

Численными методами исследуется формирование тонкой структуры течений, которая влияет на перенос вещества, процессы разделения компонент течения и повышения локальной концентрации примесей. Одновременный расчет всех макро- и микрокомпонент течений в полной нелинейной постановке представляет собой сложную актуальную задачу, на сегодняшний день не решенную с практически необходимой степенью точности. Полученные результаты представляют важность для фундаментальной и прикладной аэро- и гидродинамики поскольку дают более глубокое понимание физических процессов в стратифицированных средах благодаря применению в численном моделировании фундаментальной системы уравнений механики неоднородных жидкостей, учитывающей влияние реальных свойств среды и внешних динамических факторов.

Целью работы является развитие методики численного моделирования течений непрерывно стратифицированных жидкостей с учетом геометрии препятствий, эффектов вязкости и диффузии.

2. Математическая модель

Решается нестационарная плоская задача обтекания горизонтального клина длиной L = 10 см и шириной h = 2 см потоком устойчиво стратифицированной жидкости с периодом плавучести $T_b = 6,28$ с.

2.1 Система уравнений

В качестве математической модели изучаемых физических процессов выбрана фундаментальная система дифференциальных балансных уравнений механики неоднородных многокомпонентных жидкостей в приближении Буссинеска, когда малые изменения плотности на масштабах задачи учитывается только в членах с силой тяжести [1]:

$$\begin{aligned} &\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v}\nabla)\mathbf{v} = -\frac{1}{\rho_{00}}\nabla P + \nu\Delta\mathbf{v} - s\mathbf{g}, \qquad \nabla\mathbf{v} = 0, \\ &\frac{\partial s}{\partial t} + \mathbf{v}\cdot\nabla s = \kappa_s \Delta s + \frac{v_y}{\Lambda}, \quad \rho = \rho_{00}\left(\exp(-y/\Lambda) + s\right) \end{aligned} \tag{1}$$

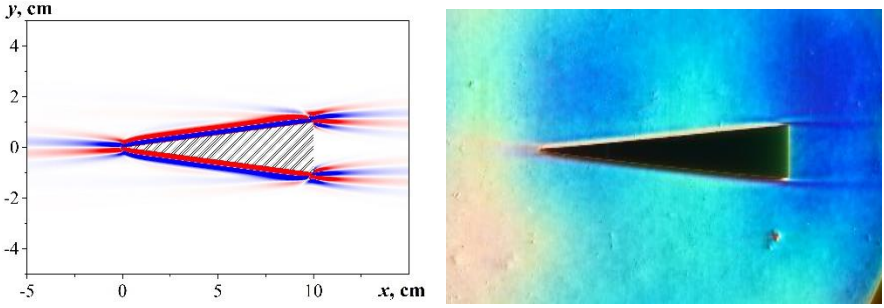
Здесь $S = S_0(y) + s$ – полная соленость, s – ее возмущенная составляющая, ρ_{00} – плотность на нулевом уровне (горизонте нейтральной плавучести), $\rho(y)$ – невозмущенное распределение плотности, которое задается профилем солености $S_0(y)$, где ось y направлена вертикально вверх, $\mathbf{v}$ – вектор скорости жидкости, P – давление за вычетом гидростатического, ν – коэффициент кинематической вязкости, κ_s – коэффициент диффузии соли, t – время, $\mathbf{g}$ – ускорение свободного падения, ∇ и Δ – операторы Гамильтона и Лапласа, $\Lambda = (d\ln\rho_0/dy)^{-1}$ – длина плавучести, $N = \sqrt{g/\Lambda}$ – частота плавучести.

2.2 Начальные и граничные условия

В качестве начального состояния стратифицированной среды рассматривается установившееся течение, индуцированное прерыванием диффузионного переноса неподвижным клином [7]. Такие течения характеризуются сложной ячеистой структурой и наличием высокоградиентных областей, визуализируемых в виде протяженных горизонтальных прослоек (рис.1).

Важную роль играют краевые эффекты, где схождение с острых кромок клиновидного тела тонких струйных течений жидкости, формирующихся вдоль каждой из его сторон, порождает внутренние волны [8]. Обычно тонкоструктурные эффекты вносят небольшие поправки в значения характеристик течений, но их действие усиливается большой величиной градиентов солености, поля которых отражают сложную периодическую

структуру течений, индуцированных диффузией (рис. 1, а). Горизонтальные полосчатые структуры качественно согласуются с экспериментальным картинам визуализации ("цветной теневой метод" с горизонтальной щелью и решеткой) распределения градиента коэффициента преломления в лабораторном бассейне ИПМех РАН (рис. 1, б).



а) *б)*

Рис. 1. Рассчитанная (а) и теневая (в) картины течения, индуцированного диффузией на клине

Fig. 1. Calculated (a) and experimental (c) pattern of flows induced by diffusion on a wedge

Физически обоснованные граничные условия для задачи обтекания горизонтальной пластины потоком непрерывно стратифицированной жидкости имеют следующий вид:

$$\mathbf{v}\big|_{\Sigma}=0,\quad v_x\big|_{x,y\to\infty}=U,\quad v_y\big|_{x,y\to\infty}=0,\quad s\big|_{x,y\to\infty}=0,\quad \left.\frac{\partial S}{\partial n}\right|_{\Sigma}=-\frac{1}{\Lambda}\left.\frac{\partial y}{\partial n}+\frac{\partial s}{\partial n}\right|_{\Sigma}=0. \quad (2)$$

где $\mathbf{n}$ – внешняя нормаль к поверхности препятствия Σ. На большом удалении от препятствия задаются условия затухания всех возмущений, U – скорость внешнего обтекания препятствия.

Адекватность выбранной математической модели подтверждается соответствием основополагающим принципам механики и согласованностью независимых аналитических, численных и экспериментальных исследований стратифицированных течений около пластины и полуплоскости [1, 2].

2.3 Характерные масштабы

Задача характеризуется набором размерных параметров: $\nu=10^{-6}$ м^2/с, $k_s=1.41\cdot10^{-9}$ м^2/с, g = 9.8 м/с^2, $N=1$ с$^{-1}$. Они формируют характерные масштабы: времени $t=T_b$, скорости – $U_N^{\nu}=\sqrt{\nu N}$, $U_N^{\kappa_S}=\sqrt{\kappa_S N}$, U, а также длины. Большие линейные масштабы характеризуют исходную стратификацию (длину плавучести Λ) и геометрию течения (размеры препятствия L и h). Скорость источника U задает длину гравитационных

поверхностных $\lambda_s = 2\pi U^2/g$ и внутренних $\lambda_i = UT_b$ гравитационных волн. Микромасштабы диссипативной природы (вязкий $\delta_N^{\nu} = \sqrt{\nu/N}$ и диффузионный $\delta_N^{\kappa_S} = \sqrt{\kappa_S/N}$ микромасштабы) определяют поперечные размеры тонкоструктурных компонентов. Компоненты структур с масштабами Прандтля $\delta_U^{\nu} = \nu/U$ и $\delta_U^{\kappa_s} = \kappa_s/U$ выражены в струях и следах.

Широкий диапазон значений масштабов длины указывает на сложность внутренней структуры стратифицированного течения, которую необходимо учитывать при разработке численной модели. Анализ линеаризованных фундаментальных уравнений и результаты лабораторного моделирования показывают, что крупномасштабные элементы течений (волны и вихри) характеризуют регулярно возмущенные компоненты полного решения [9]. Обширное семейство сингулярно возмущенных компонент описывает тонкоструктурные элементы течений, которые проявляются во всем диапазоне параметров изучаемых процессов. Все компоненты активно взаимодействуют между собой, формируя эволюционирующую тонкую структуру, которая влияет на перенос вещества, процессы разделения компонент течений.

Рассмотренная постановка задачи позволяет одновременный расчет всех элементов течений в рамках единого описания в естественных физических переменных без привлечения дополнительных констант и связей. Макромасштабы характеризуют размер области решения задачи, которая должна содержать все изучаемые компоненты течения, а микромасштабы – пространственное разрешение расчетной сетки.

3. Численная модель

Численное решение поставленной задачи было реализовано в свободно распространяемом пакете OpenFOAM, открытость исходного кода которого позволила построить оригинальный решатель, реализующий систему уравнений (1) методом конечных объемов. Для учета эффектов стратификации и диффузии стандартный решатель пакета icoFOAM был дополнен новыми переменными (ρ и s) и соответствующими уравнениями, а также новыми вспомогательными параметрами (Λ, κ_s, N, $\mathbf{g}$ и др.) [10]. Граничное условие возмущения солености s на поверхности клина реализовано с использованием утилиты funkySetBoundaryField, которая позволяет задавать аналитические выражения в выбранных подобластях границы расчетного домена.

Интерполяция конвективных членов проводилась по TVD схеме (Total Variation Diminishing), которая вносит минимальную численную диффузию и обеспечивает отсутствие осцилляций решения [11]. Для дискретизации производной по времени прибегали к неявной трехточечной несимметричной схеме второго порядка с разностями назад (backward differencing).

Для решения полученной системы линейных алгебраических уравнений применялись итерационные методы сопряженных градиентов с предобусловливанием PCG для симметричных матриц, а для асимметричных матриц — метод бисопряженных градиентов с предобусловливанием PBiCG. В качестве предобусловливателя для симметричных матриц была выбрана процедура DIC, базирующаяся на упрощенной схеме неполной факторизации Холецкого. Для асимметричных матриц, соответственно, запускался предобусловливатель DILU. Для связанного расчета полей скорости и давления использовался устойчивый, хорошо сходящийся алгоритм PISO, который показал высокую эффективность в нестационарных задачах.

Дискретизация расчетной области осуществлялась с использованием утилит blockMesh пакета OpenFOAM. Геометрия изучаемого тела позволила построить блочно-структурированную гексаэдральную расчетную сетку с совмещением линий на границах блоков. Область решения задачи предложено разделить на семь блоков, как показано на рис. 2.

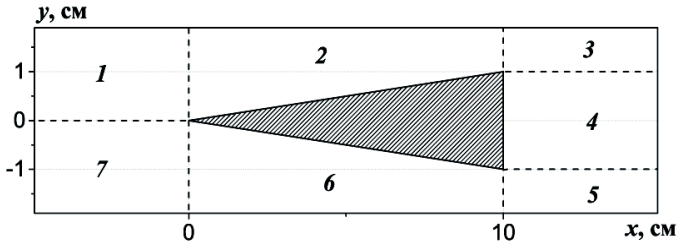


Рис. 2. Схема разбиения расчетной области на блоки

Fig. 2. Partition scheme of the computational domain

Тестовые расчеты с различным измельчением расчетной сетки подтвердили необходимость разрешения наименьших микромасштабов задачи δ_N^{ν} и $\delta_N^{\kappa_S}$ [12]. На рис. 3 приведены зависимости от времени давления вблизи экстремальной вершины клина, где наиболее отчетливо проявляются тонкоструктурные компоненты индуцированных диффузией течений.

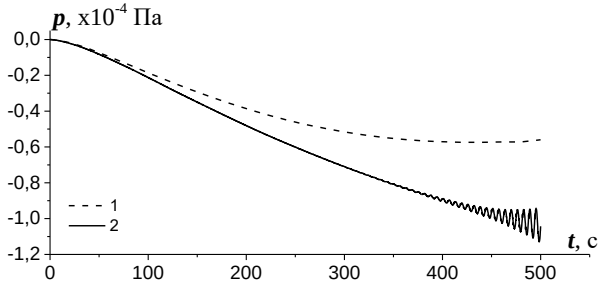
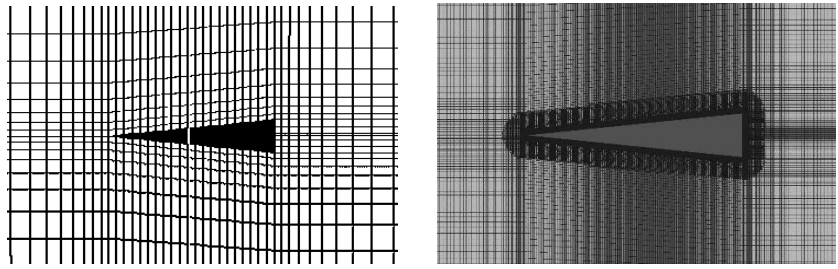


Рис. 3. Эволюция во времени давления в контрольной точке с координатами (0,101;0,011): 1 – сетка с минимальным размером ячейки $0{,}5\cdot10^{-4}$ м в окрестности контрольной точки, 2 – $1{,}5\cdot10^{-4}$ м

Fig. 3. Evolution of the pressure at the control point with coordinates (0,101;0,011): 1 – grid with a minimum size of the cell $0{,}5\cdot10^{-4}$ m near the control point, 2 – $1{,}5\cdot10^{-4}$ m

Решение является неустойчивым на грубой сетке *2* с общим количеством ячеек $0{,}5\cdot10^{6}$ ячеек при минимальном размере в окрестности контрольной точки около $1{,}5\cdot10^{-4}$ м. Для сравнения сетка *1* с минимальным размером ячейки $0{,}5\cdot10^{-4}$ м имеет 10^{6} ячеек.

Алгоритм разбиения расчетной области предполагает сгущение ячеек в направлении препятствия при условии сохранения соотношения размеров граней гексаэдров не более 2, как показано на рис. 4,*а*. Вблизи обтекаемого тела соотношение размеров гексаэдров приблизительно равнялось единице, что положительно влияет на сходимость решения. Однако в этом случае необходимость измельчения сетки в одной подобласти течения влечет излишне мелкую сетку в других областях, где особой потребности в мелкой сетке нет, что приводит к нерациональному использованию вычислительных ресурсов.



а) *б)*

Рис. 4. Схема разбиения расчетной области с линейным сгущением (а) и с дополнительным локальным разбиением (б)

Fig. 4. Partition scheme of the computational domain linear condensation (a) and additional local partition (b)

С целью улучшения качества дискретизации области решения задачи дополнительно использовались утилиты topoSet и refineMesh, позволяющие на основе геометрических либо параметрических признаков выделять подобласти расчетной сетки и локально измельчать их в соответствии с заданными масштабами и выбранными направлениями (рис. 4,*б*). Минимальный размер ячейки $0{,}2\cdot10^{-4}$ м вблизи непроницаемых границ удовлетворительно разрешает диффузионный микромасштаб $\delta_N^{\kappa_S}$ при относительно небольшом общем количестве ячеек сетки, равном $0{,}44\cdot10^{6}$.

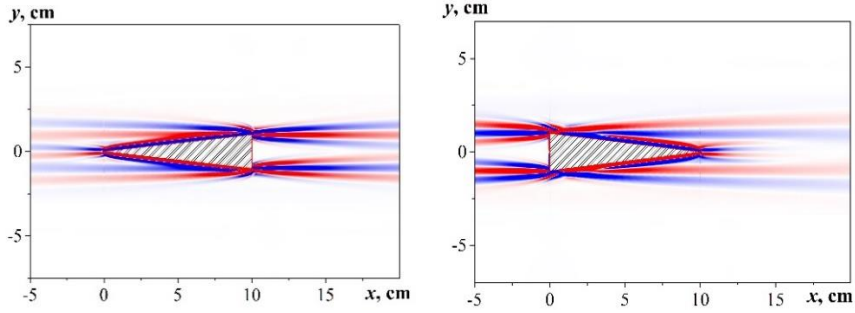
Вместе с тем уменьшение пространственного разбиения, даже на небольшом участке расчетной области, требует соответственного уменьшения шага по времени, что увеличивает время расчета. Существенным недостатком поэтапного разбиения сетки является резкое изменение размера ячейки на границе заданной области, что может отразится на результатах вычислений.

Вычисления проводились в параллельном режиме с привлечением сервисов платформы UniHUB. Проведение тестовых параллельных вычислений с достаточно высоким пространственным разрешением расчетной области показали эффективность распараллеливания счета даже в двумерном случае. Так, на 8 ядрах расчет одной итерации по времени занял около 37 сек, на 16 ядрах – 7 сек, а на 24 – 3 сек. С дальнейшим увеличением числа задействованных ядер скорость вычислений практически не меняется, поэтому в данном конкретном случае проведение расчетов на 24 ядрах кластерной системы является наиболее оптимальным.

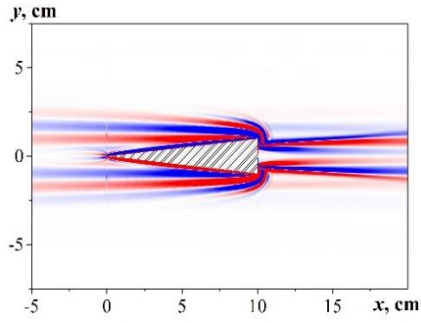
4. Результаты и обсуждение

В рамках настоящей работы были проведены расчеты обтекания клина потоком стратифицированной жидкости в диапазоне скоростей $U = 10^{-5} \div 10^{-1}$ м/с.

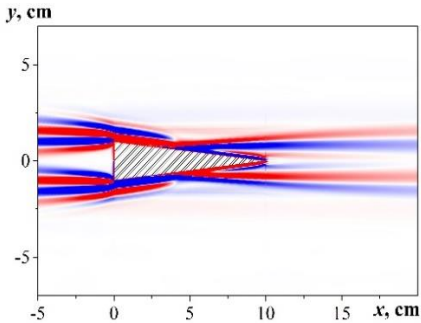
Сложную многомасштабную структуру стратифицированных течений иллюстрирует поле горизонтальной компоненты градиента возмущения солености (рис. 5). В толще непрерывно стратифицированной жидкости формируются опережающие возмущения, розетки нестационарных и поля присоединенных внутренних волн, а также протяжённый след за экстремальными точками.



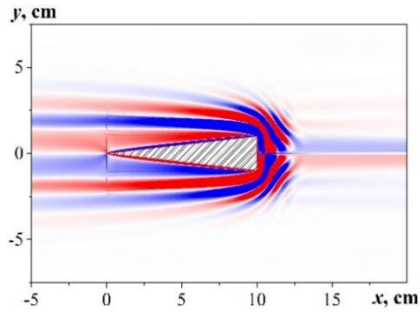
а) *б)*


y, cm
5
0
-5
-5
0
5
10
15
x, cm

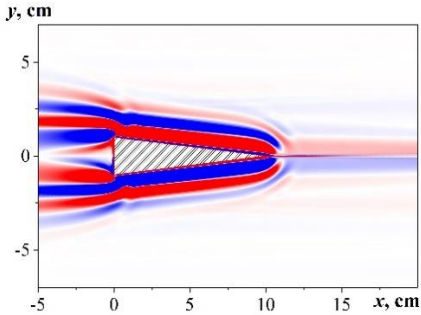
в)


y, cm
5
0
-5
-5
0
5
10
15
x, cm

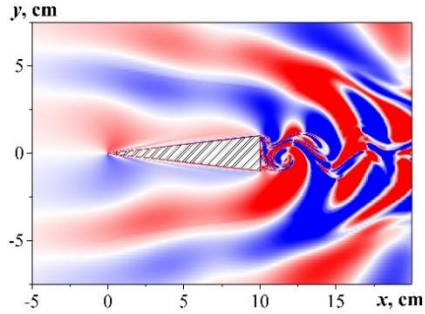
г)


y, cm
5
0
-5
-5
0
5
10
15
x, cm

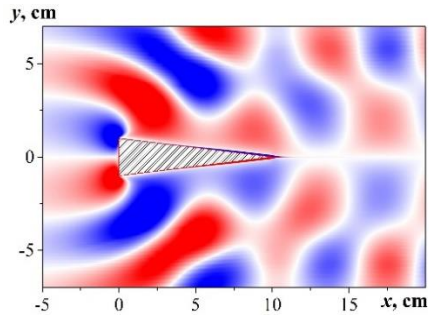
д)


y, cm
5
0
-5
-5
0
5
10
15
x, cm

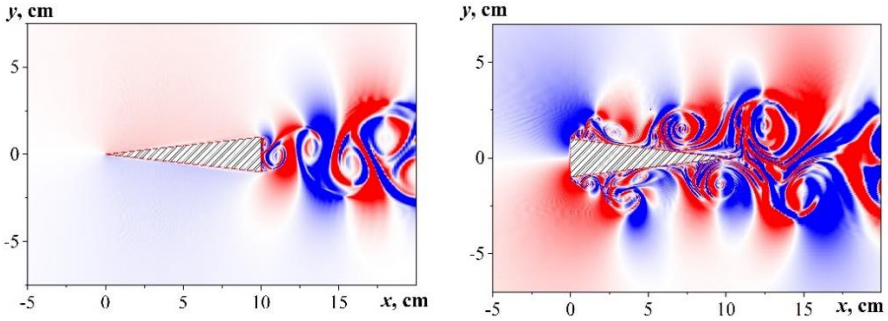
е)


y, cm
5
0
-5
-5
0
5
10
15
x, cm

ж)


y, cm
5
0
-5
-5
0
5
10
15
x, cm

з)



и) *к)*

Рис. 5. Горизонтальная компонента градиента возмущения солености ∂s/∂x при обтекании клина со скоростью U = 10^{-5} м/с (а, б), U = 10^{-4} м/с (в, г), U = 10^{-3} м/с (д, е), U = 10^{-2} м/с (ж, з), U = 10^{-1} м/с (и, к). (L = 10 см, h = 2 см, T_b = 6.28 с, красный цвет – положительные значения, синий – отрицательные)

Fig. 5. Horizontal component of salinity gradient perturbations ∂s/∂x with increase of the external flow velocity: U = 10^{-5} m/s (а, б), U = 10^{-4} m/s (в, г), U = 10^{-3} m/s (д, е), U = 10^{-2} m/s (ж, з), U = 10^{-1} m/s (и, к). (L = 10cm, h = 2 cm, T_b = 6.28 s, red - positive values, blue – negative ones)

Принципиальное различие картины обтекания тела стратифицированной жидкостью от однородной проявляется вблизи экстремальных точек обтекаемого тела. Когда скорость внешнего потока сравнима по порядку величины с характерной скоростью диффузии индуцированных потоков $U_N^{\kappa_S}$, в течение длительного времени структура течения сохраняет элементы исходного поля (рис. 5, а, б). Возле острых углов проявляются системы периодических структур, иллюстрирующие внутренние волны. Источником внутренних волн служат краевые сингулярности, генерирующие интенсивное вертикальное вытеснение жидкости, что приводит к отклонению от изначального положения нейтральной плавучести и, как следствие, формированию периодических затухающих колебаний жидкости.

Увеличение скорости движения приводит к пропорциональному увеличению длины присоединенной внутренней волны в соответствии с формулой линейной теории $\lambda = UT_b$. Фазовые поверхности, разделяющие волновые возмущения противоположных знаков, загибаются в сторону движения пластины. При скоростях движения U > 10^{-2} м/с в следе за клином формируются вихревые возмущения (рис.5, ж-к). На границах раздела внутренних присоединенных волн и вихревого следа формируются высокоградиентные области. Рассчитанные картины обтекания клина по своей структуре согласуются с результатами экспериментальных и численных исследований обтекания тел с другими геометрическими формами потоком стратифицированной жидкости [1].

Выявлены существенные различия внутренней структуры полей различных физических величин, которые говорят о сложности изучаемого явления и высокой размерности пространства поставленной задачи. Структурные элементы полей физических величин отличаются друг от друга размерами и законами изменения по пространству и по времени. Причем градиенты физических величин, формирующиеся неоднородностями молекулярного потока стратифицирующей примеси, ведут себя немонотонно вблизи обтекаемого тела и достигают высоких численных значений.

6. Заключение

- Вычисления, проведенные в открытом пакете OpenFOAM показали возможность расчета многомасштабных структурированных течений в широком диапазоне параметров задачи.
- Проанализированы нестационарные стратифицированные течения около клина. Определены механизмы формирования вихрей в областях больших градиентов плотности в окрестности кромок препятствия.
- Результаты расчетов качественно согласуются с данными лабораторных опытов.

Работа выполнена с использованием сервисов платформы UniHUB – http://www.unihub.ru

Список литературы

[1]. Чашечкин Ю. Д. Структура и динамика природных течений: теоретическое и лабораторное моделирование. В сб.: Актуальные проблемы механики. 50 лет Институту проблем механики им. А. Ю. Ишлинского РАН. Под ред. Ф. Л. Черноусько. М.: Наука, 2015. С. 63-78. ISBN: 978-5-02-039181-9

[2]. Чашечкин Ю.Д., Бардаков Р.Н., Загуменный Я.В. Расчет и визуализация тонкой структуры полей двумерных присоединенных внутренних волн, Морской гидрофизический журнал, № 6. 2010 г. стр. 3-15.

[3]. Брузе К., Доксуа Т., Ерманюк Е., Жубо С., Крапошин М., Сибгатуллин И. Прямое численное моделирование аттракторов внутренних волн стратифицированной жидкости в трапециедальной области с колеблющейся вертикальной стенкой. Труды ИСП РАН, том 26, вып. 5, 2014 г., с. 117-141. DOI: 10.15514/ISPRAS-2014-26(5)-6

[4]. Phillips O. M. On flows induced by diffusion in a stably stratified fluid. Deep-Sea Res., volume 17, 1970. P. 435–443.

[5]. Shapiro A., Fedorovich E. A boundary-layer scaling for turbulent katabatic flow. Boundary-layer meteorology, volume 153, Issue 1, 2014. P. 1-17. DOI: 10.1007/s10546-014-9933-3.

[6]. Mercier M. J., Ardekani F. M., Allshouse M. R., Doyle B., Peacock T. Self-propulsion of immersed object via natural convection. Physical review letters, volume 112, 2014. P. 204501(5). DOI: https://doi.org/10.1103/PhysRevLett.112.204501

[7]. Dimitrieva N. F., Zagumennyi Ia. V. Diffusion-driven flows on a wedge-shaped obstacle. Physica Scripta, vol. 91, no 8, 2016, P. 084002. DOI: 10.1088/0031-8949/91/8/084002

[8]. Димитриева Н. Ф., Чашечкин Ю. Д. Структура индуцированных диффузией течений на клине с искривленными гранями // Морской гидрофизический журнал, № 3, 2016 г. стр. 77–86. Доступно по ссылке: http://мгфж.рф/index.php/repository?id=120

[9]. Байдулов В. Г., Чашечкин Ю. Д. Сравнительный анализ симметрий моделей механики неоднородных жидкостей. Доклады Академии Наук, том 444, № 1, 2012 г. стр. 38–41. DOI: 10.1134/S1028335812050011

[10]. Димитриева Н. Ф., Чашечкин Ю. Д. Высокопроизводительное численное моделирование стратифицированных течений около клина в OpenFOAM. Труды ИСП РАН, том 28, вып. 1, 2016 г., с. 207-220. DOI: 10.15514/ISPRAS-2016-28(1)-12

[11]. Евстегнеев Н. М. Конечно-объемная TVD схема для решения 2D эволюционных уравнений мелкой воды. Вычислительные методы и программирование, том 7, № 1, 2006 г., с. 108-112. Доступно по ссылке: http://num-meth.srcc.msu.ru/zhurnal/tom_2006/pdf/v7r113.pdf

[12]. Димитриева Н.Ф., Чашечкин Ю.Д. Численное моделирование динамики и структуры индуцированного диффузией течения на клине. Вычислительная механика сплошных сред, том 8, № 1, 2015 г. стр. 102– 110. doi: 10.7242/1999-6691/2015.8.1.9

The numerical solution of the problem of stratified fluid flow around a wedge using OpenFOAM

N.F. Dimitrieva < dimitrieva@list.ru >
Institute of Hydromechanics of the National Academy of Sciences of Ukraine, 8/4 Zheliabova Street, Kiev, 03680, Ukraine

Abstract. Based on the open source software 2D numerical simulations of incompressible stratified fluids flows have been performed. They are characterized by a wide range of values of internal scales that are not in a homogeneous liquid. Mathematical model is based on the fundamental set of differential equations of inhomogeneous multicomponent fluid mechanics. The problem is solved using the finite volume method in an open source package OpenFOAM. The method allows analyzing in a single formulation the dynamics and fine structure of flow patterns past obstacles in a wide range of flow parameters. A particular attention is focused at construction of a high quality computational grid which satisfies basic requirements for resolution of all the microscales of the problem in high-gradient regions of the flow. The calculations were performed in parallel regime on computational facilities of the web-laboratory UniHUB (www.unihub.ru). The same system of equations and a general numerical algorithm were used for the whole range of the parameters under consideration. The calculation results are in a qualitative agreement with the data from laboratory experiments. Transient flow patterns past obstacles are analyzed, and physical mechanisms are determined, which are responsible for formation of vortices in regions with high density gradients near the edges of an obstacle. For all the velocities of the body motion, the flow

field is characterized by a complicated internal structure. In the flow pattern around motionless body dissipative gravity waves are manifested at the edges of the strip. Around the slowly moving body a group of attached waves, are formed in opposite phases at the edges of the wedge. Then, the main flow components become vortices, which are formed around the edge of the wedge and manifested downstream in the wake. With further increase in velocity of the body motion, the flow pattern becomes more non-stationary.

Keywords: numerical simulation; open source computational packages; stratified flows.

DOI: 10.15514/ISPRAS-2017-29(1)-1

For citation: Dimitrieva N.F. The numerical solution of the problem of stratified fluid flow around a wedge using OpenFOAM. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 1, 2017, pp. 7-20 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-1

References

[1]. Chashechkin Yu. D. Structure and dynamics of natural flows: theoretical and laboratory simulation. In Proc.: Aktual'nye problemy mekhaniki. 50 let Institutu problem mekhaniki im. A. Ju. Ishlinskogo RAN [Actual Problems in Mechanics. 50 Years of the A. Yu. Ishlinskiy Institute for Problems in Mechanics of the RAS . Ed. F. L. Chernousko. Moskow: Nauka [Science], 2015, pp. 63-78. ISBN: 978-5-02-039181-9

[2]. Chashechkin Yu. D., Bardakov R. N., Zagumennyi Ia. V. Calculation and visualization of the fine structure of fields of two-dimensional attached internal waves. Physical Oceanography, 2010, no. 6, pp. 3-15. (in Russian)

[3]. Brouzet C., Dauxois T., Ermanyuk E., Joubaud S., Kraposhin M., Sibgatullin I. Direct numerical simulation of internal gravity wave attractor in trapezoidal domain with oscillating vertical wall. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 117-141 (in Russian). DOI: 10.15514/ISPRAS-2014-26(5)-6

[4]. Phillips O. M. On flows induced by diffusion in a stably stratified fluid. Deep-Sea Res., 1970, vol. 17, pp. 435–443.

[5]. Shapiro A., Fedorovich E. A boundary-layer scaling for turbulent katabatic flow. Boundary-layer meteorology, 2014, vol. 153, issue 1, pp. 1-17. DOI: 10.1007/s10546-014-9933-3.

[6]. Mercier M. J., Ardekani F. M., Allshouse M. R., Doyle B., Peacock T. Self-propulsion of immersed object via natural convection. Physical Review Letters, 2014, vol. 112, pp. 204501(5). DOI: https://doi.org/10.1103/PhysRevLett.112.204501

[7]. Dimitrieva N. F., Zagumennyi Ia. V. Diffusion-driven flows on a wedge-shaped obstacle. Physica Scripta, vol. 91, no 8, 2016, pp. 084002. DOI: 10.1088/0031-8949/91/8/084002

[8]. Dimitrieva N.F., Chashechkin Yu.D. The structure of induced diffusion flows on a wedge with curved edges. Physical Oceanography, 2016, no 3, pp. 77–86. Available at: http://мгфж.рф/index.php/repository?id=120

[9]. Baydulov V.G., Chashechkin Yu.D. Comparative analysis of symmetries for the models of mechanics of nonuniform fluids. Doklady Physics, 2012, vol. 57, no. 5, pp. 192–196. doi: 10.1134/S1028335812050011

[10]. Dimitrieva N.F., Chashechkin Yu.D. High-performance numerical simulation of stratified flows around a wedge in OpenFOAM. Trudy ISP RAN/Proc. ISP RAS, 2016, vol. 28, issue 1, pp. 207-220 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-12

[11]. Evstigneev N.M. A finite-volume TVD Riemann solver for the 2D shallow water equations. Vychislitel'nye Metody i Programmirovanie [Numerical Methods and Programming], 2006, vol. 7, no 1, pp. 108-112. Available at: http://num-meth.srcc.msu.ru/zhurnal/tom_2006/pdf/v7r113.pdf

[12]. Dimitrieva N.F., Chashechkin Yu.D. Numerical simulation of the dynamics and structure of a diffusion-driven flow on a wedge. Vychislitel'naya mekhanika sploshnykh sred [Computational continuum mechanics], 2015, vol. 8, no. 1, pp. 102– 110. (in Russian) doi: 10.7242/1999-6691/2015.8.1.9

Numerical study of characteristic modes and frequencies of flow in high-speed compressors

[1] *M. Kalugin <m.kalugin@ispras.ru>*
[2] *I. Evdokimov <evdokimov.ilya@gmail.com>*
[1] *Institute for System Programming of RAS,*
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
[2] *AVIATICA Ltd, 14, Kolodezny per., Moscow, 107076, Russia*

Abstract. In this paper, we present the newly developed open-source density-based solver pisoCentralDyMFoam and investigate an application of the Proper Orthogonal Decomposition (POD) algorithms for industrial turbomachinery-related problems. POD is implemented in Apache Spark framework for distributed data processing. This solver is based on hybrid Kurganov-Tadmore/PISO scheme. The research was conducted for geometry close to its real prototype with known resonance frequencies. The solver was previously validated on simple industrial case ERCOFTAC centrifugal pump. The POD coefficient matrices were constructed using data set of the snapshots representing each saved time-step of the whole Navier-Stokes numerical model. Several hundred consecutive snapshots of the static pressure field on the impeller wheel surface as well as the velocity field were used for computation of the POD modes. The eigenvalues determined by POD method corresponds to the kinetic energy contained in each mode. The spatial coefficients represents contribution of each elementary volume to the whole mode and helps to locate the region having influence on the mean flow at specific frequencies after Fast-Fourier-Transform (FFT) applied to time-dependent coefficients of the decomposition. The POD modes were sorted by kinetic energy and the zeroth mode was most energetic representing mean flow with relatively small amplitudes. The described concept was extensively validated using computationally cheap 2D-case and then extended to the high-speed centrifugal pump of the small-scale turbojet. It was found that the third mode of the flow has first peak at 12970 Hz right between 2 construction resonance points at 12000 Hz and 13700 Hz. The third and fourth modes represent pressure fluctuations in the wakes region in the diffuser behind vanes. The demonstrated approach allows engineers to analyze flow dynamics more effectively compare to traditional FFT at certain points or cross-section. In addition, it can be useful for data compression and Reduced-Order Model development.

Keywords: proper orthogonal decomposition; centrifugal pump;, fluid dynamics

DOI: 10.15514/ISPRAS-2017-29(1)-2

For citation: M. Kalugin, I. Evdokimov. Numerical study of characteristic modes and frequencies of flow in high-speed compressors. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 1, 2017, pp. 21-38. DOI: 10.15514/ISPRAS-2017-29(1)-2

1. Introduction

In the turbomachinery-related field, Epureanu et al. [1] elaborated theoretical aspects and application of the POD models itself. The authors investigated the influence of the amount of POD modes on resulting aerodynamic characteristics if the airfoil cascade. They noticed that subsonic and transonic simulations require different degrees of freedom of the POD model to predict correctly flow dynamics for all interblade phase angles.

The early work of the ONERA research group [2] was dedicated to extraction of the most energetic modes of the flow in the centrifugal pump to catch rotor-stator interaction using most reliable and robust techniques. Rochuon et al. [2] investigated both Fourier-transform and POD to obtain information about flowfield. They found that POD allows getting first most energetic modes representing 98% of the flow kinetic energy by nine times less harmonics. Clark et al. [3] demonstrated the application of POD-analysis for turbomachinery-related dynamics problems very well in their work about reduced-order models designed to diagnose flow-induced Nonsynchronous vibration (NSV) in turbomachines. The paper contains well-described techniques dedicated to obtain POD modes and visualize POD's harmonics as flow field in the blade channel. Also the solution predicted by ROM correlates very well to CFD solution in “blind” test of the two-dimensional Navier-Stokes model of the C1 single stage axial compressor. The authors concluded that POD methods could be successfully used to reduce computational cost of traditional CFD models that can predict NSV correctly.

Danaila et al. [4] did POD of the centrifugal pump CFD-solution to investigate rotor-stator interaction and used method as concurrent to traditional Fourier—transformation to obtain frequency characteristics of the flow-field. To sum up, they reproduced ONERA results obtained by Rochuon et al. [2]. Danaila's group explained numerical effects arising due to interfaces in the fluid domain and pointed reliability of the POD for saving information about unsteady complex flows in the numerical models. Fossati et al. [5] developed the reduced-order model of axial turbomachine. They showed the same advantages of POD as Danaila's group.

The numerous publications dedicated to POD analysis shows that this method has various applications in industry and applied science despite its statistical nature. In this paper, the certain problems of relation between physically reasonable simulations and statistical analysis are also investigated. From the engineering point of view, the POD method could be very reliable and useful as a replacement to FFT at certain points or planes. Similar to Rochuon's [2] work we also use Fourier-transform results to prove POD robustness and reliability, but they were obtained by other researchers for the same case.

1.1. Research objectives

The aim of this research effort consists of three subtasks. These are:

- test POD as potential technique for simplification of the large amount of CFD data by relatively small set of eigenvalues;
- compare POD-analysis of the test pump with FFT-analysis in case of amount of harmonics need for pressure-signal reconstruction and obtained information;
- determine modes that carry information about dangerous regions of the flow in the sense of high probability of resonance between structure and flow.

The first one deals specifically with POD algorithm development and validation, which was carried using various tests. The second one allows us to prove POD advantages relate to Fourier-transform techniques and prove the concept using sophisticated industrial-related problems, taking a simple pump model as an instance. The complexity and specialization of the developed techniques increases significantly at the last stage when all developed software stack is applied to industrial case. Concerning to the third objective we use FFT transformation of the time-dependent POD coefficients to identify frequencies of the modes and then compare them to the resonance frequencies presented in the documentation for the chosen compressor.

2. Numerical approach

2.1. The solver

The pisoCentralDyMFoam solver is an extension of the newly developed pisoCentralFoam [6] with an option of the mesh motion handling in the numerical model. The OpenFOAM dynamicFvMesh abstract class generalizes methods to handle mesh transformation at the level of points and cells. In our work we only use mesh revolution around one axis of rotation but the mesh motion could be more complex. In the connection with turbomachinery problems such dynamic mesh motion solvers usually applicable to complex transient flows. The possible applications could include rotor precession modelling and tip clearance variation due to thermo-mechanical expansion very similar to Amirante's et al. work [7]. The dynamic mesh modelling deals with the sliding interface approach which is opposite to frozen rotor when stationary problem is simulated.

The main advantage of the hybrid Kurganov-Tadmor solver is in the possibility of simulation compressible flows in wide range of Mach numbers ($0<M<6$ [6]). The numerous amount of turbomachines equipped with multistage compressors have different operating conditions varying from relatively low velocities at the inlet to transonic and supersonic at the outlet of the last compressor stage. Thus, using one

universal open-source solver makes typical engineering workflow even simpler and closer to commercial computational codes.

2.2. The test case

The ERCOFTAC centrifugal pump is a basic well-described test case for CFD-code validation. Petit et al. [8] carried out successful testing of their codes specially developed for turbomachinery applications. Combès [9] carried out the initial publication of the described test case at a Turbomachinery Flow Prediction ERCOFTAC Workshop in 1999. The experimental results were obtained at the test rig built by M.Ubaldi et al. [10].

We used two-dimensional version of the initial case due to its simplicity and possibility to include as basic solver validation test. The geometry of the model is identical to description given in [9, 10] and all dimensions are shown on the Figure 1. The test-case mesh was generated using ICEM-HEXA, consists of 94 000 cells, with average Y+ value about 35. For sliding mesh interface, we used Arbitrary Mesh Interface (AMI). Also we used slightly different boundary conditions: the average static pressure at the inlet and the mass flow rate at the outlet. Petit and Nilsson [11] used constant velocity at the inlet, which is equivalent to mass flow, and constant static pressure at the outlet.

The same as in [8, 11], the tip clearance which is equal to 1% (0.4 mm) of the blade span was not modelled in our simplified 2D verification simulation. Also we used slightly different boundary conditions: the average static pressure at the inlet and the mass flow at the outlet. Petit and Nilsson [11] used constant velocity at the inlet, which is equivalent to mass flow, and constant static pressure at the outlet.

Based on the experimental data [10] we supposed that the inlet turbulent intensity is assumed to be 5% with a viscosity ratio of 10. The static pressure at the inlet is constant and equal to 101325 Pa (not given explicitly by [10]). Subsequently, the outlet boundary conditions for all variables except velocity are given a zero gradient condition, but for velocity are equal to flow rate. Mass flow was calculated using M.Ubaldi [10] pump performance data by the simple formula:

$$Q = \frac{U_2 \pi D_2}{4} = 0.292\ m^3 / s$$

The simulations were conducted using variable time-step adjusted by the maximum Courant number (equal to 0.5). The typical value for the time-step during simulation was about 5e-6 s. After several rotations the final residuals was equal to values: RMS U_x = 1.56629e-11, RMS U_y = 1.36025e-11, RMS p = 7.74527e-10, Continuity errors: sum local = 4.22966e-12, global = 1.10594e-12, cumulative = -3.41295e-09, RMS ω = 7.6337e-12, RMS k = 7.98248e-12.

2.3. Validation results

The used solver with k – ω SST turbulence model demonstrates good corresponding to both experimental and simulation results obtained by Petit and Nilsson using k −

ε turbulence model. The calculated velocity profiles are presented in the Figures 1, 2. The difference between solvers could be explained by different types of solutions (2D in our and 3D in [11]) including numerical schemes effects and different turbulence treatment.

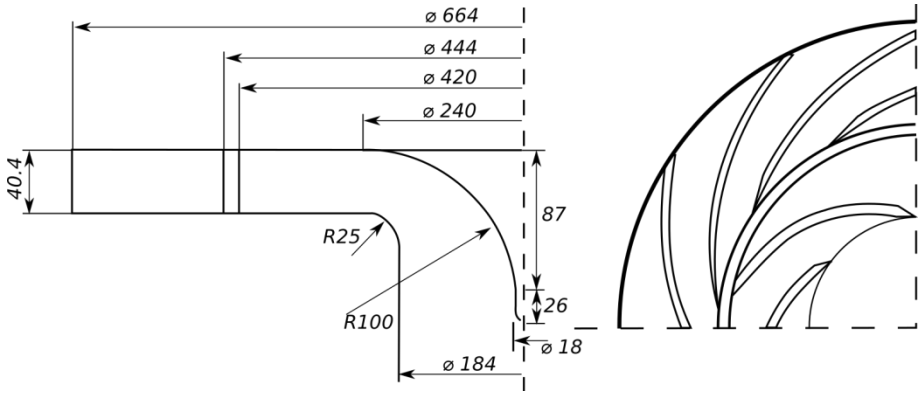
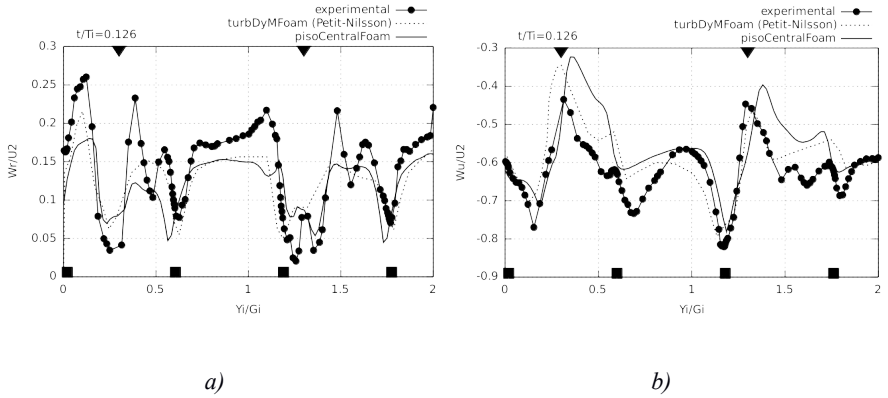


Fig. 1. The ERCOFTAC Centrifugal Pump geometry

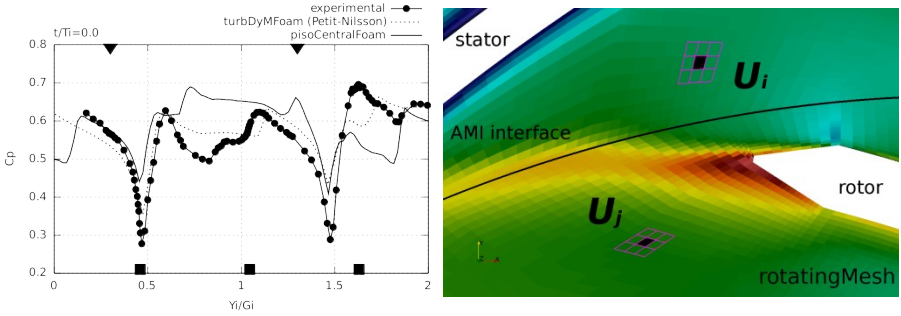


a) *b)*

Fig. 2. The velocity profiles in the radial gap (R/R2 = 1.02): a) radial, b) tangential

Simulated static pressure coefficient distribution is less accurate in comparison with Petit and Nilsson [11] results in the Figure 3. The velocity and static pressure coefficient distribution graphs show that computational model adequately predicts wakes position. However, the Cp downfalls are underestimated. Petit and Nilsson investigated discrepancies between OpenFOAM simulation and experimental data more elaborately and concluded that major differences are the result of the complex motion of the flow and unresolved secondary flows due to tip clearance influence in

their three-dimensional case [11]. In spite of the visible differences, we concluded that used hybrid Kurganov-Tadmor solver is validated successfully and could be used effectively for turbomachinery applications.



a) *b)*

Fig. 3. a) Static pressure coefficient C_p in the radial gap (R/R2 = 1.02); b) the scheme illustrating selection of the elementary volumes in the different part of computational domain

3. Application of the Proper Orthogonal Decomposition for turbomachinery cases

The vast majority of papers dealing with POD applications in turbomachinery (for instance [2], [3], [12]) uses traditional derivation of the basic equations as it was initially described by Lumley and then developed by him with co-authors in latest work [13]. The ground work [13] also discuss in detail the physical meaning of the evaluated modes of the experimentally observed or simulated flow quantities. Using POD we are seeking for approximation:

$$u(x,t) \approx \sum_{i=1}^{k} a_i(t)\varphi_i(x), \qquad (1)$$

where *u(x,t)* usually implies function (i.e. velocity) in certain spatial location, $\varphi_i(x)$ represents spatial modes and $a_i(t)$ its time-dependent coefficients. Since POD is purely statistical method and do not account dimensionality (and physics of the phenomena), using it for problems with mesh motion or geometrical deformations is quite problematic and needs to be carefully considered. Present well-described works about POD applications in the turbomachinery do not include elaboration of the rotating regions (Fjällman J. et al. [12]), process only section in the stator part (ONERA work [2]) or dealing with “frozen rotor” approach ([3], [4], [14]). For non-stationary, full-dimensional “sliding mesh” approach, the proper

application of the POD stays questionable. Closely connected research works about deforming meshes use two different methods to handle mesh moving or deformation. Anttonen et al. [15] introduced to use computational index space and Erwan et al. [16], Troshin et al. [17] interpolated time-dependent computational fluid domain on the separate static mesh used to evaluate POD modes. In our work, we evaluate flow modes in the indexed space (see Figure 3b) of the elementary fluid volumes due to simplicity of the implementation in the CFD code and reliability when cylindrical coordinate system, which is typical for turbomachinery, is used. Thus, the equation (1) for the velocity component could be rewritten as:

$$u(x,t) \approx \sum_{i=1}^{k} a_i(t)\varphi_i(n), \tag{2}$$

where n – volume number in the CFD mesh hierarchy. Consequently in the each moment of time (snapshot), $a_i(t_N)$ represents the contribution of the whole mode the in elementary volume, $\varphi_i(n)$ – time-invariant spatial contribution to the kinetic energy of the incompressible flow. According to [13] for compressible flow we should introduce non-dimensionalized sum of the flow parameters. Across the whole snapshot collection the relative position of the different volumes in the index list does not change (mesh transformation handles only on the point level which is the lowest in the hierarchy), so main limitation (points does not change its relative position in index space) of the POD is fulfilled [13]. It needs to be pointed that POD - algorithm itself was tested on various simple cases including vibrating string with various initial and boundary conditions and Korteweg–de Vries equation using known soliton solution as reference. In the present work, we validate and demonstrate POD capabilities using only Ubaldi [10] test pump results.

3.1 POD analysis of the ERCOFTAC compressor

Relatively low computational cost of the 2D test case is good advantage for the developing of various algorithms to meet the technology needs. That is the main reason why Ubaldi [10] centrifugal pump was chosen as “proof-of-concept” case for demonstrations of the POD applications in the turbomachinery field. The studying of the flow phenomena using simple 2D representation of the ERCOFTAC compressor have an additional property such as simplicity of the visualization and analysis. The calculated lowest mode 0 for radial and tangential velocities is shown in Figure 4. Higher 6th mode is shown in the Figure 5.

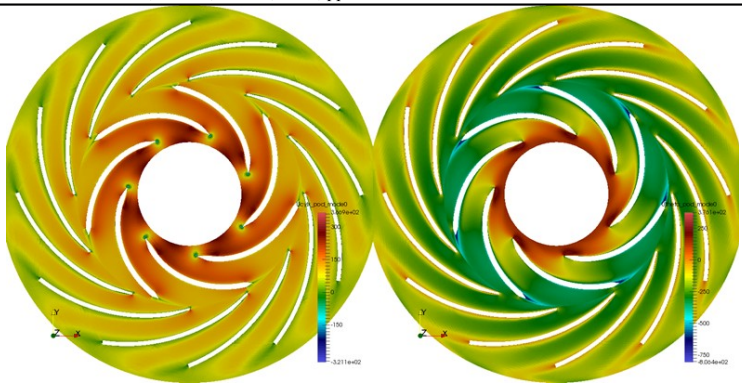


Fig. 4. Mode 0 for radial (left) and tangential (right) components of the flow velocity in the ERCOFTAC centrifugal pump (cylindrical coordinate system)

The characteristic of the spatial distribution of the spatial coefficients across the domain on the mode 0 (Figure 4) is very different compare to 6th mode where we can observe separation on the static and rotating regions. The wakes behind the impeller blades are almost uniform and independent from the relative position of the stator and rotor. We can say that mode 0 represents mean flow distributed across all computational domain and various turbulent effects in the wakes regions (for example, wake breaks when impeller blade passes near stator one) are captured by higher modes. The comparison between modes of the radial and tangential components also tells that tangential mode has larger absolute contribution to wakes region than radial (Figure 5). All these facts say that POD as statistical method could carry physically reasonable information about flow in compressor if we chose proper coordinate basis (using orthogonal coordinate for mode decomposition is potentially erroneous or needs much more modes and makes further analysis very hard).

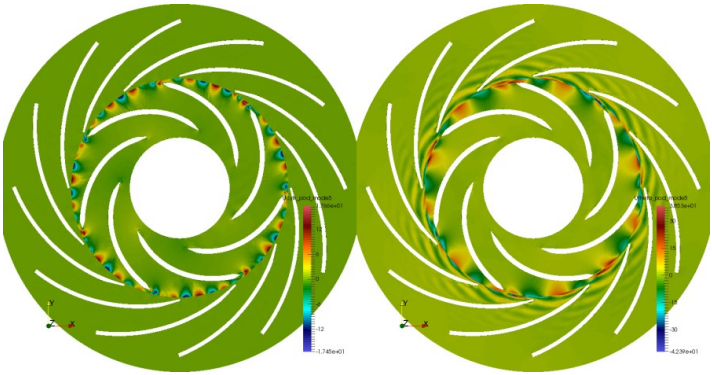


Figure 5 Mode 5 for radial (left) and tangential (right) components of the flow velocity in the ERCOFTAC centrifugal pump (cylindrical coordinate system)

3.2. POD versus FFT

In the original work about flow analysis, in the chosen test pump [11] the authors done FFT analysis using two probes at specific points and 20 frequencies for signal reconstruction. It can be shown that POD could catch all detected peak frequencies using fewer modes than frequencies in FFT, which has correspondence to the results of the ONERA group ([2]). In addition to the demonstrated visualization capabilities, the FFT analysis of the modal time-coefficients allows engineer to catch characteristic frequencies of the flow and reconstruct certain value of the quantity in the specified location. In the Figure 6 the value versus time, graphs were plotted to demonstrate time-dependent properties of the found POD modes. It can be seen, that mode 0 has constant value while higher modes have not. However, it is only due to the scale of the signal versus time graph: the FFT analysis of the mode 0 time coefficient shows that mode 0 has small peaks in the specific frequencies but with different amplitude from the other higher modes (Figure 7).

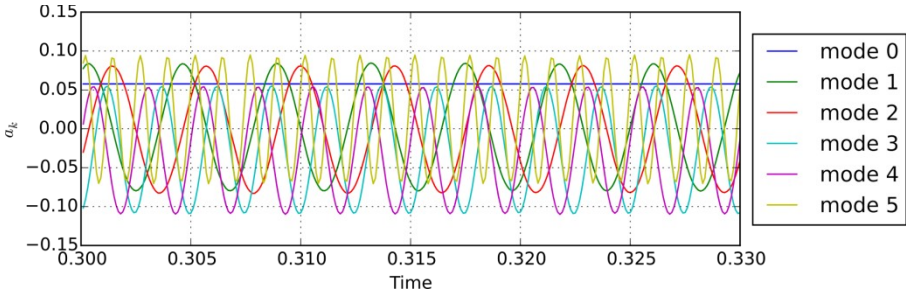


Fig. 6. The time coefficient for the first six modes of the radial flow component, obtained during one revolution of the impeller

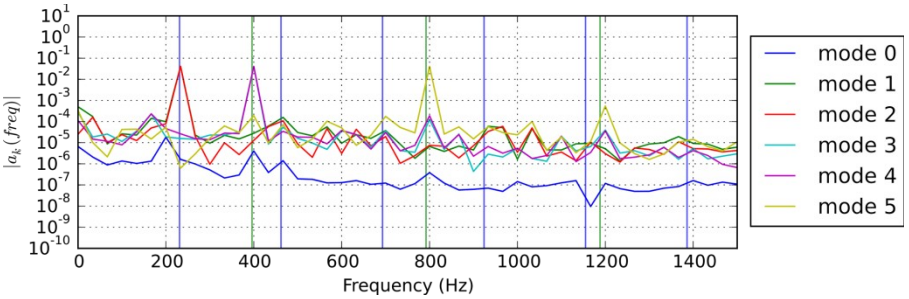


Fig. 7. The FFT-transform of the time coefficients of the first six modes of the radial flow component. The colored vertical lines are the impeller (blue), diffuser (green) frequencies, and its harmonics (y-axis is in the log scale)

For the given case the rotor frequency is $f_r = 2000/60 = 33$ *Hz*, impeller frequency equal to $f_i = f_r * z_i = 232$ *Hz* and diffuser frequency has value $f_d = f_r * z_d = 400$ *Hz*. Both radial and tangential components have similar neighboring modes. The only difference of such modes could be in phase change (modes 3 and 4 in Figures 6, 7), on the phase-space diagram, such coefficients form concentric circles and the distribution of spatial mode coefficients is similar except little relative rotation. Spatial concentration of the non-zero modal coefficients for mode 3-4 is located in the rotor-stator clearance (very similar to mode 5 in Figure 5). These modes catch frequencies 400, 800, 1200 Hz, which is the frequency of the diffuser blades. Both radial and tangential components have these frequencies (Figure 7).There is no significant difference between velocity and pressure POD (see Figure 8), if we have an intention to describe frequencies of the flow. The pressure scalar field is much simpler to decompose and analyze. Thus for complex 3-dimensional flows we can evaluate only pressure field modes if we do not need to capture and store separation and wake – effects in the velocity vector field.

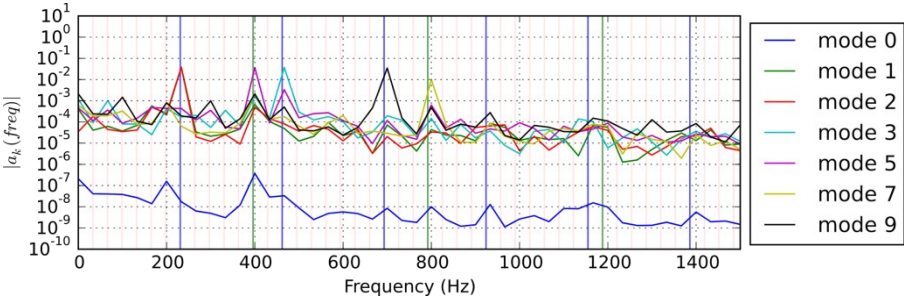
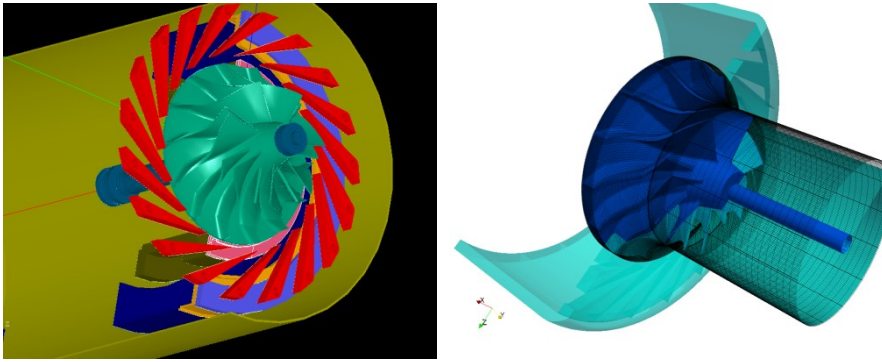


Fig. 8. The FFT-transform of the time coefficients of the first ten odd modes of the pressure field. The colored vertical lines are the impeller (blue), diffuser (green) frequencies, and its harmonics. The light pink vertical lines are the rotor harmonics

4. POD analysis of the industrial centrifugal pump

The present research work was conducted using model of the small-scale jet engine compressor developed in the early 90's by Andreas Funke and Klaus Wittig (Figure 9). The whole engine is also used as test case for open-source finite-element software CalculiX [18]. The aluminium impeller of the selected compressor is designed for operation with maximum circumferential tip velocity 600 m/s, which is achieved for the diameter of 44 mm at 130000 revolutions per minute.



a) *b)*

Fig. 9. a) The compressor geometry [18]; b) simulation domain (1.5e6 cells mesh size)

4.1. Case setup

The operation conditions were selected using documentation [19] to converge with original design and checking calculations. These are:

- Working fluid – air;
- Static temperature at the inlet 293.15 ºK (20.0 ºC);
- Static pressure at the inlet 101325 Pa;
- Air density 1.225 kg/m^3;
- Air mass flow 0.42 kg/s;
- Rotating frequency 108000 rev/min.;
- Gas constant 287.1 kJ/kg/K.

Figure 9b shows the simulation domain including static and rotating regions of the mesh.

For the given case, initial and boundary conditions were set up as described:

- undisturbed fluid in the domain at the starting point;
- mass flow and static temperature at the inlet;
- the fixed static pressure at the outlet 400 kPa and zero gradient for all others variables;

- slip condition for velocity at the walls, zero gradient for pressure, adiabatic for temperature and wall-functions for turbulent parameters (so LES actually transforms to DES);
- various models for turbulence treatment: k-ω SST and LES with Smagorinsky sub-scale model and cube-root filter.

4.2. CFD simulation results

Due to limited space we would not show various figures with simulated flow quantities and elaborate only integral results to control quality of the whole centrifugal pump model. Mesh convergence tests were performed to prove efficiency of the new hybrid density-based central difference solver in relation with standard pressure-based open-source *sonicDyMFoam* solver. Various meshes were generated using the same geometry and block structure as it was shown in Figure 9b except that the edge discretization variated from case to case.

For the hybrid solver equipped with 1.5 mill. mesh the calculated pressure ratio was 3.17 at N=108 000 rev/min which is close to the documentation value 3.30 [18] for the given mass flow 4,2 kg/s at N=110000 rev/min. In such conditions the error was -4%. The evaluated shaft power was 73.4 kW.

The convergence dynamic for the pressure-based solver is shown on the Table 1. We conclude that it has less performance than density-based central-difference solver using only 1.5 mil mesh and the last is significantly closer to the experimental results.

Table 1. Mesh convergence test results (modified pressure-based sonicDyMFoam)

Mesh/Cells	Pressure ratio	Power,kW
Coarse/ 1.5 mil.	3,78	55
Refined / 3.0 mil.	3,3	60
Fine / 26.0 mil.	3,2	62

4.3. POD modes and flow dynamics

For the investigated centrifugal pump the simulation results are presented excluding repeating neighboring modes. The FFT-transform of the time coefficients of the odd modes from 0th to 9th is presented in the Figure 10. The visualization of the selected odd modes is presented in the Figures 11-13.

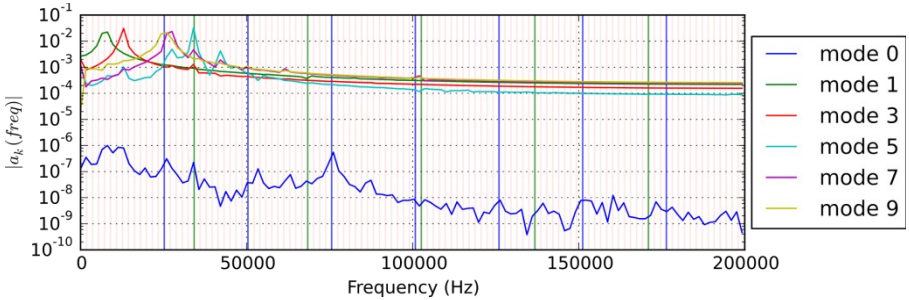
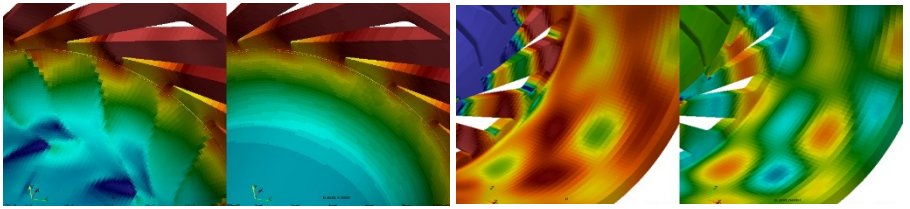


Fig. 10. The FFT-transform results of the time coefficient of the first odd modes from 0th to ninth. The colored vertical lines are the impeller (blue), diffuser (green) frequencies, and its harmonics. The light pink vertical lines are the rotor harmonics

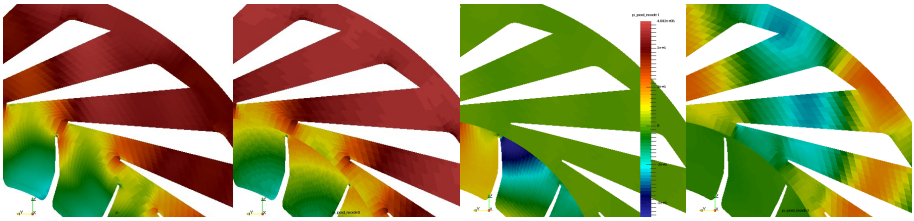


a) *b)*

Fig. 11. a)The original pressure field (left) and mode 0 (right) of the high-speed centrifugal pump impeller region; b) The original pressure field (left) and mode 3 (right) of the high-speed centrifugal pump diffuser region

The same as in simple two-dimensional validation case, three-dimensional compressor has main 0th mode with significantly lower amplitudes of the time-coefficients. Figure 12 demonstrates distribution of the spatial coefficients across the domain in the impeller and diffuser parts consequently. For the main 0th mode at the Figure 12a we can see that the coefficients are quite uniform and do not depend on what part of the domain, rotating or stating, they belong. Specific wake pressure fluctuations behind vanes were captured by third mode (and its neighboring mode 4th, which is not showed here). Combining it with the FFT results of the third mode (Figure 10) we can clearly say that characteristic frequency (12970 Hz peak) of this wake could potentially coincide with resonance frequencies 12000 Hz and 13700 Hz ([19]) in case of changing operating conditions.

The cross-section data (Figures 12, 13) obtained at approximately half of the channel height in the outlet of the impeller demonstrates the same results as three-dimensional data in the Figure 11. The distribution of the spatial coefficients in the cross-section similar to pressure distribution at the 0th mode and various non-stationary effects are localized by higher modes. All these effects are concentrated in the clearance regions and depend of the time-coefficient amplitude and spatial coefficient contribute more or less at certain moments of time (Figures 12b, 13).



a) *b)*

Fig. 12. a) The original pressure field (left) and mode 0 (right) of the high-speed centrifugal pump in the diffuser cross-section; b) Pressure field mode 1 (left) and mode 3 (right) of the high-speed centrifugal pump in the diffuser cross-section

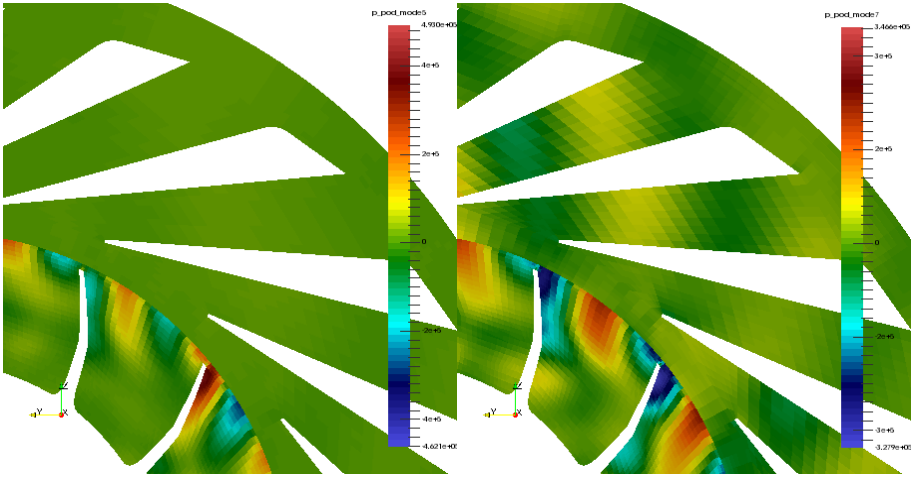


Fig. 13. Pressure field mode 5 (left) and mode 7 (right) of the high-speed centrifugal pump in the diffuser cross-section

5. Conclusion

The Proper Orthogonal Decomposition could be reliable and effective tool for analysis of the transient properties of the flow. Evaluated modes help to locate specific regions of the simulation domain, which are connected to its own frequencies. With the help of the FFT these frequencies could be easily identified.

The application of the method to the industrial case demonstrated successful identification of the flow instabilities with certain frequency near construction resonance. It is the main advantage of the POD relate to FFT analysis in which researcher have to specify the region of interest before processing it. Approaching simulation of the large-scale problems, POD can save more information for further analysis and be an effective basis for ROM.

References

[1]. Epureanu, BI; Dowell, EH; Hall, KC. A parametric analysis of reduced order models of potential flows in turbomachinery using proper orthogonal decomposition. Proceedings of the ASME Turbo Expo, vol. 1 (2001). doi: 10.1115/2001-GT-0434.

[2]. Rochuon, N., Trébinjac, I., Billonnet, G. An Extraction of the Dominant Rotor-Stator Interaction Modes by the Use of Proper Orthogonal Decomposition (POD). Journal of Thermal Science, Science Press, Vol.15, N°2, pp.109-114, June 2006.

[3]. Clark ST, Besem FM, Kielb RE, Thomas JP. Developing a Reduced-Order Model of Nonsynchronous Vibration in Turbomachinery Using Proper-Orthogonal Decomposition Methods. ASME. J. Eng. Gas Turbines Power. 2015;137(5). doi:10.1115/1.4028675.

[4]. Danaila, S., Niculescu, M. L. Unsteady effects at the interface between impeller-vaned diffuser in a low pressure centrifugal compressor. INCAS Bulletin5.1, 2013, p. 71-86.

[5]. Fossati,M., Nilamdeen, S., Habashi, W.G., Moustapha, H. Parametric Analysis of 3D Turbomachinery flows via Reduced Order Modelling. Conference Paper, 21st International Symposium on Air Breathing Engines, September 2013.

[6]. Kraposhin, M., Bovtrikova, A., Strijhak, S. Adaptation of Kurganov-Tadmor Numerical Scheme for Applying in Combination with the PISO Method in Numerical Simulation of Flows in a Wide Range of Mach Numbers. Procedia Computer Science, Vol. 66, 2015, Pages 43-52, ISSN 1877-0509, doi: 10.1016/j.procs.2015.11.007.

[7]. Amirante, D. and Hills, N.J. and Barnes, C.J. A moving mesh algorithm for aero-thermo-mechanical modelling in turbomachinery. International Journal for Numerical Methods in Fluids, Vol. 70 (2012), Number 9, p. 1118–1138, doi: 10.1002/fld.2734

[8]. Petit, O., Nilson, H., Page, M. and Beaudoin. The ERCOFTAC Centrifugal Pump OpenFOAM Case-Study. In Proceedings of the 3rd IAHR International Meeting of the Workgroup on Cavitation and Dynamic Problem in Hydraulic Machinery and Systems, Brno, Czech Republic.

[9]. Combès, J.F., Test Case U3: Centrifugal Pump with a Vaned Diffuser. ERCOFTAC Seminar and Workshop on Turbomachinery Flow Prediction VII, Aussois, jan 4-7, 1999.

[10]. Ubaldi M., Zunino P., Barigozzi G. and Cattanei A. An Experimental Investigation of Stator Induced Unsteadiness on Centrifugal Impeller Outflow. Journal of Turbomachinery, vol.118, 41-54, 1996.

[11]. Petit, P. and Nilsson, H. Numerical Investigations of Unsteady Flow in a Centrifugal Pump with a Vaned Diffuser, International Journal of Rotating Machinery, Volume 2013, 14 pages, doi: 10.1155/2013/961580

Численное исследование характеристических мод и частот течения в высокоскоростных компрессорах

[1] М.Д. Калугин <m.kalugin@ispras.ru>
[2] И.Е. Евдокимов <evdokimov.ilya@gmail.com>
[1] Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, 25
[2] ООО "АВИАТИКА", 107076, Россия, г. Москва, Колодезный переулок, 14

Аннотация. В статье описан разработанный решатель pisoCentralDyMFoam с открытым исходным кодом и исследуется применение метода главных

компонент в промышленных задачах моделирования турбокомпрессоров. Метод главных компонент реализован в Apache Spark, что позволяет использование распределенных вычислений. Разработанный решатель основан на гибридной схеме Курганова-Тадмора/PISO. Исследования проводились для геометрии компрессора, приближенной к реальному прототипу с известными резонансными частотами. Предварительно решатель был валидирован на примере центробежного компрессора ERCOFTAC. Коэффициенты матриц собственного ортогонального разложения получены из набора временных срезов, рассчитанных с помощью модели Навье-Стокса. Для вычисления мод собственного ортогонального разложения использовалось несколько сотен последовательных временных срезов полей скорости и давления, взятых на поверхности крыльчатки. Собственные значения, рассчитанные методом собственного ортогонального разложения, соответствуют кинетической энергии, содержащейся в каждой характеристической моде. Коэффициенты, не зависящие от времени, определяют вклад каждого элементарного объема и позволяют определить области, влияющие на среднее течение на определенной частоте после применения быстрого преобразования Фурье к временным коэффициентам ортогонального разложения. В результате применения метода, характеристические моды отсортированы в соответствии с их кинетической энергией. Нулевая мода соответствует наибольшей энергии, описывает среднее течение и имеет относительно малые амплитуды. Описанный подход был проверен на вычислительно простой двумерной задаче и затем применен к высокоскоростному компрессору. Было показано, что третья характеристическая мода течения имеет пик на частоте 12970 Гц, около двух резонансных частот 12000 Гц и 13700 Гц. Третья и четвертая характеристические моды представляют собой флуктуации давления в спутном следе позади лопаток. Описанный подход позволит анализировать динамику течения более эффективно, в сравнении с быстрым преобразованием Фурье. Кроме того, метод может быть использован для сжатия данных и разработки моделей пониженной размерности.

Ключевые слова: собственное ортогональное разложение; центробежный компрессор; механика жидкостей.

DOI: 10.15514/ISPRAS-2017-29(1)-2

Для цитирования: Калугин М.Д., Евдокимов И.Е. Численное исследование характеристических мод и частот течения в высокоскоростных компрессорах. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 21-38 (на английском). DOI: 10.15514/ISPRAS-2017-29(1)-2

Список литературы

1 Epureanu, BI; Dowell, EH; Hall, KC. A parametric analysis of reduced order models of potential flows in turbomachinery using proper orthogonal decomposition. Proceedings of the ASME Turbo Expo, vol. 1 (2001). doi: 10.1115/2001-GT-0434.

[12]. Rochuon, N., Trébinjac, I., Billonnet, G. An Extraction of the Dominant Rotor-Stator Interaction Modes by the Use of Proper Orthogonal Decomposition (POD). Journal of Thermal Science, Science Press, Vol.15, N°2, pp.109-114, June 2006.

[13]. Clark ST, Besem FM, Kielb RE, Thomas JP. Developing a Reduced-Order Model of Nonsynchronous Vibration in Turbomachinery Using Proper-Orthogonal Decomposition Methods. ASME. J. Eng. Gas Turbines Power. 2015;137(5). doi:10.1115/1.4028675.

[14]. Danaila, S., Niculescu, M. L. Unsteady effects at the interface between impeller-vaned diffuser in a low pressure centrifugal compressor. INCAS Bulletin5.1, 2013, p. 71-86.

[15]. Fossati,M., Nilamdeen, S., Habashi, W.G., Moustapha, H. Parametric Analysis of 3D Turbomachinery flows via Reduced Order Modelling. Conference Paper, 21st International Symposium on Air Breathing Engines, September 2013.

[16]. Kraposhin, M., Bovtrikova, A., Strijhak, S. Adaptation of Kurganov-Tadmor Numerical Scheme for Applying in Combination with the PISO Method in Numerical Simulation of Flows in a Wide Range of Mach Numbers. Procedia Computer Science, Vol. 66, 2015, Pages 43-52, ISSN 1877-0509, doi: 10.1016/j.procs.2015.11.007.

[17]. Amirante, D. and Hills, N.J. and Barnes, C.J. A moving mesh algorithm for aero-thermo-mechanical modelling in turbomachinery. International Journal for Numerical Methods in Fluids, Vol. 70 (2012), Number 9, p. 1118–1138, doi: 10.1002/fld.2734

[18]. Petit, O., Nilson, H., Page, M. and Beaudoin. The ERCOFTAC Centrifugal Pump OpenFOAM Case-Study. In Proceedings of the 3rd IAHR International Meeting of the Workgroup on Cavitation and Dynamic Problem in Hydraulic Machinery and Systems, Brno, Czech Republic.

[19]. Combès, J.F., Test Case U3: Centrifugal Pump with a Vaned Diffuser. ERCOFTAC Seminar and Workshop on Turbomachinery Flow Prediction VII, Aussois, jan 4-7, 1999.

[20]. Ubaldi M., Zunino P., Barigozzi G. and Cattanei A. An Experimental Investigation of Stator Induced Unsteadiness on Centrifugal Impeller Outflow. Journal of Turbomachinery, vol.118, 41-54, 1996.

[21]. Petit, P. and Nilsson, H. Numerical Investigations of Unsteady Flow in a Centrifugal Pump with a Vaned Diffuser, International Journal of Rotating Machinery, Volume 2013, 14 pages, doi: 10.1155/2013/961580

Тестирование возможностей открытого кода BEM++ по решению задач акустики

П.С. Лукашин[1] <ski1l@mail.ru>
С.В. Стрижак[2] <strijhak@yandex.ru>
Г.А. Щеглов[1] <shcheglov_ga@bmstu.ru>
[1] *МГТУ им. Н.Э. Баумана,*
105005, г. Москва, ул. 2-я Бауманская, д.5. стр.1
[2] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Проводится тестирование возможностей открытого программно-математического обеспечения BEM++ по решению задач акустики в области средних и высоких частот. Пакет BEM++, аналогично пакету OpenFOAM, является универсальным инструментом, который позволяет строить дискретные модели для граничных интегральных операторов (потенциальные операторы простого и двойного слоев, сингулярные операторы, сопряженные операторы двойного слоя и др.), и программировать с использованием библиотек языка Python решение различных МГЭ-задач для уравнений Лапласа, Гельмгольца и Максвелла. Сравнение с известными аналитическими решениями тестовых задач рассеяния акустической волны на сфере методом граничных элементов показывает, что открытый пакет BEM++ можно использовать «как есть» в качестве альтернативы известным коммерческим пакетам для получения результатов с точностью порядка 5%, достаточной в инженерных приложениях. Пакет позволяет эффективно проводить расчеты в диапазоне частот от 5 Гц до 5 кГц, важном с точки зрения разработки аэрокосмических систем, что дает возможность перехода к более сложным прикладным задачам. Главным ограничением при решении задач в настоящее время служит распараллеливание расчетов, которое ограничивается только системами с общей памятью. Однако, открытая архитектура BEM++ позволит при дальнейшей работе устранить данный недостаток. Возможности BEM++ позволяют работать с сетками большой размерности, описывающими сложные геометрические объекты, построенными на базе конструкторских электронных геометрических моделей. Следует, однако, отметить, что для внедрения в инженерную практику желательна разработка интерфейса с существующими интерактивными системами препостпроцессинга, например, SALOME.

Ключевые слова: Акустика; метод граничных элементов; задачи рассеяния; BEM++; уравнение Гельмгольца; жесткое рассеяние; мягкое рассеяние; граничные интегральные уравнения.

DOI: 10.15514/ISPRAS-2017-29(1)-3

Для цитирования: Лукашин П.С., Стрижак С.В., Щеглов Г.А. Тестирование возможностей открытого кода BEM++ по решению задач акустики. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 39-52. DOI: 10.15514/ISPRAS-2017-29(1)- 3

1. Введение

Решение задач математического моделирования акустических полей имеет ключевое значение при разработке новых образцов авиационной и ракетно-космической техники, поскольку позволяет повысить эффективность поиска способов и средств снижения негативного влияния данных воздействий. В частности, большую важность имеет расчет акустического давления, создаваемого двигателями ракеты-носителя космического назначения при старте, на элементы стартового комплекса и полезную нагрузку, находящуюся под головным обтекателем [1-3]. Полетные условия акустического нагружения летательных аппаратов в наземных условиях практически не воспроизводятся и здесь потребность в проведении математического моделирования еще более возрастает [4].

При проектировании аэрокосмических систем акустические процессы необходимо рассматривать в широком диапазоне частот от 5 Гц до 5 кГц [5]. Эффективность численных методов конечных и граничных элементов, наиболее часто используемых в настоящее время для расчета акустических воздействий существенно зависит от диапазона исследуемых частот. Если в области низких и средних частот данные методы позволяют получать результаты с малыми затратами вычислительных ресурсов, то в области высоких частот указанные методы работают на пределе возможностей современных компьютеров, поскольку увеличение исследуемой частоты влечет за собой существенное сгущение расчетной сетки. Необходимость проведения исследований в высокочастотном диапазоне для таких крупногабаритных объектов, как стартовый комплекс ракеты космического назначения, делает выбор наилучшего инструмента из существующего программно-математического обеспечения актуальной задачей.

Для численного моделирования акустических явлений в настоящее время используются специализированные коммерческие пакеты прикладных программ, такие как MSC Actran [6], LMS Virtual.Lab (со встроенным пакетом Sysnoise) [7], ANSYS [8], которые позволяют решать широкий класс задач, однако имеют все недостатки, присущие проприетарному программному обеспечению: закрытый исходный код, высокую стоимость лицензий и полную юридическую зависимость от владельца лицензии вплоть до возможности отзыва лицензии в случае применения экономических санкций.

В качестве альтернативы данным пакетам можно рассмотреть программное обеспечение с открытым исходным кодом, находящееся в общественном пользовании. В настоящее время доступно достаточное количество подобных акустических пакетов, чаще всего представляющих собой приложения для

моделирующих сред типа MATLAB/GNU Octave [9]. Производительность таких приложений недостаточна для решения индустриальных задач.

Возможность расчета акустических задач методом конечных элементов имеется в открытых пакетах Code-Aster [10] и Elmer [11], имеющих значительные сообщества пользователей и серьезные индустриальные приложения. Акустические аналогии используются для обработки результатов расчета течений методом конечного объема в открытом пакете OpenFOAM [12]. Однако, указанные подходы требуют построения сетки в расчетной области, имеющей конечные размеры, а для расчета акустического воздействия на стартовый комплекс желательно решать задачу в неограниченной области. Такую возможность дает метод граничного элемента (МГЭ), имеющий английскую аббревиатуру BEM (Boundary Element Method) [13, 14].

В МГЭ условия затухания возмущений на бесконечном удалении от исследуемых объектов выполняются автоматически, а построение сетки требуется только на границах тел. Недостатком МГЭ является заполненная матрица системы линейных алгебраических уравнений, которую необходимо вычислять заново для каждой исследуемой частоты. Затраты вычислительных ресурсов при этом оказываются пропорциональны кубу от числа узлов сетки, что существенно затрудняет расчеты в высокочастотной области. Для снижения вычислительных затрат разработаны модификации МГЭ, в которых используются эффективные методы работы с матрицами, например, иерархические матрицы (H-matrices) [15], метод мультипольных разложений [16], а также различные допущения, позволяющие упростить вычисления высокочастотных воздействий, например метод HFBEM (High Frequency BEM) [17]. Эти эффективные методы реализованы как в коммерческих пакетах, например, FastBEM [18], так и в программах с открытым исходным кодом, например, в коде AcouSTO, разработанном коллективом итальянских ученых, но к сожалению, не получившем пока серьезного развития и поддержки [19] или коде BEM++, разработанном международным сообществом ученых из University College London и Pontificia Universidad Católica de Chile и имеющем быстро растущее сообщество пользователей [20].

Пакет BEM++, аналогично пакету OpenFOAM, является универсальным открытым программно-математическим обеспечением, которое позволяет строить дискретные модели для граничных интегральных операторов (потенциальные операторы простого и двойного слоев, сингулярные операторы, сопряженные операторы двойного слоя и др.), и программировать с использованием библиотек языка Python решение различных МГЭ-задач для уравнений Лапласа, Гельмгольца и Максвелла. К сожалению, документация по пакету BEM++ в настоящее время является недостаточно подробной. В связи с этим требуется проведение большого объема методической работы для всесторонней оценки пакета и внедрения его в практику инженерных

расчетов. Данная методическая работа начата коллективом авторов и первые результаты, полученные в 2016 году представлены в статье.

Целью работы является тестирование возможностей пакета BEM++ по решению тестовых задач акустики в области средних и высоких частот. Для этого рассматриваются классические модельные задачи, имеющие аналитическое решение. Проводится оценка затрат вычислительных ресурсов и точность получаемых результатов.

2. Математическое описание тестовых задач

Распространение упругих волн в однородной среде описывается уравнением

$$\nabla^2 u - \frac{1}{c^2}\frac{\partial^2 u}{\partial t^2} = 0, \quad (1)$$

где $u = f(x,t)$ – акустическое давление; x – вектор координат; t – время; c – скорость звука в среде. В предположении, что решение данного уравнения является произведением стационарного поля $U(x)$ и периодической функции времени

$$u(x,t) = U(x)e^{-i\omega t},$$

уравнение (1) сводится к уравнению Гельмгольца:

$$\nabla^2 U + k^2 U = 0, \quad (2)$$

где $k = \frac{\omega}{c} = \frac{2\pi f}{c}$ – волновое число; ω – циклическая частота колебаний (рад/с); f – частота колебаний (Гц).

Для внешних задач, исследующих поле давления в неограниченной области, необходимо учесть условие Зоммерфельда (затухание волн на бесконечности). В наиболее общем виде условия на границе исследуемого тела можно записать в виде:

$$\frac{\partial U}{\partial n} - i\beta U = h. \quad (3)$$

В зависимости от граничных условий, задачи акустики можно разделить на задачи излучения и рассеяния. В задачах излучения, где рассматривается некоторая конструкция, являющаяся источником акустических волн, правая часть выражения (3), которая описывает движение поверхности данной конструкции, не равна нулю. Для задач рассеяния, где внешнее акустическое воздействие рассеивается на препятствии, функция h равна нулю. В зависимости от значения коэффициента β можно получить граничные условия Дирихле ($|\beta| \gg 1$), Неймана ($\beta = 0$) или смешанные граничные условия.

Для тестирования пакета BEM++ использовалась классическая задача рассеяния плоской волны на сфере единичного радиуса с центром в начале координат, расчетная схема которой показана на рис. 1.

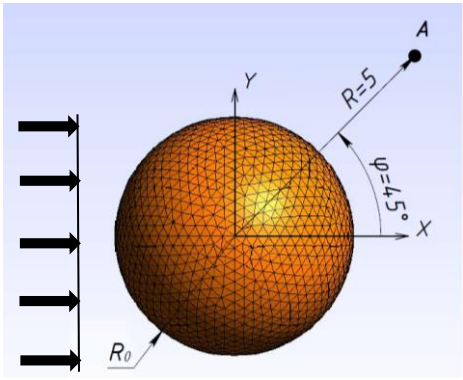


Рис. 1. Рассеяние плоской волны на сфере
Fig. 1. Plane wave scattering by a sphere

При наличии в пространстве сферического препятствия искомое результирующее поле давления есть сумма плоской падающей и сферической рассеянной волн:

$$U = U^S + U^I. \tag{4}$$

Здесь падающая волна U^I описывается уравнением плоской волны:

$$U^I = U_0 e^{ikx\cdot a} \tag{5}$$

где a – направляющий вектор; U_0 – амплитуда; k – волновое число.

На рассеянную волну U^S накладывается условие затухания на бесконечности:

$$\lim_{R\to\infty}\left[R\left|\frac{\partial U^S}{\partial R} - ikU^S\right|\right] = 0. \tag{6}$$

Тестовые задачи были решены для двух типов граничных условий на сфере: акустически жесткое рассеяние $\frac{\partial U}{\partial n}|_S = 0$ (нулевые граничные условия Неймана) и акустически мягкое рассеяние $U|_S = 0$ (нулевые граничные условия Дирихле).

Полная математическая формулировка задачи имеет вид:

$$\begin{cases} \nabla^2 U + k^2 U = 0, \\ \frac{\partial U}{\partial n} - i\beta U = 0, \\ U = U^S + U^I, \\ U^I = U_0 e^{ikx\cdot a}, \\ \lim_{R\to\infty}\left[R\left|\frac{\partial U^S}{\partial R} - ikU^S\right|\right] = 0. \end{cases} \tag{7}$$

Она может быть сведена к форме граничного интегрального уравнения; для акустически жесткого рассеяния граничное интегральное уравнение имеет вид

$$\left(-H + i\eta\left(\frac{1}{2}I - K\right)\right)U^* = -\left(\frac{\partial U^I}{\partial n} - i\eta U^I\right)|_s; \tag{8}$$

для акустически мягкого рассеяния граничное интегральное уравнение принимает форму

$$\left(\frac{1}{2}I + T - i\eta S\right) U_n^* = \left(\frac{\partial U^I}{\partial n} - i\eta U^I\right)|_s. \quad (9)$$

Здесь η – произвольно выбираемая константа с ненулевой вещественной частью.

Неизвестными являются соответственно поля давления U^* и нормальная производная давлений U_n^* на поверхности.

В уравнениях (8), (9) под соответствующими обозначениями применяются следующие граничные интегральные операторы:

- оператор простого слоя $S[U](x) = \int_S G(x,y)U(y)dS(y)$;
- оператор двойного слоя $K[U](x) = \int_S \frac{\partial G(x,y)}{\partial n(y)}U(y)dS(y)$;
- сопряженный оператор двойного слоя $T[U](x) = \int_S \frac{\partial G(x,y)}{\partial n(x)}U(y)dS(y)$;
- гиперсингулярный оператор $H[U](x) = -\frac{\partial}{\partial n(x)}\int_S \frac{\partial G(x,y)}{\partial n(y)}U(y)dS(y)$;
- тождественный оператор $I[U](x) = U(x)$.

Здесь $G(x,y) = \frac{e^{ik|x-y|}}{4\pi|x-y|}$ – функция Грина для уравнения Гельмгольца,

Задача (7) для сферы имеет аналитическое решение [21]. Падающую волну раскладывают по полиномам Лежандра $P_l(\cos\theta)$

$$U^I = \sum_{l=0}^{\infty}(2l+1)i^{-l}j_l(kr)\,P_l(\cos\theta). \quad (10)$$

Рассеянная волна, удовлетворяющая условиям Зоммерфельда и уравнению Гельмгольца, описывается суммой следующего вида:

$$U^S = -\sum_{l=0}^{\infty} A_l h_l(kr) P_l(\cos\theta). \quad (11)$$

В уравнениях (10) и (11) $j_l(kr)$ – сферические функции Бесселя *l*-го порядка; $h_l(kr)$ – сферические функции Ганкеля *l*-го порядка; коэффициент A_l получают из граничных условий:

- для задачи Неймана: $A_l = -U_0(2l+1)i^{-l}\frac{j_l(kR_0)}{h_l(kR_0)}$;
- для задачи Дирихле: $A_l = -U_0(2l+1)i^{-l}\frac{j_l(kR_0)-(l+1)j_{l+1}(kR_0)}{h_l(kR_0)-(l+1)h_{l+1}(kR_0)}$.

Общее поле давлений получают суммированием полей (10) и (11).

3. Особенности решения задач в BEM++

Пакет BEM++ представляет собой открытую (с лицензией MIT) Python-библиотеку, в которой реализована технология распараллеливания кода для работы на серверах с общей памятью, а также метод иерархических матриц (*h*-matrices) для проведения операций с заполненными матрицами. На данный момент поддерживается работа пакета на платформах Mac и Linux.

Общая структура библиотеки включает 5 модулей. Ключевым является модуль Fiber (Fast Integration Boundary Element Routines), содержащий

процедуры для быстрого интегрирования взаимного влияния граничных элементов. Модуль Space работает с пространствами функций, определенных на элементах сетки. Модуль Assembly предназначен для сборки матриц граничных интегральных операторов и функций, определенных на сетке, т.е. формирует матрицу из элементов, предоставленных модулем Fiber. Модуль Grid предназначен для работы с сетками. Модуль Linalg обеспечивает возможность решения СЛАУ. Детальное описание процедур представлено в документации [20, 22].

Пакет не имеет графического интерфейса пользователя. Ниже, в качестве иллюстрации, рассмотрены примеры реализации основных этапов решения задачи (8) с помощью инструментов библиотеки BEM++.

Создание сетки для сферы с линейным размером элемента h:

```
grid = bempp.api.shapes.sphere (h=0.1)
```

Создание пространства кусочно-постоянных функций на элементах сетки:

```
space = bempp.api.function_space(grid, “DP”,0)
```

Определение значений функции на сетке созданием объекта GridFunction:

```
grid_fun = bempp.api.GridFunction(space, fun=function)
```

Определение граничных операторов:

тождественный оператор

```
I = bempp.api.operators.boundary.sparse.identity(space, space, space)
```

сингулярный оператор

```
H = bempp.api.operators.boundary.helmholtz.hypersingular (space, space, space, k)
```

оператор двойного слоя

```
K = bempp.api.operators.boundary.helmholtz.double_layer (space, space, space, k)
```

При этом учет граничных условий на бесконечности обеспечивается структурой операторов библиотеки BEM++.

Формирование и решение системы уравнений, соответствующих (8):

```
lhs=-H+1j*k*(0.5*I-K)
func,info=bempp.api.linalg.gmres(lhs, grid_fun, tol=1e-3)
```

Сохранение результатов

```
res = np.absolute(u_inc + K.evaluate(func))
```

Таким образом, расчетная схема задачи описывается в виде обычной Python-программы. Следует отметить, что подобный интерфейс представляет некоторые трудности для инженерного анализа конструкций и одним из направлений развития данного пакета может быть разработка интерфейсов с открытыми графическими средами типа SALOME или FreeCAD.

4. Результаты

Численное решение задач рассеяния проводилось для плоской волны с параметрами $U_0 = 1;\ a = \{1; 0; 0\}$ на сфере единичного радиуса ($R_0 = 1$), находящейся в однородной среде, скорость звука в которой $c = 331$ м/с (соответствует скорости звука в воздухе при нормальных условиях). Во всех расчетах использовались сетки с треугольными элементами, максимальный линейный размер которых не менее 6 раз укладывается в длину волны,

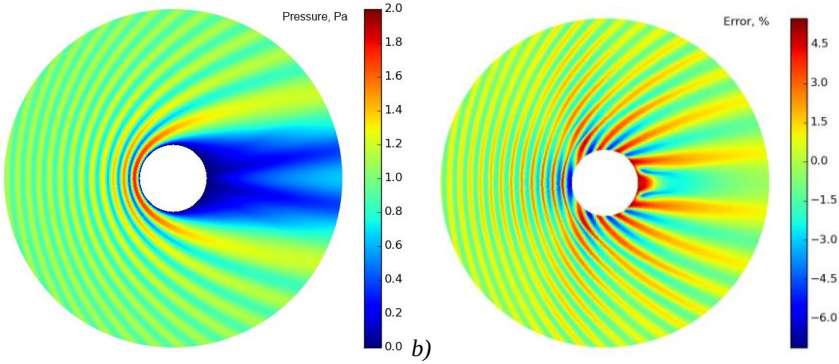
соответствующей заданному волновому числу k. В частности, при $k = 10$ использовалась сетка с 3206 ячейками.

Для оценки точности рассчитывалась относительная погрешность как разность между аналитическим и численным решением, отнесенная к амплитуде давления падающей волны:

$$\varepsilon = \frac{U_{an} - U_{bem}}{U_0} \cdot 100\ \%.$$

При сравнении результатов необходимо принять во внимание влияние параметра точности итерационного метода решения СЛАУ, а также количество учитываемых членов ряда для аналитического решения. В данной задаче решение СЛАУ осуществлялось с точностью 10^{-3}, а в аналитическом решении учитывалось 220 членов ряда.

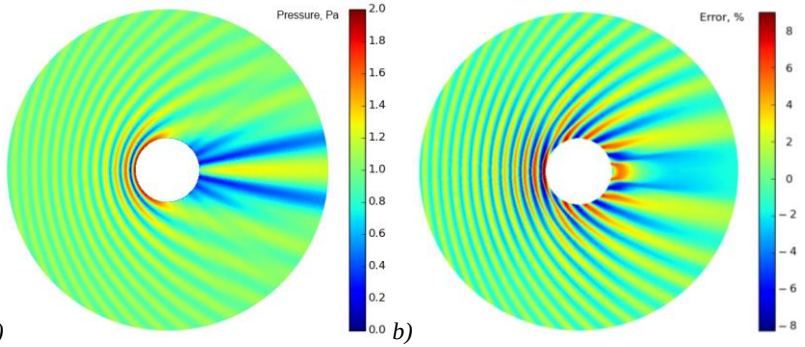
На рис. 2, *a* дана эпюра распределения давления, полученная в BEM++ для задачи акустически мягкого рассеяния (задача Дирихле) с волновым числом $k = 10$ (527 Гц). Соответствующая эпюра погрешности ε, приведенная на рис. 2, *b*, показывает, что наибольшая погрешность порядка ±5,5 % имеет место вблизи поверхности сферы.



a) *b)*

Рис. 2. Результат решения задачи мягкого рассеяния (задача Дирихле, $k = 10$):
a) Эпюра давления в плоскости OXY в пределах круга радиуса R = 5.
b) Эпюра относительных погрешностей.
Fig. 2. The result for sound-soft scattering (Dirirchlet problem, k=10):
a) Pressure plot in the plane OXY inside the circle of radius 5.
b) The plot of relative pressure error.

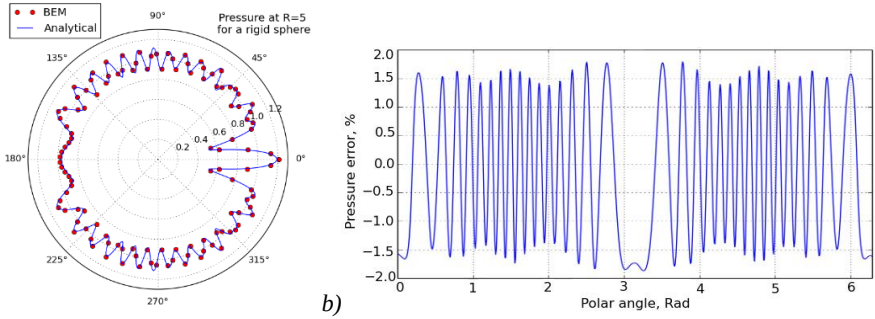
На рис. 3 отображены эпюры распределения давления и относительной погрешности, полученные в BEM++ для задачи акустически жесткого рассеяния (задача Неймана) с волновым числом $k = 10$ (527 Гц). Здесь наибольшая погрешность вблизи поверхности сферы составляет ±7,5%.



a) *b)*

Рис. 3. Результат решения задачи жесткого рассеяния (задача Неймана, $k = 10$):
a) Эпюра давления в плоскости OXY в пределах круга радиуса R = 5.
b) Эпюра относительных погрешностей.
Fig. 3. The result for sound-hard scattering (Neumann problem, k=10):
a) Pressure plot in the plane OXY inside the circle of radius 5.
b) The plot of relative pressure error.

Как следует из полярного графика давлений, построенного на окружности радиуса $R = 5$ (рис. 4, *a*), на большом удалении от сферы результаты численного и аналитического решения совпадают значительно лучше. Наибольшее значение погрешности, как видно из рис. 4, *b*, не превышает 2 %.

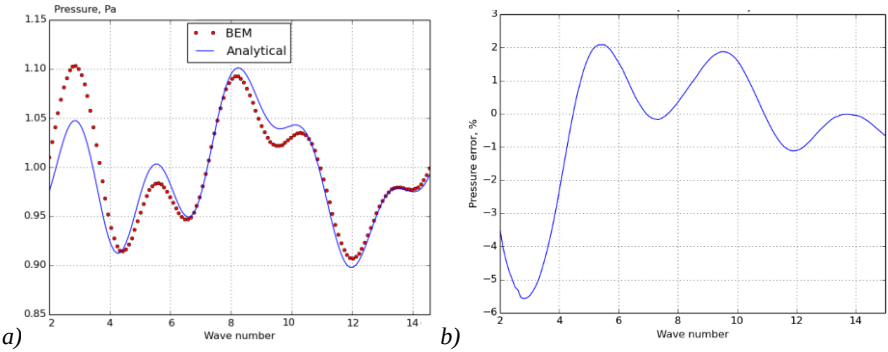


a) *b)*

Рис. 4. Результат решения задачи жесткого рассеяния (задача Неймана, $k = 10$):
a) Полярный график давления на окружности радиуса R = 5.
b) График относительной погрешности давлений на окружности радиуса R = 5.
Fig. 4. The result for sound-hard scattering (Neumann problem, k = 10):
a) The pressure polar plot at circumference of radius R=5.
b) The plot of relative pressure error at circumference of radius R=5.

Для многих прикладных задач требуется определить спектр акустического давления в некоторой исследуемой точке пространства. На основе описанной ранее модели был построен спектр давления для диапазона $k \in [2; 15]$ (105…790 Гц) в точке *A* (см. рис. 1) при граничных условиях Неймана. При решении задачи использовались различные сетки с линейным размером

элемента $h = \frac{2\pi}{6k}$, таким образом, в каждой расчётной точке линейный размер элемента укладывался в длину волны 6 раз. Время выполнения расчета для одной точки возрастает с увеличением частоты и соответствующим сгущением сетки. Как следует из рис. 5, *a*, положения максимумов давления в аналитическом и численном решении практически совпадают, а погрешность уменьшается с ростом волнового числа. Максимальное значение погрешности, как показывает рис. 5, *b*, составляет 5,5 % для $k \approx 3$, а для $k = 15$ погрешность не превышает 1 %.



a) *b)*

Рис. 5. Результаты расчета спектра давления в точке A:
a) Сравнение спектров в точке A. b) Погрешность давлений в точке A.
Fig. 5. The result for pressure range computation
a) Comparison of range at point A. b) The pressure error at point A.

Для определения возможности решения с помощью BEM++ задач с высокочастотным воздействием была решена задача жесткого рассеяния при $k = 95$ ($f = 5$ кГц). Для данного волнового числа число ячеек сетки составило около 300 тысяч. Время счета на 12 ядрах составило 2 часа при тактовой частоте 3 ГГц. Для решения системы линейных алгебраических уравнений был использован метод GMRES с настройками по умолчанию. Погрешности определения давления на окружности радиуса $R = 5$ не превышают 3 %.

5. Выводы

Сравнение с известными аналитическими решениями результатов решения тестовых задач рассеяния акустической волны на сфере методом граничных элементов показывает, что открытый пакет BEM++ можно использовать «как есть» в качестве альтернативы известным коммерческим пакетам для получения результатов с точностью, достаточной в инженерных приложениях.

Пакет позволяет эффективно проводить расчеты в диапазоне частот от 5 Гц до 5 кГц, важном с точки зрения разработки аэрокосмических систем, что дает возможность перехода к решению более сложных прикладных задач. Главным

ограничением при решении задач в настоящее время является невозможность проведения расчетов в параллельном режиме на системах с распределенной памятью (кластерах).

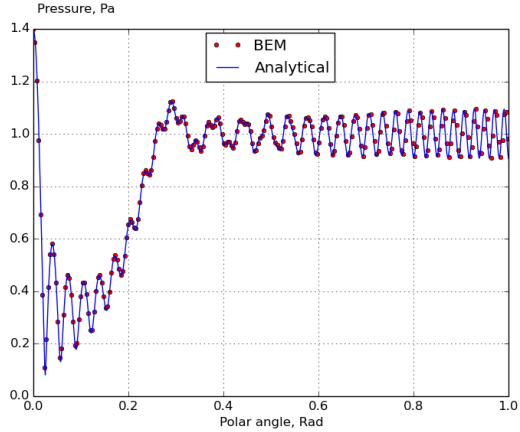


Рис. 6. График давления на окрсти радиуса R = 5 для задачи Неймана при $k = 95$ ($f = 5$ кГц)

Fig.6. The pressure plot at circumference of radius R = 5 for Neumann problem in case $k = 95$ ($f = 5$ kHz)

Возможности BEM++ позволяют работать с сетками большой размерности, описывающими сложные геометрические объекты, построенными на базе конструкторских электронных геометрических моделей. Следует, однако, отметить, что для внедрения в инженерную практику желательна разработка интерфейса с существующими интерактивными системами препостпроцессинга, например, SALOME.

Список литературы

[1]. Носатенко П.Я., Бобров А.В., Баранов М.Л., Шляпников А.Н. Экспериментальное определение акустических нагрузок при пусках РН «Стрела» и расчётное определение режимов экспериментальной отработки выводимых космических аппаратов. Вестник Самарского государственного аэрокосмического университета. 2010. № 2. С. 112-123.

[2]. Дядькин А.А. Аэрогазодинамика ракетно-космического комплекса «Морской старт». Космическая техника и технологии. 2014. № 2 (5). С. 14-31.

[3]. Troclet B., Alestra S., Srithammavanh V., Terrasse I. A Time Domain Inverse Method for Identification of Random Acoustic Sources at Launch Vehicle Lift-Off. J. Vib. Acoust. 2011. Vol. 133, No. 2. Pp. 1-11.

[4]. Колесников А.В. Лекции по курсу «Испытания конструкций и систем космических аппаратов». 2007. URL: airspot.ru/book/file/659/ispytanija_ka.pdf (дата обращения: 28.12.2016).

[5]. Либерман М.Ю. О моделировании процессов формирования пусковых нагрузок, оказывающих динамическое воздействие на космический аппарат. Вопросы электромеханики. Труды ВНИИЭМ. 2013. Т. 136, № 5. С. 19-30.

[6]. Actran - программный комплекс для анализа акустики.

URL: www.mscsoftware.ru/products/actran (дата обращения: 28.12.2016).
[7]. LMS Virtual.Lab Acoustics for Acoustic Simulation. URL: http://www.plm.automation.siemens.com/ru_ru/products/lms/virtual-lab/acoustics (дата обращения: 28.12.2016).
[8]. ANSYS Structures Harmonic Vibrations and Acoustics, URL: ansys.com/products/structures/vibrations/harmonic-vibrations-and-acoustics, (дата обращения: 28.12.2016).
[9]. Free Acoustics and Ultrasound Software. URL: www.k-wave.org/acousticsoftware.php (дата обращения: 28.12.2016).
[10]. CodeAster manual. R4.02.01. Finite elements in acoustic. URL: www.code-aster.org/doc/v12/en/man_r/r4/r4.02.01.pdf (дата обращения: 28.12.2016).
[11]. Elmer. URL:csc.fi/web/elmer/elmer (дата обращения: 28.12.2016).
[12]. Kraposhin M.V., Strizhak S.V. How to Implement Simple Acoustic Analogy in OpenFOAM. 8th International OpenFOAM Workshop 2013, Jeju, Korea.
[13]. Купрадзе В.Д. Граничные задачи теории колебаний и интегральные уравнения. М.: ГИТТЛ, 1950. 280 с.
[14]. Boundary element method. URL: www.boundary-element-method.com (дата обращения: 28.12.2016).
[15]. Hackbusch W. Hierarchical matrices: Algorithms and analysis. Springer, 2015. 510 p.
[16]. Liu Y.J. Fast Multipole Boundary Element Method - Theory and Applications in Engineering. New-York, Cambridge University Press, 2009. 235 p.
[17]. Chandler-Wilde S.N., Langdon S., Graham I.G., Spence E.A. Numerical-asymptotic boundary integral methods in high-frequency acoustic scattering. Acta Numerica. 2012. Vol. 21. Pp. 89-305.
[18]. FastBEM Acoustics. URL: www.fastbem.com (дата обращения: 28.12.2016).
[19]. AcouSTO. URL: acousto.sourceforge.net, (дата обращения: 28.12.2016).
[20]. The BEM++ project. URL: www.bempp.org, (дата обращения: 28.12.2016).
[21]. Лепендин Л.Ф. Акустика. М.: Высшая школа, 1978. 448 с.
[22]. Betcke T., Arridge S., Phellips J., Schweiger M. Solving Boundary Integral Problems with BEM++. URL: http://www.bempp.org/files/bempp-toms-preprint.pdf (дата обращения: 28.12.2016).

Validation of open source code BEM++ for simulation of acoustic problems

P.S. Lukashin[1] <ski1l@mail.ru>
S.V. Strijhak[2] <strijhak@yandex.ru>
G.A. Shcheglov[1] <shcheglov_ga@bmstu.ru>
[1]Bauman Moscow State Technical University,
5/1, 2-th Baumanskaya st., Moscow. 105005, Russia
[2] Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. Testing of capabilities of open-source BEM++ code for simulation of acoustics problems at medium and high frequencies is presented. The BEM++ library is a universal

tool, which allows to build discrete models for boundary integral operators (single-, double- and adjoint double-layer potential operators and hypersingular boundary operators) and solve boundary element method problems for Helmholtz, Laplace and Maxwell equations using Python libraries. Solution for the test problem of scattering plane wave on spherical obstacle with using BEM++ demonstrates good convergence with the results of analytical solutions. The relative errors satisfy to acceptable values 5% in solving engineering tasks, this fact allows to use this library as an alternative to commercial software. Capability of BEM++ library to calculate acoustic fields for frequencies from 5 Hz to 5 kHz enables move to solving more difficult engineering challenges of the aerospace industry. The main restriction for this is a time of computation, because only shared-memory technology of the code parallelization is implemented. However, open architecture of the library allows to remove this disadvantage. Meshes for BEM++ can have big size and be based on E geometric model with complex geometrical objects. Also, it should be noted, that for implementation to engineering practice it is desirable to integrate the library with existing interactive systems of pre- and post-processing, for example, with Salome.

Keywords: Acoustics; boundary element method; scattering problem; BEM++; Helmholtz equation; rigid scattering; soft scattering; boundary integral equations.

DOI: 10.15514/ISPRAS-2017-29(1)-3

For citation: Lukashin P.S., Strijhak S.V., Shcheglov G.A Validation of open-source BEM++ code for simulation of acoustics problems. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 1, 2017, pp. 39-52 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-3

References

[1]. Nosatenko P.YA., Bobrov A.V., Baranov M.L., Shlyapnikov A.N. Experimental determination of acoustic loads at launching of rocket media “Strela” and calculation of the modes of experimental testing of spacecrafts. Vestnik Samarskogo gosudarstvennogo aehrokosmicheskogo universiteta [Vestnik of Samara state aerospace University], 2010, № 2. pp. 112-123 (in Russian).

[2]. Dyad'kin A.A. Aerogasdynamics of rocket and space complex "Sea launch". Kosmicheskaya tekhnika i tekhnologii [Space engineering and technology], 2014, № 2 (5). pp. 14-31 (in Russian).

[3]. Troclet B., Alestra S., Srithammavanh V., Terrasse I. A Time Domain Inverse Method for Identification of Random Acoustic Sources at Launch Vehicle Lift-Off. J. Vib. Acoust. 2011, 133(2) pp. 1-11.

[4]. Kolesnikov A.V. Lecture series "Testing of the constructions and systems of space vehicles", 2007, Available at: airspot.ru/book/file/659/ispytanija_ka.pdf, accessed 28.12.2016 (in Russian).

[5]. Liberman M.YU. Modeling the formation of a launcher loads that have a dynamic effect on spacecraft. Voprosy ehlektromekhaniki. Trudy VNIIEM, [Questions of electromechanics, Proc. VNIIEM], 2013, v. 136. №5. pp. 19-30 (in Russian).

[6]. Actran – software package for an acoustic analysis, www.mscsoftware.ru/products/actran, accessed 28.12.2016.

[7]. LMS Virtual.Lab Acoustics for Acoustic Simulation, http://www.plm.automation.siemens.com/ru_ru/products/lms/virtual-lab/acoustics, accessed 28.12.2016.
[8]. ANSYS Structures Harmonic Vibrations and Acoustics, www.ansys.com/products/structures/vibrations/harmonic-vibrations-and-acoustics, accessed 28.12.2016.
[9]. Free Acoustics and Ultrasound Software, www.k-wave.org/acousticsoftware.php, accessed 28.12.2016.
[10]. CodeAster manual. R4.02.01. Finite elements in acoustic. Available at: www.code-aster.org/doc/v12/en/man_r/r4/r4.02.01.pdf, accessed 28.12.2016.
[11]. Elmer, csc.fi/web/elmer/elmer, accessed 28.12.2016.
[12]. Kraposhin, M.V. and Strizhak, S.V., How to Implement Simple Acoustic Analogy in OpenFOAM. 8th International OpenFOAM Workshop 2013, Jeju, Korea.
[13]. Kupradze V. D. Boundary problems of oscillation theory and integral equations. Moskva, Gos. izd-vo tekhniko-teoreticheskoj literatury [Moscow, State publishing technical and theoretical literature], 1950, 280 p. (in Russian).
[14]. Boundary element method, www.boundary-element-method.com, accessed 28.12.2016.
[15]. W. Hackbusch, Hierarchical matrices: Algorithms and analysis, Springer, Berlin, 2015, 510 p., doi: 10.1007/978-3-662-47324-5
[16]. Liu Y. J., Fast Multipole Boundary Element Method - Theory and Applications in Engineering, Cambridge University Press, New York, 2009, 235 p.
[17]. S.N.Chandler-Wilde, S.Langdon, I.G.Graham, E.A.Spence, Numerical-asymptotic boundary integral methods in high-frequency acoustic scattering, Acta Numerica, 2012, p. 89-305, doi: 10.1017/S0962492912000037.
[18]. FastBEM Acoustics, www.fastbem.com, accessed 28.12.2016.
[19]. AcouSTO, acousto.sourceforge.net/index.php, accessed 28.12.2016.
[20]. The BEM++ project: www.bempp.org, accessed 28.12.2016.
[21]. L.F. Lependin, Acoustics. Moskva «Vysshaja shkola» [Moscow "High school"], 1978, 448 p. (in Russian)
[22]. M. Betcke T., Arridge S., Phellips J., Schweiger M. Solving Boundary Integral Problems with BEM++. Available at: http://www.bempp.org/files/bempp-toms-preprint.pdf, accessed 28.12.2016.

Особенности построения расчетной схемы для моделирования динамики стабилизатора расхода в пакете OpenFOAM

В.Г. Мельникова <vg-melnikova@yandex.ru>
О.С. Коцур <oskotsur@gmail.com>
Г.А. Щеглов <shcheglov_ga@bmstu.ru>
Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана),
105005, Россия, Москва, 2-я Бауманская ул., д. 5, стр. 1

Аннотация. Представлены результаты методического исследования, направленного на тестирование возможностей свободно распространяемого пакета OpenFOAM по построению расчетной схемы для моделирования динамики гидравлических агрегатов с помощью метода контрольного объема и подвижных сеток. Рассмотрены выбор наиболее подходящих технологий динамических скользящих сеток; построение расчетной схемы для сопряженной задачи FSI о взаимодействии подвижного регулирующего элемента (золотника) с течением рабочей жидкости; расчет переходного режима движения золотника для прототипа стабилизатора расхода новой конструкции; анализ нестационарных течений во внутренней полости агрегата; анализ устойчивости и производительности счета. На примере упрощенной осесимметричной модели стабилизатора расхода несжимаемой среды описаны основные этапы подготовки расчетной схемы для моделирования задачи об установлении золотника в положение равновесия под действием гидродинамической силы и силы реакции пружины. Приведены результаты предварительного стационарного расчета при фиксированном золотнике, которые использовались в качестве начального условия. Подробно описаны методы реализации движения сетки, отслеживающей движение золотника и технология скользящих сеток GGI, которая использовалась для моделирования перекрытия отверстия. Приведены результаты расчета переходного режима: поля скоростей и давления, графики обобщенных сил, действующих на золотник, график перемещений золотника от времени. Расчеты показали механизм возникновения нестационарных струйных и вихревых течений в проточной части стабилизатора. Проведенное методическое исследование позволяет сделать вывод о том, что пакет с открытым исходным кодом OpenFOAM в версии extend может быть успешно использован в качестве альтернативы коммерческим пакетам программ вычислительной гидродинамики (CFD), поскольку в нем имеются все необходимые

средства для построения расчетных схем с деформируемыми сетками и расчета переходных режимов в агрегатах гидравлической автоматики, имеющих в конструкции подвижные элементы.

Ключевые слова: численное моделирование; вычислительная гидродинамика; FSI; OpenFOAM; открытое программное обеспечение; стабилизатор расхода

DOI: 10.15514/ISPRAS-2017-29(1)-4

Для цитирования: Мельникова В.Г., Коцур О.С., Щеглов Г.А. Особенности построения расчетной схемы для моделирования динамики стабилизатора расхода в пакете OpenFOAM. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 53-70. DOI: 10.15514/ISPRAS-2017-29(1)-4

1. Введение

Для наиболее полного удовлетворения специальным эксплуатационным требованиям при создании новых агрегатов гидравлической автоматики аэрокосмической техники важно проведение математического моделирования режимов их работы. В частности, для обеспечения синхронности выполнения операций исполнительными органами важной задачей является расчет параметров процессов, протекающих в стабилизаторах расхода (СР) золотникового типа. Данные агрегаты, за счет малых перемещений золотника, должны поддерживать с заданной точностью расход рабочей жидкости (РЖ) при значительных (более 10 МПа) перепадах давления между входом и выходом. При этом в проточной части регулируемых сечений возникают гидродинамические силы, вызывающие заклинивание золотника, что искажает расходно-перепадную характеристику СР. Движение РЖ здесь является существенно нестационарным из-за чего применение приближенных аналитических моделей гидродинамического нагружения золотника для выбора проектных параметров конструкции СР не позволяет получить удовлетворительного согласования с данными экспериментов [1].

При численном моделировании течения РЖ в регуляторах методом конечного объема в современных программных комплексах вычислительной гидродинамики (CFD) подвижностью золотника часто пренебрегают для уменьшения вычислительных затрат [2, 3]. Однако, чтобы в условиях больших перепадов давления в полной мере проанализировать нелинейное нестационарное нагружение золотника, необходимо рассматривать сопряженную задачу его движения при взаимодействии с потоком среды во внутренней полости агрегата – т.н. задачу FSI (Fluid-Structure Interaction).

Полноценное FSI-моделирование работы СР в программах CFD требует построения расчетных схем с использованием подвижной (перестраиваемой) сетки, описывающей такие особенности работы гидравлического агрегата как перекрытие отверстий и зазоров, резкое изменение объемов внутренних полостей при перемещениях золотника. Наиболее приспособленными для работы с подобными сложными расчетными схемами являются универсальные

коммерческие программы CFD, в частности, Fluent и CFX, входящие в комплекс ANSYS [4, 5]. Однако высокая стоимость владения пакетом, закрытость исходного кода и ограничения коммерческой лицензии, которые не защищают промышленные предприятия от возможных санкций со стороны иностранных правообладателей, делают задачу поиска альтернативных программных средств с открытым исходным кодом безусловно актуальной.

В настоящее время среди программ с открытым исходным кодом наибольшими возможностями по расчету полей различной природы методом контрольного объема обладает пакет OpenFOAM, который предоставляет пользователю большую свободу, позволяя модифицировать существующие и реализовывать новые численные модели с использованием обширной библиотеки алгоритмов [6]. Но универсальность данного пакета и полный доступ ко всем настройкам имеет в качестве побочного эффекта сложность построения и настройки расчетных схем. Для отработки и отладки технологий работы с подвижными деформируемыми и скользящими сетками в данной работе рассматривается тестовая модель СР, основанная на новой конструкции золотника, обеспечивающей высокую точность управления расходом РЖ в широком диапазоне изменений перепадов давления на регуляторе (от 5 до 35 МПа) [7].

Сечение геометрической модели прототипа СР показано на рис. 1. Агрегат состоит из разъемного корпуса со штуцерами входа и выхода, гильзы с отверстиями золотникового дросселя, установленной в корпусе, а также из размещенного в гильзе подпружиненного золотника, который выполнен в форме стакана с двумя скользящими опорами на концах. Одна из опор имеет острую кромку, которая перекрывает проходные сечения отверстий в гильзе как показано на рис. 2.

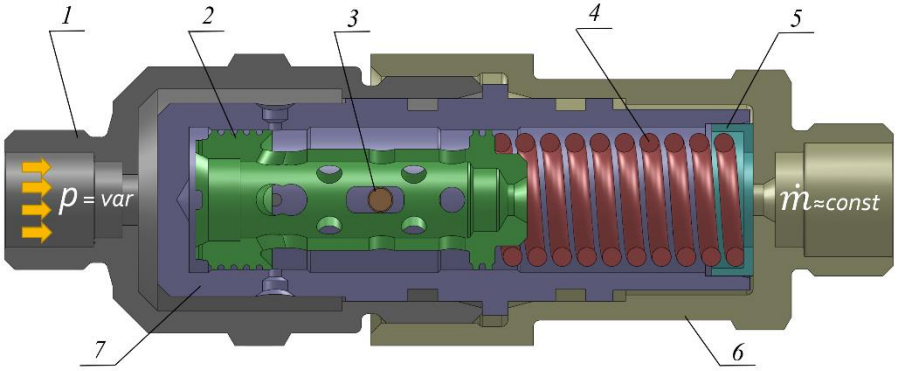


Рис. 1. Исследуемый стабилизатор расхода золотникового типа: 1 – штуцер входа, 2 – золотник, 3 – направляющий штифт, 4 – пружина, 5 – опора пружины, 6 – штуцер выхода, 7 – гильза

Fig. 1. The studied flow rate regulator valve prototype: 1 – inlet fitting, 2 – plunger, 3 – pin, 4 – spring, 5 – adjustment washer, 6 – outlet fitting, 7 – cylinder

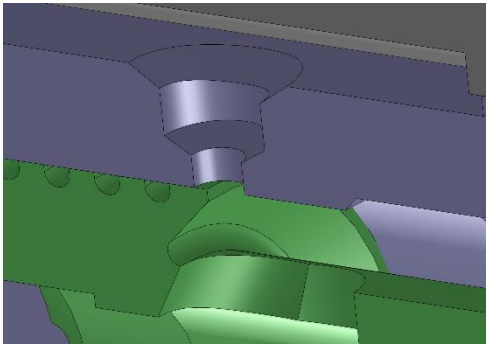


Рис. 2. Увеличенная зона с отверстием золотникового дросселя

Fig. 2. Scaled-up zone with the piston throttle hole

Кольцевой канал, образованный внутренней поверхностью гильзы и поверхностью золотника между опорами, сообщен с внутренней полостью золотника системой отверстий. Часть отверстий расположена напротив отверстий золотникового дросселя и способствует взаимной компенсации энергии истекающих струй РЖ. Направляющий штифт служит для точного позиционирования отверстий в гильзе напротив отверстий в золотнике. Другие отверстия предназначены для уменьшения воздействия на золотник радиальной составляющей гидродинамической силы путем снижения перепада давления между его внутренней полостью и кольцевым каналом. Таким образом, золотник СР разгружается от неравномерного распределения давления на цилиндрических поверхностях золотника, возникающих из-за нестационарных эффектов течения РЖ. Вынесение пружины из зоны регулирования дополнительно снижает нежелательное возмущающее действие гидродинамических сил [7].

В процессе работы при изменении перепада давления на агрегате золотник перемещается и изменяет размер проходного сечения отверстий золотникового дросселя (рис. 2), тем самым сохраняя постоянной величину расхода. Например, при уменьшении давления на входе расход уменьшается относительно требуемого значения. Соответственно, гидродинамическая сила, действующая на золотник, уменьшается, а сила упругости поджатой пружины превосходит гидродинамическую силу в данном положении, и золотник смещается влево, приоткрывая отверстие дросселя (разжимая пружину) до того момента, пока не будет достигнуто новое положение равновесия.

Результаты экспериментальных исследований СР показали, что описанную конструкцию возможно применять в широком диапазоне перепадов давления, однако отклонение величины массового расхода от номинала превышает заданное поле допуска [1]. Снижение статизма характеристики массового

расхода возможно путем выбора наилучшей геометрии золотника, для чего требуется проведение численного моделирования режима работы СР.

Целью работы является тестирование возможностей свободно распространяемого пакета OpenFOAM по построению расчетной схемы для моделирования динамики стабилизатора расхода с помощью метода контрольного объема и подвижных сеток. В качестве основных задач были определены: выбор наиболее подходящих технологий динамических скользящих сеток [8]; построение расчетной схемы для сопряженной тестовой задачи FSI; расчет переходного режима движения золотника для прототипа СР; анализ нестационарных течений во внутренней полости агрегата; анализ устойчивости и производительности счета.

2. Постановка задачи и метод решения

Для ускорения методических расчетов с подвижной сеткой задача численного моделирования переходного режима работы СР рассматривается с учетом следующих допущений: РЖ считается несжимаемой вязкой средой, а ее течение считается ламинарным. Тепловые эффекты не рассматриваются. Учет сжимаемости и турбулентности течения жидкости, а также тепловых эффектов может быть выполнен средствами OpenFOAM в дальнейшем.

С целью отработки методики построения расчетной схемы со скользящими сетками модель СР была упрощена до осесимметричной. Четыре отверстия золотникового дросселя при этом фактически заменены кольцевым каналом. Также кольцевыми каналами заменены отверстия в золотнике. Таким образом, золотник является в данной схеме набором из четырех тел, движущихся как единое целое. В силу сделанного допущения в области течения штифт был заменен неподвижной сферой и из рассмотрения исключены геометрическая модель пружины, канавки лабиринтных уплотнений опор золотника, а также зазоры между золотником и гильзой. Считается, что золотник имеет одну поступательную степень свободы вдоль продольной оси агрегата. Действие пружины на золотник заменено линейной вязкоупругой восстанавливающей силой. Такие упрощения искажают реальную картину течения РЖ в СР, однако сохраняют все основные особенности расчетной схемы, которые должны учитываться при построении подвижной сетки, давая возможность проводить методические расчеты с малыми затратами ресурсов.

Математическая задача FSI включает две подзадачи: динамики РЖ и динамики золотника. Задача динамики РЖ включает уравнение неразрывности и уравнение сохранения импульса несжимаемой среды, записанные относительно неизвестных полей скоростей и давления с граничными и начальными условиями, рассмотренными ниже. Динамика золотника описывается уравнением его малых колебаний с учетом линейного демпфирования и действия нестационарной позиционной силы, вычисляемой путем интегрирования распределения давления РЖ по поверхности золотника и проецирования полученной силы на продольную ось агрегата. Нулевое

начальное положение золотника выбрано таким образом, что отверстие золотникового дросселя частично перекрыто. Начальная скорость золотника принята нулевой.

Численное моделирование производится методом контрольного объема в пакете OpenFOAM-extend-3.2 [6]. На этапе препроцессинга проводится генерация сетки, задание граничных условий и параметров расчета, получение начальных условий для динамического расчета.

Расчет переходного режима производится в течение заданного промежутка времени при известном постоянном перепаде давления. При этом золотник в процессе движения переходит от начального положения в положение равновесия, обусловленное равенством гидродинамической силы и силы упругости пружины. На этапе постпроцессинга исследуются параметры движения золотника и течения РЖ.

2.1 Подготовка расчетной сетки

В пакете OpenFOAM для работы с динамической сеткой имеется несколько классов. Объект класса dynamicTopoFvMesh позволяет изменять топологию сетки при больших деформациях методом mesquiteMotionSolver (аналог метода remeshing в ANSYS). Родительский класс dynamicFvMesh, позволяет производить добавление и удаление слоев сетки методом layerAdditionRemoval (аналог метода layering в ANSYS), а также деформацию сетки методом displacementLaplacian (аналог метода mesh smoothing в ANSYS). Для описания перемещений золотника использован последний из указанных методов. Однако из-за сильного вырождения ячеек прямое использование деформируемой сетки в зоне перекрытия отверстия золотникового дросселя невозможно. Решением этой проблемы стало разделение расчетной области на две подобласти – неподвижную (область течения между штуцером входа и гильзой) и подвижную (области течения во внутренней полости золотника, внутри гильзы и в выходном штуцере), – взаимодействующих в зоне отверстия с помощью скользящего интерфейса. Хотя объект класса dynamicFvMesh имеет встроенные интерфейсы для скользящих сеток slidingInterface и attachDetach of boundaries, в данной расчетной схеме был использован обобщенный интерфейс GGI, описанный ниже.

Подготовка сетки в OpenFOAM производится в несколько этапов, как показано на рис. 3. На первом этапе для каждого из доменов по отдельности при помощи утилит blockMesh и snappyHexMesh строится пространственная расчетная сетка с использованием фоновой блочной сетки расчетной области и электронной геометрической модели обтекаемого тела в формате стереолитографии (*.stl). Для повышения качества сетки в области со сложной формой границы используется утилита surfaceFeatureExtract, позволяющая выделить особенности границы в отдельный файл. На втором этапе производится объединение расчетных доменов в один с помощью утилиты

mergeMeshes. Для получения сетки осесимметричной задачи на третьем этапе проводится вращение передней грани трехмерной стеки вокруг продольной оси агрегата на 1°, в результате чего получается сетка в виде сектора цилиндра толщиной в одну ячейку, как показано на рис. 4. На четвертом шаге при помощи утилит autoPatch и createPatch сетка разбивается на участки для задания граничных условий.

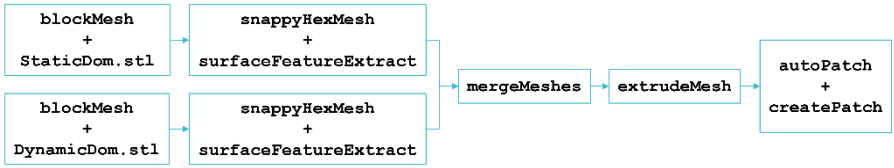


Рис. 3. Этапы создания расчетной сетки в OpenFOAM

Fig. 3. OpenFOAM mesh creation pipeline

Полученная сетка, показанная на рис. 4, имеет четыре уровня сгущения относительно базовой. Лучше всего разрешена область вокруг отверстия золотникового дросселя и вблизи поверхности обтекаемого тела. Более крупные ячейки находятся во внутренней полости золотника. Самая грубая сетка - в зонах, где отсутствуют подвижные элементы: в полостях штуцера входа и выхода и в зоне расположения пружины.

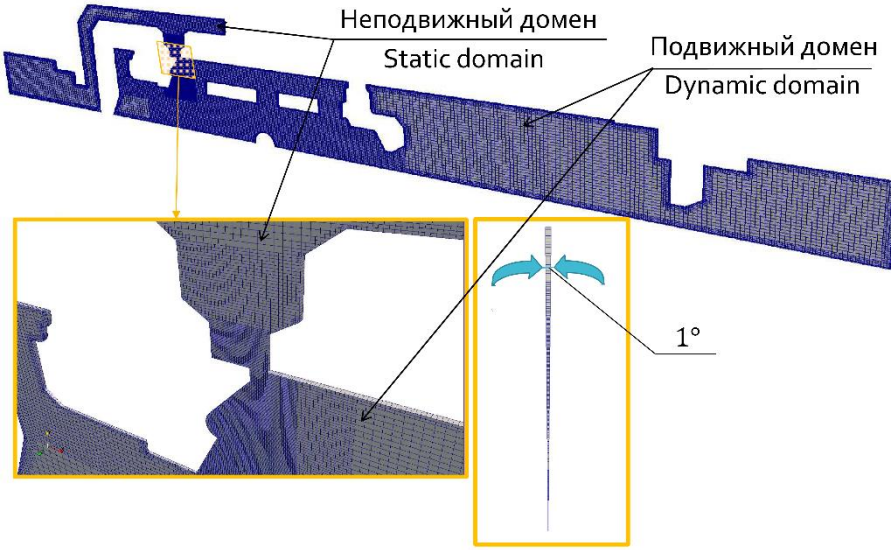


Рис. 4. Двухмерная расчетная сетка осесимметричной задачи

Fig. 4. 2-dimensional axisymmetric mesh

2.2 Подготовка начальных условий

Для получения начальных полей давления и скорости РЖ перед расчетом переходного режима выполняется стационарный расчет с неподвижным золотником в заданном начальном положении. При этом в качестве граничных условий на всех поверхностях кроме входа, выхода и боковых граней задано условие жесткой непроницаемой стенки (для давления – zeroGradient, для скорости – нулевое значение). Для боковых граней задано условие циклической симметрии (wedge). На входе в СР задано полное давление 22,6 МПа, а на выходе – атмосферное давление (0,1 МПа). Таким образом перепад давления на агрегате составил 22,5 МПа.

Вычисления проводятся в ламинарной постановке с помощью решателя simpleFoam для несжимаемого течения, использующего алгоритм SIMPLE [9].

Параметры РЖ соответствуют авиационному маслу с плотностью 860 кг/м^3 и кинематической вязкостью 21,5·10^{-6} м^2/с. Пример распределений модуля скорости и давления представлен на рис. 5. Видно, что в течении вблизи отверстия золотникового дросселя имеют место струйные эффекты.

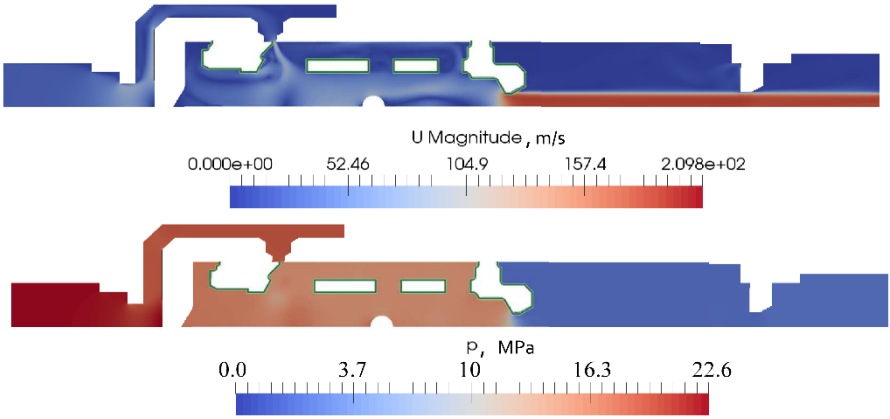


Рис. 5. Полученные поля давлений (снизу) и скоростей (сверху) при стационарном расчете. Зеленым цветом выделены поверхности золотника, по которым интегрировалась гидродинамическая сила

Fig. 5. Pressure (bottom) and velocity (top) fields at steady-state solution. Green lines – plunger surfaces where hydrodynamic force was calculated

График проекции на продольную ось суммарной гидродинамической силы, действующей на золотник, приведенный на рис. 6, показывает, что наличие струй нарушает монотонную сходимость стационарного решения: гидродинамическая сила пульсирует с периодом около 55 итераций, что затрудняет получение начальных условий. В качестве стационарного решения принимались осредненные по нескольким периодам значения рассчитываемых

полей. На основе среднего значения гидродинамической силы был проведен анализ сеточной сходимости, результаты которого показаны на рис. 7. В результате анализа была выбрана сетка из 138 000 ячеек, на которой в дальнейшем проводились расчеты нестационарной задачи.

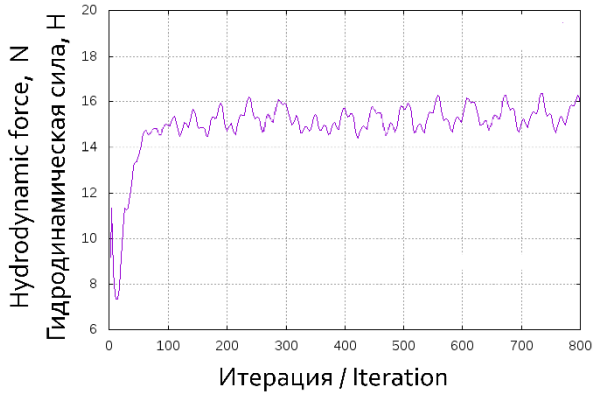


Рис. 6. График гидродинамической силы, действующей на золотник

Fig. 6. Plunger hydrodynamic force

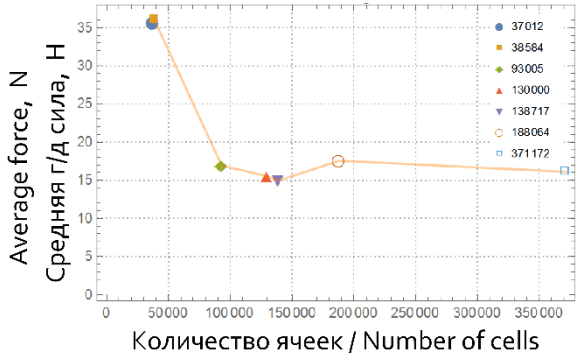


Рис. 7. График сходимости значения гидродинамической силы по сетке
Fig. 7. Mesh convergence

2.3 Подготовка граничных условий

Расчет переходного режима производится с помощью нестационарного решателя pimpleDyMFoam для несжимаемого течения, использующего гибридный алгоритм PISO-SIMPLE и возможность движения сетки [10]. На каждом шаге по времени делается около 20 итераций по давлению и 1-2 итерации по скорости. Основные трудности при построении расчетной схемы задачи связаны с заданием граничных условий на подвижном золотнике с

учетом действия суммарной гидродинамической силы, а также моделирование перекрытия отверстия золотникового дросселя.

Положения подвижных и неподвижных границ течения показаны на рис. 8, а описание соответствующих им граничных условий представлено в табл. 1. Для течения несжимаемой жидкости значение давления задается отнесенным к плотности – кинематическое давление. На участке channel_inlet, где входная скорость потока неизвестна, а известно только значение давления, задана комбинация граничных условий totalPressure и pressureInletVelocity. В качестве параметра условия totalPressure задается значение полного давления. Граничные условия pressureInletVelocity и pressureInletOutletVelocity предназначены для входных/выходных отверстий, где определено значение давления. Условие movingWallVelocity задает условие прилипания на движущейся стенке.

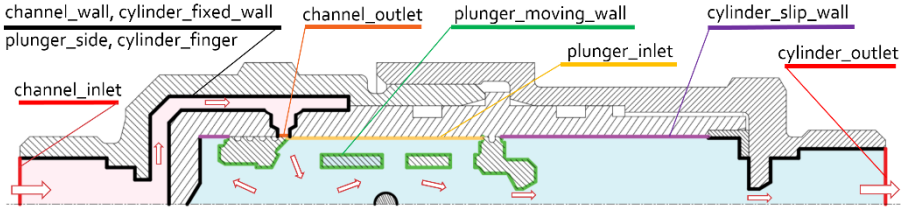


Рис. 8. Схема используемых названий участков граничных поверхностей при расчете.

Fig. 8. The scheme of using names of boundary surfaces patches.

Табл. 1. Задание граничных условий OpenFOAM

Table 1. Boundary conditions in OpenFOAM

Название участка	Тип	P	U	pointDisplacement
channel_inlet	patch	totalPressure (26227)	pressureInletVelocity	fixedValue (0 0 0)
channel_outlet	ggi	ggi	ggi	fixedValue (0 0 0)
plunger_inlet	ggi	ggi	ggi;	slip
plunger_moving_wall	wall	zeroGradient	movingWallVelocity (0 0 0)	sixDoFRigidBody
cylinder_slip_wall	wall	zeroGradient	fixedValue (0 0 0)	slip
channel_wall, cylinder_fixed_wall, plunger_side, cylinder_finger	wall	zeroGradient	fixedValue (0 0 0)	fixedValue (0 0 0)
cylinder_outlet	patch	fixedValue (1163)	pressureInletOutletVelocity;	fixedValue (0 0 0)
side1, side 2	wedge	wedge	wedge	wedge

В расчетной схеме отслеживание подвижных границ золотника (plunger_moving_wall на рис. 8) реализовано с помощью объекта класса dynamicFvMesh [11, 12]. Перестроение сетки производится на основании

перемещения граничных поверхностей подвижных тел. Модель расчета перемещения узлов сетки displacementLaplacian широко применяется для задач FSI в OpenFOAM, поскольку она не изменяет топологию сетки. Однако при больших перемещениях тела возможна потеря качества сетки и вырождение ее ячеек [13]. Используемая модель диффузии – directional. Величина смещения узлов сетки рассчитывается на основании решения уравнения Лапласа.

$$\nabla \cdot \left(\gamma \nabla \vec{d}_m\right) = \vec{0}$$

где $\vec{d}_m$ – вектор перемещения узлов, м; γ – коэффициент «диффузии» узлов сетки при движении. Смещение границ производится с использованием объекта класса sixDoFRigidBodyDisplacement (движение границ как абсолютно жесткого тела) с учетом различных механических связей, а также поверхностных гидродинамических сил [14].

Участок сетки, на котором происходит перекрытие отверстия при движении золотника, моделируется в версии OpenFOAM-extend с помощью технологии Generalized Grid Interface (GGI) [8, 15]. GGI позволяет установить связь между двумя смежными областями сетки, ячейки которых не совпадают на границе, в том числе и для сеток в зоне отверстия, где одна область сетки является неподвижной, а другая – подвижной (неподвижная и подвижная подобласти на рис 4), скользящей по границе первой. Сопряжение между поверхностями channel_outlet и plunger_inlet в зоне контакта (рис. 8) определяется набором перекрывающихся (где протекает поток) и неперекрывающихся (где задано условие стенки) ячеек как показано на рис. 9.

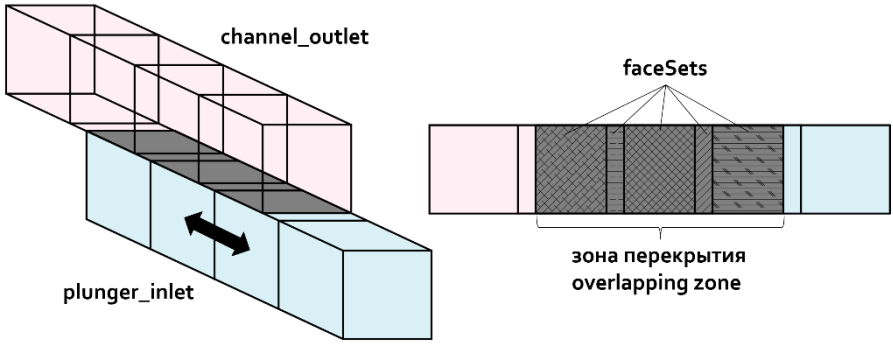


Рис. 9. Принцип работы GGI

Fig. 9. GGI principle of work

Для каждого временного шага зона перекрытия разбивается на участки (faceSets) между пересечениями ячеек двух смежных доменов. Через эти участки передаются рассчитываемые параметры потока. Для каждого участка оценивается его весовой коэффициент по отношению к содержащей его

ячейке в каждом из смежных доменов (зоны inlet и outlet) и вводятся условия совместности:

$$\phi_i^I = \sum_{k=1}^{n} W_k^{O\to I}\phi_k^O, \qquad \phi_j^O = \sum_{k=1}^{m} W_k^{I\to O}\phi_k^I;$$

$$\sum_{k=1}^{n} W_k^{O\to I} = 1, \qquad \sum_{k=1}^{m} W_k^{I\to O} = 1;$$

$$W_k^{O\to I} = \frac{S_{OI}}{S_k^O} \in [0;1], \qquad W_k^{I\to O} = \frac{S_{OI}}{S_k^I} \in [0;1].$$

где ϕ – численный поток; W_k – веса; i – номер ячейки inlet; j – номер ячейки outlet; m – число ячеек inlet соприкасающихся с j-й ячейкой; n – число ячеек outlet соприкасающихся с i-й ячейкой; S_k^O– площадь k ячейки зоны outlet; S_k^I – площадь k ячейки зоны inlet; S_{OI} – площадь зоны пересечения outlet и inlet.

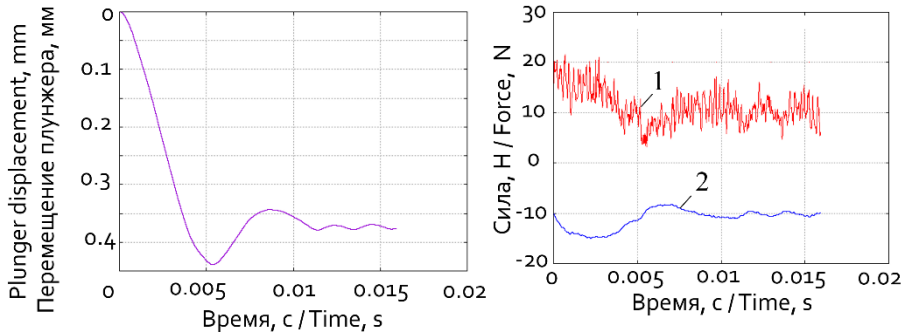
3. Результаты расчета

Расчет переходного режима проводился для перепада давления 22,5 МПа. Рассматривался золотник массой (с учетом приведенной массы пружины) 0,046 кг. Свободная длина пружины выбрана равной 0,060 м. Параметры вязкоупругой восстанавливающей силы, моделирующей пружину: жесткость – 400 Н/м, коэффициент демпфирования - 40 Н·с/м. Собственная частота колебаний данной колебательной системы в вакууме (без демпфирования) составила 14,8 Гц (период колебаний 0,068 с). Демпфирование обуславливает апериодическое свободное движение золотника в вакууме. В начальный момент времени золотник расположен так, что отверстие дросселя перекрыто на 30 % (0,0006 м), длина пружины при этом равна – 0,035 м.

В результате расчета были получены картины нестационарного течения, а также характеристики гидродинамического нагружения золотника. На рис. 10 представлены зависимости от времени перемещений золотника, а также обобщенных гидродинамической и восстанавливающей сил, действующих на золотник. Движение золотника включает переход в положение равновесия - затухающие колебания с периодом около 0,0055 с (частота около 180 Гц) и вибрацию вблизи положения равновесия с периодом около 0.0025 с (частота около 400 Гц), вызываемые нестационарностью течения в перекрытом дросселе. Максимальное смещение золотника составило 0,44 мм, при ширине всего отверстия 2 мм. В момент окончания расчета (0.016 с) отверстие дросселя перекрыто на 11 % (0,0002 м), а длина пружины составляет 0,0346 мм. В процессе перекрытия отверстия скорость в струе возрастает, и она притягивается к наклонной поверхности золотника как показано на рис. 11.

На рис. 12 представлены характерные фазы течения, показывающие изменение модуля скорости в конце переходного режима за период времени

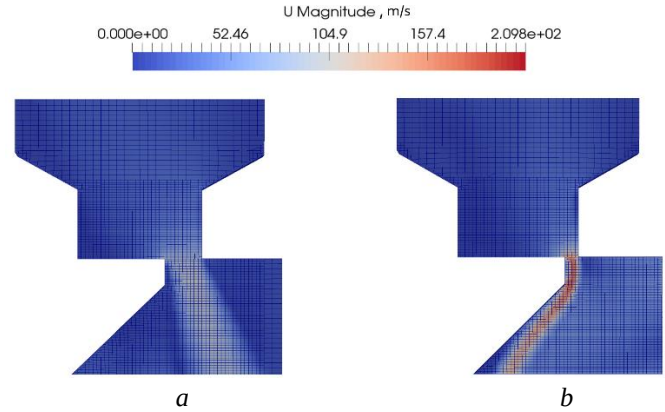
порядка 0,001 с. Видно, что вблизи наклонной кромки золотника струйное течение эволюционирует, порождая вихревые структуры в проточной части золотника. Обнаруженные струйные эффекты и вихревые течения могут возникать и при пространственном течении в проточной части реального агрегата, приводя к существенным изменениям распределения давления по поверхности золотника, возникновению дисбаланса радиальной составляющей гидродинамических сил и могут быть причиной различия расчетных и экспериментальных характеристик СР.



a *b*

Рис. 10. Результаты расчета переходного режима: a – перемещение золотника; b – обобщенные силы, действующие на золотник: 1 – гидродинамическая сила, 2 – восстанавливающая сила

Fig. 10. Transient solution graphs: a – plunger displacement; b – forces applied on plunger: 1 – hydrodynamic forces, 2 – restoring force



a *b*

Рис. 11. Поле скоростей вблизи отверстия золотникового дросселя в начальный (a) и конечный моменты времени (b)

Fig. 11. Velocity field in zone of the piston throttle hole at initial (a) and final (b) time steps

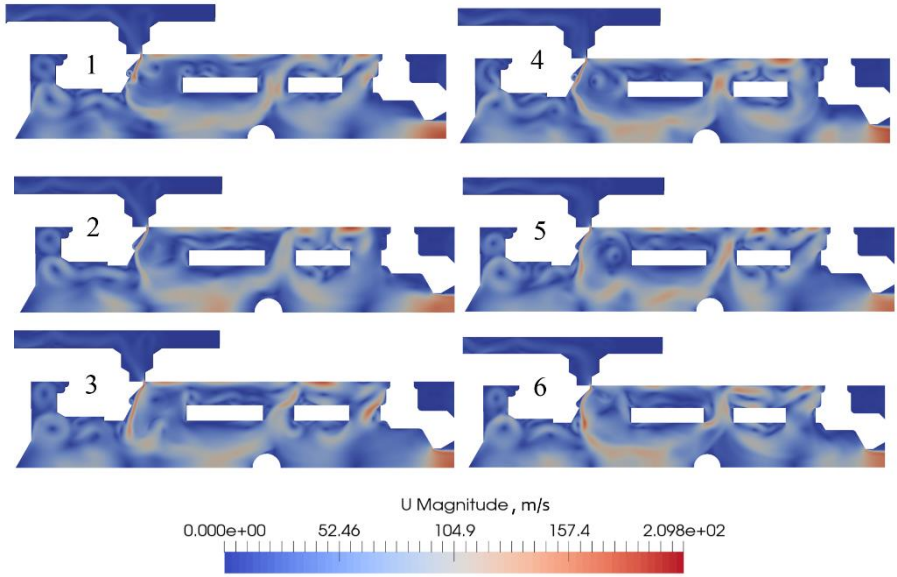


Рис. 12. Поле скоростей в золотнике в разные моменты времени:
1 – 0.012 с, 2 – 0.0122 с, 3 – 0.0123 с, 4 – 0.0126 с, 5 – 0.0127 с, 4 – 0.0129 с.

Fig. 12. Velocity field at time moments: 1 – 0.012 s, 2 – 0.0122 s, 3 – 0.0123 s, 4 – 0.0126 s, 5 – 0.0127 s, 4 – 0.0129 s.

Для решения системы уравнений для давления применяется метод GAMG (Geometric agglomerated algebraic multigrid solver). Для сглаживания используется метод Гаусса – Зейделя (предсглаживающих итераций не делалось, постсглаживающих итераций – 2). Система уравнений для прогнозов скоростей решается методом BiCGStab (Biconjugate gradient stabilized method) с предобуславливателем, основанным на DiLU-факторизации. Итерации выполнялись до достижения точности 10^{-7}. Шаг по времени выбирался автоматически из условия $Co_{max} < 2$. Характерная величина шага по времени – 10^{-7} с. Время расчета переходного режима длительностью 0,016 с составило 128 ч на 1 ядре (Intel(R) Xeon(R) CPU X5670, 2.93GHz). Для проведения расчетов был использован открытый облачный сервис UNIHUB разработки ИСП РАН [16].

4. Заключение

Проделанная работа позволяет сделать вывод о том, что пакет с открытым исходным кодом OpenFOAM в версии extend может быть использован в качестве альтернативы коммерческим пакетам программ вычислительной гидродинамики (CFD) для расчета переходных режимов в агрегатах гидравлической автоматики. В данном пакете имеются все необходимые

средства для построения расчетных схем с деформируемыми сетками, описывающими движение золотников и перекрытие отверстий золотниковых дросселей. Подобные расчетные схемы также можно использовать при моделировании в OpenFOAM двигателей внутреннего сгорания, гидравлических дросселей и других элементов пневмогидросистем, имеющих в конструкции подвижные элементы: поршни, клапаны и пр.

Расчеты модельной задачи показали, что наличие струй вблизи наклонной кромки золотника нарушает монотонную сходимость стационарного решения при вычислении начальных условий и существенно влияет на динамику движения золотника. В целом можно сделать вывод, что технология построения расчетной схемы успешно отлажена в двумерной постановке и готова к переносу на трехмерный случай. Дальнейшая работа предполагает полноценное моделирование реальной конструкции СР в условиях, приближенных к экспериментальному исследованию [1]. Сравнение результатов расчета с экспериментом позволит произвести верификацию расчетной схемы и использовать ее в дальнейшем для оптимизации внутренней геометрии СР с целью уменьшения статизма его расходной характеристики.

Работа поддержана грантом РФФИ (проект № 17-08-01468 А).

Список литературы

[1]. Копков Г. А., Кучин А. П., Новиков А. Е., Иванов М. Ю., Реш Г. Ф., Антонов Д. С. Стабилизаторы расхода для синхронизации перемещения исполнительных органов систем летательных аппаратов. Научно-технический юбилейный сборник. КБ химавтоматики: В 3 томах. Под ред. В. С. Рачука. Воронеж: Кварта, 2012. Т.1. , стр. 219-223.

[2]. Салман М. И., Попов Д. Н. Компьютерное исследование и расчет гидродинамических нагрузок на золотник. Наука и образование. МГТУ им. Н. Э. Баумана. Электрон. журн. 2012. № 10. С. 79-92. DOI: 10.7463/1112.0491484. Режим доступа: http://technomag.bmstu.ru/doc/491484.html (дата обращения: 02.01.2017).

[3]. Широкова К. А., Целищев В. А., Целищев Д. В., Галлямов Ш. Р. Численное моделирование потоков в струйно-золотниковом гидроусилителе. Вестник УГАТУ. 2008. Т.11, №2 (29). С. 55-59.

[4]. ANSYS Fluent: CFD Simulation: сайт. Режим доступа: http://www.ansys.com/products/fluids/ansys-fluent (дата обращения: 29.12.2016).

[5]. ANSYS CFX: Turbomachinery CFD Simulation: сайт. Режим доступа: http://www.ansys.com/products/fluids/ansys-cfx (дата обращения: 29.12.2016).

[6]. OpenFOAM. Free CFD Software: сайт. Режим доступа: http://openfoam.org/ (дата обращения: 29.12.2016).

[7]. Регулятор расхода: пат. 2548613 Рос. Федерация: МПК G05D 7/01 (2006.01). Дергачев А. А., Иванов М. Ю., Копков Г. А., Кучин А. П., Новиков А. Е., Реш Г. Ф., Синявин В.Г.; заявитель и патентообладатель: Открытое акционерное общество «Военно-промышленная корпорация «Научно-производственное объединение машиностроения»– № 2014102669/28; заявл. 29.01.14; опубл. 20.04.15, Бюл. № 11.

[8]. Beaudoin M., Jasak H. Development of a Generalized Grid Interface for Turbomachinery simulations with OpenFOAM. Open Source CFD International Conference 2008, Berlin, Germany.
[9]. Ferziger J.H., Peric M. Computational Methods for Fluid Dynamics, Springer, 3rd Ed., 2001. 431 p.
[10]. OpenFOAM guide. The PIMPLE algorithm in OpenFOAM – OpenFOAMWiki: сайт. Режим доступа: http://openfoamwiki.net/index.php/OpenFOAM_guide/The_PIMPLE_algorithm_in_OpenFOAM (дата обращения: 29.12.2016).
[11]. OpenFOAM dynamicFvMesh. C++ Source Code Guide. Режим доступа: http://www.openfoam.com/documentation/cpp-guide/html/a05056.html (дата обращения: 29.12.2016).
[12]. Gonzlez A.O. Mesh motion alternatives in OpenFOAM. Tech. rep.. Institution of Applied Mechanics at Chalmers Technical University, Gteborg, Sweden, 2009. http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/AndreuOliverGonzalez/ProjectReport Corrected.pdf (дата обращения: 29.12.2016).
[13]. Kotsur O., Scheglov G., Leyland P. Verification of Modelling of Fluid-structure Interaction (FSI) Problems Based on Experimental Research of Bluff Body Oscillations in Fluids. 29th Congress of the International Council of the Aeronautical Sciences, 7-12 September 2014, Saint-Petersburg, Russia. Paper ICAS2014-2.6.3-2014_0953.
[14]. OpenFOAM sixDoFRigidBodyMotion. C++ Source Code Guide. Режим доступа: http://cpp.openfoam.org/v3/a09731.html (дата обращения: 29.12.2016).
[15]. Jasak H. NUMAP-FOAM Summer School, Zagreb 2-15 Sep 2009 General Grid Interface Theoretical Basis and Implementation. Режим доступа: http://powerlab.fsb.hr/ped/kturbo/OpenFOAM/SummerSchool2009/lectures/TurboGGI.pdf (дата обращения: 03.01.2017).
[16]. UniHUB: сайт. Режим доступа: http://www.unicluster.ru/unihub.html (дата обращения: 29.12.2016).

Numerical simulation of the flow rate regulator valve using OpenFOAM

V.G. Melnikova <vg-melnikova@yandex.ru>
O.S. Kotsur <oskotsur@gmail.com>
G.A. Shcheglov <shcheglov_ga@bmstu.ru>
Federal state budgetary institution of higher professional education «Bauman Moscow State Technical University (National research university of technology) », ul. Baumanskaya 2-ya, 5/1, Moscow, 105005, Russia

Abstract. The results of methodical investigation, aimed on testing of the performances of free opensource CFD toolbox OpenFOAM in the field of simulation of hydraulic units' dynamics using Finite Volume Method and dynamic meshes are presented. The following key features are reviewed: the choice of appropriate algorithms managing dynamic sliding meshes; design of the model case for the FSI-problem of interaction between a moving regulating element (plunger) and power fluid; plunger dynamics simulation for the prototype of the flow rate regulator of new design; transient analysis of regulator's internal flow; analysis of stability and computational efficiency. As the example simplified axisymmetric

regulator model with incompressible power fluid is considered. The main steps of model case preparation are described. The model case is designed to simulate the problem of plunger equilibration under hydrodynamic forces and spring reaction. Results are given also for a preliminary steady-state simulation with fixed plunger, they have been used as initial conditions. Detailed description is given for the methods of mesh motion simulation, which follows plunger, as well as technology of sliding meshes (GGI), used for the piston throttle hole overlapping. Results are given for transient simulation: velocity, pressure fields, forces graphs, acting on the plunger, its displacement over time. The mechanism of jet streams and vertical flows creation in regulator's flow channel is described. Methodical investigation, held in this paper, confirms that OpenFOAM in the "extend" version can be successfully used as an alternative for commercial CFD codes, as it contains all necessary tools to create and simulate cases incorporating dynamic meshes. It provides means for simulation of transient problems of hydraulic units which have moving parts.

Keywords: CFD; numerical simulation; OpenFOAM; open source CFD codes; flow rate regulator.

DOI: 10.15514/ISPRAS-2017-29(1)-4

For citation: Melnikova V.G., Kotsur O.S., Shcheglov G.A. Numerical simulation of the flow rate regulator valve using OpenFOAM. *Trudy ISP RAN/Proc. ISP* RAS, vol. 29, issue 1, 2017, pp. 53-70 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-4

References

[1]. Kopkov G.A., Kuchin A.P., Novikov A.E., Ivanov M.Ju., Resh G.F., Antonov D.S. Flow regulators for synchronization of executive devices of aircraft systems. *Nauchno-tehnicheskij jubilejnyj sbornik. KB himavtomatiki* [Anniversary proc. DD Chemical automatics]. 2012, vol.1, pp. 219-223 (in Russian).

[2]. Salman M.I., Popov D.N. Computer study and calculation of hydrodynamic loads on the valve. *Nauka i obrazovanie. MGTU im. N.E. Baumana. Jelektron. zhurn* [Science and education. BMSTU. Electronic journal]. 2012, no. 10, pp. 79-92. (in Russian). DOI: 10.7463/1112.0491484. Available at: http://technomag.bmstu.ru/doc/491484.html (Accessed 2 January 2017)

[3]. Shirokova K.A., Celishhev V.A., Celishhev D.V., Galljamov Sh.R. Numerical simulation of the flows in hydraulic amplifier with a sliding valve. *Vestnik UGATU* [Proc. of USATU]. 2008, vol. 11, no. 2 (29), pp. 55-59 (in Russian).

[4]. ANSYS Fluent: CFD Simulation. Available at: http://www.ansys.com/products/fluids/ansys-fluent (Accessed 29 December 2016).

[5]. ANSYS CFX: Turbomachinery CFD Simulation. Available at: http://www.ansys.com/products/fluids/ansys-cfx (Accessed 29 December 2016).

[6]. OpenFOAM. Free CFD Software. Available at: http://openfoam.org/ (Accessed 29 December 2016).

[7]. Dergachev A. A., Ivanov M. Ju., Kopkov G. A., Kuchin A. P., Novikov A. E., Resh G. F., Sinjavin V. G. *Reguljator rashoda* [Flow rate regulator]. Patent RF, no. 2548613, 2006.

[8]. Beaudoin M., Jasak H. Development of a Generalized Grid Interface for Turbomachinery simulations with OpenFOAM. Open Source CFD International Conference 2008, Berlin, Germany.

[9]. Ferziger J. H., Peric M., Computational Methods for Fluid Dynamics, Springer, 3rd Ed., 2001.

[10]. OpenFOAM guide. The PIMPLE algorithm in OpenFOAM – OpenFOAMWiki. Available at: http://openfoamwiki.net/index.php/OpenFOAM_guide/The_PIMPLE_algorithm_in_OpenFOAM (Accessed 29 December 2016).

[11]. OpenFOAM dynamicFvMesh. C++ Source Code Guide. Available at: http://www.openfoam.com/documentation/cpp-guide/html/a05056.html (Accessed 29 December 2016).

[12]. Gonzlez A.O. Mesh motion alternatives in OpenFOAM, tech. rep., Institution of Applied Mechanics at Chalmers Technical University, Gteborg, Sweden, 2009. Available at: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2009/AndreuOliverGonzalez/ProjectReport_Corrected.pdf (Accessed 29 December 2016).

[13]. Kotsur O., Scheglov G., Leyland P. Verification of Modelling of Fluid-structure Interaction (FSI) Problems Based on Experimental Research of Bluff Body Oscillations in Fluids. 29th Congress of the International Council of the Aeronautical Sciences, 7-12 September 2014, Saint-Petersburg, Russia. Paper ICAS2014-2.6.3-2014_0953.

[14]. OpenFOAM sixDoFRigidBodyMotion. C++ Source Code Guide. Available at: http://cpp.openfoam.org/v3/a09731.html (Accessed 29 December 2016).

[15]. Jasak H. NUMAP-FOAM Summer School, Zagreb 2-15 Sep 2009 General Grid Interface Theoretical Basis and Implementation. Available at: http://powerlab.fsb.hr/ped/kturbo/OpenFOAM/SummerSchool2009/lectures/TurboGGI.pdf (Accessed 03 January 2017).

[16]. UniHUB. Available at: http://www.unicluster.ru/unihub.html (Accessed 29 December 2016).

Модификация метода погруженных границ LS-STAG для моделирования течений вязкоупругих жидкостей

В.В. Пузикова <valeria.puzikova@gmail.com>
МГТУ им. Н.Э. Баумана,
105005, Россия, г. Москва, ул. 2-я Бауманская, дом 5

Аннотация. Представлена авторская модификация метода погруженных границ LS-STAG для моделирования течений вязкоупругих жидкостей, описываемых линейными и квазилинейными моделями скоростного типа (моделями Максвелла, Джеффри, Джонсона – Сигельмана, Максвелла-А, Олдройда-Б, Олдройда-А, верхней конвективной моделью Максвелла). Построены дискретные аналоги различных конвективных производных (Олдройда, Коттера – Ривлина, Яумана – Зарембы – Нолла). К трем разнесенным сеткам базового метода LS-STAG добавлена четвертая сетка, ячейки которой являются контрольными объемами для дискретизации уравнения для расчета касательных неньютоновских вязкоупругих напряжений. Нормальные неньютоновские напряжения вычисляются в центрах ячеек основной сетки, а касательные – в углах ячеек данной сетки. Интегрирование по времени получающейся после LS-STAG-дискретизации по пространству дифференциально-алгебраической системы производится при помощи метода, основанного на схеме предиктор-корректор первого порядка. Этот метод состоит из двух шагов. Шаг предиктора приводит к решению разностного аналога уравнения Гельмгольца для прогноза скорости, а шаг корректора – к решению разностного аналога уравнения Пуассона для поправки давления. После этого решаются уравнения для упругой составляющей тензора неньютоновских напряжений. Для верификации метода использовались различные модельные задачи, в частности, моделирование течения Пуазейля. Полученная модификация метода LS-STAG реализована в разрабатываемом автором программном комплексе, позволяющем проводить моделирование течений вязкой несжимаемой среды. Комплекс позволяет моделировать обтекание движущихся профилей произвольной формы и систем из любого числа профилей, имеющих одну или две степени свободы, в т.ч. турблулентным потоком. При решении тестовых задач метод обеспечивает второй порядок точности.

Ключевые слова: несжимаемая среда; вязкоупругие жидкости; модели вязкоупругих жидкостей скоростного типа; метод погруженных границ; метод LS-STAG

DOI: 10.15514/ISPRAS-2017-29(1)-5

Для цитирования: Пузикова В.В. Модификация метода погруженных границ LS-STAG для моделирования течений вязкоупругих жидкостей. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 71-84. DOI: 10.15514/ISPRAS-2017-29(1)-5

1. Введение

Существует широкий класс разнообразных сред, для которых закон вязкого трения Ньютона не выполняется. Такие жидкости называются неньютоновскими [1]. Вязкость таких жидкостей зависит от скорости течения, второго инварианта тензора скоростей, времени. Примерами неньютоновских жидкостей являются суспензии, эмульсии, цельная кровь, нефти, мазуты, глинистые и цементные растворы, растворы полимеров и т.д. Традиционно неньютоновские жидкости делятся на три основные группы: обобщенные ньютоновские жидкости, неньютоновские нереостабильные жидкости и неньютоновские вязкоупругие жидкости [2].

В данной работе будут рассмотрены вязкоупругие жидкости скоростного типа. Помимо числа Рейнольдса Re такие жидкости характеризуются числом Вайсенберга $\mathrm{We} = \dfrac{\lambda \cdot V_\infty}{D}$ (λ – время релаксации; V_∞ – характерная скорость; D – характерный размер), которое отражает упругие свойства вязкоупругого материала. Очень серьезным вопросом при численном моделировании течений неньютоновских вязкоупругих жидкостей, описываемых моделями скоростного типа, является отсутствие сходимости численных методов для высокоупругих течений – так называемая проблема большого числа Вайсенберга (*high Weissenberg number problem, HWNP*). Значение $\mathrm{We}_{\mathrm{crit}}$, начиная с которого численный метод перестает сходиться, сильно зависит от способа дискретизации уравнений. При этом измельчение сетки не позволяет повысить значение $\mathrm{We}_{\mathrm{crit}}$. Более того, наблюдается парадоксальная ситуация: измельчение сетки приводит к уменьшению значения $\mathrm{We}_{\mathrm{crit}}$ [3].

В последние годы в вычислительной гидродинамике возрастает популярность методов погруженных границ [4], в которых сетка не связана с границей тела и не изменяется на протяжении всего расчета, несмотря на движение погруженных границ. Данные методы предполагают использование прямоугольных сеток. При этом важно обеспечить высокую точность решения задачи в усеченных ячейках, через которые проходит погруженная граница.

К наиболее эффективным методам этого класса относят метод LS-STAG [5], в котором для представления погруженной границы используется аппарат функций уровня [6]. LS-STAG-дискретизация производится по одним и тем же формулам, как в прямоугольных ячейках, так и в усеченных, причем шаблон дискретизации имеет в двумерном случае пятиточечную структуру. К настоящему моменту построена LS-STAG-дискретизация двумерных уравнений Навье – Стокса для вязкой несжимаемой среды [5], разработаны модификации метода LS-STAG, позволяющие использовать модели турбулентности Смагоринского, Спаларта – Аллмараса, $k-\varepsilon$, $k-\omega$ и $k-\omega$ SST в рамках RANS, LES и DES подходов [7, 8], а также модификации для решения сопряженных задач гидроупругости [9]. Как и многие

«нестандартные» высокоточные методы, метод погруженных границ LS-STAG не реализован в широко распространенных пакетах вычислительной гидродинамики, поэтому весьма актуальной задачей является разработка эффективной программной реализации метода LS-STAG и его модификаций [10].

Целью данной работы является разработка модификации метода LS-STAG для расчета течений вязкоупругих жидкостей, описываемых моделями скоростного типа, и верификация полученного метода на модельных задачах.

2. Постановка задачи

Рассматривается течение вязкоупругой жидкости в прямоугольной расчетной области Ω (рис. 1). В области может находиться профиль характерного размера D (или система профилей) с границей K. Предполагается, что погруженные границы являются неподвижными, однако в дальнейшем полученные формулы могут быть обобщены на случай движущихся погруженных границ.

В безразмерных переменных математическая постановка задачи имеет вид:

$$
\begin{cases}
\nabla \cdot \vec{v} = 0, \quad \dfrac{\partial \vec{v}}{\partial t} + (\vec{v} \cdot \nabla)\vec{v} + \nabla p - \dfrac{1}{\mathrm{Re}} \Delta \vec{v} = \nabla \cdot \hat{\tau}^e, \\
\hat{\tau}^e + \lambda \tilde{D} \hat{\tau}^e = 2\nu_e (\hat{S} + \lambda_r \tilde{D} \hat{S}), \\
\vec{v}(x, y, 0) = \vec{v}_0(x, y), \; \hat{\tau}^e(x, y, 0) = \hat{\tau}^e_0(x, y), \quad (x, y) \in \Omega, \\
\vec{v}\,|_{\Gamma_2} = \vec{V}_{\text{inflow}} = \vec{V}_{\text{inflow}}(y), \; \vec{v}\,|_{\Gamma_1} = \vec{v}\,|_{\Gamma_3}, \; \vec{v}\,|_K = \vec{0}, \; \dfrac{\partial \vec{v}}{\partial \vec{n}}|_{\Gamma_4} = \vec{0}, \\
\hat{\tau}^e\,|_{\Gamma_1 \cup \Gamma_2 \cup \Gamma_3} = \hat{\tau}^{e,bc}(x, y, t), \; \hat{\tau}^e\,|_K = \hat{0}, \; \dfrac{\partial \hat{\tau}^e}{\partial \vec{n}}|_{\Gamma_4} = \hat{0}.
\end{cases}
$$

Здесь $\vec{n}$ – внешняя нормаль; t – безразмерное время; x, y – безразмерные координаты; Re – число Рейнольдса; p – безразмерное давление; $\vec{v} = \vec{v}(x, y, t) = u \cdot \vec{e}_x + v \cdot \vec{e}_y$ – безразмерная скорость; λ – время релаксации; λ_r – время запаздывания; ν_e – вязкость вязкоупругой составляющей; $\tilde{D}$ – дифференциальный оператор производной по времени (в зависимости от модели неньютоновской жидкости это может быть частная или одна из конвективных производных); $\hat{\tau}^e$ – вязкоупругая составляющая тензора напряжений [11]; $\hat{S} = \frac{1}{2}\left(\nabla \vec{v} + [\nabla \vec{v}]^T\right)$ – тензор скоростей деформации.

В зависимости от вида оператора $\tilde{D}$ и значения λ_r рассматриваемая жидкость описывается одной из следующих моделей скоростного типа:

Максвелла, Максвелла-А или верхней конвективной моделью Максвелла [12], Джеффри [13], Олдройда-Б или Олдройда-А [14], Джонсона – Сигельмана [15].

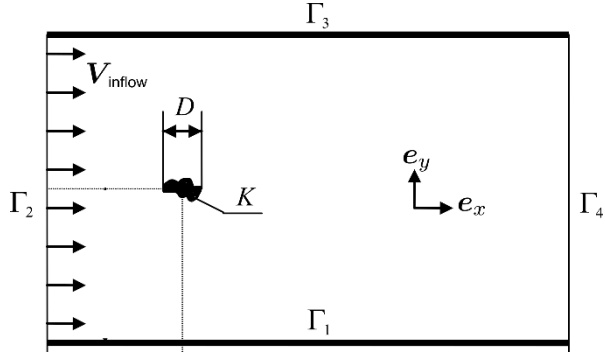


Рис. 1. Расчетная область Ω

Fig. 1. Computational domain Ω

3. Модификация метода LS-STAG

Прямоугольная расчётная область Ω делится на прямоугольные ячейки $\Omega_{i,j} = (x_{i-1}, x_i) \times (y_{j-1}, y_j)$ с площадями $V_{i,j} = \Delta x_i \Delta y_j$ и центрами $\vec{x}^c_{i,j} = (x^c_i, y^c_j)$. Ячейка $\Omega_{i,j}$ данной сетки, которая далее будет называться «основной», является контрольным объёмом, который используется для дискретизации уравнения неразрывности, а также уравнений для нормальных неньютоновских вязкоупругих напряжений. Ячейки смещенных *x*-сетки и *y*-сетки $\Omega^u_{i,j} = (x^c_i, x^c_{i+1}) \times (y_{j-1}, y_j)$ и $\Omega^v_{i,j} = (x_{i-1}, x_i) \times (y^c_j, y^c_{j+1})$ являются контрольными объёмами для дискретизации уравнений импульса в проекции на оси *Ox* и *Oy* соответственно (рис. 2). Также вводится дополнительная четвертая разнесенная разнесенная сетка, *xy*-сетка, ячейки которой $\Omega^{xy}_{i,j} = (x^c_i, x^c_{i+1}) \times (y^c_{j-1}, y^c_j)$ являются контрольными объемами для дискретизации уравнения для расчета касательных неньютоновских вязкоупругих напряжений.

В двумерном случае все усеченные ячейки можно разделить на три группы: трапециевидные, треугольные и пятиугольные. Положения точек, в которых вычисляются неизвестные величины, зависят от типа ячейки. Примеры всех типов ячеек представлены на рис. 3.

Интегрирование по времени получающейся после LS-STAG-дискретизации по пространству дифференциально-алгебраической системы производится при помощи метода, основанного на схеме предиктор-корректор первого порядка.

Этот метод состоит из двух шагов. Шаг предиктора приводит к решению разностного аналога уравнения Гельмгольца для прогноза скорости $\tilde{U}$ в момент времени $t_{n+1} = (n+1)\Delta t$:

$$\frac{M(\tilde{U} - U^n)}{\Delta t} + C[\overline{U}^n]U^n + S^{ib,c,n} - D^T(P^n - T^{e,n}_{norm}) - D^\tau T^{e,n}_{xy} - \nu K\tilde{U} - \nu S^{ib,\nu} = 0.$$

Здесь U^n – вектор, компонентами которого являются значения скоростей $u^n_{i,j}$ и $v^n_{i,j}$; P^n – вектор с компонентами $p^n_{i,j}$; $T^{e,n}_{norm}$ – вектор, компонентами которого являются значения нормальных вязкоупругих напряжений $\tau^e_{xx}|^n_{i,j}$ и $\tau^e_{yy}|^n_{i,j}$; $T^{e,n}_{xy}$ – вектор с компонентами $\tau^e_{xy}|^n_{i,j}$; M – диагональная матрица, элементы которой – площади ячеек $\Omega^u_{i,j}$ и $\Omega^v_{i,j}$; $C[\overline{U}]$ и K – матрицы, получаемые при LS-STAG-дискретизации конвективных и вязких потоков соответственно; $-D^T$ – матрица, задающая дискретный аналог оператора градиента; D^τ – матрица, задающая дискретный аналог оператора дивергенции на *xy*-сетке; $S^{ib,c,n}$ и $S^{ib,\nu}$ – источниковые члены, возникающие в силу граничных условий; $\nu = \frac{1}{\mathrm{Re}}$.

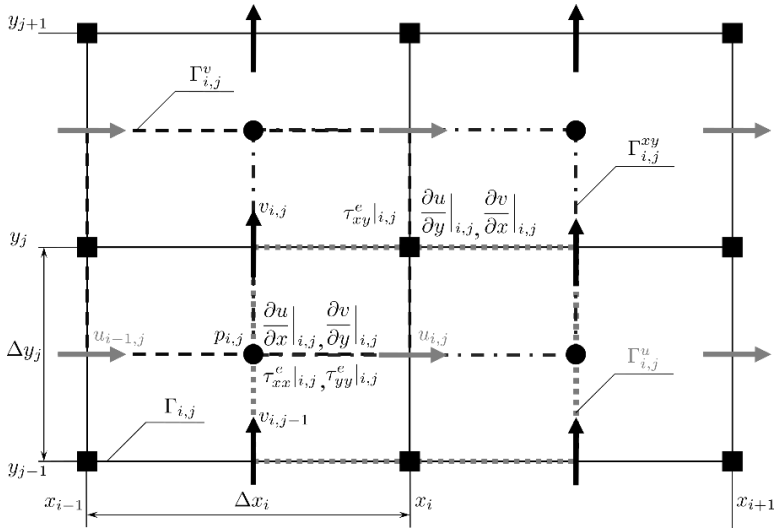


Рис. 2. Разнесенные сетки

Fig. 2. Staggered meshes

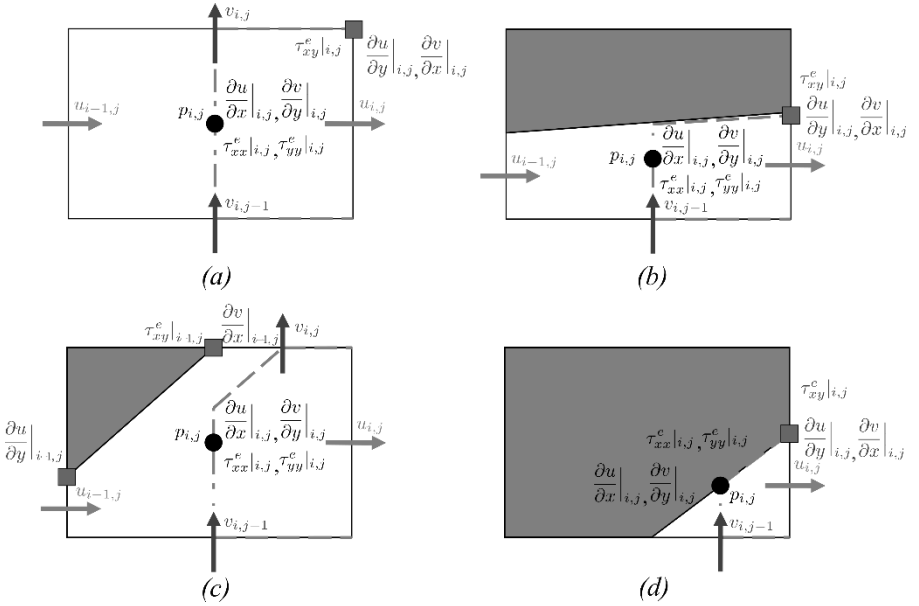


Рис. 3. Положение точек вычисления давления, скоростей и напряжений для основных типов ячеек LS-STAG-сетки: (a) прямоугольная «жидкая» ячейка; (b) усеченная северная трапециевидная ячейка; (c) усеченная северо-западная пятиугольная ячейка; (d) усеченная северо-западная треугольная ячейка

Fig. 3. Location of the variables discretization points on the LS-STAG mesh: (a) Cartesian Fluid Cell; (b) North Trapezoidal Cell; (c) Northwest Pentagonal Cell; (d) Northwest Triangle Cell

Шаг корректора приводит к решению разностного аналога уравнения Пуассона для функции давления $\Phi = \Delta t(P^{n+1} - P^n)$:

$$A\Phi = D\tilde{U} + \overline{U}^{ib,n+1},$$

где $A = -DM^{-1}D^T$. Затем определяются скорости и давление в момент времени t_{n+1}:

$$U^{n+1} = \tilde{U} + M^{-1}D^T\Phi,\ P^{n+1} = P^n + (\Phi/\Delta t).$$

После этого рассчитываются значения вязкоупругих напряжений $T^{e,n+1}$ в момент времени t_{n+1}:

$$M_e T^{e,n+1} + \lambda\frac{M_e(T^{e,n+1} - T^{e,n})}{\Delta t} = 2\nu_e M_e S^{n+1} + \frac{2\nu_e\lambda_r M_e(S^{n+1} - S^n)}{\Delta t} + CD(\overline{U}^{n+1}, X^n).$$

Здесь M_e – диагональная матрица, элементы которой – площади ячеек $\Omega_{i,j}$ и $\Omega_{i,j}^{xy}$; S – вектор, компонентами которого являются компоненты тензора скоростей деформации в соответствующих точках; $X^n = 2\nu_e\lambda_r S^{n+1} - \lambda T^{e,n}$; $CD(\overline{U}^{n+1}, X^n)$ – дискретный аналог конвективной производной X^n (без учета частной производной по времени). Отметим, что при построении LS-STAG-дискретизации конвективной производной учитывается, что уравнение для расчета неньютоновских напряжений в рассматриваемых моделях вязкоупругих жидкостей является уравнением гиперболического типа.

4. Программный комплекс «LS-STAG»

Созданная модификация метода LS-STAG для моделирования течений вязкоупругих жидкостей реализована в разрабатываемом автором программном комплексе. Программа написана на языке C++ и имеет объектно-ориентированную легко расширяемую структуру. Общая схема работы программного комплекса представлена в работе [10]. При проведении расчетов возможно использование таких технологий параллельного программирования, как Intel® Cilk™ Plus [16], Intel® TBB [17], OpenMP (реализация из Intel® Parallel Studio XE 2015, стандарт 4.0). Помимо моделирования течений вязкоупругих жидкостей комплекс позволяет моделировать обтекание профилей произвольной формы и их систем, в т.ч. движущихся и имеющих степени свободы. В расчетах возможно использовать модели турбулентности Смагоринского, Спаларта – Аллмараса, $k-\varepsilon$, $k-\omega$ и $k-\omega$ SST в рамках RANS, LES и DES подходов.

На рис. 4 представлена иерархия структур, реализующих работу с моделями неньютоновских жидкостей: хранение, инициализацию, загрузку, сохранение и вычисление неньютоновских напряжений. В базовой структуре `NonNewtonianModelInterface` хранятся указатели на структуру, в которой хранится дискретный аналог тензора неньютоновских напряжений `SymTensor* stress`, и структуру, которая реализует работу с дискретным аналогом тензора скоростей деформации `StrainRateTensor* strainRate`.

К настоящему моменту в программном комплексе реализованы структуры для работы с линейными и квазилинейными моделями скоростного типа. Основная часть работы по вычислению компонент дискретного аналога тензора вязкоупругих напряжений реализована в структуре `MaxwellFluid` (реализует модель Максвелла), унаследованной от структуры `LinearRateTypeViscoElasticModelInterface`. Учет времени запаздывания λ_r происходит в структуре `JeffreysFluid` (реализует модель Джеффри), унаследованной от структуры `MaxwellFluid`. Модели

Максвелла и Джеффри являются линейными – в качестве оператора $\tilde{D}$ в них выступает обычная частная производная по времени:

$$\tilde{D}X = \frac{\partial X}{\partial t}.$$

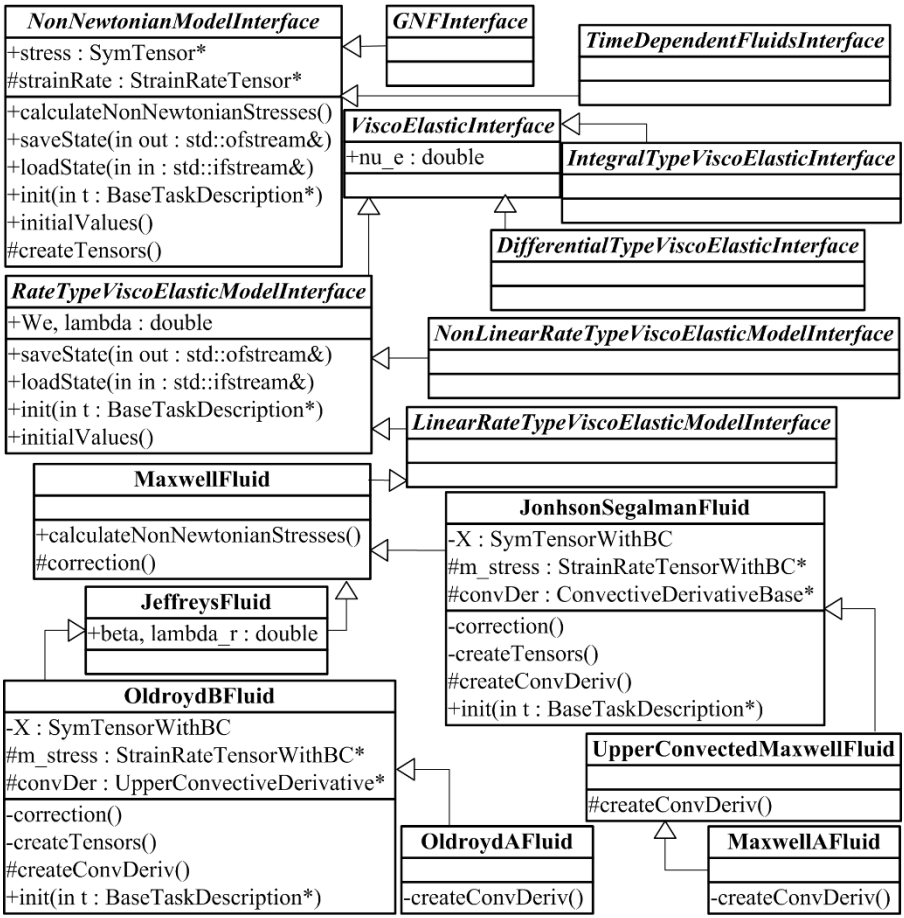


Рис. 4. Иерархия структур, реализующих работу с моделями неньютоновских жидкостей, в частности с линейными и квазилинейными моделями вязкоупругих жидкостей скоростного типа

Fig. 4. The hierarchy of structures, implementing the work with models of non-Newtonian fluids, such as linear and quasi-linear rate type models of viscoelastic fluids

Квазилинейные модели, в которых оператор $\tilde{D}$ является одной из конвективных производных, реализованы в структурах, представленных

в табл. 1. В этих структурах хранится указатель `convDer` на структуру, в которой реализована работа с конвективной производной, используемой в данной модели.

Табл. 1. Структуры, реализующие работу с квазилинейными моделями вязкоупругих жидкостей скоростного типа
Table 1. Structures, implementing the work with viscoelastic quasi-linear rate type models

Модель	Структура	Унаследована от	Конвективная производная
Джонсона – Сигельмана	`JonhsonSegalmanFluid`	`MaxwellFluid`	вращательная
Верхняя конвективная модель Максвелла	`UpperConvectedMaxwellFluid`	`JonhsonSegalmanFluid`	верхняя
Максвелла-А	`MaxwellAFluid`	`UpperConvectedMaxwellFluid`	нижняя
Олдройда-Б	`OldroydBFluid`	`JeffreysFluid`	верхняя
Олдройда-А	`OldroydAFluid`	`OldroydBFluid`	нижняя

В рассматриваемых моделях используются следующие виды конвективных производных [2]: верхняя (производная Олдройда):

$$\tilde{D}X = \overset{\nabla}{X} = \frac{\partial X}{\partial t} + (\vec{v}\cdot\nabla)X - X(\nabla\vec{v}) - (\nabla\vec{v})^T X \ ;$$

нижняя (производная Коттера – Ривлина):

$$\tilde{D}X = \overset{\Delta}{X} = \frac{\partial X}{\partial t} + (\vec{v}\cdot\nabla)X + X(\nabla\vec{v}) + (\nabla\vec{v})^T X \ ;$$

и вращательная (производная Яумана, или производная Яумана – Зарембы – Нолла):

$$\tilde{D}X = \overset{o}{X} = \frac{1}{2}\left(\overset{\nabla}{X} + \overset{\Delta}{X}\right).$$

5. Вычислительные эксперименты

В качестве примера приведем результаты численного моделирования установившегося течения Пуазейля, для которого известно точное решение [18, 19], в области $\Omega = [0;\ 20D]\times[0;\ 2D]$. Жидкость описывается моделью Олдройда-Б. Погруженное тело отсутствует. Функции, задающие

граничные условия в приведенной в начале статьи постановке задачи, в данном случае имеют следующий вид:

$$\vec{V}_{\text{inflow}}(y) = V_\infty y(2D-y)\vec{e}_x\,;\ \tau_{xx}^{e,bc} = 2\lambda\nu_e\left(\frac{\partial u}{\partial y}\right)^2;\ \tau_{yy}^{e,bc} = 0;\ \tau_{xy}^{e,bc} = \nu_e\frac{\partial u}{\partial y}.$$

Пусть характерный размер $D = 1$, характерная скорость $V_\infty = 1$. Тогда

$$\text{We} = \frac{\lambda\cdot V_\infty}{D} = \lambda,\ \text{Re} = \frac{V_\infty\cdot D}{\nu} = \frac{1}{\nu} = \frac{1}{\nu_s+\nu_e},\ \lambda_r = \beta\cdot\lambda = \frac{\nu_s\cdot\lambda}{\nu_s+\nu_e} = \frac{\nu_s\cdot\text{We}}{\text{Re}}.$$

Здесь ν_s – вязкость растворителя, который является ньютоновской жидкостью; β – доля вязкости растворителя. Будем рассматривать течение при $\text{Re} = 0{,}1$, $\beta = 0{,}1$. Этим значениям параметров соответствуют следующие значения вязкости растворителя и вязкоупругой составляющей:

$$\nu_s = \frac{\beta\cdot V_\infty\cdot D}{\text{Re}} = \frac{\beta}{\text{Re}} = 1,\ \nu_e = \frac{(1-\beta)\cdot V_\infty\cdot D}{\text{Re}} = \frac{1-\beta}{\text{Re}} = 9.$$

Будем рассматривать течения при значениях числа Вайсенберга в следующем диапазоне:

$$\text{We} = 0{,}1\ldots1{,}0.$$

Для оценки точности разработанной модификации метода LS-STAG задачу будем решать на равномерных сетках (табл. 2). Точное стационарное решение поставленной задачи имеет вид [18,19]

$$u = \frac{\partial p}{\partial x}\cdot\frac{y(2D-y)}{2\nu};\ v = 0;\ \tau_{xx}^{e} = 2\lambda\nu_e\left(\frac{\partial u}{\partial y}\right)^2;\ \tau_{yy}^{e} = 0;\ \tau_{xy}^{e} = \nu_e\frac{\partial u}{\partial y}.$$

На рис. 5 показаны зависимости L_∞-нормы ошибки (E_h) от выбранного шага по пространству h для u, v, τ_{xx}^e, τ_{xy}^e. Видно, что точность полученной модификации метода близка к $O(h^2)$.

Табл. 2. Сетки, используемые при решении модельной задачи
Table 2. Meshes used for model problem

Сетка	Число ячеек	Шаг по пространству h	Шаг по времени Δt
M_1	4000	0,1	0,05
M_2	16000	0,05	0,025
M_3	64000	0,025	0,0125

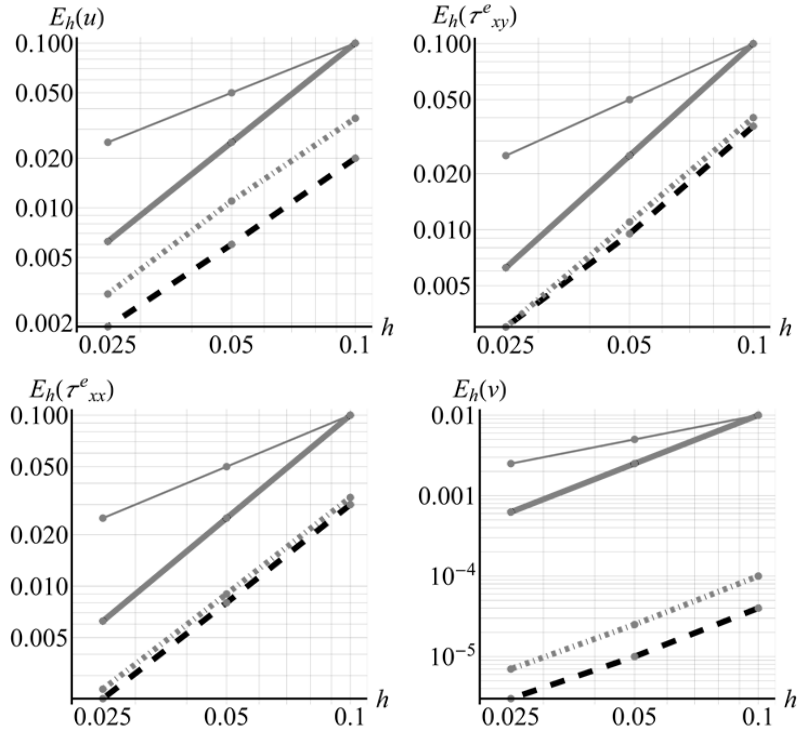


Рис. 5. Зависимости L_∞ *-нормы ошибки (* E_h *) от шага по пространству* h *: тонкая сплошная линия –* $O(h)$ *; жирная сплошная линия –* $O(h^2)$ *; пунктирная линия – расчет при* $\mathrm{We}=0{,}1$ $O(h^2)$ *; штрихпунктирная линия – расчет при* $\mathrm{We}=1{,}0$

Fig. 5. L_∞ *norm of the errors (* E_h *) versus mesh size: thin solid line –* $O(h)$ *; bold solid line –* $O(h^2)$ *; dotted line – computation at* $\mathrm{We}=0{,}1$ $O(h^2)$ *; dash-doted line – computation at* $\mathrm{We}=1{,}0$

7. Заключение

Разработана модификация метода погруженных границ LS-STAG для моделирования тячений вязкоупругих жидкостей, описываемых линейными и квазилинейными моделями скоростного типа (моделями Максвелла, Джеффри, Джонсона – Сигельмана, Максвелла-А, Олдройда-Б, Олдройда-А, верхней конвективной моделью Максвелла). Данная модификация реализована в программном комплексе, позволяющем проводить моделирование течений вязкой несжимаемой среды методом LS-STAG. В

статье представлены результаты верификации метода на примере тестовой задачи о моделировании течения Пуазейля. Как показывают вычислительные эксперименты, точность полученной модификации метода LS-STAG близка к $O(h^2)$.

Список литературы

[1]. Owens R.G., Phillips T.N. Computational Rheology. London: Imperial College Press, 2002. 417 p.

[2]. Galdi G.P., Rannacher R., Robertson A.M., Turek S. Hemodynamical Flows: Modeling, Analysis and Simulation. N.-Y.: Springer, 2008. 501 p.

[3]. Kim J.M., Kim C., Kim J.H., Chung C., Ahn K.H., Lee S.J. High-resolution finite element simulation of 4:1 planar contraction flow of viscoelastic fluid. J. Non-Newtonian Fluid Mech., 2005, № 129, pp. 23–37.

[4]. Mittal R., Iaccarino G. Immersed boundary methods. Annu. Rev. Fluid Mech., 2005, № 37, pp. 239–261.

[5]. Cheny Y., Botella O. The LS-STAG method: A new immersed boundary/level-set method for the computation of incompressible viscous flows in complex moving geometries with good conservation properties. J.Comp. Phys.2010, №229, pp. 043-1076.

[6]. Osher S., Fedkiw R.P. Level set methods and dynamic implicit surfaces. N. Y.: Springer, 2003. 273 p.

[7]. Puzikova V.V., Marchevsky I.K. Extension of the LS-STAG immersed boundary method for RANS-based turbulence models and its application for numerical simulation in coupled hydroelastic problems. Proc. VI International Conference on Coupled Problems in Science and Engineering. Venice. 2015, pp. 532–543.

[8]. Puzikova V.V. On generalization of the LS-STAG immersed boundary method for Large Eddy Simulation and Detached Eddy Simulation. Proc. Advanced Problems in Mechanics International Summer School-Conference. St.-Petersburg. 2015, pp. 411-417.

[9]. Marchevsky I., Puzikova V. Application of the LS-STAG Immersed Boundary Method for Numerical Simulation in Coupled Aeroelastic Problems. Proc. 11th World Congress on Computational Mechanics, 5th European Conference on Computational Mechanics, 6th European Conference on Computational Fluid Dyn. Barcelona, 2014, pp.1995-2006.

[10]. Пузикова В.В. Реализация параллельных вычислений в программном комплексе «LS-STAG_turb» для моделирования течений вязкой несжимаемой среды на системах с общей памятью. Труды ИСП РАН, том 28, вып. 1, 2016 г., стр. 221-242. DOI: 10.15514/ISPRAS-2016-28(1)-13.

[11]. Уилкинсон У.Л. Неньютоновские жидкости. М.: Мир, 1964. 216 с.

[12]. Maxwell J.C. On the dynamical theory of gases. Philos. Trans. R. Soc. 1867. № 157. P. 49–88.

[13]. Jeffreys H. The Earth Its Origin, History and Physical Constitution. Cambridge: Cambridge University Press, 1929. 612 p.

[14]. Oldroyd J.G. On the formulation of rheological equations of state. Proc. Roy. Soc. London, 1950, № 200, pp. 523–541.

[15]. Johnson M.W., Segalman D. A model for viscoelastic fluid behavior which allows non-affine deformation. J. Non-Newton. Fluid Mech., 1977, № 2, pp. 255–270.

[16]. Intel® Cilk™ Plus. URL: https://software.intel.com/ru-ru/node/522579 (accessed: 25.10.2015).

[17]. Reinders J. Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. Sebastopol: O'Reilly, 2007, 336 p.
[18]. Keiller R.A. Spatial decay of steady perturbations of plane Poiseuille flow for the Oldroyd-B equation. J. Non-Newton. Fluid Mech., 1993, vol. 46, pp. 129–142.
[19]. Hayat T., Khan M., Ayub M. Exact solutions of flow problems of an Oldroyd-B fluid. · J. Applied Math. and Comp., 2004, vol. 151, pp. 105–119.

The LS-STAG Immersed Boundary Method Modification for Viscoelastic Flow Computations

V. Puzikova <valeria.puzikova@gmail.com>
BMSTU, 5 2nd Baumanskaya st., Moscow, 105005, Russian Federation

Abstract. The LS-STAG immersed boundary cut-cell method modification for viscoelastic flow computations is presented. Rate type viscoelastic flow models (linear and quasilinear) are considered. Formulae for differential types of convected time derivatives the LS-STAG discretization was obtained. Normal non-newtonian stresses are computed at the centers of base LS-STAG mesh cells and shear non-newtonian stresses are computed at the cell corners. The LS-STAG-discretization of extra-stress equations for viscoelastic Maxwell, Jeffreys, upper-convected Maxwell, Maxwell-A, Oldroyd-B, Oldroyd-A, Johnson — Segalman fluids was developed. Time-stepping algorithm is defined by the following three steps. Firstly, a prediction of the velocity and pressure correction are computed by means of semi-implicit Euler scheme. Secondly, the provisional velocity is corrected to get a solenoidal velocity and the corresponding pressure field. After this the extra-stress equations are solved. Applications to popular benchmarks for viscoelastic flows with stationary boundaries and comparisons with experimental and numerical studies are presented. The results show that the developed LS-STAG method modification demonstrates an accuracy comparable to body-fitted methods. The obtained modification is implemented in the «LS-STAG» software package developed by the author. This software allows to simulate viscous incompressible flows around a moving airfoil of arbitrary shape or airfoils system with one or two degrees of freedom. For example, it allows to simulate rotors autorotation and airfoils system wind resonance. Intel® Cilk™ Plus, Intel® TBB and OpenMP parallel programming technologies are used in the «LS-STAG».

Keywords: Incompressible Flows; Viscoelastic Flows; Rate Type Viscoelastic Flow Models; Immersed Boundary Methods; the LS-STAG Method.

DOI: 10.15514/ISPRAS-2017-29(1)-5

For citation: Puzikova V. The LS-STAG Immersed Boundary Method Modification for Viscoelastic Flow Computations. *Trudy ISP RAN/Proc. ISP* RAS, vol. 29, issue 1, 2017, pp. 71-84 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-5

References

[1]. Owens R.G., Phillips T.N. Computational Rheology. *London: Imperial College Press*, 2002. 417 p.

[2]. Galdi G.P., Rannacher R., Robertson A.M., Turek S. Hemodynamical Flows: Modeling, Analysis and Simulation. *N. Y.: Springer*, 2008. 501 p.
[3]. Kim J.M., Kim C., Kim J.H., Chung C., Ahn K.H., Lee S.J. High-resolution finite element simulation of 4:1 planar contraction flow of viscoelastic fluid. *J. Non-Newtonian Fluid Mech*. 2005, no. 129, pp. 23–37.
[4]. Mittal R., Iaccarino G. Immersed boundary methods. *Annu. Rev. Fluid Mech*., 2005, no. 37, pp. 239–261.
[5]. Cheny Y., Botella O. The LS-STAG method: A new immersed boundary/level-set method for the computation of incompressible viscous flows in complex moving geometries with good conservation properties. *J.Comp.Phys.*, 2010, no.229, pp.1043-1076.
[6]. Osher S., Fedkiw R.P. Level set methods and dynamic implicit surfaces. *N. Y.: Springer*, 2003. 273 p.
[7]. Puzikova V.V., Marchevsky I.K. Extension of the LS-STAG immersed boundary method for RANS-based turbulence models and its application for numerical simulation in coupled hydroelastic problems. Proc. VI International Conference on Coupled Problems in Science and Engineering. Venice, 2015, pp. 532–543.
[8]. Puzikova V.V. On generalization of the LS-STAG immersed boundary method for Large Eddy Simulation and Detached Eddy Simulation. Proc. Advanced Problems in Mechanics International Summer School-Conference. St.-Petersburg. 2015, pp. 411-417.
[9]. Marchevsky I., Puzikova V. Application of the LS-STAG Immersed Boundary Method for Numerical Simulation in Coupled Aeroelastic Problems. *Proc. 11th World Congress on Computational Mechanics, 5th European Conference on Computational Mechanics, 6th European Conference on Computational Fluid Dyn.* Barcelona, 2014, pp.1995-2006.
[10]. Puzikova V.V. Realization of parallel computations in the software package «LS-STAG_turb» for viscous incompressible flow simulation on systems with shared memory. *Trudy ISP RAN / Proc. ISP RAS*, vol. 28, issue 1, 2016, pp. 221-242 (in Russian). DOI: 10.15514/ISPRAS-2016-28(1)-13.
[11]. Uilkinson U.L. Non-newtonian fluids. *Moscow*: Мир, 1964. 216 p. (in Russian)
[12]. Maxwell J.C. On the dynamical theory of gases. *Philos. Trans. R. Soc*. 1867, no. 157, P. 49–88.
[13]. Jeffreys H. The Earth Its Origin, History and Physical Constitution. *Cambridge: Cambridge University Press*, 1929. 612 p.
[14]. Oldroyd J.G. On the formulation of rheological equations of state. *Proc. Roy. Soc. London*. 1950, no. 200, pp. 523–541.
[15]. Johnson M.W., Segalman D. A model for viscoelastic fluid behavior which allows non-affine deformation. *J. Non-Newton. Fluid Mech*., 1977, no. 2, pp. 255–270.
[16]. Intel® Cilk™ Plus. URL: https://software.intel.com/ru-ru/node/522579 (accessed: 25.10.2015).
[17]. Reinders J. Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. *Sebastopol: O'Reilly*, 2007. 336 p.
[18]. Keiller R.A. Spatial decay of steady perturbations of plane Poiseuille flow for the Oldroyd-B equation. *J. Non-Newton. Fluid Mech*., 1993, vol. 46, pp. 129–142.
[19]. Hayat T., Khan M., Ayub M. Exact solutions of flow problems of an Oldroyd-B fluid. · *J. Applied Math. and Comp.* 2004, vol. 151, pp. 105–119.

Трёхмерное моделирование схода лавинных потоков средствами пакета OpenFOAM

Д.И. Романова <romanovadi@gmail.com>
Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1
Федеральное государственное учреждение "Федеральный научный центр
Научно-исследовательский институт системных исследований
Российской академии наук",
117218, Москва, Нахимовский просп., 36, к.1

Аннотация. В работе создана модель схода лавинного потока в пакете OpenFOAM, где лавина смоделирована как двухфазный турбулентный поток (снег - воздух). За основу был взят численный метод для решения задач движения двухфазных сред с поверхностью раздела - метод переноса объёмной доли (VOF). Данный алгоритм реализован в решателе InterFoam, который и был использован в работе. Использовалась K − ε модель турбулентности. В модели снег был представлен как нелинейно вязкая жидкость, описываемая реологическими соотношениями Хершеля-Балкли. Воздух представляет собой вязкую ньютоновскую среду. Для описания модели используются осреднённые по Рейнольдсу уравнения Навье-Стокса, реологические соотношения, а также уравнения для турбулентной кинетической энергии и диссипации. Построена расчётная область лавинного очага номер 22 горы Юкспор Хибинских гор на основе цифровой модели рельефа формата ASCII GRID. Взяты карты зоны зарождения и зоны отложений 129-ой лавины, глубина снежного покрова в зоне зарождения составила 1,5 метра, рассчитан поток, возникающий в зоне зарождения, и сравнивается модельная зона отложения с натурными данными. Также смоделировано движение снего-пылевого облака. Получены распределения скоростей, давления и объёмной доли снега во все моменты времени, форма лавинных отложений. Средняя скорость потока составила 44,8 м/с, что близко к действительным скоростям движения лавин в данном очаге. Максимальная скорость лавинного потока (включая снего-пылевое облако) составила 78 м/с. Данные результаты позволяют оптимально проектировать противолавинные сооружения и определять лавиноопасные территории.

Ключевые слова: лавиноведение; турбулентный поток; двухфазная среда; жидкость Хершеля-Балкли.

DOI:10.15514/ISPRAS-2016-29(1)-6

Для цитирования: Романова Д.И. Трёхмерное моделирование потоков жидкости Хершеля-Балкли на склоне в OpenFOAM. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 85-100. DOI: 10.15514/ISPRAS-2017-29(1)-6

1. Введение

Склоновые потоки, в том числе лавины, — распространённое явление в горах. В России они встречаются на Кавказе, в Хибинах, горах Прибайкалья, Забайкалья, на Урале, Чукотке — словом, всюду, где углы падения склонов больше 15°, и имеется снежный покров глубиной 30 - 40 см и выше. Громадные разрушения, вызываемые обвалами снега, человеческие жертвы, миллионные убытки, невозможность поддержать круглогодичное бесперебойное движение по горным дорогам, требуют уделять большое внимание проблеме борьбы с лавинами, исследовать и моделировать их. Для оценки лавинной опасности при изысканиях и строительстве в лавиноопасных районах необходимо определять наибольшую дальность выброса лавин и возможную силу удара лавинного снега. Для определения силы удара требуется знать скорость движения лавины (точнее, её фронтальной части), а также высоту фронта.

2. Модель движущейся среды

Лавина представлена как несжимаемый турбулентный двухфазный поток — снег и воздух. Используются уравнения, осреднённые по Рейнольдсу. Модель описывается системой уравнений, включающей уравнения Рейнольдса, уравнение неразрывности, уравнение объёмной доли фазы, уравнение для кинетической энергии турбулентных пульсаций, уравнение диссипации и реологические соотношения.

Уравнения Рейнольдса выглядят следующим образом (черта над буквой означает осреднение по Рейнольдсу):

$$\frac{\partial(\rho\bar{u}_i)}{\partial t} + \frac{\partial}{\partial x_j}\left(\partial\bar{u}_i\bar{u}_j + \rho\overline{u'_i u'_j}\right) = \rho g_i - \frac{\partial\bar{p}}{\partial x_j} + \frac{\partial\bar{\tau}_{ij}}{\partial x_j},$$

где $\bar{\tau}_{ij}$ - компоненты осреднённого по Рейнольдсу тензора вязких напряжений:

$$\bar{\tau}_{ij} = 2\mu\bar{e}_{ij}, \qquad \mu = \mu(|\bar{\dot{\gamma}}|),$$

$$\bar{e}_{ij} = \frac{1}{2}\left(\frac{\partial\bar{u}_i}{\partial x_j} + \frac{\partial\bar{u}_j}{\partial x_i}\right), \qquad |\bar{\dot{\gamma}}| = \sqrt{2I_2(\bar{e})}, \qquad I_2(\bar{e}) = e_{ij}e_{ij}.$$

Далее μ выражается через кинематическую вязкость: $\mu = \rho\nu$,

а ρ и ν вычисляются через объёмную долю снега α

$$\nu = (1-\alpha)\nu_{air} + \alpha\nu_{snow},$$

$$\rho = (1-\alpha)\rho_{air} + \alpha\rho_{snow},$$

$$\alpha = \begin{cases} 1 & - \text{снег} \\ 0 < \alpha < 1 & - \text{в переходном слое} \\ 0 & - \text{воздух} \end{cases}$$

Здесь индекс «air» соответствует воздуху, а индекс «snow» - снегу.

Для различных фаз использованы различные реологические соотношения, для снега взяты реологические соотношения Хершеля-Балкли [1, 2], воздух представлен как ньютоновская среда.

Ньютоновская модель: $\tau = \mu\dot{\gamma},\ \mu = const$

Модель Хершеля-Балкли: $\tau = \mu\dot{\gamma},\ \ \mu = \frac{\tau_0 + k|\dot{\gamma}|^n}{|\dot{\gamma}|},\quad k, n = const$

Здесь τ – сдвиговое напряжение, $|\dot{\gamma}|$ - скорость сдвига.

В случае простого сдвигового потока соответствующие реологические соотношения имеют вид (см. рис. 1)

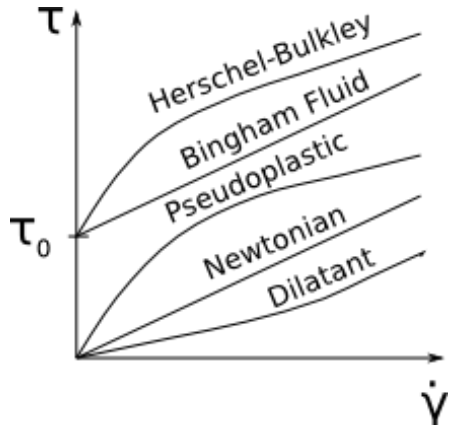


Рис. 1. Зависимость сдвигового напряжения от скорости сдвига в простом сдвиговом потоке для разных моделей сред.

Fig. 1. The shear rate dependency of shear stress in a simple shear flow for different rheological models.

В силу выбранных реологических соотношений кинематическая вязкость вычисляется по-разному для воздуха и для снега:

$\nu_{air} = const = 1.48 \cdot 10^{-5}$ м^2/сек, $\qquad \nu_{snow} = min\left(\nu_{snow_0}, \frac{\tau_0 + k|\dot{\bar{\gamma}}|^n}{\rho_{snow}|\dot{\bar{\gamma}}|}\right),$

где $\nu_{snow_0} = const$ и задаётся при калибровке модели в соответствии с данными исследуемой лавины.

Необходимость введения константы ν_{snow_0} связана с тем, что значение кинематической вязкости снега при $\tau_0 \neq 0$, стремится к бесконечности, когда $|\dot{\gamma}| \to 0$, как показано на рис. 2.

В расчетах задана плотность воздуха $\rho_{air} = 1$ кг/м^3. Параметры для снега, такие как $\rho_{snow}, k, n, \tau_0$, будут подобраны при калибровке модели в соответствии с натурными данными. Скорости фаз считаются одинаковыми.

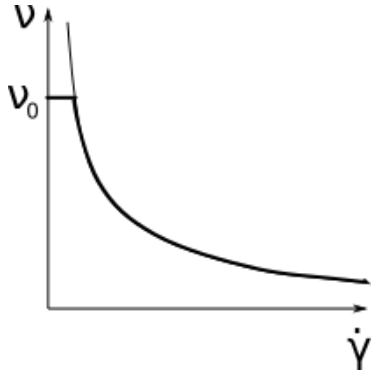


Рис. 2. Зависимость эффективной вязкости от скорости сдвига для среды Хершеля-Балкли при n < 1.

Fig. 2. The shear rate dependency of effective viscosity at n < 1.

Для описания модели использованы следующие уравнения.

Уравнение неразрывности:

$$\frac{\partial \rho}{\partial t} + \frac{\partial (\rho \bar{u}_i)}{\partial x_i} = 0.$$

Уравнение объёмной доли фазы:

$$\frac{\partial \alpha}{\partial t} + \frac{\partial (\alpha u_j)}{\partial x_j} = 0.$$

В качестве модели турбулентности берётся $K - \varepsilon$ модель. Уравнение для турбулентной кинетической энергии K:

$$\frac{\partial (\rho K)}{\partial t} + \frac{\partial (\rho \bar{u}_j K)}{\partial x_j} = \frac{\partial}{\partial x_j}\left(\mu_t \frac{\partial K}{\partial x_j}\right) + \frac{\partial}{\partial x_j}\left(\frac{\mu_t}{\sigma_K}\frac{\partial K}{\partial x_j}\right) + P_K - \rho\varepsilon.$$

Уравнение для диссипации ε:

$$\frac{\partial (\rho \varepsilon)}{\partial t} + \frac{\partial (\rho u_j \varepsilon)}{\partial x_j} = C_{\varepsilon 1} P_K \frac{\varepsilon}{K} - \rho C_{\varepsilon 2} \frac{\varepsilon^2}{K} + \frac{\partial}{\partial x_j}\left(\frac{\mu_t}{\sigma_\varepsilon}\frac{\partial \varepsilon}{\partial x_j}\right),$$

$$-\rho \overline{u'_\iota u'_j} = 2\mu_t \bar{e}_{ij} - \frac{2}{3}\rho \delta_{ij} K, \quad \mu_t = \rho C_\mu \frac{K^2}{\varepsilon}, \quad P_K = -\rho \overline{u'_\iota u'_j} \frac{\partial \bar{u}_i}{\partial x_j}$$

Приведённая выше модель турбулентности содержит пять констант [1], которые задаются следующим образом:

$$C_\mu = 0.09;\ C_{\varepsilon 1} = 1.44;\ C_{\varepsilon 2} = 1.92;\ \sigma_K = 1.0;\ \sigma_\varepsilon = 1.3.$$

3. Программное обеспечение

Модель лавинного потока была построена в пакете OpenFOAM.

OpenFOAM (Open Source Field Operation And Manipulation CFD ToolBox) - открытая интегрируемая платформа для численного моделирования задач механики сплошной среды. Используется решатель InterFoam, предназначенный для расчета нестационарного течения двух сред, разделённых

границей раздела или свободной поверхностью. Для нахождения формы свободной поверхности двухфазный алгоритм в InterFoam использует один из методов, в котором интерфейс не рассматривается как резкая граница (метод захвата интерфейса). Расчет происходит на фиксированной сетке, включающей свободную поверхность. Форма свободной поверхности определяется расчётом доли заполненности каждой ячейки вблизи поверхности. Это можно сделать, решая уравнение транспорта для доли ячейки, занимаемой жидкой фазой; такой метод называется "объём жидкости"(Volume-of-Fluid) или схема VOF. Более подробное описание использованного решателя можно найти в [3].

4. Объект моделирования

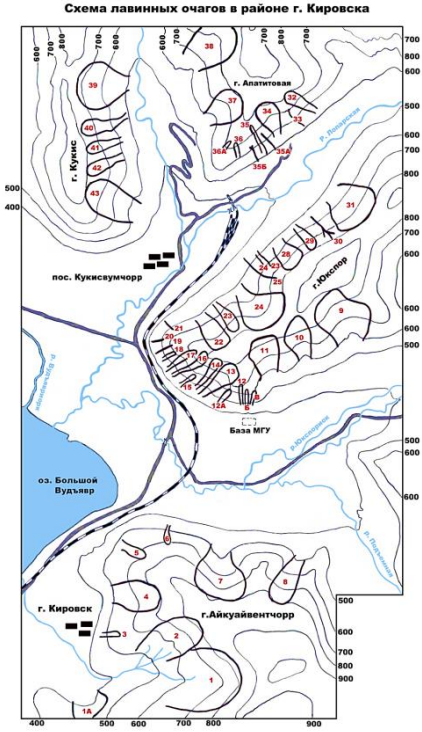


Рис. 3. Схема лавинных очагов в районе г. Кировска.

Fig. 3. Scheme of avalanche sites nearby city Kirovsk

При помощи построенной модели была рассчитана лавина, сошедшая в 1965 году из 22-ого лавинного очага горы Юкспор в Хибинах, вблизи города Кировск, см. рис. 3, 4.

Именно в данном очаге был произведён профилактический спуск лавины в 22:00 18 февраля 2016 года. Лавина приобрела катастрофический характер и унесла жизни троих людей, в том числе одного из сотрудников лавинной службы. Были засыпаны железная и автомобильная дороги, выбиты стёкла в трёх близлежащих домах.

На основе цифровой растровой карты 22-ого лавинного очага формата ASCII GRID была построена расчётная область (рис. 5), которая представляет собой пространственную область, ограниченную снизу поверхностью склона, а сверху– поверхностью такой же формы, сдвинутой вверх на 25 метров.

Из паспорта лавины, сошедшей в 1965г., были взяты следующие параметры:

- максимальная толщина слоя снега в месте отрыва лавины - 1.5 м;
- средняя толщина слоя снега в месте отрыва лавины - 0.8 м;
- максимальная плотность лавинных отложений - 320 кг/м3;
- средняя плотность лавинных отложений - 290 кг/м3;
- минимальная плотность лавинных отложений - 260 кг/м3;
- состояние снега - сухой;
- максимальная толщина лавинных отложений - 4 м;

Рис. 4. Фотография лавинного очага №22 (НИЛ снежных лавин и селей)

Fig. 4. Photo of 22 avalanche cite (Research Laboratory of Snow Avalanches and Debris Flows).

- средняя толщина лавинных отложений - 0.72 м.

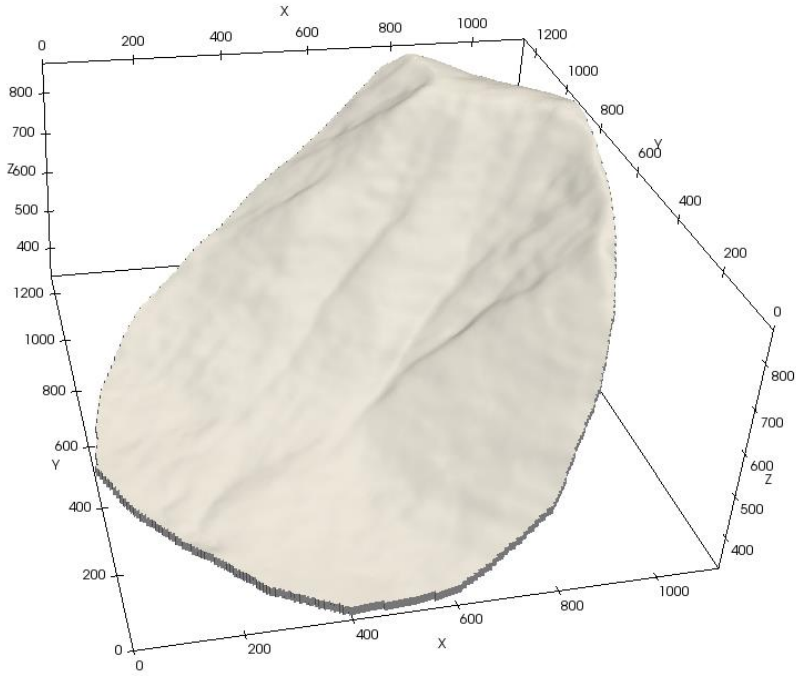


Рис. 5. Расчётная область.

Fig. 5. Computational domain.

5. Расчётная сетка

Расчёт производился на сетке из ячеек в форме параллелепипедов с линейным размером 5м, состоящей из:

- 206 630 точек
- 532 887 граней
- 447 345 внутренних граней
- 163 372 ячеек

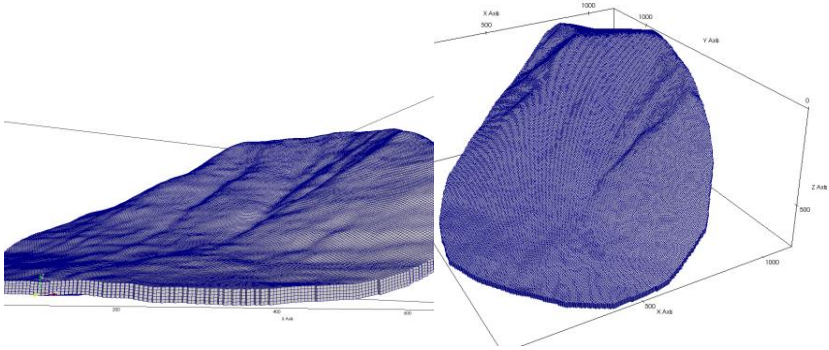


Рис. 6. Сеточные линии расчётной области.

Fig. 6. Mesh grid of computational domain.

6. Начальные и граничные данные

В задаче заданы следующие граничные условия.

На нижней границе расчётной области установлено условие прилипания.

Боковые и верхняя границы расчётной области заданы как стандартные участки границы с условием протекания, на которых действует атмосферное давление.

На рис. 7 зелёным цветом обозначен слой снега, лежащий в зоне зарождения лавины, при $t = 0$ он покоится ($\vec{U} = 0$ всюду). Глубина снега во всех точках при $t = 0$ не превышает 1.5 м. Математически это выражается так, что маркерная функция α равна 1 в тех ячейках расчётной сетки, где снег есть, и 0 в остальных ($\alpha = \alpha(x, y, z)$). Также при $t = 0$: $\varepsilon = 0$, $K = 0$, $\mu_t = 0$.

Серым цветом на рис. 7 изображена реальная зона отложений исследуемой лавины.

Зона зарождения лавины и зона лавинных отложений соответствуют начальному и конечному этапам рассматриваемого процесса соответственно.

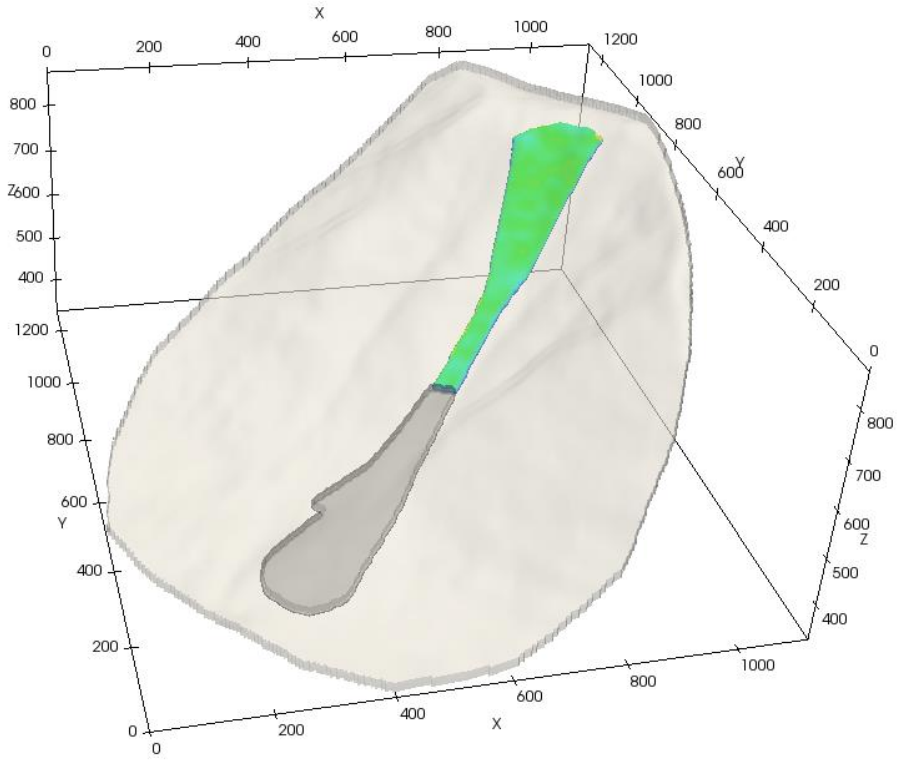


Рис. 7. Лавинный поток в начальный момент времени.

Fig. 7. Avalanche at initial time.

7. Расчёт

В результате дискретизации исходных уравнений в частных производных методом конечных объёмов были получены системы линейных алгебраических уравнений, которые решались итерационными методами.

Для решения дискретизированных уравнений для объёмной доли фазы α, турбулентной кинетической энергии K, диссипации ε и уравнения движения используется решатель со сглаживанием с помощью метода Гаусса-Зейделя.

Уравнение на давление, которое получено при помощи уравнения неразрывности, решается методом сопряжённых градиентов с предобуславливателем.

Расчёты производились на высокопроизводительном вычислительном сервере НИИСИ РАН, состоящем из:

- 2 x Xeon E5-2670v1 (Sandy Bridge) 8 cores 2.6 GHz (16 threads)
- 4xchannel DDR3 memory controller
- MEM 256 GB DDR3-1333 ECC
- OS Ubuntu 15.10 64-bit

8. Результаты

В процессе вычислений были получены значения $u_i(x,y,z,t)\ i = 1,2,3$, $\alpha(x,y,z,t)$, $p(x,y,z,t)$, а также граница лавинного потока, которая была определена из условия $\alpha = 0.1$.

Проведены расчёты со значениями параметров ρ_{snow}, ν_{snow_0} , τ_0, k, n для снега в следующих диапазонах:

ρ [кг/м^3]	ν_0 [м^2/с]	τ_0/ρ [м^2/с^2]	k/ρ [м^2/с]	n
200	100	1	10^{-5}	2
300	10^5	0.1	10^{-4}	0.5
	10^{-2}	10	10^{-6}	

Было посчитано 108 случаев, что заняло 36 часов машинного времени. Были сделаны выводы, что параметры τ_0 и ν_0 влияют на динамику потока, и при подходящем выборе поток начинает замедляться на склоне.

Скорости потока в данных калибровочных расчётах получились в среднем 60 м/с, что много для данной лавины. Было найдено, что на величину скорости существенно влияет параметр вязкости k и проведено дополнительное исследование по параметру k с остальными зафиксированными подходящими параметрами, взятыми из предыдущих расчётов. Рассчитывались варианты с k/ρ = 2, 3, 4, 5, 6, 7, 8, 9, 10, 50, 100 м^2/с, $\tau_0/\rho = 10$ м^2/с^2, $\nu_0 = 10^5$ м^2/с, $\rho =$ 200 кг/м^3, $n = 0.5$. Расчёт с параметром $k/\rho = 5$м^2/с наиболее точно описал натурные данные.

На рис. 8 - 11 изображены границы лавинного потока и распределение величины скорости на поверхности потока в разные моменты времени. На рис. 12 - 15 изображено распределение величины скорости на верхней границе расчётной области в те же моменты времени. Оно отображает распространение снего-пылевого облака.

Оси x, y – горизонтальные, ось z – вертикальная. На цветной легенде U Magnitude - модуль скорости, $U\ Magnitude = \sqrt{u_x^2 + u_y^2 + u_z^2}$.

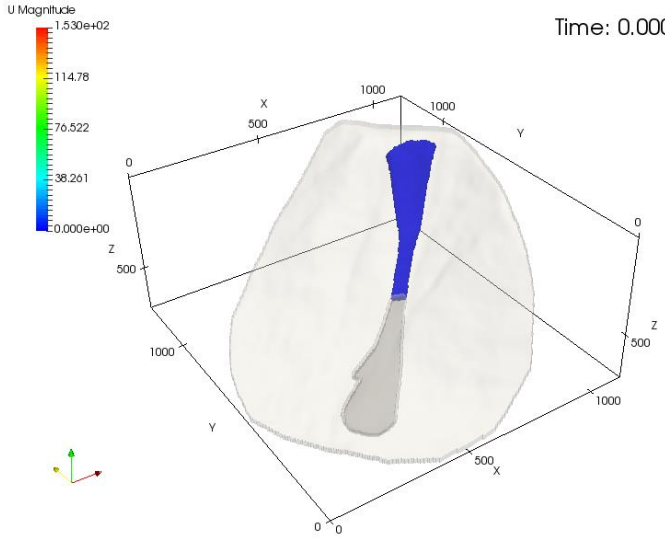


Рис. 8. Лавинный поток в начальный момент времени.
Fig. 8. Avalanche at start moment.

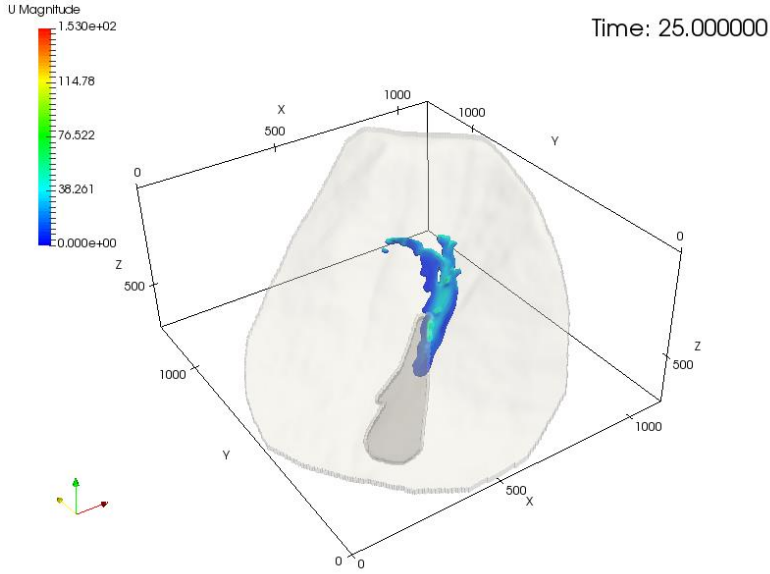


Рис. 9. Ускорение лавинного потока.
Fig. 9. Avalanche acceleration

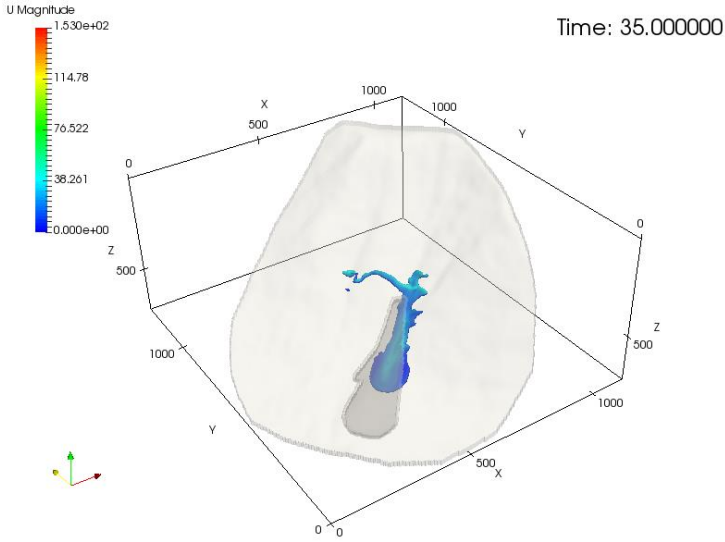


Рис. 10. Замедление лавинного потока.
Fig. 10. Avalanche deceleration.

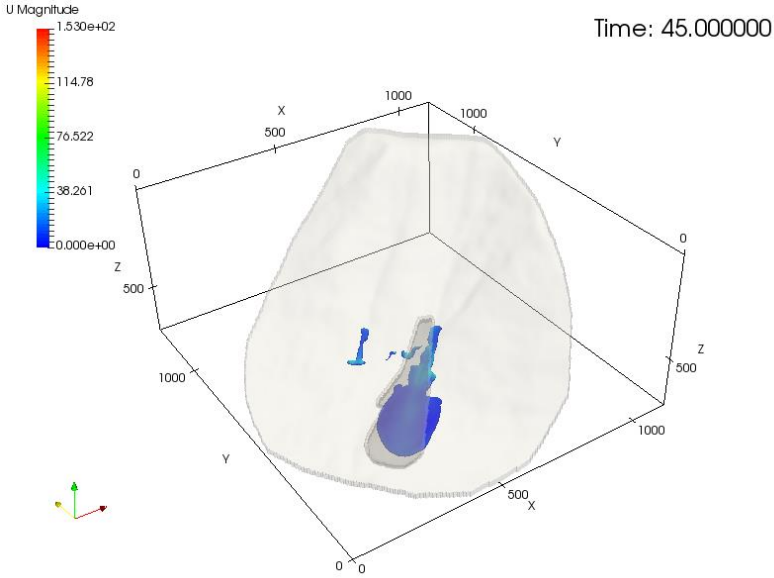


Рис. 11. Остановка лавинного потока.
Fig. 11. Avalanche stop.

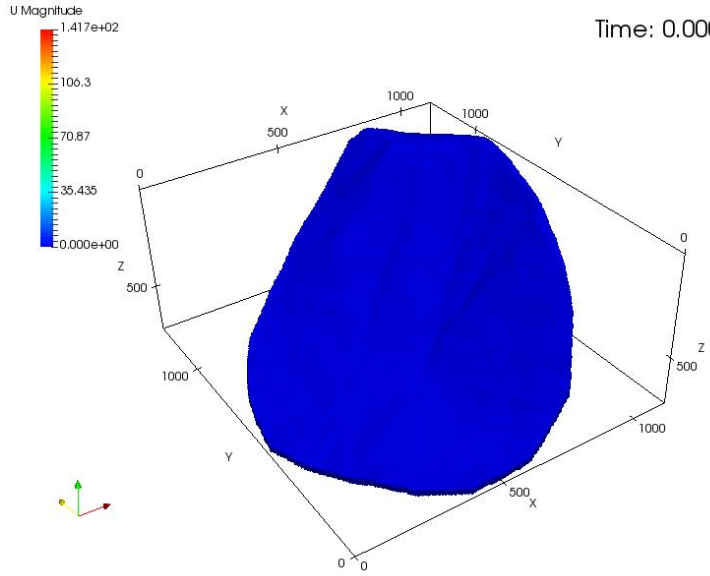


Рис. 12. Снего-пылевое облако в начальный момент времени.
Fig. 12. Snow-dust cloud at start moment.

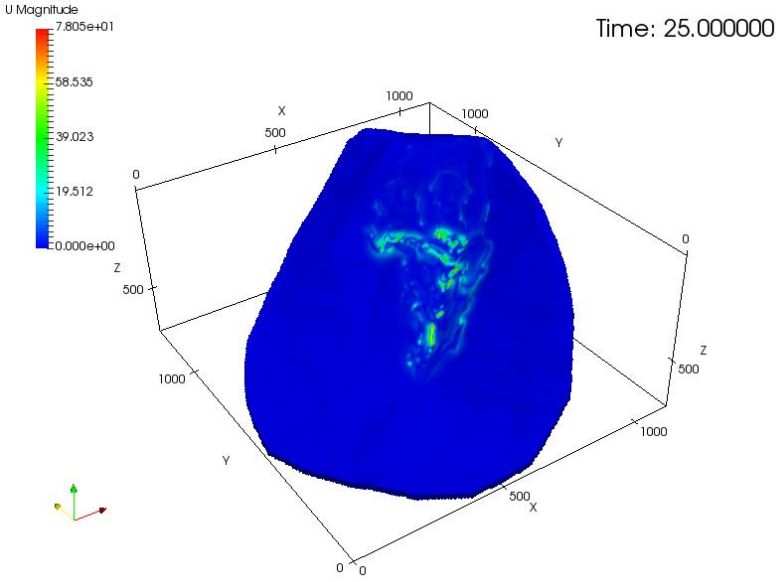


Рис. 13. Ускорение снего-пылевого облака.
Fig. 13. Snow-dust cloud acceleration

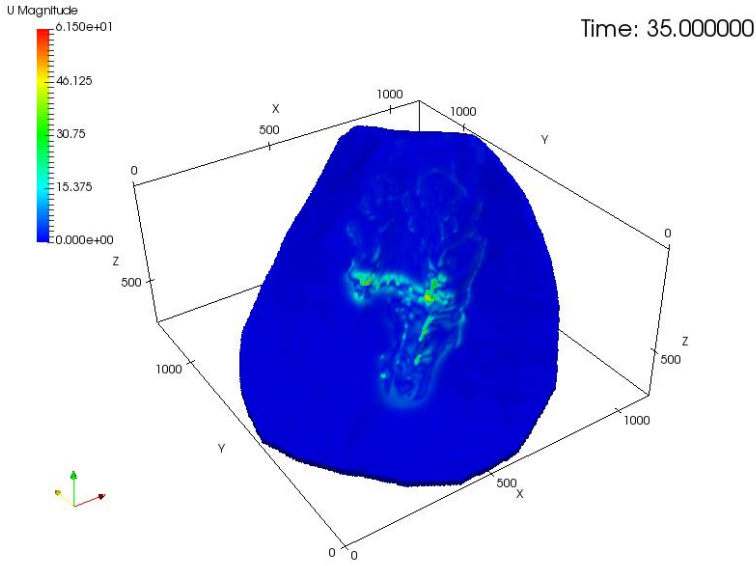


Рис. 14. Движение снего-пылевого облака.
Fig. 14. Snow-dust cloud motion.

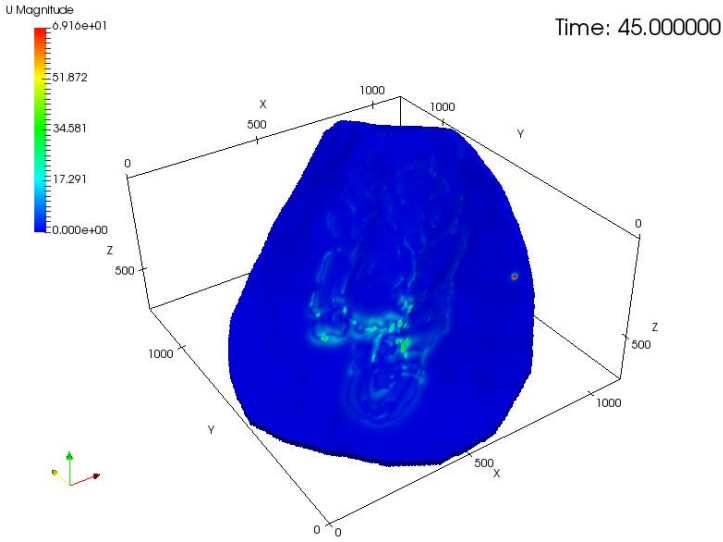


Рис. 15. Замедление снего-пылевого облака.
Fig. 15. Snow-dust cloud deceleration.

Можно видеть, что в данном расчёте после полной остановки потока форма лавинных отложений, полученная в процессе вычислений, близка к натурной форме лавинных отложений. Средняя скорость потока составляет 44,8 м/с, что близко к действительным скоростям движения лавин в данном очаге.

Максимальная скорость лавинного потока (включая снего-пылевое облако) составляет 78 м/с.

9. Заключение

Была создана модель лавинного потока в пакете OpenFOAM, позволяющая получить значения компонент скорости, давление, объёмную долю снега в каждой точке расчётной области в различные моменты времени. Была рассчитана лавина, сошедшая в Хибинах, на горе Юкспор, в 22-ом лавинном очаге. По полученным данным была построена зона лавинных отложений, границы которой близки к зафиксированным непосредственно после схода реальной лавины. Вычислены распределения скоростей, объёмной доли снега и давления во все моменты времени, в частности максимальная скорость лавинного потока. Данные результаты позволяют определить лавиноопасную зону и помогают оптимально спроектировать защитные сооружения.

В дальнейшем планируется рассчитать задачу на более подробной сетке, а также построить для сравнения данную модель в другом пакете, например, INMOST или при помощи собственного кода.

Автор благодарит профессора МГУ М.Э. Эглит за многочисленные обсуждения модели, П.Б. Богданова, сотрудника ФГУ ФНЦ НИИСИ РАН, за помощь в организации вычислительного алгоритма и его программной реализации и Ю.Г. Селиверстова, Т.Г. Глазовскую, А.С. Турчанинову сотрудников Научно-исследовательской лаборатории снежных лавин и селей Географического факультета МГУ за предоставленные данные и обсуждение результатов работы.

Список литературы

[1]. Эглит М.Э., Якубенко А.Е. Влияние захвата донного материала и неньютоновской реологии на динамику турбулентных склоновых потоков. Известия Российской академии наук. Механика жидкости и газа. 2016, № 3, стр. 3-15. DOI: 10.7868/S056852811603004X

[2]. Eglit M.E., Yakubenko A.E. Numerical modeling of slope flows entraining bottom material. Cold Regions Science and Technology 108 (2014), 139–148

[3]. Joel H. Ferziger, Milovan Perit. Computational Methods for Fluid Dynamics - 3., rev. ed. - Berlin; Heidelberg; New York; Barcelona; Hong Kong; London; Milan; Paris; Tokyo: Springer, 2002.

3D avalanche flow modeling using OpenFOAM

D.I. Romanova<romanovadi@gmail.com>
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
Federal State Institution
«Scientific Research Institute for System Analysis of the Russian Academy of Sciences»,
36к1, Nakhimovskiy pr., Moscow, 117218, Russia

Abstract. In this paper, a model of a snow avalanche was created with the help of open source CFD software OpenFOAM. The avalanche is considered as a turbulent two-phase flow — snow and air. We take incompressible Herschel-Balkley fluid as a model of snow. Air is a newtonian fluid. For tracking and locating the free surface we use the volume of fluid (VOF) method. In calculations we use solver interFoam which is based on the VOF method. The K – ε turbulence model was used. Navier–Stokes equations, rheological ratios, equations for turbulent kinetic energy and dissipation are used to determine the model. The avalanche occurred at the 22nd site on the Ukspor mountain was modeled. Computational domain was made using digital terrain model in ASCII GRID format. The shape of snow deposits area was calculated and compared with the real data. The velocity field of flow, pressure distribution, the field of volume snow fraction were obtained for different time instances. The value of the average velocity of the flow was 44,8 m/s, the value of the maximum velocity of the flow (including snow-dust cloud) was 78 m/s. These results allow to determine the avalanche hazard area and help to optimally design the defense systems. In future it is planned to simulate this event using more detailed grids and realize this model in other software for example INMOST or with own code to compare the results.

Keywords: glaciology; turbulent flow; two-phase medium; Herschel-Bulkley fluid.

DOI: 10.15514/ISPRAS-2017-29(1)-6

For citation: Romanova D.I. 3D flow modeling of Herschel-Bulkley fluid on the slope in OpenFOAM. Trudy ISP RAN/Proc. ISP RAS, vol. 29, issue 1, 2017, pp. 85-100. DOI: 10.15514/ISPRAS-2017-29(1)-6

References

[1]. Eglit M.E., Yakubenko A.E. Effect of the bottom material capture and the non-Newtonian rheology on the dynamics of turbulent downslope flows. Fluid Dynamics. 2016, vol. 51, № 3, pp. 299-310. DOI: 10.1134/S0015462816030017

[2]. Eglit M.E., Yakubenko A.E. Numerical modeling of slope flows entraining bottom material. Cold Regions Science and Technology, 108 (2014), pp. 139–148

[3]. Joel H. Ferziger, Milovan Perit. Computational Methods for Fluid Dynamics - 3., rev. ed. - Berlin; Heidelberg; New York; Barcelona; Hong Kong; London; Milan; Paris; Tokyo: Springer, 2002

Моделирование перемещения клиновидного виброробота в вязкой жидкости при различных законах движения внутренней массы в пакете OpenFOAM

[1,2] *А.Н. Нуриев <nuriev_an@mail.ru>*
[3] *А.И. Юнусова <yunusova24@gmail.com>*
[2] *О.Н.Зайцева <olga_fdpi@mail.ru>*
[1] *Нижегородский государственный университет,*
603022, Россия, г. Н.Новгород, пр. Гагарина, 23
[2] *Казанский федеральный университет,*
420008, Россия, РТ, г. Казань, ул. Кремлевская, д.18
[3] *Казанский национальный исследовательский технологический университет*
420015, Россия, РТ, г. Казань, ул. К. Маркса, 68

Аннотация: Работа посвящена исследованию движения двухмассовой вибрационной системы в вязкой жидкости. Система состоит из замкнутого клиновидного корпуса и подвижной внутренней массы, совершающей колебания вдоль продольной оси корпуса. Описанная механическая система имитирует виброробот. Рассматривается комплексная модель взаимодействия робота со средой, в рамках которой движение жидкости описывается полной нестационарной системой уравнений Навье-Стокса. Исследуются вопросы повышения эффективности движения виброробота за счет выбора специального закона перемещения внутренней массы. Для этого проводится сравнительный анализ характеристик движения (средней скорости и показателя эффективности) и режимов обтекания при простом гармоническом законе колебания внутренней массы и специальном двухфазном, характеризующимся чередованием медленной продолжительной и быстрой короткой фаз движения, во время которых внутренняя масса перемещается с постоянной скоростью. Решение задачи выполняется численно на базе пакта с отрытым исходным кодом OpenFOAM. Численная схема реализуется в рамках конечно-объемного подхода дискретизации. Для совместного решения системы уравнений Навье-Стокса и механической системы, описывающей взаимодействие составляющих виброробота и вязкой среды применяется специальная итерационная схема, встраиваемая в стандартный решатель пакета icoFoam. Результаты исследования показывают, что направленное движение виброробота с клиновидной формой корпуса возможно как для гармонического, так и для двухфазного законов колебания внутренней массы. В каждом из случаев удается обнаружить устойчивые режимы движения, реализуемые в широком диапазоне чисел Рейнольдса. Анализ

средней скорости и эффективности режимов позволяет найти оптимальные параметры движения виброробота.

Ключевые слова: виброробот; вязкая жидкость; система уравнений Навье-Стокса; режимы движения; эффективность движения; OpenFOAM.

DOI: 10.15514/ISPRAS-2017-29(1)-7

Для цитирования: Нуриев А.Н., Юнусова А.И., Зайцева О.Н. Моделирование перемещения клиновидного виброробота в вязкой жидкости при различных законах движения внутренней массы в пакете OpenFOAM. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 101-118. DOI: 10.15514/ISPRAS-2017-29(1)-7

1. Введение

Вибрационный принцип движения тел уже многие годы вызывает интерес у инженеров. Описания многочисленных устройств с вибрационным движителем появлялись в технической литературе еще в первой половине 20-го века. В настоящее время вибрационное движение — это динамично развивающийся раздел прикладной механики и робототехники.

Одна из простейших моделей вибрационного устройства, способного перемещаться в сопротивляющейся среде, может быть представлена в виде двухмассовой системы, состоящей из замкнутого корпуса и подвижного внутреннего груза. Устройства подобной архитектуры часто называют вибророботами. Перемещение системы как целого происходит за счет продольного периодического движения одного тела (внутренней массы) относительно другого (корпуса). Такой принцип передвижения представляется целесообразным для мини- и микро-устройств. Герметичность, отсутствие подвижных внешних частей – свойства вибророботов, позволяющие использовать их для неразрушающей инспекции миниатюрных технических объектов, таких как тонкостенные трубопроводы малого диаметра, а также в медицине, о чем упоминалось рядом авторов [1, 2].

Несмотря на простую архитектуру, вопросы управления вибророботами образуют целый ряд нетривиальных задач, ключевыми из которых являются анализ взаимодействия устройств со средой и оптимизация их движения в соответствии с особенностями этого взаимодействия. Исследования возможностей движения вибророботов проводились ранее для сред с различными законами сопротивления. В работах [3-5] рассматривалась возможность движения в идеальной жидкости, связанная с деформациями внешней оболочки, статьи [6, 7] были посвящены изучению движения по шероховатой плоскости при наличии Кулоновского трения, в работе [8] проводилось экспериментальное исследование движения по поверхности жидкости.

Движение вибророботов в ньютоновской жидкости рассматривалось в работах [9-14]. В [12 - 14] в частности поднимались вопросы оптимизации движения в рамках квазистационарных моделей взаимодействия с вязкой средой. В [12]

решалась задача оптимизации при наличии произвольной степенной зависимости сил сопротивления от скорости, в том числе квадратичной, которая часто используется как приближение для выражения сил сопротивления, возникающих при движении тела в ньютоновской жидкости. В работе [13] оптимизировалось движение виброробота в вязкой жидкости, закон сопротивления был сконструирован на основе экспериментальных данных по стационарному обтеканию сферы. В [14] задача оптимизации решалась в рамках модели взаимодействия, учитывающей зависимость силы не только от мгновенных скоростей, но и от так называемых наследственных эффектов, связанных с предысторией движения. Именно увеличение роли наследственных эффектов делает квазистационарные модели непригодными для описания высокочастотного движения, а существующие аппроксимации их влияния на силы взаимодействия применимы только для очень малых чисел Рейнольдса Re~1. Поэтому изучение взаимодействия при высокочастотных колебаниях является на сегодняшний день актуальной задачей. В работе [15] исследование движения виброробота в жидкости проводилось на базе численного моделирования, основанного на совместном решении механической и существенно нестационарной гидродинамической задач. Результаты показали, что даже для простого гармонического закона движения внутренней массы в диапазоне низких чисел Рейнольдса течение вокруг робота имеет комплексный характер, связанный в первую очередь с интенсивным вихреобразованием и переключением между режимами течения жидкости. Структура течения, создаваемая движением робота, существенно влияет на характеристики движения, в том числе определяет направление перемещения системы.

Исследования в настоящей работе направлены на дальнейшее изучение взаимодействия виброробота с вязкой жидкостью. Рассматривается движение виброробота с клиновидным корпусом в области низких чисел Рейнольдса. Исследуются вопросы повышения эффективности и скорости движения устройства за счет выбора специального закона перемещения внутренней массы. Для этого проводится сравнительный анализ характеристик движения и режимов обтекания при простом гармоническом законе колебания внутренней массы и специальном двухфазном (характеризуется чередованием быстрой и медленной фаз движения), полученным в [12] в ходе решения задачи оптимизации энергозатрат виброробота на базе упрощенной модели взаимодействия с вязкой жидкостью. В обоих случаях рассматривается периодическое движение, происходящее вдоль оси симметрии корпуса робота. В качестве основного инструмента исследования используется прямое численное моделирование. Вычислительная схема реализуется в открытом пакете OpenFOAM [16], на базе модели предложенной в [15]. Расчеты проводятся на высокопроизводительных кластерах КФУ и проекта unihub [17].

2. Постановка задачи

Рассмотрим прямолинейное движение двухмассового клиновидного

виброробота, состоящего из корпуса массы M, находящегося в вязкой несжимаемой жидкости, и подвижной внутренней массы m, совершающей периодическое движение внутри него. Обозначим через u_M скорость корпуса, а через s и $u_m = \dot{s}$ – перемещение и скорость внутренней массы относительно корпуса. Уравнения движения этой системы в неподвижной системе координат имеют вид:

$$m(\dot{u}_M + \dot{u}_m) = -G,\ M\dot{u}_M = G + F. \qquad (1)$$

Здесь F – сила, действующая на тело со стороны жидкости, G – сила взаимодействия внутренней массы и корпуса. Исключая силу G из уравнения (1), нормируя скорость u на амплитуду скорости U_0 колебания внутренней массы, время t на RU_0^{-1}, где R – характерный размер тела, получим основное уравнение движения двухмассовой системы в следующем виде:

$$\dot{u}_M = -\mu_2\,\dot{u}_m + \mu_1 \frac{R^2}{S} F \qquad (2)$$

Здесь μ_2 – отношение подвижной массы к полной массе виброробота ($\mu_2 = \dfrac{m}{M+m}$), μ_1 – отношение массы вязкой жидкости, занимающей тот-же объем, что и виброробот, к массе виброробота ($\mu_1 = \dfrac{M_f}{M+m}$), S – площадь поперечного сечения корпуса.

Движение жидкости вокруг виброробота описывается системой уравнений Навье-Стокса. Нормируя пространственные координаты, время и скорость на R, RU_0^{-1}, U_0 соответственно, запишем управляющую систему уравнений в декартовой системе координат как

$$\frac{\partial \boldsymbol{U}}{\partial t} + \boldsymbol{U}\cdot\nabla\boldsymbol{U} = -\nabla p + \frac{1}{\mathrm{Re}}\Delta\boldsymbol{U},\ \nabla\cdot\boldsymbol{U} = 0. \qquad (3)$$

где $\boldsymbol{U} = (u, v)$ – безразмерная скорость, p – безразмерное давление, $\mathrm{Re} = U_0 R/\nu$ – число Рейнольдса.

Для численного решения данной задачи удобно перейти в подвижную систему координат связанную с вибророботом. Для сохранения основной системы уравнений движения жидкости в форме (3), определим давление как:

$$p = \tilde{p} + x\dot{w}.$$

Здесь первое слагаемое в правой части – давление в неподвижной системе координат, а второе – вклад от инерциальных составляющих, $\dot{w}$ – ускорение подвижной системы координат.

На границе виброробота в новой системе координат задаются условия прилипания:

$$u|_c = v|_c = 0. \quad (4)$$

Условия на бесконечности получаются из уравнения взаимодействия (2):

$$\dot{u}|_\infty = \mu_2 \dot{u}_m - \mu_1 \frac{R^2}{S} F,\ \dot{v}|_\infty = 0\,. \quad (5)$$

Вычисление сил, действующих на корпус виброробота со стороны вязкой жидкости, в безразмерной постановке проводится по формуле:

$$\boldsymbol{F_p} = \int_\Omega p\boldsymbol{n}ds - \int_\Omega \bar{\boldsymbol{\sigma}} \cdot \boldsymbol{n}ds, \quad (6)$$

где $\bar{\boldsymbol{\sigma}}$ – тензор вязких напряжений, Ω – поверхность виброробота, $\boldsymbol{n}$ – внешняя нормаль к поверхности виброробота.

В силу ограничений рассматриваемой модели взаимодействия (предполагающей только прямолинейное движение системы в жидкости), в уравнение (5) войдет только продольная составляющая найденной силы F_x ($\boldsymbol{F_p} = (F_x, F_y)$), из которой необходимо вычесть дополнительный вклад, связанный с переходом в подвижную систему координат. Этот вклад определяется как:

$$F_{fk} = \int_S x\dot{w}nds. \quad (7)$$

С учетом (7) условие на бесконечности (5) может быть переписано в виде:

$$\dot{u}|_\infty = \mu_2 \dot{u}_m - \mu_1 \frac{R^2}{S}(F_x - F_{fk})\,. \quad (8)$$

Система уравнений (3), (4), (8) полностью описывает движение виброробота. Предполагая малые (микро) размеры моделируемого устройства, ограничимся далее диапазоном низких чисел Рейнольдса, в котором справедлива гипотеза о плоском ламинарном течении вокруг корпуса робота.

3. Численная модель

3.1 Дискретизация

Численное решение задачи проводится в пакете OpenFOAM. Плоскость течения ограничивается прямоугольной областью размерами 50×30, в центре которой находится корпус виброробота – равносторонний треугольник. В используемой декартовой системе координат стороны расчетной области параллельны основным осям, колебания системы происходят вдоль оси Ox.

Для дискретизации расчетной области используются структурированные блочные сетки. Для повышения разрешающей способности сеток в окрестности корпуса выполняется сгущение узлов. Максимальное количество узлов, используемых расчетных сеток, составляет $2.9 \cdot 10^5$.

Дискретизация системы уравнений движения жидкости проводится по методу конечных объемов (FVM) в декартовой системе координат. Дискретные

значения составляющих скорости и дискретные давления локализуются в центрах ячеек расчетных сеток. Для вычисления объемных интегралов по контрольному объему используется общая процедура Гаусса. Для аппроксимации градиента давления в расчетах применяется линейная интерполяция. Для интерполяции переменных в конвективных слагаемых используется нелинейная NVD (normalised variable diagram) схема «Gamma», предложенная в работе [18]. В диффузионных слагаемых при дискретизации оператора Лапласа нормальные градиенты скорости на поверхности ячейки аппроксимируются с помощью симметричной схемы второго порядка с поправкой на неортогональность [19].

3.2 Итерационная схема

Решение дискретизованной задачи проводится на основе метода PISO [20]. На каждой временной итерации алгоритма определяются дополнительные шаги, отвечающие за обновление граничных условий (8). Обновление проводится по схеме предиктор с отложенной коррекцией. Результирующая итерационная схема для вычисления значений дискретных неизвестных на j-ом временном слое может быть представлена в следующем виде:

- Вычисляется предиктор для ускорения подвижной системы координат

$$\dot{w}_p^j = 2\,\dot{w}^{j-1} - \dot{w}^{j-2}.$$

- Определяются граничные условия на входной и выходной границах по формуле (9), где ускорение вычисляется как сумма предиктора и корректора для старого временного слоя, а скорость находится с помощью направленной разности второго порядка точности:

$$\dot{u}_\infty^j = -\dot{w}_p^j + \dot{w}_c^{j-1},\ u_\infty^j = (-2\dot{u}_\infty^j dt + 4u_\infty^{j-1} - u_\infty^{j-2})/3$$

- Проводится решение уравнений движения жидкости (3) по методу PISO, вычисляется сила F_x, действующая со стороны жидкости на виброробот.

- По найденной силе вычисляется реальное ускорение системы:

$$\dot{w}^j = -\mu_2\,\dot{u}_m{}^j + \mu_1 \frac{R^2}{S} F^j$$

- Вычисляется корректор:

$$\dot{w}_c^j = \dot{w}_p^j - \dot{w}^j.$$

Для решения системы уравнений для давления применяется метод сопряженных градиентов (PCG) с геометро-алгебраическим многосеточным предобуславливателем (GAMG). В реализации GAMG используется метод Гаусса-Зейделя с одной предрелаксацией и двумя пострелаксациями для сглаживания и алгоритм faceAreaPair [21] для агломерации ячеек сетки. Системы уравнений для компонент скорости решаются методом бисопряженных градиентов (PBiCG) с предиктором на основе неполной LU

факторизации. Вычисления выполняются распределенным образом по технологии MPI на основе метода декомпозиции расчетной области.

3.3 Апробация

Апробация численной схемы проводилась на базе задачи о гармонических колебаниях треугольного цилиндра в покоящейся жидкости [22]. Результаты численного моделирования показали хорошее согласование с экспериментальными данными.

4. Результаты

В рамках численного моделирования рассматривалось движение виброробота при простом гармоническом законе колебания внутренней массы и специальном двухфазном (см. рис. 1). Расчеты в работе выполнялись в диапазоне чисел Рейнольдса $50 < \mathrm{Re} < 270$ при $\mu_1 = 0.06$, $\mu_2 = 0.61$ и условии максимально допустимого размаха колебаний внутренней массы A, равного безразмерной высоте треугольника $H = \sqrt{0.75}$. Вычисления проводились с разных начальных приближений, определяемых различной начальной скоростью набегающего потока. Для определения устойчивости режимов в течение вносились возмущения по методу Мартинеса [23].

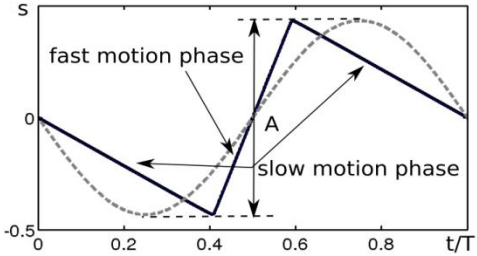


Рис.1. Перемещение внутренней массы за период при гармоническом и двухфазном законах движения.

Fig1. The movement of internal mass during one period according to the harmonic and two-phase laws of motion.

4.1 Гармонический закон движения

При гармоническом законе движения внутренней массы разность сил на прямой и возвратной фазах движения, действующих на внешний корпус со стороны жидкости, необходимая для движения системы как целого, достигается за счет несимметричного относительно полупериода обтекания корпуса. В исследуемом диапазоне чисел Рейнольдса было обнаружено три основных устойчивых режима движения, каждый из которых характеризуется своей структурой течения вокруг корпуса. Визуализация картин течения в

этих режимах за один период колебания внутренней массы представлена на рис. 2. Изображения получены путем подкрашивания жидкости, вытекающей из пограничного слоя корпуса. Внизу каждого рисунка представлено положение внутренней массы в заданный момент времени.

В зоне малых чисел Рейнольдса ($\mathrm{Re} < 170$) наблюдается единственный периодический режим движения Н. Перемещение робота в этом режиме осуществляется вперед вершиной. Обтекание корпуса имеет симметричный характер. Каждый полупериод при движении вперед вершиной с углов корпуса сбрасываются 2 симметричных противоположно вращающихся вихря, которые диссипируют при столкновении с корпусом на возвратной фазе. Таким образом, все нестационарное вихревое движение происходит в малой окрестности корпуса.

При $\mathrm{Re} > 170$ одновременно с базовым режимом Н появляется второй устойчивый режим движения робота Н1. В отличие от базового режима перемещение в жидкости здесь реализуется вперед основанием. Таким образом, возникает гистерезис режимов движения. Переход в тот или иной режим зависит от начальных параметров движения робота: при положительных значениях начальной средней скорости робота устанавливается режим Н, при отрицательных значениях – режим Н1. Структура течения жидкости в окрестности корпуса остается очень похожей на наблюдаемую в режиме Н. Возможность движения в противоположном направлении, вероятно, обусловлена вихревыми структурами вокруг корпуса, которые в силу небольшой средней скорости движения (см. пункт 4.3) не сносятся во внешнюю область течения и снижают сопротивление корпуса.

Режим Н2 приходит на смену режиму Н при $\mathrm{Re} \approx 240$. Направление движения робота при этом сохраняется, однако структура течения вокруг корпуса претерпевает существенные изменения. Разрушается симметрия течения, появляется квазипериодичность. Сброс вихревых структур в режиме Н2 происходит исключительно с одного угла треугольного цилиндра. На рисунке 2 представлен режим с верхним сбросом. Аналогичным образом реализуется режим движения с нижним сбросом вихрей, общая картина течения при этом зеркально отображается по горизонтали. Разрушение симметрии течения приводит к появлению подъемной силы и крутящего момента, влияние которых не учитывается в рамках рассматриваемой модели взаимодействия. По этой причине дальнейшее исследование развития этого типа течений ограничивается.

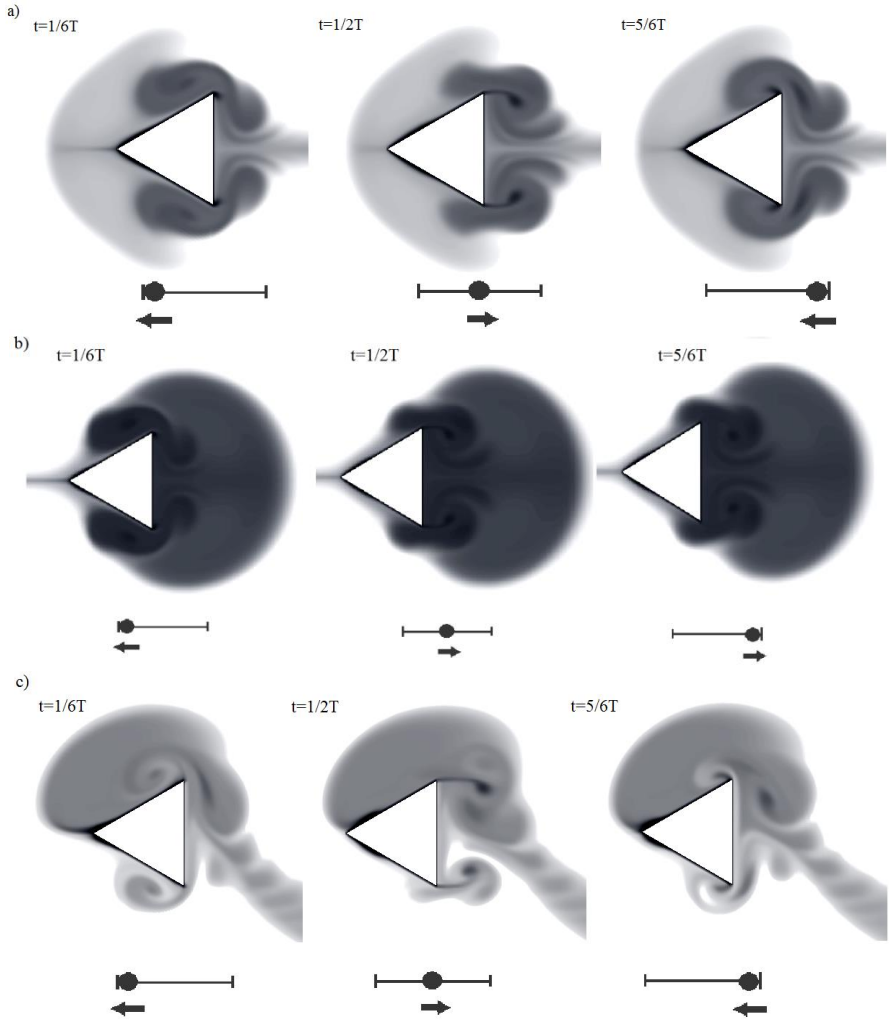


Рис. 2. Мгновенные картины течения около корпуса виброробота при гармонических режимах движения за один период колебания внутренней массы.a) Режим H Re=180,b) Режим H1 Re=180,c) Режим H2 Re=240

Fig. 2. Instantaneous flow patterns around the shell of the vibration-driven robot for the harmonic regimes of motion during one period of the internal mass oscillations. .a) Regime H Re=180,b) Regime H1 Re=180,c) Regime H2 Re=240

4.2 Двухфазный закон движения

Структура рассматриваемого двухфазного закона представлена на рис. 1. Как видно полный период движения стоит из двух основных фаз:

продолжительной фазы медленного движения и короткой фазы быстрого движения, во время которых внутренняя масса движется с постоянной скоростью. Переход между фазами реализуется на коротком временном интервале с большим ускорением, продолжительность которого составляет менее 2% от полного периода движения. Основная идея этого закона состоит в том, что во время продолжительной фазы корпус виброробота, двигаясь в направлении противоположном движению внутренней массы, за счет малой скорости движения должен испытывать меньшее сопротивление жидкости, чем на быстрой возвратной фазе, когда за счет нелинейной зависимости сопротивления от скорости, силы противодействующие движению робота существенно возрастают. Оптимальность этого класса законов была доказана в работах [12-14] на базе нескольких упрощенных моделей взаимодействия жидкости с виброробота.

Как показывают исследования, перемещение робота в жидкости за период при двухфазном законе всегда происходит по направлению движения корпуса в продолжительной медленной фазе. Таким образом, переключения направления можно добиться только путем смены знака управляющего закона. Более того, движение робота вперед вершиной (режим T) здесь имеет существенные преимущества перед движением вперед основанием (режим T1). Тестовые расчеты, проведенные для Re= 180, показывают, что средняя скорость у этих режимов отличается более чем на 50% в пользу режима T. Это объясняется отличным, по сравнению с гармоническими режимами, взаимодействием с окружающей робот вязкой жидкостью.

Картины течения при движении вперед вершиной и вперед основанием представлены на рис. 3. В обоих случаях обтекание корпуса происходит симметрично относительно оси колебания. При этом формирующиеся в течении вихревые структуры располагаются большей частью за корпусом относительно направления перемещения робота. Таким образом, на прямой фазе движения корпус взаимодействует непосредственно с невозмущенной жидкостью. Поэтому в режиме движения T1 робот испытывает (как для случая простого обтекания (см. например [24])) большее сопротивление, чем в режиме T.

С ростом чисел Рейнольдса отрыв вихрей с корпуса робота становится несимметричным, что сопряжено с переключением режимов движения. Режим T теряет устойчивость в пользу квазипериодического режима T2 (см. визуализацию течения на рис. 3) в окрестности Re=225. Исследование режимов с несимметричным обтеканием, как отмечалось выше, ограничено возможностями представленной модели.

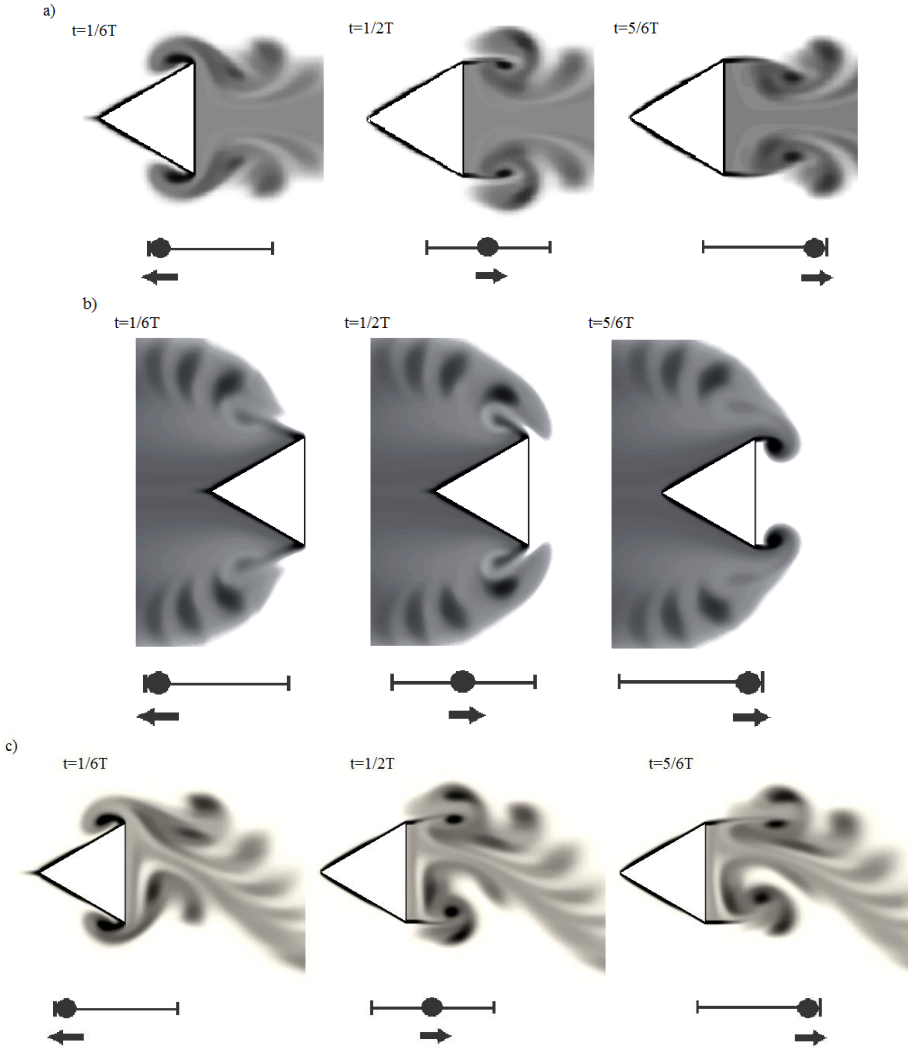


Рис 3. Мгновенные картины течения около корпуса виброробота при двухфазных режимах движения за один период колебания внутренней массы.a) Режим T Re=180, b) Режим T1 Re=180, c) Режим T2 Re=240

Fig. 3. Instantaneous flow patterns around the shell of the vibration-driven robot for the two-phase regimes of motion during one period of the internal mass oscillations. .a) Regime T Re=180,b) Regime T1 Re=180,c) Regime T2 Re=240

4.3 Анализ и сравнение характеристик режимов движения

В качестве основных характеристик движения рассматривались: средняя скорость движения U_a и показатель эффективности движения η, который отражает энергетические затраты на движение тела с помощью внутреннего движителя. Обозначая угловыми скобками, среднее по периоду, определим их следующим образом:

$$U_a = < U >,\ \eta = \frac{N_0}{N_{vbr}} 100\%$$

Здесь $N_0 = N(<U>)$ – минимальная мощность, необходимая для движения тела со скоростью U_a, $N_{vbr} = N(U)$ – мощность, затрачиваемая при движении виброробота с этой скоростью.

Графики изменения характеристик движения виброробота с ростом числа Рейнольдса при разных законах движения изображены на рисунке 4. Для двухфазного закона колебания представлены только наиболее эффективные режимы T и T2.

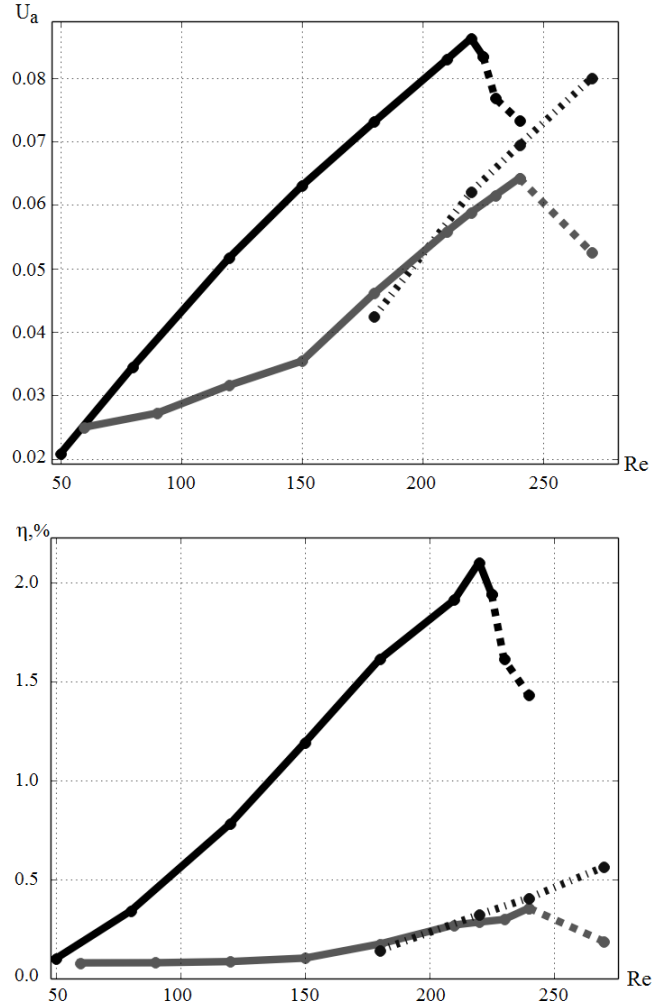


Рис.4. Зависимости средней скорости (верхнее изображение) и эффективности (нижнее изображение) движения от числа Рейнольдса. Черная сплошная линия – режим T, черная штриховая – T2, серая сплошная – H, , серая штриховая – H2, черная пунктирная – режим H1.

Fig. 4. Dependencies of average speed (top image) and efficiency (lower image) of movement on the Reynolds number. The black solid line - T regime, the black dashed - T2, solid gray - H,, gray dashed - the H2, the black dotted - H1 regime.

Во всех режимах с симметричным обтеканием (H, H1, T) показатели средней скорости и эффективности растут с увеличением числа Рейнольдса.

Гармонические режимы при этом дают существенно более низкие значения по сравнению с двухфазными. Более того сравнение результатов между режимами H и H1 показывает, что при гармонических колебаниях режим движения вперед основанием имеет схожую эффективность с режимом движения вперед вершиной. Разрушение симметрии течения, наблюдаемое в режимах H2, T2, сопряжено с кризисом по основным показателям движения, который особенно хорошо заметен на графике изменения коэффициента эффективности (рис 4.). Максимальная эффективность движения достигается в режиме T при Re=220 и составляет $\eta = 2.1\%$.

5. Выводы

Результаты исследования показывают, что направленное движение виброробота с клиновидной формой корпуса реализуется как для гармонического, так и для двухфазного законов. При этом для случая гармонического закона ненулевая разность суммарных сил сопротивления на прямой и возвратной фазах движения обеспечивается за счет несимметричного относительно полупериода обтекания корпуса. Более того движение системы в этом случае, при одинаковых параметрах колебания внутренней массы, возможно как вперед основанием так и вперед вершиной, выбор направления движения определяется начальными условиями процесса. Двухфазный закон колебания внутренней массы обеспечивает единственное возможное направление движения системы. Перемещение робота в жидкости за период происходит по направлению движения корпуса в продолжительной медленной фазе. Максимальные характеристики достигаются при перемещении робота вперед вершиной. Сравнение результатов моделирования для разных законов движения внутренней массы показывает, что двухфазный закон обеспечивает более высокую скорость движения (до 50%) в жидкости, а также является значительно более эффективным (до 3 раз) с точки зрения энергозатратности.

Работа выполнена при поддержке грантов РФФИ 16-31-00462 (мол_а) и РНФ 15-19-10039. Численная схема разработана в ННГУ им. Н.И. Лобачевского.

Список литературы

[1]. Черноусько Ф.Л. Оптимальные периодические движения двухмассовой системы в сопротивляющейся среде. ПММ, 2008, т. 72, вып. 2, с. 202-215.

[2]. Болотник Н.Н., Фигурина Т.Ю., Черноусько Ф.Л. Оптимальное управление прямолинейным движением системы двух тел в сопротивляющейся среде. ПММ, 2012, т. 76, вып. 1, с. 3-22.

[3]. Lighthill M.J. On the Squirming Motion of Nearly Spherical Deformable Bodies through Liquids at Very Small Reynolds Numbers. Comm. Pure Appl. Math, 5(2), 1952, pp. 109–118.

[4]. Saffman P.G. The Self-Propulsion of a Deformable Body in a Perfect Fluid. J. Fluid

Mech., 28(2), 1967, pp. 385–389.
[5]. Ramodanov S.M., Tenenev V.A., Treschev D.V. Self-propulsion of a Body with Rigid Surface and Variable Coefficient of Lift in a Perfect Fluid. Regul. Chaotic Dyn, 17(6), 2012, pp. 547–558.
[6]. Черноусько Ф. Л. О движении тела, содержащего подвижную внутреннюю массу. Докл. РАН, 2005, т. 405, вып. 1, с. 56–60.
[7]. Черноусько Ф.Л. Анализ и оптимизация движения тела, управляемого посредством подвижной внутренней массы. ПММ, 2006, т.70, вып. 6, с. 915–941.
[8]. Волкова Л.Ю., Яцун С.Ф. Управление движением трехмассового робота, перемещающегося в жидкой среде. Нелинейная динамика, 2011, т.7, вып. 4, с. 845–857.
[9]. Childress S., Spagnolie S.E., Tokieda T.A. Bug on a Raft: Recoil Locomotion in a Viscous Fluid. J. Fluid Mech., 669, 2011, pp. 527–556.
[10]. Auziņš J., Beresņevičs V., Kaktabulis I., Kuļikovskis G. Dynamics of Water Vehicle with Internal Vibrating Gyrodrive. Vibration Problems ICOVP 2011. Supplement: The 10th International Conference on Vibration Problems, Czech Republic, Prague, 5-8 September.
[11]. Vetchanin E. The Self-propulsion of a Body with Moving Internal Masses in a Viscous Fluid. Regular and Chaotic Dynamics, 18, 2013, pp. 100–117.
[12]. Егоров А.Г., Захарова О.С. Оптимальное по энергетическим затратам движение виброробота в среде с сопротивлением. ПММ, 2010, т.74, вып. 4, с. 620–632.
[13]. Егоров А.Г., Захарова О.С. Оптимальное квазистационарное движение виброробота в вязкой жидкости. Известия ВУЗов. Математика, 2012, вып. 2, с. 57–64.
[14]. Егоров А.Г., Захарова О.С. Энергетически оптимальное движение виброробота в среде с наследственным законом сопротивления . Известия РАН. Теория и системы управления, 2015, № 3, с. 168-176.
[15]. Нуриев А.Н., Захарова О.С. Численное моделирование движения клиновидного двухмассового виброробота в вязкой жидкости. Вычислительная механика сплошных сред, 2016, Т. 9., № 1., С. 5-15.
[16]. Open foam (the open source cfd toolbox): User guide version 2.2.1, Доступно по ссылке: http://www.openfoam.org/docs/user/, 2.07.2016.
[17]. Unihub.ru, Доступно по ссылке: https://unihub.ru/about, 2.07.2016.
[18]. Jasak H., Weller H.G., Gosman A D. High resolution NVD differencing scheme for arbitrarily unstructured meshes. Int. J. Numer. Meth. Fluids, 1999, vol.31, pp. 431-449.
[19]. Jasak H. Error analysis and estimation for the finite volume method with applications to fluid flows. Ph.D. thesis, London: Imperial College, University of London, 1996.
[20]. Issa R.I. Solution of implicitly discretised fluid flow equations by operator-splitting. J. Comput. Phys., 1986, vol. 62, pp. 40–65.
[21]. Behrens T. Openfoam's basic solvers for linear systems of equations, Technical Report, Technical University of Denmark, Lingby Доступно по ссылке: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/TimBehrens/tibeh-report-fin.pdf, 10.10.2016.
[22]. Нуриев А.Н., Зайцева О.Н., Мощева Е.Е., Юнусова А.И. Исследование структуры вторичных течений вокруг треугольного цилиндра, совершающего гармонические колебания в вязкой несжимаемой жидкости. Вестник Казанского технологического университета,. 2015, Т. 18, № 16. С. 239-242.
[23]. Martinez G. Caractristiques dynamiques et thermiques de l'coulement autour d'un

cylindre circulaire a nombres de reynolds moderes. Ph.D. thesis,I.N.P. Toulouse, 1979

[24]. De A. K., Dalal A. Numerical simulation of unconfined flow past a triangular cylinder. Int. J. Numer. Meth. Fluids, 2006, No. 52, p. 801-821.

Simulation of the wedge-shaped vibration-driven robot motion in the viscous fluid forced by different laws of internal mass movement in the package OpenFOAM

[1,2] *A.N. Nuriev <nuriev_an@mail.ru>*
[3] *A.I. Yunusova < yunusova24@gmail.com >*
[2] *O.N. Zaitseva <olga_fdpi@mail.ru >*
[1] *Nizhny Novgorod State University,*
23, pr. Gagarina, Nizhny Novgorod, 603022, Russia
[2] *Kazan Federal University,*
18, str. The Kremlin, Kazan, Republic of Tatarstan, 420008, Russia
[3] *Kazan State Technological University*
68, str. Marx, Kazan, Republic of Tatarstan, 420015, Russia

Abstract. The work is devoted to the study of the two-mass vibration-driven system motion in the viscous fluid. The system consists of a closed wedge-shaped body, placed in a fluid, and a movable internal mass, oscillated harmonically inside the shell. The described mechanical system simulates a vibration-driven robot. The complex model of the robot interaction with the medium is considered, where fluid motion is described by the full unsteady Navier-Stokes equations. The problems of improving the efficiency of vibration-driven robot motion by choosing a special law internal mass movement are investigated. For these purposes, a comparative analysis of the characteristics of the motion and flow regimes around the robot are carried out for the simple harmonic law of the internal mass motion and the special two-phase law of the internal mass motion. The analysis of the flows around the robot and their influence on the characteristics (the average speed and the efficiency) of the movement is carried out. The numerical solution of the problem is carried out in the OpenFOAM open-source software package. The numerical scheme is implemented on the basis of the finite-volume discretization approach. For joint solution the Navier-Stokes equations and the mechanical system, which describes the interaction of components of vibration-driven robot and viscous media, a special iteration scheme is constructed. Results of the study show that the directional movement of the wedge-shaped vibration-driven robot is possible for both harmonic and two-phase laws of internal mass motion. In each of the cases it is possible to find stable regimes of motion observed in a wide range of Reynolds numbers. Analysis of average speed and efficiency of regimes allows finding the optimal parameters of vibration-driven robot motion.

Keywords: Vibration-driven robot; Viscous fluid; Navier-Stokes equations; Motion regimes; Motion efficiency; OpenFOAM.

DOI: 10.15514/ISPRAS-2017-29(1)-7

For citation: Nuriev A.N., Yunusova A.I., Zaitseva O.N. Simulation of the wedge-shaped vibration-driven robot motion in the viscous fluid forced by different laws of internal mass movement in the package OpenFOAM. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 1, 2017, pp. 101-118 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-7

References

[1]. Chernous'ko F. L. The optimal periodic motions of a two-mass system in a resistant medium. PMM [J. Appl. Math. Mech.], 72, 2008, pp. 202-215 (in Russian).

[2]. Bolotnik N.N., Figurina T.Y., Chernous'ko F.L. Optimal control of the rectilinear motion of a two-body system in a resistive medium. PMM [J. Appl. Math. Mech], 76, 2012, pp. 3-22 (in Russian).

[3]. Lighthill M.J. On the Squirming Motion of Nearly Spherical Deformable Bodies through Liquids at Very Small Reynolds Numbers. Comm. Pure Appl. Math, 5(2), 1952, pp. 109–118.

[4]. Saffman P.G. The Self-Propulsion of a Deformable Body in a Perfect Fluid. J. Fluid Mech., 28(2), 1967, pp. 385–389.

[5]. Ramodanov S.M., Tenenev V.A., Treschev D.V. Self-propulsion of a Body with Rigid Surface and Variable Coefficient of Lift in a Perfect Fluid. Regul. Chaotic Dyn., 17(6), 2012, pp. 547–558.

[6]. Chernous'ko F.L. On the motion of a body containing a movable internal mass. Dokl. RAN [Dokl. Phys.], 450(1), 2005, pp. 56-60 (in Russian).

[7]. Chernous'ko F.L. Analysis and optimization of the motion of a body controlled by means of a movable internal mass. PMM [J. Appl. Math. Mech.], 70(6), 2006, pp. 915-941 (in Russian).

[8]. Volkova L.Yu., Jatsun S.F. Control of the Three-Mass Robot Moving in the Liquid Environment. Nelinejnaja dinamika [J. Nonlin. Dyn.], 7(4), 2011, pp. 845–857 (in Russian).

[9]. Childress S., Spagnolie S.E., Tokieda T.A. Bug on a Raft: Recoil Locomotion in a Viscous Fluid. J. Fluid Mech., 669, 2011, pp. 527–556.

[10]. Auziņš J., Beresņevičs V., Kaktabulis I., Kuļikovskis G. Dynamics of Water Vehicle with Internal Vibrating Gyrodrive. Vibration Problems ICOVP 2011. Supplement: The 10th International Conference on Vibration Problems, Czech Republic, Prague, 5-8 September.

[11]. Vetchanin E. The Self-propulsion of a Body with Moving Internal Masses in a Viscous Fluid. Regular and Chaotic Dynamics, 18, 2013, pp. 100–117.

[12]. Egorov A.G., Zakharova O.S. The energyoptimal motion of a vibrationdriven robot in a resistive medium. PMM [J. Appl. Math. Mech.], 74(4), 2010, pp. 620-632 (in Russian).

[13]. Egorov A.G., Zakharova O.S. Optimal quasistationar motion of vibrationdriven robot in a viscous liquid. Izvestija VUZov. Matematika [Russian Mathematics (Iz. VUZ)], 2, 2012, pp. 57–64 (in Russian).

[14]. Egorov A.G., Zakharova O.S. The Energy-Optimal Motion of a Vibration-Driven Robot in a Medium with a Inherited Law of Resistance. Izvestija RAN. Teorija i sistemy upravlenija [J. of Computer and Systems Sciences International], 3, 2015, pp. 168-176 (in Russian).

[15]. Nuriev A. N., Zakharova O.S. Simulation of the wedge-shaped two-mass vibration-driven robot motion in a viscous fluid. [Computational Continuum Mechanics], 9, 2016, pp. 5-15 (in Russian).
[16]. Open foam (the open source cfd toolbox): User guide version 2.2.1, URL: http://www.openfoam.org/docs/user/, 2.07.2016.
[17]. Unihub.ru, Available at: https://unihub.ru/about, accessed 2.07.2016.
[18]. Jasak H., Weller H.G., Gosman A D. High resolution NVD differencing scheme for arbitrarily unstructured meshes. Int. J. Numer. Meth. Fluids, 1999, vol.31, pp. 431-449.
[19]. Jasak H. Error analysis and estimation for the finite volume method with applications to fluid flows. Ph.D. thesis, London: Imperial College, University of London, 1996.
[20]. Issa R.I. Solution of implicitly discretised fluid flow equations by operator-splitting. J. Comput. Phys., 1986, vol. 62, pp. 40–65.
[21]. Behrens T. Openfoam's basic solvers for linear systems of equations. Technical Report, Technical University of Denmark, Lingby Available at: http://www.tfd.chalmers.se/~hani/kurser/OS_CFD_2008/TimBehrens/tibeh-report-fin.pdf, accessed 10.10.2016.
[22]. Nuriev A. N., Zaytseva O.N., Moscheva E.E., Yunusova A.I. Structure of secondary flow around cylinder triangular, performs harmonic oscillations in a viscous incompressible fluid. Vestn. Kaz. tehnologicheskogo universiteta
[Heald of Kazan Technological University], 16, 2015, pp. 239-242 (in Russian).
[23]. Martinez G. Caractristiques dynamiques et thermiques de l'coulement autour d'un cylindre circulaire a nombres de reynolds moderes. Ph.D. thesis,I.N.P. Toulouse, 1979
[24]. De A. K., Dalal A. Numerical simulation of unconfined flow past a triangular cylinder. Int. J. Numer. Meth. Fluids, 2006, No. 52, p. 801-821.

Проведение итеративного динамического анализа приложений, предоставляющих графический интерфейс пользователя[1]

М. К. Ермаков <mermakov@ispras.ru>
А. Ю. Герасимов <agerasimov@ispras.ru>
Д. О. Куц <kutz@ispras.ru>
А. А. Новиков <a.novikov@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В настоящее время в промышленной разработке программного обеспечения с графическим пользовательским интерфейсом преобладают полуавтоматические подходы к тестированию, требующие участия эксперта для создания наборов тестовых сценариев. Повышение сложности программных систем приводит к снижению эффективности применения методов, полагающихся на участие эксперта в процессе тестирования. С учётом возрастания доступности и снижения стоимости вычислительных ресурсов становятся экономически выгодны методы автоматического анализа программ. В рамках данной статьи предлагается метод полностью автоматического динамического анализа программ, предоставляющих графический пользовательский интерфейс. Среди существующих инструментов тестирования и анализа программного обеспечения на основе обзора, приведённого в статье, выделяется свободно распространяемое программное инструментальное средство GUITAR, обеспечивающее максимальную степень автоматизации. Рассматриваются основные ограничения подхода, реализованного в средстве GUITAR: недостаточная степень точности модели графического интерфейса, недостаточность полноты описания атрибутов элементов графического интерфейса. Данные ограничения приводят к невозможности покрытия наборами тестовых воздействий отдельных элементов графического интерфейса в рамках анализа и созданию тестовых сценариев, которые не могут быть воспроизведены на практике. В статье предлагается ряд модификаций подхода: итеративное построение модели графического интерфейса, расширение списка атрибутов элементов графического интерфейса, алгоритм итеративного построения тестовых наборов по графу потока событий с целью явной проверки функциональности всех элементов графического интерфейса. В статье рассмотрены результаты практических экспериментов применения предложенных модификаций для набора проектов с открытым исходным кодом, демонстрирующие

[1] Работа проводится в рамках научно-исследовательских работ Института системного программирования РАН в 2014 – 2017 годах

повышение эффективности анализа и полноты покрытия графического интерфейса создаваемыми тестовыми наборами. В заключение статьи обсуждаются перспективные направления дальнейшей работы, включающие применение методов символьного исполнения и анализа помеченных данных.

Ключевые слова: динамический анализ программ; анализ программ; тестирование GUI, тестовое покрытие.

DOI: 10.15514/ISPRAS-2017-29(1)-8

Для цитирования: Ермаков М.К., Герасимов А.Ю., Куц Д.О., Новиков А.А. Проведение итеративного динамического анализа приложений, предоставляющих графический интерфейс пользователя. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 119-134. DOI: 10.15514/ISPRAS-2017-29(1)-8

1. Введение

В повседневной жизни современному человеку всё чаще приходится сталкиваться с использованием высокотехнологичных устройств как для выполнения работы, так и для общения, отдыха и развлечений. Ключевым элементом, обеспечивающим разнообразие функциональных возможностей высокотехнологичных устройств, является программное обеспечение, под управлением которого работает устройство. Во избежание экономического и материального ущерба, а также для обеспечения безопасности человека при использовании устройств, предъявляются высокие требования к качеству программного обеспечения. Стоимость исправления ошибки в программном обеспечении значительно возрастает при приближении к концу жизненного цикла. В связи с этим разработка средств обнаружения ошибок на ранних этапах разработки программ является крайне актуальной задачей.

Анализ программного кода на наличие ошибок и уязвимостей может производиться как вручную, что требует большого количества времени и трудозатрат, так и полуавтоматически и автоматически. Наиболее перспективным подходом является применение методов и технологий, позволяющих частично или полностью автоматизировать отдельные этапы процесса анализа программ. Среди данных подходов можно выделить группу методов динамического анализа, основывающихся на исследовании программного обеспечения во время его выполнения. Данная группа методов нацелена на построение и проверку сценариев использования программного обеспечения, близких к ожидаемым от конечных пользователей. Одной из основных проблем подобных методов является необходимость обеспечить достаточную полноту покрытия кода программы минимальным набором сценариев работы с программой. В рамках задачи автоматизации проверки программ можно выделить полуавтоматические методы, основанные на участии эксперта в создании базы сценариев для проверки, и полностью автоматические методы, ориентированные на извлечение информации о

структуре приложения с целью генерации тестовых сценариев без участия эксперта.

В то же время, в связи с бурным развитием устройств с возможностью предоставления графического пользовательского интерфейса, особый практический интерес при разработке программного обеспечения представляет анализ поведения именно таких программ. Ошибки проектирования и нарушение корректности работы обработчиков событий от элементов графического пользовательского интерфейса могут приводить к невозможности использования целевой функциональности программ, а тестирование и анализ непосредственно самого графического интерфейса становятся необходимым этапом при создании программных продуктов.

Для проверки качества реализации программ с графическим пользовательским интерфейсом традиционно используются инструменты автоматизации анализа и тестирования, которые ограничиваются тем, что исключают необходимость вручную взаимодействовать с элементами графического интерфейса. Как будет показано в обзоре области, представленном в следующей главе, данные инструменты фокусируются на точности определения соответствия между элементами, указанными в сценарии тестирования, и реальными элементами, отображаемыми в процессе работы целевой программы.

Данная работа нацелена на развитие методов полностью автоматического динамического анализа программ, предоставляющих графический интерфейс пользователя, позволяющих минимизировать степень участия эксперта в процессе создания набора тестовых сценариев, сохраняя при этом полноту покрытия программы.

Далее статья имеет следующую структуру — в разд. 2 представлен обзор существующих инструментов полуавтоматического и автоматического анализа программ, предоставляющих графический пользовательский интерфейс. Разд. 3 описывает общие принципы работы одного из наиболее перспективных инструментов тестирования программ с графическим пользовательским интерфейсом (GUITAR), выбранного в рамках данной работы в качестве основы для реализации. Разд. 4 описывает модификации, внесенные в инструмент GUITAR в рамках данной работы. Разд. 5 посвящен описанию эксперимента и оценке практических результатов применения модифицированной версии инструмента GUITAR для ряда свободно распространяемых приложений. В заключении приводится общая оценка проделанной работы и рассматриваются наиболее перспективные направления дальнейших исследований.

2. Обзор существующих решений для анализа программ с графическим пользовательским интерфейсом

На сегодняшний день существует множество инструментов, осуществляющих анализ приложений с графическим интерфейсом. Большинство из них

основано на полуавтоматическом подходе, который заключается в записи действий пользователя с приложением и последующем их воспроизведении. Инструмент производит запуск исследуемого приложения, регистрирует элементы графического интерфейса и взаимодействие между ними, в то время как пользователь производит некоторые действия с приложением, такие как нажатие кнопок, ввод данных в текстовые поля и пр. На основе собранной информации о структуре графического интерфейса приложения пользователю предлагается возможность составить последовательности действий над выделенными графическими элементами, а также программный интерфейс, с помощью которого можно описывать более сложные тестовые сценарии.

Также некоторые современные средства предлагают возможности, в частности для Java-приложений, модульного тестирования (unit testing) для проверки корректности сценариев работы графического приложения. Очевидно, что такой подход также позволяет сократить затраты на построение тестового покрытия – сценарии работы отдельного графического элемента можно непосредственно использовать при составлении сценария работы приложения.

В качестве примеров средств, предлагающих пользователю описанные выше возможности проведения тестирования, можно привести следующие инструменты.

Ranorex [1] – проприетарный инструмент автоматизации тестирования приложений, разработанных под операционную систему Windows. Ranorex позволяет работать с приложениями, основанными на распространенных для Windows библиотеках графического интерфейса, предоставляет возможности тестирования Java-приложений, основанных на библиотеке SWT. Также Ranorex поддерживает тестирование приложений на мобильных платформах, таких как Android и iOS. Инструмент представляет графический пользовательский интерфейс программы в виде дерева (леса), где корнем является главное окно (или несколько окон) программы, промежуточными узлами являются контейнеры элементов графического пользовательского интерфейса, а листьями — конечные элементы, такие как поля ввода, экранные кнопки и др., и использует XPath-подобный язык RanoreXpath для описания последовательностей взаимодействия с элементами GUI, а также предоставляет пользователям возможность описывать в виде программы базовые взаимодействия с приложением на языках C#, VB.net и IronPython.

Abbot [2] – среда автоматизации тестирования для Java-приложений с открытым исходным кодом. Данный инструмент позволяет динамически идентифицировать элементы графического приложения по набору атрибутов. Таким образом исключается привязка элемента к его положению в интерфейсе, что делает анализ менее зависимым от изменения положения элементов графического пользовательского интерфейса.

Maveryx [3] – это инструмент автоматизированного тестирования приложений с графическим интерфейсом, реализованных на языке Java, в том числе и для

платформы Android. Отличительной особенностью этого инструмента является то, что для создания и воспроизведения тестов не требуется модель, описывающая графические элементы. В Maveryx реализован принцип динамического определения структуры графического интерфейса приложения при воспроизведении каждого тестового сценария, что повышает гибкость и эффективность проведения тестирования.

Squish [4] – проприетарный инструмент автоматизации тестирования приложений с графическим интерфейсом для платформ Java AWT/Swing, SWT, Windows, Android и пр. Squish не требует модификации кода тестируемого приложения. Для тестирования приложения используется специальный управляющий модуль, который внедряется в адресное пространство приложения, данный модуль запускает приложение и связывается с инструментом. Squish имеет возможность проверки корректности выполнения тестов, сопоставление состояний некоторых виджетов производится с помощью анализа снимков экрана (screenshots).

Sikuli [5] – среда автоматизации тестирования приложений. Особенность данного инструмента заключается в предоставлении пользователю возможности строить тесты приложения, основываясь на изображениях графических элементов приложения. Для поиска и взаимодействия с элементами графического интерфейса используются их снимки (screenshots).

Инструмент GUITAR [6] выделяется среди всех рассмотренных решений тем, что позволяет проводить полностью автоматическое тестирование графических Java-приложений, основанных на AWT/Swing, SWT, а также приложений для платформы Android. GUITAR производит анализ структуры графического интерфейса на основе исполняемых файлов приложения и составляет модель данной структуры. На основе этой модели автоматически строится множество тестовых наборов – возможные последовательности действий определенной длины.

В то же время, у инструмента GUITAR есть недостатки, которые не позволяют производить анализ программы в полной мере. В главе 3 подробно рассматриваются устройство, особенности и ограничения инструмента GUITAR.

3. Особенности инструмента GUITAR

Инструмент GUITAR состоит из четырех компонентов, схема взаимодействия которых представлена на рис. 1.

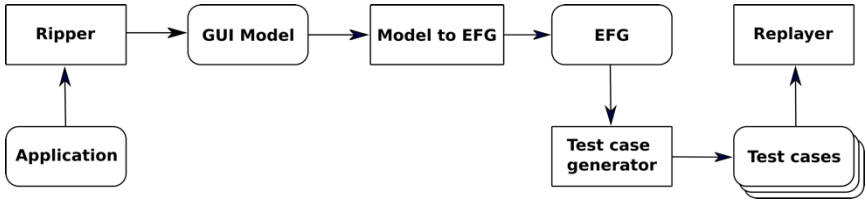


Рис. 1. *Схема работы GUITAR*

Fig. 1. *GUITAR tool scheme*

3.1 Алгоритм работы компонента Ripper

Компонент Ripper проводит анализ окон, открываемых приложением, и сохраняет информацию об элементах этих окон в формате XML. В каждом окне присутствует набор графических элементов. Все элементы объединены в общую иерархию — самым верхним уровнем является непосредственно само окно, далее лежат графические контейнеры верхнего уровня (такие как, например, группа вкладок). Данные контейнеры соответствуют базовым элементам библиотек построения графического интерфейса (например, Swing, SWT и др.). Содержимое каждого контейнера состоит из контейнеров более низкого уровня и конкретных графических элементов. В начальный момент времени для анализа доступно лишь главное окно приложения. Во время анализа окон Ripper производит для каждого видимого в данный момент графического элемента (кнопка, поле ввода, выпадающий список и т. д.) действие, соответствующее данному элементу (например, нажатие для кнопки). При проведении действий сохраняется информация об изменении графической структуры (например, открытие нового окна или закрытие текущего). В случае открытия новых окон при выполнении действия с некоторыми элементами проводится анализ данного окна.

3.2 Построение графа потока событий и тестового покрытия

Далее описание графической структуры, построенной компонентом Ripper, переводится в ориентированный граф потока событий (Event Flow Graph), который описывает взаимосвязи между отдельными графическими элементами на основе эффектов событий, соответствующих данным элементам. В частности, наличие ребра, следующего из события A в событие B, означает, что если в некоторый момент времени было совершено действие над элементом графического интерфейса, соответствующее событию A, то графическое приложение перешло в состояние, в котором возможно совершение действия, соответствующего событию B. На основе графа потока событий происходит генерация тестовых сценариев взаимодействия с программой. Каждый тестовый сценарий представляет собой последовательность событий. Компонент Replayer осуществляет

воспроизведение набора тестовых сценариев с целью обнаружения ошибок и проверки корректности работы программы.

3.3 Полнота анализа

Таким образом, полностью автоматически осуществляется генерация тестовых наборов и запуск приложения на этих наборах. Однако, как уже было сказано, использование инструмента GUITAR не позволяет добиться достаточной полноты покрытия. Рассмотрим простой пример, иллюстрирующий вышесказанное.

Предположим, одно из окон приложения имеет два переключателя и кнопку. Нажатие кнопки при активированном первом переключателе открывает диалоговое окно A, нажатие кнопки при активированном втором переключателе открывает диалоговое окно B. Во время анализа компонент Ripper совершает действия над элементами в той последовательности, в которой элементы доступны из данного окна. При этом каждое действие совершается один раз. Допустим, что элементы обрабатываются в следующей последовательности: переключатель №1, переключатель №2, кнопка. Тогда Ripper активирует сначала переключатель №1, затем переключатель №2 (при этом переключатель №1 деактивируется), затем произведет нажатие кнопки. Откроется диалоговое окно B, Ripper произведет анализ нового окна. Таким образом, сведения о существовании окна A и его элементах не будут доступны на последующих этапах анализа.

Данный пример характеризует основную проблему анализа, проводимого инструментом GUITAR, – ограниченность модели, соответствующей первичному запуску и разбору приложения. В качестве дополнительных недостатков можно выделить следующие:

- в процессе составления модели графического интерфейса программы для каждого активного элемента интерфейса и действия над данным элементом создается информация только о том, на какие окна приложения влияет данный элемент. В то же время действия над элементами могут не только закрывать и открывать новые окна, но и создавать, удалять, активировать и деактивировать отдельные графические элементы;
- при работе компонентов Ripper и Replayer используются простые эвристики для сопоставления окон приложения по их заголовкам. При использовании значительного количества диалоговых окон в программе механизм идентификации осуществляет сопоставление окон и модели некорректно.

Для исправления указанных ограничений были разработаны следующие модификации инструмента GUITAR, нацеленные на повышение полноты и точности анализа:

- итеративное построение и обновление модели графического интерфейса и графа потока событий;
- улучшение точности алгоритмов сопоставления элементов графического интерфейса с моделью при запусках тестовых наборов;
- расширение набора параметров отдельных элементов графического пользовательского интерфейса, описывающих эффект от взаимодействия с данными элементами на модель интерфейса.

4. Итеративный анализ программ с графическим пользовательским интерфейсом

Для реализации первого из указанных выше модификаций инструмента GUITAR была предложена альтернативная схема работы компонентов данного инструмента (рис. 2).

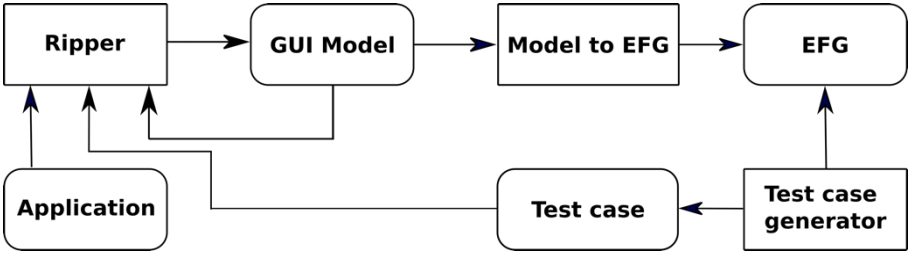


Рис. 2. *Изменение схемы работы GUITAR*

Fig. 2. *GUITAR scheme modification*

Компонент Replayer был исключен из модифицированной схемы взаимодействия инструмента GUITAR. Функциональность данного компонента, включающая возможности воспроизведения последовательности событий, была перенесена в компонент Ripper. Во время работы новой версии компонента Ripper при воспроизведении некоторой последовательности регистрируются все ранее не обнаруженные эффекты выполнения действий над элементами графического пользовательского интерфейса, и информация о них заносится в текущую модель структуры графического интерфейса. В том случае, когда при воспроизведении последовательности открывается ранее не проанализированное окно, компонент Ripper переключается в обычный режим и производит полный анализ данного окна.

Использование модифицированной схемы работы инструмента позволило также повысить точность определения параметров графических элементов и побочных эффектов, соответствующих действиям с данными элементами. Так, например, характеристики элемента, соответствующие возможным типам действий над этим элементом, определяются с большей точностью, чем ранее. В частности, элементы, которые позволяют осуществлять закрытие текущего

окна, могут быть однозначно определены. Ранее в инструменте GUITAR для определения подобных элементов использовалось следующее правило — если тип элемента и текст, соответствующий элементу (например, кнопка и заголовок кнопки) присутствуют в списке терминальных элементов, то действие, соответствующее данному элементу, вызывает закрытие текущего окна. Список терминальных элементов было необходимо задавать вручную перед началом анализа.

4.1 Изменение параметров модели графического пользовательского интерфейса.

Для эффективного применения итеративного анализа в модель графического пользовательского интерфейса программы были внесены следующие изменения.

- Во-первых, был изменен принцип идентификации различных окон. Ранее в инструменте GUITAR идентификация происходила исключительно по заголовку окна. Подобный способ идентификации не позволял эффективно различать окна в некоторых ситуациях (в частности, для стандартных диалоговых окон определённого типа, таких как сообщения об ошибке и информационные сообщения, заголовки которых совпадают). Для разрешения конфликтов в данной ситуации было принято решение проводить сравнение двух окон по заголовку и информации об элементах в составе данного окна. Эти параметры определяют идентификатор окна (к строке, составленной из заголовка и идентификаторов вложенных элементов, применяется хеш-функция, определяющая целочисленное значение идентификатора). Это изменение позволило добиться повышения полноты информации о графической структуре исследуемого приложения.
- Во-вторых, для графических элементов модели был расширен набор характеристик, описывающих эффекты взаимодействия с данными элементами. В частности, были добавлены следующие параметры.
    - Параметр DISABLE_LIST – список элементов, которые становятся неактивными после выполнения действия над ними. Неактивное состояние соответствует невидимым элементам (скрытым встроенными средствами взаимодействия с графическими элементами) или так называемым «выключенным» элементам, которые видимы, но недоступны пользователю. Неактивное состояние означает, что совершение действий над элементом невозможно.

  - Параметр ENABLE_LIST – список элементов, которые становятся активными после выполнения действия над ними.
  - Параметр DESTROY_LIST – список элементов, которые удаляются из содержащего их контейнера после выполнения действия над ними.
  - Параметр CREATE_LIST – список элементов, которые создаются и добавляются в один или различные контейнеры после выполнения действия над ними.
- Для существующих характеристик элементов модели графического интерфейса были произведены следующие модификации:
  - параметр EFFECT_TYPE, определяющий тип воздействия события, связанного с элементом, на общее состояние программы, был дополнен значением SYSTEM_EXIT, соответствующим попытке завершить работу приложения с помощью вызова функции `System.exit();`
  - возможные значения параметра INVOKE, определяющие список окон, которые открываются при совершении действия над элементом, были изменены со списка заголовков окон на список идентификаторов окон.

Дополнительно была изменена концепция работы с элементами, поддерживающими несколько возможных типов действий. В исходной версии инструмента GUITAR каждому элементу в модели графического интерфейса соответствовало единственное возможное действие. Для повышения полноты анализа был реализован механизм определения всех возможных действий, выполнение которых допустимо для элемента (например, длительное нажатие на кнопку для приложений на платформе Android, которое заменяет двойной щелчок). Во время анализа приложения компонентом Ripper происходит выполнение всех возможных действий для каждого элемента и в модели сохраняется информация о побочных эффектах для пар ⟨ элемент, тип действия⟩.

4.2 Направленная генерация тестовых сценариев

Для осуществления генерации тестовых сценариев в рамках схемы итеративного анализа с помощью инструмента GUITAR разработан модуль UndefinedComponentPath. Этот модуль предназначен для построения последовательностей событий на основе поиска «нерассмотренных элементов», т. е. тех элементов, для которых компонент Ripper не осуществил все возможные действия на предыдущих итерациях анализа. UndefinedComponentPath рассматривает в качестве входных данных граф потока событий. Соответственно, построение последовательностей для достижения «нерассмотренных элементов» сводится к задаче поиска путей в

графе до непосещенных вершин. В графе имеется набор начальных вершин, соответствующий действиям над элементами, которые можно произвести непосредственно после запуска приложения. Искомый путь для достижения «нерассмотренных элементов» должен исходить из одной из начальных вершин. Для облегчения процедуры поиска в граф вводится дополнительная вершина, которая объявляется единственной начальной и из которой имеются переходы в начальные вершины исходного графа.

Однако, граф потока событий не отражает полную информацию, полученную на этапе построения модели. В частности, в переходах никак не учитываются параметры ENABLE_LIST, CREATE_LIST, DISABLE_LIST и DESTROY_LIST, которые могут непосредственно влиять на достижимость некоторого элемента. Рассмотрим простой пример: в приложении имеется три элемента **A**, **B** и **C**, содержащиеся в одном контейнере (например, три кнопки), которым соответствуют действия **x**, **y** и **z**. Так как элементы содержатся в одном контейнере, то в графе потока событий имеются все возможные переходы между вершинами, соответствующими **x**, **y** и **z**. При этом, однако, при выполнении действия **x** над элементом **A**, элемент **C** переводится в неактивное состояние и действие **z** не может быть выполнено. Выполнение действия **y** над элементом **B**, напротив, переводит элемент **C** в активное состояние. Это означает, что последовательности событий, в которых действие **z** находится непосредственно после действия **x**, не являются допустимыми относительно действия **z**, так как это действие не может быть выполнено в связи с тем, что элемент **C** находится в неактивном состоянии. Напротив, в том случае, если элемент **C** изначально неактивен (то есть неактивен при запуске приложения), допустимыми относительно действия **z** являются только пути, содержащие действие **y,** переводящее элемент **C** в активное состояние.

Для непосредственной реализации модуля был выбран алгоритм Флойда-Уоршелла [6], который позволяет рассчитать кратчайшие пути между всеми вершинами рассматриваемого графа. Далее, на основе параметров ENABLE_LIST, CREATE_LIST, DISABLE_LIST и DESTROY_LIST вершин графа производится уточнение найденных путей.

Рассмотрим следующие модификации путей. Пусть путь содержит вершины **X** и **Y**, которые соответствуют элементам **A** и **B**, и выполняются следующие свойства:

- элемент **B** содержится в DISABLE_LIST(**X**) или DESTROY_LIST(**X**);
- событие **Y** находится в последовательности после события **X;**
- для всех событий **Z**, которые находятся между **X** и **Y**, элемент **B** не содержится ни в ENABLE_LIST(**Z**), ни в CREATE_LIST(**Z**).

Данный путь является недопустимым для элемента **B** и необходимо произвести разбиение пути на составляющие, не содержащие событие **X**.

Пусть путь содержит вершину **Y**, которая соответствуют элементу **B**, причем справедливо:

- элемент **B** содержится в ENABLE_LIST(**X**) или в CREATE_LIST(**X**);
- элемент **B** либо ещё не создан при запуске приложения, либо находится в неактивном состоянии**.**

Данный путь является недопустимым для элемента **B** и необходимо произвести разбиение пути на три составляющие:

- от начальной вершины до вершины **X**;
- от вершины **X** до вершины **Y**;
- от вершины **Y** до целевой вершины.

Использование компонента UndefinedComponentPath, использующего описанный выше метод поиска путей в графе потока событий и уточнение найденных путей, позволяет последовательно исследовать необработанные элементы графического пользовательского интерфейса. Каждый запуск программы на выполнение, нацеленный на обработку одного из элементов, позволяет обновить модель интерфейса, граф потока событий и увеличить полноту описания графического интерфейса.

5. Результаты применения инструмента

Исследование эффективности предложенного подхода проводилось на наборе тестовых приложений и наборе реальных проектов. Рассматриваемые проекты выбирались из списка приложений, которые ранее анализировались инструментом GUITAR. Приведём краткое описание данных проектов:

- JabRef – менеджер научных публикаций;
- Rachota – планировщик рабочего времени;
- TippyTipper – средство на платформе Android для расчёта чаевых и распределения платы на несколько человек;
- Browser – приложение операционной системы Android для просмотра веб-страниц.

Использование итеративного анализа с обновлением модели структуры графического интерфейса приложения позволило увеличить полноту данной модели.

Результаты применения итеративного анализа приведены в следующей таблице (в левой части представлены параметры модели, создаваемой при помощи оригинальной версии инструмента, в правой части — параметры модели, создаваемой с помощью модифицированной версии GUITAR после обхода всех необработанных элементов):

Табл. 1. Результаты применения итеративного анализа
Table 1. Results of applying iterative analysis

Программы	GUITAR			Iterative Ripper		
	число окон	число элементов	число событий	число окон	число элементов	число событий
Rachota	11	455	166	13	512	201
JabRef	42	1301	706	48	2005	1201
TippyTipper	3	45	24	5	58	34
Browser	3	43	27	5	51	31

Применение модифицированной схемы инструмента позволило построить более точную и полную модель графического интерфейса, которая может быть использована для создания тестовых сценариев с целью проверки корректности работы программ.

6. Заключение

В данной статье рассмотрена задача автоматизации динамического анализа приложений, предоставляющих графический пользовательский интерфейс. Среди существующих инструментов, применяемых в промышленной разработке программного обеспечения и в академических исследованиях, было выбрано средство GUITAR, обеспечивающее максимальную степень автоматизации процесса построения тестовых сценариев для проверки работы программы. На основе понятий модели представления структуры графического интерфейса и графа потока событий, введенных авторами инструмента GUITAR, была предложена схема итеративного анализа программы, позволяющая последовательно расширять и дополнять модель пользовательского интерфейса программы и строить новые тестовые наборы для обеспечения наиболее полного покрытия кода программы.

Предложенная схема показала лучшие результаты, чем оригинальная схема инструмента GUITAR, на всех исследованных приложениях с открытым исходным кодом. Выигрыш по эффективности анализа был достигнут за счёт создания модели, более точно описывающей реальную структуру графического пользовательского интерфейса. Теоретически показано, что большая точность модели позволит уменьшить количество тестовых наборов, за счет исключения тестовых сценариев, невозможных при реальной работе с приложением.

Стоит отметить, что в подходе к проведению автоматического динамического анализа программ, предоставляющих графический пользовательский

интерфейс, предложенном авторами инструмента GUITAR и расширенном в рамках исследования, существует ряд практических и теоретических ограничений. В частности, даже для приложений с малым количеством элементов графического интерфейса возникает проблема большого количества возможных тестовых наборов и выбора наиболее приоритетных тестовых наборов, позволяющих минимизировать время анализа, сохраняя полноту покрытия кода программы. В настоящее время активно развиваются подходы, связанные с интеллектуальным обходом путей выполнения в программе, использующие концепции символьного исполнения, статического анализа исходного и исполняемого кода. Исследования, связанные с возможностями совмещения методов, базирующихся на модели графического интерфейса, и указанных выше методов, представляются крайне перспективными в свете высокой востребованности автоматических инструментов оценки качества программного обеспечения.

Список литературы

[1]. Страница проекта Ranorex. http://www.ranorex.com/ [HTML]. Обращение от 10.10.2016
[2]. Abbot framework for automated testing of Java GUI components and programs. https://abbot.sourceforge.net. Обращение от 10.10.2016
[3]. Страница проекта Maveryx. https://sourceforge.net/projects/maveryx/ [HTML]. Обращение от 10.10.2016
[4]. Страница проекта Squish. https://www.froglogic.com/squish/ [HTML]. Обращение от 10.10.2016
[5]. Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, Atif Memon. GUITAR: an innovative tool for automated testing of GUI-driven software. Automated software engineering, 2013, vol. 21(1), pp. 65-105.
[6]. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 2-е изд. М.:«Вильямс», 2006. 1296 с.

Applying iterative dynamic analysis to programs with graphical user interface

M.K. Ermakov <mermakov@ispras.ru>
A.Y. Gerasimov <agerasimov@ispras.ru>
D.O. Kutz <kutz@ispras.ru>
A.A. Novikov <a.novikov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. This paper is dedicated to practical research in the field of automated testing and analysis of software that features a graphical user interface. Current tendencies in user interface development favour semi-automatic approaches that employ human experts to create and prepare test suites. The ever-increasing complexity of software leads to decreased

effectiveness of these approaches, especially when one considers large amounts of computational resources available during development. We present a fully automatic approach to dynamic analysis of program graphical interfaces. Our approach is based on the open-source GUITAR tool, which we have identified among other industrial and academic tools as the one closest to full automation. While highly efficient, GUITAR nevertheless has certain drawbacks and limitations which might cause insufficient accuracy in modelling the graphical interface structure and its individual elements. In turn, these limitations lead to fragments of graphical interface not getting processed during analysis or cause incorrect (i.e. not reproducible in practice) test cases to be generated. Our contributions include a set of modifications: incremental graphical interface model generation, improved identification of graphical interface element attributes and side effects, and finally a test case generation algorithm that focuses on reaching unprocessed graphical interface elements to check their functionality and improve the model. We have tested our modifications on a set of open source projects originally checked by GUITAR developers and achieved positive results: increased precision of GUI structure model and theoretically can decrease number of inapplicable test cases. Finally, we discuss several potential improvements for future work, including, in particular, the use of dynamic symbolic execution methods.

Keywords: dynamic analysis; program analysis; GUI testing; test coverage.

DOI: 10.15514/ISPRAS-2017-29(1)-8

For citation: Ermakov M.K., Gerasimov A.Y., Kutz D. O., Novikov A.A. Applying iterative dynamic analysis of programs with graphical user interface. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 1, 2017, pp. 119-134 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-8

References

[1.] Ranorex. http://www.ranorex.com/ [HTML]. Accessed at 10.10.2016
[2.] Abbot framework for automated testing of Java GUI components and programs. https://abbot.sourceforge.net. Accessed at 10.10.2016
[3.] Maveryx. https://sourceforge.net/projects/maveryx/ [HTML]. Accessed at 10.10.2016
[4.] Squish. https://www.froglogic.com/squish/ [HTML]. Accessed at 10.10.2016
[5.] Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, Atif Memon. GUITAR: an innovative tool for automated testing of GUI-driven software. Automated software engineering, 2013, vol. 21(1), pp. 65-105.
[6.] Tomas H. Kormen, Charl'z I. Lejzerson, Ronal'd L. Rivest, Klifford Shtajn. Introduction to Algorithms, 2-e izd. M.:«Vil'jams», 2006. 1296 p. (in Russian)

Прикладное применение динамического анализа программ, исполняющихся в интерпретирующих средах[1]

С. П. Вартанов <mermakov@ispras.ru>
М. К. Ермаков <mermakov@ispras.ru>
А. Ю. Герасимов <agerasimov@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Сложность современного программного обеспечения постоянно растет, в связи с чем возникает потребность в автоматических инструментах выявления ошибок в разработанных программах. В рамках данной статьи мы представляем решение некоторых задач, возникающих в процессе разработки программного обеспечения. Профилирование работы программ с динамической оперативной памятью, символьное исполнение программ с графическим пользовательским интерфейсом, обнаружение ошибок в параллельных программах – небольшой, но крайне важный класс задач, решение которых востребовано индустрией разработки программного обеспечения. В связи с отсутствием в виртуальной машине Dalvik операционной системы Android стандартных средств подключения агентов, на базе которых возможно проведение динамической инструментации байт-кода, в статье рассмотрен подход к профилированию использования динамической памяти Java-программами при помощи инструмента, реализованного как модифицированная виртуальная машина Dalvik операционной системы Android. Показана обоснованность примененного подхода, приведены практические результаты анализа нескольких программ из комплекта поставки операционной системы Android. Также в статье описано решение задачи динамического символьного исполнения программ с графическим пользовательским интерфейсом с целью генерации минимальных последовательностей управляющих воздействий на пользовательский интерфейс, обеспечивающих тестовое покрытие программы на базе статической инструментации байт-кода Java-программ и модификации инструмента генерации тестовых наборов для приложений с графическим пользовательским интерфейсом GUITAR. В завершении статьи рассматривается применение особенности реализации инструмента обнаружения ошибок синхронизации параллельных программ на языке Java, исполняемых виртуальной машиной Dalvik операционной системы Android на базе статической инструментации байт-кода Java-программ и применения инструмента ThreadSanitizer.

[1] Работа проводится в рамках научно-исследовательских работ Института системного программирования РАН в 2014 – 2017 годах

Ключевые слова: динамический анализ программ; анализ программ.

DOI: 10.15514/ISPRAS-2017-29(1)-9

Для цитирования: Вартанов С.П., Ермаков М.К., Герасимов А.Ю. Прикладное применение динамического анализа программ, исполняющихся в интерпретирующих средах. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 135-148. DOI: 10.15514/ISPRAS-2017-29(1)-9

1. Введение

В настоящее время в индустрии программного обеспечения наблюдается развитие сектора разработки программного обеспечения на языках программирования с интерпретирующей средой, например, Java Virtual Machine (JVM) или Common Language Runtime (CLR) как для серверного программного обеспечения, так и для настольных и мобильных приложений (например, на базе операционных систем Android или Windows Phone). В связи с этим возникает задача анализа программ, выполняющихся в интерпретирующей среде, с учетом специфики как самой среды, так и окружающего системного программного обеспечения.

Анализ программ может производиться для проверки различных свойств программного обеспечения, например, на наличие дефектов и уязвимостей, производительности и ресурсоемкости целевого программного обеспечения, а также с целью восстановления алгоритмов и моделей работы целевой программы. Таким образом наличие методик и средств анализа программ является одной из важнейших компонент обеспечения качества продуктов на протяжении всего жизненного цикла программного обеспечения.

Исследования в рамках данного проекта направлены на разработку подходов и средств автоматического и полуавтоматического анализа приложений, выполняемых в рамках интерпретирующих сред. Подобный анализ может проводиться для поиска дефектов и уязвимостей, обнаружения частей программ, осуществляющих неэффективное использование временных и системных ресурсов и др. Актуальность подобных средств анализа в настоящее время определяется распространённостью приложений, исполняющихся в интерпретирующих средах (например для языка Java, являющегося целевым для данного проекта, или для языка C# со средой выполнения CLR), в частности на мобильных платформах, таких как Android и Windows Phone.

В статье рассмотрены особенности методов динамического анализа, нацеленных на профилирование программ, а также методов для анализа программ, обладающих графическим пользовательским интерфейсом и использующих параллельные вычисления. Также в работе помимо анализа бинарного кода для архитектуры x86 и x86-64, рассматриваются особенности анализа программ, работающих на архитектуре ARM, и программ, написанных на языке Java.

2. Профилирование программ

Динамический анализ эффективно применяется для проверки свойств времени выполнения программы, таких как количество потребляемой программой памяти, объём потребляемых вычислительных ресурсов, статистика использования кэша и т. п. В рамках данного проекта было проведено исследование методов сбора информации об эффективности использования памяти программами операционной системы Android [1]. Основная часть программ для этой платформы написана на языке Java и выполняется виртуальной машиной Dalvik [2]. Особенности профилирования программ тесно связаны с особенностями как самой операционной системой, так и с особенностями используемой виртуальной машины.

Виртуальная машина Dalvik реализует лишь часть стандартных интерфейсов виртуальных машин Java. Так, в ней присутствуют механизм вызова функций в бинарном коде JNI, протокол передачи данных JDWP и механизм создания слепков кучи, но отсутствуют средства поддержки динамически подключаемых агентов. Последнее ограничение означает невозможность использования стандартных средств динамической инструментации Java программ и средств динамического анализа программ, реализация которых основана на применении динамической инструментации. Протокол JDWP описывает способы обмена информацией между отладчиком и виртуальной машиной Dalvik. С его помощью можно запрашивать информацию о текущем состоянии памяти виртуальной машины, управлять потоками, механизмом сборки мусора и модифицировать параметры объектов и классов, а также описывать события, при возникновении которых данные о состоянии виртуальной машины автоматически отправляются отладчику.

Виртуальная машина Dalvik предоставляет интерфейс для извлечения в заданные моменты времени слепков кучи, которые содержат в себе информацию о состоянии объектов, хранящихся в памяти. Существует множество инструментов, нацеленных на анализ слепков, сделанных на разных стадиях процесса выполнения анализируемой программы, однако точность таких инструментов напрямую зависит от частоты извлечения слепков, создание которых требует полной остановки работы виртуальной машины. К тому же при помощи анализа разницы между слепками не представляется возможным сделать выводы о том, какие из выделенных и освобождённых объектов были использованы в ходе выполнения программы. Это не позволяет в полной мере делать выводы об эффективности использования памяти анализируемой программой.

В связи с указанными особенностями в настоящем проекте с целью извлечения всей необходимой информации для полноценного анализа использования памяти приложениями платформы Android предлагается производить постоянный сбор всех событий, связанных с выделением, освобождением и использованием памяти при помощи внесения изменений в виртуальную машину Dalvik, упаковку собранной информации в специальные

пакеты и передачу их с использованием протокола JDWP специальному программному средству, анализирующему указанную информацию.

2.1 Сбор информации

Осуществляется сбор следующих типов событий: загрузка и инициализация класса, создание экземпляра класса, использование экземпляра при помощи интерфейсов доступа к нему, а также освобождение памяти при помощи сборщика мусора.

Собранная информация упаковывается в пакеты определённых типов:

- "request allocations" для информации о созданных объектах;
- "request freed" для информации об их освобождении;
- "object usage" для информации об использовании созданных объектов;
- "class information" для информации о внутренней структуре загружаемых классов.

Создание новых объектов в виртуальной машине Dalvik происходит в рамках единой процедуры. Это позволяет использовать её для точечного сбора данных о выделенных объектах.

Освобождение созданных объектов происходит в рамках процедуры сбора мусора и осуществляется в соответствии с алгоритмом MarkSweep [3]. Сбор информации об освобождённой памяти происходит на финальном этапе работы алгоритма, в рамках которого известен полный список объектов, память которых необходимо освободить.

Анализ использования памяти может быть произведён с разной степенью точности в зависимости от задач, стоящих перед анализатором. В первом случае, если достаточно информации о том, какие из объектов были использованы перед тем, как занимаемая ими память была освобождена, а какие – нет, и при этом не имеет значения, какая именно часть объектов была использована, достаточно использовать единственный дополнительный флаг на каждый объект, а передачу информации об использовании можно объединить с передачей отладчику информации об освобождении памяти, разделив объекты на два множества: те, к которым был осуществлён доступ по крайней мере один раз, и те, которые не были использованы вплоть до их уничтожения сборщиком мусора.

Если же для анализа необходима информация о том, какие именно части объекта были использованы (например, для определения эффективности выбора внутренней структуры классов), в ходе работы программы происходит отслеживание всех методов, осуществляющих доступ к внутренним структурам объектов через внешние интерфейсы. В этом случае для передачи информации отладчику используется специальный тип пакетов, в которых сохраняется идентификатор объекта и смещение, по которому был осуществлён доступ. Информация о внутренней структуре загруженных

классов используется для восстановления полей объектов по сохранённым смещениям.

2.2 Инструментальное средство

На основе описанных подходов был реализован прототип инструментального средства, который предназначен для работы на внешней рабочей станции, имеющей подключение к устройству с платформой Android, и который обеспечивает сохранение, обработку и анализ информации, поступающей от модифицированной виртуальной машины Dalvik. Инструментальное средство производит установку соединения с виртуальной машиной и отправляет команду запуска сбора информации. В ходе работы анализируемого приложения инструмент прослушивает канал связи и осуществляет приём пакетов описанных типов с информацией об использовании памяти.

Инструментальное средство также обладает графическим пользовательским интерфейсом, отображающим процесс использования памяти в анализируемой программе в режиме реального времени, и системой команд для управления отображением.

2.3 Практические результаты

С целью проверки практической применимости разработанного прототипа инструмента и предложенных методов, прототип инструмента был использован для анализа работы с памятью в ряде стандартных программ операционной системы Android: Mms (программа для обмена сообщениями), Calendar (органайзер), Browser (средство для просмотра содержимого страниц сети Интернет), Deskclock (стандартные часы) и Contacts (средство для работы со списком контактов). В качестве целевой платформы анализа использовалась версия операционной системы Android 4.2.2, работающая на устройстве PandaBoard. Сбор информации об использовании памяти производился для различных сценариев использования приложений. По результатам исследований был выявлен ряд закономерностей, свойственных всем проанализированным программам.

Для отдельных классов объектов программ Java были выявлены шаблоны частот использования полей этих классов, на основе которых можно сделать вывод об эффективности применения механизма наследования.

Было выяснено, что основная доля выделенной и используемой памяти приходится на массивы примитивных типов. Так от 50 до 80 % всей выделенной памяти составляли объекты типов `byte[]`, `char[]` и `int[]`. Это объясняется использованием их для хранения изображений, которые формируются для графического пользовательского интерфейса, работой со строками и использованием системы обработки исключительных ситуаций соответственно. По результатам проведенного исследования программ для операционной системы Android можно сделать вывод, что как в конечных

классах, так и в классах стандартной библиотеки Android, от которых они были унаследованы существует ряд полей, которые не используются ни в одном из созданных объектов. Если применить методы рефакторинга стандартной библиотеки классов операционной системы Android и классов приложений, то можно значительно сократить использование оперативной динамической памяти программами операционной системы Android.

3. Анализ программ с графическим интерфейсом

Использование графического пользовательского интерфейса с одной стороны облегчает взаимодействие пользователя с программным средством и расширяет возможности его использования, с другой стороны значительно усложняет логику программного обеспечения. Механизмы графического пользовательского интерфейса подразумевают ожидание действий пользователя, как правило представляемых в программе в виде событий, и реакцию на них, как правило в виде вызова функций-обработчиков событий графического пользовательского интерфейса, что при значительном количестве активных графических элементов порождает множество различных сценариев работы программы. В связи с большим количеством возможных комбинаций событий и, следовательно, вызовов обработчиков событий, в программах с графическим пользовательским интерфейсом, ручное тестирование подобных программ становится весьма трудоёмким процессом, что приводит к сложности как обнаружения ошибок в программе, так и воспроизведения и отладки. Поэтому, возникает необходимость создания инструментальных средств автоматической генерации тестовых наборов для программ с графическим пользовательским интерфейсом, не требующих постоянного участия человека в процессе анализа поведения программы.

Для анализа программ с графическим пользовательским интерфейсом классические методы анализа, основанные на символьном выполнении, не подходят без дополнительных модификаций, поскольку потоки данных часто проходят через функции библиотек, отвечающих за отображение графического интерфейса, чем сильно усложняют формулу в булевских ограничениях операциями, не влияющими на логику исполнения программы. В связи с этим, для проведения анализа методами символьного исполнения требуется предварительное построение модели графического пользовательского интерфейса и структуры анализируемой программы.

Большинство современных программ обладают сложной структурой и большим количеством элементов графического интерфейса, и их структура часто не полностью определяется статически, а дополняется или модифицируется в процессе выполнения программы. Существующие средства анализа осуществляют построение модели графического интерфейса на основе запуска программы. Одним из наиболее эффективных инструментов для автоматического анализа программ с графическим интерфейсом на платформе Android является инструмент GUITAR [4], который позволяет проводить без

участия эксперта проводить построение моделей программ, графический интерфейс которых основан на использовании AWT/Swing, SWT, и программ платформы Android. Основным недостатком этого инструмента является то, что построенная модель графического пользовательского интерфейса основывается на информации, собранной в процессе одного запуска приложения и конкретном сценарии его использования. При различных сценариях работы с программой эта модель может быть разной, а значит требуется извлекать информацию о структуре графического пользовательского интерфейса программы с учётом разных запусков.

В рамках проекта рассмотрен гибридный метод, совмещающий автоматическое построение модели графического интерфейса на основе нескольких запусков программы с анализом, основанном на символьном исполнении.

Существует множество инструментов, осуществляющих динамический анализ программ на языке Java методами символьного исполнения. К таким можно отнести инструмент Java PathFinder [5], который представляет собой символьную виртуальную машину Java. Инструмент JDart [6] позволяет строить новые пути выполнения программы при помощи символьного исполнения и основан на инструменте Java PathFinder. Инструмент Coffee Machine [7], использующий динамическое символьное исполнение, основан на методах статической инструментации байт-кода языка Java и позволяет производить анализ вне зависимости от виртуальной машины.

В связи с тем, что на платформе Android для запуска Java-программ используется собственная виртуальная машина Dalvik, в рамках проекта использовался механизм статической инструментации инструмента Coffee Machine, а также возможности инструмента GUITAR для построения модели графического пользовательского интерфейса приложения.

GUITAR строит модель графического пользовательского интерфейса приложения, основываясь на иерархической структуре элементов графического пользовательского интерфейса и при помощи построения ориентированного графа потока событий. Ребро графа потока событий между событиями A и B обозначает возможность возникновения события B после события A. Связи между элементами в этом графе означают возможные события между графическими элементами. Таким образом, пути в построенном графе соответствуют некоторым сценариям работы с анализируемым приложением. Одним из компонентов инструмента осуществляется построение таких сценариев, а другим — их воспроизведение.

Механизм символьного исполнения применительно к программам с графическим пользовательским интерфейсом предлагается использовать, основываясь на следующих принципах:

- традиционная трасса символьного выполнения строится исключительно для событий, связанных с графическим интерфейсом;

- помеченными (а, соответственно, и символьными) считаются не только данные, косвенно зависящие от входных, но и поля примитивных типов и массивы примитивного типа и поля объектов, к которым существует доступ из статических полей классов;
- в трассу добавляются специальные маркеры начала и конца события.

Общий механизм динамического символьного выполнения применяется не только для инвертирования условий, но и для перестановки частей трассы в построенных ограничениях, что позволяет исследовать различные пути выполнения программы и различные сценарии с точки зрения пользовательского интерфейса.

Практическая применимость предложенных методов анализа программ с графическим пользовательским интерфейсом была проверена на программах Calculator и ProgCalc операционной системы Android. В результате было обнаружено 17 и 2 дефекта соответственно. Для первого приложения причиной аварийного завершения явилась не обработанная должным образом исключительная ситуация типа `NumberFormatException`, для второго — `NumberFormatException` и `ArithmeticException`.

4. Анализ многопоточных программ

В настоящее время активно развивается отрасль создания вычислительных устройств, обладающих двумя и более вычислительными ядрами, а также создаются механизмы синхронизации между вычислительными процессами средствами операционных систем. В связи с этим, для более эффективного использования вычислительных ресурсов активно разрабатываются программы, использующие параллельные вычисления. Особенно это справедливо для программ, использующих графический пользовательский интерфейс, поскольку для снижения задержек при работе с графическим пользовательским интерфейсом программы используется несколько вычислительных потоков, в одном из которых реализуется цикл обработки сообщений от элементов графического пользовательского интерфейса.

Высокая сложность логики программы, использующей параллельные вычисления, приводит к возникновению специфических ошибок, связанных с синхронизацией вычислительных потоков. Основная сложность таких ошибок заключается в том, что их проявление зависит не только от входных данных программы, но также и от алгоритмов работы планировщика потоков операционной системы, т. е. недетерминированно с точки зрения пользователя, что сильно затрудняет их поиск, отладку и воспроизведение.

К наиболее распространённым ошибкам синхронизации относятся взаимная блокировка потоков и состояние гонки.

В рамках данного исследования рассматривается анализ пользовательских приложений для платформы Android. Большинство приложений этой операционной системы используют графический пользовательский интерфейс

и параллельные вычисления. Также приложения для Android разрабатываются на языке Java и других, которые могут быть транслированы в Java байт-код, и выполняются на виртуальной машине Dalvik.

4.1 Методы поиска ошибок синхронизации

Основными механизмами, которые используются для поиска ошибок синхронизации, являются построение и отслеживание множеств блокировок и построение модели на основе отношения предшествования между событиями.

Первый механизм предполагает сопоставление каждому отдельному событию в программе состояния, которое включает в себя информацию о правилах предоставления отдельным нитям эксклюзивного доступа к разделяемым данным, которые действуют в выполнении данного события. Описанные состояния обновляются при входе и выходе из критической секции. Недостатком этого метода является то, что он постулирует слишком строгое условие отсутствия ошибки синхронизации.

Второй механизм предполагает установление на всём множестве событий конкретного пути выполнения программы отношения предшествования по следующему правилу: одно событие называется предшествующим другому, если оно предшествует ему (возможно транзитивно) в рамках одного вычислительного потока, либо одно событие является отправкой, а другое — принятием одного и того же сообщения.

В рамках проекта предполагается использование гибридного метода, который сочетает в себе идеи обоих методов.

4.2 Существующие решения

На сегодняшний день существует множество средств, предоставляющих возможность поиска ошибок синхронизации в программах, использующих параллельные вычисления. Это инструменты Intel Thread Checker [8], Sun Thread Analyzer [9], а также инструменты, основанные на среде динамической инструментации Valgrind [10]: helgrind [11], DRD [12] и ThreadSanitizer [13].

Первые два приведённых инструмента реализуют метод, основанный на отношениях предшествования, и гибридный метод соответственно. Они, однако, не доступны для архитектур семейства ARM.

Инструменты, основанные на фреймворке Valgrind, поддерживают архитектуру ARM. Наиболее эффективным из них является инструмент ThreadSanitizer.

4.3 ThreadSanitizer

ThreadSanitizer представляет собой не отдельный инструмент, а целый набор инструментов, предназначенных для поиска ошибок взаимной блокировки и состояния гонки в программах и основанных на общих подходах для решения этой проблемы. Одним из его компонентов является ThreadSanitizer Offline,

инструмент, способный осуществлять поиск ошибок в построенной модели выполнения программы, другим — Java ThreadSanitizer, осуществляющий построение такой модели для программ, которые были транслированы в Java байт-код.

Основой методов, реализованных в интрументах набора ThreadSanitizer, является рассмотрение участков программы, в которых отсутствуют синхронизационные событие, т. е. участков, результат выполнения которых не зависит от последовательности выполнения нитей. Под синхронизационными событиями здесь понимаются операции доступа к общей памяти, выставление или снятие блокировок, операции передачи и принятия сигналов, а также операции над потоками.

В основе алгоритма лежит построение модели выполнения программы с использованием отношения предшествования на множестве описанных участков программы. На основе построенной модели ThreadSanitizer Offline принимает решение о том, какие события, осуществляющие доступ к разделяемым данным (одно из которых их модифицирует), не связаны отношением предшествования. Это означает, что выполнение программы зависит от порядка выполнения нитей, что в большинстве случаев не предусмотрено программистом и означает наличие в программе дефекта. Формат модели достаточно универсален и подходит для описания выполнения программ, использующих параллельные вычисления, написанных на различных языках.

Инструмент Java ThreadSanitizer строит модель в описанном формате для программы, представленной в виде байт-кода языка Java, с использованием динамической инструментации, которая предоставляется библиотекой ASM [14]. Эта библиотека использует стандартный механизм виртуальной машины Java для проведения динамической инструментации, который отсутствует в виртуальной машине Dalvik операционной системы Android, что не позволяет использовать инструмент Java ThreadSanitizer на данной платформе.

4.4 Coffee Machine

Инструмент Coffee Machine, разработанный в ИСП РАН, производит статическую инструментацию байт-кода языка Java на основе библиотеки BCEL [15]. Это означает, что преобразование программы происходит до её выполнения, а не во время. Такой подход позволяет нужным образом модифицировать программу вне зависимости от того, каким образом она будет выполняться и какая виртуальная машина для этого будет применена.

В рамках проекта на базе инструмента Coffee Machine был реализован гибридный метод, совмещающий в себе отслеживание множеств блокировок и вычисление отношений предшествования между синхронизационными событиями, что в сочетании с использованием инструмента ThreadSanitizer Offline для проверки построенной модели позволило использовать его для поиска синхронизационных ошибок в программах платформы Android.

4.5 Результаты

Практические подтверждение применимости предложенного подхода проводились на ряде приложений операционной системы Android. Для автоматического построения сценариев работы использовалась модифицированная версия инструмента GUITAR. Результатом стало обнаружение некоторых реальных дефектов, связанных с ошибками в реализации синхронизации потоков. В качестве примера можно привести дефект, обнаруженный в приложении Email, которое входит в набор предустановленных программ операционной системы. Обнаруженная ситуация гонки при определённом порядке выполнения потоков приводит к возникновению исключению `NullPointerException` и аварийному завершению программы.

5. Заключение

В данной статье рассмотрены прикладные задачи, решение которых возможно при помощи динамического анализа программ. Описанные методы реализованы в виде экспериментальных прототипов инструментов для анализа приложений на языке Java и проверены на программах из поставки операционной системы для мобильных устройств Android.

Список литературы

[1]. Официальный анонс выпуска Android 3.0 [HTML] (http://developer.android.com/about/versions/android-3.0-highlights.html). Обращение от 01.12.2016

[2]. D. Bornstein. Dalvik VM internals (http://sites.google.com/site/io/dalvik-vm-internals) Обращение от 01.12.2016

[3]. David Detlefs, Christine Flood, Sete Heller, Tony Printezis. Garbage-first garbage collection. *ISMM'4 Proceedings of the 4th international symposium on Memoty management*, pp. 37-48. Vancouver, BC, Canada. October 24-25, 2004

[4]. Bao N. Nauyen, Bryan Robbins, Ishan Banerjee, Atif Memon. GUITAR: an innovative tool for automated testinf of GUI-driven software. *Automated Software Engineering, Volutme 21, Issue 1, March 2014*. pp. 65-105.

[5]. Willem Visser, Corina S. Pāsāreanu, Sarfranz Khurshid. Test input generation with java PathFinder. *ISSTA '04 Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*. pp. 97-107. Boston, Massachusetts, USA. July 11-14, 2004.

[6]. Kasper Luckow, Marko Dimjašević, Dimitra Giannakopoulou, Falk Howar, Malte Isberner, Temesghen Kahsai, Zvonimir Rakamarić, Vashwanath Raman. JDart: a dynamic symbolic analysis framework. *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Volume 9636*. pp. 442-459. Springer-Verlang New Yor, Inc, New York, NY, USA. April 02-08, 2016.

[7]. С.П. Вартанов, М.К. Ермаков. Применение статической инструментации байт-кода языка Java для динамического анализа программ. Труды ИСП РАН, том 27, выпуск 1, 2015 г., стр. 25-38. DOI: 10.15514/ISPRAS-2015-27(1)-2.
[8]. U. Banerjee, B. Bliss, Zh. Ma, P. Petersen. Unraveling Data Race Detection in the Intel® Thread Checker. In Proceedings of STMCS '06. Manhattan, NY, USA, 2006
[9]. Руководство по инструменту Sun Thread Analyzer [HTML] (http://docs.oracle.com/cd/E19205-01/820-4155/tha.html) Обращение от 01.12.2016
[10]. N. Nethercote and J. Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007), San Diego, California, USA, 2007
[11]. Ali Jannesari, Walter F. Tichy, Victor Pankratius, Kaibin Bao. Helgrind+: an efficient dynamic race detector. *Parallel and Distributed Processing Symposium, International(2009)*. pp. Rome, Italy, May 23-29, 2009
[12]. K. Serebryany and T. Iskhodzhanov. ThreadSanitizer – data race detection in practice. WBIA '09, New York City, NY, USA, 2009
[13]. Ali Jannesari, Markus Westpahl-Futuya, Walter F Tichy. Dynamic data race detection for correlated variables. *ICA3PP'11 Proceedings of the 11th international conference on Algorithms and architectures for parallel processing, Colume Part I*. pp. 14-26. Melbourne, Austratia. October 24-26, 2011
[14]. E. Bruneton, R. Lenglet, T. Coupaye. ASM: a code manipulation tool to implement adaptable systems. *Adaptable and extensible systems*, November 2002. Grenoble, France.
[15]. Apache Commons Byte Code Engineering Library [HTML] (http://commons.apache.org/bcel) Обращение от 01.12.2016

Applying dynamic analysis to programs running in interpreted environments

S.P. Vartanov <svartanov@ispras.ru>
M.K. Ermakov <mermakov@ispras.ru>
A.Y. Gerasimov <agerasimov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. The present-day trends in software engineering include the steady increase of code and design complexity which reinforces the high demand in automated software testing and analysis tools. In this paper, we showcase several dynamic program analysis applications and present our solutions. These applications include memory profiling, automated test generation using dynamic symbolic execution and automated detection of concurrency bugs in multithreaded programs. Our memory profiling tool is designed for Java applications for Android and it is implemented through Android Dalvik VM modification. This approach allowed us to overcome existing Dalvik VM limitations that make existing profiling tools based on dynamic bytecode instrumentation inaccessible. We have successfully applied our tool to several core Android applications - the results provided in the paper outline the effectiveness of the approach. The second solution we discuss in the paper - dynamic

symbolic execution for test generation automation - allows us to efficiently generate test scenarios for Java program graphical user interface. The core technologies of the approach include the use of static bytecode instrumentation and automatic GUI model extraction. We implement the approach on top of a user interface test automation framework GUITAR. Finally, we present our approach to automatically identify concurrency bugs in multithreaded Java applications. The approach is based on static bytecode instrumentation for trace generation and employs ThreadSanitizer defect detection tool for identifying bugs.

Keywords: dynamic analysis; program analysis.

DOI: 10.15514/ISPRAS-2017-29(1)-9

For citation: Vartanov S.P., Ermakov M.K., Gerasimov A.Y. Applications of dynamic analysis of programs executed inside of interpreting environments. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 1, 2017. pp. 135-148 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-9

References

[1]. The official announce of the Android 3.0 release [HTML] (http://developer.android.com/about/versions/android-3.0-highlights.html). Accessed at 01.12.2016

[2]. D. Bornstein. Dalvik VM internals (http://sites.google.com/site/io/dalvik-vm-internals) Accessed at 01.12.2016

[3]. David Detlefs, Christine Flood, Sete Heller, Tony Printezis. Garbage-first garbage collection. *ISMM'4 Proceedings of the 4th international symposium on Memoty management.* pp. 37-48. Vancouver, BC, Canada. October 24-25, 2004

[4]. Bao N. Nauyen, Bryan Robbins, Ishan Banerjee, Atif Memon. GUITAR: an innovative tool for automated testinf of GUI-driven software. *Automated Software Engineering, Volutme 21, Issue 1, March 2014*. pp. 65-105.

[5]. Willem Visser, Corina S. Pāsāreanu, Sarfranz Khurshid. Test input generation with java PathFinder. *ISSTA '04 Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*. pp. 97-107. Boston, Massachusetts, USA. July 11-14, 2004.

[6]. Kasper Luckow, Marko Dimjašević, Dimitra Giannakopoulou, Falk Howar, Malte Isberner, Temesghen Kahsai, Zvonimir Rakamarić, Vashwanath Raman. JDart: a dynamic symbolic analysis framework. *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Volume 9636*. pp. 442-459. Springer-Verlang New Yor, Inc, New York, NY, USA. April 02-08, 2016.

[7]. S. P. Vartanov, M. K. Ermakov. Applying Java bytecode static instrumentation for software dynamic analysis. Trudy ISP RAN / Proc. ISP RAS, volume 27, issue 1, 2015, pp. 25-38 (in Russian). DOI: 10.15514/ISPRAS-2015-27(1)-2.

[8]. U. Banerjee, B. Bliss, Zh. Ma, P. Petersen. Unraveling Data Race Detection in the Intel® Thread Checker. In Proceedings of STMCS '06. Manhattan, NY, USA, 2006

[9]. The manual for Sun Thread Analyzer [HTML] (http://docs.oracle.com/cd/E19205-01/820-4155/tha.html) Accessed at 01.12.2016

[10]. N. Nethercote and J. Seward. Valgrind: A Framework for Heavyweight Dynamic Binary Instrumentation. Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007), San Diego, California, USA, 2007

[11]. Ali Jannesari, Walter F. Tichy, Victor Pankratius, Kaibin Bao. Helgrind+: an efficient dynamic race detector. *Parallel and Distributed Processing Symposium, International (2009)*. pp. 1-13. Rome, Italy, May 23-29, 2009

[12]. K. Serebryany and T. Iskhodzhanov. ThreadSanitizer – data race detection in practice. WBIA '09, New York City, NY, USA, 2009

[13]. Ali Jannesari, Markus Westpahl-Futuya, Walter F Tichy. Dynamic data race detection for correlated variables. *ICA3PP'11 Proceedings of the 11th international conference on Algorithms and architectures for parallel processing, Colume Part I*. pp. 14-26. Melbourne, Austratia. October 24-26, 2011

[14]. E. Bruneton, R. Lenglet, T. Coupaye. ASM: a code manipulation tool to implement adaptable systems. *Adaptable and extensible systems*, November 2002. Grenoble, France.

[15]. Apache Commons Byte Code Engineering Library [HTML] (http://commons.apache.org/bcel) Accessed at 01.12.2016

Динамический анализ приложений с графическим пользовательским интерфейсом на основе символьного исполнения[1]

С.П. Вартанов <svartanov@ispras.ru>
А.Ю. Герасимов <agerasimov@ispras.ru>
М.К. Ермаков <mermakov@ispras.ru>
Д.О. Куц <kutz@ispras.ru>
А.А. Новиков <a.novikov@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. Данная статья посвящена исследованию возможностей применения современных методов динамического анализа программ на основе символьного исполнения к программному обеспечению, предоставляющему графический пользовательский интерфейс. Отличительными особенностями подобных программ является интерактивная обработка данных и применение параллельных потоков команд. Данные особенности значительно усложняют эффективность применения подходов автоматического обхода путей выполнения на основе символьного исполнения. В рамках статьи предлагается проводить анализ программ, предоставляющих графический интерфейс, с помощью гибридного метода, включающего символьное исполнение и стандартные подходы к извлечению модели графического интерфейса для построения тестовых сценариев. В статье представлен обзор существующих программных средств, предоставляющих возможности анализа и тестирования программ, и выделены средства GUITAR и Coffee Machine, совмещение которых позволяет эффективно анализировать Java-программы с графическим пользовательским интерфейсом. Рассматривается схема внедрения модулей инструментации байт-кода системы Coffee Machine в рабочий цикл инструмента GUITAR. Модель структуры графического интерфейса, извлекаемая инструментом GUITAR, расширяется фрагментами предикатов пути, построенных с помощью символьного исполнения. Представлен алгоритм составления предикатов в сложные трассы, обрабатываемые инструментами проверки выполнимости булевых ограничений, позволяющий автоматически генерировать тестовые сценарии для обхода различных путей выполнения по коду функций обработки событий

[1] Работа проводится при финансовой поддержке Российского фонда фундаментальных исследований, номер проекта 14-07-00609

взаимодействия с элементами графического интерфейса. Представлены практические результаты применения совмещенного метода, позволившего обнаружить необработанные исключения в ряде проектов с открытым исходным кодом, и дана оценка полученных результатов. В заключении статьи даётся оценка эффективности предложенного метода и рассмотрены основные ограничения, избавление от которых представляется актуальным направлением дальнейших исследований.

Ключевые слова: динамический анализ программ; анализ программ; тестирование GUI, тестовое покрытие.

DOI: 10.15514/ISPRAS-2017-29(1)-10

Для цитирования: Вартанов С.П., Герасимов А.Ю., Ермаков М.К., Куц Д.О., Новиков А.А. Динамический анализ приложений с графическим пользовательским интерфейсом. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 149-166. DOI: 10.15514/ISPRAS-2017-29(1)-10

1. Введение

В настоящее время современные высокотехнологичные устройства становятся неотъемлемой частью повседневной жизни человека. Растущая популярность компьютерных технологий неразрывно связана с применением графического пользовательского интерфейса операционной системы и прикладных приложений, который позволяет предоставлять информацию и управлять устройствами в интуитивно понятном виде. Например, количество пользователей услуги мобильного банкинга для частных лиц в России за 2015 год увеличилось на 58 % [1]. Создание приложений с графическим пользовательским интерфейсом сопряжено с достаточно трудоёмким процессом тестирования разработанного приложения. Сложность процесса тестирования связана с количеством элементов графического пользовательского интерфейса приложения: чем больше элементов управления графическим пользовательским интерфейсом используется приложением, тем больше количество возможных сценариев работы, которые необходимо протестировать.

Современные средства разработки программ предоставляют возможности автоматизации создания приложений с графическим пользовательским интерфейсом. Данные инструментальные средства ориентированы на создание модели, описывающей структуру графического пользовательского интерфейса. На основе модели при участии эксперта создаются тестовые сценарии, описывающие последовательность действий над элементами графического пользовательского интерфейса. Воспроизведение тестовых наборов и отслеживание работы программы осуществляется автоматически. По результатам исполнения программы с использованием тестовых наборов контролируется качество программного обеспечения. В то время как такой подход позволяет приблизить набор тестовых сценариев к ожидаемым от пользователя действиям, он не позволяет подробно исследовать алгоритмы,

лежащие в основе функций-обработчиков событий от элементов пользовательского интерфейса. Если данные алгоритмы достаточно сложны, то вероятность обнаружения последовательности действий пользователя, приводящей к нарушению работы программы, является достаточно низкой. Подобная проблема актуальна, например, для программ, предоставляющих графический интерфейс пользователя для составления сложных выражений и запуска обработки данных выражений. Простой обход последовательности взаимодействия с элементами графического пользовательского интерфейса не позволит обнаружить экстремальные ситуации и краевые случаи для встроенного разборщика выражений и модуля подсчета значений. Можно отметить, что графический интерфейс в данном случае маскирует наиболее сложные фрагменты кода программы, а инструменты, составляющие тестовые сценарии по извлеченной модели структуры графического интерфейса, будут проводить недостаточно эффективный анализ.

В настоящее время развиваются методы итеративного динамического анализа программ, проводящие обход различных путей выполнения с целью выявления краевых значений входных данных для алгоритмов программы и обнаружения потенциальных дефектов и уязвимостей. Данные методы используют принципы символьного исполнения и анализа потока помеченных данных программы для сбора предикатов пути (трасс ограничений) и решатели булевых ограничений (солверы) для автоматического построения наборов входных данных.

К сожалению, их прямое применение к приложениям, предоставляющим графический пользовательский интерфейс (и в общем смысле – к интерактивным приложениям) часто не бывает достаточно эффективным. К основным факторам, обуславливающим данный эффект, можно отнести следующие:

- активные потоки данных проходят через библиотеки, осуществляющие отображение элементов графического интерфейса и обработку взаимодействия с данными элементами со стороны пользователя, что приводит к необоснованному увеличению объема трасс выполнения;
- многопоточное выполнение, в рамках которого происходит ожидание и отклик на действия пользователя, затрудняет выделение актуальных фрагментов трассы.

В связи с этим становится актуальной задача создания и развития методов анализа программ, объединяющих подходы автоматического тестирования приложений с графическим пользовательским интерфейсом и динамического символьного исполнения. Построение модели графического интерфейса будет предоставлять возможность создания корректных сценариев взаимодействия с программой, а применение символьного исполнения позволит исследовать логику работы программы в рамках данных сценариев.

В данной статье описан подход, совмещающий автоматическое построение модели графического интерфейса с анализом, основанным на символьном исполнении кода программ. В рамках подхода проводится исследование программ на языке Java.

Статья имеет следующую структуру. В разд. 2 приводится краткое описание методик анализа, совмещение которых позволит получить преимущества для автоматического анализа программ с графическим пользовательским интерфейсом, и существующих инструментов, реализующих методики в рамках данных направлений. В разд. 3 описаны основные положения предлагаемого совмещенного метода анализа. В разд. 4 рассматриваются результаты применения совмещенного метода анализа к ряду проектов с открытым исходным кодом и дается оценка полученных результатов. В заключении представлена оценка проделанной работы и рассмотрены перспективные направления дальнейшего развития темы.

2. Обзор существующих решений

В данной главе производится обзор существующих методов и инструментов, используемых для тестирования программ с графическим пользовательским интерфейсом и применяемых для динамического символьного исполнения программ.

2.1 Автоматизация тестирования программ с графическим пользовательским интерфейсом.

Большинство средств тестирования программ с графическим интерфейсом в своей работе используют подходы, не зависящие от кода исследуемых программ. Стандартная схема данных подходов включает в себя работу с моделью графического пользовательского интерфейса программы, описывающей структурные элементы данного интерфейса и особенности взаимодействия между ними. С помощью языка описания тестовых сценариев на основе данной модели осуществляется создание тестового покрытия; для выполнения данных сценариев используются компоненты, программно эмулирующие действия пользователя по взаимодействию с графическим пользовательским интерфейсом программы.

Среди успешных и распространённых инструментов, следующих данной методике, можно выделить такие средства, как Ranorex [2], Abbot [3], Maveryx [4], Squish [5], Sikuli [6]. Основные отличия данных инструментальных средств сводятся к особенностям модели структуры графического интерфейса, механизмам идентификации и взаимодействия с элементами графического интерфейса, языкам описания тестовых сценариев.

Отдельно стоит выделить инструмент GUITAR [7], разработанный в Мэрилендском университете. В отличие от указанных выше инструментов GUITAR позволяет составлять сценарии тестирования в полностью

автоматическом режиме без участия эксперта. За счёт данного свойства инструмент GUITAR наилучшим образом подходит для совмещения с другими автоматическими методами динамического анализа программ и в частности, динамического анализа на основе символьного исполнения.

Инструмент GUITAR имеет четыре взаимосвязанных компоненты, схема работы которых представлена на рис. 1.

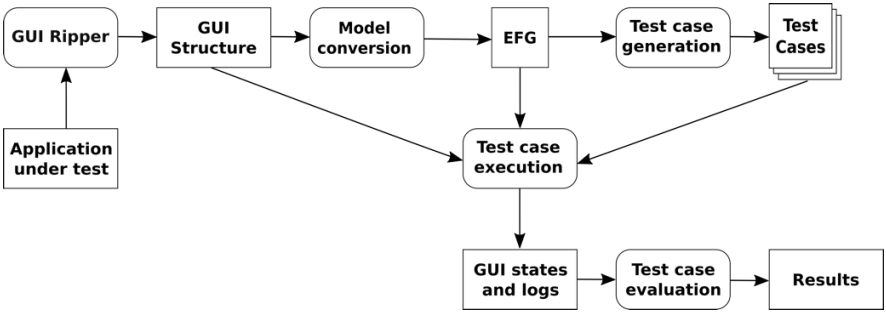


Рис. 1. Схема работы инструмента GUITAR

Fig. 1. GUITAR tool scheme

Практически каждое приложение с графическим пользовательским интерфейсом имеет несколько окон, которые открываются в результате определенных действий пользователя. Графический пользовательский интерфейс приложения представляет собой иерархическую структуру, в которой верхним элементом является главное окно, на средних уровнях располагаются контейнеры элементов графического пользовательского интерфейса и на нижних — контейнеры и отдельные элементы. Компонент инструмента GUITAR Ripper проводит анализ всех окон и иерархии контейнеров и элементов управления графического пользовательского интерфейса. При анализе окон, отображаемых на экране, компонент Ripper инициирует выполнение обработчиков событий взаимодействия с элементами интерфейса, находящихся в отображаемых окнах. Таким образом имитируются действия пользователя. Компонент Ripper сохраняет информацию об элементах графического интерфейса, об изменениях структуры графического пользовательского интерфейса, которые произошли после совершения действий, а также параметры отдельных объектов. Сохраняемая информация записывается в формате XML и составляет модель структуры графического интерфейса, с которой будет производиться дальнейшая работа инструмента.

После получения необходимой информации об элементах графического интерфейса, строится EFG (Event Flow Graph) – ориентированный граф потока событий. В нем описываются взаимосвязи между элементами графического интерфейса. Связи между элементами в графе описывают множество возможных последовательностей взаимодействия с приложением.

Произвольный путь в графе от одного из элементов в окне приложения, отображаемом при запуске, до другого элемента задает возможный тестовый сценарий. Генерация тестовых сценариев проводится компонентом Test case generator, осуществляющим выделение путей в графе по заданным алгоритмам (например, случайная выборка или исчерпывающая выборка всех путей заданной длины и т. д.). Воспроизведение тестового сценария осуществляется компонентом Replayer.

Описанный алгоритм предоставляет возможность инструменту автоматически формировать тестовое покрытие в виде воздействия на элементы графического пользовательского интерфейса и воспроизводить его.

2.2 Автоматизация обхода путей выполнения на основе символьного исполнения

Методы итеративного динамического анализа предоставляют возможность автоматически обходить возможные пути выполнения программ. Центральным механизмом в рамках итеративного динамического анализа является символьное исполнение. В рамках символьного исполнения для произвольного пути выполнения собирается трасса булевых ограничений (предикат пути), задающая связь между входными данными программы и внутренними данными программы, обрабатываемыми во время выполнения. В составляемых трассах ограничений можно выделять точки ветвления, соответствующие условным конструкциям. Модификация булевого ограничения, порожденного инструкциями в точке ветвления, позволяет описать предикат, соответствующий альтернативному пути выполнения.

Модифицированные предикаты пути передаются на обработку инструментам проверки выполнимости булевых формул, которые позволяют установить, является ли предикат выполнимым на каком-либо наборе значений входных данных. В случае выполнимости инструменты позволяют построить один из подобных наборов. Запуск программы на полученном наборе приведет к ее выполнению по альтернативному пути. Последовательный обход и обработка точек ветвления в программе позволяют осуществлять создание наборов входных данных для исследования различных путей выполнения, обеспечивая высокий уровень покрытия кода.

Существующие инструменты, использующие рассмотренные выше принципы, отличаются подходами к обработке кода программы с целью построения предикатов пути, дополнительными возможностями по оптимизации предикатов пути и метриками, определяющими последовательность обхода точек ветвления в программе.

Инструмент Java Pathfinder [8, 9], разработанный в NASA и представляющий из себя специализированную виртуальную машину Java, стал основой для создания различных инструментальных средств для анализа и тестирования Java-программ. В инструменте реализован подход к проведению

динамического анализа, базирующийся на символьном исполнении. Код исследуемой программы инструментируется с целью получения дерева состояний приложения. Данное дерево как пространство состояний анализируется с помощью проверки на модели (model-checking). В процессе динамического символьного исполнения строятся булевы ограничения, на основе которых генерируются наборы входных значений для тестирования программы. Java PathFinder позволяет генерировать тестовые входные данные и строить контрпримеры для отдельных свойств модели.

JDart [10] – инструмент динамического анализа программ, разработанный в Исследовательском центре Эймса NASA. Инструмент реализует символьное исполнение программ и состоит из двух основных модулей — *Executor* и *Explorer*. Модуль Executor отвечает за исполнение анализируемой программы и создание символьных ограничений на используемые данные. Данный модуль реализован как расширение инструмента Java PathFinder. Explorer позволяет находить неисследованные пути исполнения программ на основе символьных ограничений. Данный модуль использует библиотеку JConstraints в качестве дополнительного уровня абстракции для эффективного преобразования символьных выражений и предоставления общего интерфейса решателям булевых ограничений. Инструмент поддерживает такие решатели, как CORAL, SMTInterpol и Z3. Дополнительным модулем JDart является JUnit для генерации тестовых наборов.

Инструмент JavaFAN [11] предназначен для формального анализа многопоточных Java-программ, как на уровне исходного кода, так и на уровне байт-кода виртуальной машины Java. Инструмент поддерживает символьное исполнение программ, обход в ширину пространства состояний программы с целью нахождения нарушений свойств безопасности, проверку на модели свойств логики линейного времени (LTL) для программ с конечным пространством состояний. В ходе работы инструмента Java-программа транслируется в Maude-описание, что позволяет проводить более эффективное исполнение, поиск и проверку на модели LTL.

Инструмент Coffee Machine [12], разработанный в Институте системного программирования РАН, предоставляет возможности статической инструментации байт-кода программ Java с целью создания трасс булевых ограничений и трасс событий, использующихся для поиска дефектов синхронизации [13]. Управляющие модули инструмента позволяют организовывать обход возможных путей выполнения программы и интеграцию с решателями булевых ограничений для построения наборов путей.

Применение статической инструментации делает средство Coffee Machine независимым от конкретной реализации виртуальной машины Java, что расширяет область применимости инструмента (в частности, позволяет осуществлять анализ приложений операционной системы Android, использующей виртуальную машину Dalvik).

3. Особенности предлагаемого совмещенного метода анализа

По итогам анализа предметной области в качестве базовых инструментов для исследования и разработки методов совмещенного анализа приложений, предоставляющих графический пользовательский интерфейс, были выбраны средства GUITAR и Coffee Machine. Выбор инструментов обусловлен следующими обстоятельствами:

- Инструменты GUITAR и Coffee Machine распространяются свободно вместе с исходным кодом, что позволяет проводить модификацию и расширение его функциональности;
- Инструмент GUITAR предоставляет поддержку ряда библиотек работы с графическим интерфейсом (SWT, Swing для стандартной виртуальной машины Java HotSpot; библиотека организации интерфейса приложений операционной системы Android для виртуальной машины Dalvik);
- Инструмент Coffee Machine использует статическую инструментацию байт-кода для обработки программ, что позволяет использовать его для различных виртуальных машин, включая целевые Java HotSpot и Dalvik. Модули инструментации, проводящие модификацию кода с целью создания трасс булевых ограничений при выполнении программы, могут использоваться независимо от управляющего модуля Coffee Machine, что позволяет с легкостью встроить их в инструмент GUITAR.

В рамках совмещенного метода анализа используется схема инструмента GUITAR для определения модели структуры графического интерфейса исследуемой программы и выявления связей между элементами графического интерфейса для создания тестовых сценариев. Определение модели и исследование тестовых сценариев проводится для заранее проинструментированной программы, что позволяет строить фрагменты трасс ограничений для функций-обработчиков событий взаимодействия с элементами пользовательского интерфейса. Фрагменты трасс объединяются в более сложные конструкции для определения тестовых наборов, позволяющих исследовать перспективные точки ветвления в обработчиках событий взаимодействия с элементами графического интерфейса.

3.1 Символьное исполнение программ с графическим пользовательским интерфейсом

Существующие инструменты динамического анализа на основе символьного исполнения стандартно обрабатывают такие источники входных данных, как файлы, сетевые интерфейсы, переменные окружения и др. Исследование того, как программа производит доступ к содержимому данных источников, обычно

реализуется с помощью перехвата входных данных, полученных от функций и системных вызовов. Перехваченные данные используются программой непосредственно и явно участвуют в потоке данных. Составление предиката пути позволяет задать функциональную зависимость между символьными переменными, описывающими значения входных данных и значения операндов инструкций.

Для приложений с графическим интерфейсом понятие входных данных определяется более широко. Помимо рассмотренных выше источников (файлы, сетевые интерфейсы и др.) в качестве входных данных необходимо рассматривать взаимодействие пользователя с элементами интерфейса. Последовательность действий (тестовый сценарий) непосредственно определяет работу программы, однако в явном виде не может быть связана с потоком данных, обрабатываемым инструкциями кода (на уровне ячеек памяти, регистров и т. д.). Для обеспечения возможности манипуляции составляющими тестового сценария необходимо ввести в трассу ограничений новые символьные переменные, задающие идентификаторы элементов графического интерфейса, и новые ограничения, связывающие данные переменные с потоком данных программы. Подобные модификации трассы ограничений являются частичным решением известной проблемы символьного исполнения, относящейся к так называемым зависимостям по потоку управления.

Построение трасс булевых ограничений в рамках совмещенного подхода базируется на следующих принципах:

- В трассы ограничений попадают только инструкции, выполняющиеся в функциях обработки действий над элементами графического интерфейса (например, в функциях, выполняющихся как отклик на нажатие кнопки или изменение состояния переключателя) или до вызова первой функции обработки действий.
- При отслеживании потока данных в программе применяется модель перепомеченности – все долгоживущие объекты, которые могут использоваться одновременно в нескольких функциях обработки действий над элементами графического интерфейса, рассматриваются как символьные переменные. Для байт-кода Java к таким объектам относятся:
    - Содержимое статических полей примитивных типов загруженных виртуальной машиной классов.
    - Массивы примитивного типа и поля объектов, доступ к которым можно осуществить из статических полей загруженных виртуальной машиной классов.
- Промежуточные объекты, создаваемые внутри функций обработки действий над элементами графического интерфейса, рассматриваются как символьные переменные только в том случае, если они напрямую зависят от объектов, указанные выше.

- Фрагмент предиката пути, порождаемый инструкциями функции обработки действия над элементом графического интерфейса, обрамляется специальными маркерами начали и конца события.

3.2 Сбор предикатов пути и составление тестовых сценариев

Обработка программы при помощи инструмента GUITAR осуществляется для исполняемых файлов, инструментированных с целью построения трасс ограничений. Поэтому при составлении модели структуры графического интерфейса существует возможность сопоставить каждой паре {элемент графической интерфейса, действие над графическим интерфейсом} фрагмент подтрассы. В получаемых фрагментах подтрасс выделяются точки ветвления. По общей схеме проведения динамического анализа на основе символьного исполнения осуществляется модификация ограничения в точке ветвления для обхода альтернативного пути выполнения. Однако расчёт входных данных для модифицированной трассы ограничений необходимо осуществлять с учетом других фрагментов трасс, соответствующих обработке действий над элементами графического интерфейса, т. к. именно последовательность действий и является набором входных данных.

Для достижения подобного эффекта в рамках GUITAR используется модифицированный модуль генерации тестовых сценариев. Данный модуль составляет трассу ограничений особой структуры, учитывая граф потока событий и модель графического интерфейса. Общая структура трассы выглядит следующим образом:

$$
\begin{gathered}
\big(((e_1 = i_{11}) \wedge (e_2 = i_{12}) \wedge \dots \wedge (e_N = i_{1N})) \vee \\
((e_1 = i_{21}) \wedge (e_2 = i_{22}) \wedge \dots \wedge (e_N = i_{2N})) \vee \\
\dots \\
((e_1 = i_{K1}) \wedge (e_2 = i_{K2}) \wedge \dots \wedge (e_n = i_{KN}))) \wedge \\
((((e_1 = i_1) \wedge it_1) \vee ((e_1 = i_2) \wedge it_2) \vee \dots \vee ((e_1 = i_M) \wedge it_M)) \vee \\
(((e_2 = i_1) \wedge it_1) \vee ((e_2 = i_2) \wedge it_2) \vee \dots \vee ((e_2 = i_M) \wedge it_M)) \vee \\
\dots \\
(((e_{N-1} = i_1) \wedge it_1) \vee ((e_{N-1} = i_2) \wedge it_2) \vee \dots \vee ((e_{N-1} = i_M) \wedge it_M))) \wedge \\
((e_N = i_X) \wedge it_X)
\end{gathered}
$$

Здесь:

e_i - специальная символьная переменная, определяющая i-ое событий в тестовом сценарии; данные переменные могут принимать значения i_1, ..., i_M, соответствующие всем событиям (парам <элемент; графического интерфейса, действие над графическим интерфейсом>) в модели

it_i – фрагмент трассы, порождаемый функцией обработки события i_i;

i_X – событие, точка ветвления в подтрассе которого обрабатывается модулем генерации тестовых сценариев;

i_{i1}, …, i_{iN} – возможный путь в графе потока событий;

N – максимальная длина тестового набора;

M – количество событий в модели графического интерфейса;

K – количество возможных путей в графе потока событий.

Инструмент проверки выполнимости булевых формул осуществляет обработку данной трассы. В случае её выполнимости инструмент предоставляет набор значений e_1, ..., e_N, который:

- позволяет обойти нужный путь выполнения по обрабатываемой точке ветвления;
- задает корректный тестовый сценарий согласно модели графического интерфейса.

Тем самым в рамках совмещенного режима осуществляется построение тестовых сценариев, направленных на исследование точек ветвления внутри обработчиков событий, что позволяет обнаруживать не только дефекты проектирования самого интерфейса, но и раскрывать ошибки во внутренних алгоритмах обработки данных.

3.3 Подход к генерации опасных входных значений

Ряд уязвимостей в программном обеспечении связан с отсутствием проверок на разрешенные значения обрабатываемых данных. Так, достаточно распространенной ошибкой является считывание данных из полей ввода (EditText) и их дальнейшая обработка без предварительной проверки на соответствие определенному формату. Например, рассмотрим следующую конструкцию:

```
float f =
Float.parseFloat(editText.getText().toString());
```

Если введенные в текстовое поле данные содержат недопустимые символы, то средствами виртуальной машины будет создано исключение `java.lang.NumberFormatException`. Анализ программы с помощью символьного исполнения построенного на инструментации кода Java-программы не сможет обнаружить данную уязвимость, если точки ветвления, отвечающие за проверку формата входных данных, скрыты в реализации виртуальной машины. К тому же, инструмент не имеет возможности генерировать данные для ввода в текстовые поля напрямую, но в случае, когда содержимое таких полей формируется посредством взаимодействий с другими элементами графической структуры (нажатий на кнопки и т. д.), становится возможным создание механизма для обнаружения такого рода ошибок. Для решения данной проблемы добавляется искусственный условный переход перед соответствующей уязвимой операцией. Условие в таком переходе

должно соответствовать проверке опасных данных на корректность, тогда во время инвертирования этого перехода будет получена последовательность событий, формирующая данные, приводящие к ошибке. Подход может быть реализован с помощью внедрения перед уязвимой инструкцией метода-симулятора, генерирующего трассу условного перехода.

В рамках реализации совмещенного метода в инструмент Coffee Machine был добавлен ряд симуляторов, направленных на обнаружение подобных ошибок, включающий проверку функций трансформации и обработки примитивных типов языка Java.

4. Оценка предложенного решения

Для исследования эффективности разработанного метода анализа было проведено тестирование инструмента на наборе реальных проектов. Для полноценной проверки качества анализа приложения выбирались в соответствии со следующими критериями:

- большое количество элементов управления;
- наличие скрытой за интерфейсом вычислительной логики приложения.

Данные требования необходимы для создания нагрузки на решатель булевых ограничений, увеличении количества помеченных данных и числа итераций. Исходя из этих критериев были проанализированы следующие проекты:

- Calculator – примитивный калькулятор для платформы Android;
- ProgCalc – калькулятор для платформы Android, позволяющий проводить вычисления в различных системах счисления;
- Calculator – калькулятор для платформы Swing.

В качестве контрольного испытания был проведен анализ предложенных программ инструментом GUITAR без модификаций, результаты которого приведены в таблице 1. Анализ заключается в построении модели графического интерфейса тестируемого приложения, генерации на основе этой модели тестового покрытия и его последующего воспроизведения. Тестовое покрытие представляет собой набор отдельных тестов – фиксированное количество взаимодействий с приложением. Тесты представляют собой все возможные комбинации элементов интерфейса, поэтому размер тестового покрытия зависит от величины модели и от количества взаимодействий в каждом тесте. Для проведения анализа было сгенерировано тестовое покрытие, состоящее из двух взаимодействий, как оптимальное с точки зрения времени воспроизведения и достаточности для обнаружения дефектов. Из-за величины тестового покрытия такой анализ проводится достаточно долго, причем большее время анализа Android-приложений обусловлено задержками при взаимодействии фреймворка GUITAR и Android-устройства (эмулятора). В результате проведенного

тестирования были обнаружены программные дефекты в приложениях и достаточно большое количество входных данных для их воспроизведения. В приложении Calculator (Android) было найдено 3 дефекта, связанных с необработанной исключительной ситуацией типа `java.lang.NumberFormatException`, и 17 различных последовательностей взаимодействий с пользовательским интерфейсом для их воспроизведения. В приложении ProgCalc (Android) было обнаружено 2 уязвимости: ошибка деления на ноль (`java.lang.ArithmeticException`) и переполнения окна ввода, приводящего к необработанной исключительной ситуации неверного формата числа (`java.lang.NumberFormatException`), для каждой уязвимости по одной последовательности событий. В приложении Calculator (Swing) была найдена 1 уязвимость, связанная с обработкой недопустимых символов (`java.lang.NumberFormatException`), и 5 различных последовательностей событий, позволяющих воспроизвести дефект.

Табл. 1. Результаты инструмента GUITAR

Table 1. GUITAR results

Проект		Calculator (Android)	PrgCalc (Android)	Calculator (Swing)
Найдено дефектов		3	2	1
Покрытие		587	1507	608
Время (ч)		4	12,5	0,66
Размер модели	Окон	1	1	2
	Элементов	42	51	47
	Событий	2	30	30

Результаты применения предложенного метода представлены в «Табл. 2». Динамический анализ на основе символьного исполнения позволил существенно сократить время тестирования, однако в результате было найдено меньшее число уязвимостей, чем при воспроизведении тестового покрытия. Это связанно со слишком сильным ограничением длины цепочек событий графического пользовательского интерфейса, которое принудительно было выставлено в значение 2. Так, в приложении Calculator (Android) было обнаружено только 2 уязвимости и 2 набора уязвимых входных данных для их воспроизведения. В приложении ProgCalc (Android) был найден один дефект, связанный с делением на ноль, а в приложении Calculator (Swing) не было найдено ни одного дефекта. Прежде всего, это обусловлено тем, что некоторые уязвимости не находятся на отдельном пути исполнения

программы, и воспроизведение только одного набора данных, позволяющего достичь такого пути, недостаточно. Часть уязвимостей была обнаружена с помощью механизма построения условий, выполнимость которых означает возникновение ошибки на данном пути, однако ограничения этого метода не позволили найти остальные уязвимости. Но в связи с ограничением на длину цепочки событий графического пользовательского интерфейса в два события у инструмента не было возможности ввести слишком длинное число, чтобы спровоцировать исключительную ситуацию неправильного формата числа. Предложенная схема работы инструмента производит обновление модели на каждой итерации, однако в данном случае модели остались неизменными в силу того, что рассматриваемые приложения либо не имеют дополнительных элементов, либо они не становятся активными при исследовании путей исполнения.

Табл. 2. Результаты символьного исполнения

Table 2. Symbolic execution results

Проект		Calculator (Android)	PrgCalc (Android)	Calculator (Swing)
Найдено дефектов		2	1	0
Количество итераций		35	17	21
Время (ч)		0,42	0,33	0,06
Размер модели	Окон	1	1	2
	Элементов	42	51	47
	Событий	2	30	30

5. Заключение

В рамках работы, представленной в статье, разработан совмещённый метод динамического анализа программ, предоставляющих графический пользовательский интерфейс. Данный метод включает в себя автоматическое извлечение модели графического пользовательского интерфейса, описывающей возможные последовательности взаимодействия с элементами интерфейса (тестовые сценарии), и направленное построение тестовых сценариев, позволяющих обойти возможные пути в программе.

Для реализации метода были использованы свободно распространяемые инструменты GUITAR и Coffee Machine; инструмент GUITAR был расширен генератором тестовых сценариев, который вычисляет тестовый сценарий используя символьное исполнение.

Данный метод позволяет целенаправленно добиваться увеличения полноты покрытия кода при анализе программ и обхода точек ветвления в коде обработчиков событий взаимодействия с элементами графического пользовательского интерфейса.

С помощью полученного инструмента был проанализирован набор свободно распространяемых приложений на языке Java (библиотека управления графическим интерфейсом в составе ОС Android; библиотека управления графическим интерфейсом Swing). С помощью направленного обхода внутренних точек ветвления в обработчиках событий были обнаружены дефекты, связанные с некорректной обработкой исключительных ситуаций.

В то же время, при проведении практических экспериментов были замечены ограничения представленного метода. В первую очередь, конструирование трасс с перебором действий над элементами интерфейса приводит к ещё большему усугублению проблемы экспоненциального роста количества путей для анализа, присущей методам анализа на основе символьного исполнения. В статьях, посвящённых инструменту GUITAR, указывались практические ограничения на перебор сценариев взаимодействия, включающие фиксацию максимальной длины создаваемых сценариев. При работе в контексте символьного исполнения данное ограничение приходится усиливать.

Во-вторых, усложнение структур трасс привело к увеличению затрат на проверку выполнимости данных трасс; в общем случае данное увеличение не ограничивается линейными зависимостями.

На основе указанных недостатков можно определить направления дальнейших исследований в области анализа программ с графическим пользовательским интерфейсом. Технические улучшения работы разработанного инструмента, направленные на отслеживание особенностей графических элементов и их свойств, позволят составлять более точную модель графического пользовательского интерфейса, что в свою очередь снизит количество сценариев выполнения программы, требующих анализа. Второе направлений исследований связано с оптимизациями трасс ограничений и использования возможностей современных решателей булевых ограничений (инкрементальное решение, выявление групп зависимых условий и пр.).

Список литературы

[1]. Сайт Mobile Banking Rank 2015 (http://markswebb.ru/e-finance/internet-banking-rank-2015/)

[2]. Страница проекта Ranorex. http://www.ranorex.com/ [HTML]. Обращение от 10.10.2016

[3]. Abbot framework for automated testing of Java GUI components and programs. https://abbot.sourceforge.net. Обращение от 10.10.2016

[4]. Страница проекта Maveryx. https://sourceforge.net/projects/maveryx/ [HTML]. Обращение от 10.10.2016

[5]. Страница проекта Squish. https://www.froglogic.com/squish/ [HTML]. Обращение от 10.10.2016

[6]. Sikuli: using GUI screenshots for search automation. UIST'09 Proceedings of the 22nd annual ACM symposium on User Interface software and technology. Victoria, BC, Canada, October 04-07, 2009. pp. 183-192

[7]. Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, Atif Memon. GUITAR: an innovative tool for automated testing of GUI-driven software. Automated software engineering, 2013, vol. 21(1), pp. 65-105.

[8]. Willem Visser, Corina S. Pāsāreanu, Sarfranz Khurshid. Test input generation with java PathFinder. ISSTA '04 Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis. Boston, Massachusetts, USA, July 11-14, 2004, pp. 97-107.

[9]. Gary Lindstrom, Peter C. Mahlitz, Willem Visser. Model checking real time java using java pathfinder. Proceeding ATVA'05 Proceedings of the Third international conference on Automated Technology for Verification and Analysis, Taipei, Taiwan, October 04-07, 2005, pp. 444-456.

[10]. Howar, Falk, Malte Isberner, Temesghen Kahsai, Zvonimir Rakamaric, and Vishwanath Raman. "JDART: A Dynamic Symbolic Analysis Framework." In Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings, vol. 9636, Springer, 2016. p. 442.

[11]. Farzan A, Chen F, Meseguer J, Roşu G. Formal analysis of Java programs in JavaFAN. International Conference on Computer Aided Verification 2004 Jul 13. Springer Berlin Heidelberg. pp. 501-505

[12]. М.К. Ермаков, С.П. Вартанов. Применение статической инструментации байт-кода языка Java для динамического анализа программ. Труды Института системного программирования РАН, том 27, вып. 1, 2015 г., стр. 25-38. DOI: 10.15514/ISPRAS-2015-27(1)-2

Dynamic analysis of programs with graphical user interface based on symbolic execution

S.P. Vartanov <svartanov@ispras.ru>
A.Y. Gerasimov <agerasimov@ispras.ru>
M.K. Ermakov <mermakov@ispras.ru>
D.O. Kutz <kutz@ispras.ru>
A.A. Novikov <a.novikov@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. In this paper, we explore the possibilities of applying dynamic symbolic execution (or concolic testing) methods to applications with graphical user interfaces. Such applications inherently feature interactive user input processing and multithreaded execution. These features typically decrease the effectiveness of dynamic symbolic execution by increasing the

volume of processed code not related to actual application functionality. We present a hybrid approach that combines commonly used GUI test automation methods based on GUI model excavation with dynamic symbolic execution methods to construct test cases for checking internal application logic. We have implemented this approach using two open-source tools – test automation framework GUITAR and Java byte-code static instrumentation framework Coffee Machine. GUI model extracted automatically by GUITAR tool is extended with symbolic traces relevant to application GUI event handlers. Our test generation module for GUITAR combines these symbolic traces into complex queries to be processed by SMT solver. The resulting test cases are valid within automatically extracted GUI structure model and allow to check different execution paths in GUI event handler code. We have checked our hybrid approach on a set of small open-source applications and identified several bugs caused by uncaught exceptions. The paper is concluded with an overview of current limitations and possible improvements of the hybrid approach.

Keywords: dynamic analysis; program analysis; GUI testing; test coverage.

DOI: 10.15514/ISPRAS-2017-29(1)-10

For citation: Vartanov S.P., Gerasimov A.Y., Ermakov M.K., Kutz D. O., Novikov A.A. Dynamic analysis of programs with graphical user interface. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 1, 2017. pp. 149-166 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-10

References

[1]. Mobile Banking Rank 2015 (http://markswebb.ru/e-finance/internet-banking-rank-2015/)
[2]. Ranorex. http://www.ranorex.com/ [HTML]. Accessed at 10.10.2016
[3]. Abbot framework for automated testing of Java GUI components and programs. https://abbot.sourceforge.net. Accessed at 10.10.2016
[4]. Maveryx. https://sourceforge.net/projects/maveryx/ [HTML]. Обращение от 10.10.2016
[5]. Squish: https://www.froglogic.com/squish/ [HTML]. Обращение от 10.10.2016
[6]. Sikuli: using GUI screenshots for search automation. UIST'09 Proceedings of the 22nd annual ACM symposium on User Interface software and technology. Victoria, BC, Canada. Octomer 04-07, 2009. pp. 183-192
[7]. Bao N. Nguyen, Bryan Robbins, Ishan Banerjee, Atif Memon. GUITAR: an innovative tool for automated testing of GUI-driven software. Automated software engineering. - 2013. - Vol. 21(1). pp. 65-105.
[8]. Willem Visser, Corina S. Pāsāreanu, Sarfranz Khurshid. Test input generation with java PathFinder. ISSTA '04 Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis. Boston, Massachusetts, USA. July 11-14, 2004. pp. 97-107.
[9]. Gary Lindstrom, Peter C. Mahlitz, Willem Visser. Model checking real time java using java pathfinder. Proceeding ATVA'05 Proceedings of the Third international conference on Automated Technology for Verification and Analysis. Taipei, Taiwan. October 04-07, 2005. pp. 444-456.

[10]. Howar, Falk, Malte Isberner, Temesghen Kahsai, Zvonimir Rakamaric, and Vishwanath Raman. "JDART: A Dynamic Symbolic Analysis Framework." In Tools and Algorithms for the Construction and Analysis of Systems: 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings, vol. 9636, Springer, 2016. p. 442.

[11]. Farzan A, Chen F, Meseguer J, Roşu G. Formal analysis of Java programs in JavaFAN. InInternational Conference on Computer Aided Verification 2004 Jul 13. Springer Berlin Heidelberg. pp. 501-505

[12]. M.K. Ermakov, S.P. Vartanov. Applying Java bytecode static instrumentation for software dynamic analysis. Trudy ISP RAN / Proc. ISP RAS, vol 27, issue 1, 2015, pp. 25-38 (in Russian). DOI: 10.15514/ISPRAS-2015-27(1)-2.

Обзор методов и средств генерации тестовых программ для микропроцессоров

А.Д. Татарников <andrewt@ispras.ru>
Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25

Аннотация. В работе дается обзор существующих методов и средств генерации тестовых программ для микропроцессоров. Генерация тестовых программ и анализ результатов их выполнения являются основным подходом к функциональной верификации микропроцессоров. Этот подход также принято называть тестированием. Несмотря на то, что методы генерации тестовых программ непрерывно совершенствуются, тестирование остается крайне трудоемким процессом. Одна из основных причин состоит в том, что средства генерации не успевают адаптироваться к изменениям. В большинстве случаев они созданы под конкретные типы микропроцессоров и предназначены для решения конкретных задач. Поэтому поддержка новых типов микропроцессоров и новых методов генерации требует значительных усилий. Часто в таких случаях приходится создавать новую реализацию с нуля. Невозможность повторного использования имеющейся реализации затрудняет развитие средств генерации и, как следствие, препятствует улучшению качества тестирования. Текущее положение дел создает мотивацию для поиска решений, позволяющих создавать более гибкие средства генерации, которые могли бы быть с минимальными усилиями адаптированы для тестирования новых типов микропроцессоров и применения новых методов генерации. Цель данной работы – обобщить имеющийся опыт генерации тестовых программ, который мог бы послужить основой для создания таких средств. В работе рассматриваются сильные и слабые стороны распространенных методов генерации, области их применения и варианты их совместного использования. Также делается сравнительный анализ возможностей существующих средств генерации, реализующих эти методы. На основе данного анализа даются рекомендации по созданию универсальной методологии разработки средств генерации тестовых программ для микропроцессоров.

Ключевые слова: микропроцессоры; цифровая аппаратура; функциональная верификация; тестирование; генерация тестовых программ.

DOI: 10.15514/ISPRAS-2017-29(1)-11

Для цитирования: Татарников А.Д. Обзор методов и средств генерации тестовых программ для микропроцессоров. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 167-194. DOI: 10.15514/ISPRAS-2017-29(1)-11

1. Введение

Микропроцессор [1] является центральным элементом любого электронного устройства. В то время как другие компоненты отвечают за ввод и вывод данных, роль микропроцессора состоит в обработке этих данных: он определяет, какие операции должны быть выполнены над данными и контролирует процесс их выполнения. Микропроцессоры состоят из множества транзисторов, объединенных в единый вычислительный элемент на полупроводниковом кристалле. Число транзисторов постоянно увеличивается и в настоящее время достигает нескольких миллиардов [2].

Микропроцессоры являются сложными устройствами, поэтому их производство требует тщательного проектирования. В настоящее время для проектирования применяются специализированные языки, известные как *языки описания аппаратуры* (*HDL, Hardware Description Languages*), которые с предельной точностью позволяют описывать структуру и поведение микропроцессора. Результатом проектирования является *модель уровня регистровых передач* (*RTL, Register Transfer Level*), также называемая *HDL-моделью,* которая затем посредством логического и физического синтеза преобразуется в представление, используемое для производства интегральных схем.

Высокая сложность современных микропроцессоров приводит к неизбежности существования ошибок. Главной причиной этого является большое число комбинаций состояний микропроцессора и их взаимозависимостей. Дальнейшее усложнение, продиктованное погоней за повышением производительности, влечет за собой рост числа ошибок. Например, по данным на август 2008-го года в микропроцессоре Intel Pentium 4 было найдено 104 ошибки, из которых 43 не исправлено и их исправление не запланировано [3]. В то же время в микропроцессоре Intel Core i7-900 по данным на февраль 2015-го года обнаружено 167 ошибок, из которых исправлено только 16 и еще для 2 запланировано исправление [4]. Следует понимать, что в обоих случаях речь идет лишь об известных проблемах, в то время как реальное число ошибок может быть гораздо выше.

Цена ошибок в микропроцессорах очень высока. В отличие от дефектов в программах, которые устраняются сравнительно просто, ошибки в микропроцессорах не могут быть исправлены и для их устранения потребуется повторный выпуск и замена микросхемы или целого блока. Например, в 1994-м году замена продукции из-за ошибки в реализации инструкции FDIV микропроцессора Pentium обошлась компании Intel в 475 миллионов долларов [5].

Обеспечение функциональной корректности микропроцессоров является фундаментальной проблемой, для решения которой применяется комплекс мер, известный как *функциональная верификация* [6]. Верификация выполняется параллельно с проектированием, и ее задача заключается в проверке соответствия результатов, полученных на текущем этапе, заданным

требованиям и ограничениям. Верификация может осуществляться как на уровне отдельных модулей микропроцессора (*модульная верификация*), так и для устройства в целом (*системная верификация*).

Существуют два основных подхода к функциональной верификации: (1) *имитационная верификация* (simulation-based verification), также называемая *тестированием*, и (2) *формальная верификация* (formal method-based verification) [7]. Первый заключается в проверке корректности реакции проектируемого устройства, работа которого имитируется при помощи программного или аппаратного симулятора, на тестовые воздействия. Второй основан на построении математической (формальной) модели и проверке выполнимости ее свойств. Формальная верификация позволяет осуществить исчерпывающую проверку всех возможных вариантов поведения, заданных моделью. Однако она имеет ряд ограничений (высокая трудоемкость и вычислительная сложность, а также трудности формализации), которые затрудняют ее использование в промышленных проектах. По этой причине на практике основной акцент делается на тестирование [8].

На системном уровне тестирование осуществляется путем генерации *тестовых программ* на языке ассемблера, которые представляют собой последовательности команд (инструкций) микропроцессора, и анализа трасс их выполнения с целью убедиться, что их поведение соответствует спецификации. Такой подход видится наиболее естественным, т.к. система команд определяет функциональность микропроцессора и является единственным доступным интерфейсом, через который пользователи могут с ним взаимодействовать.

Тестовые программы создаются при помощи специальных программных средств, известных как *генераторы тестовых программ*, которые используют широкий набор методов генерации для обеспечения максимального покрытия тестируемой функциональности. Методы генерации эволюционировали по мере усложнения микропроцессоров от случайной генерации до нацеленной генерации, основанной на формальных методах (известны как *гибридные* методы [9]). Применяемые в настоящий момент методы можно разделить на следующие основные группы [10]: (1) *случайная генерация*; (2) *комбинаторная генерация*; (3) *генерация на основе ограничений* и (4) *генерация на основе моделей*. Важно понимать, что каждый метод имеет свою область применения и ни один из них не является универсальным решением для всех задач верификации. На практике применяется целый набор методов, взаимно дополняющих друг друга. Этот набор может пополняться по мере изобретения новых методов.

В общем случае генерация осуществляется на основе *тестовых шаблонов*, абстрактных описаний тестовых программ, которые задают используемые инструкции, их порядок и входные данные. Способы построения последовательностей инструкций и генерации тестовых данных зависят от методов генерации и используемых ими настроек.

Реализация инструментов генерации, основанных на различных методах, может существенно отличаться. Методы, позволяющие сгенерировать более точные тестовые воздействия, требуют большего количества информации о тестируемом микропроцессоре. Например, если методам случайной генерации, как правило, нужна только информация о синтаксисе команд, то для генерации тестов для подсистемы памяти необходимо знание о компонентах подсистемы памяти и семантике команд чтения и записи [11]. Это затрудняет интеграцию методов в единый инструмент. В результате, на практике для тестирования одного микропроцессора может применяться несколько инструментов, реализующих различные методы и использующих описания тестируемого микропроцессора разной степени детализации. Это отрицательно влияет на эффективность тестирования, т.к. поддержка нескольких инструментов, использующих различные форматы описания конфигурации, требует дополнительных усилий. Кроме того, совместное использование различных методов для решения общей задачи генерации часто оказывается невозможным.

Другая проблема, которая часто возникает, – это адаптация инструмента к изменениям в архитектуре тестируемого микропроцессора и поддержка новых архитектур. Многие инструменты генерации разработаны под конкретный микропроцессор и не предусматривают возможность подобных изменений (логика генерации тесно связана с конфигурацией). В таких случаях требуется внесение серьезных изменений в реализацию инструмента, что влечет за собой значительные трудозатраты.

Цель данной работы – проанализировать задачи, решаемые существующими генераторами тестовых программ, и присущие им проблемы. На основании этой информации будут сформулированы основные требования, которым должен удовлетворять универсальный инструмент генерации.

Оставшаяся часть статьи организована следующим образом. В разд. 2 рассказывается о том, как осуществляется верификации при помощи тестовых программ. В разд. 3 дается обзор существующих методов генерации. Раздел 4 содержит описание подхода к тестированию и инструмента генерации, используемого в НИИСИ РАН. Разд. 5 посвящен описанию инструментов, разрабатываемых и применяемых в компании ARM. В разд. 6 дается обзор инструментов IBM Research. Разд. 7 содержит краткий анализ методов и инструментов генерации, созданных в других компаниях. В разд. 8 делаются выводы и формулируется требования, которым должен удовлетворять универсальный инструмент генерации. Разд. 9 завершает статью.

2. Верификация при помощи тестовых программ

Верификация при помощи тестовых программ предполагает их выполнение на HDL-модели и последующий анализ результатов. Существуют два основных подхода к проверке корректности поведения HDL-модели [6]: (1) сравнение

трасс выполнения с эталонными трассами и (2) использование *встроенных проверок* (*self-checks*).

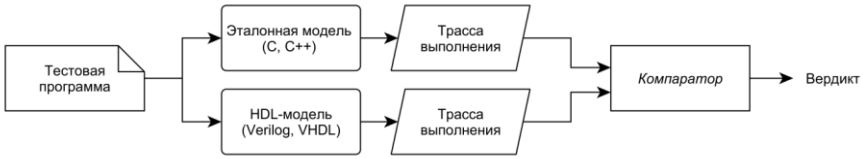


Рис. 1. Тестирование посредством сравнения трасс

Fig 1. Testing by comparing the traces

В первом случае (см. Рис. 1) при выполнении программы на HDL-модели создается трасса выполнения, фиксирующая события, которые происходят микропроцессоре. Полученная трасса сравнивается с эталонной трассой, полученной в результате выполнения той же программы на *эталонной модели (reference model)*. Для сравнения трасс используются специальные программы, называемые *компараторами*. Эталонная модель создается независимо от HDL-модели на языке высокого уровня (например, C или C++) и является более абстрактной. Часто такая модель представляет собой *симулятор уровня инструкций (instruction-level simulator)*. Более абстрактная модель, как правило, содержит меньшее количество ошибок, а тот факт, что она разрабатывается независимо, уменьшает вероятность повторения ошибок, допущенных при создании HDL-модели. Данный подход позволяет универсальным образом осуществлять проверку корректности поведения микропроцессора при выполнении различных тестовых программ. Таким образом, отпадает необходимость реализовывать проверки для отдельных тестов. К недостаткам подхода, можно отнести то, что трассы, полученные при помощи упрощенной эталонной модели, могут не отражать все события, которые происходят в HDL-модели (и реальном процессоре). Это приводит к трудностям при сопоставлении трасс.

Второй подход предполагает, что код тестовой программы содержит проверки, которые необходимо осуществить в процессе ее выполнения. Такие программы можно выполнять не только на HDL-моделях, но и на аппаратных ускорителях, ПЛИС-прототипах и экспериментальных образцах микросхем. Это позволяет повысить производительность, так как время выполнения программы на аппаратных симуляторах существенно меньше, чем на программных. Однако при таком подходе снижается точность проверки, так как множество событий, которые может фиксировать тестовая программа, ограничено.

Решение о завершении верификации принимается на основе набора критериев, который включает в себя следующее [12]:

- выполнение плана верификации;
- отсутствие ошибок при прогоне регрессионных тестов;

- отсутствие ошибок на 10^5 псевдослучайных тестов для каждого из имеющихся тестовых шаблонов;
- достижение заданного уровня покрытия HDL-кода и функционального покрытия;
- отсутствие изменений в HDL-коде в течение длительного времени (3-4 недели);
- отсутствие новых ошибок в течение определенного времени (2-3 недели);
- истечение отведенного согласно плану времени на разработку и верификацию.

Особого внимания заслуживают критерии, основанные на достижении требуемого уровня покрытия. Достигнутый уровень покрытия оценивается при помощи *метрик тестового покрытия* [13]. Они представляют собой количественные характеристики полноты покрытия выбранной *модели тестового покрытия,* которая описывается как конечный набор *тестовых ситуаций*. Числовое значение метрики покрытия представлено как отношение числа достигнутых при выполнении теста ситуаций к общему числу ситуаций в модели. Выделяют два типа моделей покрытия: (1) основанные на реализации и (2) основанные на функциональных требованиях.

В моделях первого типа тестовые ситуации связаны непосредственно с кодом HDL-модели или с производными от нее структурами. Т.е. метрики покрытия на основе реализации позволяют оценить, в какой мере код RTL-модели был задействован при выполнении набора тестов. Примерами таких метрик являются: покрытие строк кода (line coverage), покрытие операторов (statement coverage), покрытие переходов управления (branch coverage), а также покрытие состояний, дуг и путей (state, arc, transition coverage) конечных автоматов [14]. Большинство современных средств автоматизированного проектирования предоставляют инструментарий для сбора и анализа данных о достигнутом покрытии реализации.

В моделях второго типа, известных как *модели функционального покрытия*, тестовые ситуации задаются в терминах абстракций микроархитектуры, архитектуры микропроцессора или абстракций более высокого уровня. Такие модели создаются на основе спецификаций функциональных требований к микропроцессору различного уровня. Они могут создаваться вручную или посредством анализа *формальных спецификаций* [15].

Важно отметить, что ни одна из метрик тестового покрытия, взятая в отдельности не даёт ответа на вопрос о полноте набора тестов. Поэтому на практике используются комбинации различных метрик, основанных как на реализации, так и на функциональных требований.

Для генерации тестов, предназначенных для достижения требуемого уровня покрытия для выбранных метрик, применяются разнообразные методы генерации тестовых программ, реализованные различными инструментами.

3. Методы генерации тестовых программ

Большинство современных инструментов в той или иной степени полагаются *случайную генерацию* [16]. Однако применяемые методы генерации могут существенно отличаться степенью *нацеленности*. Под нацеленностью следует понимать ориентированность на покрытие конкретных тестовых ситуаций или классов тестовых ситуаций. Следует заметить, что т.к. генерация тестовых программ предполагает повторяемость результатов, алгоритмы случайной генерации, как правило, основаны на детерминированных генераторах псевдослучайных чисел.

В простейшем случае инструкции и их входные значения выбираются случайным образом [17]. Инструмент, решающий подобную задачу, можно разработать достаточно быстро, и он позволит сгенерировать набор тестов, обеспечивающих базовый уровень покрытия. К сожалению, такой подход не является систематическим и не позволит достичь полного покрытия. Подобные инструменты не располагают сведениями о семантике инструкций и особенностях реализации микроархитектуры, поэтому не гарантируют достижения всех интересных с точки зрения тестирования ситуаций, а также корректности построенной программы (отсутствие зацикливаний, переходов на секцию данных, нарушений инвариантов инструкций, и т.д.).

Первый шаг в направлении повышения нацеленности случайных тестов – ограничение области допустимых значений для случайного выбора и присвоение *весов* выбираем вариантам [18]. Такой подход предполагает описание задач генерации в виде шаблонов, которые задают списки инструкций для выбора (в том числе иерархические), веса отдельных элементов и области значений их входных аргументов. Это позволит более тщательно протестировать отдельные инструкции или классы инструкций. Веса и области значений подбираются на основе полученных значений метрик тестового покрытия.

Следующий шаг в сторону улучшения покрытия предполагает использование методов *комбинаторной генерации* [19]. Такой подход позволит нацелить генератор на тестирование ситуаций, связанных с совместным выполнением инструкций на конвейере. Суть метода заключается в систематическом переборе коротких последовательностей инструкций (включая зависимости между ними). Помимо этого, можно получать входные аргументы инструкций путем перебора данных из некоторого предопределенного набора. Это позволит достичь лучшего покрытия *пограничных случаев (corner cases)*. Путем совместного использования методов случайной и комбинаторной генерации можно построить более сложные тесты, которые помогут покрыть

многие маловероятные и трудновообразимые ситуации в работе микропроцессора.

Для проверки поведения микропроцессора в конкретных ситуациях часто прибегают к *детерминированной симуляции (deterministic simulation)* [20]. Идея заключается в создании и симуляции тестовых программ с детерминированным поведением, включающих встроенные проверки. Изначально такие программы разрабатывались вручную, что делало их создание крайне трудоемкой задачей. Автоматическая генерация таких программ предполагает симуляцию инструкций в процессе генерации и использование результатов для создания проверок. Симуляция в процессе генерации также позволяет гарантировать корректность построенных программ. При данном подходе программы стоятся по шаблонам, которые описывают тестовый сценарий. Симуляцию генератор осуществляет самостоятельно или использует для этого внешние эталонные модели. Построенные таким образом тесты можно считать полностью нацеленными. Основным недостатком данного подхода остается трудоемкость создания тестов, которая сопоставима с ручной разработкой. Как и при ручной разработке, чтобы создать код, покрывающий нетривиальные ситуации в работе микропроцессора, от разработчика требуется глубокое знание особенностей микроархитектуры. Кроме того, есть риск, что некоторые ситуации будут пропущены. С целью улучшения покрытия и уменьшения трудозатрат шаблоны могут использовать рандомизацию или комбинаторный перебор. Например, построение тестовых программ может осуществляться путем перебора разнообразных графов потока управления (*структур переходов*), описанных в шаблоне, и маршрутов в них (*трасс выполнения*) [21].

Построение нацеленных тестов можно упростить при помощи *генерации на основе ограничений* [22]. При таком походе в шаблонах указываются ограничения, которым должны удовлетворять операнды инструкций. Ограничения соответствуют тестовым ситуациям, основанным на особенностях реализации микроархитектуры или на функциональных требованиях. В процессе обработки шаблона генератор, используя некоторый механизм разрешения ограничений, который зависит от их типа, строит случайные значения, удовлетворяющие заданным ограничениям. Использование ограничений позволяет значительно сократить трудозатраты, связанные с подбором значений операндов, требуемых для покрытия тестовых ситуаций. Однако, т.к. тестовые шаблоны по-прежнему разрабатываются вручную, остается риск упустить важную для тестирования ситуацию.

Систематическое тестирование может быть обеспечено путем создания *формальных моделей* тестируемого микропроцессора [23] и применения к ним методов проверки моделей для построения тестов, охватывающих все пространство состояний. Такой подход называют *генерацией на основе моделей*. Т.к. современные микропроцессоры имеют огромное количество

состояний и переходов между ними, у этого метода будут высокие вычислительные затраты. Для их сокращения можно использовать тестовые шаблоны, которые будут задавать направление тестирования. К недостаткам данного метода относятся высокая трудоемкость его реализации (пока ведутся исследования, но нет промышленных инструментов), а также трудности создания формальных моделей.

Как можно заметить, перечисленным методам требуется разное количество входных данных. В табл. 1 приводится список методов и требуемых для них входных данных в порядке увеличения их количества.

Табл. 1. Методы генерации и требуемые входные данные

Table 1. Methods of generation and required input data

Метод генерации	**Входные данные**
Случайная генерация	Ассемблерный формат инструкций; Тестовые шаблоны (списки инструкций, веса)
Комбинаторная генерация	Все выше перечисленное; Тестовые шаблоны (правила комбинирования)
Генерация детерминированных тестов со встроенными проверками	Все выше перечисленное; Тестовые шаблоны (описание тестовых сценариев); Семантика инструкций (эталонная модель)
Генерация на основе ограничений	Все выше перечисленное; Тестовые шаблоны (задание ограничений); Модели покрытия (наборы ограничений) различного типа
Генерация на основе моделей	Все выше перечисленное; Формальные модели микропроцессора

Перечисленные методы по-разному реализованы в различных инструментах генерации. Отдельный инструмент может поддерживать один или несколько методов. Особенности реализации методов в конкретных инструментах будут рассмотрены в последующих разделах.

4. Инструменты НИИСИ РАН

В НИИСИ РАН разрабатывается система INTEG [18][24], предназначенная для тестирования микропроцессоров MIPS64 [25]. Используемый ею подход к тестированию называется *стохастическим*. Он предполагает генерацию тестовых программ по заданному шаблону с параметризированным случайным выбором инструкций и их аргументов. После этого осуществляется симуляция сгенерированных программ на HDL-модели и на эталонной модели, в качестве которой выступает симулятор VMIPS [26], и сравнение полученных результатов.

Генератор тестовых программ системы INTEG поддерживает случайные и комбинаторные методы генерации. Тестовые шаблоны для него разрабатываются на специализированном языке, конструкции которого основаны на синтаксисе языка C. Система предоставляет графический

интерфейс, который упрощает их разработку. Шаблоны задают последовательность фрагментов кода в тестовой программе, состав входящих в них инструкций и аргументы этих инструкций. Все параметры, оставленные свободными, генератор выбирает случайно, в соответствии с заданными для них вероятностями. Основные возможности, предоставляемые языком описания тестовых шаблонов, включают: (1) задание областей памяти для кода и для данных; (2) описание последовательностей инструкций (в том числе конструкции для описания циклов); (3) выбор инструкций; (4) выбор регистров, используемых в качестве аргументов инструкций; (5) задание значений аргументов инструкций; (6) задание адресов данных и адресов передачи управления; (7) описание правил генерации случайных значений; (8) создание макросов для повторного использования шаблонного кода. При разработке тестовых шаблонов для системы INTEG руководствуются планом тестирования и метриками покрытия. Для устранения пробелов в покрытии изменяются настройки шаблонов, такие как степень случайности выбора, область допустимых значений, т.д.

В процессе генерации система INTEG отслеживает состояние микропроцессора, используя упрощенный подход. При таком подходе симуляция инструкций не осуществляется, а значение регистров в некоторой точке выполнения считается известным или неопределенным. При необходимости генератор обновляет значения регистров и обеспечивает защиту от записи в регистры, часто используемые для чтения.

Тестовые шаблоны, сгенерированные тестовые программы и отчеты о тестировании можно хранить в специальной базе данных. Это позволяет объединять информацию о работе нескольких экземпляров системы INTEG, запущенных на разных компьютерах.

Система INTEG предоставляет достаточно мощный и удобный инструментарий для создания случайных тестов. Средства разработки тестовых шаблонов, генерации тестовых программ, выполнения тестирования, а также анализа и хранения его результатов интегрированы в единую систему. Однако данная система имеет ряд ограничений:

- Поддерживаются только случайная и комбинаторная генерация. Остается неясно, какой уровень покрытия можно обеспечить этими методами и насколько сложно разработать шаблоны, удовлетворяющие требованиям по тестовому покрытию.
- Не предусмотрено добавление пользовательских методов генерации и пользовательских конструкций в язык описания тестовых шаблонов.
- В процессе генерации построенные инструкции не симулируется. Таким образом, генератор не может отслеживать состояния микропроцессора в любой точке выполнения тестовой программы. Эта информация необходима для контроля корректности сгенерированных программ и создания встроенных проверок. Кроме того она требуется методам генерации, основанным на разрешении

ограничений. Таким образом, реализация перечисленных возможностей столкнется со значительными трудностями.

- Поддерживается только архитектура MIPS64. В работе [24] есть упоминание о возможности настройки под другие архитектуры. Однако неясно, как такая настройка будет осуществляться и каких трудозатрат она потребует.

5. Инструменты ARM

5.1 Инструменты RIS

В компании ARM [27] разработано семейство инструментов генерации тестовых программ, получившее название RIS (Random Instruction Sequence) [28][29]. Это узкоспециализированные инструменты, предназначенные для решения различных задач тестирования микропроцессоров с архитектурой ARM. Решаемые ими задачи включают тестирование таких механизмов, как многоядерность [30] и управление памятью [31]. Настройка инструментов RIS осуществляется при помощи конфигурационных файлов, которые задают используемые инструкции, их веса, ограничения на их операнды, способы размещения кода и данных в памяти, а также цепочки инструкций, решающих специальные задачи (вытеснение данных из кэш-памяти и т.п.). Инструменты RIS, описанные в работах [30] и [31], не используют эталонные модели и не осуществляют симуляцию инструкций в процессе генерации. В тех случаях, когда требуются встроенные проверки, тесты выполняются дважды и полученные результаты сравниваются (применимо только для детерминированных результатов). Такой подход упрощает разработку инструмента и делает возможным его использование непосредственно на ПЛИС-прототипе или экспериментальном образце микропроцессора. К сожалению, доступная информация об инструментах RIS не позволяет в полной мере оценить их функциональные возможности. Из того, что известно можно сделать вывод, что основным ограничением инструментов RIS является то, что они жестко привязаны к архитектуре ARM и ориентированы на решение конкретных задач. Поддержка других архитектур и методов генерации в них не предусмотрена.

5.2 Инструмент RAVEN

Другим инструментом, используемым в компании ARM, является RAVEN (Random Architecture Verification Machine) [32][33][34], разработанный в компании Obsidian Software, поглощенной ARM в 2011 году. Это универсальное средство, применимое для широкого спектра архитектур, в котором ядро, реализующее логику генерации, отделено от конфигурации для конкретной архитектуры. RAVEN позволяет создавать как случайные, так и нацеленные тесты. В процессе работы инструмент отслеживает состояние

микропроцессора путем симуляции построенных программ на внешней эталонной модели. Это позволяет гарантировать корректность построенных программ и использовать информацию о текущем состоянии микропроцессора для генерации тестов.

Конфигурация тестируемого микропроцессора задается в виде XML-файлов. Данные файлы содержат следующую информацию об архитектуре тестируемого микропроцессора: (1) перечень поддерживаемых инструкций и их групп, организованный в виде дерева; (2) сигнатуры инструкций (ассемблерный синтаксис, двоичная кодировка); (3) операнды инструкций, их свойства, используемые ими режимы адресации и правила их группировки; (4) семантика инструкций, представленная в виде формул; (5) ресурсы микропроцессора, к которым обращаются инструкции. Кроме того там также описываются специальные условия такие, как исключения, ситуации в работе конвейера инструкций и подсистемы памяти. Эти описания используются генератором для построения тестовых программ.

Другой важный аспект конфигурации – это внешняя эталонная модель, используемая генератором для отслеживания состояния микропроцессора. Она интегрируется в ядро инструмента при помощи специальных библиотек на языке C++. Как правило, модель разрабатывают производители микропроцессора, а интеграцию осуществляют разработчики инструмента. Это требует скоординированных совместных усилий т.к. для успешной интеграции необходимо, чтобы модель удовлетворяла требованиям, предъявляемым инструментом.

Задачи тестирования описываются при помощи шаблонов, которые разрабатывается вручную или создаются при помощи графического интерфейса. В шаблонах задаются используемые инструкции, вероятности их появления, правила генерации входных значений, целевые тестовые ситуации т.д. Тестовые шаблоны разрабатываются в соответствии с таблицами тестового покрытия, которые формулируют цели тестирования. На основе метрик покрытия, полученных в результате выполнения построенных тестов, в набор шаблонов вносятся изменения до тех пор, пока требуемый уровень покрытия не будет достигнут.

Тестовые ситуации описываются в виде правил, основанных на ограничениях и различных эвристиках, которые хранятся в специальной базе знаний. Набор правил может пополняться. Это позволяет накаливать знания, используемые для построения тестов, и повторно применять их для тестирования других микропроцессоров.

Список тестовых ситуаций, покрытие которых может быть обеспечено при помощи инструмента RAVEN включает: (1) комбинации следующих друг за другом инструкций; (2) ситуации в работе операций с плавающей точкой; (3) исключения в работе инструкций; (4) конвейерные конфликты; (5) различные сценарии обработки запросов к иерархии памяти; (6) ситуации,

связанные с совместным использованием памяти несколькими ядрами многоядерного процессора.

К сожалению, информации об инструменте RAVEN, имеющаяся в открытом доступе, недостаточно подробна. Остается неясно, какой уровень нацеленности он позволяет достичь, насколько трудоемко писание тестовых ситуаций и предусматривается ли поддержка новых типов тестовых ситуаций. Также к недостаткам инструмента можно отнести то, что описание микропроцессора, используемое для генерации тестов, и эталонная модель являются дублирующимися представлениями одного и того же знания. Это усложняет поддержку особенно инструмента, если эти представления разрабатываются разными командами.

6. Инструменты IBM Research

В IBM Research разрабатывается несколько инструментов генерации тестовых программ, которые имеют различное назначение и используются в различных промышленных проектах. Изначально эти инструменты создавались для верификации микропроцессоров семейства PowerPC [35], а в дальнейшем применялись и для других архитектур (таких, как ARM и x86).

6.1 Инструмент Genesys-Pro

В настоящее время основным средством генерации тестовых программ, используемым в IBM, является Genesys-Pro [8]. Это универсальный инструмент, позволяющий создавать случайные и нацеленные тесты для различных типов микропроцессоров. Данный инструмент разделен на *ядро*, которое реализует методы генерации, применимые для любых архитектур, и *модель*, которая содержит всю информацию о тестируемом микропроцессоре. Задачи тестирования формулируются при помощи тестовых *шаблонов*. Сгенерированные инструкции выполняются на внешнем симуляторе с целью отслеживания состояния микропроцессора и контроля корректности построенной программы.

Инструмент Genesys-Pro включает в себя специальную среду моделирования, позволяющую создавать модели микропроцессоров на основе набора высокоуровневых блоков. Модели содержат информацию двух видов: (1) декларативное описание микропроцессора, которое включает в себя инструкции, ресурсы (регистры, память) и некоторые механизмы (например, трансляция адресов) и (2) тестовое знание для данного микропроцессора, которое представляет собой набор эвристик, позволяющих повысить уровень покрытия. Семантика инструкций описывается в виде ограничений на их входные аргументы. Описание инструкций также включает их сигнатуры, используемые ими ресурсы, типы данных аргументов и допустимые входные значения. Инструкции могут объединяться в группы. Среда моделирования имеет некоторые ограничения, не позволяющие описывать семантику

некоторых типов инструкций. В частности для инструкций арифметики с плавающей точкой и инструкций доступа к памяти используются дополнительные инструменты, о которых будет рассказано в следующих разделах. Кроме того, семантику некоторых сложных инструкций приходится описывать на языке C++.

Шаблоны создаются на специализированном языке [36], предоставляющем конструкции для описания последовательностей инструкций, распределений вероятностей и ограничений. Последовательности инструкций могут строиться при помощи методов случайной и комбинаторной генерации. Также язык предоставляет средства для описания последовательностей инструкций, которые будут выполняться в разных потоках (или разных ядрах). Входные аргументы инструкций генерируются случайным образом (с заданной вероятностью) или путем решения ограничений. Ограничения, задающие те или иные аспекты поведения инструкций, берутся из модели. Кроме этого существуют универсальные ограничения, которые можно разделить на следующие типы: (1) ограничения на выравнивание адресов; (2) зависимости по ресурсам для инструкций; (3) события, связанные с работой подсистемы памяти (промахи и попадания в различные буферы). Ограничения могут быть обязательными (hard) и необязательными (soft). Первые имеют более высокий вес при решении систем ограничений и, как правило, основаны на семантике инструкций. Вторые могут быть проигнорированы, если система ограничений не имеет решения, и обычно основаны на тестовом знании. Для ограничений можно задавать вероятности, с которыми они должны быть применены для выбранных инструкций. Также язык описания тестовых шаблонов поддерживает условную генерацию. Т.е. можно задавать пред- и постусловия, которые должны выполняться для того, чтобы фрагмент кода был добавлен в тестовую программу.

Генерация тестовых программ включает следующие стадии: (1) построение последовательностей инструкций; (2) решение ограничений (или генерация случайных значений) для каждой из инструкций; (3) симуляция инструкций на эталонной модели. В случае если не удается решить ограничения, инструмент может вносить коррективы в последовательность инструкций (добавлять дополнительные инструкции, повторно генерировать предыдущие инструкции).

Инструмент Genesys-Pro позволяет достичь достаточно высокого уровня нацеленности. Степень случайности тестов задается тестовыми шаблонами, которые разрабатываются в соответствии с планом верификации. Решение относительно полноты набора тестовых шаблонов принимается на основе метрик покрытия, основанных на реализации и функциональных требованиях.

К недостаткам Genesys-Pro можно отнести ограничения среды моделирования, которые требуют использования дополнительных средств для моделирования инструкций арифметики с плавающей точкой и доступа к памяти. Другим слабым местом является использование внешней эталонной модели для симуляции инструкций. Это требует дополнительных трудозатрат на

интеграцию и поддержку данной модели. Также непонятно насколько расширяемым является инструмент (возможность добавления новых методов генерации).

6.2 Инструмент FPGen

Для верификации модулей арифметики с плавающей точкой IBM Research был разработан инструмент FPGen [37]. Этот инструмент расширяет Genesys-Pro средствами генерации тестовых данных для покрытия всевозможных ситуаций в работе операций с плавающей точкой [38] (требования определены в стандарте IEEE 754 [39]).

Инструмент FPGen генерирует значения входных аргументов инструкций, разрешая специализированные ограничения. Ограничения определяют значения отдельных входных аргументов, отношения между значениями входных аргументов, результат промежуточных вычислений или конечный результат выполнения инструкции. Для описания ограничений используется язык XML. Генератор FPGen не привязан к какой-либо конкретной архитектуре, он генерирует наборы данных, которые сохраняются в специальном формате [40]. Эти данные используются Genesys-Pro при генерации тестовых программ, использующих инструкции арифметики с плавающей точкой. В процессе генерации Genesys-Pro комбинирует результаты решения ограничений, которые он разрешает самостоятельно и которые он разрешает при помощи FPGen.

6.3 Инструмент DeepTrans

Еще одно известное расширение Genesys-Pro – это инструмент DeepTrans [41], предназначенный для тестирования механизмов трансляции адресов. Этот инструмент использует спецификации подсистемы памяти, разработанные на специализированном языке моделирования. Данный язык позволяет представить процесс преобразования адреса в виде ориентированного ациклического графа, вершины которого соответствуют стадиям процесса, а дуги — переходам между стадиями. Множество путей от истока до стока задает конкретные варианты преобразования адреса и соответствует тестовым ситуациям. Тестовые ситуации представляются в виде ограничений, на которые можно ссылаться из тестовых шаблонов. За обработку шаблонов отвечает инструмент Genesys-Pro, который разрешает ограничения и трансформирует результаты в последовательности инструкций. К достоинствам DeepTrans следует отнести развитый язык моделирования механизмов трансляции. Он позволяет достаточно быстро осуществить настройку инструмента для тестирования механизмов трансляции адресов произвольного микропроцессора. Известный недостаток состоит в том, что не поддерживается автоматическое извлечение зависимостей между инструкциями (конфликтов использования устройств); их приходится дополнительно указывать в тестовых шаблонах.

7. Другие разработки

7.1 Среда RDG

В компании Samsung разработана среда RDG (Random Diagnostics Generator) [42], предназначенная для случайной генерации тестовых программ для реконфигурируемых микропроцессоров (Samsung Reconfigurable Processor, SRP). Эта среда использует в качестве входных данных описание синтаксиса инструкций тестируемого микропроцессора и тестовые шаблоны на языке C++. Тестовые шаблоны задают, какие инструкции будут использованы в тестовой программе, и описывают ограничения, накладываемые на входные значения этих инструкций. Среда RDG не владеет информацией о семантике инструкций и не осуществляет симуляцию тестовых программ с целью предсказания результатов. Такой подход был выбран из-за особенностей тестирования реконфигурируемых микропроцессоров (набор инструкций зависит от конфигурации, и их реализация может отличаться). Кроме того, это позволяет с минимальными усилиями добавлять поддержку новых инструкций и обеспечить высокую скорость генерации. Недостаток такого подхода в том, что для генерации нацеленных тестов требуется описывать ограничения вручную и отсутствует возможность отслеживать состояние микропроцессора. Стоит сказать, что инструмент RDG позволяет эффективно решать задачу тестирования реконфигурируемых микропроцессоров Samsung, но он не является универсальным. Поддержка новых типов микропроцессоров и методов генерации в него не заложена.

7.2 Генератор MA^{2}TG

Исследования в области генерации тестовых программ для функциональной верификации микропроцессоров проводились в Оборонном научно-техническом университете Китая (National University of Defense Technology) [43]. В рамках этих исследований был разработан прототип генератора тестовых программ MA^{2}TG. Данный генератор применим для различных типов микропроцессоров и поддерживает случайную генерацию и генерацию на основе ограничений. В качестве входных данных для генератора MA^{2}TG используются спецификации на языке EXPRESSION [44], описывающие архитектуру тестируемого микропроцессора, и шаблоны на специализированном языке, формулирующие задачи генерации. Инструмент транслирует входные в файлы в программу-генератор на языке C++, которая генерирует тесты. В процессе генерации симуляция построенных последовательностей инструкций не осуществляется.

Спецификации на языке EXPRESSION содержат информацию о структуре и поведенческих свойствах микропроцессора, а также о связи между ними. Для генерации тестовых программ в первую очередь необходима информация о

поддерживаемых инструкциях, которая включает текстовый и бинарный формат инструкций, списки их операндов и семантику инструкций (условия возникновения тестовых ситуаций). MA^{2}TG строит на основе спецификаций библиотеку шаблонов инструкций (Instruction Template Library, ITL), которая используется при генерации. Каждая инструкция описывается классом на языке C++, который содержит методы для ее печати, получения списка аргументов и доступа к ассоциированным с ней ограничениям. Использование формальных спецификаций позволяет относительно просто сконфигурировать инструмент для тестирования микропроцессора с новой архитектурой. Однако подход MA^{2}TG имеет некоторые недостатки. Информация о структуре микропроцессора, которая описывается в спецификациях на языке EXPRESSION, чаще всего не требуется для генерации тестовых программ. Эта информация может быть достаточно объемной, и ее специфицирование требует дополнительных трудозатрат. Кроме того, она может быть недоступна на ранних стадиях работы над проектом. А в случаях, когда требуется тестирование микропроцессоров, по-разному реализующих одну и ту же архитектуру, она может препятствовать повторному использованию спецификаций. Все это увеличивает трудоемкость разработки и поддержки спецификаций. Использование более простого языка, описывающего только поведенческие свойства, могло бы упростить эту задачу. Также, следует заметить, что MA^{2}TG не строит эталонный симулятор на основе спецификаций, хотя язык EXPRESSION предоставляет для этого достаточное количество информации. Эталонный симулятор был бы полезен для проверки корректности построенных программ, создания встроенных проверок и разрешения ограничений.

Задачи генерации описываются на специализированном языке. Такие описания называют ограничениями (constraints), хотя, по сути, они являются вариантом тестовых шаблонов. На их основе MA^{2}TG создает программы на языке C++, которые при помощи ITL и внешних библиотек разрешения ограничений, генерируют тесты. Язык описания тестовых шаблонов позволяет задавать следующие параметры: (1) используемые инструкции, их количество, порядок и вероятность появления; (2) ограничения на значения операндов инструкций; (3) зависимости по операндам между инструкциями. Данный язык ориентирован на случайную выборку инструкций и разрешения ограничений для выбранных инструкций. Это позволяет быстро создавать большое количество небольших тестов для верификации определенных инструкций. Однако он не позволяет описывать сценарии со сложной структурой переходов. Также не предоставляется возможностей для описания многопоточных сценариев. Т.к. инструмент не осуществляет симуляцию на эталонной модели, не поддерживается создание встроенных проверок. Кроме того, отсутствие информации о состоянии микропроцессора усложняет решение ограничений. Из описания инструмента неясно, какие типы ограничений он поддерживает и насколько сложно в него добавить поддержку новых типов.

Инструмент MA^{2}TG реализован в виде прототипа. Несмотря на то, что его подход имеет ряд преимуществ, его функциональные возможности ограничены. Остается непонятно насколько он соответствует требованиям, предъявляемым инструментам, используемым в промышленных проектах. Из очевидных недостатков можно перечислить излишне детальные спецификации, отсутствие контроля состояния микропроцессора в процессе генерации, ограниченные возможности языка тестовых шаблонов и ограниченный набор поддерживаемых методов генерации.

7.3 Исследования Университета Флориды и Калифорнийского университета в Ирвайне

В Университете Флориды (UFL) и Калифорнийском университет в Ирвайне (UCI) был разработан метод генерации тестовых программ, нацеленных на проверку корректности работы конвейера команд микропроцессора [15]. Данный метод использует спецификации на языке EXPRESSION [44], на основе которых строится модель, описывающая архитектуру микропроцессора в виде графа. Кроме этого разрабатывается модель ошибок, которая описывает типичные ошибки проектирования. На основе модели ошибок для модели микропроцессора строятся формулы, которые задают условия возникновения конкретных ошибок для данного микропроцессора. Для этих формул при помощи инструмента SMV (Symbolic Model Verifier) [45], который использует метод проверки моделей, стоятся тестовые примеры (контрпримеры для отрицания формул), на основе которых генерируются тестовые программы. По мнению авторов, метод не масштабируется на сложные микропроцессоры, поэтому предлагается дополнительно использовать тестовые шаблоны. Они создаются вручную и содержат описания цепочек инструкций, которые вызывают определенные ситуации в поведении микропроцессора (прежде всего, конвейерные конфликты). К достоинствам данного метода можно отнести то, что он применим для различных типов микропроцессоров и позволяет обеспечить покрытие ситуаций в работе конвейера команд, используя минимальное количество тестов. Главный недостаток метода - сложность разработки детальных спецификаций и модели ошибок. Следует также отметить, что данные исследования носили академический характер и на их основе не были разработаны инструменты, которые могли бы использоваться в промышленных проектах.

7.4 Инструмент μGP

Исследователями Туринского политехнического университета (Politecnico di Torino) был предложен интересный подход к генерации тестовых программ, основанный на использовании генетических алгоритмов [46][47]. Данный подход был реализован в прототипе инструмента μGP. Основная идея подхода состоит в следующем. Для генератора тестовых программ создается

библиотека инструкций, которая описывает синтаксис языка ассемблера целевого микропроцессора. Задачи генерации описываются в виде ациклического графа, который описывает поток выполнения тестовой программы. Каждая вершина графа содержит ссылку на описание инструкции в библиотеке инструкций и значения ее операндов. Библиотека инструкций и задачи генерации описываются на языке XML. Генерация тестовых программ осуществляется путем мутации структуры графа и значений операндов инструкций внутри отдельных вершин. Генератор пытается построить тестовую программу, которое обеспечит наилучшее покрытие для заданной метрики. Значение метрики получается путем выполнения построенных программ на симуляторе RTL модели.

Достоинством данного подхода является его гибкость и универсальность. Он позволяет достичь высокого уровня тестового покрытия для различных типов микропроцессоров, используя различные метрики покрытия. Главный недостаток – высокая вычислительная сложность, из-за которой скорость генерации получается достаточно низкой. Инструмент μGP реализует только один метод генерации и не предусматривает поддержку новых методов. Еще один возможный недостаток – это то, что входные форматы конфигурационных файлов не являются интуитивно понятными.

8. Анализ возможностей инструментов генерации тестовых программ

В предыдущих разделах были рассмотрены основные из существующих инструментов генерации тестовых программ для микропроцессоров. Они отличаются набором поддерживаемых методов генерации и способами реализации этих методов. В данном разделе выводятся их общие свойства, и формулируется концепция универсального инструмента генерации, сочетающего в себе данные свойства.

Для всех рассмотренных инструментов, независимо от поддерживаемых ими методов, можно выделить два свойства: (1) *реконфигурируемость* и (2) *расширяемость*. Оба этих свойства показывают, какое количество усилий требуется для адаптации инструмента к решению новых задач. Под реконфигурируемостью понимается возможность поддержки тестирования микропроцессоров с новой конфигурацией, а под расширяемостью – возможность интеграции компонентов, которые реализуют новые методы генерации.

Другим важным свойством инструмента генерации является возможность *контроля корректности* построенных программ. Корректные программы не должны содержать бесконечные циклы и приводить к переходу микропроцессора в состояния, в которых его поведение не определено. Как правило, подобный контроль осуществляется путем симуляции программ на эталонной модели в процессе их построения. Эталонные модели разрабатываются отдельно от инструмента генерации и интегрируются в него

при помощи специальных библиотек. По сути, они являются частью конфигурации инструмента т.к. используют знание о синтаксисе и семантике инструкций для их интерпретации.

Основным свойством, характеризующим методы генерации, является *нацеленность*. Методы, основанные на случайной генерации, не позволяют систематическим образом обеспечить покрытие функциональных требований. Современные микропроцессоры с конвейерной архитектурой и сложной организацией памяти имеют огромное пространство возможных состояний, которые с малой вероятностью будут достигнуты при использовании случайных тестов. Поэтому для проверки конкретных ситуаций или классов ситуаций требуются специализированные методы, основанные на разрешении ограничений и техниках проверки моделей.

Еще одним важным свойством современных инструментов верификации является возможность построения тестовых программ для многоядерных микропроцессоров. На базовом уровне она присутствует практически во всех инструментах, т.к. для этого достаточно включить в построенные программы инструкции, обеспечивающие разветвление потока выполнения на несколько ядер. Однако, методы нацеленной генерации, использующие информацию о текущем состоянии микропроцессора, должны учитывать тот факт, что каждое из ядер имеет свое состояние. Кроме того, инструменты должны обеспечивать покрытие ситуаций, связанных с параллельным выполнением программ.

В табл. 2 сравниваются функциональные возможности рассмотренных ранее инструментов по перечисленным критериям.

Табл. 2. Сравнение возможностей рассмотренных инструментов генерации

Table 2. Comparison of the considered generation tools

Инструмент	**Реконфигурируемость**	**Расширяемость**	**Контроль Корректности**	**Уровень Нацеленности**	**Поддержка Многоядерности**
INTEG	Нет	Нет	Нет	Средняя	Нет
RIS	Нет	Нет	Нет	Средняя	Да
RAVEN	Да	Нет	Да	Средняя	Да
Genesys-Pro, FPGen, DeepTrans	Да	Да (частично)	Да	Высокая	Да
RDG	Да	Нет	Нет	Низкая	Нет
MA^{2}TG	Да	Нет	Нет	Высокая	Нет
UFL/UCI	Да	Нет	Нет	Высокая	Нет
μGP	Да	Нет	Нет	Средняя	Нет

Как можно заметить, большинство инструментов предназначено для решения конкретных задач и не предполагают расширяемость. Это объясняется тем, что коммерческие компании занимаются верификацией выпускаемых ими микропроцессоров и создают инструменты для решения текущих задач верификации, а исследовательские институты изобретают новые методы верификации и создают инструменты для их апробации. При этом задача интеграции методов, которые не используются в данный момент, не ставится.

Ситуация с реконфигурируемостью несколько сложнее. Несмотря на то, что некоторые инструменты позволяют конфигурирование под различные архитектуры, интеграция внешней эталонной модели сопряжена с трудностями. Большинство из перечисленных реконфигурируемых инструментов не используют эталонные модели и не осуществляют контроль корректности построенных инструкций. Однако даже без необходимости интеграции эталонной модели создание пользовательских конфигураций может иметь значительную трудоемкость.

Уровень нацеленности и поддержка многоядерности характеризуют методы генерации и зависят от области применения инструмента. При этом они в той или иной степени связаны с расширяемостью и реконфигурируемость. Более гибкие инструменты реализуют большее количество методов генерации и предоставляют больше возможностей.

Т.к. применимость отдельных инструментов ограничена, актуальной задачей является создание *универсального инструмента*, который позволил бы объединить различные методы генерации, и был бы применим для различных типов микропроцессоров. Подобный инструмент должен сочетать все перечисленные выше свойства. Его характеристики могут быть сформулированы следующим образом.

Реконфигурируемость предполагает описание конфигурации тестируемого микропроцессора в простом, удобном и понятном инженеру-верификатору формате. Расширяемость предполагает, что описания, созданные для одних методов генерации, могут быть использованы другими методами. Т.к. методы генерации требуют разное количество информации, то для более сложных методов будут требоваться описания, которые дополнят информацию, представленную в описаниях, используемых более простыми методами. Информацию о конфигурации микропроцессора, используемую рассмотренными инструментами, можно разделить на четыре уровня: (1) формат инструкций (текстовый и бинарный), (2) семантика инструкций, (3) организация подсистемы памяти и (4) организация конвейера инструкций. Один из возможных способов представления этой информации – применение формальных спецификаций. Это позволит использовать единое описание для извлечения ограничений, построения формальных моделей и эталонных симуляторов. Для решения данной задачи требуется формальный язык, на котором можно в зависимости от потребности описывать различные аспекты

конфигурации микропроцессора. Другой вариант решения – использование нескольких языков, расширяющих возможности друг друга.

9. Заключение

В статье сделан обзор распространенных подходов к генерации тестовых программ для функциональной верификации микропроцессоров. Рассмотренные подходы различаются областью применения и сложностью реализации. Наиболее простые основаны на случайной генерации, наиболее сложные используют техники проверки моделей для построения тестов, нацеленных на покрытие определенных ситуаций в работе микропроцессора. Инструментам, использующим различные методы генерации, требуется разное количество информации о тестируемом микропроцессоре. Эта информация или является частью инструмента или поставляется в виде конфигурационных файлов. Кроме этого некоторые инструменты используют внешние эталонные модели для контроля корректности построенных программ и предсказания состояния микропроцессора. Эти модели требуется интегрировать в инструмент.

Основная проблема современных инструментов генерации тестовых программ для микропроцессоров – это то, что они, как правило, созданы для конкретных типов микропроцессоров и предназначены для решения конкретных задач верификации. Возможность поддержки новых типов микропроцессоров и новых методов генерации или не предусмотрена, или ограничена ввиду трудоемкости конфигурирования инструмента и трудностей интеграции эталонной модели. Т.к. ни один метод или инструмент не является универсальным решением для всех задач верификации, интеграция методов генерации является актуальной задачей. При этом важно чтобы имеющийся инструментарий методов генерации был применим к различным типам микропроцессоров.

К сожалению, в настоящее время не существует универсального инструмента, который позволил бы применять широкий набор методов генерации для тестирования различных типов микропроцессоров. Создание такого инструмента имело бы большое практическое значение. Это позволило бы снизить затраты на поддержку инструментов и повысить качество тестирования.

Список литературы

[1]. Grant McFarland. Microprocessor Design: A Practical Guide from Design Planning to Manufacturing Professional Engineering. McGraw Hill Professional, 2006, 408 p.

[2]. Статистика числа транзисторов в микропроцессорах — http://en.wikipedia.org/wiki/Transistor_count

[3]. Intel® Pentium® 4 Processor. Specification Update, August 2008 (http://download.intel.com/design/intarch/specupdt/24919969.pdf)

[4]. Intel® Core™ i7-900 Desktop Processor Extreme Edition Series and Intel® Core™ i7-900 Desktop Processor Series. Specification Update, February 2015. (http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/core-i7-900-ee-and-desktop-processor-series-spec-update.pdf)
[5]. Beizer B. The Pentium Bug – An Industry Watershed // Testing Techniques Newsletter, TTN Online Edition, September 1995.
[6]. А.С. Камкин, А.М. Коцыняк, С.А. Смолов, А.А. Сортов, А.Д. Татарников, М.М. Чупилко. Средства функциональной верификации микропроцессоров, Труды ИСП РАН, том 26, выпуск 1, 2014, с. 149-200. DOI: 10.15514/ISPRAS-2014-26(1)-5.
[7]. W.K. Lam. Hardware Design Verification: Simulation and Formal Method-Based Approaches. Prentice Hall, 2005. 624 p.
[8]. A. Adir, E. Almog, L. Fournier, E. Marcus, M. Rimon, M. Vinov, A. Ziv. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification. IEEE Design & Test of Computers, 21(2), 2004, pp. 84-93.
[9]. J. Bhadra, M. Abadir, S. Ray, L. Wang. A Survey of Hybrid Techniques for Functional Verification. IEEE Design & Test of Computers, 24(22), 2007, pp. 112-122.
[10]. Камкин А.С., Сергеева Т.И., Смолов С.А., Татарников А.Д., Чупилко М.М. Расширяемая среда генерации тестовых программ для микропроцессоров. Программирование, № 1, 2014, стр. 3-14.
[11]. A.Kamkin, A.Protsenko, A.Tatarnikov. An Approach to Test Program Generation Based on Formal Specifications of Caching and Address Translation Mechanisms. Trudy ISP RAN / Proc. ISP RAS, vol. 27, issue 3, 2015, pp. 125-138. DOI: 10.15514/ISPRAS-2015-27(3)-9.
[12]. Бобков С.Г., Чибисов П.А. Повышение качества тестирования высокопроизводительных микропроцессоров методами встречного тестирования с анализом функционального тестового покрытия выделенных приложений. Информационные технологии, No 8, 2013, с. 26-33.
[13]. Хисамбеев И.Ш., Чибисов П.А. Об одном методе построения метрик функционального покрытия в тестировании микропроцессоров. Проблемы разработки перспективных микро- и наноэлектронных систем - 2014. Сборник трудов под общ. ред. академика РАН А.Л. Стемпковского. М.: ИППМ РАН, 2014, часть II, стр. 63-68.
[14]. Piziali A. Functional Verification Coverage Measurement and Analysis. New York: Kluwer Academic Publishers. 2004, 216 p.
[15]. P. Mishra, N. Dutt. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design Automation of Electronic Systems, 13(3), 2008, pp. 1-36.
[16]. E.A. Poe. Introduction to random test generation for processor verification. Technical report. Obsidian Software, 2002, 7 p.
[17]. Генератор тестовых программ RISU для тестирования симулятора QEMU - https://git.linaro.org/people/peter.maydell/risu.git/about/.
[18]. Грибков И.В., Захаров А.В., Кольцов П.П. и др. Стохастическое тестирование в системе INTEG. Программные продукты и системы. 2007. № 2. с. 22-26.
[19]. Камкин А.С. Генерация тестовых программ для микропроцессоров. Труды ИСП РАН, том 14, часть 2, 2008, с. 23–63.
[20]. N. Sharma, B. Dickman, Verifying an ARM Core, EE Times, 2001, 7 p.

[21]. А.С. Камкин. Некоторые вопросы автоматизации построения тестовых программ для модулей обработки переходов микропроцессоров. Труды ИСП РАН, 18, 2010, стр. 129-149.
[22]. Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. Marcus, G. Shurek. Constraint-Based Random Stimuli Generation for Hardware Verification. AI Magazine, 28(3), 2007, pp. 13-30.
[23]. P. Mishra and N. Dutt. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 13, Issue 3, 2008, pp. 1–36.
[24]. Грибков И.В., Захаров А.В., Кольцов П.П. и др. Развитие системы стохастического тестирования микропроцессоров INTEG. Программные продукты и системы. 2010, № 2, стр. 14–23.
[25]. MIPS64™ Architecture For Programmers. Volume 1: Introduction to the MIPS64™ Architecture. Revision 6.01. MIPS Technologies Inc. 2014. 148 p.
[26]. Программный симулятор VMIPS – http://www.dgate.org/vmips/.
[27]. Сайт компании ARM — http://www.arm.com.
[28]. N. Sharma and B. Dickman. Verifying an ARM Core. EE Times. 2001, p. 7.
[29]. Hrishikesh M.S., Rajagopalan M., Sriram S., Mantri R. System Validation at ARM — Enabling our Partners to Build Better Systems. White Paper. April 2016 (http://www.arm.com/files/pdf/System_Validation_at_ARM_Enabling_our_partners_to_build_better_systems.pdf).
[30]. Venkatesan D., Nagarajan P. A Case Study of Multiprocessor Bugs Found Using RIS Generators and Memory Usage Techniques. Workshop on Microprocessor Test and Verification, 2014, pp. 4-9. DOI: 10.1109/MTV.2014.28.
[31]. Hudson J., Kurucheti G. A Configurable Random Instruction Sequence (RIS) Tool for Memory Coherence in Multi-processor Systems. Workshop on Microprocessor Test and Verification, 2014, pp. 98-101. DOI: 10.1109/MTV.2014.26.
[32]. Генератор тестовых программ RAVEN - http://www.slideshare.net/DVClub/introducing-obsidian-software-andravengcs-for-powerpc.
[33]. Obsidian Software Inc. “Raven: Product datasheet”. 6 p.
[34]. R.N. Mahapatra, P. Bhojwani, J. Lee, and Y. Kim. Microprocessor Evaluations for Safety-Critical, Real-Time Applications: Authority for Expenditure, No.43, Phase 3 Report, 2009, 43 p.
[35]. Архитектура PowerPC - https://en.wikipedia.org/wiki/PowerPC
[36]. M. Behm, J. Ludden, Y. Lichtenstein, M. Rimon, M. Vinov. Industrial Experience with Test Generation Languages for Processor Verification. Proceedings of the Design Automation Conference, 2004, pp. 36-40.
[37]. M. Aharoni, S. Asaf, L. Fournier, A. Koifman and R. Nagel. FPgen – A Test Generation Framework for Datapath Floating-Point Verification. Proceedings of the Eighth IEEE International Workshop on High-Level Design Validation and Test Workshop (HLDVT'03), 2003, pp. 17–22.
[38]. M. Aharony, E. Gofman, E. Guralnik, A. Koyfman. Injecting floating-point testing knowledge into test generators. Proceedings of the 7th international Haifa Verification conference on Hardware and Software: Verification and Testing (HVC’11), 2011, pp. 234-241, ISBN 978-3-642-34187-8.
[39]. IEEE standard for binary FP arithmetic. An American National Standard, ANSI/IEEEE Std. 754-2008, 58 p.

[40]. IBM Floating-Point Test Suite for IEEE 754R Standard - https://www.research.ibm.com/haifa/projects/verification/fpgen/ieeets.html
[41]. Adir A., Fournier L., Katz Y., Koyfman A. DeepTrans – Extending the Model-based Approach to Functional Verification of Address Translation Mechanisms. High-Level Design Validation and Test Workshop, 2006, pp. 102-110.
[42]. Seonghun Jeong, Youngchul Cho, Daeyong Shin, Changyeon Jo, Yenjo Han, Soojung Ryu, Jeongwook Kim, and Bernhard Egger. Random Test Program Generation for Reconfigurable Architectures. Proceedings of 13th International Workshop on Microprocessor Test and Verification (MTV), 2012, 6 p.
[43]. T.Li, D.Zhu, Y.Guo, G.Liu, S.Li. MA2TG: A Functional Test Program Generator for Microprocessor Verification. Euromicro Conference on Digital System Design, 2005, pp.176-183.
[44]. P. Grun, A. Halambi, A. Khare, V. Ganesh, N. Dutt and A. Nicolau. EXPRESSION: An ADL for System Level Design Exploration. Technical Report 1998-29, University of California, Irvine, 1998, 26 p.
[45]. Инструмент - http://www.cs.cmu.edu/~modelcheck/smv.html
[46]. F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero. Efficient Machine-Code TestProgram Induction. CEC'2002: Congress on Evolutionary Computation, Honolulu, Hawaii, USA, 2002.
[47]. F. Corno et al., "Fully Automatic Test Program Generation for Microprocessor Cores," Proc. IEEE Design, Automation and Test in Europe (DATE 03), IEEE CS Press, 2003, pp. 1006-1011.
[48]. F. Corno, E. Sanchez, M. Sonza Reorda, G. Squillero. Automatic Test Program Generation – A Case Study. IEEE Design and Test, Special Issue on Functional Verification and Testbench Generation, Volume 21, Issue 2, 2004, pp. 102-109.

A Survey of Methods and Tools for Test Program Generation for Microprocessors

A.D. Tatarnikov <andrewt@ispras.ru>
Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia

Abstract. This paper gives a survey of existing methods and tools for test program generation for microprocessors. Test program generation and analysis of their execution traces is the main approach to functional verification of microprocessors. This approach is also known as testing. Despite continuous progress in test program generation methods, testing remains an extremely laborious process. One of the main reasons is that test program generation tools are unable to quickly enough adapt to changes. In the majority of cases, they are created for specific microprocessor types and are designed to solve specific tasks. For this reason, support for new microprocessors types and generation methods requires a significant effort. Often, in such situations, tools have to be implemented from scratch. Inability to reuse existing implementations of generation methods complicates evolution of test generation tools and, consequently, prevents improvement of testing quality. The present situation creates motivation to search for solutions to developing more flexible tools which could be easily adapted to testing new microprocessor types and applying new generation methods. The goal of the present work is to summarize existing experience in test program generation,

which could serve as a basis for creating such tools. The paper considers strengths and weaknesses of popular generation methods, their application domains and cases of their combined use. It also makes a comparative analysis of facilities of existing generation tools implementing these methods. Based on the analysis, it gives recommendations on creating a unified methodology to develop tools for test program generation for microprocessors.

Keywords: microprocessors; hardware; functional verification; testing; test program generation.

DOI: 10.15514/ISPRAS-2017-29(1)-11

For citation: Tatarnikov A.D. A survey of methods and tools of test program generation for microprocessors. *Trudy ISP RAN / Proc. ISP RAS*, vol. 29, issue 1, 2017. pp. 167-194 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-11

References

[1]. Grant McFarland. Microprocessor Design: A Practical Guide from Design Planning to Manufacturing Professional Engineering. McGraw Hill Professional, 2006, 408 p.

[2]. Transistor count in microprocessors — http://en.wikipedia.org/wiki/Transistor_count

[3]. Intel® Pentium® 4 Processor. Specification Update, August 2008 (http://download.intel.com/design/intarch/specupdt/24919969.pdf)

[4]. Intel® Core™ i7-900 Desktop Processor Extreme Edition Series and Intel® Core™ i7-900 Desktop Processor Series. Specification Update, February 2015. (http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/core-i7-900-ee-and-desktop-processor-series-spec-update.pdf)

[5]. Beizer B. The Pentium Bug – An Industry Watershed // Testing Techniques Newsletter, TTN Online Edition, September 1995.

[6]. A. Kamkin, A. Kotsynyak, S. Smolov, A. Sortov, A. Tatarnikov, M. Chupilko. Tools for Functional Verification of Microprocessors. Trudy ISP RAN / Proc. ISP RAS, vol. 26, issue. 1, 2014. pp. 149-200 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-5.

[7]. W.K. Lam. Hardware Design Verification: Simulation and Formal Method-Based Approaches. Prentice Hall, 2005, 624 p.

[8]. A. Adir, E. Almog, L. Fournier, E. Marcus, M. Rimon, M. Vinov, A. Ziv. Genesys-Pro: Innovations in Test Program Generation for Functional Processor Verification. IEEE Design & Test of Computers, 21(2), 2004, pp. 84-93.

[9]. J. Bhadra, M. Abadir, S. Ray, L. Wang. A Survey of Hybrid Techniques for Functional Verification. IEEE Design & Test of Computers, 24(22), 2007, pp. 112-122.

[10]. A.S. Kamkin, T.I. Sergeeva, S.A. Smolov, A.D. Tatarnikov, M.M. Chupilko. Extensible Environment for Test Program Generation for Microprocessors. Programming and Computer Software, 40(1), 2014, pp. 1-9. DOI: 10.1134/S0361768814010046.

[11]. A.Kamkin, A.Protsenko, A.Tatarnikov. An Approach to Test Program Generation Based on Formal Specifications of Caching and Address Translation Mechanisms. Trudy ISP RAN / Proc. ISP RAS, vol. 27, issue 3, 2015, pp. 125-138. DOI: 10.15514/ISPRAS-2015-27(3)-9.

[12]. Bobkov S.G., Chibisov P.A. Improving of Quality of High-performance Microprocessors Testing by Counter-Testing Methods with Analysis of the Functional Coverage of the Selected User Control Tasks. Informacionnye tehnologii [Information Technology], No. 8, 2013, pp. 26-33 (in Russian)

[13]. Khisambeev I.Sh., Chibisov P.A. On one method of defining functional coverage metrics for microprocessor testing. Proceedings of the conference on Problems of Perspective Micro- and Nanoelectronic Systems Development (MES-2014), Part II, 2014, pp. 63-68 (in Russian).
[14]. Piziali A. Functional Verification Coverage Measurement and Analysis. New York: Kluwer Academic Publishers. 2004. 216 p.
[15]. P. Mishra, N. Dutt. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design Automation of Electronic Systems, 13(3), 2008, pp. 1-36.
[16]. E.A. Poe. Introduction to random test generation for processor verification. Technical report. Obsidian Software, 2002, 7 p.
[17]. RISU test program generator for testing QEMU simulator - https://git.linaro.org/people/peter.maydell/risu.git/about/.
[18]. Gribkov I.V., Zaharov A.V., Kol'cov P.P., et al. Stochastic testing in the INTEG system. Programmnye Produkty i Sistemy [Programming Products and Systems], 2007, no. 2, pp. 22–26 (in Russian).
[19]. Kamkin A.S. Test Program Generation for Microprocessors. Trudy ISP RAN / Proc. ISP RAS, vol. 14, issue. 2, 200, pp. 23-63 (in Russian).
[20]. N. Sharma, B. Dickman, Verifying an ARM Core, EE Times, 2001, 7 p.
[21]. A. Kamkin. Some issues of automation of test program generation for branch processing units of microprocessors. Trudy ISP RAN / Proc. ISP RAS, vol. 18, 2010, pp. 129-149 (in Russian).
[22]. Y. Naveh, M. Rimon, I. Jaeger, Y. Katz, M. Vinov, E. Marcus, G. Shurek. Constraint-Based Random Stimuli Generation for Hardware Verification. AI Magazine, 28(3), 2007, pp. 13-30.
[23]. P. Mishra and N. Dutt. Specification-Driven Directed Test Generation for Validation of Pipelined Processors. ACM Transactions on Design Automation of Electronic Systems (TODAES), Volume 13, Issue 3, 2008, pp. 1–36.
[24]. Gribkov I.V., Zaharov A.V., Kol'cov P.P., et al. Evolution of stochastic testing system of microprocessors INTEG. Programmnye Produkty i Sistemy [Programming Products and Systems], 2010, no. 2, pp. 14–23 (in Russian).
[25]. MIPS64™ Architecture For Programmers. Volume 1: Introduction to the MIPS64™ Architecture. Revision 6.01. MIPS Technologies Inc. 2014, 148 p.
[26]. VMIPS virtual machine simulator – http://www.dgate.org/vmips/.
[27]. ARM web site — http://www.arm.com.
[28]. N. Sharma and B. Dickman. Verifying an ARM Core. EE Times. 2001, p. 7.
[29]. Hrishikesh M.S., Rajagopalan M., Sriram S., Mantri R. System Validation at ARM — Enabling our Partners to Build Better Systems. White Paper. April 2016 (http://www.arm.com/files/pdf/System_Validation_at_ARM_Enabling_our_partners_to_build_better_systems.pdf).
[30]. Venkatesan D., Nagarajan P. A Case Study of Multiprocessor Bugs Found Using RIS Generators and Memory Usage Techniques. Workshop on Microprocessor Test and Verification, 2014, pp. 4-9. DOI: 10.1109/MTV.2014.28.
[31]. Hudson J., Kurucheti G. A Configurable Random Instruction Sequence (RIS) Tool for Memory Coherence in Multi-processor Systems. Workshop on Microprocessor Test and Verification, 2014, pp. 98-101. DOI: 10.1109/MTV.2014.26.
[32]. RAVEN test program generatort -http://www.slideshare.net/DVClub/introducing-obsidian-software-andravengcs-for-powerpc.

[33]. Obsidian Software Inc. “Raven: Product datasheet”. 6 p.
[34]. R.N. Mahapatra, P. Bhojwani, J. Lee, and Y. Kim. Microprocessor Evaluations for Safety-Critical. Real-Time Applications: Authority for Expenditure, no.43, Phase 3 Report, 2009, 43 p.
[35]. PowerPC architecture - https://en.wikipedia.org/wiki/PowerPC
[36]. M. Behm, J. Ludden, Y. Lichtenstein, M. Rimon, M. Vinov. Industrial Experience with Test Generation Languages for Processor Verification. Proceedings of the Design Automation Conference, 2004, pp. 36-40.
[37]. M. Aharoni, S. Asaf, L. Fournier, A. Koifman and R. Nagel. FPgen – A Test Generation Framework for Datapath Floating-Point Verification. Proceedings of the Eighth IEEE International Workshop on High-Level Design Validation and Test Workshop (HLDVT'03), 2003, pp. 17–22.
[38]. M. Aharony, E. Gofman, E. Guralnik, A. Koyfman. Injecting floating-point testing knowledge into test generators. Proceedings of the 7th international Haifa Verification conference on Hardware and Software: Verification and Testing (HVC’11), 2011, pp. 234-241.
[39]. IEEE standard for binary FP arithmetic. An American National Standard, ANSI/IEEEE Std. 754-2008, 58 p.
[40]. IBM Floating-Point Test Suite for IEEE 754R Standard - https://www.research.ibm.com/haifa/projects/verification/fpgen/ieeets.html
[41]. Adir A., Fournier L., Katz Y., Koyfman A. DeepTrans – Extending the Model-based Approach to Functional Verification of Address Translation Mechanisms. High-Level Design Validation and Test Workshop, 2006, pp. 102-110.
[42]. Seonghun Jeong, Youngchul Cho, Daeyong Shin, Changyeon Jo, Yenjo Han, Soojung Ryu, Jeongwook Kim, and Bernhard Egger. Random Test Program Generation for Reconfigurable Architectures. Proceedings of 13th International Workshop on Microprocessor Test and Verification (MTV), 2012, 6 p.
[43]. T.Li, D.Zhu, Y.Guo, G.Liu, S.Li. MA2TG: A Functional Test Program Generator for Microprocessor Verification. Euromicro Conference on Digital System Design, 2005, pp.176-183.
[44]. P. Grun, A. Halambi, A. Khare, V. Ganesh, N. Dutt and A. Nicolau. EXPRESSION: An ADL for System Level Design Exploration. Technical Report 1998-29, University of California, Irvine, 1998, 26 p.
[45]. SMV model checker - http://www.cs.cmu.edu/~modelcheck/smv.html
[46]. F. Corno, G. Cumani, M. Sonza Reorda, G. Squillero. Efficient Machine-Code TestProgram Induction. CEC’2002: Congress on Evolutionary Computation, Honolulu, Hawaii, USA, 2002.
[47]. F. Corno et al., “Fully Automatic Test Program Generation for Microprocessor Cores,” Proc. IEEE Design, Automation and Test in Europe (DATE 03), IEEE CS Press, 2003, pp. 1006-1011.
[48]. F. Corno, E. Sanchez, M. Sonza Reorda, G. Squillero. Automatic Test Program Generation – A Case Study. IEEE Design and Test, Special Issue on Functional Verification and Testbench Generation, Volume 21, Issue 2, 2004, pp. 102-109.

Обзор подходов к моделированию памяти в инструментах статической верификации[1]

М.У. Мандрыкин <mandrykin@ispras.ru>
В.С. Мутилин <mutilin@ispras.ru>
ИСП РАН, 109004, Россия, г. Москва, ул. А. Солженицына, дом 25

Аннотация. В статье приведен обзор существующих подходов к моделированию памяти Си-программ в инструментах статической верификации. Обозначены основные проблемы, возникающие при разработке моделей памяти для языка Си. В обзоре рассматриваются две основные группы моделей памяти в зависимости от полноты поддержки областей памяти наперед не ограниченного размера. Среди моделей для ограниченных областей памяти рассматриваются модель, использующая результаты предварительного анализа алиасов, и модель на основе слабейших предусловий, использующая теорию неинтерпретируемых функций и логику первого порядка. Среди моделей для областей памяти наперед не ограниченного размера рассматривается типизированная модель, модель Бурсталла-Борната, модель с регионами и полная модель памяти для теории интерпретируемых множеств элементов списков, использованная ранее в инструменте дедуктивной верификации HAVOC.

Ключевые слова: статическая верификация; модели памяти; SMT-решатели.

DOI: 10.15514/ISPRAS-2017-29(1)-12

Для цитирования: Мандрыкин М.У., Мутилин В.С.. Обзор подходов к моделированию памяти в инструментах статической верификации. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 195-230. DOI: 10.15514/ISPRAS-2017-29(1)-12

1. Введение

Язык программирования Си продолжает оставаться одним из наиболее широко используемых языков программирования в области системного

[1] Исследования проводились при финансовой поддержке РФФИ в рамках проекта №15-01-03934.

программирования, в частности при написании операционных систем, драйверов, сред окружения и средств поддержки времени выполнения для различных языков программирования, а также при написании других систем, предъявляющих высокие требования к производительности или объему исполняемых программ. При этом одним из основных преимуществ использования языка Си для реализации высокопроизводительных систем является слабая статическая типизация и присутствие широкого набора доступных операций с указателями, позволяющими эффективно управлять использованием памяти, в том числе явно манипулируя адресами размещаемых в ней данных. Ко многим высокопроизводительным системам предъявляются среди прочих требования по высокой надежности, а в некоторых случаях – и по безопасности. Поэтому продолжает оставаться актуальной задача верификации Си-программ.

Эта задача может решаться с применением методов динамической, статико-динамической и статической верификации. В силу того, что каждый из подходов обладает своими преимуществами и ограничениями необходимо вести развитие всех этих направлений. Вместе с тем, только статическая верификация может дать доказательство отсутствия ошибок, по крайней мере, некоторых классов ошибок, или дать доказательство полного соответствия программ заданным формальным спецификациям.

Методы статической верификации ранее показали свою применимость, в частности, для верификации модулей и отдельных подсистем в гипервизорах и ядрах операционных систем, что в свою очередь является важным аргументом, подтверждающим актуальность развития этих методов.

Во многих современных инструментах статической верификации используются SMT-решатели, и, таким образом, семантика языка программирования, на котором написана верифицируемая программа, частично или полностью моделируется с помощью логических формул в соответствующих теориях (SMT-формул, от англ. Satisfiability Modulo Theories) [1]. Подходы к соответствующему эффективному моделированию семантики в виде SMT-формул могут очень существенно различаться друг от друга. При этом даже небольшие изменения в одном методе моделирования семантики могут приводить к изменению, к примеру, времени работы SMT-решателей на результирующих формулах (и, как следствие, времени работы всего инструмента верификации) в несколько десятков раз [2]. В таком контексте разработка соответствующих методов эффективного моделирования семантики используемых языков программирования и спецификации с помощью SMT-формул становится важной и актуальной задачей.

Для языка Си основной проблемой при моделировании семантики в виде логических формул является моделирование семантики операций с указателями, в частности, указателями на динамически выделяемые области

памяти наперед не ограниченного размера, а также моделирование различных приведений типов указателей и случаев использования объединений. Поэтому задача разработки соответствующих методов эффективного и точного моделирования памяти Си-программ для используемых на практике инструментов статической верификации является одной из ключевых.

Помимо развития собственно методов моделирования памяти Си-программ важной задачей является также оценка эффективности этих методов при использовании для верификации реальных промышленных программных систем.

2. Использование решателей логических формул в теориях в инструментах статической верификации

Решатели логических формул в теориях широко используются во всех основных группах инструментов статической верификации:

- проверки моделей программ (software model checking),
- анализа программ,
- дедуктивной верификации.

Среди разновидностей инструментов проверки моделей программ наиболее распространенными являются инструменты, основанные на методах итеративного построения абстракции для программы с помощью уточнения по неосуществимым в исходной программе (фиктивным) контрпримерам (CEGAR, от англ. Counter-Example Guided Abstraction Refinement) [3] и методах ограничиваемой верификации (BMC, англ. Bounded model checking) [4]. Данные инструменты используют SMT-решатели для:

- поиска контрпримеров во всем пространстве состояний модели, представленном в виде логической формулы [4];
- проверки осуществимости трасс выполнения, ведущих к ошибочным состояниям [5];
- проверки k-индуктивности заданного свойства программы [6];
- автоматического построения предикатных абстракций [7, 5];
- автоматического уточнения предикатных абстракций по неосуществимым контрпримерам с помощью построения интерполянтов Крейга [8];
- автоматического поиска индуктивных инвариантов с помощью процедуры индуктивного обобщения в рамках метода проверки моделей программ IC3 [9];

а также других, в том числе вспомогательных, целей.

В инструментах анализа программ (англ. program analysis), которые отличаются от проверки моделей модульностью верификации и применением менее точных и менее ресурсоемких способов абстракции, SMT-решатели используются для [10, 11]:

- абстрактной интерпретации с использованием различных абстрактных доменов, в том числе в предикатной абстракции;
- символьного выполнения с целью поиска входных данных, при выполнении на которых программа достигает заданный целевой оператор;
- дополнительной проверки осуществимости путей выполнения для уменьшения числа ложных предупреждений.

В основе методов дедуктивной верификации лежит логика Хоара [12], методы индуктивных утверждений и оценочных функций Флойда [13], преобразователи предикатов, такие как слабейшее предусловие [14], а также аксиоматическая семантика. С их помощью корректность программы относительно проверяемых свойств сводится к проверке набора формальных математических утверждений — условий верификации (УВ, англ. verification condition, VC), представляющих собой формулы в различных формальных логических системах, таких как пропозициональная или сепарационная [15] логика. В инструментах дедуктивной верификации решатели используются для [16, 17]:

- разрешения (проверки выполнимости) условий верификации;
- поиска контрпримеров для реализаций или спецификаций, не соответствующих друг другу;
- проверки используемых аксиоматических формализаций на непротиворечивость;
- выделения существенно используемых утверждений в контексте условия верификации, например, для поиска причин возникающих противоречий или для упрощения полученного условия верификации с целью повторной проверки с помощью другого решателя.

Кроме того, ведутся исследования в направлении интеграции SMT-решателей в инструменты интерактивного доказательства теорем [18, 19], используемых в подходах к верификации программ, основанных на использовании денотационной семантики.

2.1 Общая схема использования SMT-решателей

Несмотря на существенные различия между разнообразными задачами, для решения которых применяются решатели логических формул в теориях, можно выделить некоторый общий подход, который описывает все

перечисленные случаи их использования в инструментах статической верификации. Этот подход можно условно изобразить в виде схемы, приведенной на рис. 1.

Для генерации запросов к решателю используется некоторое внутреннее представление программы или ее фрагмента, использующее сущности соответствующего входного или промежуточного языка программирования (и, возможно, спецификации), такие как «переменная типа `int`», «указатель на структуру **`struct`** `mutex`», «оператор разыменования `*`», «конструктор `None`» алгебраического типа `Option` (из языка OCaml) и т.д. Так как современные решатели применяются не только для проверок выполнимости логических формул, но также, например, для поиска моделей выполнимых формул, поиска интерполянтов Крейга для невыполнимых формул, упрощения формул и других целей, на процесс генерации запросов к решателю влияют типы решаемых задач. Суть же генерации запроса к решателю, как правило, состоит в преобразовании внутреннего представления программы, использующего сущности языка программирования, в некоторое представление логической формулы в теориях, использующее сущности, определяемые этими теориями, например, «неинтерпретируемая целочисленная константа», «неинтерпретируемая функция», «логический массив», «битовый вектор» и т.д.

Таким образом, имеет смысл ввести условное (неформальное) понятие *модельной семантики языка программирования*, обозначающее способ представления семантики сущностей языка программирования и операций над ними в виде логических формул в теориях. Это понятие с одной стороны отличается от общего понятия *семантики языка программирования*, которое никак не ограничивает используемый способ описания семантики, позволяя выражать её, например, с помощью произвольных определений отношения вычисления [20], произвольных наборов логических утверждений или произвольных денотаций [21]. С другой стороны, оно не ограничивает возможностей использования как произвольного исходного языка программирования, так и произвольного целевого набора логических теорий, что не позволяет определить это понятие достаточно формально. В дальнейшем будем подразумевать, что для используемых в модельной семантике логических теорий существуют автоматические решающие процедуры, возможно, не обладающие полнотой (то есть допускающие незавершение решателя или вердикт «`unknown`» – «неизвестно»). Также для уменьшения возможных неоднозначностей будем в дальнейшем называть исходно заданную в произвольном виде семантику рассматриваемого языка программирования его *исходной семантикой*. Процесс преобразования внутреннего представления программы в запрос к решателю, содержащий логическую формулу в теориях, в соответствии с используемой модельной

семантикой и типом решаемой задачи будем называть *кодированием* (англ. *encoding*) запроса.

Очевидно, используемая модельная семантика существенно зависит от используемых языков программирования и спецификации. В частности, для императивных языков программирования (то есть основанных на операторах, изменяющих состояние программы), допускающих использование синонимичных (англ. aliasing) указателей или ссылок, например, таких как Си, Си++, Java или C#, модельная семантика должна определять способ кодирования операций с указателями или ссылками. Соответствующую часть модельной семантики, содержащую формальные определения понятий указателя или ссылки и формализацию операций над ними с использованием логических формул в теориях, будем называть *моделью памяти*. В данной работе рассматриваются только модели памяти для языка программирования Си в сочетании с различными языками спецификации или другими способами спецификации проверяемых свойств.

Рис. 1. Общая схема использования SMT-решателей в инструментах верификации.
Fig. 1. General diagram illustrating SMT-solver usage in static verification tools.

2.2 Теории, поддерживаемые современными решателями

Набор поддерживаемых решателем теорий в конечном счете определяет набор сущностей, с помощью которых может быть формализована модельная семантика используемого языка программирования, а алгоритмическая разрешимость или сложность решения задач выполнимости, соответствующих используемым комбинациям теорий, определяет вероятность успешного разрешения (с вердиктом «выполнимо»/«невыполнимо») генерируемых логических формул и требуемое для этого количество ресурсов – времени и памяти. Обзор основных теорий, поддерживаемых современными SMT-решателями, дан в диссертации [22].

В данной работе будут описаны методы моделирования семантики Си-программ с использованием комбинаций теории неинтерпретируемых функций с теориями целочисленной и вещественной линейной арифметики с кванторами и без кванторов (то есть с использованием логик QF_UFLRA, UFLRA, UFLIA и QF_UFLIA).

3. Классификация моделей памяти

В дальнейшем для каждого рассматриваемого подхода к моделированию семантики Си-программ и в частности, соответствующей модели памяти, будем рассматривать следующие характеристики:

1. Теории, используемые при моделировании. Выбор используемых теорий непосредственно влияет на эффективность метода моделирования, а также в некоторых случаях и на область его применимости (например, при необходимости поддержки интерполяции Крейга).
2. Поддерживаемые возможности языка Си. Многие распространенные методы моделирования семантики языка Си являются в той или иной степени неполными, поэтому имеет смысл говорить о полноте или, иначе говоря, о выразительности рассматриваемых методов. Выразительность метода особенно важна для инструментов дедуктивной верификации, так как они предъявляют более высокие требования к корректности работы инструмента верификации. В то время как при использовании неподдерживаемой возможности инструмент автоматической статической верификации, к примеру, может продолжить работу с выдачей предупреждения о возможности пропуска ошибки (заменив неподдерживаемый фрагмент кода на некоторое приемлемое приближение), в надежде обнаружить одну или несколько ошибок, не связанных с неподдерживаемой возможностью, инструмент дедуктивной верификации должен давать как можно больше гарантий именно для случая отсутствия

обнаруженных ошибок (так как чаще всего применяется к коду, уже прошедшему множество проверок различными инструментами верификации и тестирования).

3. Поддержка областей памяти наперед не ограниченного размера. Будем далее называть такие области памяти неограниченными. Как и выразительность метода, поддержка неограниченных областей памяти наиболее важна для инструментов дедуктивной верификации.
4. Масштабируемость метода моделирования. Под масштабируемостью будем понимать примерное число операторов языка Си, для которых соответствующая формула пути (для свойства достижимости соответствующего состояния программы), полученная с использованием рассматриваемого метода, может быть разрешена хотя бы одним из современных решателей за некоторое фиксированное время. Масштабируемость становится важной для применения метода моделирования семантики в инструментах автоматической статической верификации, использующих встраивание функций, и таким образом анализирующих длинные последовательности операторов на путях от точки входа до предполагаемого ошибочного состояния.

Будем классифицировать рассматриваемые методы моделирования памяти Си-программ по поддержке в них неограниченных областей памяти.

4. Модели для ограниченных областей памяти

В методах, поддерживающих моделирование памяти не более чем наперед заданного размера, возможно использование теорий, для которых существуют более эффективные алгоритмы разрешения формул, в частности, отказ от использования теорий массивов и логики первого порядка в пользу неинтерпретируемых функций или констант и пропозициональной логики. Самым простым с точки зрения используемых теорий является моделирование памяти Си-программ с использованием неинтерпретируемых констант, которое, как правило, опирается на результаты работы некоторого алгоритма анализа синонимичных указателей — *алиасов.*

4.1 Использование анализа алиасов

Рассматриваемый метод применялся в инструменте автоматической статической верификации BLAST [5], использующем предикатную абстракцию с уточнением по невыполнимым контрпримерам. Метод описан в статьях [23, 8], а также в [24, 25] и диссертации [26] (глава 4).

Суть данного метода можно описать следующим образом. Пусть имеется некоторый алгоритм межпроцедурного нечувствительного к контексту и

потоку управления анализа алиасов, например, [27]. Такой анализ алиасов может определять возможную синонимичность указательных выражений глобально для всей анализируемой программы. Пусть для каждого указательного выражения e из множества всех выражений $\mathcal{E}$ в программе в результате работы анализа алиасов определено множество $\mathcal{A}(e)$ указательных выражений, значение которых может совпадать со значением выражения e. Для хранения отображения $e \to \mathcal{A}(e)$ можно использовать, к примеру, его представление в виде отношения с помощью BDD [28]. Для представления изменяющихся значений всех выражений в исходной программе в рассматриваемом методе предлагается использовать последовательности индексированных неинтерпретируемых констант. Обозначим каждую индексированную неинтерпретируемую константу из конечной последовательности, представляющей изменяющееся в зависимости от состояния программы значение синтаксического выражения e (рассматриваемого в контексте некоторой функции) через $[\![e]\!]_\theta$, где θ — изменяющееся отображение синтаксических выражений в текущие индексы в соответствующих последовательностях. Рассмотрим оператор присваивания по указателю на простой тип данных, то есть не структуру, не объединение и не массив. Пусть правая часть оператора присваивания v может быть представлена в виде формулы $[\![v]\!]_\theta$. Тогда сильнейшее постусловие [29] оператора присваивания по указателю на простой тип будет выглядеть следующим образом:

$$SP(*\,e = v) \equiv Update_\theta(e, v) \wedge \bigwedge_{e' \in \mathcal{A}(e)} \mathbf{ite}([\![e]\!]_\theta = [\![e']\!]_\theta, Update_\theta(e', v), Retain_\theta(e'))$$

$$Update_\theta(e, v) \equiv [\![*\,e]\!]_{\theta'} = [\![v]\!]_\theta \wedge closure_\theta(*\,e, v)$$

$$Retain_\theta(e) \equiv [\![*\,e]\!]_{\theta'} = [\![*\,e]\!]_\theta \wedge closure_\theta(*\,e, *\,e)$$

$$\theta' = \theta \ddagger Incr_\theta(\{*\,e\} \cup \bigcup_{\substack{k=1 \\ *^k e \in \mathcal{E}}}^{\infty} \{*^k\,e\} \cup \bigcup_{e' \in \mathcal{A}(e)} \bigcup_{\substack{k=1 \\ *^k e' \in \mathcal{E}}}^{\infty} \{*^k\,e'\})$$

$$Incr_\theta(S) \equiv \bigcup_{e \in S} \{e \to \theta(e) + 1\}$$

$$closure_\theta(e, v) \equiv \bigwedge_{\substack{k=1 \\ \{*^k e, *^k v\} \subset \mathcal{E}}}^{\infty} [\![*^k\,e]\!]_{\theta'} = [\![*^k\,v]\!]_\theta$$

$$\mathbf{ite}(C, A, B) \equiv (C \wedge A) \vee (\neg C \wedge B)$$

Здесь при обновлении значения выражения по указателю происходит одновременное обновление всех его возможных алиасов (разыменований

указателей с тем же значением), а также всех присутствующих в программе разыменований обновляемого значения и его алиасов, так как записываемое по указателю значение может также являться указателем. При этом происходит увеличение на единицу индексов во всех соответствующих последовательностях.

Части формулы, соответствующие обновлению разыменований левой части оператора присваивания по указателю или синонимичных ей выражений, назовем замыканиями (англ. closure) операции присваивания по указателю. Этим частям соответствует обозначение $closure_\theta(e, v)$. Здесь $*^k\, e$ обозначает $\underbrace{* \ldots *}_{k} e$. В случае введения дополнительного ограничения наличия в исходном тексте программы соответствующих выражений ($\{*^k\, e, *^k\, v\} \subset \mathcal{E}$), глубину разыменования можно не ограничивать ($k = \infty$), получая при этом формулы ограниченного размера.

Обновления всех возможных синонимичных выражений — условные, значение разыменования синонимичного указательного выражения обновляется (обозначение $Update_\theta(e, v)$) в случае, если значение самого выражения совпадает со значением выражения в левой части оператора присваивания и остается прежним (обозначение $Retain_\theta(e)$) в противном случае.

Отображение θ' соответствует обновленному отображению θ после увеличения на единицу индексов всех потенциально обновленных выражений. Для формализации этого отображения использована операция перезаписывающего объединения отображений $m_1 \ddagger m_2$, которая обозначает отображение m_2, дополненное парами $k \to m_1(k)$, где $k \notin \operatorname{dom} m_2, k \in \operatorname{dom} m_1$.

Чтение значения по указателю e представляется в виде формулы непосредственно как $[\![e]\!]_\theta$. Благодаря использованию дополнительного ограничения $\{*^k\, e, *^k\, v\} \subset \mathcal{E}$ (рассматриваются только выражения, присутствующие в исходном коде программы) данный метод позволяет полностью поддерживать указатели на любые простые типы данных, включая другие указатели (поддерживаются типы вида $t \underbrace{* \ldots *}_{k}$ для любого $k \geq 0$). Используя эквивалентность $* \&a \equiv a$, можно также поддерживать взятие адреса у переменной. С помощью ряда приемов, описанных в [8], данный метод может быть расширен на структуры, в том числе вложенные.

При моделировании семантики операций с указателями с использованием неинтерпретируемых констант и анализа алиасов непосредственно используется только теория равенства. Поэтому данный метод может использовать различные теории, в зависимости от используемой модельной семантики операций над значениями. В инструменте BLAST использовалась

теория вещественной линейной арифметики и неинтерпретируемых функций (для приближения нелинейных операций). В принципе возможно также использование теорий нелинейной вещественной арифметики, целочисленной арифметики (линейной и нелинейной), теории битовых векторов конечной длины. Поддерживаемые возможности языка Си в данном методе ограничены простыми и структурными типами данных, не поддерживаются массивы, адресная арифметика, объединения, произвольные приведения типов указателей. Неограниченные наперед области памяти в общем случае (например, массивы переменной длины) не поддерживаются, однако может поддерживаться проверка некоторых простых свойств (не выходящих за рамки используемой логики) для произвольных рекурсивных структур данных, в том числе неограниченного размера. Для этого вместо множества $\mathcal{E}$ выражений в программе необходимо рассмотреть множество всех выражений, значение которых прямо или косвенно доступно на текущем рассматриваемом пути выполнения. Например, для пути, в котором имеются операторы $q = p;$ и $p \to f = q;$ и поле f имеет тот же тип, что и указатель p, доступно не только значение выражения $p \to f$ (прямо), но и $p \to f \to f$ (косвенно). В силу конечности длины пути, соответствующее множество доступных выражений также конечно. Благодаря использованию теорий, эффективно поддерживаемых современными решателями, данный метод хорошо масштабируем. Из данных, приведенных в статье [30], можно сделать вывод, что реализация инструмента BLAST, использующая данный метод, успешно работала для трасс длиной в несколько сотен операторов.

Рассмотрим пример построения формулы в модели памяти на основе анализа алиасов. Формулу будем строить для следующего фрагмента Си-программы с заданным проверочным условием:

```
struct range {
      int start, end;
} default = {0, 0};
…
struct range *r0 = &default, *r1;
r1 = malloc(sizeof (struct range));
r1->start = r0->start;
r0 = r1;
assert (r0->start == 0);
```

Формулу построим для случая, когда вызов malloc успешно выделяет память, возвращая ненулевой указатель. Здесь для присваиваний r0 = &default; и r0 = r1; в указатель r0 на структуру range необходимо генерировать формулы замыкания на глубину 1. То есть в формуле необходимо представлять тот

факт, что после присваивания в указатель r0 значения r0->start и r0->end также изменятся соответствующим образом. Для определения новых значений r0->start и r0->end достаточно взять соответствующие поля от правых частей соответствующих операторов присваивания. Подформулы замыкания для присваиваний в указатели на структуры далее отмечены серым цветом. Для присваивания r1->start = r0->start;, соответствующего изменению значения поля start структуры range необходимо генерировать формулы изменения значения возможных алиасов указателя r1. Будем полагать, что возможными алиасами указателя r1 являются указатели r0 и &default (в общем случае возможные алиасы зависят от используемого алгоритма анализа алиасов). В таком случае для каждого из них нужно сгенерировать формулу возможного обновления значения при условии равенства указателю r1 и формулу возможного сохранения прежнего значения при условии неравенства указателю r1. Соответствующие подформулы также отмечены далее серым цветом. Будем использовать для последовательностей значений переменных, соответствующих изменяющимся значениям полей структур, имена вида имя_структуры$имя_поля. Все последовательности будем индексировать, начиная с нуля, соответствующего начальному значению, заданному при инициализации соответствующей переменной (или поля структуры). Для проверки выполненности целевого свойства объединим полученную формулу пути с помощью конъюнкции с отрицанием формулы, выражающей целевое свойство. Если полученная формула окажется невыполнимой, то можно будет утверждать, что целевое свойство выполнено для любого выполнения проверяемого фрагмента Си-программы.

В целом построенная формула будет иметь вид:

$$
\begin{gathered}
default\$start_0 = 0 \wedge default\$end_0 = 0 \wedge \\
r0_1 = default \wedge (r0\$start_1 = default\$start_0 \wedge r0\$end_1 = default\$end_0) \wedge \\
r1_1 > 0 \wedge r1_1 \neq default \wedge \\
r1\$start_1 = r0\$start_1 \\
\wedge ((r1_1 = r0_1 \wedge r0\$start_2 = r0\$start_1 \wedge r0\$end_2 \\
= r0\$end_1) \\
\vee (r1_1 \neq r0_1 \wedge \ r0\$start_2 = r0\$start_1 \wedge r0\$end_2 \\
= r0\$end_1)) \\
\wedge ((r1_1 = defualt \wedge default\$start_1 \\
= r0\$start_1 \wedge default\$end_1 = r0\$end_1) \vee (r1_1 \\
\neq default \wedge default\$start_1 \\
= default\$start_0 \wedge default\$end_1 = default\$end_0))) \wedge \\
r0_2 = r1_2 \wedge (r0\$start_3 = r1\$start_1 \wedge r0\$end_3 = r1\$end_1) \wedge \\
\neg(r0\$start_3 = 0)
\end{gathered}
$$

Для того чтобы убедиться в ее невыполнимости, оставим в формуле только существенные конъюнкты:

$$default\$start_0 = 0 \wedge r0\$start_1 = default\$start_0 \wedge r1\$start_1 \\ = r0\$start_1 \wedge r0\$start_3 = r1\$start_1 \wedge \neg(r0\$start_3 = 0)$$

В полученной формуле легко видеть, что $r0\$start_3 = r1\$start_1 = default\$start_0 = 0$ противоречит условию $\neg(r0\$start_3 = 0)$. Таким образом, полученная формула невыполнима, а целевое свойство выполнено.

4.2 Использование неинтерпретируемых функций и логики первого порядка

Модель памяти на основе анализа алиасов неприменима для верификации программ, в которых на выполненность целевых свойств существенное влияние оказывают различные операции с массивами. Для массивов наперед не заданного размера, а также для массивов достаточно большого размера соответствующие формулы для обновлений значений переменных после присваивания в указатели на элементы массивов, а также после присваиваний в сами элементы массивов либо в принципе не могут быть получены, либо слишком быстро растут пропорционально размерам используемых массивов. Метод моделирования памяти, описанный в этом разделе, также основан на использовании анализа возможных алиасов, но позволяет поддерживать проверку некоторых свойств для программ с массивами.

В статье [31] описан метод моделирования семантики предикатов с указателями, используемый в инструменте верификации SLAM2 [32]. В методе предлагается использование теории неинтерпретируемых функций в сочетании с логикой первого порядка. При кодировании используются три неинтерпретируемые функции: A, V и S, соответствующие трем возможным операциям над *размещениями* (англ. *locations*) – произвольными объектами в памяти программы. А именно: $A(X)$ соответствует взятию адреса размещения X, $V(X)$ — взятию значения размещения X, $S(X, f)$ — получению вложенного размещения, соответствующего полю или индексу f составного размещения X (структуры или массива), а специальный случай $S(X, -1)$ (предполагается, что любое другое $f \neq -1$) — разыменованию указателя, являющегося значением размещения X. Семантика неинтерпретируемых функций формализуется с помощью набора аксиом, например, $\forall X, Y.\ A(X) = A(Y) \Longrightarrow X = Y$. Метод применим для переменных простых типов, в том числе указателей, а также структур, в том числе вложенных, и массивов. Не поддерживаются объединения и произвольные приведения типов указателей. В статье [31] не описывается моделирование семантики оператора присваивания (например, в виде соответствующего слабейшего предусловия или сильнейшего постусловия), что не позволяет однозначно сделать вывод о наборе

поддерживаемых конструкций языка Си, однако в статьях [32, 33] упоминается использование слабейших предусловий и анализа алиасов. Таким образом, вероятно, поддерживается проверка некоторых простых (неиндуктивных) свойств для рекурсивных структур данных и массивов наперед не ограниченного размера. Исходя из данных, приведенных в статье [34], масштабируемость модели памяти, используемой SLAM, сравнима с масштабируемостью модели памяти, используемой BLAST (длина трасс до нескольких сот операторов).

Рассмотрим пример фрагмента программы, использующего массивы, и проверим заданное для него целевое свойство с помощью модели памяти на основе неинтерпретируемых функций с кванторами, используемой в инструменте SLAM2.

```
struct array {
      int len, *data;
} a = {2, (int []){0, 0}};
…
struct array *b = &a;
b->data[0] = 1;
assert (b == &a && a.data[0] == 1 && b->len == 2);
```

Несмотря на то, что рассматриваемая модель памяти использует логику первого порядка, для построения формул в ней также требуется использование анализа алиасов. Для целевого пути в программе строится слабейшее предусловие по рекурсивным правилам преобразования предикатов, которые требуют знания о возможных алиасах всех указательных выражений в пути для правильной подстановки значений переменных в формуле при обработке операторов присваивания. Более точно, соответствующее преобразование слабейшего предусловия для присваивания вида $* x = e$:

$(x = \&y_1 \wedge Q[e/y_1]) \vee \ldots \vee (x = \&y_k \wedge Q[e/y_k])$

Здесь Q — текущее слабейшее предусловие для нижней части пути, $y_1, \ldots, y_k$ — переменные, которые может адресовать указательное выражение x, $[x\ /\ y]$ означает подстановку выражения x вместо переменной y (правила построения формулы гарантируют корректность такой подстановки). Таким образом, каждое присваивание по указателю дублирует нижнюю часть формулы отдельно для каждого возможного рассматриваемого значения указательного выражения. В приведенном примере значение указателя `b->data[0]` в операторе присваивания `b->data[0] = 1;` может быть разрешено однозначно. Поэтому в результирующей формуле рассматривается

только один случай `b->data[0]==&a.data[0]`. Формула пути так же, как и в примере для модели памяти на основе анализа алиасов, объединяется с отрицанием проверяемого свойства. Поля структуры array имеют индексы: len — 0, data — 1. Для получения размещения поля структуры используется функция $S(l,f)$, где l — размещение структуры, f — индекс поля. Для моделирования операции разыменования используется частный случай применения функции S при $f=-1$. Для доказательства невыполнимости формулы в данном примере требуется только одна аксиома, задающая свойство функций S,V и A: $\forall X,Y.V(X)=A(Y)\Longrightarrow S(X,-1)=Y$. Аксиома утверждает, что если значение указателя X равно адресу переменной Y, то результатом разыменования указателя X является переменная Y (соответствующее ей размещение). Полученная формула имеет вид:

$$\begin{aligned}(\forall X,Y.V(X)=A(Y)\Longrightarrow S(X,-1)=Y)\wedge V\big(S(a,0)\big)=2\wedge V(b)\\ =A(a)\wedge V\big(S(S(S(S(b,-1),1),-1),0)\big)\\ =1\\ \wedge\neg\big(V(b)=A(a)\wedge V\big(S(S(S(a,1),-1),0)\big)\\ =1\wedge V\big(S(S(b,-1),0)\big)=2\big)\end{aligned}$$

Аксиома позволяет заменить выражения $S(b,-1)$ внутри аргументов функции V на a, показав тем самым невыполнимость построенной формулы.

Модель памяти на основе теории неинтерпретируемых функций с кванторами, однако, плохо справляется с некоторыми случаями вложенных массивов (массивов, содержащих внутри элементов указатели на элементы других массивов). Рассмотрим следующий пример и соответствующую формулу:

```
struct array *b[] = {&a, &a};
int i = nondet_int() ? 0 : 1;
b[i]->data[0] = 1;
assert (a.data[0] == 1);
```

$$\begin{aligned}(\forall X\,Y.X=A(Y)\Longrightarrow S(X,-1)=Y)\wedge((V\big(S(S(b,0),-1)\big)=A(a)\wedge V(i)\\ =0\wedge V\big(S(S(S(S(b,0)),-1),1)-1),0)\big)\\ =1\wedge\neg\big(V\big(S(S(S(a,0),-1),0)\big)=1\big)\vee(V\big(S(S(b,1),-1)\big)\\ =A(a)\wedge V(i)=1\wedge\\ V\big(S(S(S(S(b,1)),-1),1)-1),0)\big)=1\wedge\neg\big(V\big(S(S(S(a,0),-1),0)\big)=1\big)))\end{aligned}$$

Здесь внешняя функция nondet_int()моделирует недетерминированный выбор целочисленного значения и отдельно рассматриваются два случая для различных возможных значений переменной i. Нетрудно видеть, что размер таких формул для фиксированной длины рассматриваемого пути пропорционален длине

используемых в программе массивов указателей и так же, как и в случае с моделью памяти на основе анализа алиасов, легко может оказаться неприемлемо большим для достаточно длинных рассматриваемых путей или достаточно больших массивов указателей. Данную проблему позволяют решить модели памяти для областей наперед не ограниченного размера.

5. Модели для неограниченных областей памяти

Модели памяти, поддерживающие произвольные области памяти наперед не ограниченного размера (в частности, массивы переменной длины), опираются на использование логики первого порядка или теории массивов. При использовании этих моделей памяти размер формулы пути никогда не зависит от размеров упоминаемых на этом пути областей памяти программы (например, размеров массивов или числа элементов в связном списке).

5.1 Типизированная модель памяти

Типизированная модель памяти реализована, в частности, в инструменте дедуктивной верификации VCC2 [35] и описана в статье [36]. Основная особенность этой модели памяти — в том, что в контексте инструмента дедуктивной верификации в этой модели памяти делаются дополнительные предположения о семантике моделируемых операторов исходной программы без потери корректности и полноты моделирования. Это возможно благодаря наличию задаваемых пользователем аннотаций, а также возможности генерации нескольких условий верификации вместо одной общей формулы пути. В частности, можно потребовать от пользователя аннотировать некоторые дополнительные аспекты семантики верифицируемой программы, осуществить моделирование семантики в заданных пользователем предположениях, а затем помимо обычных условий верификации сгенерировать дополнительные проверки для заданных пользователем предположений. При этом в контексте дедуктивной верификации при моделировании памяти помимо собственно состояния памяти программы можно использовать также и дополнительное модельное (англ. model или ghost) состояние, изменяемое как в результате выполнения операторов исходной программы, так и в результате выполнения специальных модельных операций, задаваемых в виде аннотаций.

В модели памяти VCC2 вводится модельное отображение пар вида (адрес, тип) в специальный булевый флаг валидности, который определяет, является ли данный адрес адресом начала объекта соответствующего типа. Так как Си является слабо типизированным языком, модельное отображение валидности типизированных указателей в VCC2 имеет состояние, которое может быть изменено с помощью специальных модельных операций. Таким образом,

модельное отображение валидности остается постоянным при моделировании семантики одного оператора, но может изменяться в ходе выполнения программы. При этом для обеспечения корректности моделирования памяти поддерживается инвариант о том, что в каждом состоянии для каждого адреса может быть не более одного соответствующего валидного типа данных. В силу отсутствия непосредственного моделирования защиты памяти в модельной семантике VCC2 можно считать, что для каждого адреса этот тип всегда ровно один. Моделирование семантики программы происходит в предположении о том, что каждый указатель содержит адрес, валидный тип которого совпадает с типом указателя. Такой указатель будем называть *валидным*. Таким образом, моделируемая программа должна обращаться к памяти только через валидные указатели. Это свойство проверяется с помощью генерации соответствующих условий верификации. В VCC2 используется две модельные операции для изменения состояния модельного отображения валидности: одна для разбиения произвольного объекта в памяти в массив байт (char), а другая – для соединения массива байт в объект произвольного типа соответствующего размера. Семантика этих двух операций включает преобразование значений в моделируемой памяти в/из типа char в соответствии с побайтовой структурой преобразуемого объекта. При этом для кодирования значений различных типов могут в принципе использоваться различные логические теории. Кроме этого, модель памяти VCC2 предполагает использование логики первого порядка для задания дополнительных аксиом, ограничивающих возможные пересечения адресуемых объектов в памяти программы случаем вложения. Вместе использование типизации памяти и дополнительных аксиом позволяет генерировать более простые условия верификации, увеличивая масштабируемость модели памяти. Кроме этого, в большинстве случаев уменьшается размер задаваемых пользователем спецификаций, так как в них не требуется явно указывать отсутствие пересечений (за исключением вложения) между объектами различающихся типов в памяти программы.

В статье [36] модель памяти описывается с помощью введения базового («игрушечного», «toy language») языка описания трасс для дедуктивной верификации и формализации исходной (нетипизированной) и модельной (типизированной) семантик этого базового языка. При этом трассы на базовом языке представляют собой последовательности операторов, а обе семантики определятся как мелкошаговые [37] с помощью правил вывода отношений вычисления и рекуррентных соотношений для функций означивания чистых выражений. Множество значений (результатов вычисления) последовательностей инструкций принимается состоящим из двух элементов – $\top$ и $\bot$, соответствующих результату проверки всех условий верификации. $\top$ означает выполненность всех условий верификации, $\bot$ – невыполненность хотя бы одного из них. Промежуточное состояние вычисления определяется

как объединение множества $\{\top, \bot\}$ и множества промежуточных состояний, содержащих оставшуюся часть последовательности операторов и некоторый набор отображений, определяющих состояние памяти программы и модельного отображения валидности после выполнения обработанной части трассы. Таким образом, моделирование памяти программы с помощью SMT-формул описывается неявно, однако достаточно компактно и удобно для доказательства свойств рассматриваемой модельной семантики (в частности, её корректности относительно исходной семантики).

Рассмотрим описанный способ формализации модельной семантики подробнее на примере оператора присваивания значения по указателю $* e_1 = e_2$. Предполагается, что сами выражения e_1и e_2 не содержат операции с указателями.

В исходной семантике присваивание значения по указателю на значение типа t будет моделироваться как одновременное обновление значения (sizeof t) последовательных байт в нетипизированной куче, начиная с адреса, определяемого значением выражения e_1. При этом после вычисления значение выражения e_2 обязательно должно быть разбито на (sizeof t) байт и каждый из них должен быть записан в нетипизированную кучу по соответствующему адресу, так как без привлечения каких-либо дополнительных предположений невозможно гарантировать, что ни один из этих изменяемых байт не попадает в область памяти, адресуемую каким-либо другим указательным выражением e_3 (возможно, имеющим тип, отличный от $t*$), по которому может происходить обращение в память далее на рассматриваемом пути выполнения. Таким образом, каждый оператор присваивания по указателю в нетипизированной семантике требует моделирования значений используемых выражений на побайтовом уровне, возможного моделирования обновления значений сразу по нескольким адресам в куче, а также явной спецификации дополнительных предположений о непересечении адресов между объектами различных типов. Все эти факторы приводят к генерации трудноразрешимых условий верификации.

Поэтому при построении формул в инструментах дедуктивной верификации вместо исходной семантики языка программирования обычно используется его модельная семантика.

В модельной семантике оператору присваивания $* e_1 = e_2$ соответствует правило вывода для соответствующего отношения вычисления :

$$e_1 : t* \Longrightarrow \langle * e_1 = e_2; R | \mathcal{E}, \mathcal{T}, \mathcal{M} \rangle \blacktriangleright \langle R | \mathcal{E}, \mathcal{T}, \mathcal{M} \ddagger \{ [\![e_1]\!]^d_{\mathcal{E}} \to [\![e_2]\!]^{8*\text{sizeof } t}_{\mathcal{E}} \} \rangle$$

Это правило вывода по сути означает, что если выражение e_1 имеет тип $t*$, то промежуточное состояние вычисления, в котором первым оператором является $* e_1 = e_2$, а оставшаяся часть пути — R, связано (несимметричным) отношением с промежуточным состоянием, в котором присутствует только

оставшаяся часть пути R (то есть обработан один оператор $*e_1 = e_2$). При этом эти два связанных отношением промежуточных состояния вычисления отличаются состоянием памяти программы. Память описывается тремя отображениями: $\mathcal{E}, \mathcal{T}$ и $\mathcal{M}$, где $\mathcal{E}$ — означивание переменных (в базовом языке они все находятся в одной области видимости), $\mathcal{T}$ — упомянутое ранее отображение валидности, отображающее адреса в однозначно соответствующие им типы данных размещенных по этим адресам значений, а $\mathcal{M}$ — отображение пар вида (адрес, тип) в значения соответствующих типов. После выполнения оператора присваивания в память по адресу $[\![e_1]\!]_{\mathcal{E}}^{d}$ (результат вычисления выражения e_1 без побочных эффектов и без операций с указателями в контексте $\mathcal{E}$, представленный в виде битового вектора длины d) будет записано значение $[\![e_2]\!]_{\mathcal{E}}^{8*\mathrm{sizeof}\,t}$ длиной (**sizeof** t) байт.

В формуле сильнейшего постусловия для представления значения отображения $\mathcal{M}$ может быть использовано несколько последовательностей логических массивов различного типа — по одной последовательности для каждого типа данных. При этом для целочисленных типов данных можно использовать, к примеру, теорию линейной арифметики, так как в модельной семантике в силу неявного предположения о разделении памяти по типам формализация операций на побитовом уровне не обязательна. Таким образом, рассмотрение семантики на побитовом уровне в каждом присваивании не происходит. Кроме этого, эффективно моделируется неявное предположение о непересечении значений различных простых типов данных в памяти программы.

Рассмотренное правило вывода, однако, нельзя непосредственно применять для каждого оператора присваивания, так как используемые в нем неявные предположения о валидности указателя в левой части оператора присваивания могут быть не выполнены. Поэтому каждый раз при использовании данного правила вывода инструмент дедуктивной верификации дополнительно генерирует проверку соответствующего предусловия $\mathcal{T}\left[[\![e_1]\!]_{\mathcal{E}}^{d}\right] = t$.

Если предусловие окажется невыполненным, то соответствующее промежуточное состояние $\langle *e_1 = e_2; R|\mathcal{E}, \mathcal{T}, \mathcal{M}\rangle$ окажется связано отношением с ошибочным состоянием $\perp$.

Более полный пример формализации модельной семантики описанным способом можно найти в статье [38].

Итак, модель памяти VCC2 предполагает использование теории массивов *и/или* логики первого порядка, а также других теорий для моделирования операций над данными, при этом для каждого типа данных может быть использован отдельный набор теорий. При отсутствии обращений к одной и той же области памяти через указатели различных типов в рамках одного оператора (это ограничение можно считать чисто синтаксическим и легко

устранимым с помощью использования дополнительных переменных), модель памяти VCC2 поддерживает все конструкции языка Си и применима для областей памяти произвольного наперед не заданного размера. Однако использование в контексте инструмента дедуктивной верификации говорит о применении модели памяти к трассам лишь небольшого размера, как правило, ограниченных рамками одной функции, то есть длиной порядка нескольких десятков операторов.

В следующем разделе на примере будет рассмотрена модель памяти, являющаяся уточнением типизированной модели памяти, использованной в инструменте VCC2.

5.2 Модель Бурстала-Борната

Модель Бурсталла-Борната, также известная как модель «поле как массив» (англ. Burstall-Bornat model, component-as-array model) была предложена и использована в работах [39, 40] в контексте частично автоматизированной дедуктивной верификации. Основная идея этой модели памяти заключается в том, чтобы раздельно представлять состояния отдельных компонентов составных объектов, что для языка Си по сути означает разделение представления состояния памяти программы по полям структур. Представление состояния каждого отдельного поля при описании модели может быть не конкретизировано, так же, как и при описании типизированной модели памяти (на практике может использоваться теория массивов или аксиоматизация в логике первого порядка). Основную проблему при описании этой модели памяти для языка Си составляют переменные и поля структур, доступные по указателям, то есть потенциально все переменные и поля, для которых в исходном коде Си-программы присутствует операция взятия адреса &.

Одним из решений этой проблемы является применение к исходному коду программы нормализующих преобразований, устраняющих адресацию переменных и полей структур с помощью переписывания их в указатели и введения специальных фиктивных структур с одним полем соответствующего типа. Такие преобразования описаны, например, в статьях [41, 42, 43]. Другим возможным решением является введение отдельных отображений для представления состояния адресуемых указателями полей и переменных, например, с использованием разделения по типам аналогично типизированной модели памяти. В рамках самой модели Бурсталла-Борната выбор одного из этих решений не существенен, так как в итоге приводит к построению одинаковых логических формул (адресуемые переменные и поля в любом случае разделяются только по типам). Однако при расширении данной модели, например, с помощью дальнейшего разделения представления состояния памяти, перетипирующие преобразования исходного кода оказываются проще, по крайней мере, при теоретическом описании полученных моделей. Такие

преобразования применяются, например, в инструменте дедуктивной верификации Jessie. Решение без дополнительных преобразований используется, например, в инструменте дедуктивной верификации WP [44], реализующем модель Бурсталла-Борната. Один из вариантов этой модели памяти реализован также в инструменте дедуктивной верификации VCC3 [2]. Для достижения полноты поддержки возможностей языка Си в инструментах дедуктивной верификации модель Бурсталла-Борната может быть расширена модельным состоянием и специальными модельными операциями разбиения и соединения аналогично типизированной модели памяти.

Модель Бурсталла-Борната в целом обладает примерно теми же характеристиками, что и типизированная модель памяти, однако обеспечивает в общем случае меньшее количество индексов, приходящихся на одно отображение, представляющее часть состояния памяти программы и, как показывают результаты сравнения моделей памяти, в частности, [2], может быть более эффективна на практике по крайней мере в контексте дедуктивной верификации.

Рассмотрим модель Бурсталла-Борната на приведенном ранее примере, в котором происходит потенциальное ухудшение масштабируемости модели памяти, используемой в инструменте SLAM:

```
struct array *b[] = {&a, &a};
int i = nondet_int() ? 0 : 1;
b[i]->data[0] = 1;
assert (a.data[0] == 1);
```

В данном примере будем представлять значения поля `data` структуры `array` с помощью последовательности индексированных неинтерпретируемых функций $array\_data_i$, значения массивов типа **`int`** — с помощью последовательности int_i, а значения массивов типа **`struct`** `array *` — с помощью последовательности $struct\_array\$_i$. Для каждого присваивания по указателю в этой модели памяти добавляется аксиома, утверждающая сохранение значений соответствующей неинтерпретируемой функции для всех индексов (адресов), отличных от изменяемого в операторе присваивания. Результирующая формула имеет вид:

$$\begin{aligned} struct\_array\$_1(b) = a \wedge struct\_array\$_1(b+1) = a \wedge i_1 \\ = \boldsymbol{ite}(x = 0{,}0{,}1) \\ \wedge\, int_1\big(array\_data_0\big(struct\_array\$_1(b+i)\big) + 0\big) \\ = 1 \\ \wedge\, \big(\forall X.\, X \neq array\_data_0\big(struct\_array\$_1(b+i)\big) + 0 \\ \Longrightarrow int_1(X) = int_0(X)\big) \wedge \neg\big(int_1\big(array\_data_0(a)\big) = 1\big) \end{aligned}$$

Здесь неинтерпретируемая константа x используется для моделирования недетерминированного выбора значения переменной i из множества $\{0, 1\}$ (соответствует оператору i = nondet_int() ? 0 : 1;). По сравнению с формулой, построенной ранее с использование модели памяти инструмента SLAM, описанной в разделе 4.2, размер данной формулы не зависит от размера используемых на рассматриваемом пути массивов (в том числе массивов указателей). Для установления ее невыполнимости необходимо рассмотрение случаев $i_1 = 0$ и $i_1 = 1$, однако оно может происходить непосредственно во время разрешения формулы, не влияет существенно на ее размер и позволяет полнее использовать возможности алгоритмов разрешения, реализованные в современных SMT-решателях — для длинных путей с большой вероятностью будут рассмотрены различные случаи только для переменных, существенно влияющих на выполнимость формулы.

В случае использования в программе большого числа структур одинакового типа, модель Бурсталла-Борната, предполагающее разбиение памяти программы по полям структур, может становиться менее эффективной. Ее можно оптимизировать с использованием результатов некоторого алгоритма анализа алиасов. В случае использования простого алгоритма на основе системы непересекающихся множеств, который обеспечивает корректность модели памяти для существенного подмножества Си-программ, получим модель памяти с регионами.

5.3 Модель с регионами

Это расширение модели памяти Бурсталла-Борната предложено в статье [41] и используется в инструменте дедуктивной верификации Jessie [45] (а также в его предшественнике, инструменте Caduceus [46]). Основная идея этой модели памяти заключается в том, что память программы в логическом представлении может разделяться не только по компонентам составных объектов (для языка Си – полям структур), как в модели Бурстала-Борната, но и по регионам – непересекающимся множествам указательных выражений, таких, что выражения из разных множеств не могут адресовать пересекающиеся области памяти. При этом эти два критерия разделения состояния памяти могут применяться независимо друг от друга.

Формально предложенный в статье [41] подход к моделированию памяти Си-программ включает в себя три составляющие.

Первая из них – это набор нормализующих преобразований, применяемых к исходной Си-программе, в результате которого получается программа на базовом языке. В программе на базовом языке в качестве типов переменных и полей структур допускаются только целые числа и указатели на структуры. В частности, структуры не могут непосредственно (без использования указателей) содержать в себе другие структуры или массивы, не допускается использование

объединений и приведение типов указателей. Все структуры и массивы в программах на базовом языке размещаются в динамической памяти.

Вторая составляющая – это специальная система типов для базового языка, в которой типы указателей на структуры параметризованы регионами. При этом регионы указательных типов параметров функций сами являются параметрами функций, что порождает систему типов с параметрическим полиморфизмом, аналогичную классической системе Хендли-Милнера [47]. Основной целью введения системы типов для базового языка является существенное упрощение доказательства теоремы об относительной корректности модели памяти с регионами. Теорема утверждает, что корректно типизированная программа на базовом языке имеет одинаковую семантику в модели памяти с регионами и в модели Бурсталла-Борната.

Наконец, третьей составляющей подхода является алгоритм вывода типов (регионов) для базового языка. В статье [41] этот алгоритм приводится без формального доказательства корректности, в предположении, что после вывода типов программа на базовом языке будет транслирована на язык WhyML [48] (имеющий систему типов с параметрическим полиморфизмом) так, что все типы, выведенные для базового языка будут непосредственно выражены с помощью типов языка WhyML в результирующей программе. Таким образом, будет осуществлена апостериорная проверка корректности вывода типов для программы на базовом языке.

Для моделирования семантики языка Си с использованием модели памяти с регионами, как и для двух ранее рассмотренных моделей, может использоваться теория массивов или комбинация теории неинтерпретируемых функций с логикой первого порядка, а также другие теории для моделирования операций над данными. Возможности языка Си, поддерживаемые моделью памяти с регионами, в том виде, в котором она описана в статье [41], ограничены возможностями используемого в статье базового языка. Таким образом, модель не поддерживает по крайней мере вложенность объектов, объединения и произвольные приведения типов указателей. В остальном, модель с регионами применима для областей памяти в том числе наперед не ограниченного размера. Масштабируемость модели памяти с регионами незначительно отличается от масштабируемости типизированной модели памяти и модели Бурсталла-Борната, однако модель с регионами обеспечивает в общем случае еще меньшее число индексов на отображение, чем в этих моделях, и может быть эффективнее на практике.

Приведем пример Си-программы и соответствующей формулы для проверки целевого свойства с использованием модели памяти с регионами:

```
struct cords {
      int len, *x, *y;
} v0 = {2, (int []){0, 0}, (int []){0, 0}};
…
```

```
struct coords *v[] = {&v0, &v0};
int i = nondet_int() ? 0 : 1;
v[i]->x[0] = 1;
assert (v0.y[0] == 0);
```

$$
\begin{aligned}
& v0\_len_0(v0) = 2 \wedge v0\_x_0(v0) > 0 \wedge v0\$x_0(v0\_x_0(v0) + 0) \\
& \quad = 0 \wedge v0\$x_0(v0\_x_0(v0) + 1) = 0 \wedge v0\_y_0(v0) \\
& \quad > 0 \wedge v0\$y_0(v0\_y_0(v0) + 0) = 0 \wedge v0\$y_0(v0\_y_0(v0) + 1) \\
& \quad = 0 \wedge v_1(v + 0) = v0 \wedge v_1(v + 1) = v0 \wedge i_1 \\
& \quad = \boldsymbol{ite}(x, 0, 1) \wedge v0\$x_1\big(v0\_x_0\big(v_1(v + i)\big) + 0\big) \\
& \quad = 1 \\
& \quad \wedge \Big(\forall X. X \neq v0\_x_0\big(v_1(v + i)\big) + 0 \Longrightarrow v0\$x_1(X) = v0\$x_0(X)\Big) \\
& \quad \wedge \neg(v0\$y_0(v0\_y_0(v0) + 0) = 0)
\end{aligned}
$$

Здесь для именования индексированных последовательностей функций используются имена вида имярегиона_имяполя[необязательное]. Имя поля отсутствует для регионов, соответствующих простым типам данных. Регионы, соответствующие разыменованиям указательных полей, обозначаются как имяуказателянаструктуру$имяполя.

В данном примере предполагается, что анализ регионов разделил указательные выражения в программе на три региона: $v0$, $v0\$x$ и $v0\$y$. При этом регион $v0$ соответствует структурному типу struct cords, который имеет три поля (len, x и y), что дает три индексированные последовательности функций: $v0\_len_i, v0\_x_i$ и $v0\_y_i$ для представления состояния этого региона. Регионы $v0\$x$ и $v0\$y$ соответствуют простому типу данных int и поэтому их состояние кодируются с помощью индексированных последовательностей $v0\$x_i$ и $v0\$y_i$.

По сравнению с формулой, построенной с использованием модели памяти Бурсталла-Борната, элементы массивов `v0.x` и `v0.y` в данной формуле представляются независимо с помощью различных последовательностей неинтерпретируемых функций ($v0\$x_i$ и $v0\$y_i$). Таким образом, состояние этих массивов изменяется независимо. Поэтому невыполнимость построенной формулы легко устанавливается по наличию двух несовместных условий: $v0\$y_0(v0\_y_0(v0) + 0) = 0$ и $\neg(v0\$y_0(v0\_y_0(v0) + 0) = 0)$ (в формуле выделены серым цветом).

5.4 Полное моделирование семантики относительно логики множеств элементов списков

Все рассмотренные ранее модели памяти обладают тем ограничением, что не позволяют проверять индуктивно определяемые свойства рекурсивных структур данных, такие, например, как “все элементы односвязного списка неотрицательны” или “данный элемент списка адресуется его головным

элементом и только им". Для одного из классов свойств рекурсивных структур данных, а именно – класса LISBQ – Logic of Interpreted Sets and Bounded Quantification – логики интерпретируемых множеств и ограниченной квантификации, существует метод построения полных относительно этой логики слабейших предусловий для операторов присваивания значения в поля структур по указателю. Этот метод был реализован в инструменте дедуктивной верификации Си-программ HAVOC [40]. Метод детально описан в статье [50] и в данной статье приведем лишь пример формулы для проверки свойства "все элементы односвязного списка отличны от нуля" для одного оператора присваивания, осуществляющего объединение (склеивание или конкатенацию) двух таких списков:

```
struct list {
      struct list *next;
      int v;
};
…
struct list *l1, *l2, *l;

assume(
```

$$(\forall X \in Btwn(list.next, l1, NULL).list.next(X) \neq NULL \wedge list.v(X) \neq 0) \wedge$$
$$(\forall X \in Btwn(list.next, l2, NULL).list.next(X) \neq NULL \wedge list.v(X) \neq 0) \wedge$$
$$l \neq NULL \wedge$$
$$Reach(list.next, l1, l, NULL) \wedge Reach(list.next, l2, NULL, NULL) \wedge$$
$$\neg Reach(list.next, l2, l, l));$$

```
l->next = l2;
assert(
```

$\forall X \in Btwn(list.next, l1, NULL).list.v(X) \neq 0);$

Здесь предикат вида $\forall X \in Btwn(s.f, e1, e2).P(X)$ обозначает выполненность свойства $P(X)$ для всех элементов, достижимых из элемента, адресуемого указателем $e1$, включительно, с помощью прохождения по полям f структур с тегом s, за исключением поля f структуры, адресуемой указателем $e2$. Предикат $Reach(s.f, e1, e2, e3)$ обозначает достижимость по полям f структур с тегом s элемента, адресуемого $e2$, из элемента, адресуемого $e1$, и элемента, адресуемого $e3$ из элемента, адресуемого $e2$.

В генерируемых формулах для каждого поля структуры $s.f$ используется отдельный предикат $Reach_{s.f}$. Так как в данном примере по существу используется только поле $list.next$, будем для удобства обозначать предикат $Reach_{s.f}(e_1, e_2, e_3)$ как $e_1 \longrightarrow e_2 \longrightarrow e_3$. С использованием дополнительно вводимого обозначения $u \underset{\nrightarrow w}{\longrightarrow} v$ формулу для данного примера можно представить следующим образом:

$$u \underset{\nrightarrow w}{\longrightarrow} v \equiv u \longrightarrow v \longrightarrow w \vee (u \longrightarrow v \longrightarrow v \wedge \neg(u \longrightarrow w \longrightarrow w))$$

$$WP\left(\begin{matrix} Pre; l \to next = l2; \\ \neg(\forall X \in Btwn(list.next, l1, NULL).\, list.v(X) \neq 0) \end{matrix}\right)$$
$$\equiv Pre$$
$$\wedge \neg\Big(\Big(l_1 \underset{\nrightarrow l}{\to} NULL \Longrightarrow \forall X.\, l_1 \longrightarrow X \longrightarrow NULL \Longrightarrow list.v(X) \neq 0\Big)$$
$$\wedge \Big(l \neq NULL \wedge l_1 \underset{\nrightarrow NULL}{\longrightarrow} l \wedge l_2 \underset{\nrightarrow l}{\to} NULL$$
$$\Longrightarrow (\forall X.\, l_1 \longrightarrow X \longrightarrow l \Longrightarrow list.v(X) \neq 0)$$
$$\wedge (\forall X.\, l_2 \longrightarrow X \longrightarrow NULL \Longrightarrow list.v(X) \neq 0)\Big)\Big)$$

Здесь Pre обозначает формулу для предусловия, заданного с помощью конструкции assume, $\mathrm{W}P(op, \varphi)$ — преобразователь предикатов для слабейшего предусловия.

Метод полного моделирования семантики относительно логики LISBQ предполагает использование логики первого порядка и теории неинтерпретируемых функций. При этом существующие реализации этого метода в инструментах дедуктивной верификации генерируют формулы, для которых выполнено дополнительное ограничение стратификации по сортам [51]. В условиях этого ограничения получаемые формулы в логике первого порядка алгоритмически разрешимы. Кроме этого, для соответствующего фрагмента логики первого порядка были предложены эффективные на практике разрешающие алгоритмы [52], реализованные, в частности, в решателе Z3. В результате масштабируемость метода (при использовании соответствующих эффективных решателей) позволила применить его в инструменте дедуктивной верификации HAVOC. Метод позволяет моделировать семантику программ, использующих рекурсивные структуры данных наперед не ограниченного размера, для проверки целевых свойств корректности, заданных в логике LISBQ. Эта логика помимо рассмотренного предиката $Reach$ и множества $Btwn$, а также соответствующего предиката $\in$, вводит еще и класс множеств вида $s.f^{-1}(o)$, каждое из которых содержит все указатели на структуры с тегом s, которые в своем поле f содержат адрес структуры, адресуемой указателем o.

Следует отметить, что поддерживаемый данным методом класс свойств не включает в себя, к примеру, такие индуктивно определяемые функции, как сумма или число элементов списка.

6. Выводы

Подведем общие итоги обзора известных существующих методов моделирования семантики операций над указателями для Си-программ с помощью логических формул в теориях, то есть моделей памяти.

1. Модели памяти могут предполагать использование различных комбинаций логических теорий (логик), причем модель памяти может как ограничивать набор возможных комбинаций теорий, так и поддерживать несколько различных комбинаций теорий, возможно с привлечением различных дополнительных предположений.
2. В различных моделях памяти делаются различные предположения о верифицируемых программах и проверяемых свойствах. Проверка этих предположений может как предусматриваться в рамках самой модели памяти или соответствующего инструмента верификации, так и не предусматриваться. Это может влиять на применимость модели памяти в различных классах инструментов верификации. Например, инструменты дедуктивной верификации должны делать как можно меньше непроверенных предположений о проверяемой программе.
3. Модели памяти различаются по полноте в смысле поддержки конструкций языка Си, а также поддержки неограниченных областей памяти. Соответствующие ограничения также влияют как на применимость модели памяти в различных инструментах верификации, так и на ее применимость для решения различных практических задач. Например, инструменты дедуктивной верификации должны как можно более полно поддерживать определенный набор конструкций языка программирования. Однако, с другой стороны, при постановке задачи реализации новой верифицированной программы в отличии от задачи верификации существующего кода, набор поддерживаемых конструкций может быть ограничен. В инструментах автоматической статической верификации, верифицирующих практически исключительно существующий код, ограничения на входные программы могут существенно сужать область применимости инструмента, однако на практике часто допускается пропуск ошибок (для небольшого числа проверяемых программ) при использовании в исходном коде программы неподдерживаемых возможностей языка.
4. Так как различные наборы логических теорий (логики) обладают разными характеристиками сложности (как правило, от NP-полной до алгоритмически неразрешимой), при использовании логик с большой алгоритмической сложностью, что практически неизбежно в некоторых контекстах, таких как дедуктивная верификация свойств функциональной корректности, большое значение может иметь принадлежность генерируемых формул некоторому фрагменту используемой логики с лучшими характеристиками сложности. Также важны некоторые другие свойства генерируемых формул, существенно влияющие на производительность разрешающих

алгоритмов, такие как количество индексов, приходящихся на логический массив или последовательность неинтерпретируемых функций.

В табл. 1 представлены основные характеристики рассмотренных в данной главе моделей памяти.

Для основных характеристик моделей памяти в таблице приведено одно из трех значений: «да», «нет» и «частично». Значение «частично» в графе «рекурсивные структуры» для всех моделей памяти, кроме LISBQ, соответствует отсутствию возможностей представления в модели памяти соответствующих индуктивных предикатов. Значение «частично» в графе «Неограниченные области памяти» для модели памяти на основе анализа алиасов соответствует отсутствию возможности задания предикатов для неограниченных областей памяти. Значение «частично» в графе «Неограниченные области памяти» для модели памяти SLAM соответствует ограниченности длин вложенных массивов.

Табл. 1. Основные характеристики моделей памяти.

Table 1. Key features of the memory models.

Модель памяти	**Логики**	**Возможности языка Си**			
		рекурсивные структуры	массивы и адресная арифметика	объединения и приведения типов ук-лей	неограниченные области памяти
На основе анализа алиасов	QF_LIA, QF_LRA	**част.**	**нет**	**нет**	**част.**
SLAM	UFLIA, UFLRA	**част.**	**да**	**нет**	**част.**
Типизированная, Бурсталла-Борната	QF_ALIA, QF_ALRA, UFLIA, UFLRA	**част.**	**да**	**да**	**да**
С регионами	QF_ALIA, QF_ALRA, UFLIA, UFLRA	**част.**	**да**	**да**	**да**
LISBQ	UFLIA, UFLRA	**да**	**нет**	**нет**	**да**

Список литературы

[1]. Leonardo de Moura, Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In Formal Methods: Foundations and Applications, 12th Brazilian Symposium on Formal Methods (SBMF 2009). Gramado, Brazil, August 19–21, 2009. Revised Selected Papers, Lecture Notes in Computer Science, vol. 5902, pp. 23–36. Springer, 2009.

[2]. Sascha Böhme, Michał Moskal. Heaps and Data Structures: A Challenge for Automated Provers. In Nikolaj Børner, Viorica Sofronie-Stokkermans, eds. Automated Deduction. Lecture Notes in Computer Science, vol. 6803, pp. 177–191. Springer, 2011.

[3]. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, Helmut Veith. Counterexample-Guided Abstraction Refinement. In Proceedings of the 12th International Conference on Computer Aided Verification (CAV '00), E. Allen Emerson, A. Prasad Sistla (Eds.). Springer-Verlag, London, UK, pp. 154–169, 2000.

[4]. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), LNCS, vol. 1579, pp. 193–207. Springer-Verlag, 1999.

[5]. Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar. The software model checker Blast: Applications to software engineering. International Journal on Software Tools for Technology Transfer, October 2007, vol. 9, issue 5, pp. 505–525.

[6]. Mary Sheeran, Satnam Singh, Gunnar Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. Formal Methods in Computer-Aided Design. Lecture Notes in Computer Science, vol. 1954, pp. 127–144.

[7]. Susanne Graf, Hassen Saïdi. Construction of abstract state graphs with PVS. Proceedings of International Conference on Computer Aided Verification, 1997. LNCS, vol. 1254, pp. 72–83. Springer, Berlin Heidelberg, 1997.

[8]. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Kenneth L. McMillan. Abstractions from proofs. In Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL '04). SIGPLAN Notes, vol. 39, no. 1, pp. 232–244. ACM, New York, NY, USA. 2004.

[9]. Aaron R. Bradley. SAT-based model checking without unrolling. In Proceedings of the 12th International conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'11), Ranjit Jhala, David Schmidt (Eds.), Springer-Verlag, Berlin, Heidelberg, pp. 70–87, 2011.

[10]. Corina S. Păsăreanu, Willem Visser, David Bushnell, Jaco Geldenhuys, Peter Mehlitz, Neha Rungta. Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis. Automated Software Engineering, 2013, vol. 20, no. 3, pp. 391–425.

[11]. В. К. Кошелев, В. Н. Игнатьев, А. И. Борзилов. Инфраструктура статического анализа программ на языке C#. Труды ИСП РАН, том 28, вып. 1, стр. 21–40. 2016. DOI: 10.15514/ISPRAS-2016-28(1)-2.

[12]. Hoare C. A. R. An Axiomatic Basis for Computer Programming. Commun. ACM., 1969, vol. 12, no. 10, pp. 576–580.

[13]. Robert W. Floyd. Assigning meanings to programs. In Proceedings of the Symposium in Applied Mathematics, vol. XIX, pp. 19–32. American Mathematical Society, April 1967.

[14]. Edsger W. Dijkstra. Guarded commands, nondeterminacy, and formal derivation of programs. Communications of the ACM. 1975, vol. 18, no. 8, pp. 453–457.

[15]. John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science. LICS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 55–74.
[16]. François Bobot, Jean-Christophe Filliâtre, Claude Marché, Andrei Paskevich. Why3: Shepherd Your Herd of Provers. Boogie 2011: First International Workshop on Intermediate Verification Languages. Wrocław, Poland. 2011, pp. 53 – 64.
[17]. K. Rustan M. Leino. Dafny: an automatic program verifier for functional correctness. In Proceedings of the 16th international conference on Logic for programming, artificial intelligence, and reasoning (LPAR'10). Edmund M. Clarke, Andrei Voronkov (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 348–370, 2010.
[18]. Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, Alwen Tiu. Expressiveness + Automation + Soundness: Towards Combining SMT Solvers and Interactive Proof Assistants. International Conference on Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2006, LNCS, vol. 3920, pp. 167–181. Berlin, Heidelberg: Springer-Verlag, 2006.
[19]. Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Thery, Benjamin Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. Proceedings of the First International Conference on Certified Programs and Proofs. CPP'11. Berlin, Heidelberg: Springer-Verlag, pp. 135–150, 2011.
[20]. Benjamin C. Pierce. Types and Programming Languages. Chapter I. Untyped Systems. The MIT Press, 2002.
[21]. Robert D. Tennent. The denotational semantics of programming languages. Commun. ACM 19, vol. 8 (August 1976), pp. 437–453, 1976.
[22]. Мандрыкин М. У. Моделирование памяти Си-программ для инструментов статической верификации на основе SMT-решателей. Диссертация на соискание ученой степени кандидата физико-математических наук. Институт системного программирования РАН, октябрь, 2016.
[23]. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar. Lazy abstraction. Symposium on Principles of Programming Languages, pp. 58–70, ACM Press, 2002.
[24]. Pavel Shved, Mikhail Mandrykin, Vadim Mutilin. Predicate Analysis with BLAST 2.7. Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems 2012 (TACAS'12), LNCS, vol. 7214,. pp. 525–527.
[25]. Pavel Shved, Vadim S. Mutilin, Mikhail U. Mandrykin. Experience of improving the Blast static verification tool. Programming and Computer Software, vol. 38, issue 3, pp. 134–142, June 2012. DOI: 10.1134/S0361768812030061.
[26]. Мутилин В.С. Верификация драйверов операционной системы Linux при помощи предикатных абстракций. Диссертация на соискание ученой степени кандидата физико-математических наук. Институт системного программирования РАН. 2012. ноябрь.
[27]. Lars O. Andersen. Program Analysis and Specialization for the C Programming Language. PhD thesis, University of Copenhagen, DIKU, 1994.
[28]. Marc Berndl, Ondřej Lhoták, Feng Qian, Laurie Hendren, Navindra Umanee. Points-to Analysis using BDDs. SIGPLAN Notes, vol. 38, no. 5, pp. 103–114. ACM. New York, NY, USA. May 2003.
[29]. Edsger W. Dijkstra, Carel S. Schölten. Predicate Calculus and Program Semantics. The strongest postcondition, pp. 209–215. Springer New York. 1990.

[30]. Alexey Khoroshilov, Vadim Mutilin, Eugene Novikov, Pavel Shved, Alexander Strakh. Towards An Open Framework for C Verification Tools Benchmarking. Proceedings of the 8th international conference on Perspectives of System Informatics (PSI'11), LNCS, vol. 7162, pp. 179–192, 2011.

[31]. Thomas Ball, Ella Bounimova, Vladimir Levin, Leonardo de Moura. Efficient evaluation of pointer predicates with Z3 SMT Solver in SLAM2. March 2010. MSR-TR-2010-24.

[32]. Thomas Ball, Ella Bounimova, Rahul Kumar, Vladimir Levin. SLAM2: Static driver verification with under 4% false alarms. Formal Methods in Computer-Aided Design (FMCAD), 2010, pp. 35–42.

[33]. Thomas Ball, Todd Millstein, Sriram K. Rajamani. Polymorphic Predicate Abstraction. ACM Trans. Program. Lang. Syst., vol. 27, no. 2, March 2005, pp. 314–343. ACM, New York, NY, USA. 2005.

[34]. Thomas Ball, Ella Bounimova, Byron Cook, Vladimir Levin, Jakob Lichtenberg, Con McGarvey, Bohus Ondrusek, Sriram K. Rajamani, Abdullah Ustuner. Thorough Static Analysis of Device Drivers. Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006. (EuroSys '06) Leuven, Belgium, pp. 73–85. ACM. New York, NY, USA. 2006.

[35]. Ernie Cohen, Markus Dahlweid, Mark Hillebrand, Dirk Leinenbach, Michał Moskal, Thomas Santen, Wolfram Schulte, Stephan Tobies. VCC: A Practical System for Verifying Concurrent C. Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs '09, pp. 23–42. Springer-Verlag. Berlin, Heidelberg. 2009.

[36]. Ernie Cohen, Michał Moskal, Stephan Tobies, Wolfram Schulte. A Precise Yet Efficient Memory Model For C. Electron. Notes Theor. Comput. Sci., October, 2009, vol. 254, pp. 85–103. Elsevier Science Publishers B. V., Amsterdam, The Netherlands. 2009.

[37]. Gordon D. Plotkin. A structural approach to operational semantics. Computer Science Department, Aarhus University. Aarhus, Denmark. Internal Report DAIMI FN-19. 1981.

[38]. Мандрыкин М. У., Хорошилов А. В. Анализ регионов для дедуктивной верификации Си-программ. Программирование, 2016, № 5, стр. 3–29.

[39]. Rodney M. Burstall. Some techniques for proving correctness of programs which alter data structures. Machine intelligence, vol. 7, no. 3, pp. 23–50, 1972.

[40]. Richard Bornat. Proving Pointer Programs in Hoare Logic. In Proceedings of the 5th International Conference on Mathematics of Program Construction (MPC '00), pp. 102–126. Springer-Verlag. London, UK, 2000.

[41]. Thierry Hubert, Claude Marché. Separation Analysis for Deductive Verification. Heap Analysis and Verification (HAV'07), Braga, Portugal, March 2007, pp. 81–93.

[42]. Yannick Moy. Union and Cast in Deductive Verification. Proceedings of the C/C++ Verification Workshop. Technical Report ICIS-R07015, pp. 1–16. Radboud University Nijmegen, July, 2007.

[43]. Мандрыкин М.У., Хорошилов А.В. Высокоуровневая модель памяти промежуточного языка Jessie с поддержкой произвольного приведения типов указателей. Программирование, 2015, т. 41, № 4, стр. 23–29.

[44]. Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, Boris Yakobowski. Frama-C: A Software Analysis Perspective. Proceedings of the 10th International Conference on Software Engineering and Formal Methods. SEFM'12.

Thessaloniki, Greece. 2012. Lecture Notes in Computer Science, vol. 7504, pp. 233–247. Springer-Verlag. Berlin, Heidelberg. 2012.
[45]. Yannick Moy. Automatic Modular Static Safety Checking for C Programs. PhD thesis. Université Paris-Sud. January, 2009.
[46]. Jean-Christophe Filliâtre, Claude Marché. Multi-prover Verification of C Programs. In Proceedings of Formal Methods and Software Engineering: 6th International Conference on Formal Engineering Methods, ICFEM 2004, Seattle, WA, USA, November 8–12, 2004. LNCS, vol. 3308, pp. 15–29. Springer Berlin Heidelberg. 2004.
[47]. Robin Milner. A theory of type polymorphism in programming. Journal of Computer and System Sciences, vol. 17, no. 3, pp. 348–375. December, 1978.
[48]. Jean-Christophe Filliâtre, Andrei Paskevich. Why3: Where Programs Meet Provers. Proceedings of the 22Nd European Conference on Programming Languages and Systems. ESOP'13. Rome, Italy. LNCS, vol. 7792, pp. 125–128. Springer-Verlag.
[49]. https://www.microsoft.com/en-us/research/project/havoc/
[50]. Shuvendu Lahiri, Shaz Qadeer. Back to the future: revisiting precise program verification using SMT solvers. In Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '08), pp. 171–182. ACM, New York, NY, USA. 2008.
[51]. Aharon Abadi, Alexander Rabinovich, Mooly Sagiv. Decidable fragments of many-sorted logic. Journal of Symbolic Computation, vol. 45, issue 2, pp. 153–172, February 2010.
[52]. Yeting Ge, Leonardo Moura. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In Proceedings of the 21st International Conference on Computer Aided Verification (CAV '09). LNCS, vol. 5643, pp. 306–320. Springer-Verlag, Berlin, Heidelberg, 2009.

Survey of memory modeling methods in static verification tools

M.U. Mandrykin <mandrykin@ispras.ru>
V.S. Mutilin <mutilin@ispras.ru>
ISP RAS, 25 Alexander Solzhenitsyn Str., Moscow, 109004, Russian Federation

Abstract. The paper presents a survey of existing approaches to modeling memory states of C programs with SMT-formulas in context of static verification. The paper highlights the essential problems of C memory model development and describes two major groups of C memory models: one comprising of models that support unbounded memory regions and another including the models that don't. Among the models for a priori bounded memory regions the paper discusses a strongest postcondition-based model relying on preliminary alias analysis and a weakest precondition-based model that uses uninterpreted functions and first-order logic to represent pointer predicates. Among the models for unbounded memory areas the paper describes a typed memory model, the Burstall-Bornat model, a region-based model and a model with full support for the Logic of Interpreted Sets and Bounded Quantification (LISBQ) earlier implemented in the HAVOC deductive verification tool.

Keywords: static verification; memory models; SMT-solvers.

DOI: 10.15514/ISPRAS-2017-29(1)-12

For citation: Mandrykin M.U., Mutilin V.S. Survey of memory modeling methods in static verification tools. *Trudy ISP RAN / Proc. ISP RAS*, vol. 29, issue 1, 2017, pp. 195-230 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-12

References

[1]. Leonardo de Moura, Nikolaj Bjørner. Satisfiability modulo theories: An appetizer. In Formal Methods: Foundations and Applications, 12th Brazilian Symposium on Formal Methods (SBMF 2009). Gramado, Brazil, August 19–21, 2009. Revised Selected Papers, Lecture Notes in Computer Science, vol. 5902, pp. 23–36. Springer, 2009.

[2]. Sascha Böhme, Michał Moskal. Heaps and Data Structures: A Challenge for Automated Provers. In Nikolaj Børner, Viorica Sofronie-Stokkermans, eds. Automated Deduction. Lecture Notes in Computer Science, vol. 6803, pp. 177–191. Springer, 2011.

[3]. Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, Helmut Veith. Counterexample-Guided Abstraction Refinement. In Proceedings of the 12th International Conference on Computer Aided Verification (CAV '00), E. Allen Emerson, A. Prasad Sistla (Eds.). Springer-Verlag, London, UK, pp. 154–169, 2000.

[4]. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In Proc. of the Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99), LNCS, vol. 1579, pp. 193–207. Springer-Verlag, 1999.

[5]. Dirk Beyer, Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar. The software model checker Blast: Applications to software engineering. International Journal on Software Tools for Technology Transfer, October 2007, vol. 9, issue 5, pp. 505–525.

[6]. Mary Sheeran, Satnam Singh, Gunnar Stålmarck. Checking Safety Properties Using Induction and a SAT-Solver. Formal Methods in Computer-Aided Design. Lecture Notes in Computer Science, vol. 1954, pp. 127–144.

[7]. Susanne Graf, Hassen Saïdi. Construction of abstract state graphs with PVS. Proceedings of International Conference on Computer Aided Verification, 1997. LNCS, vol. 1254, pp. 72–83. Springer, Berlin Heidelberg, 1997.

[8]. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, Kenneth L. McMillan. Abstractions from proofs. In Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of Programming Languages (POPL '04). SIGPLAN Notes, vol. 39, no. 1, pp. 232–244. ACM, New York, NY, USA. 2004.

[9]. Aaron R. Bradley. SAT-based model checking without unrolling. In Proceedings of the 12th International conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'11), Ranjit Jhala, David Schmidt (Eds.), Springer-Verlag, Berlin, Heidelberg, pp. 70–87, 2011.

[10]. Corina S. Păsăreanu, Willem Visser, David Bushnell, Jaco Geldenhuys, Peter Mehlitz, Neha Rungta. Symbolic PathFinder: integrating symbolic execution with model checking for Java bytecode analysis. Automated Software Engineering, 2013, vol. 20, no. 3, pp. 391–425.

[11]. V. K. Koshelev, V. N. Ignatyev, A. I. Borzilov. C# static analysis framework. Trudy ISP RAN / Proc. ISP RAS. vol. 28, issue 1, 2016, pp. 21-40 (*In Russian*). DOI: 10.15514/ISPRAS-2016-28(1)-2.

[12]. Hoare C. A. R. An Axiomatic Basis for Computer Programming. Commun. ACM., 1969, vol. 12, no. 10, pp. 576–580.

[13]. Robert W. Floyd. Assigning meanings to programs. In Proceedings of the Symposium in Applied Mathematics, vol. XIX, pp. 19–32. American Mathematical Society, April 1967.

[14]. Edsger W. Dijkstra. Guarded commands, nondeterminacy, and formal derivation of programs. Communications of the ACM. 1975, vol. 18, no. 8, pp. 453–457.

[15]. John C. Reynolds. Separation Logic: A Logic for Shared Mutable Data Structures. Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science. LICS '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 55–74.

[16]. François Bobot, Jean-Christophe Filliâtre, Claude Marché, Andrei Paskevich. Why3: Shepherd Your Herd of Provers. Boogie 2011: First International Workshop on Intermediate Verification Languages. Wrocław, Poland. 2011, pp. 53 – 64.

[17]. K. Rustan M. Leino. Dafny: an automatic program verifier for functional correctness. In Proceedings of the 16th international conference on Logic for programming, artificial intelligence, and reasoning (LPAR'10). Edmund M. Clarke, Andrei Voronkov (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 348–370, 2010.

[18]. Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, Alwen Tiu. Expressiveness + Automation + Soundness: Towards Combining SMT Solvers and Interactive Proof Assistants. International Conference on Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2006, LNCS, vol. 3920, pp. 167–181. Berlin, Heidelberg: Springer-Verlag, 2006.

[19]. Michaël Armand, Germain Faure, Benjamin Grégoire, Chantal Keller, Laurent Thery, Benjamin Werner. A Modular Integration of SAT/SMT Solvers to Coq through Proof Witnesses. Proceedings of the First International Conference on Certified Programs and Proofs. CPP'11. Berlin, Heidelberg: Springer-Verlag, pp. 135–150, 2011.

[20]. Benjamin C. Pierce. Types and Programming Languages. Chapter I. Untyped Systems. The MIT Press, 2002.

[21]. Robert D. Tennent. The denotational semantics of programming languages. Commun. ACM 19, vol. 8 (August 1976), pp. 437–453, 1976.

[22]. Mandrykin M. U. Modeling memory of C programs in SMT-based static verification tools. PhD Thesis. ISP RAS, October 2016 (*In Russian*).

[23]. Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar. Lazy abstraction. Symposium on Principles of Programming Languages, pp. 58–70, ACM Press, 2002.

[24]. Pavel Shved, Mikhail Mandrykin, Vadim Mutilin. Predicate Analysis with BLAST 2.7. Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems 2012 (TACAS'12), LNCS, vol. 7214,. pp. 525–527.

[25]. Pavel Shved, Vadim S. Mutilin, Mikhail U. Mandrykin. Experience of improving the Blast static verification tool. Programming and Computer Software, vol. 38, issue 3, pp. 134–142, June 2012. DOI: 10.1134/S0361768812030061..

[26]. Mutilin V. S. Linux driver verification using predicate abstraction. PhD Thesis. ISP RAS, November 2012 (*In Russian*).

[27]. Lars O. Andersen. Program Analysis and Specialization for the C Programming Language. PhD thesis, University of Copenhagen, DIKU, 1994.
[28]. Marc Berndl, Ondřej Lhoták, Feng Qian, Laurie Hendren, Navindra Umanee. Points-to Analysis using BDDs. SIGPLAN Notes, vol. 38, no. 5, pp. 103–114. ACM. New York, NY, USA. May 2003.
[29]. Edsger W. Dijkstra, Carel S. Schölten. Predicate Calculus and Program Semantics. The strongest postcondition, pp. 209–215. Springer New York. 1990.
[30]. Alexey Khoroshilov, Vadim Mutilin, Eugene Novikov, Pavel Shved, Alexander Strakh. Towards An Open Framework for C Verification Tools Benchmarking. Proceedings of the 8th international conference on Perspectives of System Informatics (PSI'11), LNCS, vol. 7162, pp. 179–192, 2011.
[31]. Thomas Ball, Ella Bounimova, Vladimir Levin, Leonardo de Moura. Efficient evaluation of pointer predicates with Z3 SMT Solver in SLAM2. March 2010. MSR-TR-2010-24.
[32]. Thomas Ball, Ella Bounimova, Rahul Kumar, Vladimir Levin. SLAM2: Static driver verification with under 4% false alarms. Formal Methods in Computer-Aided Design (FMCAD), 2010, pp. 35–42.
[33]. Thomas Ball, Todd Millstein, Sriram K. Rajamani. Polymorphic Predicate Abstraction. ACM Trans. Program. Lang. Syst., vol. 27, no. 2, March 2005, pp. 314–343. ACM, New York, NY, USA. 2005.
[34]. Thomas Ball, Ella Bounimova, Byron Cook, Vladimir Levin, Jakob Lichtenberg, Con McGarvey, Bohus Ondrusek, Sriram K. Rajamani, Abdullah Ustuner. Thorough Static Analysis of Device Drivers. Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006. (EuroSys '06) Leuven, Belgium, pp. 73–85. ACM. New York, NY, USA. 2006.
[35]. Ernie Cohen, Markus Dahlweid, Mark Hillebrand, Dirk Leinenbach, Michał Moskal, Thomas Santen, Wolfram Schulte, Stephan Tobies. VCC: A Practical System for Verifying Concurrent C. Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs '09, pp. 23–42. Springer-Verlag. Berlin, Heidelberg. 2009.
[36]. Ernie Cohen, Michał Moskal, Stephan Tobies, Wolfram Schulte. A Precise Yet Efficient Memory Model For C. Electron. Notes Theor. Comput. Sci., October, 2009, vol. 254, pp. 85–103. Elsevier Science Publishers B. V., Amsterdam, The Netherlands. 2009.
[37]. Gordon D. Plotkin. A structural approach to operational semantics. Computer Science Department, Aarhus University. Aarhus, Denmark. Internal Report DAIMI FN-19. 1981.
[38]. M. U. Mandrykin, A. V. Khoroshilov. Region analysis for deductive verification of C programs. Programming and Computer Software, 2016, vol. 42, no. 5, pp. 257–278. DOI: 10.1134/S0361768816050042.
[39]. Rodney M. Burstall. Some techniques for proving correctness of programs which alter data structures. Machine intelligence, vol. 7, no. 3, pp. 23–50, 1972.
[40]. Richard Bornat. Proving Pointer Programs in Hoare Logic. In Proceedings of the 5th International Conference on Mathematics of Program Construction (MPC '00), pp. 102–126. Springer-Verlag. London, UK, 2000.
[41]. Thierry Hubert, Claude Marché. Separation Analysis for Deductive Verification. Heap Analysis and Verification (HAV'07), Braga, Portugal, March 2007, pp. 81–93.

[42]. Yannick Moy. Union and Cast in Deductive Verification. Proceedings of the C/C++ Verification Workshop. Technical Report ICIS-R07015, pp. 1–16. Radboud University Nijmegen, July, 2007.
[43]. M. U. Mandrykin, A. V. Khoroshilov. High-level memory model with low-level pointer cast support for Jessie intermediate language. Programming and Computer Software, 2015, vol. 41,. no. 4,. pp. 197–207. DOI: 10.1134/S0361768815040040.
[44]. Pascal Cuoq, Florent Kirchner, Nikolai Kosmatov, Virgile Prevosto, Julien Signoles, Boris Yakobowski. Frama-C: A Software Analysis Perspective. Proceedings of the 10th International Conference on Software Engineering and Formal Methods. SEFM'12. Thessaloniki, Greece. 2012. Lecture Notes in Computer Science, vol. 7504, pp. 233–247. Springer-Verlag. Berlin, Heidelberg. 2012.
[45]. Yannick Moy. Automatic Modular Static Safety Checking for C Programs. PhD thesis. Université Paris-Sud. January, 2009.
[46]. Jean-Christophe Filliâtre, Claude Marché. Multi-prover Verification of C Programs. In Proceedings of Formal Methods and Software Engineering: 6th International Conference on Formal Engineering Methods, ICFEM 2004, Seattle, WA, USA, November 8–12, 2004. LNCS, vol. 3308, pp. 15–29. Springer Berlin Heidelberg. 2004.
[47]. Robin Milner. A theory of type polymorphism in programming. Journal of Computer and System Sciences, vol. 17, no. 3, pp. 348–375. December, 1978.
[48]. Jean-Christophe Filliâtre, Andrei Paskevich. Why3: Where Programs Meet Provers. Proceedings of the 22Nd European Conference on Programming Languages and Systems. ESOP'13. Rome, Italy. LNCS, vol. 7792, pp. 125–128. Springer-Verlag.
[49]. https://www.microsoft.com/en-us/research/project/havoc/
[50]. Shuvendu Lahiri, Shaz Qadeer. Back to the future: revisiting precise program verification using SMT solvers. In Proceedings of the 35th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL '08), pp. 171–182. ACM, New York, NY, USA. 2008.
[51]. Aharon Abadi, Alexander Rabinovich, Mooly Sagiv. Decidable fragments of many-sorted logic. Journal of Symbolic Computation, vol. 45, issue 2, pp. 153–172, February 2010.
[52]. Yeting Ge, Leonardo Moura. Complete Instantiation for Quantified Formulas in Satisfiabiliby Modulo Theories. In Proceedings of the 21st International Conference on Computer Aided Verification (CAV '09). LNCS, vol. 5643, pp. 306–320. Springer-Verlag, Berlin, Heidelberg, 2009.

Обзор состояния области потоковой обработки данных

Р.С. Самарев <samarev@acm.org>
Московский государственный технический университет имени Н.Э. Баумана
105005, Москва, 2-я Бауманская ул., д. 5, стр. 1

Аннотация. В статье рассматриваются аспекты построения и использования современных фреймворков для организации потоковой обработки данных. Уделяется внимание архитектурным аспектам фреймворков, а также связанными с ними достоинствами и недостатками. Рассматривается проблема объективного оценивания характеристик потоковых фреймворков.

Ключевые слова: большие данные; тестирование; storm; spark; flink; samza

DOI: 10.15514/ISPRAS-2016-29(1)-13

Для цитирования: Самарев Р.С. Обзор состояния области потоковой обработки данных. Труды ИСП РАН, том 29, вып. 1, 2017 г., стр. 231-260. DOI: 10.15514/ISPRAS-2017-29(1)-13

1. Введение

Термин "потоковая обработка данных" (Stream processing) подразумевает обработку любых данных как потока, однако сейчас под ним принято понимать скорее обработку некоторых крупных пакетов данных, таких как некоторое сообщения, но не обработку аудио и видео данных. В этой статье мы будем говорить именно о состоянии области обработки больших данных как потоков, то есть об их обработке с минимальной задержкой.

Идеи потоковой обработки данных совершенно не новы. В 80-е годы они появились под терминами Dataflow processing, Dataflow database machine и пр. Однако в близком к современному пониманию первые работы в этой области стали появляться в конце 90-х – начале 2000-х годов на волне развития Интернет, появления идей Интернета вещей как итог понимания того, что традиционный способ обработки данных через сохранение в статичную базу данных не подходит для обработки постоянно обновляемых данных. Интересен один из обзоров в этой области, который был представлен в 2003-м году в работе [26]. Из числа первых потоковых процессоров и фреймворков (будем называть так программные продукты, которые предназначены для

потоковой обработки данных) Aurora, Borealis, COUGAR, Gigascop, NiagaraCQ, OpenCQ, StatStream, STREAM, TelegraphCQ, Tribeca ни один не получил коммерческого развития. (Аббревиатура CQ в названии проектов - это Continuous Query.) Более подробно см. в [19].

Чуть более подробно рассмотрим академический проект Aurora [16], который включал в себя довольно много новшеств для своего времени. Этот продукт позволял описать процесс непрерывной обработки данных, включая несколько параллельных входов и выходов, поддерживалась возможность оперативного накопления данных в локальном хранилище для их обработки. Модель программирования - визуальная, в терминах алгебры SQuAl ([S]tream [Qu]ery [Al]gebra). Поддерживались основные операции filter, map, union, sort, aggregate, join, resample. Описание процесса обработки данных представляет собой последовательноть блоков-операций и стрелок, означающих передачу данных, что образует направленный ациклический граф (DAG). Итогом этого проекта стала работа [40], в которой М. Стойнбрейкер, У. Гетинтемел и С. Здоник сформулировали требования к системам потоковой обработки реального времени.

Приведём эти требования, сформулированные в виде правил:

1. Сохраняйте данные движущимися.
2. Формулируйте запросы с использованием SQL на потоках (StreamSQL).
3. Справляйтесь с дефектностью потоков (задержка, отсутствие и нарушение порядка данных).
4. Генерируйте предсказуемые результаты.
5. Интегрируйте хранимые и потоковые данные.
6. Гарантируйте безопасность и доступность данных.
7. Автоматически разделяйте и масштабируйте приложения.
8. Мгновенно обрабатывайте и выдавайте результаты.

Как видим, несмотря на более чем десятилетний возраст публикации, эти требования ещё актуальны, хотя и с некоторыми оговорками, и не в полной мере реализуются в современных средствах потоковой обработки. Не все выполняют требование 2. В ряде случае проблемы с требованием 5, часто имеются проблемы с требованиями 6-8.

Близкой к области потоковой обработки данных является так называемая сложная обработка событий – CEP (Complex Event Processing). Она заключается в том, что на основании заданных правил обработки сообщений и их последовательности выявляются и обрабатываются некие события. Это направление традиционно отличается собственными языками написания правил, например, Drools [7]. Условно CEP отличают от потоковой обработки тем, что CEP обычно не являются массово параллельными системами обработки, они обеспечивают минимальную задержку времени обработки,

отсутствует необходимость программировать на традиционных языках программирования (часто достаточно SQL или событийного языка типа Drools) [19]. Также в CEP очень часто не требуется гарантированная обработка всех сообщений и в случае сбоя часть их можно отбросить. Однако принципиально CEP – это та же обработка потока сообщений. И, более того, современные потоковые фреймворки типа Apache Flink [2], Apache Kafka Streams [29] претендуют на возможность их использования в качестве средств CEP.

2. Типовая схема приёма и обработки потоковых данных

В данной статье рассматриваются, прежде всего, доступные программные продукты с открытым исходным кодом. Поэтому приводимые здесь схемы будут относиться именно к ним.

Одно из основных требований при обработке потоков – это возможность принять тот поток, который поступает на вход. Отметим, что в зависимости от решаемых задач, этот поток может быть постоянным или сильно изменяющимся во времени, предсказуемым или содержащим резкие всплески, на которые необходимо корректно реагировать. Следует отличать потоки от погодных сенсоров и, например, пиков активности пользователей сетевых игр, информацию о которых нам надо собрать.

Сложившаяся схема обработки выглядит примерно так, как это представлено на рис. 1.

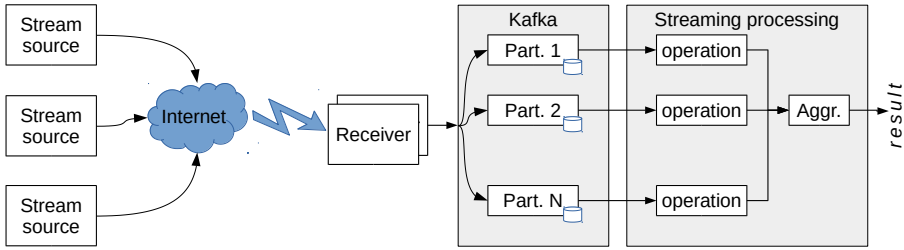


Рис. 1: Типовой пример сбора и обработки сообщений

Fig. 1: Typical process of data gathering and processing

Входные потоки, образованные внешними источниками, необходимо принять и упорядочить. Чаще всего для этого используют программный продукт Apache Kafka [3]. Однако заметим, что сейчас это правило уже может быть поставлено под сомнение, поскольку другие потоковые фреймворки решают сходные задачи самостоятельно. Основное назначение Apache Kafka – принять входные сообщения и гарантировать их доставку потребителям. Сообщения помещаются в очередь (topic), которая может быть сохранена в промежуточную БД, что гарантирует последующее получение данных потребителями, которые сейчас не активны. По умолчанию в качестве СУБД для промежуточной БД используется RocksDB. Дополнительно Kafka может

разложить сообщения по узлам кластера в соответствии с использованной функцией размещения. Это позволяет равномерно распределить дальнейшую вычислительную нагрузку. Поэтому следующий этап – использование соответствующего потокового фреймворка, который обеспечит размещение задач обработки данных, их координацию и чтение из очереди сообщений Kafka. Подробнее см. [28].

3. Основные различия потоковых фреймворков

Массовой проблема использования потоковых фреймворков стала лишь в последние годы. В отношении того, как должен выглядеть современный потоковый фреймворк, существует множество мнений. Приходится наблюдать, что то, о чём писали буквально год-два назад, уже стало устаревшим, а выглядевшие год назад истинными утверждения стали ложными.

Современное понимание потоковых процессоров изложено в статьях [37, 38]. На первом месте выделяют различия в обработки данных и модели программирования.

3.1 Модель обработки данных

С точки зрения модели обработки данных выделяют системы естественной потоковой обработки, в которых исходные сообщения поступают на обработку строго последовательно. К этой группе относятся продукты Apache Storm [6], Apache Samza [4], Apache Flink [2]. Другой вариант обработки – пакетная (micro-batches). Она отличается тем, что перед тем как попасть на обработку, сообщения группируются в пакеты. К этой группе относятся Apache Storm/Trident и Apache Spark [5].

Ещё год назад публиковались статьи [34], где пакетная модель обработки демонстрировалась как достоинство. Проблема же заключается в особенности реализации конкретного фреймворка, и ответ на то, какая модель является предпочтительной, не является очевидным. Например, по состоянию на 2016 год, программы, использующие только Apache Storm, работают существенно быстрее, чем использующие его пакетную надстройку Apache Trident. Однако Apache Spark с пакетной моделью демонстрирует существенно большую производительность, чем Apache Storm, но также существенно проигрывает Apache Flink с естественной потоковой моделью обработки.

3.2 Модель программирования

Следующим аспектом использования фреймворков является модель программирования. Различают композиционную и декларативную модели. Композиционная модель подразумевает объединение элементарных операторов и источников данных в некоторую сетевую "топологию", то есть фактическое соединение создание элементов обработки и соединение их входов и выходов. Этой модели придерживаются Apache Storm и Apache Flume [12]. Декларативная модель в современном понимании предполагает

использование высокоуровневых операций, предписывающих определённый тип обработки данных. По сути, это напоминает функциональный стиль программирования. В следующем листинге приведён пример программы вычисления количества слов, реализованный с помощью Apache Flink.

```
StreamExecutionEnvironment env =
    StreamExecutionEnvironment
        .getExecutionEnvironment( );
DataStream<Tuple2<String,Integer>> dataStream=
     env.socketTextStream ("localhost", 9999)
        .flatMap (new Splitter ( ) )
        .keyBy( 0 )
        .timeWindow( Time.seconds ( 5 ) )
        .sum( 1 );
dataStream.print( );
env.execute ( "example")
```

Модель называется декларативной, потому что на момент выполнения этого кода, по сути, лишь выстраивается схема обработки потока, но ещё не происходит какой-либо реальной обработки данных. Это позволяет делать описание модели в общем-то на любом языке программирования при условии, что будут вызваны соответствующие методы класса, конструирующего этот логический план обработки. Это может быть специализированный язык предметной области или, например, может использоваться аналогичный код, написанный на языке Ruby и выполняемый в окружении JRuby с подключением соответствующих библиотек фреймворка. Или же может быть реализован транслятор xml-спецификаций по аналогии с тем, как организована модель спецификаций соединения модулей в Spring Framework [10]. Ограничением является лишь наличие соответствующих операторов обработки в терминах классов Java, если говорить о фреймворках Apache Flink или Apache Spark. В примере таким классом является класс Splitter, реализация которого не приводится. С использованием этого интерфейса могут быть реализовано и выполнение SQL-запросов. Также отметим тенденцию последних лет – создание специальных средств типа проекта Emma [18] или Apache Beam, назначением которых является автоматическая трансляция в вызовы именно того фреймворка, который выбрал пользователь-программист в данный момент.

Следует добавить несколько слов относительно изменений в понимании того, какие языки должны быть использованы для описания процесса обработки данных. Как уже упоминалось ранее [19], развитие потоковых фреймворков началось как ветвь области СУБД, поэтому изначально описание процесса обработки данных рассматривалось только с использованием SQL. Дальнейшее развитие, совпавшее с переходом от консольного интерфейса пользователя к графическому и соответствующей эволюцией идей программирования на уровне схем, привело к появлению проектов типа Aurora [16], где схема обработки была именно графической.

В 2000-е годы можно видеть массовый переход к графическим методам программирования. Это и появление языка BPMN, представляющего собой графические схемы бизнес-процесса, и массовое использование его модификаций в средствах ESB и ETL, и появление средств типа KNIME, которые позволяют описать процесс обработки данных как графическую схему. Очевидно, что использование графического языка программирования является привлекательной альтернативой SQL. Безусловно, нельзя забывать о языке описания правил типа Drools, однако его использование не было массовым.

Конец этого периода характеризуется всплеском интереса к функциональным языка программирования и к функциональному стилю программирования как таковому. Существенное влияние оказала популярность платформы JVM, на которой созданы такие языки, как Scala и Clojure. Сейчас функциональный стиль реализован и в самом языке Java. Вероятно, поэтому основной способ программирования для потоковых фреймворков – декларативный, в функциональном стиле. Отметим, что попытки вернуться к SQL были актуальны всё время, поэтому разработчики современных потоковых фреймворков заявляют о таких возможностях. Это относится к Apache Flink, Apache Spark, Apache Kafka Streams. Реально же всё это требует отдельной оценки.

В настоящий момент можно заключить, что именно декларативное программирование в функциональном стиле с использованием языка Java 8 является наиболее распространённой практикой при создании приложений с использованием потоковых фреймворков. Остальные языки программирования являются скорее нишевыми. Всплеск популярности языков типа Scala и Clojure следует считать временным явлением. Если пару лет назад Apache Spark позиционировался как фреймворк на Scala для Scala-программистов, то сейчас существенная часть его интерфейсного кода продублирована на Java для того, чтобы не терять клиентов и иметь возможность привлекать разработчиков, желающих использовать Java 8. Apache Storm пытались реализовывать на языке Clojure, но сейчас принято решение продолжать развитие фреймворка на языке Java.

3.3 Гарантированность обработки сообщений и устойчивость к сбоям

Следующей особенностью потоковых фреймворков является метод гарантирования обработки сообщений. Это обусловлено их распределённой архитектурой и необходимостью согласования данных на разных узлах кластера, что напрямую влияет на производительность. Различают обработку "максимум один раз" (at most once), "по крайней мере один раз" (at least once) и "строго один раз" (exactly once).

"Максимум один раз" означает, что сообщение может быть доставлено ноль или 1 раз, то есть может быть потеряно. "По крайней мере один раз"

предполагает, что сообщение будет доставлено на обработку в любом случае, но может быть более 1 раза. Очевидный недостаток этого метода – необходимость учитывать возможность появления дубликатов и, соответственно, нагрузку на программиста по их учёту в своих алгоритмах. "Точно один раз" означает, что сообщение будет гарантированно доставлено, но при этом исключены дубликаты. Этот метод самый удобный для программиста, но и, очевидно, самый сложный и медленный в реализации потокового фреймворка.

Очевидным требованием к распределённой системе обработки данных является требование корректной обработки в случае выхода из строя одного или нескольких узлов кластера. Обзор нескольких методов восстановления приводится в [30].

Одним из простейших механизмов восстановления обладают Apache Storm и Twitter Heron [13]. Этот механизм основан на отправке подтверждения каждым следующим оператором предыдущему. В терминологии Storm имеются источники данных (spout) и операторы обработки (bolt). Для каждой отправляемой записи (tuple) генерируется случайное 64-х битное число. После получения записи оператор обработки отправляет сообщение о получении. Если это сообщение не получено, исходная запись будет отправлена ещё раз. Следует заметить, что при этом существует проблема поддержки целостности данных в распределённой системе. Кроме того, подобный механизм является медленным и может порождать дубликаты [8].

Очевидной идеей ускорения является метод микропакетов, который использован в Apache Storm Trident и Apache Spark Streaming. Идея заключается в том, чтобы подтверждать не каждую запись, а небольшой пакет записей, если оператор выполнил их обработку (см. рис. 2). Это действительно позволяет существенно ускорить обработку и гарантировать доставку строго одного сообщения, однако создаёт следующие проблемы.

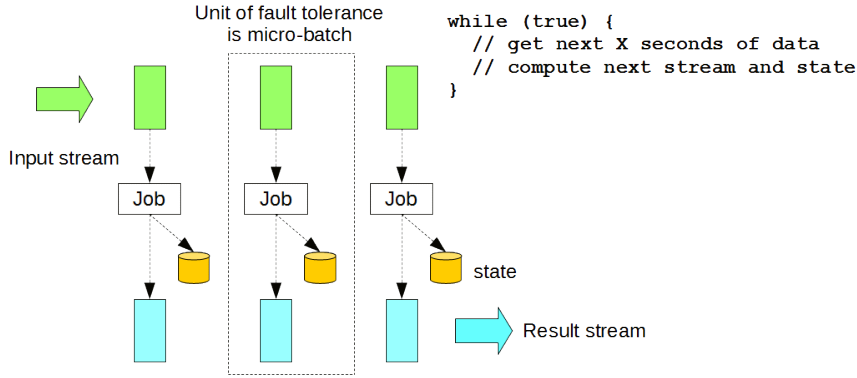


Рис. 2: Схема формирования контрольных точек с использованием микропакетов
Fig 2. Checkpointing of microbatches

В потоковых задачах часто требуется создание скользящего окна. Это окно может быть ограничено временем обработки, диапазоном времени сообщений или количеством сообщений. В случае же использования микропакетов нет возможности обрабатывать строго заданное количество сообщений. Другой серьёзный недостаток – сложность контроля темпа обработки в конвейере, образованном цепочкой операторов. То есть нет возможности контроля обратного давления данных. И последней проблемой является рост задержки обработки данных, поскольку подтверждение обработки будет получено лишь после обработки всего пакета. Подробнее см. [30].

Альтернативой микропакетам является использование транзакционного журнала. Этот подход применяется в Google Cloud Dataflow и Apache Samza. Основная идея – факт обработки сообщения фиксируется в журнале. В случае сбоя восстановление состояния возможно на основе записей в этом журнале. Такой подход позволяет решить большинство проблем предыдущих моделей, позволяя рассматривать поток сообщений и их обработку как непрерывный процесс, обеспечивая при этом высокую скорость обработки. Однако, например, в Apache Samza не решена проблема дубликатов сообщений.

В Apache Flink используется ещё одна модель восстановления, называемая авторами Asynchronous Barrier Snapshotting (ABS). Суть этой модели описана в [21]. Идея заключается в том, чтобы сохранять контрольные точки обработки сразу для последовательности сообщений. Имеются определённые точки снимков состояний – барьеры. В отличии от микропакетов, здесь процесс обработки не прерывается.

В момент объединения данных, получаемых от разных операторов, происходит так называемое выравнивание барьеров и запуск данных на обработку с формированием очередной метки-барьера в выходном потоке (см. рис. 3. Несмотря на то, что оператор может блокировать определённые входы до момента выравнивания барьеров, то есть появления методов-барьеров на всех входах, невозможна ситуация, когда оператор ничего не обрабатывает (хотя бы с одного входа данные поступают, иначе там уже присутствует метка-барьер), в отличии от микропакетов, где состояние простоя оператора вполне допустимо.

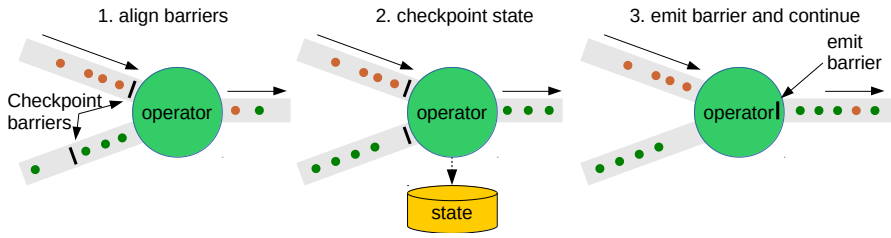


Рис. 3: Схема формирования контрольных точек Asynchronous Barrier Snapshotting

Fig. 3. Data processing with Asynchronous Barrier Snapshotting

3.4 Сохранение состояния и операции с окном данных

Для выполнения агрегационных запросов важно наличие возможности сохранение состояния со стороны приложения, выполняемого фреймворком. Под сохранением состояния здесь понимается возможность сохранять данные между обработкой сообщений, например, накопление данных в аккумуляторе для вычисления среднего значения. Другими словами, фреймворк без хранения состояний не может выполнить агрегацию данных.

Большая часть современных фреймворков поддерживает сохранение состояния, включая Apache Storm версии 1.0. Однако существуют различия в программировании накопления состояния, поскольку в зависимости от фреймворка это может быть локальное состояние данного оператора на данном узле, автоматически синхронизируемое глобальное состояние или явно разделяемая переменная, значение которой будет автоматически рассылаться по всем узлам кластера, где производится обработка.

В отношении оконных операций с данными следует отметить то, что в зависимости от фреймворка, поддерживается несколько типов окон: окна на фиксированное количество сообщений или на время. Время может быть системным или событийным, то есть отсчитываться на основе времени поступления сообщений в обработку или на основе времени, записанного внутри сообщений.

Также различают окна, скользящие с перекрытием данных, и последовательные окна обработки, где следующее окно начинается сразу после окончания предыдущего. Вариант окна с перекрытием позволяет, например, установить обработку данных за 1 минуту с выдачей результата раз в 5 секунд.

Как уже отмечалось ранее, реализуемость тех или иных типов окон зависит от реализованной во фреймворке модели гарантирования обработки сообщений.

Отметим, что в большинстве случаев хранение состояния технически реализуется с помощью встраиваемой СУБД RocksDB [9].

3.5 Распределение данных по узлам кластера и модель распараллеливания

Аспект разделения данных по узлам кластера и модель управления распараллеливанием операций является одним из существенных аспектов выбора фреймворка.

Следует сразу разделить фреймворки, способные выполнять операции распараллеливания автоматически, и фреймворки, не способные это делать.

К первой группе относится большинство потоковых фреймворков. Для примера на рис. 4 продемонстрирован процесс размещения и запуска приложения для Apache Flink. Необходимо выполнить команду размещения приложения на кластере под управлением уже запущенного Apache Flink, после чего построение физического плана выполнения, загрузка кода

операторов на соответствующих узлах, балансирование загрузки будут выполняться автоматически.

Ко второй группе можно отнести Apache Kafka Streams и Java Reactor. Особенность Apache Kafka Streams заключается в том, что приложение, написанное с помощью этого фреймворка, является автономным Java-приложением, выполняемом на одном узле (см. рис. 5). Фреймворк не предоставляет никаких средств передачи данных на уровне операций. Если требуется обмен данными с другими операторами, это означает, что данные следует выгрузить в очередь Kafka с необходимой функцией распределения данных, и должно быть написано ещё одно приложение, принимающие эти данные и продолжающее их обработку. То есть Kafka становится средством, обеспечивающим гарантированность обработки данных и их доставки по графу операторов, однако проблемой программиста является создание нужного количества приложений, их распределение по узлам кластера, а также запуск и мониторинг их состояния.

Отметим, что прямым конкурентом Apache Kafka Streams следует считать Java Reactor [15] и её подсистему Flux. Несмотря на то, что это средство не позиционируется как потоковый процессор, а создано как системный компонент Spring Framework и Java 9, Java Reactor обеспечивает выполнение типового набора операций над потоком сообщений, включая операции агрегации данных в окне.

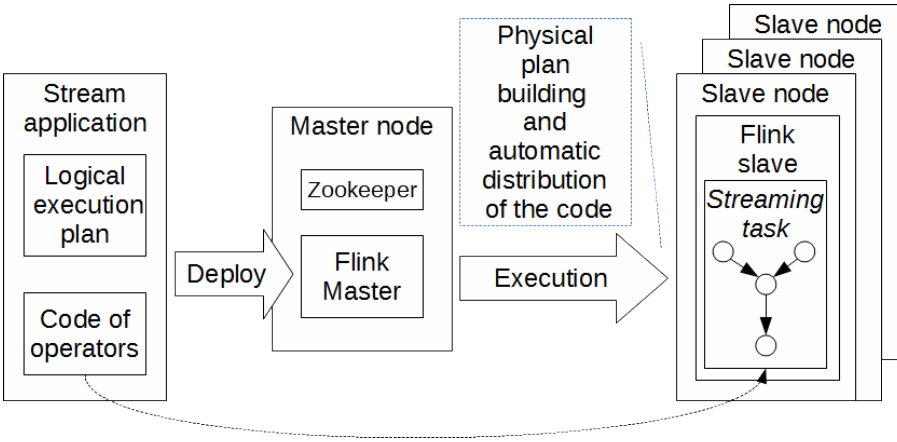


Рис. 4: Схема запуска приложения для Apache Flink
Fig. 4. Execution of an application with Apache Flink

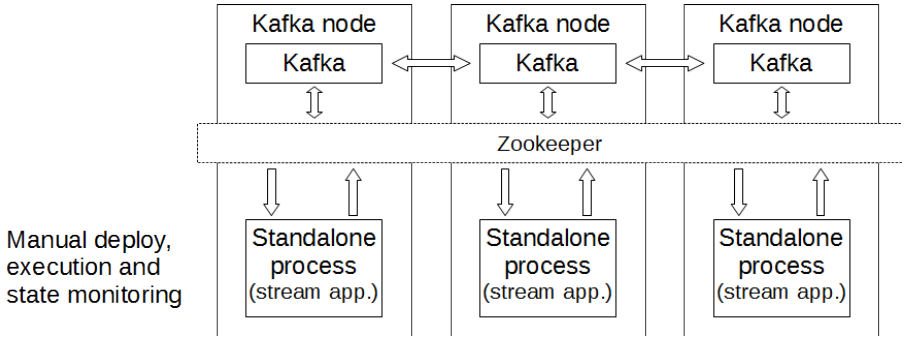


Рис. 5: Схема запуска приложения для Apache Kafka Streams
Fig. 5. Execution of an application with Apache Kafka Streams

4. Модели оценки характеристик потоковых фреймворков

Серьёзной проблемой выбора потоковых фреймворков в настоящее время является отсутствие единых и объективных критериев оценки производительности. Есть публикации, авторы которых проводят определённые сравнения, однако проблемой является узкая специализация и ограниченность применения этих тестов.

Одним из первых известных тестов производительности был так называемый линейный дорожный тест (the Linear Road benchmark) [20]. Тест имитирует дорожное движение в некотором абстрактном городе с параллельными дорогами. Каждый автомобиль раз в 30 секунд отправляет информацию о своём положении. Задачи потоковых фреймворков – определить текущий трафик на каждой из дорог, выявить аварии, учитывать количество активных полос движения, которое постоянно изменяется из-за аварий. Кроме того, требуется динамически подсчитывать затраты времени на проезд транспортными средствами по дорогам за заданный период времени. Проблемой применимости этого теста является то, что для каждого фреймворка необходима собственная реализация тестирующего приложения. Этот тест был разработан для оценки проекта Aurora и, несмотря на проработанность модели данных и критериев оценки, он так и не стал универсальным средством оценки производительности потоковых фреймворков.

В работе [31] рассматривается комбинированный тест StreamBench для оценки производительности и задержки обработки, в котором используются 7 программ тестирования с различной сложностью и различными операциями (фильтрация по образцу, поиск, извлечение поля, подсчёт количества слов и пр.). Проведено тестирование Apache Spark и Apache Storm. Этот тест также не получил развития, вероятно, потому, что исходные тексты не доступны.

В работе [25] представлен тест BigBench. Предполагается, что необходимо включать тестирование большого объема, различного типа данных и скорости обработки (volume, variety, velocity). Объем подразумевает петабайты данных,

различный тип – структурированные, слабо структурированные и неструктурированные данные, скорость обработки – требование "текучести данных" при решении задач ETL. Принципиальным отличием от предыдущих работ является то, что сделана попытка использования стандартизованных наборов тестов TPC-DS (TPC Benchmark DS: 'The' Benchmark Standard for decision support solutions including Big Data).

В работе [35] представлена модификация BigBench, в рамках которой сравниваются Apache Flink и Hive с использованием SQL-подобного языка HiveQL. Реализовано несколько запросов TPC-DS в терминах декларативной Java-модели Apache Flink и HiveQL. Широкого распространения эти тесты также не получили. В работе [39] представлено видение теста BigBench, как приближенного к решению конкретных бизнес-задач, включая потоковую обработку данных, машинного обучения, анализ графов, мультимедиа. Этот тест не доведён до реализации.

Обзор других тестов фреймворков больших данных, не ограниченный потоковыми фреймворками, приводится в работе [23].

Следует отметить, что несмотря на годы развития области потоковой обработки данных, сейчас нет ни готовых к использованию тестов, ни даже единых методик тестирования. Поэтому рассмотрим отдельные аспекты тестирования потоковых фреймворков.

Типовая схема тестирования приводится на рис. 6. Kafka(1) и Kafka(2) – входная и выходная очереди сообщений, которые в ряде случаев могут быть заменены самими потоковыми фреймворками.

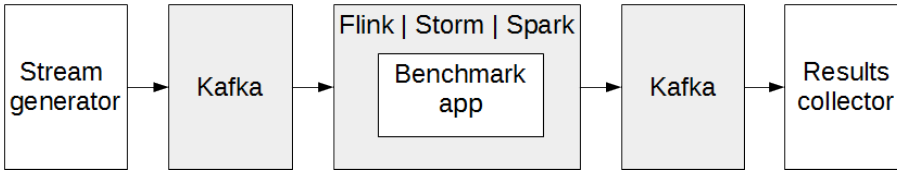


Рис. 6: Типовая схема тестирования потоковых фреймворков

Fig. 6 Typical benchmarking schema for streaming frameworks

4.1 Оценка задержки обработки данных

Одним из самых известных тестов потоковых фреймворков сейчас является Yahoo Streaming Benchmark [22]. Исходные тексты этого теста доступны и могут быть повторно использованы. Опубликованы результаты тестирования Apache Storm, Apache Spark Streaming и Apache Flink. Целью теста является оценка задержки времени обработки данных (Latency) при выполнении типовой цепочки обработки данных с простейшей агрегацией и чтением/записью данных в СУБД Redis. Целью теста не является оценка производительности, поэтому данный тест может быть использован лишь для

оценки применимости потоковых фреймворков для использования в интерактивных системах обработки данных.

В Yahoo Streaming Benchmark используется схема тестирования, приведённая на рис. 7. Задержка вычисляется по формуле:

$$window.final\_event\_latency = (window.last\_updated\_at - window.timestamp) - window.duration$$

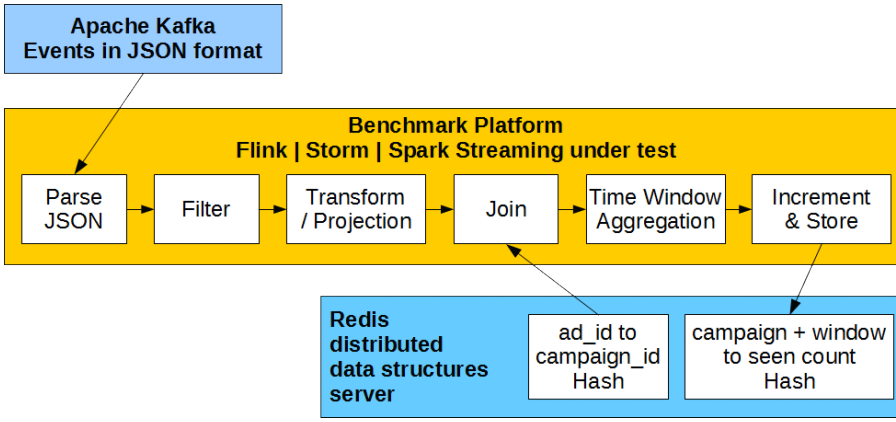


Рис. 7: Схема тестирования в Yahoo Streaming Benchmark

Fig. 7: Benchmarking schema in Yahoo Streaming Benchmark

В рамках теста реализовано 3 приложения для фреймворков Apache Storm, Apache Spark и Apache Flink.

4.2 Оценка автоматической подстройки конвейера обработки данных

Помимо задержки обработки, для интерактивных систем обработки данных важной характеристикой является регулярность выдачи результатов. Например, пусть установлена оконная операция с выдачей результата раз в 1 минуту. Независимо от того, сколько входных данных было получено, необходимо обеспечить обработку данных и выдачу результата. Подстройка скорости работы конвейера (backpressure) позволяет потоковому фреймворку обеспечить обработку данных в цепочке операторов со скоростью самого медленного оператора, исключая накопление перед ними больших массивов необработанных данных и потери возможности формирования отклика после них.

Модель оценки заключается в том, что на вход потоковой задачи, обрабатывающую данные в рамках скользящего окна, необходимо подать поток с плотностью больше, чем может быть выполнено за время, равное

смещению окна. То есть до момента переключения окна на следующий интервал надо обеспечить выдачу результата предыдущей обработки.

Пример, реализованный в https://github.com/rssdev10/spark-kafka-streaming, показывает, что в Apache Spark, например, отсутствуют эффективные механизмы подстройки конвейера. Это выражается в том, что если в момент фазы чтения данных из источника источник содержит больше данных, чем Spark может обработать, все данные будут загружены в Spark. Однако их полная обработка состоится неизвестно когда, независимо от установленного времени выдачи результата, см. рис. 8 b). Если же источник данных выдаёт входной поток с резкими всплесками, для Spark это приводит к резкому увеличению интервалов выдачи результатов. Например, вместо 10 секунд выдача результата может произойти раз в несколько минут. Если же рассматривать поведение Apache Flink в этой же ситуации, то результат обработки будет выдан почти гарантированно в установленное для окна время. См. рис. 8 a).

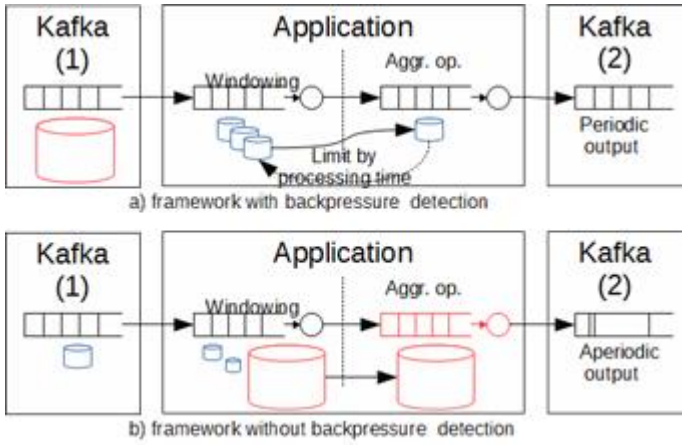


Рис. 8: Расположение данных в зависимости от подстройки конвейера

Fig. 8. Data concentration depending on presence of backpressure detector

Несмотря на то, что входные данные надо обработать в любом случае, тот факт, что фреймворк не успевает это сделать, означает лишь то, что данные где-то должны быть накоплены. Модель Flink упрощает дальнейшую обработку для программиста, которому не приходится делать дополнительные проверки времени выдачи результата.

То есть данная проверка позволяет оценить пригодность потоковых фреймворков для задач, требующих выдачи данных по строгому регламенту.

4.3 Оценка восстанавливаемости узлов кластера после сбоев

Распределённая кластерная система состоит из множества взаимодействующих узлов, каждый из которых может выйти из строя. Проверка потокового фреймворка на устойчивость работы в случае отключения отдельных узлов важна потому, что в отличии от пакетной обработки, потоковая задача запускается однократно и долговременно. Потоковая задача не может быть завершена без прекращения обработки запросов вообще. Если потоковый фреймворк не обеспечивает автоматическое восстановление узлов после сбоя, это ведёт к деградации производительности задачи. Именно поэтому в системах высокой доступности неприемлемо использование потоковых фреймворков, не способных обеспечить восстановление.

Модель оценки здесь достаточно простая. Она заключается в имитации сбоев узлов кластера, которая может быть реализована как вспомогательная программа, используемая совместно с любым другим тестом фреймворка. Необходимо имитировать аппаратные сбои (для потокового фреймворка выглядит как потеря процесса) или программные сбои задач (например, из-за нехватки оперативной памяти при выполнении).

Согласно тесту, проведенному в [33], например, Apache Spark 1.3 не способен восстановить работоспособность после одиночных сбоев. В то же время Apache Flink может работать долговременно без последствий от сбоев одиночных узлов.

4.4 Оценка возможности организации последовательной обработки данных без лишних сетевых обменов

В отличии от пакетной обработки, где данные могут быть заранее подготовлены и размещены в непосредственной близости от места обработки, в задачах потоковой обработки имеется проблема доставки этих данных. Кроме того, решаемые задачи в потоковой обработке часто алгоритмически более легковесны, чем задачи, решаемые при пакетной обработке. Это приводит к тому, что если фреймворк не способен обеспечить обработку данных (например, задачи агрегации по разделам) без лишних пересылок, то горизонтальное масштабирование производительности кластера будет невозможным.

Модель тестирования заключается в том, что данные для обработки должны быть предварительно распределены по узлам кластера, например, при помощи Apache Kafka, а потоковый фреймворк обязан построить план выполнения так, чтобы каждый узел использовал только данные из того раздела, который размещён на узле.

Проблема состоит в том, что в настоящее время потоковые фреймворки, как правило, лишь теоретически позволяют это сделать. Методы, позволяющие

организовать обработку подобным образом, обычно не документированы. К примеру, подобную схему можно организовать при помощи Apache Flink, но данные должны быть разложены по узлам при помощи нестандартной для Kafka функции распределения.

На рис. 9 представлена схема обработки с предварительным распределением данных. Цветом обозначены связанные операторы, распараллеливание которых обеспечивает потоковый фреймворк. Для простоты считаем, что входные данные одной и той же очереди Kafka(1) распределены на разделы по числу узлов кластера. Данные подвергаются некоторой предварительной обработке, операциям агрегации по каждому из разделов, после чего собираются и отправляются в выходные очереди Kafka(2).

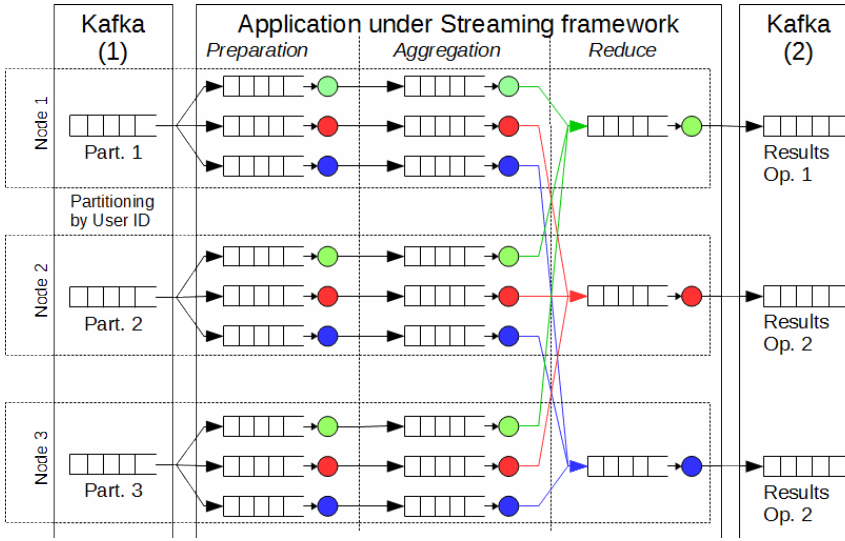


Рис. 9: Схема обработки с предварительным распределением данных

Fig. 9. Data processing with preliminary partitioning

4.5 Оценка предельных возможностей сохранения состояния (оконные операции)

Потоковые фреймворки используются для решения задач оценки входных данных по определённым критериям во времени, близком к реальному. Для этого применяются операции агрегации, которые могут быть выполнены над данными в рамках окна. При этом размер окна определяется решаемой задачей и может быть от нескольких секунд до суток. В последнем случае обычно используют так называемую лямбда-архитектуру [32], в рамках которой имеется постоянное хранилище для больших объёмов и долговременного хранения данных, а также небольшой объем оперативных

данных, принимаемых из входного потока. Однако надо понимать, что причина использования нескольких подсистем – ограничения существующих фреймворков.

Каппа-архитектура – это попытка построения систем обработки данных без использования отдельного долговременного хранилища и фреймворка для пакетной обработки данных. Любая операция агрегации предполагает накопление состояния, и, следовательно, наличие локального хранилища данных для каждого оператора, выполняемого в распределённой системе.

Легко подсчитать размер этого хранилища. Так, для случая входного потока с плотностью 100 тыс. сообщений в секунду и окна, размером в 1 час, необходимо хранить 360 млн. сообщений при условии, что потоковый фреймворк не создаёт дубликаты для случая скользящего окна с перекрытием. Или же данный размер следует кратно увеличить, если фреймворк реализует простейший случай скользящего окна.

Модель реализации этого теста достаточно проста. Необходимо обеспечить выполнение операций агрегации. Например, подходит решение задачи подсчёта количества уникальных пользователей и среднюю длительность сессии за 1 час с периодом обновления информации раз в 1 минуту на плотности запросов 100 тыс. в секунду.

Основная сложность этого теста заключается в том, что при подобных объемах поведение фреймворков становится нестабильным, и очень часто тесты вообще не могут быть выполнены.

4.6 Оценка прочих характеристик

В большинстве потоковых фреймворков в 2016-м году начала декларироваться поддержка языка SQL для написания запросов на выборку и обработку данных. Поскольку сейчас нет стандарта SQL, удовлетворяющего задачам потоковой обработки данных, разработчики потоковых фреймворков, по сути, ничем не ограничены в создании собственных диалектов SQL. Тем не менее, факт поддержки SQL заставляет задуматься над тем, как тестировать и этот аспект. В данный момент, вероятно, следует ставить вопрос о принципиальной возможности реализовать те или иные задачи с помощью SQL для каждого фреймворка конкретно и оценивать их производительность.

Кроме очевидных требований, приведённых выше, существуют специальные методы оптимизации выполнения графа операторов; состав этих методов описан в работе [27]. Среди них переупорядочивание операторов, устранение лишних операторов, балансировка нагрузки, рассылка состояния и пр. Очевидно, что это также влияет на скорость работы программ, однако тесты с учетом этих оптимизаций направлены в первую очередь на способность потоковых фреймворков автоматически обеспечивать оптимизацию не оптимальных программ. Это также влияет на скорость разработки и отладки программ.

Если вернуться назад к требованиям к потоковым системам реального времени, сформулированным в работе [40], то не рассмотренными остались требование 3 (дефектность входных потоков), поскольку это скорее задача прикладного программиста, а не фреймворка, а также требование 5 (интегрированность хранимых и потоковых данных), поскольку это зависит от модели информационной системы в целом.

5. Потоковые фреймворки с открытым исходным кодом

5.1 Apache Storm/Trident, Twitter Heron

Продукты Apache Storm [6] и Storm/Trident были популярны несколько лет назад и рекомендовались в качестве основы для систем потоковой обработки данных [32]. Однако в настоящее время их можно отнести к устаревшим программным продуктам, которые нельзя использовать в высоконагруженных системах. Основные недостатки – низкая производительность Trident, сложность настройки для работы в кластере, которая сводится к подбору параметров распараллеливания. Также следует отметить неспособность обеспечить нормированное время формирования ответа при резких всплесках плотности входного потока сообщений. Из отличительных достоинств следует назвать концепцию DRPC (distributed remote procedure call), в рамках которой Storm предоставляет программный интерфейс для интерактивных запросов, обеспечивая автоматическое перенаправление на свободные узлы кластера. Это также позволяет реализовывать микросервисы, способные выдать ответ на основе сохранённого состояния.

Twitter Heron [13] – новый проект, в котором декларируется обратная совместимость с приложениями, написанными для Apache Storm. Однако в данный момент продукт находится в разработке и не может быть рекомендован для реального использования.

5.2 Apache Spark Streaming

Apache Spark [5] является, пожалуй, самым известным фреймворком для решения задач машинного обучения. Модуль Apache Spark Streaming предназначен для потоковой обработки. Проблема Apache Spark заключается в его архитектуре. Изначально продукт ориентирован на пакетную обработку больших данных как замена Apache Hadoop и Map/Reduce. Apache Spark Streaming гипотетически позволяет решать потоковые задачи, однако наследие пакетной обработки сказывается негативно. Декларируемая возможность использования единого программного интерфейса для пакетного и потокового режимов работы, но в реальности это реализовано с большими ограничениями.

Рассмотрим схему тестирования фреймворка, представленную на рис. 10. Stream generator - генератор потока данных, Zookeeper - координатор кластера, Kafka - единственный узел Kafka, тестируемый фреймворк Flink или Spark, Benchmark – тестирующее приложение. Схема является простейшей, причём

вся генерация данных осуществляется на единственном узле управления, все остальные узлы - вычислительные. Это позволяет очень просто определить ограничения вычислительной сети.

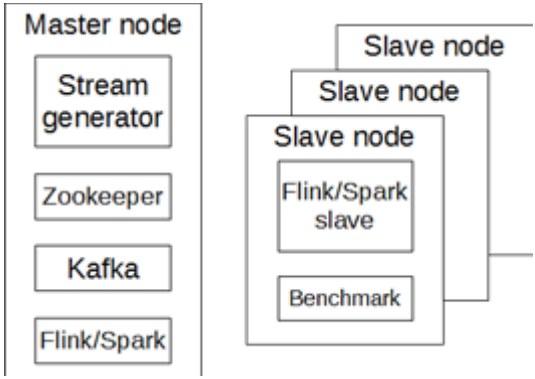


Рис. 10: Схема тестирования в режиме одного управляющего и множества вычислительных узлов

Fig. 10. Benchmarking with one master node and multiple slave nodes

На графике загрузки сети (рис. 11) очевиден характер работы Spark, где чётко прослеживаются фазы загрузки данных и фазы обработки. Это приводит к естественному физическому ограничению вычислительной сети и к ограничению производительности. Показатели загрузки сети получены при помощи утилиты dstat.

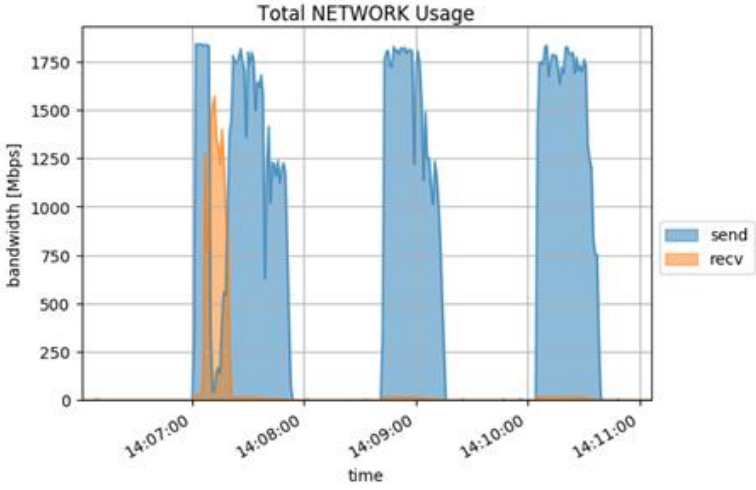


Рис. 11: График загрузки сетевого адаптера управляющего узла Spark

Fig. 11. Master node network utilization by Apache Spark

Существенной проблемой является то, что Spark не способен восстанавливать работу узлов кластера, которые выпали в результате сбоя [33]. Если речь идёт о пакетном режиме работы, это не является проблемой, поскольку пакетная задача будет выгружена после завершения обработки. В случае же потоков, где программы запускаются один раз и должны работать постоянно, это неприемлемо, поскольку ведёт к неминуемой деградации производительности.

Проблемой также является необходимость вручную подбирать и устанавливать ограничения плотности входного потока, поскольку особенность Spark в том, что не имея эффективных средств отслеживания обратного давления, всплеск входных данных приводит к тому, что Spark пытается захватить их все, игнорируя выставленное время обработки в окне.

Можно предположить, что в ближайшие 1-2 года разработчики не решатся изменить его ядро, поскольку потенциально это приведёт к потере совместимости с написанными ранее приложениями. В настоящее время продукт (версия 2.0.1) непригоден для работы в системах с нерегулярной нагрузкой, непригоден для интерактивных систем. Учитывая не слишком высокую производительность, высокое время задержки, а также массу ручных операций подбора параметров, можно предположить, что Spark будет одним из самых дорогих в эксплуатации продуктом.

5.3 Apache Flink

Apache Flink [2] позиционируется как универсальный фреймворк, то есть способный выполнять как потоковые, так и пакетные задачи, однако для того, чтобы избегать прямой конкуренции с Apache Spark, разработчики сместили основной акцент именно на обработку потоковых данных. Проект родился из академического проекта Stratosphere и отличается глубокой теоретической проработкой архитектуры [17].

В Apache Flink реализуется естественная потоковая обработка данных, гарантируется обработка сообщений строго одни раз, то есть "exactly-once", обеспечивается автоматическое балансирование нагрузки и восстановление после сбоев, а также присутствуют механизмы подстройки скорости обработки конвейера.

Из очевидных достоинств работы Apache Flink следует отметить непрерывность выполнения операторов без разделения на фазы загрузки данных и обработку. Тест, проведённый по схеме, представленной на рис. 10, показывает почти ровный график загрузки сетевого адаптера управляющего узла (см. рис. 12).

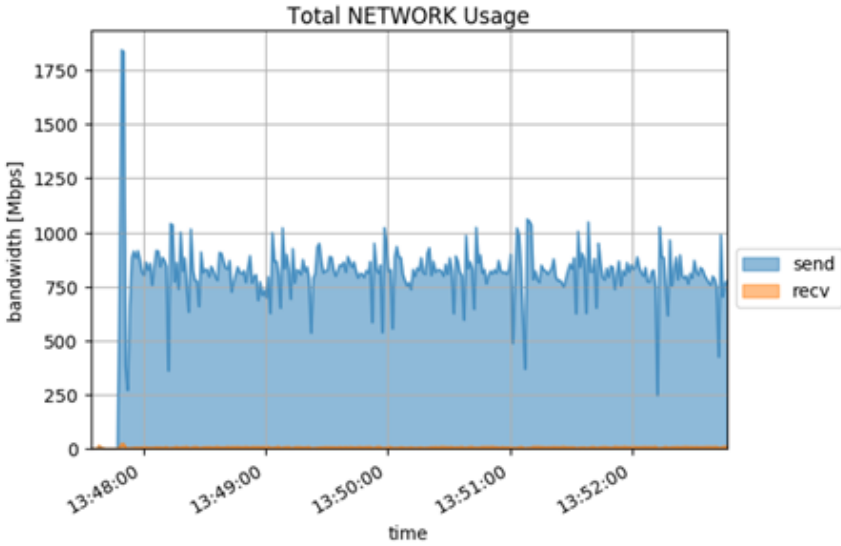


Рис. 12: График загрузки сетевого адаптера управляющего узла Flink

Fig. 12. Master node network utilization by Apache Flink

Apache Flink характеризуется низкими задержками выполнения [22] и высокой производительностью.

Из недостатков традиционно отмечают то, что проект достаточно молодой, и немногие крупные компании реально решили его использовать, однако динамика его использования за 2016 год говорит о достаточном его уровне развития.

Apache Flink имеет библиотеку для описания обработки в терминах CEP [42]]. Примером задачи CEP может быть выполнение действий при достижении некоторой температуры. Также поддерживается SQL [24].

5.4 Apache Kafka Streams

Kafka Streams [29] является результатом развития проекта Apache Kafka [3]. Поскольку Apache Kafka широко используется в бизнес-приложениях в качестве средства поддержки очередей сообщений, прочие потоковые фреймворки вынуждены обеспечивать чтение и отправку сообщений из Kafka. При этом на стыке безусловно возникают проблемы гарантированности обработки сообщений и производительности. Поэтому для компании Confluent было логичным предложить собственный инструмент потоковой обработки, который входит в состав Apache Kafka, начиная с версии 0.10. В настоящее время Kafka Streams – легковесная библиотека, позволяющая

выполнять традиционные операции с окнами, агрегацию данных и описывать собственные операторы для обработки данных.

Теоретическим плюсом данного фреймворка является доступность всех внутренних механизмов Apache Kafka, которые недоступны или не документированы для использования сторонними пользователями Kafka.

Разработчики Kafka делают акцент на том, что потоковые данные и таблицы по сути являются одним и тем же [29]. То есть таблица – это результат накопления некоторых событий за определенное время, или окно в терминах современного потокового процессора. Соответственно, в программном интерфейса Kafka предусмотрены классы KTable и KStream. Если же обеспечить долговременное хранение всего потока, получаем темпоральную СУБД.

Среди недостатков Kafka Streams следует отметить то, что продукт не является устойчивым, возможны изменения API, семантика операций агрегации отличается от других фреймворков. Например, операция aggregateByKey вместо накопления и выдачи нескольких значений в количестве, равном количеству ключей, будет выдавать частные агрегаты по каждому ключу на каждое входящее сообщение. Возможно, это поведение будет изменено в следующих версиях.

Ещё одним специфическим моментом Kafka Streams является то, что нет поддержки распараллеливания данных, масштабирования и автоматического контроля состояния, поскольку потоковое приложение для Kafka Streams – это автономное Java-приложение, связанное с Apache Kafka только стандартным протоколом взаимодействия. Иллюстрация работы приложения, использующего Kafka Streams, была приведена на рис. 5.

Если эти возможности не будут предоставлены программистам в ближайшее время, то прямым конкурентом Kafka Streams является Project Reactor [15]. Его достоинством является глубокая интеграция с Java (является кандидатом на включение в состав Java 9). Project Reactor уже использован в проекте Spring Framework 5. В активной разработке находится проект Reactor Kafka.

5.5 Apache Samza

Apache Samza [4] – потоковый фреймворк, демонстрирующий очень высокую скорость обработки сообщений и низкую задержку их обработки [41]. Samza позиционируется как продукт той же категории, что и Apache Storm.

Samza поддерживает режим обработки "exactly-once", то есть дубликаты сообщений возможны. Сохранение состояния обеспечивается за счёт БД «ключ-значение», связанной с каждой потоковой задачей.

Модель программирования – композиционная, причём программа создаётся в терминах классов Java, реализующих определённые интерфейсы. То есть, в отличии от Apache Flink или Spark, нет выбора языка программирования за пределами JVM.

Дополнительные особенности Samza рассматриваются в [14].

5.6 Apache Apex

Apache Apex [1] так же, как Apache Spark, Apache Flink, позиционируется как универсальный фреймворк для потоковой и пакетной обработки данных.

Так же, как и в Apache Flink, поддерживается режим "exactly-once", высокоуровневая декларативная модель программирования с возможностью работы с окнами данных и обработкой сообщений по их внутреннему времени или по системному времени. Поддерживаются средства ETL, имеется набор средств интеграции с различными очередями сообщений, включая Kafka, а также с различными СУБД. Декларируется интеграция со средствами Apache Beam, Apache SAMOA.

В статье [43] сравниваются Apache Apex и Apache Flink по схеме Yahoo Streaming Benchmark. При этом Apex показывает несколько меньшие абсолютные задержки и разброс их значений на больших объемах данных. Однако никаких оценок по другим критериям в статье нет.

6. Коммерческие программные продукты с закрытым исходным кодом

В отношении коммерческих потоковых фреймворков ситуация совершенно иная. В исследовании [36], например, сравниваются различные коммерческие фреймворки типа Cisco Connected Streaming Analytics, Data Torrent RTS, Esper Enterprise Edition, IBM Streams, Impetus Technologies StreamAnalytix, Oracle Stream Explorer, SAP Event Stream Processor, SAS Event Stream Processing, Software AG Apama Streaming Analytics Platform, SQLstream Blaze, Striim, TIBCO StreamBase, WSO2 Complex Event Processor и пр.

Главная проблема заключается в том, что часть вендоров выпустили продукты для потоковой обработки чисто номинально. В большинстве случаев потребитель не имеет возможности объективно сравнить их с открытыми потоковыми процессорами и должен довериться лишь авторитету марки. В данном случае следует отбрасывать компании, для которых программные разработки не являются основными. Указанный отчёт как раз и демонстрирует непроработанность продуктов у ряда компаний, что, скорее всего, приведёт в дальнейшем к отказу от их поддержки.

7. Заключение

В настоящее время наблюдается многообразие потоковых фреймворков. При этом следует отметить, с одной стороны, попытки разработчиков позиционировать свои продукты как универсальные и пригодные для всех случаев жизни, с другой стороны, существует тенденция создания специализированных фреймворков под конкретные задачи, свойственная

скорее крупным компаниям. И в одном, и в другом случае результат является набором компромиссов.

Дополнительно следует указать статью [11], где приводится сравнительная таблица 12-ти продуктов Apache, относящихся к группе потоковых фреймворков. В таблицу сведены лишь характеристики, декларируемые разработчиками этих продуктов.

Несмотря на годы развития потоковых фреймворков и их многообразие, до сих пор не существует никаких единых подходов для оценки их характеристик и экспериментальной проверки. Рассмотренные в этой статье методы могут быть использованы для проверки конкретных аспектов функционирования фреймворков и выбора подходящего продукта для конкретного случая. Задача создания общей методики тестирования, набора тестов и набора данных для проверки потоковых фреймворков всё ещё остаётся актуальной.

Список литературы

[1]. Apache Apex. https://apex.apache.org/. [Обращение 2017-01-02].

[2]. Apache Flink: Scalable Batch and Stream Data Processing. https://flink.apache.org/. [Обращение 2017-01-02].

[3]. Apache Kafka. https://kafka.apache.org/. [Обращение 2017-01-02].

[4]. Apache Samza. http://samza.apache.org/. [Обращение 2017-01-02].

[5]. Apache Spark™ - Lightning-Fast Cluster Computing. https://spark.apache.org/. [Обращение 2017-01-02].

[6]. Apache Storm. https://storm.apache.org/. [Обращение 2017-01-02].

[7]. Drools - Business Rules Management System (Java™, Open Source). https://www.drools.org/. [Обращение 2017-01-02].

[8]. Guaranteeing message processing. http://storm.apache.org/releases/current/Guaranteeing-message-processing.html. . [Обращение 2016-12-23].

[9]. RocksDB a persistent key-value store. http://rocksdb.org/. [Обращение 2017-01-02].

[10]. Spring. https://spring.io/. [Обращение 2017-01-02].

[11]. An Overview of Apache Streaming Technologies. https://databaseline.wordpress.com/2016/03/12/an-overview-of-apache-streaming-technologies/, 2016. [Обращение 2017-01-02].

[12]. Apache Flume. https://flume.apache.org/, 2016. [Обращение 2017-01-02].

[13]. Heron. A realtime, distributed, fault-tolerant stream processing engine from Twitter. https://twitter.github.io/heron/, 2016. [Обращение 2017-01-02].

[14]. Samza. Comparison Introduction. http://samza.apache.org/learn/documentation/latest/comparisons/introduction.html, 2016. [Обращение 2017-01-02].

[15]. Project Reactor. https://projectreactor.io/, 2017. [Обращение 2017-01-02].

[16]. Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: A new model and architecture for data stream management. The VLDB Journal, 12(2):120–139, August 2003.

[17]. Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix

Naumann, Mathias Peters, Astrid Rheinländer, Matthias J. Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, and Daniel Warneke. The stratosphere platform for big data analytics. The VLDB Journal, 23(6):939–964, December 2014.

[18]. Alexander Alexandrov, Andreas Salzmann, Georgi Krastev, Asterios Katsifodimos, and Volker Markl. Emma in action: Declarative dataflows for scalable data analysis. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016, pages 2073–2076. ACM, 2016.

[19]. Henrique C. M. Andrade, Bugra Gedik, and Deepak S. Turaga. Fundamentals of Stream Processing: Application Design, Systems, and Analytics. Cambridge University Press, New York, NY, USA, 1st edition, 2014.

[20]. Arvind Arasu, Mitch Cherniack, Eduardo F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: A stream data management benchmark. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004, pages 480–491. Morgan Kaufmann, 2004.

[21]. Paris Carbone, Gyula Fóra, Stephan Ewen, Seif Haridi, and Kostas Tzoumas. Lightweight asynchronous snapshots for distributed dataflows. CoRR, abs/1506.08603, 2015.

[22]. S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky. Benchmarking streaming computation engines: Storm, flink and spark streaming. In 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pages 1789–1792, May 2016.

[23]. [Saliya Ekanayake. Towards a systematic study of big data performance and benchmarking. PhD thesis, the School of Informatics and Computing, Indiana University, United States – Indiana, 10 2016. http://pqdtopen.proquest.com/doc/1845860615.html?FMT=ABS.

[24]. Hueske Fabian. Stream Processing for Everyone with SQL and Apache Flink. https://flink.apache.org/news/2016/05/24/stream-sql.html, 2016. [Обращение 2017-01-02].

[25]. Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pages 1197–1208, New York, NY, USA, 2013. ACM.

[26]. Lukasz Golab and M. Tamer Özsu. Issues in data stream management. SIGMOD Rec., 32(2):5–14, June 2003.

[27]. Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A catalog of stream processing optimizations. ACM Comput. Surv., 46(4):46:1–46:34, March 2014.

[28]. Kreps Jay. Putting Apache Kafka To Use: A Practical Guide to Building a Stream Data Platform. https://www.confluent.io/blog/stream-data-platform-1/, https://www.confluent.io/blog/stream-data-platform-2/, 2015. [Обращение 2017-01-02].

[29]. Kreps Jay. Introducing kafka streams: Stream processing made simple - confluent. https://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple/ , 2016. [Обращение 2017-01-02].

[30]. Kostas, Ewen Stephan, and Metzger Robert. High-throughput, low-latency, and exactly-once stream processing with apache flink. http://data-artisans.com/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink/, 2015. [Обращение 2016-12-23].

[31]. Ruirui Lu, Gang Wu, Bin Xie, and Jingtong Hu. Stream bench: Towards benchmarking modern distributed stream computing frameworks. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC '14, pages 69–78, Washington, DC, USA, 2014. IEEE Computer Society.

[32]. Nathan Marz and James Warren. Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2015.

[33]. Diana Matar. Benchmarking Fault-Tolerance in Stream Processing Systems. Master's thesis. TU-Berlin, 2016, 57 p.

[34]. Zaharia Matei, Wendell Patrick, and Das Tathagata. Diving into apache spark streaming's execution model. https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html, 2015. [Обращение 2016-12-23].

[35]. Guido Mazza. big data streaming processing engines under the umbrella of the apache foundation: benchmark and industrial applications. Master's thesis. Universita` degli Studi di Modena e Reggio Emilia, 2015. http://www.dbgroup.unimo.it/tesi/Magistrale/201516_Guido_Mazza_tesi.pdf

[36]. Gualtieri Mike, Curran Rowan, Kisker Holger, Miller Emily, and Izzi Matthew. The forrester wave™: Big data streaming analytics, q1 2016. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2, https://www.sas.com/content/dam/SAS/en_us/doc/analystreport/forrester-big-data-streaming-analytics-108218.pdf, 2016. [Обращение 2017-01-02].

[37]. Zapletal Petr. Comparison of apache stream processing frameworks: Part 1. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-1, 2016. [Обращение 2016-12-23].

[38]. Zapletal Petr. Comparison of apache stream processing frameworks: Part 2. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2, 2016. [Обращение 2016-12-23].

[39]. Tilmann Rabl, Michael Frank, Manuel Danisch, Hans-Arno Jacobsen, and Bhaskar Gowda. The vision of bigbench 2.0. In Proceedings of the Fourth Workshop on Data Analytics in the Cloud, DanaC'15, pages 3:1–3:4, New York, NY, USA, 2015. ACM.

[40]. Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. SIGMOD Rec., 34(4):42–47, December 2005. Перевод http://citforum.ru/database/articles/stream_8_req/.

[41]. Feng Tao. Benchmarking Apache Samza: 1.2 million messages per second on a single node. https://engineering.linkedin.com/performance/benchmarking-apache-samza-12-million-messages-second-single-node, 2015. [Обращение 2016-12-01].

[42]. Rohrmann Till. Introducing Complex Event Processing (CEP) with Apache Flink. https://flink.apache.org/news/2016/04/06/cep-monitoring.html, 2016. [Обращение 2017-01-02].

[43]. Rozov Vlad. Throughput, Latency, and Yahoo! Performance Benchmarks. Is there a winner? https://community.mapr.com/community/exchange/blog/2016/12/05/throughput-latency-and-yahoo-performance-benchmarks-is-there-a-winner-by-vlad-rozov, 2016. [Обращение 2017-01-02].

Survey of streaming processing field

R.S. Samarev <samarev@acm.org>
Bauman Moscow State Technical University,
ul. Baumanskaya 2-ya, 5/1, Moscow, 105005, Russia

Abstract. This article is devoted to review of current state of streaming processing field including. This includes some historical aspects and mentioning of leaps of technologies development. Big part is attended to aspects of working of streaming processing frameworks at the point of view programmers which are using ones to create own streaming applications. These aspects includes framework selection issues which are criteria of selections, hidden aspects of functioning, existing benchmarks for big data and streaming frameworks review, and different benchmarking techniques description. Also, this article contains comprehensive list of references.

Keywords: big data, benchmark, apache storm, spark, flink, samza, apex, kafka

DOI: 10.15514/ISPRAS-2017-29(1)-13

For citation: Samarev R.S., Review of streaming processing field. *Trudy ISP RAN/Proc. ISP RAS*, vol. 29, issue 1, 2017. pp. 231-260 (in Russian). DOI: 10.15514/ISPRAS-2017-29(1)-13

References

[1]. Apache Apex. https://apex.apache.org/. [Online; accessed 2017-01-02].
[2]. Apache Flink: Scalable Batch and Stream Data Processing. https://flink.apache.org/. [Online; accessed 2017-01-02].
[3]. Apache Kafka. https://kafka.apache.org/. [Online; accessed 2017-01-02].
[4]. Apache Samza. http://samza.apache.org/. [Online; accessed 2017-01-02].
[5]. Apache Spark™ - Lightning-Fast Cluster Computing. https://spark.apache.org/. [Online; accessed 2017-01-02].
[6]. Apache Storm. https://storm.apache.org/. [Online; accessed 2017-01-02].
[7]. Drools - Business Rules Management System (Java™, Open Source). https://www.drools.org/. [Online; accessed 2017-01-02].
[8]. Guaranteeing message processing. http://storm.apache.org/releases/current/Guaranteeing-message-processing.html. [Online; accessed 2016-12-23].
[9]. RocksDB a persistent key-value store. http://rocksdb.org/. [Online; accessed 2017-01-02].
[10]. Spring. https://spring.io/. [Online; accessed 2017-01-02].
[11]. An Overview of Apache Streaming Technologies. https://databaseline.wordpress.com/2016/03/12/an-overview-of-apache-streaming-technologies/, 2016. [Online; accessed 2017-01-02].
[12]. Apache Flume. https://flume.apache.org/, 2016. [Online; accessed 2017-01-02].
[13]. Heron. A realtime, distributed, fault-tolerant stream processing engine from Twitter. https://twitter.github.io/heron/, 2016. [Online; accessed 2017-01-02].

[14]. Samza. Comparison Introduction. http://samza.apache.org/learn/documentation/latest/comparisons/introduction.html, 2016. [Online; accessed 2017-01-02].
[15]. Project Reactor. https://projectreactor.io/, 2017. [Online; accessed 2017-01-02].
[16]. Daniel J. Abadi, Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Aurora: A new model and architecture for data stream management. The VLDB Journal, 12(2):120–139, August 2003.
[17]. Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, Felix Naumann, Mathias Peters, Astrid Rheinländer, Matthias J. Sax, Sebastian Schelter, Mareike Höger, Kostas Tzoumas, and Daniel Warneke. The stratosphere platform for big data analytics. The VLDB Journal, 23(6):939–964, December 2014.
[18]. Alexander Alexandrov, Andreas Salzmann, Georgi Krastev, Asterios Katsifodimos, and Volker Markl. Emma in action: Declarative dataflows for scalable data analysis. In Fatma Özcan, Georgia Koutrika, and Sam Madden, editors, Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016, pages 2073–2076. ACM, 2016.
[19]. Henrique C. M. Andrade, Bugra Gedik, and Deepak S. Turaga. Fundamentals of Stream Processing: Application Design, Systems, and Analytics. Cambridge University Press, New York, NY, USA, 1st edition, 2014.
[20]. Arvind Arasu, Mitch Cherniack, Eduardo F. Galvez, David Maier, Anurag Maskey, Esther Ryvkina, Michael Stonebraker, and Richard Tibbetts. Linear road: A stream data management benchmark. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, (e)Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004, pages 480–491. Morgan Kaufmann, 2004.
[21]. Paris Carbone, Gyula Fóra, Stephan Ewen, Seif Haridi, and Kostas Tzoumas. Lightweight asynchronous snapshots for distributed dataflows. CoRR, abs/1506.08603, 2015.
[22]. S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbaum, K. Patil, B. J. Peng, and P. Poulosky. Benchmarking streaming computation engines: Storm, flink and spark streaming. In 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pages 1789–1792, May 2016.
[23]. Saliya Ekanayake. Towards a systematic study of big data performance and benchmarking. PhD thesis, the School of Informatics and Computing, Indiana University, United States – Indiana, 10 2016. http://pqdtopen.proquest.com/doc/1845860615.html?FMT=ABS.
[24]. Hueske Fabian. Stream Processing for Everyone with SQL and Apache Flink. https://flink.apache.org/news/2016/05/24/stream-sql.html, 2016. [Online; accessed 2017-01-02].
[25]. Ahmad Ghazal, Tilmann Rabl, Minqing Hu, Francois Raab, Meikel Poess, Alain Crolotte, and Hans-Arno Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, SIGMOD '13, pages 1197–1208, New York, NY, USA, 2013. ACM.
[26]. Lukasz Golab and M. Tamer Özsu. Issues in data stream management. SIGMOD Rec., 32(2):5–14, June 2003.

[27]. Martin Hirzel, Robert Soulé, Scott Schneider, Buğra Gedik, and Robert Grimm. A catalog of stream processing optimizations. ACM Comput. Surv., 46(4):46:1–46:34, March 2014.

[28]. Kreps Jay. Putting Apache Kafka To Use: A Practical Guide to Building a Stream Data Platform. https://www.confluent.io/blog/stream-data-platform-1/, https://www.confluent.io/blog/stream-data-platform-2/, 2015. [Online; accessed 2017-01-02].

[29]. Kreps Jay. Introducing kafka streams: Stream processing made simple - confluent. https://www.confluent.io/blog/introducing-kafka-streams-stream-processing-made-simple/ , 2016. [Online; accessed 2017-01-02].

[30]. Tzoumas Kostas, Ewen Stephan, and Metzger Robert. High-throughput, low-latency, and exactly-once stream processing with apache flink. http://data-artisans.com/high-throughput-low-latency-and-exactly-once-stream-processing-with-apache-flink/, 2015. [Online; accessed 2016-12-23].

[31]. Ruirui Lu, Gang Wu, Bin Xie, and Jingtong Hu. Stream bench: Towards benchmarking modern distributed stream computing frameworks. In Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC '14, pages 69–78, Washington, DC, USA, 2014. IEEE Computer Society.

[32]. Nathan Marz and James Warren. Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2015.

[33]. Diana Matar. Benchmarking Fault-Tolerance in Stream Processing Systems. Master's thesis. TU-Berlin, 2016, 57 pp.

[34]. Zaharia Matei, Wendell Patrick, and Das Tathagata. Diving into apache spark streaming's execution model. https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html, 2015. [Online; accessed 2016-12-23].

[35]. Guido Mazza. big data streaming processing engines under the umbrella of the apache foundation: benchmark and industrial applications. Master's thesis. Universita` degli Studi di Modena e Reggio Emilia, 2015. http://www.dbgroup.unimo.it/tesi/Magistrale/201516_Guido_Mazza_tesi.pdf

[36]. Gualtieri Mike, Curran Rowan, Kisker Holger, Miller Emily, and Izzi Matthew. The forrester wave™: Big data streaming analytics, q1 2016. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2, https://www.sas.com/content/dam/SAS/en_us/doc/analystreport/forrester-big-data-streaming-analytics-108218.pdf, 2016. [Online; accessed 2017-01-02].

[37]. Zapletal Petr. Comparison of apache stream processing frameworks: Part 1. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-1, 2016. [Online; accessed 2016-12-23].

[38]. Zapletal Petr. Comparison of apache stream processing frameworks: Part 2. http://www.cakesolutions.net/teamblogs/comparison-of-apache-stream-processing-frameworks-part-2, 2016. [Online; accessed 2016-12-23].

[39]. Tilmann Rabl, Michael Frank, Manuel Danisch, Hans-Arno Jacobsen, and Bhaskar Gowda. The vision of bigbench 2.0. In Proceedings of the Fourth Workshop on Data Analytics in the Cloud, DanaC'15, pages 3:1–3:4, New York, NY, USA, 2015. ACM.

[40]. Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. SIGMOD Rec., 34(4):42–47, December 2005.

[41]. Feng Tao. Benchmarking Apache Samza: 1.2 million messages per second on a single node. https://engineering.linkedin.com/performance/benchmarking-apache-samza-12-million-messages-second-single-node, 2015. [Online; accessed 2016-12-01].

[42]. Rohrmann Till. Introducing Complex Event Processing (CEP) with Apache Flink. https://flink.apache.org/news/2016/04/06/cep-monitoring.html, 2016. [Online; accessed 2017-01-02].

[43]. Rozov Vlad. Throughput, Latency, and Yahoo! Performance Benchmarks. Is there a winner? https://community.mapr.com/community/exchange/blog/2016/12/05/throughput-latency-and-yahoo-performance-benchmarks-is-there-a-winner-by-vlad-rozov, 2016. [Online; accessed 2017-01-02].