ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 8
N 12
2017

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.272.45, 004.031.6, 004.4'233 DOI: 10.17587/prin.8.531-542

С. Г. Елизаров, канд. физ.-мат. наук, ст. науч. сотр., **Г. А. Лукьянченко,** науч. сотр.,
Д. С. Марков, ст. науч. сотр., МГУ имени М. В. Ломоносова, Физический факультет,
В. А. Роганов, ст. науч. сотр., e-mail: radug-a@ya.ru, МГУ имени М. В. Ломоносова,
Институт механики

Средства отладки программного обеспечения многоядерных процессоров

Рассмотрены разработка и применение инструментальных средств для отладки и профилирования прикладных программ на языках C/C++, исполняемых на перспективных многоядерных системах на кристалле. Цель данной статьи — поделиться с читателями уникальным опытом, полученным в ходе разработки на базе open source-решений системного эмулятора, отладчика и профилировщика для оригинального микропроцессора, содержащего сотни асинхронно взаимодействующих вычислительных ядер.

Ключевые слова: *технологии отладки, many-core-системы, системы на кристалле, эмуляторы оборудования, инструменты профилирования*

Введение

Последние годы число различных разрабатываемых и используемых на практике вычислительных систем растет экспоненциально, в том числе появляются и высокопроизводительные многоядерные системы на кристалле (many-core system on a chip). Если в прежние годы число ядер у процессора исчислялось единицами, то в настоящее время в перспективных многоядерных системах оно составляет тысячи ядер, что не может не влиять на инструментальные средства и технологии, используемые в процессе разработки их программного обеспечения [1—3].

Средства отладки для систем класса many-core аналогичны тем, что используют в случае традиционных систем. Однако их архитектура и режимы использования обладают определенной спецификой, к которой следует быть готовым разработчикам, планирующим заняться разработкой прикладного и системного программного обеспечения для подобных платформ.

Эта специфика связана прежде всего с тем, что в отличие от традиционных многопроцессорных систем кластерного типа, все взаимодействия между счетным ядрами (синхронизация, обмены данными) происходят исключительно внутри кристалла и недоступны для перехвата привычным для программных средств образом. Заметно усложняющим ситуацию фактором являются и ограниченные возможности по доступу к портам ввода—вывода для большинства счетных ядер, вследствие чего традиционная реализация синхронной отладочной печати на консоль оказывается невозможной.

В случае использования отладчика приходится задумываться о том, что происходит в системе, когда мы "ходим по шагам" в теле одной из процедур. Останавливаются ли в этот момент другие ядра или пока мы делаем один шаг и выполняем единственный оператор, они успевают выполнить тысячи? При наличии общей памяти это обстоятельство может заметно повлиять на то, что мы увидим в качестве значения переменных на следующем шаге.

Далее рассмотрим эти и другие вопросы, а пока проведем краткий обзор того, каким на сегодняшний день является ассортимент инструментальных средств для отладки программ, на которые могут рассчитывать разработчики прикладного программного обеспечения. Отметим, что наличие качественных инструментальных средств значительно повышает привлекательность и конкурентноспособность программно-аппаратной платформы, поэтому эта информация может быть интересна и разработчикам many-core-систем.

Требования к средствам отладки, трассировки и профилирования программ

Основные требования к современным отладочным средствам для встраиваемых систем можно свести к следующим основным пунктам:

- работоспособность при различных условиях и режимах работы;
- удобство и простота использования;
- уровень сервиса, сравнимый с отладочными средствами для обычных компьютеров.

Дополнительные возможности современных отладочных средств позволяют создавать различные пользовательские расширения, для этого необходимо выполнение следующих требований:

- параметризуемость, в том числе с помощью процедур на интерпретируемом языке;

• программная расширяемость, которая диктуется необходимостью предоставления API для поддержки пользовательских, предметно-ориентированных средств программирования.

Расширения отладочных средств позволяют снабжать прикладные пакеты дополнительными функциями, при использовании которых базовые отладочные средства могут визуализировать специфичные характеристики системы. Например, если для некоторой предметной области разработан специальный проблемно-ориентированный язык, то базовые отладчик и профилировщик смогут при отладке программ на этом языке выдавать информацию в более удобном для разработчика виде, который отражает специфику предметной области.

Далее будет дана краткая характеристика отладочных средств для различных программируемых систем по мере возрастания сложности последних, от классических однопроцессорных до современных многоядерных систем на кристалле класса many-core.

Классические средства отладки однопоточного программного обеспечения

Наиболее универсальным средством отладки практически любых программ является отладочная печать. Вне зависимости от того, на какой платформе и какое приложение реализует программа, возможность вывода на консоль или в журнал — это та возможность, которую в первую очередь хочет иметь в своем распоряжении любой программист. Расстановка отладочной печати также является примитивным способом трассировки (снятия трассы исполнения операторов) программы.

В случае языков C/C++ наиболее популярными являются функции типа printf(...). Важно понимать следующую особенность: для Unix-систем правильнее использовать для отладки вызовы вида fprintf(stderr,...), так как поток stderr обычно не буферизуется. При использовании же вызовов вида stdout обычно происходит буферизация вывода и в случае аварийного завершения программы можно не увидеть несколько последних сообщений. Заметим, что для Windows-программ этого может быть недостаточно. Поэтому поток, в который осуществляется буферизованный форматный вывод, необходимо "выталкивать" с помощью вызова функции fflush(...).

Когда поведение программы никак не укладывается в представления о ее функционале, практически любой разработчик периодически пользуется отладчиком, который в настоящее время встроен практически в любую интегрированную среду разработки. Для отладчиков существует богатый набор как текстовых, так и графических интерфейсных надстроек.

Поскольку основные функции программного отладчика общеизвестны, приведем лишь основные принципы работы традиционных отладчиков для Unix-подобных систем.

• Поддержка отладчиков в операционной системе реализуется с помощью специальных системных вызовов, которые позволяют подключаться к отлаживаемому процессу и контролировать его выполнение, а также обращаться к содержимому его памяти и регистров.

• Исполняемые модули, собранные с отладочной информацией, содержат почти полную информацию об адресах переменных и функций, которая создается компилятором и находится вместе с кодом в объектных модулях.

• Современные программируемые открытые системы предоставляют массу лазеек, востребованных при отладке программ, начиная с таблиц функций для динамической загрузки библиотек и заканчивая специальными файловыми системами типа "/proc".

Для профилирования программ в открытых Unix-подобных операционных системах используют различные подходы. Наиболее распространенный из них состоит в компиляции и сборке программ со специальной опцией (-pg в случае компилятора gcc). В программу при этом добавляют дополнительный код, который увеличивает значения счетчиков при вызове тех или иных процедур либо синхронно при их вызове, либо с помощью сигналов от таймера. Последний случай несколько ухудшает точность сбора статистики о времени исполнения отдельных программных секций, но оказывает меньшее влияние на время исполнения программного кода.

Средства отладки многопоточного программного обеспечения для одноядерного процессора

К имитации многопоточности (с помощью *green threads* [4]) часто прибегают в тех случаях, когда желательно, чтобы некоторые операции (чаще всего операции ввода/вывода) не блокировали основную ветвь программы. В настоящее время многие системы программирования прямо или косвенно используют такой подход ввиду его простоты и относительной эффективности. Для многих платформ имитацию многопоточности обеспечивает библиотека GNU Pth, которая предоставляет полноценный Posix Threads API. Часто имитация многопоточности реализуется еще проще: создается очередь "легковесных задач" (*tasklets*), поочередно выполняющихся процессором. При наличии такого "логического" параллелизма, но отсутствии физического процесс отладки все еще достаточно прозрачен для программиста.

Стандартные отладочные функции типа printf(...) работают корректно, поскольку в их функционирование, по понятным причинам, никто не вмешивается.

С отладчиком и профилировщиком в псевдопараллельных программах дела обстоят примерно так же хорошо, как и в случае последовательных программ, поскольку с точки зрения системы это, по сути, обычная последовательная программа. Имитируемые потоки представляются в ней как структуры данных, а их состояние можно визуализировать в отладчике стандартным способом.

Если в системе используется вытесняющая многозадачность, когда операционная система или специальная библиотека прерывает работу потока в произвольные моменты времени (например, по таймеру) и переключает процессор на другой поток, то ситуация радикально меняется. Стандартные функции для отладочной печати становятся малопригодны, так как результаты отдельных их вызовов могут как минимум перепутаться произвольным образом. Более того, последовательность исполнения операторов может непредсказуемо меняться от запуска к запуску.

Однако следует заметить, что в случае запрета переключения контекста для единственного процессорного ядра на время отладочной печати ситуация с печатью нормализуется. Упрощается и само программирование таких систем в целом. Функции запрета/разрешения переключений контекста действуют как естественный барьер и временно переводят недетерминированную асинхронную систему в детерминированный синхронный режим со всеми возможными последствиями.

Отладчик показывает системные потоки программы с помощью специальных команд. Можно посмотреть список активных потоков, переключаться между ними и выполнять отдельные потоки "по шагам". Ниже приведены некоторые команды для отладчика GDB и примерный вид их выдачи на консоль.

Просмотр активных потоков:

```
(gdb) info threads
       Id  Target Id          Frame
     * 1   process 35 thread  13  main (argc=1, argv=0x7ffffff8)
       2   process 35 thread  23  0x34e5 in sigpause ()
       3   process 35 thread  27  0x34e5 in sigpause ()
         at threadtest.c:68
```

Переключение на интересующий нас поток:

```
(gdb) thread 2
     [Switching to thread 2 (Thread 0xb7fdab70 (LWP 12747))]
     #0 some _ function (ignore = 0x0) at example.c:8
     8    printf ("hello\n");
```

Добавление точки останова программы в нужном месте кода для выбранного потока:

```
(gdb) break frik.c:13 thread 28 if bartab > lim
```

Такие точки останова (с указанным thread id) могут быть автоматически удалены отладчиком, если соответствующий поток прекратит свое существование:

```
(gdb) c
Thread-specific breakpoint 3 deleted - thread 28 no longer
   in the thread list.
```

Для отладчика не составляет особого труда контролировать ход программы в случае одноядерного процессора, так как в каждый момент времени в программе существует лишь единственная активная ветвь исполнения. Средства профилирования также особого усложнения в этом случае не претерпевают, за исключением того, что им уже требуется минимальная аккуратность при работе со счетчиками. Для решения этого вопроса можно использовать атомарные операции типа atomic_inc. Отметим, что такое решение с некоторыми дополнительными модификациями подходит также и в случае многоядерных систем.

Средства отладки программного обеспечения для доступных многоядерных процессоров

При наличии физического параллелизма и полноценной асинхронности ситуация с отладкой многопоточных программ резко усложняется. Здесь уже для отладочной печати из нескольких потоков требуется как минимум реализация межпотоковой блокировки. Но у такой простейшей реализации появляется нежелательный побочный эффект: выполнение множества потоков частично синхронизуется при печати. При этом тайминги (временные интервалы при выполнении процедур) и общий характер работы программы резко меняются, а сама отладочная печать превращается в нежелательный механизм сброса кэш-памяти разных уровней.

Опытным системным программистам хорошо известна ситуация, когда неправильная программа начинает работать корректно после добавления отладочной печати, что достигается не только в силу дополнительной синхронизации, но и за счет того, что компилятор, обрабатывая дополнительные вызовы отладочных функций, начинает вытеснять кэшированные в регистрах значения переменных в память, при этом другие потоки, а также аппаратура (если это значения портов ввода—вывода) начитают на это реагировать.

При использовании отладчика в многоядерных системах появляется интересный вопрос: чем занимаются другие ядра, пока пошагово выполняются операторы программы в одном из потоков? Совершенно ясно, что здесь кроме технических вопросов (при реализации отладчика) есть и чисто философский: должны остальные ядра простаивать, временно запускаться при каждом шаге программы в отладчике или работать не останавливаясь? В зависимости от желаемого поведения здесь, по сути, получаем разные режимы отладки.

Забегая вперед, отметим сейчас лишь то обстоятельство, что наиболее востребованным является компромисный режим, когда все потоки приостанавливаются при останове программы (*Stop World*) и возобновляют свою работу только во время выполнения единственной строки кода отлаживаемого потока.

Среди специалистов встречается скептическое мнение, что многоядерные системы привычными средствами отлаживать практически невозможно. Такой пессимизм имеет под собой, как следует из изложенного выше, определенные основания. Однако, если посмотреть на фактор физического параллелизма как на определенную специфику и сопоставить это с возможностью учета в современных отладочных средствах разного рода специфики с помощью расширений, то нетрудно прийти к выводу, что здесь все зависит исключительно от продуманности реализаций этих самых специфических расширений, а также от того, какой класс параллельных программ необходимо отлаживать.

Средства отладки заказных систем класса many-core

Если, так или иначе, необходимо расширять возможности привычных отладочных средств для обеспечения приемлемого уровня сервиса в случае отладки параллельных программ для обычных многоядерных процессоров с несколькими равноправными ядрами (чаще всего они относятся к классу SMP — *Symmetric Multiprocessor*), то можно углубиться и рассмотреть произвольную, неоднородную по своей структуре систему со значительным числом разных по своей архитектуре счетных ядер. Такие системы разрабатывают сегодня для решения различных прикладных задач в виде заказных сверхбольших интегральных схем (СБИС). Они могут содержать тысячи и более слабо связанных между собой процессорных элементов. В некотором смысле такая заказная СБИС по своей структуре напоминает вычислительный кластер, который полностью умещается на одном кристалле.

Преимущества таких систем обычно связаны с тем, что процессорные элементы разрабатываются под определенный набор задач, имеют специализированное арифметико-логическое устройство (АЛУ) и, как следствие, могут быть существенно эффективнее универсальных процессоров в части производительности и энергопотребления.

Разработку подобных СБИС/ПЛИС осуществляют обычно на языках описания аппаратуры, таких как VHDL и Verilog. С помощью низкоуровневых сущностей языков описания аппаратуры приходится описывать не только поведение всех типов используемых процессорных элементов, но и внутрисистемную шину с нетривиальной топологией, которая используется для межпроцессорного взаимодействия и для доступа в общую память.

Разработка высокоэффективных параллельных программ для many-core-систем является не простой задачей, так как даже для обеспечения загрузки работы базовых системных функций требуется написать объемный программный код на низкоуровневых (на языке ассемблер) и высокоуровневых языках типа C/C++. В связи с чем надежно работающие отладочные средства востребованы при разработке не только прикладного, но и системного ПО. Заметим, что для получения надежно работающих конструкций типа printf(...) и assert(...) требуются большие трудозатраты.

Разумеется, адаптацией отладочных средств для специализированных многоядерных процессоров давно занимаются их производители. Например, для технологии CUDA [5] предоставляется специально адаптированный отладчик и возможность отладочной печати непосредственно со счетных ядер. Однако далеко не все коммерческие фирмы предоставляют исчерпывающую документацию для своих продуктов, да и использовать средства, адаптированные к одним устройствам, для совершенно другой по архитектуре системы невозможно. Поэтому при разработке заказных специализированных многоядерных процессоров, кроме первоочередных по важности задач по разработке ассемблера и компилятора, приходится уделять внимание развитию средств отладки, трассировки и профилировки программ.

Постановка задачи

Рассмотрим основные возможности, которые отладочные средства должны предоставлять для систем класса many-core.

- Снятие полной или частичной, а в силу сложности таких систем — потактовой трассы исполнения программы для выбранного подмножества процессорных ядер.
- Функции отладочной печати и остановки программы при заданных условиях, применимые для любого счетного ядра системы с возможностью его идентификации.
- Подключение отладчика к работающей системе при выполнении заданных условий, просмотр состояния каждого вычислительного ядра, переключение между ними и пошаговое/потактовое выполнение машинных инструкций с возможностью возобновления работы после просмотра/изменения содержимого общей и локальной памяти, а также регистров вычислительных ядер. Приостановка исполнения программы и возвращение управления в отладчик по специальной команде оператора.
- Возможность получения как оперативной, так и итоговой статистической информации о программе — число тактов, проведенное каждым процессорным ядром в различных режимах и программных секциях.
- Средства отладки должны быть привычны для разработчиков, удобны в использовании и оставаться расширяемыми для учета возможной специфики разрабатываемых прикладных программ.

Далее будут рассмотрены опробованные решения для реализации перечисленных возможностей, а пока заметим только то, что сделать аппаратное устройство на кристалле, которое было бы максимально эффективным и одновременно полностью прозрачным для процесса отладки, представляется малореалистичным. Поэтому при разработке подобных систем часть задач по отладке решают с помо-

щью программного эмулятора, который создает программную модель аппаратуры для разрабатываемого программного обеспечения.

Эмулятор системы класса many-core

В данном разделе описаны основные особенности программного эмулятора для many-core-системы по опыту отладки программ, с использованием которого написана данная статья.

Требования к эмулятору

Основное требование к любому эмулятору системы с нестандартной архитектурой — максимально близкое повторение аппаратных особенностей целевой платформы. Эмулятор должен обеспечивать валидацию программ для целевой платформы, накладывая на их исполнение те же ограничения, что и аппаратная реализация. Однако в отличие от аппаратуры эмулятор должен сообщать о недопустимых операциях, выполняемых в эмулируемой программе, максимально удобно для программиста, чтобы упростить адаптацию ПО.

Реализация многих средств отладки и профилировки (например, полноценного отладчика GDB) в нестандартной аппаратуре может быть существенно затруднена. В то же время их реализация средствами эмулятора может быть существенно проще и при этом предоставлять программисту практически полностью аналогичные возможности для адаптации и отладки ПО. Таким образом, эмулятор должен дополнять, где это необходимо, аппаратную реализацию в части отладочных средств.

Структура и возможности эмулятора

Рассматриваемый эмулятор реализован на языке C++, при этом каждый аппаратный компонент исходной системы имитируется с помощью отдельного (C++)-объекта. Общая структура эмулируемой системы показана на рис. 1, внутренняя иерархия эмулятора, соответственно, следует той же схеме.

Эмуляция вычислительных ядер. Эмуляция вычислительных ядер в описываемом эмуляторе проводится методом раздельного виртуального тактирования всех ядер системы с исполнением одной инструкции на каждом эмулируемом процессорном ядре за такт. В рассматриваемой системе все ядра являются полностью независимыми исполнителями, не имеющими общих компонентов (кроме общей внешней памяти) и не обладающими кэш-когерентностью. По этой причине исполнение инструкций на них происходит полностью независимо друг от друга.

Эмуляция внутрикристальных шин. Виртуальные внутрикристальные шины в эмуляторе должны обеспечивать программный интерфейс для передачи данных между компонентами системы, полностью аналогичный аппаратной реализации. Для проверки корректности многопоточных программ эмуляторная модель внутрикристальной шины обеспечивает латентность передачи, эквивалентную аппаратной реализации.

Рис. 1. Структурная схема эмулируемой системы: сплошными стрелками показаны внутрикристальные шины; SOC — система на кристалле; выч. ядро — вычислительное ядро

Эмуляция общей памяти. Эмулятор системы класса many-core, обладающей общей памятью, должен иметь массив данных, эмулирующий память системы. При этом доступ к данному массиву должен обеспечиваться с каждого виртуального вычислительного устройства с учетом ограничений, накладываемых аппаратной реализацией системы. Примером таких ограничений является требование к выравниванию адресов. Важным аспектом эмуляции общего ресурса является корректное обеспечение арбитража доступа, который аналогичен таковому в аппаратуре.

Так как доступ к общему ресурсу всегда связан с латентностью, а на многоядерных системах она может достигать крайне больших значений и отличаться для различных ядер системы, то для корректного профилирования производительности программ эмулятор должен правдоподобно имитировать латентность доступа к памяти.

Эмуляция периферии. В любом эмуляторе процессорной системы необходимо программным образом имитировать доступ к периферии, подключенной к вычислительным ядрам. Например, виртуальный UART обычно выводится в терминал, из которого запущен эмулятор. В рассматриваемой системе вся периферия представлена устройствами, отображенными на память, и аналогично остальным компонентам реализована в эмуляторе в виде объектов языка C++.

Кроме устройств, присутствующих в аппаратуре, эмулятор создает дополнительные, виртуальные устройства на каждом вычислительном ядре. Такие устройства обеспечивают интерфейс из исполняемых на ядрах программах к некоторым функциям эмулятора (например, к виртуальной файловой системе, реально находящейся на хост-машине).

Отладочные возможности эмулятора

Рассмотрим реализованные непосредственно в эмуляторе (без использования отладчика) средства, упрощающие отладку программ.

Проверка корректности доступа к памяти. В рассматриваемом эмуляторе каждое обращение виртуальных ядер в общую память проходит ряд проверок, исключающих их некорректную обработку в аппаратуре. Проверяется адрес обращения для исключения обращений за пределы памяти или к несуществующему отображенному на память устройству, а также выравненность обращений по адресам. В случае недопустимого обращения работа эмулятора останавливается с исключением и на экран выводится участок кода, в котором произошла ошибка.

Проверка переполнения стека. Для контроля над переполнением программного стека после каждой машинной инструкции проверяется его значение. Если оно вышло за допустимые пределы, то выполнение программы аварийно завершается с выводом на консоль указания на место в программе, где произошло данное событие.

Наблюдение за ячейками памяти (функция *watch*). Эмулятор предоставляет возможность проводить мониторинг всех обращений к некоторым ячейкам памяти, которые могут быть заданы при старте эмулятора или использовании специального интерфейса к эмулятору из исполняемой в нем программы.

Показ исходного кода при аварийном завершении программы. При нештатном завершении программы или невыполнении условия assert(...) практически всегда интересно знать, где это произошло. Поэтому помимо численного значения адреса, на котором "споткнулась" программа, имеет смысл печатать и строчки выполняемого исходного текста.

Ниже представлен пример сообщения об ошибке с расшифровкой программного контекста.

```
### Write to NULL
### Exiting from emulator (419580 cycles, 7268601 cycles/sec)
### Finish: MBLite Core 0x0 @ 0x8000066c (active)
### At smalloc.cc:105
103           while (sz) {
104             i++; sz >>= 1;
105     =>      *(int*)nullptr = 0;
106           }
107             return i;
### At smalloc.cc:111
109           inline void* malloc(size_t sz) {
110             //cprintf("malloc\n");
111     =>      return pool[lg2m(sz)].get();
112           }
113           void inline free(void *mem) {
### At smalloc.cc:138
136             //cprintf("alloc\n");
137             size = rand(MAX_MEM_SIZE);
138     =>      mem = MALLOC(size);
139             #if FILL_MEM
140             memset(mem,0xAA,size);
### At smalloc.cc:157
155           WITH_TIMER(t_comp);
156           for (int i = 0; i<N_LOOPS; i++)
157     =>      cell[rand(N_CELLS)].toggle();
158           }
159           smalloc.printStat();
```

Мониторинг состояния программного обеспечения и аппаратуры. Поскольку эмулятор выполняет программы на несколько порядков медленнее реальной аппаратуры, также полезна возможность "наблюдения" за состоянием работающей программы непосредственно во время ее исполнения. В рассматриваемом эмуляторе это осуществляется с помощью псевдографических возможностей терминала, в верхней части которого выводится относительно существенная информация — загруженность ядер и список наиболее часто исполняемых программных секций. Пример вывода такой информации представлен на рис. 2, см. третью сторону обложки.

Рассмотренные отладочные возможности эмулятора решают значительную долю задач по отладке ПО, но не могут заменить полноценный отладчик, который применим также для отладки программ на реальной аппаратуре. Методике адаптации традиционных отладчиков (на примере GDB) к many-core-системам и посвящен следующий раздел данной статьи.

Отладчики для систем класса many-core

В отличие от случая традиционных десктопных приложений программный отладчик функционирует не внутри рассматриваемой отлаживаемой системы на кристалле, а на машине разработчика. Внутри отлаживаемой системы (реальная аппаратура или эмулятор) функционирует лишь так называемая ответная часть, которая в случае отладчика GDB именуется GDBstub.

Специфика many-core-систем проявляется в данном случае в том, что счетные ядра являются практически независимыми устройствами, что заметно усложняет управление ими для GDBstub по сравнению с другими системами на кристалле.

Схема работы GDB при отладке на реальном оборудовании

Взаимодействие между отладчиком и ответной частью происходит по специальному протоколу RSP (*Remote Serial Protocol*) (рис. 3) [6]. Данный подход хорошо отработан и широко применяется для отладки кода операционных систем, а также для отладки встроенных систем.

Демультиплексор последовательного канала KDMX. При реализации описанной выше схемы возникает следующий вопрос: если порт ввода/вывода используется отладчиком, то куда программа печатает свои сообщения? Для решения этого вопроса есть два пути: реализация и подключение дополнительного порта для связи с отладчиком, что накладывает

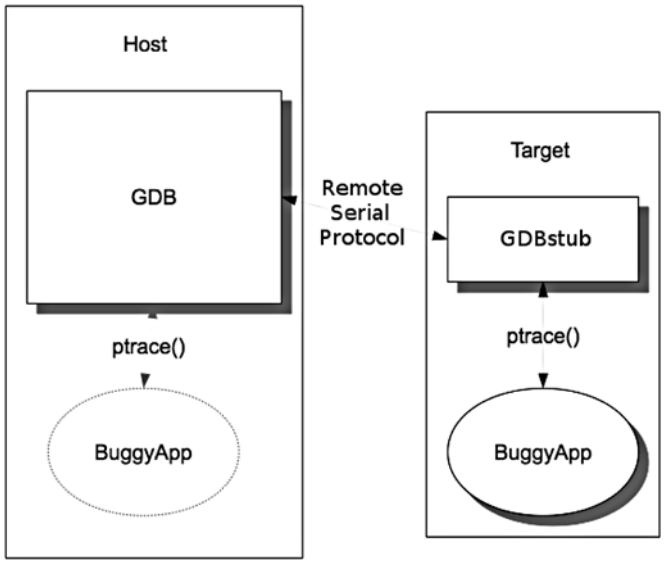


Рис. 3. Схема взаимодействия с отлаживаемой системой для стандартного (*слева*) и удаленного (*справа*) способов отладки программ:
Host — машина, с которой осуществляется отладка; GDBstub — ответная часть отладчика; Remote Serial Protocol — протокол связи с удаленным GDBstub; Target — целевое устройство; BuggyApp — отлаживаемое приложение

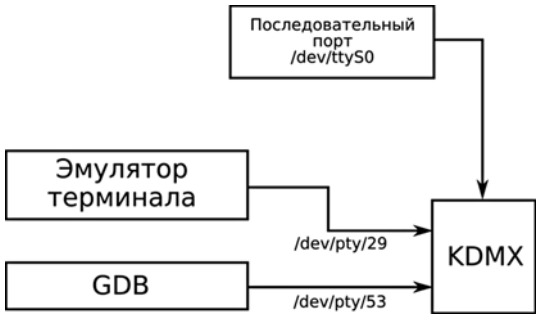


Рис. 4. Схема работы оригинальной утилиты KDMX: /dev/pty/* — устройства псевдотерминала

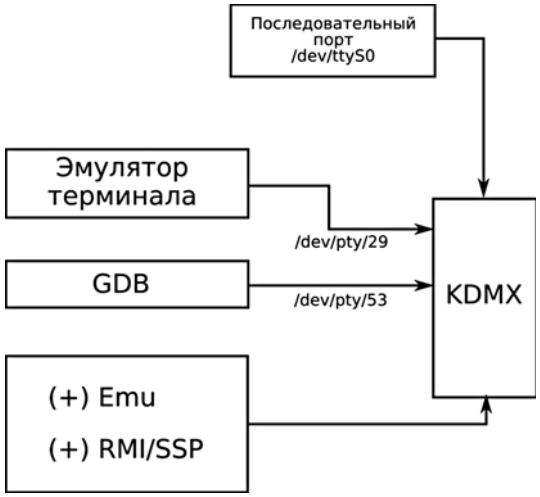


Рис. 5 Схема работы доработанной утилиты KDMX:
Emu — эмулятор системы many-core; /dev/pty/* — устройства псевдотерминала

дополнительные требования на оборудование, и разделение (мультиплексирование) единственного порта с использованием специфики вышеупомянутого протокола RSP.

В качестве такого мультиплексора можно использовать утилиту KDMX [7], общая схема работы которой представлена на рис. 4.

Авторами была предложена и проведена доработка KDMX, которая позволила подключить в качестве абонента коммуникационного канала эмулятор, а также дополнительно выполнять разного рода вспомогательные действия на машине разработчика с помощью упрощенной реализации известного принципа RMI (*Remote Method Invocation*) [8]. Схема работы этой связки представлена на рис. 5.

Протокол RSP и поддержка многопоточных программ. Когда на машине разработчика запускается отладчик GDB, для соединения с отлаживаемой системой по протоколу RSP необходимо использовать команду target remote. На целевой системе при этом должна нормально функционировать соответствующая ответная часть, именуемая GDBstub. После успешного обмена начальными сообщениями отладчик готов выполнять команды пользователя, которые в этом случае ничем не отличаются от команд, используемых при отладке локально запущенного процесса.

Однако следует иметь в виду, что для многоядерных систем реализация корректно работающей ответной части представляет собой непростую задачу. Для удобства отладки ответной части в GDB предусмотрен специальный режим трассировки RSP-сообщений.

Данные пакета RSP-протокола обычно состоят из последовательности текстовых символов, за исключением # и $. Общий формат пакетов представлен на рис. 6.

Во время отладки программы рабочая машина (в лице GDB) посылает команды, а целевая (в лице ответной части, включенной в программу) посылает ответы. В случае команд пошагового выполнения и продолжения ответ посылается только тогда, когда текущая операция закончена (отлаживаемая программа приостановлена).

Для демонстрации более сложных ситуаций далее представлен пример того, что происходит при асинхронной остановке программы по нажатию комбинации Ctrl + C на клавиатуре рабочей машины.

1. Обнаружив, что с клавиатуры пришла комбинация Ctrl + C, отладчик посылает удаленной отлаживаемой системе одиночный байт 0x03.

2. Отлаживаемая система, обнаружив приход байта 0x03 по каналу связи (по прерыванию от последовательного порта либо каким-то иным способом),

Рис. 6. Общий формат используемого в RSP пакета данных

пытается остановить все активные ядра. После этого она передает управление в GDBstub. Программа GDBstub собирает информацию о состоянии отлаживаемой программы и посылает отладчику соответствующий RSP-пакет.

3. Все активные ядра ожидают от выделенного процессора, который обрабатывает RSP-пакеты, разного рода информационные запросы. Отладчик может попросить переслать ему содержимое фрагментов локальной памяти или регистров для каждого конкретного процессорного элемента.

4. Все системные прерывания ставятся в очередь на обработку. В режиме Stop World, который является основным, исполняются только команды отладчика.

5. Когда оператор попросит возобновить выполнение программы, счетные ядра покидают отладочный режим, и система продолжает свою работу до следующей точки останова.

Далее приведен фрагмент дампа протокола RSP со стороны GDB в аналогичной ситуации (при выполнении отлаживаемой программой машинной инструкции TRAP) с краткими комментариями.

```
Waiting for remote debugger
Ack - пришло подтверждение готовности к работе от target

Sending packet: $Hg0#df...Ack
Packet received: OK - запрос на подключение к активному потоку

Sending packet: $?#3f...Ack
Packet received: S05 - причина остановки: TRAP instruction
Sending packet: $qfThreadInfo#bb...Ack
Packet received: mffffffe,01 - информация о текущем потоке

[...]

Sending packet: $g#67...Ack
Packet received:
0000000080002d70000000000000000000800143a883fffefc0000002b0000000
00000000080002e34000000000000000b0000002580002d64000000050000000000000000100000000
00000000000000000000000000000000000000000000000633961000000576200000000000000000000
00000000000000000800081300000000080008138000000000000000000000000ffffff80000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000
- запросили регистры CPU текущего потока

main () at test.cc:76
76 HDB _ BREAK; - GDB показал, наконец, текущую строчку кода.
```

Специфика отладки программ на реальном оборудовании. Не все проблемы могут проявиться на эмуляторе. В качестве примера можно рассмотреть программную ошибку, которая проявляется через пару минут работы на реальном устройстве. Под эмулятором проявления этой ошибки можно ожидать на протяжении нескольких часов. Кроме того, в разработанном оборудовании могут также наблюдаться различные эффекты, для наблюдения и анализа которых полезен отладчик. Поэтому, несмотря на все преимущества эмулятора, реализация ответной части для реального оборудования во многих случаях вполне оправдана.

Отладка в режиме эмуляции

При использовании отладчика GDB для отладки программ, работающих под эмулятором, нет необходимости в реализации ответной части GDBstub. Достаточно реализовать ответы на высокоуровневые команды пользователя непосредственно в самом GDB, который может получать любую информацию от эмулятора. Это легко можно сделать путем использования Python API, как описано в следующем разделе.

Расширения функционала GDB с помощью языка Python. Весьма перспективным решением для расширения возможностей отладчика GDB является использование последним встроенного интерпретатора языка Python. Следует отметить, что данная возможность является недостаточно известной в среде разработчиков. Однако она постепенно обретает популярность в качестве средства поддержки предметно-ориентированных режимов отладки программ.

Так, например, текущие версии отладчика GDB печатают содержимое объектов типа std::string в языке C++ в более наглядном и понятном для пользователя виде, в то время как базовая реализация этих объектов (в виде структур данных) была бы менее наглядна:

```
(gdb) print str
$1 = {static npos = 4294967295,
  _ M _ dataplus = {<std::allocator<char>>=
```

```
    {< __ gnu _ cxx::new _ allocator<char>>
     = {<No data fields>}, <No data fields>},
  _ M _ p = 0x804a014 "hello world"}}

>>>  gdb.pretty _ printers['^std::basic _ string<char,.*>$'] =
>>>       StdStringPrinter
(gdb) print str
$1 = hello world
```

Язык Python можно использовать для реализации дополнительных команд, проверки различных условий в программе, работе с потоками (они представлены в виде специальных объектов в поддерживаемом GDB Python API).

Приведем в качестве примера фрагмент кода, демонстрирующего возможность создания сложных сценариев отладки, открывающих на экране дополнительные окна с графической информацией для визуализации состояния отслеживаемых объектов:

```
import gdb, threading, Queue, gtk, glib, os, pynotify

(read _ pipe, write _ pipe) = os.pipe()

event _ queue = Queue.Queue()

def send _ to _ gtk(func):
    event _ queue.put(func)
    os.write(write _ pipe, 'x')
def on _ stop _ event(event):
    n = pynotify.Notification('Your program stopped in gdb')
    n.show()
class GtkThread(threading.Thread):

def handle _ queue(self, source, condition):
    global event _ queue
    os.read(source, 1)
    func = event _ queue.get()
    func()
```

Отладочная печать и многоядерные системы. Как упомянуто ранее, даже такое простое средство, как отладочная печать, имеет свои тонкости реализации в случае многоядерных систем и в особенности many-core, поскольку в последних лишь некоторые ядра имеют непосредственный доступ к портам ввода/вывода. По этой причине для реализации отладочной печати одно из таких ядер периодически опрашивает очередь отладочных сообщений от остальных ядер. При этом такое взаимодействие надежнее реализовать посредством использования общей памяти.

Для аварийной остановки программы в случае ошибочных ситуаций (например, при невыполнении условия в макросе assert) можно зарезервировать специальный тип сообщений. Для сообщений разного типа и от разных счетных ядер используется цветная печать.

Трассировка программ. При трассировке многопоточных программ под эмулятором фиксируются и выводятся на экран или в журнал происходящие на всех ядрах значимые события: исполняемая машинная инструкция, измененные значения регистров и т. д.

Приведем пример короткого фрагменты такой трассы для ядра процессора с архитектурой Microblaze:

```
-> R1=8EFFFEB8 '?'
[PC=80002F84]Dealing with jump to 800030F8, type=0, delay=was
[PC=800030F8; SREG=00000000] 30790001 addik r3, r25, 1 (imm=1)
// 00000001
-> R3=8000B1B9 '?'
[PC=800030FC; SREG=00000000] E0C3FFFF lbui r6, r3, -1 (imm=-1)
// FFFFFFFF
-> address=8000B1B8
-> R6=0000006B 'k'
[PC=80003100; SREG=00000000] 12F9B000 addk r23, r25, r22
-> R23=00000004 '?'
[PC=80003104; SREG=00000000] A48600FF andi r4, r6, 255 (imm=FF)
// 000000FF
-> R4=0000006B 'k'
[PC=80003108; SREG=00000000] A8E40025 xori r7, r4, 37 (imm=25, '%')
// 00000025
-> R7=0000004E 'N'
[PC=8000310C; SREG=00000000] 10A40000 addk r5, r4, r0
-> R5=0000006B 'k'
[PC=80003110; SREG=00000000] BE270010 bneid r7, 16 (imm=10)
// 80003120
```

В случае если требуется анализировать трафик внутрикристальной шины при обменах данными между ядрами, с помощью дополнительной опции эмулятор может предоставить и эту информацию:

```
### SPbus send from 0x0000 to 0x0100
### SPbus on 0x0100 received type 0xC1: 0x00000000 & 0x80003024 to 0x0 from core 0x0000
### IC on core 0x0100 received a packet ictype 0x0
### Core 0x0000 IC (C1) data sent: 0x00000000 & 0x80003024 laddr 0x00000008 to 0x0100
### SPbus send from 0x0100 to 0x01FF
### Core 0x0100 IC (C0) data sent: 0x00000000 & 0x00000000 laddr 0x00000008 to 0x01FF
### SPbus send from 0x0000 to 0x0200
### SPbus on 0x0200 received type 0xC1: 0x00000000 & 0x80003024 to 0x0 from core 0x0000
### IC on core 0x0200 received a packet ictype 0x0
### Core 0x0000 IC (C1) data sent: 0x00000000 & 0x80003024 laddr 0x00000008 to 0x0200
### SPbus send from 0x0200 to 0x02FF
### Core 0x0200 IC (C0) data sent: 0x00000000 & 0x00000000 laddr 0x00000008 to 0x02FF
### SPbus send from 0x0000 to 0x0300
### SPbus on 0x0300 received type 0xC1: 0x00000000 & 0x80003024 to 0x0 from core 0x0000
### IC on core 0x0300 received a packet ictype 0x0
### Core 0x0000 IC (C1) data sent: 0x00000000 & 0x80003024 laddr 0x00000008 to 0x0300
```

Эмулятор может отслеживать и выводить на экран любые виды событий. Выводимая информация на экран является низкоуровневой и для ее интерпретации требуются заметные трудозатраты. Однако следует заметить, что при отладке системного ПО такая низкоуровневая информация является наиболее значимой.

Профилирование программного обеспечения систем класса many-core

Основной целью профилирования программы является получение информации для оценки ее итоговой эффективности. Если полученная производительность заметно отличается от расчетной, то информация от профилировщика является первичным материалом для оптимизации такой программы.

Следует отметить, что вследствие неоднородной слабосвязанной памяти программ и данных расчетное время исполнения кода может быть гораздо больше, чем сумма рабочих тактов всех ядер, деленная на их число. Иными словами, реальная производительность many-core-систем может быть значительно ниже их пиковой производительности.

Основные методы сбора данных при профилировании

Существует два основных режима сбора данных при профилировании: точный синхронный и приближенный, который на реальном оборудовании функционирует как асинхронный. Первый способ является точным, однако он требует некоторой модификации кода программы и может заметно влиять на время ее выполнения. По этой причине для высокопроизводительных систем чаще применяют второй способ.

При приближенном способе в системе время от времени проводится анализ того, чем занято то или иное счетное ядро. Информация об этом сохраняется в специальных таблицах, которые периодически обрабатываются и сохраняются для постобработки. Поскольку при приближенном подходе анализируется не каждый шаг исполнения программы, то следует проявлять осторожность и избегать простых детерминированных схем. Например, если некоторый внутренний цикл программы занимает 1500 тактов, а профилировщик засекает каждый 1000-й такт, то он зафиксирует пребывание программы лишь в трех точках цикла и не заметит те подпрограммы, которые вызывались между ними. Чтобы избежать подобных явлений, при реализации в эмуляторе следует использовать два простых числа, одно из которых достаточно велико и определяет период сохранения, а второе определяет период наблюдения. Таким образом, фазу наблюдения достаточно просто сдвигать на единицу после каждого сохранения профилировочной информации.

Встроенные в эмулятор средства профилирования

Для профилирования many-core-систем на кристалле проще всего использовать реализацию профилировщика, встроенного в эмулятор. Этот способ хорош тем, что в эмуляторе имеется полная информация о состоянии всех ядер и ее можно без труда собрать и обработать, преобразовав в удобный для чтения вид. При текстовом представлении профилировочной информации в рассматриваемом эмуляторе используется простой формат markdown, а информация о времени, потраченном при исполнении процедур, упорядочивается по убыванию.

В качестве демонстрационного примера полезности использования профилировщика приведем два результата его работы — до и после оптимизации простого интерпретатора языка JavaScript. До оптимизации значительную часть времени занимало исполнение функции malloc:

```
Routine                                  Count
-------------------------------          ----------------------
malloc                                   79 %  (2239738/2017407)
jsS_insert                                4 %  (124652/116889)
strcmp                                    2 %  (70485/64809)
insert                                    2 %  (62628/58417)
l1                                        2 %  (61690/50122)
jsR_defproperty                           1 %  (33453/31328)
memset                                    1 %  (33372/25884)
js_defproperty.constprop.125              1 %  (21395/19667)
jsV_setproperty                           1 %  (19510/18515)
js_malloc                                 1 %  (14440/13620)
kvprintf                                  0 %  (12550/11076)
jsV_toobject                              0 %  (12095/11293)
```

После оптимизации этой функции скорость работы данного интерпретатора увеличилась приблизительно в пять раз:

```
Routine                                  Count
-------------------------------          ----------------------
jsS_insert                               20 %  (124682/116855)
strcmp                                   11 %  (70481/64859)
insert                                   10 %  (62636/58310)
l1                                       10 %  (61690/50122)
jsR_defproperty                           5 %  (33474/31322)
memset                                    5 %  (33356/25864)
js_defproperty.constprop.125              3 %  (21371/19635)
jsV_setproperty                           3 %  (19452/18438)
my_malloc                                 3 %  (15523/14493)
js_malloc                                 2 %  (14455/13589)
jsG_freeobject                            2 %  (14224/13531)
kvprintf                                  2 %  (12541/11041)
js_defaultalloc                           2 %  (12253/11122)
jsV_toobject                              2 %  (12093/11326)
```

Использование графических диаграмм

Наглядное представление о распределении времени вычислений в программе по программным секциям дают разного рода графические диаграммы, причем существует значительная свобода в выборе такого представления. Часто используется, например, представление в виде столбцов, фрагменты которых, пропорциональные по размеру времени счета, выделяют цветом. Если ряд таких столбцов для расчетных конфигураций с разным числом ядер расположить по горизонтали, то получается достаточно наглядное представление о том, как масштабируются различные аспекты параллельной программы.

Расширения профилировщика

Низкоуровневая информация от профилировщика не всегда удобна, поэтому часто используют и более высокоуровневые инструментальные средства. Примером такого средства в случае использования языка C++ может служить класс Timer.

Работа с рассматриваемым таймером выглядит следующим образом: объявляются именованные объекты класса Timer, а внутри каждого из блоков кода, время исполнения которых в течение работы всей программы нужно просуммировать, используется макрос WITH_TIMER(...).

Для засечек времени в профилировочных таймерах следует использовать аппаратные счетчики, которыми снабжают современные процессоры. При этом необходимо учитывать, если в системе используется вытесняющая многозадачность и логические потоки мигрируют с одного ядра на другое, то и значения аппаратных счетчиков могут считываться с разных ядер, что приведет к ошибочному результату. Как следствие, даже для десктопных многоядерных систем следует избегать использования наивных реализаций, состоящих из пары ассемблерных вставок, а использовать специализированные библиотеки.

Заключение

На практическом примере показано, что на основе традиционных программных средств и технологий отладки и профилировки с открытым исходным кодом, разработанных для отладки прикладных программ, исполняемых на одном вычислительном ядре, можно построить инструментарий, включающий эмулятор, отладчик и профилировщик для систем класса many-core, содержащих сотни асинхронно взаимодействующих вычислительных ядер.

В настоящей статье предложены конкретные приемы, позволяющие учесть факторы физического параллелизма many-core-систем в разрабатываемом инструментарии, как на уровне архитектуры отладочного программного обеспечения, так и с помощью разного рода расширений к существующим инструментам. Показано, как грамотное использование этих приемов позволяет решать вопросы локализации ошибок на всех уровнях отладки прикладного программного обеспечения.

Особое внимание в статье уделено практичности с точки зрения пользователя разрабатываемых средств отладки и профилировки прикладных программ, показано, что рассматриваемый в статье подход позволяет существенно снизить трудоемкость разработки нового и переноса унаследованного программного обеспечения на перспективные программно-аппаратные платформы за счет использования традиционных, принятых в open source-сообществе интерфейсов и представлений.

Список литературы

1. **Calray** MPPA Manycore processor. URL: http://www.kalrayinc.com/IMG/pdf/FLYER_MPPA_MANYCORE.pdf

2. **Cisco** CloudFlare processor. URL: https://blogs.cisco.com/datacenter/cisco-cloud-scale-asic-switches-get-a-2-year-advantage-over-merchant-silicon-switches

3. **Елизаров С. Г., Лукьянченко Г. А., Корнеев В. В.** Технология параллельного программирования экзафлопсных компьютеров // Программная инженерия. 2015. № 7. С. 3—10.

4. **Информация** о принципах работы green threads. URL: https://c9x.me/articles/gthreads/intro.html

5. **Информация** об отладчике для графпроцессоров CUDA. URL: http://docs.nvidia.com/cuda/cuda-gdb/index.html#cuda-gdb-extensions

6. **Информация** об устройстве ответной части отладчика GDB. URL: https://sourceware.org/gdb/onlinedocs/gdb/Remote-Stub.html#Remote-Stub

7. **Средство** мультиплексирования последовательного канала KDMX. URL: https://elinux.org/Kdmx

8. **Принципы** реализации удаленных вызовов функций RPC и RMI. URL: https://cseweb.ucsd.edu/classes/wi17/cse291-d/applications/ln/lecture3.html

Debugging Software for Multi-Core Processors

S. G. Elizarov, elizarov@physics.msu.ru, **G. A. Lukyanchenko,** egorxe@yandex.ru,
D. S. Markov, markovds@maltsystem.com, **V. A. Roganov,** radug-a@ya.ru,
Lomonosov Moscow State University, Moscow, 119991, Russian Federation

Corresponding author:

Roganov Vladimir A., Lomonosov Moscow State University, Moscow, 119991, Russian Federation,
E-mail: radug-a@ya.ru

Received on September 18, 2017
Accepted on October 04, 2017

In this article we discuss our experience in designing a set of debugging tools for a novel many-core CPU consisting of hundreds of computational cores communicating with each other asynchronously. We describe an approach based on which we have created an emulator, debugger and profiler for such many-core system. As a base of this software toolset we've used standard open source debugging and profiling software created for use in traditional programming environment. The main reason of debugging software complexity for many-core architectures is that all interactions between computational cores of the CPU (synchronization, data exchange, etc.) take place deeply inside the chip, so they cannot be intercepted using classical techniques designed for traditional single- and multi-core CPUs. Execution interception in many-core systems on a chip is complicated even more by lack of access to IO ports from the most of computational cores. This fact complexifies implementation of even the simplest debug methods (printing to console, for example) on such systems. In this article we suggest several methods which allowed us to create debugging tools taking into account physical parallelism on all execution levels. We demonstrate how applying these methods allows us to localize bugs on all software levels and to optimize system performance. In the article we advert to practical aspects of using standard open source software for debugging and performance profiling of parallel software made for prospective many-core processors.

Keywords: *debugging technologies, many-core processors, systems on a chip, hardware emulators, profiling tools*

For citation:

Elizarov S. G., Lukyanchenko G. A., Markov D. S., Roganov V. A. Debugging Software for Multi-Core Processors, *Programmnaya Ingeneria,* 2017, vol. 8, no. 12, pp. 531—542.

DOI: 10.17587/prin.8.531-542

References

1. **Calray** MPPA Manycore processor, available at: http://www.kalrayinc.com/IMG/pdf/FLYER_MPPA_MANYCORE.pdf

2. **Cisco** CloudFlare processor, available at: https://blogs.cisco.com/datacenter/cisco-cloud-scale-asic-switches-get-a-2-year-advantage-over-merchant-silicon-switches

3. **Elizarov S. G., Lukyanchenko G. A., Korneev V. V.** Tehnologija parallel'nogo programmirovanija jekzaflopsnyh komp'juterov (Exascale Computer Parallel Programming Technology), *Programmnaya Ingeneria,* 2015, no. 7, pp. 3—10 (in Russian).

4. **Information** on the principles of work green threads, available at: https://c9x.me/articles/gthreads/intro.html

5. **Information** on the debugger for GPUs CUDA, available at: http://docs.nvidia.com/cuda/cuda-gdb/index.html#cuda-gdb-extensions

6. **Information** on the debugger mate unit GDB, available at: https://sourceware.org/gdb/onlinedocs/gdb/Remote-Stub.html#Remote-Stub

7. **Segment** channel multiplexing means KDMX, available https://elinux.org/Kdmx

8. **Principles** of implementing remote function calls RPC and RMI, available at: https://cseweb.ucsd.edu/classes/wi17/cse291-d/applications/ln/lecture3.html

УДК 519.685.4 DOI: 10.17587/prin.8.543-555

А. П. Соколов, канд. физ.-мат. наук, доц., e-mail: alsokolo@bmstu.ru,
Московский государственный технический университет имени Н. Э. Баумана,
А. Ю. Першин, аспирант, e-mail: tony.pershin@gmail.com, mmap@leeds.ac.uk,
Университет Лидса, Лидс, Великобритания

Программный инструментарий для создания подсистем ввода данных при разработке систем инженерного анализа

Представлен подход к проектированию и разработке программных подсистем ввода исходных данных при создании систем инженерного анализа, в основе которого лежит применение разработанного авторами специализированного формата данных и его интерпретатора. Приведены примеры формируемых динамических графических пользовательских интерфейсов для определения исходных данных для вычислительных задач, решенных в рамках выполнения ряда НИР и ОКР, включавших необходимость разработки программного обеспечения.

Ключевые слова: *разработка подсистем ввода данных, динамические графические пользовательские интерфейсы, автоматизация процессов ввода данных, интерпретаторы текстовых данных, технологии разработки систем инженерного анализа, распределенные программные системы, модель — вид — контроллер*

Введение

В статье представлен подход и созданные авторами программные средства, упрощающие разработку графических пользовательских интерфейсов (*graphical user interface*, далее — GUI) прикладного программного обеспечения (ПО) для математического моделирования физических процессов в целях решения различных задач прикладных и научных исследований и проведения сложных инженерных расчетов. Программное обеспечение для решения такого типа задач называют программным обеспечением инженерного анализа, а специализированные системы такого класса — системами инженерного анализа (*computer-aided engineering*, далее — CAE-системы) [1]. Отдельные идеи, реализованные в представленном подходе, заимствованы из области разработки крупных распределенных промышленных программных систем (а именно BAAN ERP, SAP R/3).

Причиной создания представленного подхода стала необходимость упрощения, стандартизации и систематизации процессов разработки программного обеспечения инженерного анализа, которые оказались востребованы в рамках многочисленных научно-исследовательских (НИР) и опытно-конструкторских (ОКР) работ, выполняемых авторами в МГТУ им. Н. Э. Баумана, в области математического моделирования различных физических процессов. Следует также отметить, что на настоящее время одной из важнейших задач, которые стоят перед современной Россией, является создание отечественного программного обеспечения [2]. Проекты этого направления активно поддерживают государственные структуры и фонды, такие как Министерство образования и науки РФ, Министерство промышленности и торговли РФ, Фонд перспективных исследований, Фонд "Сколково", Фонд содействия развитию малых форм предприятий в научно-технической сфере и др. Предполагается, что развиваемый авторами подход станет заделом для создания конкурентоспособных специализированных отечественных CAE-систем с учетом наличия на рынке известных зарубежных аналогов (ANSYS, ABAQUS, MatLab и др.).

Опишем преимущества, которые дает применение представленного подхода при изменении требований к разрабатываемому ПО.

Традиционный жизненный цикл разработки ПО инженерного анализа, как и любого другого коммерческого ПО, в рамках каскадной технологии разработки включает в себя следующие стадии [3]: системный анализ, анализ требований, проектирование, кодирование, тестирование и сопровождение (рис. 1, *а*), которые должны быть пройдены однократно. Для других технологий разработки ПО (например, V-образная модель, модель быстрой разработки, спиральная модель, компонентно-ориентированная модель, модель экстремального программирования в рамках методологии Agile и пр.) характерны в том или ином виде те же стадии, но реализуемые в цикле с существенно укороченной продолжительностью итерации [3—5]. Выбор технологии разработки ПО обычно определяется масштабом программного проекта. На каждой из стадий возможны "откаты" на более ранние стадии, что, как правило, обусловлено необходимостью изменения

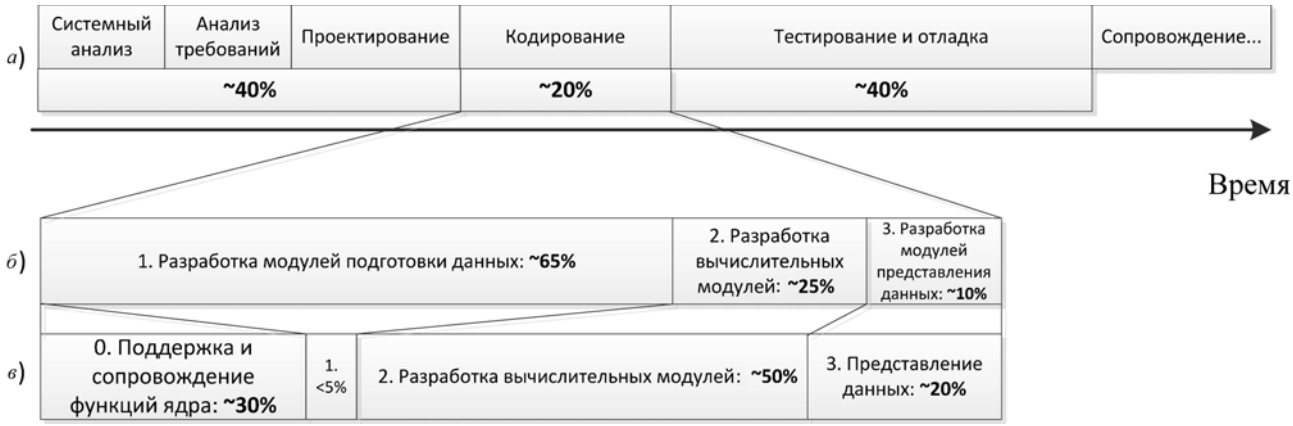


Рис. 1. Стадии разработки ПО инженерного анализа с указанием оценок трудозатрат на каждую:

а — традиционный жизненный цикл ПО; *б* — в процессе кодирования не применяются технологии динамически формируемых GUI; *в* — в процессе кодирования применяются технологии динамически формируемых GUI

исходных требований. Распределение трудозатрат на каждую стадию создания коммерческого ПО инженерного анализа определяется зрелостью команды разработки, применением программных инструментариев автоматизации разработки программного обеспечения (*computer-aided software engineering* — CASE) [3] и, в частности, применением технологий динамически формируемых GUI. На рис. 1, *б* представлено распределение трудозатрат на создание программных модулей различных типов при разработке ПО инженерного анализа без применения технологий динамически формируемых GUI, а на рис. 1, *в* — с применением таких технологий. Процентные соотношения, представленные на рис. 1, получены на основе опыта разработки и эксплуатации программного обеспечения "Распределенная вычислительная система GCD" (РВС GCD), а также сведений, представленных в работе [3]. Для примера прикладного использования представленного в настоящей статье подхода приведены данные по функциональным возможностям, созданным в рамках программного обеспечения РВС GCD. Детальная информация о системе представлена на официальном сайте: http://gcad.bmstu.ru (РВС GCD, GCAD v. 4) [6, 7]. Количественные данные о функциональных возможностях РВС GCD представлены в табл. 1.

Отличительной особенностью ПО инженерного анализа является возможность разделения состав-

Таблица 1

Сведения о функциональных возможностях РВС GCD на 11.07.2017 г. Срок активной разработки и эксплуатации РВС GCD при реализации прикладных НИР, ОКР в МГТУ им. Н. Э. Баумана с 2009 г.

Показатель	Абсолютное значение	Значение от общего числа функций, %	Число реально реализованных функций	Трудозатраты при использовании динамических GUI, %	Грубая оценка числа возможно реализуемых функций	Трудозатраты на реализацию аналогичных задач без динамических GUI, %
Системные функции ядра	62	13,5	62	31,8	0	—
Динамические плагины, автоматически формирующие GUI, модули подготовки данных (модули группы 1)	268	57,8	0	0 (не более 5 %*)	268	66,8
Модули обработки данных (модули группы 2)	97	20,9	97	**49,7**	97	**24,2**
Модули представления данных (модули группы 3)	36	7,8	36	18,5	36	9
Итого	463	100	195	100	401	100
* Требуется время на формальное определение исходных данных в специальном формате.						

ляющих его программных модулей на три основные группы:

1) модули подготовки данных;
2) модули обработки данных;
3) модули представления результатов.

В PBC GCD активно используются методы автоматической генерации графических пользовательских интерфейсов. На основе этой системы были реализованы ряд НИР и ОКР, связанных с разработкой ПО инженерного анализа. Как видно из представленных данных, на решение основных задач НИР и ОКР, связанных с разработкой программных модулей группы 2, с использованием PBC GCD было потрачено около 50 % времени исследователей, тогда как без использования данной системы это время составило бы всего около 25 % общего времени разработки (см. рис. 1).

Представленный в настоящей статье подход позволяет вносить изменения в требования к перечню и типам входных данных широкого класса инженерных задач на любой стадии разработки соответствующего ПО без необходимости изменения исходного кода, что существенно ускоряет процесс разработки.

Качество разрабатываемого ПО определяется множеством критериев, а именно: функциональной исправностью; функциональной совместимостью; безопасностью; точностью; соответствием стандартам; удобством изучения, использования и сопровождения; понятностью; эффективностью, надежностью и др. [2].

В силу высокой трудоемкости разработки модулей группы 2 [7—12], обусловленной необходимостью реализации различных сложных вычислительных методов, включая численные методы решения систем дифференциальных, интегральных, алгебраических уравнений и др., трудоемкостям разработки модулей групп 1 и 3 зачастую не уделяют должное внимание.

Качественная программная реализация вычислительных методов для решения задач прикладных и научных исследований и проведения сложных инженерных расчетов, т. е. разработка модулей группы 2, может не предполагать существенного объема исходных кодов [13], однако требует серьезной математической подготовки разработчика. Качественная разработка программных модулей групп 1 и 3 требует, как правило, написания больших объемов исходных кодов и высокой квалификации разработчика в области компьютерных наук [6, 7, 14].

Объем кода зависит от выбранного языка программирования. Например, известно примерное соответствие числа строк кода (*Lines Of Code* — LOC) для различных языков программирования, приведенное к обобщенным функциональным указателям (*Feature Points* — FP) [3]:

- Ассемблер — 320 LOC = 1FP;
- C — 128 LOC = 1FP;
- C++ — 64 LOC = 1FP;
- Delphi — 29 LOC = 1FP;
- Haskell — 38 LOC = 1FP.

В качестве приоритета для ПО инженерного анализа, в особенности для программных модулей группы 2, остается быстродействие, т. е. чем более высок уровень языка программирования, тем более "медленным" оказывается код на его основе, поэтому ученые часто продолжают использовать достаточно низкоуровневые языки программирования. Для модулей групп 1 и 3 не требуется реализация высокопроизводительного кода. Таким образом, для качественной разработки ПО инженерного анализа разработчик должен активно использовать различные языки программирования, что зачастую возможно только при командной (коллективной) разработке.

Однако при выполнении НИР и ОКР, которые проводят в вузах, в рамках проектов на договорной основе в работе над созданием ПО инженерного анализа обычно задействованы специалисты, получившие хорошую базовую математическую подготовку на кафедрах численного анализа, вычислительной математики и подобных им, на которых подготовка в области информационных технологий не является основным приоритетом. В результате в таком коллективе качественная реализация программных модулей групп 1 и 3 невозможна. Как следствие, многие вузовские исследователи зачастую вынуждены при ограничениях по времени выполнения проекта использовать готовые коммерческие программные пакеты инженерного анализа, такие как MatLab, Maple, Mathematica и пр., в рамках которых уже реализованы соответствующие функциональные возможности. Такой подход позволяет реализовать задачи конкретной работы, но существенно ограничивает возможности создания задела для разработки собственного ПО и тем более не может быть основной для коммерциализации создаваемых научно-технических решений.

Цель настоящей статьи — представление авторской разработки эффективного (с позиций предъявляемых далее требований) программного инструментария (средства) для проектирования и разработки гибких динамических адаптивных программных подсистем ввода исходных данных, автоматизирующего процессы построения графических пользовательских интерфейсов для программных модулей обработки данных.

Следует отметить, что задача автоматического построения GUI не является новой и рассматривалась многими исследователями и разработчиками ПО [15—17]. Однако несмотря на широкую известность различных подходов к решению этой задачи [18—21], генерация GUI для ПО инженерного анализа чаще реализуется зарубежными организациями (см., например, патенты [22—24]), тогда как отечественные разработки в этой области либо являются закрытыми [25], либо не получили широкого распространения [26—29]. Существует большое число специализированных многофункциональных (web-ориентированных и др.) программных средств построения GUI общего назначения (React JS, Angular, возможности систем управления контен-

том, библиотеки GTK+, QT, QML, Tcl/Tk, QML и др.). Как правило, такие универсальные средства могут применяться только разработчиками ПО и не используются для разработки подсистем ввода входных данных для ресурсоемких систем инженерного анализа.

1. Постановка задачи

С учетом обозначенных выше особенностей разработки ПО инженерного анализа и известных авторам особенностей постановок задач прикладных и поисковых исследований, реализуемых в вузах, были определены требования к программному инструментарию, призванному сократить трудозатраты на ввод и подготовку исходных данных для широкого класса вычислительных задач. Расширение потенциальной области приложения разработанного программного инструментария на уровне его проектирования было сделано намеренно для упрощения в перспективе процессов внедрения разрабатываемого на его основе ПО.

Далее используем следующие обозначения: *прикладная программа* — программа, реализующая решение вычислительной задачи, входящей в состав системы инженерного анализа, для которой требуется создание GUI для ввода исходных данных; *автоматическое построение GUI* — построение графических форм ввода исходных данных для *прикладной программы* без участия человека с использованием программы-генератора GUI (далее — *GUI-генератор*) и файла исходных данных в специальном формате; *метаданные* — список входных параметров и их текущие значения в рамках выполнения прикладной программой вычислительной задачи.

Определенные требования к программному инструментарию.

Т1. Программный инструментарий должен включать в свой состав *GUI-генератор*, обеспечивающий автоматическое построение GUI на основе заранее определенного списка входных параметров (*метаданных*).

Т2. Для составления списка входных параметров должен быть разработан специализированный текстовый формат данных.

Т3. Специализированный текстовый формат файла исходных данных должен быть ориентирован, в первую очередь, на хранение скалярных параметров различных типов, включая: строки; вещественные и целые числа; множества; логические значения; ссылки на файлы; ссылки на данные из внешних источников (файлы, записи из таблиц баз данных), а также небольших статических массивов.

Т4. Список поддерживаемых типов параметров не должен быть фиксирован, должна оставаться возможность его расширения.

Т5. Программа-генератор GUI должна зависеть не от конкретных *метаданных*, а лишь от грамматики специализированного текстового формата, определяющей произвольные *метаданные*.

Т6. Подготовка списка входных параметров должна быть доступной для неподготовленного специалиста, не владеющего навыками программирования, в том числе для специалистов, представляющих заказчика разрабатываемой *прикладной программы*.

Т7. При изменении исходного списка входных параметров соответствующий GUI должен быть перестроен автоматически.

Т8. Процесс построения GUI должен осуществляться во время выполнения разрабатываемой *прикладной программы*, использующей программный инструментарий.

Т9. Программный инструментарий должен обеспечивать возможность автоматического определения параметров различных типов (строки, массивы, целые числа, множества и т. д.).

10. Специализированный текстовый формат данных, обеспечивающий возможность определения *метаданных*, должен предоставлять возможность группировки входных параметров по их назначению.

Т11. Должна быть обеспечена возможность редактирования списка входных параметров независимо от программы-генератора GUI.

Т12. Программный инструментарий должен иметь возможность встраивания в web-ориентированные приложения, приложения для мобильных платформ и приложения для операционных систем семейств Windows, Linux, macOS. Для поддержки соответствующих платформ должен быть разработан свой GUI-генератор.

Таким образом, была поставлена задача разработки программного инструментария, удовлетворяющего представленным выше требованиям к автоматизации процесса генерации графических пользовательских интерфейсов, которые необходимы для определения значений входных параметров для программных модулей обработки данных, обеспечивающих решение различных вычислительных задач, включая задачи инженерного анализа.

2. Основы подхода и разработка программного инструментария

Для автоматического построения GUI необходимы источники данных, которые могли бы быть использованы для формирования экранных форм и управляющих элементов в автоматическом режиме. Хорошо себя зарекомендовали и активно применяются для этой цели XML-технологии [22]. Такие технологии позволяют предварительно определить XML-схему (в формате DTD или XSD), с помощью которой может быть зафиксирован формат файла для описания GUI, включая экранные формы и управляющие элементы. Несмотря на широкие возможности XML-технологий, подготовка XML-документа, тем более соответствующего заранее определенной XML-схеме, возможна только при наличии и с помощью соответствующих XML-редакторов или программным образом для частных случаев. Это обстоятельство существенно усложняет процедуру создания таких форм неподготовленным специа-

листом-исполнителем и тем более специалистом, представляющим заказчика. Еще одним известным текстовым форматом, который может применяться для достижения поставленных целей, является формат JSON [30]. Этот формат намного удобнее, однако также имеет излишние функциональные возможности с точки зрения определения входных данных для вычислительных задач, включая задачи инженерного анализа.

Таким образом, для описания списков входных параметров было принято решение использовать более ограниченный, но существенно более простой текстовый формат. За основу был взят формат INI [31], который был существенно доработан с учетом его ориентации на ранее определенные требования. В результате был создан новый формат aINI® [32, 33] (требование Т2). Формат позволил создавать файлы исходных данных специалистами, не владеющими навыками программирования, что было определено требованием Т6. Для разбора файла в формате aINI был создан синтаксический анализатор (aINI-парсер).

В целях предоставления возможности формирования различных управляющих графических элементов для каждого входного параметра, определенного в файле входных параметров в формате aINI, должен быть автоматически (с помощью aINI-парсера) определен его тип. Реализация такой функции оказалась возможной с использованием различных синтаксических конструкций при определении каждого параметра. Каждому типу параметра ставится в соответствие конкретный управляющий графический элемент (поле ввода EditBox, выбор из списка ListBox, альтернативный выбор RadioGroup, множественный альтернативный выбор CheckBox и др.).

Конкретная графическая подсистема может быть реализована как на базе специализированных библиотек, используемых при разработке приложений для операционных систем семейства Windows, Linux, macOS (например, на основе кроссплатформенной библиотеки Qt), так и на основе web-технологий.

После синтаксического разбора aINI-файла и определения типа каждого параметра aINI-парсером может быть автоматически, посредством соответствующего GUI-генератора, построен графический пользовательский интерфейс под ту или иную платформу или web-ориентированный интерфейс.

Определение текущих значений каждого параметра предусмотрено форматом aINI, поэтому значения параметров могут быть автоматически подставлены в сформированную экранную форму. При внесении изменений в экранной форме соответствующие изменения также синхронно вносятся в исходный aINI-файл.

Таким образом, в **состав программного инструментария** для соответствующей платформы вошли:

- *aINI-парсер;*
- *GUI-генератор*;
- специальный класс, обеспечивающий создание объектов для сохранения списков входных параметров с их значениями (далее — класс *AnyMap*);
- функции, реализующие вспомогательные задачи программного инструментария;
- описание синтаксиса текстового aINI-формата.

Процесс применения созданного программного инструментария представлен на рис. 2 в форме диаграммы потоков данных (*data flow diagram*) [2].

Прокомментируем обозначения, используемые на рис. 2, с указанием соответствующих, подлежащих выполнению требований.

Р0. Процедура создания и редактирования списка входных данных осуществляется вручную с использованием доступных текстовых редакторов в формате aINI, что обеспечивает выполнение требований Т6, Т11.

Р1. "Чтение" (автоматическая программная интерпретация) aINI-файла осуществляется aINI-парсером, обеспечивающим синтаксический разбор

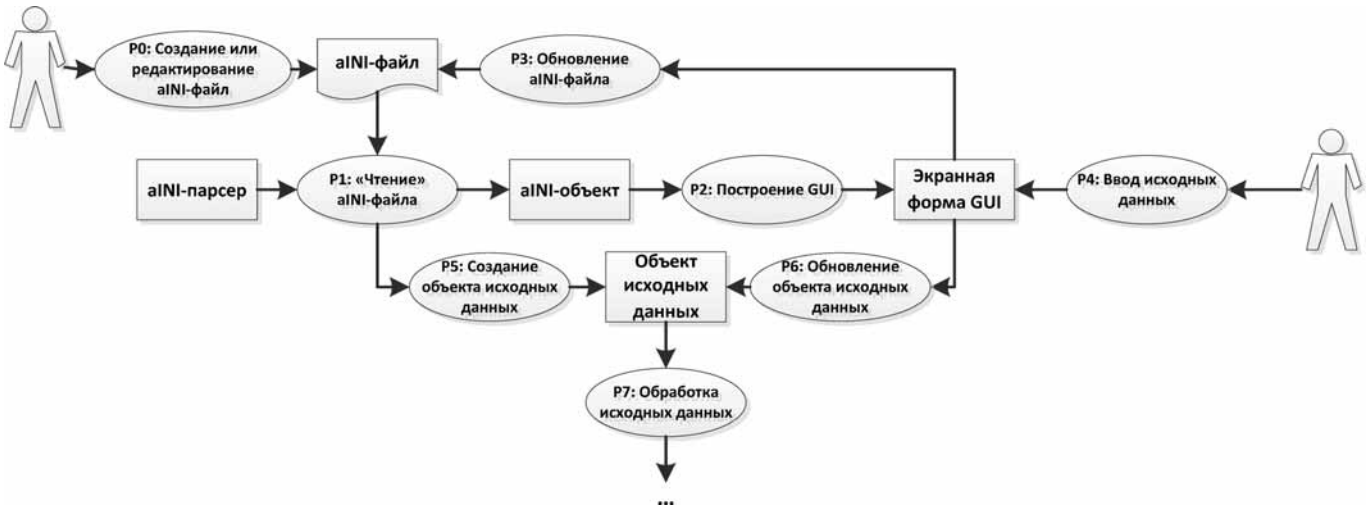
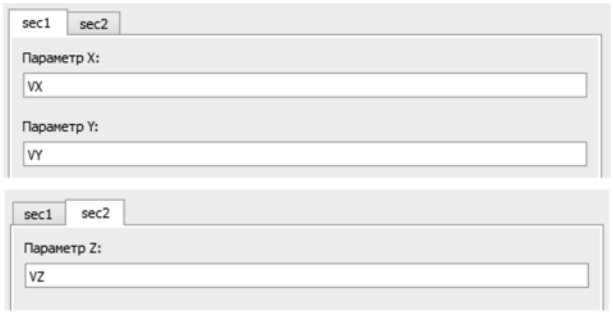


Рис. 2. Диаграмма потока данных при построении и эксплуатации GUI на основе созданного программного инструментария и формата aINI

```
[sec1]
X=VX // Параметр X
Y=VY // Параметр Y

[sec2]
Z=VZ // Параметр Z
```

а)



б)

Рис. 3. Листинг aINI-файла (*а*) и соответствующие автоматически сформированные экранные формы, построенные с использованием библиотеки Qt (*б*)

конструкций aINI-файла, включая типы каждого параметра (требование Т9), и формирование aINI-объекта.

Р2. Построение GUI осуществляется с помощью платформозависимого (с учетом требования Т12) GUI-генератора, на вход которого подается aINI-объект во время выполнения прикладной программы, использующей соответствующий aINI-файл в качестве входных данных (требования Т1, Т8).

Р3. Автоматическое обновление aINI-файла согласно внесенным пользователем изменениям в значения входных параметров в экранной форме. Такой подход в совокупности с Р2 позволяет удовлетворить требование Т7.

Р4. Процедура ввода данных включает изменение текущих значений входных параметров и осуществляется в сформированной экранной форме. Количество и типы входных параметров при этом не изменяются.

Р5. Создание объекта класса AnyMap, содержащего все входные параметры и их текущие значения (метаданные), осуществляется для возможности дальнейшей передачи на вход функциям обработки данных или для возможности передачи по сети на удаленный высокопроизводительный узел с целью проведения расчета на нем.

Р6. Обновление объекта исходных данных класса AnyMap осуществляется после изменения значений в экранных формах.

Р7. Непосредственная обработка объекта исходных данных класса AnyMap.

Укажем подзадачи, которые решены при создании программного инструментария:

1) разработана грамматика текстового описательного формата aINI, определяющая возможности хранения списков входных параметров и их значений (метаданных);

2) реализована программа "чтения" (aINI-парсер) созданного текстового формата;

3) реализован программный интерпретатор (GUI-генератор) загружаемых и разбираемых метаданных с их представлением в виде графических пользовательских интерфейсов (GUI).

Простейший пример листинга файла исходных данных в формате aINI представлен на рис. 3, *а*. Соответствующие ему автоматически сформированные

Рис. 4. Автоматически сформированные экранные web-формы на основе aINI-файла, представленного на рис. 3, *а*

с помощью GUI-генератора графические формы ввода данных: для платформы Windows, построенные с использованием библиотеки Qt, представлены на рис. 3, *б*. На рис. 4 даны web-формы (использовался web-ориентированный программный каркас Django, aINI-парсер реализован на языке Python). Чтение представленного документа осуществлялось с помощью aINI-парсера. Синтаксис формата aINI представлен в следующем разделе.

3. Формат aINI

Для обеспечения независимости процесса задания входных данных и процесса их чтения была определена и зафиксирована грамматика файла входных данных в формате aINI (требование Т5). Синтаксис формата aINI представлен рядом синтаксических диаграмм [34]. Документ aINI состоит из секций (рис. 5), имя секции заключают в квадратные

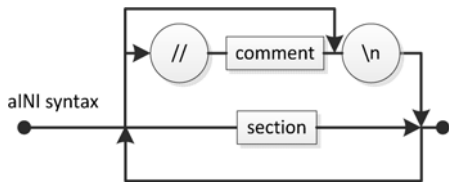


Рис. 5. Файл в формате aINI, состоящий из секций, как и INI-файл

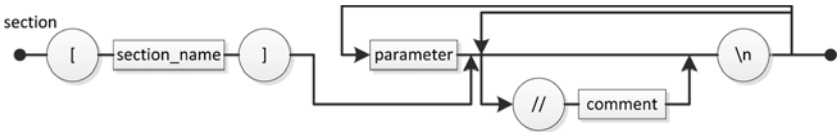


Рис. 6. Синтаксическая диаграмма секции в формате aINI

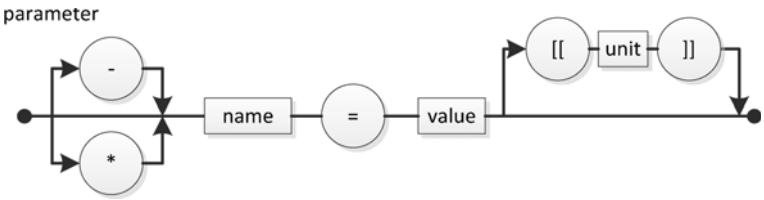


Рис. 7. Синтаксическая диаграмма параметра в формате aINI

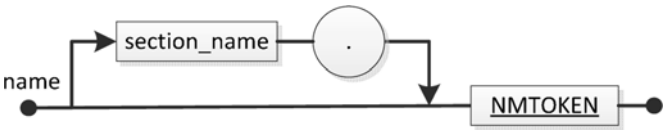


Рис. 8. Синтаксическая диаграмма имени параметра в формате aINI. Полное имя параметра используется только для доступа к нему в рамках других секций этого же aINI-файла

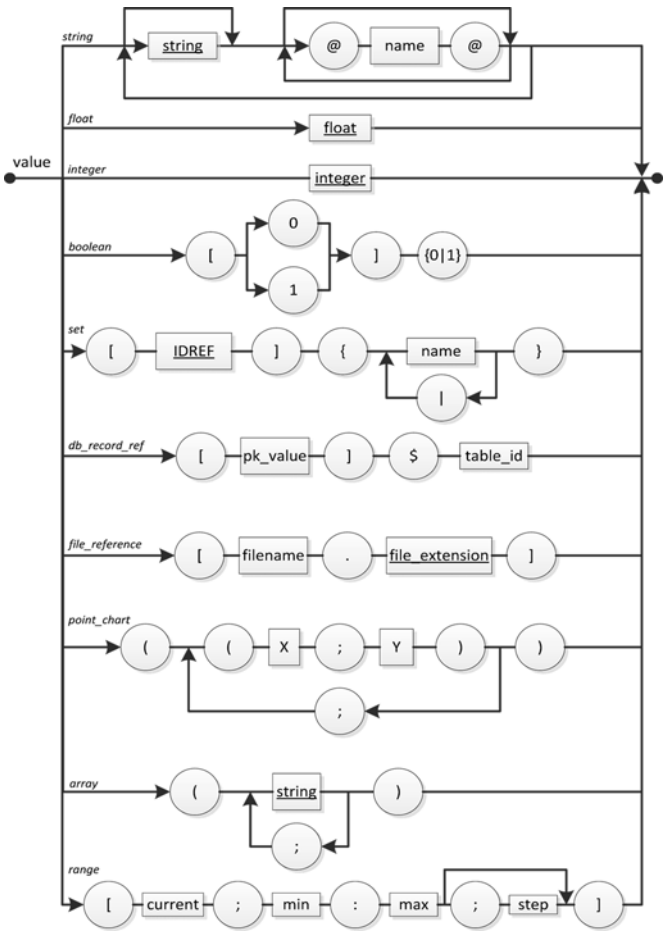


Рис. 9. Синтаксическая диаграмма значения *value* параметра в формате aINI

скобки, секции могут включать произвольное число параметров с комментариями или без них (рис. 6). Параметры имеют специальный формат, синтаксис которого представлен на рис. 7.

Для параметра могут быть указаны комментарий и размерность, а также дополнительный префиксный атрибут, позволяющий указать, что параметр скрыт (‘–’) или обязателен (‘*’) для определения.

Терминальные элементы грамматики aINI-формата на диаграммах подчеркнуты. Идентификатор NMTOKEN (named token) соответствует одноименному понятию в DTD-схемах, применяемых в XML-технологиях, и представляет собой произвольную последовательность символов, не содержащую пробелов и не начинающуюся с цифры.

Область видимости каждого параметра ограничена секцией его определения. Короткое имя параметра определяется его названием *name* и может быть использовано для ссылки на него в рамках той же секции. Полное имя необходимо для ссылки в рамках всего файла. Синтаксическая диаграмма имени параметра представлена на рис. 8. Объединение параметров в секции позволило удовлетворить требование Т10.

Тип конкретного параметра определяется автоматически с помощью aINI-парсера на основе используемого синтаксиса определения его значения *value*, что одновременно обеспечивает выполнение требований Т4 и Т9. Поддерживаются параметры следующих типов (требование Т3):

- строки;
- вещественные числа;
- целые числа;
- логический параметр;
- множество выбора;
- ссылка на запись в таблице удаленной базы данных (БД);
- файловый параметр (ссылка на файл);
- скалярная функция одного аргумента, заданная по точкам (параметр-график);
- строковый массив;
- диапазон.

На рис. 9 представлена синтаксическая диаграмма значения *value* произвольного параметра в рамках формата aINI.

Диаграмма на рис. 10 иллюстрирует синтаксис определения значения параметра *ссылка на запись в таблице базы данных*, которая может указывать наименование БД как явно, так и неявно. Непосредственное подключение к той или иной БД осущест-

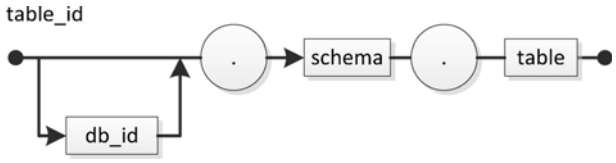


Рис. 10. Синтаксическая диаграмма идентификатора таблицы *table_id* БД в формате aINI

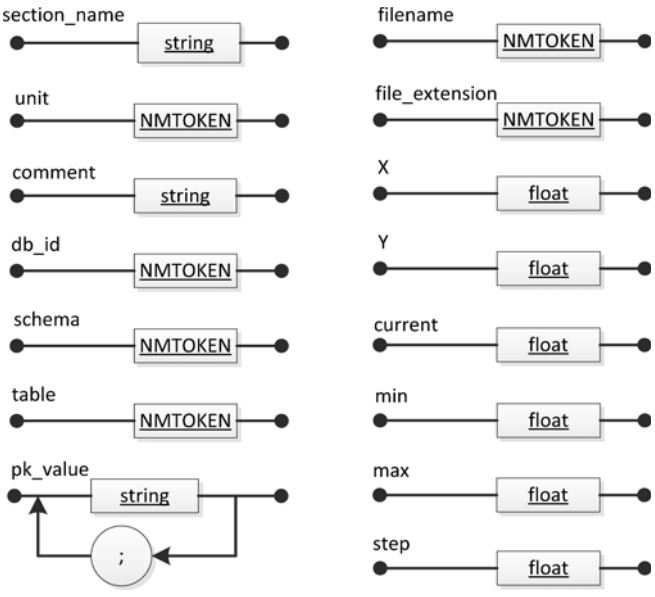


Рис. 11. Синтаксические диаграммы простейших элементов нетерминального алфавита грамматики формата aINI

вляется на уровне интерпретации формата aINI и в процессе формирования на его основе GUI.

Дополнительные синтаксические диаграммы, замыкающие определение грамматики файлов формата aINI, представлены на рис. 11.

Представленная организация файлов в формате aINI позволяет удобно хранить скалярные параметры различных типов (требование Т3).

Более подробно остановимся на зависимых параметрах и параметрах типа *ссылка на запись в таблице базы данных*. Назначение и способы применения параметров остальных типов очевидны.

3.1. Зависимые параметры

Особо следует отметить возможности определения зависимых параметров (см. рис. 9, тип *string*). При определении строкового значения параметра можно сослаться на другой параметр, используя его короткое или полное имя, заключив его в символы @ и включив его в состав значения в виде подстроки. В рамках одного значения можно указать множество ссылок на разные параметры, определенные в разных секциях одного aINI-файла.

Например, может быть удобным определение параметра, задающего формат имени файла результатов работы программы, составив его из значений входных параметров.

Пример использования зависимых параметров представлен на рис. 12, см. третью сторону обложки. Согласно представленному листингу файла входных данных (рис 12, *а*), значение параметра с именем *Z*, определенного в секции *sec2*, зависит от значения параметра *Y*, определенного в секции *sec1*. Зависимые параметры удобно скрывать с помощью использования префиксного спецификатора (‘–’), поскольку их значения уже определены.

3.2. Особенности разбора параметров типа ссылка на запись в таблице БД

При постановке прикладных вычислительных задач может потребоваться определить параметр, ссылающийся на запись в таблице БД. Реализация таких задач необходима при сохранении результатов расчетов с целью проведения дальнейших сравнительных исследований [35]. Такие параметры позволяют осуществлять выбор конкретной записи из связанной таблицы БД. Синтаксическая диаграмма параметров такого типа была представлена ранее на рис. 10.

Получив имя связанной таблицы из aINI-файла (рис. 13, *а*, см. третью сторону обложки), aINI-парсер P2 (см. рис. 2) при наличии подключения к соответствующему источнику данных (базе данных) формирует соответствующее GUI (рис. 13, *б*, см. третью сторону обложки).

Важной особенностью параметров типа *ссылка на запись в таблице базы данных* является дополнительное неявное определение серии новых параметров с автоматически формируемыми новыми именами в формате *<db_table_param_name>.<table_attr>*. Для примера, представленного на рис. 13, помимо явно определенного параметра *teres* будут дополнительно определены параметры со значениями каждого атрибута из связанной таблицы БД (табл. 2).

Подробное описание формата aINI доступно в сети Интернет [33].

4. Программная реализация

Программные реализации инструментальных средств aINI-парсера и GUI-генератора были осуществлены в двух версиях:

- на языке программирования C++ (с использованием библиотеки Qt помимо прочих) в форме

Таблица 2

Параметры, явно и неявно определяемые при чтении данных из aINI-файла, представленного на рис. 13, *а*, см. третью сторону обложки

Имя параметра (короткое/полное)	Значение параметра	Определяется при синтаксическом разборе
teres / sec.teres	*TSTS0028,pC00000iso,CURRENT*	Явно
teres.tesid / sec.teres.tesid	*TSTS0028*	Неявно
teres.ioprm / sec.teres.ioprm	*pC00000iso*	Неявно
teres.valty / sec.teres.valty	*CURRENT*	Неявно
teres.ioprv / sec.teres.ioprv	*11.1*	Неявно
teres.unitt / sec.teres.unitt	*kgf/mm2*	Неявно
teres.dscra / sec.teres.dscra	*Получено из таблицы от...*	Неявно

```
[Author]//Идентификация автора разработки
AuthorName=[alsokolo]$sys.users//Автор разработки
*AuthorSID=sa//SID Автора ываыва
-OutputFilename=@AuthorSID@_@CodeObjectName@.res//

[Generator parameters]//Параметры генерации
CopyObjectToRep=[1]{0|1}//Перенести объект генерации в репозиторий (ONLINE-MODE)
RepPath=C:\rep_new//Путь к репозиторию
TemplatesPath=C:\rep_new\dev\res\Templates//Путь к каталогу с шаблонами
TemporaryPath=C:\rep_new\_tmp//Путь к каталогу временного хранения

[Object parameters]//Параметры генерируемого объекта
CodeObjectName=COMPMATH_LEC1//*Наименование объекта(varchar(25))
Description=Введение. Погрешности. Численное решение нелинейных уравнений. Численное
дифференцирование //*Описание
ParametersFile=[EVN_COMPMATH_LEC1_15_02_2018.imp]//Имя файла дополнительных параметров
TemplateSID=[EVN]$gen.tmpls//Тип генерируемого объекта из БД

[Project data]//Идентификация объекта
ComplexSID=[edu]$sys.cmplx//Идентификатор комплекса
SolutionSID=[cmm]$sys.solun//Идентификатор решения
ProjectSID=[rgc]$sys.prjct//Идентификатор проекта
```

Рис. 14. Исходные данные в формате aINI функции подготовки входных данных для последующей генерации объектов на основе шаблонов CASE-инструментария PBC GCD

динамической библиотеки раннего связывания для встраивания в приложения для операционных систем семейств Windows, Linux;

- с использованием программного каркаса Django и языков программирования Python, JavaScript, CSS, HTML5 для встраивания в web-приложения в форме отдельных сценариев.

Созданный программный инструментарий был интегрирован в состав клиента PBC GCD для операционной системы Windows и web-клиента PBC GCD в форме подсистем. Детальное описание методов интеграции выходит за рамки настоящей статьи.

5. Примеры прикладного использования

Число реализованных функций PBC GCD на основе aINI-формата довольно велико (см. табл. 1). С использованием представленного подхода были созданы aINI-файлы для определения исходных данных отдельных активно эксплуатируемых в системе функций решения прикладных задач, а на их основе с использованием GUI-генератора были построены графические пользовательские интерфейсы.

В состав PBC GCD входит программный CASE-инструментарий, реализованный в виде программной подсистемы. Функциональные возможности CASE-инструментария обеспечивают автоматизацию процессов разработки новых функциональных возможностей системы, а также автоматизацию процессов генерации документации.

На рис. 14 представлен исходный aINI-файл, а на рис. 15 соответствующие ему автоматически сформированные GUI функции подготовки входных данных для последующей генерации объектов на основе шаблонов CASE-инструментария PBC GCD [36].

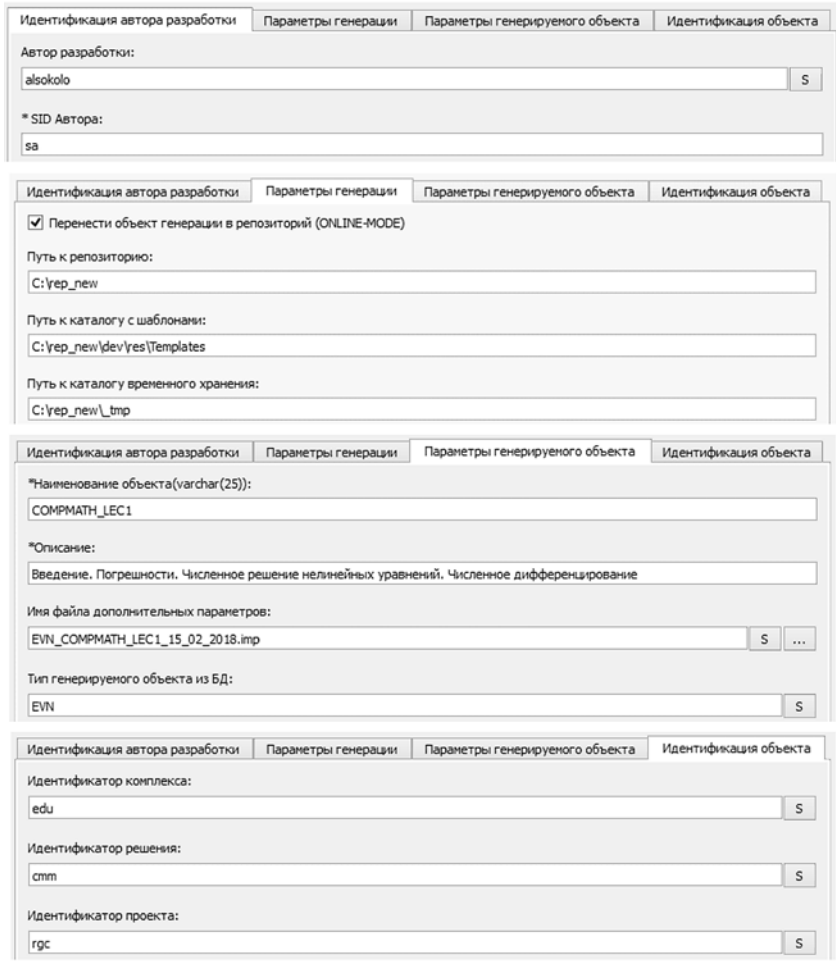
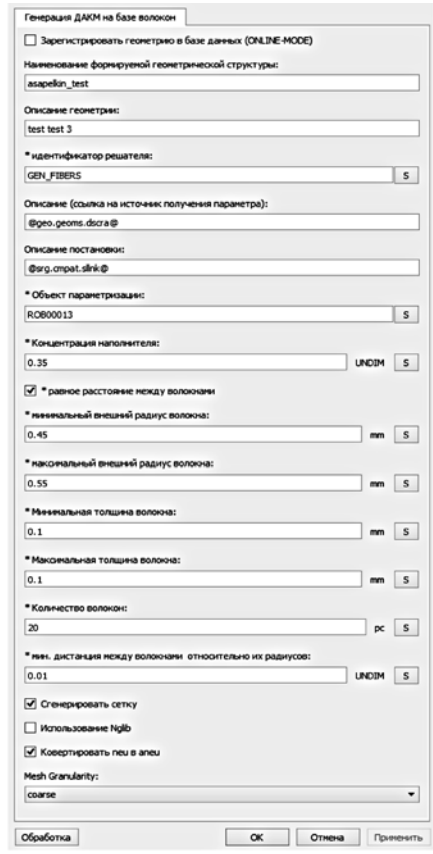


Рис. 15. Автоматически построенные GUI для функций генерации объектов CASE-инструментария PBC GCD

```
[Generator input]//Генерация ДАКМ на базе волокон
RegisterGEO=[0]{0|1}//Зарегистрировать геометрию в базе данных (ONLINE-MODE)

geo.geoms.geonm=asapelkin_test//Наименование формируемой геометрической структуры
geo.geoms.dscra=test test 3//Описание геометрии

*srg.cmput.slver=[GEN_FIBERS]$com.slvrs//идентификатор решателя
srg.cmpat.sllnk=@geo.geoms.dscra@//Описание (ссылка на источник получения параметра)
tsk.tasks.dscra=@srg.cmpat.sllnk@//Описание постановки
*tsk.tasks.rsobj=[ROB00013]$srg.rsobs//Объект параметризации

-WidthX=1//Ширина ЯП
-HeightY=@WidthX@//Высота ЯП
-LengthZ=@HeightY@//Длина ЯП

*concentration=0.35[[UNDIM]]//Концентрация наполнителя

*evenly_gen=[1]{0|1}//равное расстояние между волокнами

// *R=[0.5;0.5:1;0.02] //Диапазон изменения радиуса волокна
*minR=0.45[[mm]]//минимальный внешний радиус волокна
*maxR=0.55[[mm]]//максимальный внешний радиус волокна

-minAD=0[[mm]]//минимальный угол отклонения волокна
-maxAD=0[[mm]]//максимальный угол отклонения волокна

*minG=0.1[[mm]]//Минимальная толщина волокна
*maxG=@minG@[[mm]]//Максимальная толщина волокна

*N=20[[pc]]//Количество волокон

*coef_min_dist=0.01[[UNDIM]]//мин. дистанция между волокнами  относительно их радиусов

mesh=[1]{0|1}//Сгенерировать сетку
lib_call=[0]{0|1}//Использование Nglib
neu2aneu=[1]{0|1}//Ковертировать neu в aneu
MeshGranularity=[coarse]{fine|veryfine|moderate|coarse|verycoarse}//Mesh Granularity
```

а)



б)

Рис. 16. Определение исходных данных в формате aINI для задачи построения модели ячейки периодичности 1D-армированного композиционного материала:

а — исходный aINI-файл; *б* — сформированная форма GUI

Ряд подсистем PBC GCD был создан для проведения комплексного исследования различных физико-механических характеристик композиционных материалов (КМ), помимо прочего, на основе методов конечных элементов и гомогенизации [37]. Применение метода гомогенизации предполагает построение модели представительного элемента объема, описывающего схему армирования исследуемого КМ. Такие представительные элементы объема в микромеханике КМ называют "ячейками периодичности" [37]. На рис. 16 представлен исходный aINI-файл и соответствующий ему автоматически сформированный GUI для ввода исходных данных функции генерации геометрии ячейки периодичности 1D-армированного КМ.

Заключение

Новый текстовый формат aINI позволил определять списки входных параметров, включая их текущие значения (метаданные), как для построения GUI, так и для его непосредственного использования при проведении расчетов. Использование aINI позволило полностью отделить процессы подготовки исходных данных от процессов их обработки [4]. Таким образом, созданный программный инструментарий обеспечил возможности независимой разработки вычислительных программных модулей и модулей подготовки входных данных.

Универсальность формата aINI обеспечила возможность создания интегрируемых web-ориентированных GUI-генераторов и GUI-генераторов для создания приложений для операционных систем семейств Windows и Linux.

Применение созданного программного инструментария позволило существенно сократить трудозатраты на разработку прикладного ПО инженерного анализа.

Представленный программный инструментарий реализован в виде автоматически интегрируемых программных модулей (плагинов) в PBC GCD. Созданный инструментарий заложил основу для разработки подсистемы ввода данных PBC GCD и обеспечил возможности определения исходных данных для большинства впоследствие реализуемых и интегрируемых новых вычислительных программных модулей. Эффективность созданного программного

инструментария подтверждена многолетней эксплуатацией PBC GCD в рамках реализации НИР и ОКР[1], включающих необходимость разработки программного обеспечения.

Разработка выполнена авторами в МГТУ им. Н. Э. Баумана на инициативной основе при разработке ядра системы PBC GCD.

Список литературы

1. **Таненбаум Э., ван Стеен М.** Распределенные системы. Принципы и парадигмы. СПб.: Питер, 2003. 877 с.
2. **Приказ** № 96 от 01.04.2015 Минкомсвязи России "Об утверждении плана импортозамещения программного обеспечения". 2015. 3 с.
3. **Орлов С. А.** Технологии разработки программного обеспечения. Разработка сложных программных систем: учебник для вузов. 3-е изд. СПб.: Питер, 2004. 526 с.
4. **Константайн Л., Локвуд Л.** Разработка программного обеспечения. СПб.: Питер, 2004. 592 с.
5. **Royce W. W.** Managing the development of large software systems: concepts and techniques // Proc. IEEE WESTCON, Los Angeles, August 1970. P. 1—9.
6. **Соколов А. П., Шпакова Ю. В., Першин А. Ю.** Проектирование и разработка распределенной программной подсистемы сетевого взаимодействия и диспетчеризации ресурсоемких процессов решения задач анализа эффективных характеристик композиционных материалов / Отчет о НИР/НИОКР 14.B37.21.1869, дата гранта: 04.10.2012, финансирующая организация: Министерство образования и науки РФ, исполнитель: МГТУ им. Н. Э. Баумана, номер государственной регистрации 01201350820. 2013. 242 с.
7. **Соколов А. П., Шпакова Ю. В., Першин А. Ю.** Проектирование распределенной программной системы GCD численного моделирования композитов // Математические методы в технике и технологиях — ММТТ-25: сб. трудов XXV Международной научной конференции: в 10 т. Т. 5. Секция 8,9 / Под. общ. ред. А. А. Большакова. Волгоград: ВолГУ, 2012. С. 79—80.
8. **Димитриенко Ю. И., Соколов А. П.** Численное моделирование композиционных материалов с многоуровневой структурой // Известия РАН. Физическая серия. 2011. Т. 75, № 11. С. 1551—1556.
9. **Димитриенко Ю. И., Дубровина А. Ю., Соколов А. П.** Моделирование усталостных характеристик композиционных материалов на основе метода асимптотического осреднения и "химического" критерия длительной прочности // Вестник МГТУ им. Н. Э. Баумана. Сер. Естественные науки. 2011. Спец. выпуск. Математическое моделирование. С. 34—49.
10. **Димитриенко Ю. И., Соколов А. П.** Многомасштабное моделирование упругих композиционных материалов // Математическое моделирование. 2012. Т. 24, № 5. С. 3—20.
11. **Димитриенко Ю. И., Сборщиков С. В., Соколов А. П.** и др. Численное и экспериментальное моделирование прочностных характеристик сферопластиков // Композиты и наноструктуры. 2013. № 3. С. 35—51.
12. **Димитриенко Ю. И., Сборщиков С. В., Соколов А. П.** Численное моделирование микроразрушения и прочностных характеристик пространственно-армированных композитов // Механика композиционных материалов и конструкций. 2013. Т. 19, № 3. С. 365—383.
13. **Press W. H., Teukolsky S. A., Vetterling W. T., Flannery B. P.** Numerical Recipes 3rd Edition: The Art of Scientific Computing. New York, USA: Cambridge University Press, 2007. 1262 p.
14. **Димитриенко Ю. И., Соколов А. П.** Система автоматизированного прогнозирования свойств композиционных материалов // Информационные технологии. 2008. № 8. С. 31—38.
15. **Myers B. A., Rosson M. B.** Survey on user interface programming // In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)/ Eds. P. Bauersfeld, J. Bennett, G. Lynch. New York, USA: ACM. 1992. P. 195—202. DOI: 10.1145/142750.142789.
16. **Shirota Y., Lizawa A.** Automatic GUI Generation from Database Schema Information // Systems and Computers in Japan. 1997. Vol. 28, No. 5. P. 1—9.
17. **Ramon O. S., Cuadrado J. S., Molina J. G.** Model-driven reverse engineering of legacy graphical user interfaces // Automated Software Engineering. 2014. Vol. 21, Issue 2. P. 147—186. DOI: 10.1007/s10515-013-0130-2.
18. **Грибова В. В., Черкезишвили Н. Н.** Развитие онтологического подхода для автоматизации разработки пользовательских интерфейсов с динамическими данными // Информационные технологии. 2010. № 10. С. 54—58.
19. **Мияйлович Ж., Миличев Д.** Технологии разработки пользовательских интерфейсов // Открытые системы. СУБД. 2013. № 10. С. 43—47.
20. **Ширяев Д. Р.** Автоматическая генерация графических пользовательских интерфейсов доступа к интегрированным данным на основе диаграмм классов UML // Труды ИСП РАН. 2003. Т. 4. С. 219—232.
21. **Falb J., Popp R., Rock T.** et al. Fully automatic generation of web user interfaces for multiple devices from a high-level model based on communicative acts // International Journal of Web Engineering and Technology. 2009. Vol. 5, No. 2. P. 135—161.
22. **Lin G., Xu X.** Method for dynamically generating a user interface from Xml-based documents. Intel Corporation, Santa Clara, CA (US). Patent US 6,941,521 B2, Sep.6, 2005.
23. **Udler A.** System and method for dynamic generation of a graphical user interface., Pittsburgh, PA (US). US 2005/0010877 Al. Jan. 13, 2005.
24. **Thunemann P., Carter M., Princehouse L.** GUI-maker (data-centric automated GUI-generation). [serial online]. 2007-06-28, 2007; Available from: USPTO Patent Applications, Ipswich, MA.
25. **Zhizhimov O. L.** Explain Services on ZooSPACE Platform and Adaptive User Interfaces // CEUR Workshop Proceedings. 2015. Vol. 1536. P. 30—36.
26. **Маккеон Б., Синклэйр Р., Вагонер П. М.** и др. Классы структур автоматизации пользовательского интерфейса и интерфейсы. Патент на изобретение RUS 2336557 17.05.2003.
27. **Нецветаева Г. А., Новиков Ф. А., Парийская Е. Ю., Скрипниченко В. И.** Автоматическая генерация интерфейсов пользователя. Сообщения ИПА РАН. СПб.: Рос. акад. наук, Ин-т приклад. астрономии. Сер. № 159. 2003.
28. **Коршунов П. В., Джазабек С.** Метод управления разработкой пользовательского интерфейса для семейства программных продуктов // Системное программирование. 2005. Т. 1. С. 168—183.
29. **Пискунов С. В., Кратов С. В., Остапкевич М. Б., Веселов А. В.** Использование сборочной технологии для построения пользовательских интерфейсов сетевой информационно-вычислительной системы // Проблемы информатики. 2010. № 4. С. 41—48.
30. **JSON format.** Wikipedia. The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/JSON (дата обращения 13.07.2017).
31. **INI file.** Wikipedia. The Free Encyclopedia. URL: https://en.wikipedia.org/wiki/INI_file (дата обращения 13.07.2017).
32. **Свид. 2013616822** Российская Федерация. Свидетельство об официальной регистрации программы для ЭВМ. Библиотека gcd_dll_INIParser поддержки чтения данных в формате aINI / Соколов А. П.; заявитель и правообладатель: МГТУ им. Н. Э. Баумана (RU). № 2013615288, заявл. 27.06.2013, опубл. 20.08.2013, Реестр программ для ЭВМ. 1 с.
33. **Соколов А. П.** Формат данных Advanced INI (aINI). Руководство системного программиста. Распределенная вычислительная система GCD [Офиц. сайт]. URL: http://sa2systems.ru/svn/public/sa2pdf/comfrm_ugd_AdvancedINI.pdf (дата обращения 13.07.2017).
34. **Хопкрофт Д., Мотвани Р., Ульман Д.** Введение в теорию автоматов, языков и вычислений. М.: Вильямс, 2002. 528 с.
35. **Соколов А. П., Михайловский К. В., Щетинин В. Н.** и др. Численное определение эффективных упруго-прочностных характеристик композитных заделок газоразделительных мембранных модулей // Материалы IX Международной конференции по неравновесным процессам в соплах и струях (NPNJ'2016). Алушта, Крым. 25—31 мая 2016. Московский авиационный институт, 2016. С. 387—389.

[1] За семь лет разработки не было создано ни одного статического GUI при решении прикладных задач из области научных исследований, тогда как использовались исключительно динамически формируемые GUI, число которых на текущий момент составило 268, включая генерируемые на основе других источников метаданных.

36. **Свид. 2014612782** Российская Федерация. Свидетельство об официальной регистрации программы для ЭВМ. Программа gcdcli_plg_CodeGenerator для автоматического формирования документов и программных объектов на основе шаблонов / Соколов А. П., Макаренков В. М.; заявитель и правообладатель: Соколов А. П., Макаренков В. М. (RU). № 2013617477, заявл. 07.08.2013, опубл. 06.03.2014, Реестр программ для ЭВМ. 1 с.

37. **Бахвалов Н. С., Панасенко Г. П.** Осреднение процессов в периодических средах. М.: Наука, 1984, 352 с.

Software Tools for Development of Input Data Subsystems of Computer-Aided Engineering Complexes

A. P. Sokolov, e-mail: alsokolo@bmstu.ru, Bauman Moscow State Technical University, Moscow, 105005, Russian Federation, **A. Yu. Pershin,** e-mail: tony.pershin@gmail.com, mmap@leeds.ac.uk, University of Leeds, Leeds, LS2 9JT, United Kingdom

Corresponding author:

Sokolov Alexandr P., Associate Professor, Bauman Moscow State Technical University, Moscow, 105005, Russian Federation
E-mail: alsokolo@bmstu.ru

Received on August 08, 2017
Accepted on September 21, 2017

A software approach to the development of subsystems for initial data input for systems of computer-aided engineering (CAE) is presented. Proposed approach is based on usage of specialized text data format named aINI, its parser and corresponding graphical user interface (GUI) generators. Graphical user interface generators and aINI-parsers were developed for various platforms using C++ and Python. The approach allows one to make changes to the requirements for the list and types of input data of a wide class of engineering tasks at any stage of development of the corresponding software without the need to change the source code, which significantly speeds up the development process. The proposed approach was implemented in software Distributed Computational System GCD in its subsystem for initial data input. Some ideas implemented in the presented approach are borrowed from the field of development of large distributed industrial software systems (ERP). The reason for creating the approach was the need to simplify, standardize and systematize the processes of software development which are required in framework of numerous research and development (R&D) projects and applied science research carrying out by the authors at the Bauman Moscow State Technical University in the field of mathematical modeling of various physical processes. Examples of dynamically generated graphical user interfaces for defining initial data for a series of computational tasks are presented. The effectiveness of the approach is confirmed by the examples of successfully implemented series of applied research and development projects based on use of Distributed Computational System GCD.

Keywords: *software development, initials input subsystems, dynamic and adaptive graphical user interfaces, automation of data entry processes, parsers, text data interpretation, technologies for the development of computer-aided engineering systems, distributed software systems, model-view controller*

For citation:

Sokolov A. P., Pershin A. Yu. Software Tools for Development of Input Data Subsystems of Computer-Aided Engineering Complexes, *Programmnaya Ingeneria*, 2017, vol. 8, no. 12, pp. 543—555.

DOI: 10.17587/prin.8.543-555

References

1. **Tanenbaum A. S., van Steen M.** *Distributed Systems: Principles and Paradigms. Second Edition,* Pearson, Prentice Hall, 2006, 686 p.

2. **Order No.96.** 01.04.2015. Minkomsvjazi Rossii "Ob utverzhdenii plana importozameshhenija programmnogo obespechenija" (Ministry of Communications of Russia "On approval of the plan for substitution of import software"), 2015, 3 p. (in Russian).

3. **Orlov S. A.** *Tehnologii razrabotki programmnogo obespechenija. Razrabotka slozhnyh programmnyh sistem* (Software development technologies. Development of complex software systems), St. Petersburg, Piter, 2004, 526 p. (in Russian).

4. **Constantine L. L., Lockwood L. A. D.** *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.* ACM Press/Addison-Wesley Publ. Co., New York, USA, 579 p.

5. **Royce W. W.** Managing the development of large software systems: concepts and techniques, *Proc. IEEE WESTCON*, Los Angeles, 1970, pp. 1—9.

6. **Sokolov A. P., Shpakova Yu. V. Pershin A. Yu.** et al. *Proektirovanie i razrabotka raspredelennoj programmnoj podsistemy setevogo vzaimodejstvija i dispetcherizacii resursoemkih processov reshenija zadach analiza jeffektivnyh harakteristik kompozicionnyh materialov* (Developing a distributed software subsystem of network interaction and dispatching of resource-intensive computational processes for solving problems of analysis of effective properties of composite materials). Report of Science Research 14.B37.21.1869, Ministry of

Education and Science of Russian Federation. Bauman Moscow State Technical University, 2013, 242 p. (in Russian).

7. **Sokolov A. P., Shpakova Yu. V., Pershin A. Yu.** Proektirovanie raspredelennoj programmnoj sistemy GCD chislennogo modelirovanija kompozitov (Design of distributed software system GCD of numerical modeling of composites), *Matematicheskie metody v tehnike i tehnologijah (MMTT-25). Trudi XXV Mezhdunarodnoj nauchnoj konferencii*, VolGU, Volgograd, Russia, 2012, no. 10, vol. 5, pp. 79—80 (in Russian).

8. **Dimitrienko Yu. I., Sokolov A. P.** Numerical Modeling of Composites with Multiscale Microstructure, *Bulletin of the Russian Academy of Sciences. Physics*, 2011, vol. 75, no. 11, pp. 1457—1461.

9. **Dimitrienko Yu. I., Dubrovina A. Yu., Sokolov A. P.** Modelirovanie ustalostnyh harakteristik kompozicionnyh materialov na osnove metoda asimptoticheskogo osrednenija i "himicheskogo" kriterija dlitel'noj prochnosti (Simulation of fatigue properties of composite materials on the basis of the method of asymptotic averaging and the "chemical" criterion of long-term strength), *Vestnik MGTU im. N. E. Baumana. Ser. Estestvennye nauki. Spec. vypusk. Matematicheskoe modelirovanie,* 2011, pp. 34—49 (in Russian).

10. **Dimitrienko Yu. I., Sokolov A. P.** Mnogomasshtabnoe modelirovanie uprugih kompozitsionnyh materialov (Multiscale modeling of elastic composite materials), *Matem. Mod.*, 2012, vol. 24, no. 5, pp. 3—20 (in Russian).

11. **Dimitrienko Yu. I., Sborchchikov S. V., Sokolov A. P., Gafarov B. R., Sadovnichiy D. N.** Chislennoe i jeksperimental'noe modelirovanie prochnostnyh harakteristik sferoplastikov (Computer and experimental study modeling of failure of micro-sphere filled composite), *Kompozity i nanostruktury,* 2013, no. 3, pp. 35—51 (in Russian).

12. **Dimitrienko Yu. I., Sborschikov S. V., Sokolov A. P.** Chislennoe modelirovanie mikrorazrushenija i prochnostnyh harakteristik prostranstvenno-armirovannyh kompozitov (Numerical modeling of microfracture and strength properties of spatially reinforced composites), *Mehanika kompozicionnyh materialov i konstrukcij,* 2013, vol. 19, no. 3, pp. 365—383 (in Russian).

13. **William H. P., Saul A. T., William T. V., Flannery B. P.** *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, New York, USA, Cambridge University Press, 2007, 1235 p.

14. **Dimitrienko Yu. I., Sokolov A. P.** Sistema avtomatizirovannogo prognozirovanija svojstv kompozicionnyh materialov (System for automated prediction of properties of composite materials), *Informacionnye tehnologii,* 2008, no. 8, pp. 31—38 (in Russian).

15. **Brad A. M., Rosson M. B.** Survey on user interface programming, *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '92)* / Eds. P. Bauersfeld, J. Bennett, G. Lynch. ACM, New York, USA, 1992, pp. 195—202. DOI: 10.1145/142750.142789.

16. **Shirota Y., Lizawa A.** Automatic GUI Generation from Database Schema Information, *Systems and Computers in Japan*, 1997, vol. 28, no. 5, pp. 1—9.

17. **Ramon O. S., Cuadrado J. S., Molina J. G.** Model-driven reverse engineering of legacy graphical user interfaces, *Automated Software Engineering,* 2014, vol. 21, issue 2, pp. 147—186, DOI:10.1007/s10515-013-0130-2.

18. **Gribova V. V., Cherkezishvili N. N.** Razvitie ontologicheskogo podhoda dlja avtomatizacii razrabotki pol'zovatel'skih interfejsov s dinamicheskimi dannymi (Development of an ontological approach for automating of development of user interfaces with dynamic data), *Informacionnye tehnologii,* 2010, no. 10, pp. 54—58 (in Russian).

19. **Mijajlovich Zh., Milichev D.** Tehnologii razrabotki pol'zovatel'skih interfejsov (User interfaces development technologies), *Open systems. RDBMS,* 2013, no. 10, pp. 43—47 (in Russian).

20. **Shirjaev D. R.** Avtomaticheskaja generacija graficheskih pol'zovatel'skih interfejsov dostupa k integrirovannym dannym na osnove diagramm klassov UML (Automatic generation of graphical user interfaces for accessing integrated data based on UML class diagrams), *Trudy ISP RAN,* 2003, vol. 4, no. 1, pp. 219—232 (in Russian).

21. **Falb J., Popp R., Rock T., Jelinek H., Arnautovic E., Kaindl H.** Fully automatic generation of web user interfaces for multiple devices from a high-level model based on communicative acts, *International Journal of Web Engineering and Technology,* 2009, vol. 5, no. 2, pp. 135—161.

22. **Lin Ge., Xu X.** Method for Dynamically Generating a User Interface from XML-Based Documents, Intel Corporation, Santa Clara, CA (US). Patent US 6,941,521 B2, Sep.6, 2005.

23. **Udler A.** System and Method for Dynamic Generation of a Graphical User Interface. Pittsburgh, PA (US). US 2005/0010877 Al. Jan. 13, 2005.

24. **Thunemann P., Carter M., Princehouse L.** GUI-maker (data-centric automated GUI-generation. 2007-06-28, 2007; Available from: USPTO Patent Applications, Ipswich, MA.

25. **Zhizhimov O. L.** Explain Services on ZooSPACE Platform and Adaptive User Interfaces, *CEUR Workshop Proceedings,* 2015, vol. 1536, pp. 30—36.

26. **Makkeon B., Sinkljejr R., Vagoner P. M., Rejd P. D., Fridman M. A., Bernz H. S.** Klassy struktur avtomatizacii pol'zovatel'skogo interfejsa i interfejsy (Classes of user interface automation structures and interfaces), Patent RUS 2336557. 17.05.2003 (in Russian).

27. **Necvetaeva G. A., Novikov F. A., Parijskaja E. Ju., Skripnichenko V. I.** Avtomaticheskaja generacija interfejsov pol'zovatelja (Automatic generation of user interfaces), *Soobshhenija IPA RAN*, Saint-Petersburg, Russian Academy of Sciences, Institute of Applied Astronomy, 2003, no. 159 (in Rissian).

28. **Korshunov P. V., Dzhazabek S.** Metod upravlenija razrabotkoj pol'zovatel'skogo interfejsa dlja semejstva programmnyh produktov (The method of managing of the development of the user interface for the family of software products), *System Programming,* 2005, vol. 1, pp. 168—183 (in Russian).

29. **Piskunov S. V., Kratov S. V., Ostapkevich M. B., Veselov A. V.** Ispol'zovanie sborochnoj tehnologii dlja postroenija pol'zovatel'skih interfejsov setevoj informacionno-vychislitel'noj system (Use of assembly technology for building user interfaces of network information and computing systems), *Problems of Informatics,* 2010, no. 4, pp. 41—48 (in Russian).

30. **JSON format.** Wikipedia. The Free Encyclopedia, available at: https://en.wikipedia.org/wiki/JSON.

31. **INI file.** Wikipedia. The Free Encyclopedia, available at: https://en.wikipedia.org/wiki/INI_file.

32. **Sokolov A. P.** Certificate 2013616822 Russian Federation. Certificate of official computer program registration. Biblioteka gcd_dll_INIParser podderzhki chtenija dannyh v formate aINI (Library gcd_dll_INIParser for reading data in the aINI format) / Sokolov A. P.; applicant and rightholder: Bauman Moscow State Technical University (RU), no. 2013615288, application 27.06.2013, published 20.08.2013, Catalog of Computer Programs of Rospatent. 1 p. (in Russian).

33. **Sokolov A. P.** *Text format Advanced INI (aINI). System architect guide.* Distributed Computational System GCD [Official site]. 2017, 18 p., available at: http://sa2systems.ru/svn/public/sa2pdf/comfrm_ugd_AdvancedINI.pdf (in Russian).

34. **Hopkroft D., Motvani R., Ul'man D.** *Introduction to Automata Theory, Languages, and Computation,* Addison-Wesley, 2002. 528 p.

35. **Sokolov A. P., Mihajlovskij K. V., Shhetinin V. N., Sapelkin A. S., Presnjakov V. V.** Chislennoe opredelenie jeffektivnyh uprugo-prochnostnyh harakteristik kompozitnyh zadelok gazorazdelitel'nyh membrannyh module (Numerical determination of effective elastic-strength properties of composite seals of gas separating membrane modules), *Materialy IX Mezhdunarodnoj konferencii po neravnovesnym processam v soplah i strujah (NPNJ'2016)*, Alushta, Crimea, 25—31 May 2016, pp. 387—389 (in Russian).

36. **Sokolov A. P., Makarenkov V. M.** Certificate 2014612782 Russian Federation. Certificate of official computer program registration. Programma gcdcli_plg_CodeGenerator dlja avtomaticheskogo formirovanija dokumentov i programmnyh ob'ektov na osnove shablonov (Program gcdcli_plg_CodeGenerator for automatic generation of documents and program objects based on templates) / Sokolov A. P., Makarenkov V. M.; applicants and rightholders: Sokolov A. P., Makarenkov V. M. (RU). No. 2013617477, application 07.08.2013, published 06.03.2014, Catalog of Computer Programs of Rospatent. 1 p. (in Russian).

37. **Bahvalov N. S., Panasenko G. P.** *Osrednenie processov v periodicheskih sredah* (Homogenization of processes in periodic media), Moscow, Nauka, 1984, 352 p. (in Russian).

УДК 004.4 DOI: 10.17587/prin.8.556-562

А. С. Козицын, канд. физ.-мат. наук, вед. науч. сотр., e-mail: alexanderkz@mail.ru,
С. А. Афонин, канд. физ.-мат. наук, вед. науч. сотр., e-mail: serg@msu.ru,
МГУ имени М. В. Ломоносова, г. Москва

Разрешение неоднозначностей при определении авторов публикации с использованием графов соавторства в больших коллекциях библиографических данных

Рассмотрены некоторые подходы к решению задач обработки библиографических данных в системах наукометрии с помощью выделения статистических закономерностей в больших коллекциях таких данных. На примере представительной информационно-аналитической системы "ИСТИНА", которая эксплуатируется в МГУ им. М. В. Ломоносова, представлены результаты исследований по идентификации авторов по библиографическим данным статей. Предлагаемый алгоритм позволяет выделять авторов с точностью более 95 %.

Ключевые слова: *наукометрия, обнаружение закономерностей, библиографические данные, граф, цитирование, автор, тематический анализ*

Введение

В настоящее время во многих ведущих учебных заведениях и научных центрах активно внедряют системы автоматизации сбора данных о научной и педагогической деятельности работников. Это обусловлено тем обстоятельством, что эффективное управление большими организациями в сфере науки и образования невозможно без внедрения методов оценки эффективности деятельности отдельных работников таких организаций. Для решения подобных задач необходимо использовать современные средства и системы автоматизированного сбора, верификации, хранения и анализа больших объемов библиографической информации, которая описывает результаты научной деятельности сотрудников отдельных организаций и предоставляет данные для оценки эффективности их деятельности. Набор используемых для оценки показателей такой эффективности (индикаторов) должен быть достаточно большим, чтобы охватывать все основные сферы деятельности организации и учитывать особенности отдельных ее структурных подразделений. При этом необходимо принимать во внимание то обстоятельство, что любые количественные наукометрические показатели не могут являться абсолютным критерием оценки деятельности каждого работника, а представляют только начальную оценку, подлежащую дальнейшему анализу экспертами.

В качестве такой системы, на средствах анализа данных которой иллюстрируются результаты исследований, представленные в настоящей статье, рассмотрим информационно-аналитическую систему (ИАС) "ИСТИНА" [1, 2], активно используемую в МГУ им. М. В. Ломоносова для подготовки принятия управленческих решений (далее по тексту ИАС "ИСТИНА" или система). С учетом масштабов МГУ и относительно большого времени эксплуатации системы эти данные могут адекватно отражать результаты применения предлагаемых подходов в других организациях в сфере науки и образования.

Одним из важнейших показателей, которые необходимо собирать для оценки научной деятельности работников организации, является число публикаций, а также их цитируемость, распределение публикаций по журналам и темам. Кроме того, необходимо учитывать следующие показатели: участие работников в научных проектах и конференциях; получение ими свидетельств о защите интеллектуальной собственности; педагогическую деятельность; представление к премиям и наградам; участие в работе диссертационных советов и др. [3]. Сбор и верификация, агрегирование и анализ отмеченных данных позволяют адекватно готовить аналитические материалы для принятия управленческих решений, проводить автоматизированный расчет рейтинговых показателей отдельных работников, оценивать работу подразделений и оперативно, без больших затрат готовить различного рода отчеты. Однако при планировании и проведении работ по сбору и анализу показателей о научной и педагогической деятельности сотрудников необходимо принимать во внимание наличие обратных связей между системой анализа и измеряемыми показателями. Следует учитывать, что, как правило, проведение измерений характеристик достаточно сложного объекта начинает оказывать влияние на сам объект. В качестве аналогий можно привести факты, что при измерении термометром

температуры газа происходит теплообмен газа с термометром, а при измерении скорости жидкости перед точкой, в которую помещен датчик, возникает зона торможения и происходит образование вихрей. Введение механизмов измерения показателей оценки научной деятельности также оказывает влияние на структуру измеряемых показателей. Проиллюстрируем это утверждение на примере числа соавторов публикуемых статей. В табл. 1 приведены данные за десять лет о среднем числе соавторов в статьях, зарегистрированных в ИАС "ИСТИНА", с разбивкой всех статей по годам, а также по статьям без учета коллабораций (больших устойчивых групп соавторов, которые публикуются вместе, например, ATLAS COLLABORATION, OPERA COLLABORATION). Из приведенных в табл. 1 данных можно сделать вывод, что с момента начала внедрения механизмов учета публикаций в 2011 г. наблюдается устойчивый рост среднего числа соавторов.

Подобный рост объясняется стремлением работников улучшить свои показатели качества работы и тем самым получить более высокий рейтинговый статус в организации. В данном случае увеличение числа соавторов используется для увеличения числа статей у сотрудников подразделения и для повышения показателя цитируемости. Подобные преднамеренные искажения объективных показателей в области наукометрии неизбежны при внедрении систем автоматизированной оценки деятельности работников. Вместе с тем при использовании подобных систем для эффективного управления крупной научной организацией или вузом возникает необходимость устранения непреднамеренных ошибок и искажений измеряемых показателей, которые являются результатом некорректного ввода данных в систему. Например, неправильное указание авторов при вводе данных о статьях, книгах и другой научной продукции значительно влияет на качество собираемых наукометрических данных. В связи с этим возникает необходимость автоматизации процесса определения авторов по фамилии и инициалам при вводе библиографических сведений о публикации.

Таблица 1

Среднее число соавторов статей

Год	Среднее число соавторов	Рост, %	Среднее число соавторов без учета коллабораций	Рост без учета коллабораций, %
До 2006	2,724	100	2,708	100
2006	2,791	102	2,780	103
2007	2,803	103	2,778	103
2008	2,739	101	2,714	100
2009	2,761	101	2,725	101
2010	2,760	101	2,726	101
2011	2,847	105	2,791	103
2012	3,017	111	2,942	109
2013	3,023	111	2,961	109
2014	3,013	111	2,954	109
2015	3,081	113	3,002	111
2016	3,194	117	3,125	115
2017	3,453	127	3,256	120

Распознавание авторов по библиографическим данным

Представляется очевидным тот факт, что только ученый или педагог может точно перечислить все результаты своей деятельности. Поэтому для достижения максимально полного состава таких данных необходимо предоставить конечным пользователям наукометрической системы возможность ввода сведений о результатах своей деятельности. При этом если какой-либо результат деятельности был получен в соавторстве, то сведения о нем должны быть введены в систему один раз, независимо от числа соавторов. Такой подход позволяет не только экономить человеческий ресурс, но и быстрее, особенно на начальном этапе становления системы, пополнять коллекцию данных. Он позволяет устранить ненужное дублирование и возникающие в связи с этим дополнительные ошибки при вводе данных.

Одной из задач, которые в связи с этим возникают в ходе разработки механизмов автоматизации процессов сбора библиографических данных о публикациях в больших наукометрических системах, является надежное определение (идентификация) автора по указанным в публикации фамилии и инициалам. Важным фактором в этом случае является то, насколько распространена фамилия автора публикации. Если для редко встречающихся фамилий поиск соответствия между указанной в статье фамилией и зарегистрированным в системе автором этот вопрос решается простым поиском совпадения строк, то для распространенных фамилий необходимо проводить более сложный анализ. Например, среди 90 тысяч авторов ИАС "ИСТИНА", являющихся сотрудниками МГУ им. М. В. Ломоносова, около тысячи имеют фамилии "Иванов", "Кузнецов", "Смирнов", "Петров", "Попов", а 50 тысяч авторов имеют более десяти однофамильцев. Кроме того, следует учитывать большое количество соавторов, которые не являются сотрудниками МГУ, но также должны правильно распознаваться системой при проведении автоматизированного разбора библиографических данных, добавляемых авторами статей. Отмеченный факт является важным, поскольку в современных научных исследованиях большое число проектов выполняется совместно работниками нескольких

организаций, а доля сотрудников МГУ среди полного списка авторов статей, загруженных в ИАС "ИСТИНА", составляет менее 30 %.

Вследствие описанных выше причин использования только фамилии и инициалов для определения автора статьи оказывается недостаточно. При вводе пользователем информации о статье после внесения библиографической информации необходимо правильно указать автора для каждой фамилии из списка соавторов статьи. Если система предлагает пользователю всех однофамильцев, то выбор проводится из слишком большого числа возможных вариантов. Многие пользователи затрудняются сделать правильный выбор, загружая в систему статьи с неправильными или незаполненными данными об авторах статей. Такие ошибки оказывают существенное влияние на корректность расчета наукометрических показателей (число статей в журналах из списка ВАК, RSCI, Web Of Science, Scopus, цитируемость и др.) как по отдельным сотрудникам, так и по организации в целом.

Точность решения задачи определения автора статьи по фамилии и инициалам, указанным в ее библиографическом описании, можно увеличивать на основе использования дополнительной информации об устойчивых группах соавторов. Один из таких методов, основанный на поиске максимально связанных подграфов в графе соавторства, описан в работах [4, 5]. Этот метод используется в ИАС "ИСТИНА" в настоящее время, однако он имеет ряд недостатков. Во-первых, этот метод имеет достаточно большую вычислительную сложность, во-вторых, при определении возможных авторов статьи не используется информация об авторизованном пользователе, который вводит информацию о статье. Последний аспект особенно важен, поскольку именно авторы статей наиболее заинтересованы в своевременном и правильном добавлении их работ в наукометрическую систему. Подтверждением тому является тот факт, что согласно проведенным статистическим оценкам по данным из ИАС "ИСТИНА" (данные приведены в табл. 2), более 93 % публикаций вносится одним из соавторов работ. При этом только менее 7 % работ вносят пользователи системы, не являющиеся авторами статей (секретари кафедр и лабораторий, а также другие лица, которым делегированы соответствующие права). В связи с этим информацию из учетной записи зарегистрированного пользователя необходимо использовать для формирования более точных механизмов распознавания авторов.

Таблица 2

Доля статей, вводимых одним из соавторов, в зависимости от числа соавторов в статье

Число соавторов статьи	Статей, тыс.	Статей, введенных авторами, тыс.	Статей, введенных авторами, %
1	246	228	93
2	113	107	94
3	84	79	94
4	58	54	94
5	37	35	94
6	24	23	94
7	14	14	94
8	9	8	94
9	6	5	94
Более 9	13	12	93

Для более точного решения задачи определения авторов по библиографическому описанию статьи требовалось построить алгоритм, который на основе библиографического описания и информации о пользователе, осуществляющем ввод данных о статье, позволяет автоматически сформировать набор идентификаторов авторов, которые адекватны библиографическому описанию. Разработанный авторами этой публикации алгоритм на первом этапе выделяет список возможных авторов для каждой фамилии, упоминающейся в библиографическом описании статьи.

Первый этап алгоритма состоит из следующих шагов.

Шаг 1. Поиск указанной в библиографических данных фамилии и инициалов среди зарегистрированных в системе фамилий авторов. При этом для каждого автора учитываются все варианты написания его фамилии и инициалов, встречавшиеся ранее. Такой анализ необходим для работы со статьями, изданными на других языках. Для фамилий требуется точное совпадение, а для инициалов — совпадение наиболее короткой строки с префиксом длинной строки. Таким образом, для записи "Иванов И И" будут добавлены в список возможных соавторов "Иванов Иван Иванович", "Иванов Иван Ив", но не будет добавлен вариант "Иванов Иван Петрович".

Шаг 2. Сравнение записей в полученном списке возможных авторов с данными о пользователе, который осуществляет ввод данных о статье.

Шаг 3. Если среди возможных авторов встречается авторизованный пользователь, осуществляющий ввод данных о статье, то он считается определенным, и остальные возможные авторы из списка для этой фамилии удаляются.

Результатом работы первого этапа алгоритма является список возможных уникальных ключей авторов, известных системе, для каждой фамилии автора из библиографического описания статьи. Программный компонент, реализующий алгоритм на первом этапе, выполнен в виде модуля в СУБД Oracle и может использоваться в SQL-запросах. Например, "select column_value IRID from table(get_man_id_by_fio_s('Иванов И И',12454))".

На втором этапе алгоритма осуществляется выделение наиболее правдоподобного уникального ключа автора для каждой фамилии. В качестве входных данных второй этап алгоритма принимает полученные на первом этапе списки возможных уникальных ключей авторов для каждого соавтора статьи. Кроме того, в качестве входных данных используется граф соавторства, вершинами которого являются известные системе авторы, а ребра имеют вес, равный числу общих публикаций двух авторов. Второй этап алгоритма состоит из следующих шагов.

Шаг 1. Сортировка всех соавторов статьи по числу вариантов возможных уникальных ключей. В начале списка располагаются соавторы, для фамилий которых на первом этапе подобрано небольшое число возможных вариантов. Соавтор, который определен на текущий момент как авторизованный пользователь, помещается на первое место списка.

Шаг 2. Если в начале списка оказались соавторы с единственным возможным уникальным ключом, которые считаются "распознанными", то для каждого из соавторов с несколькими возможными вариантами уникальных ключей выбираются ребра связи с "распознанными" первичными ключами. После этого выбирается уникальный ключ, имеющий ребро связи с наибольшим весом. Выбранный уникальный ключ считается "распознанным".

Шаг 3. Повторение шага 2, пока в списке соавторов остаются "нераспознанные" ключи или пока в результате выполнения шага 2 не перестают появляться новые "распознанные" ключи.

Шаг 4. Если не все соавторы распознаны, то среди этих соавторов определяется ребро с наибольшим весом для каждой пары вариантов ключей.

Тестирование алгоритма проводилось на графе соавторства, имевшем около 226 тыс. вершин (авторов) и 5 млн ребер. Для построения графа соавторства использовалась информация из статей, тезисов, книг и проектов. Вес ребра определяется как число общих работ у заданных двух авторов. Следует отметить, что большинство ребер в анализируемом графе имеет вес 1. Данные о весах ребер используемого графа приведены ниже.

Вес ребра	**Число ребер**
1	3 506 738
2	808 090
3	270 726
4	119 922
5	69 254
6	44 388
7	31 652
8	23 418
9	18 116
10	15 086
Более 10	104 926

В ходе тестирования было обработано 540 тыс. статей. Совпадение результатов расчета алгоритма с реальными данными составило 520 тыс. записей:

Результат разбора	**Число записей**
Автор распознан и совпадает с автором, указанным в системе пользователем	503 824
Автор не распознан и не указан в системе пользователем	18 807
Автор распознан, но не указан в системе пользователем	47 244
Автор распознан, но не совпадает с автором, указанным пользователем	19 337
Автор не распознан, но указан в системе пользователем	3256

Следует отметить причины того, что для значительной части статей автор распознан, но не указан пользователем. С одной стороны, это может быть результат плохого качества автоматического определения автора при вводе данных о статье в предыдущей версии системы. С другой стороны, это может быть связано с отсутствием заинтересованности пользователя в правильном выборе соавторов вручную. Как следствие отмеченных причин, значительная часть статей введена в систему с единственным автором из всего списка соавторов, а именно — пользователем, который вносил данные о статье в систему. Использование нового алгоритма позволит предлагать пользователю более адекватный (точный) распознанный список авторов для новых вносимых в систему статей, а также определить соавторов для старых статей, в которых при загрузке были указаны не все авторы.

Скорость обработки тестируемой реализации представленного алгоритма составила более 10 000 тыс. статей за 40 с, или 0,004 с на одну статью, что позволяет давать рекомендации пользователю по выбору авторов при вводе статьи без задержек. Программный компонент, реализующий алгоритм на втором этапе, также выполнен в виде модуля в СУБД Oracle и может использоваться в SQL-запросах. Например, "SELECT * FROM TABLE(get_man_by_name(ARRAY_VARCHAR2('КОЗИЦЫН А С', 'АФОНИН С А', 'ЗАНЧУРИН М А', 'КОРШУНОВ А А'),0)) ".

Дальнейшее улучшение полученных на настоящее время результатов возможно за счет использования двудольного графа (пользователь — автор), позволяющего учесть факт регулярного ввода информации о публикации нескольких авторов одним пользователем. Например, в случае ввода информации обо всех сотрудниках кафедры ученым секретарем кафедры (лаборатории) или другим лицом, которому эти полномочия делегированы. Перспективным также является использование весовой функции при определении возможных авторов на первом шаге алгоритма. Однако некоторые ошибки являются неустранимыми, например, в случае смены фамилии автором или неправильного ее указания.

Недостатком предлагаемого алгоритма является отсутствие возможности его использования для статей, написанных одним автором. Однако этот недостаток частично компенсируется тем обстоятельством, что данные о таких статьях авторы обычно

вводят самостоятельно и авторство однозначно определяется по авторизованному пользователю.

Разработанный алгоритм позволяет при вводе данных в систему подсказывать пользователю правильный вариант выбора автора для каждой фамилии из библиографических данных статьи. Он повышает точность вносимых данных, а также позволяет анализировать внесенные данные на наличие потенциальных ошибок.

Выделение тематических направлений исследований авторов

Автоматическое определение областей интересов пользователей позволяет решить две практически важные задачи: поиск "похожих" по тематике авторов и поиск авторов по описанию темы, например, по ключевым словам. Кроме того, такой подход позволяет рекомендовать пользователям конференции и журналы на основе статистики публикаций авторов схожей тематики.

При проведении классификации необходимо учитывать то обстоятельство, что авторы могут иметь публикации в нескольких тематических направлениях. Интересы людей могут со временем меняться, автор может одновременно с успехом работать в нескольких научных областях. Следует также отметить наличие отдельного класса авторов, которых можно определить как "руководители". Для таких авторов, как правило, характерно наличие соавторства в статьях, проектах и докладах сразу по многим тематическим направлениям.

Наиболее простым способом классификации по областям научной деятельности является классификация авторов по тематической направленности журналов, в которых авторы печатают свои работы. В настоящее время в ИАС "ИСТИНА", например, используется несколько различных тематических классификаторов журналов: ГРНТИ (1800 рубрик), Scopus (310 рубрик), Web of Science (230 рубрик) и медицинский классификатор Medline MeSH (124 рубрики) [6]. Недостатком такого метода является отсутствие возможности точного определения тематики статьи и тематических направлений ее авторов, поскольку тематика журнала, как правило, формулируется слишком широко. Таким же недостатком обладает тематическая классификация авторов на основе описания своих достижений с указанием этих рубрик.

В экспериментах, которые проводили с целью анализа и устранения отмеченных недостатков, использовали векторную модель описания тематических интересов авторов, а тематическую близость авторов оценивали как косинус угла между их векторами. В качестве координат векторов использовали четыре варианта характеристик:

- число статей автора, которые напечатаны в журнале с заданной рубрикой;
- число журналов с данной рубрикой, в которых напечатаны статьи автора;
- число статей автора, которые напечатаны в журнале с заданной рубрикой, деленное на общее число статей в журналах с данной рубрикой;
- число журналов с данной рубрикой, в которых напечатаны статьи автора, деленное на общее число журналов с данной рубрикой.

Во всех случаях точность определения "похожих" авторов была очень низкой и не превышала 20 %. Это объясняется двумя факторами. Во-первых, многие журналы используют очень широкий набор классификаторов. Например, журнал "Информационное общество" имеет 19 общих рубрик, в том числе "Социология", "Философия", "Государство и право. Юридические науки", "Информатика", "Кибернетика". Публикация статьи в таком журнале создает много тематических неправильных связей между авторами. Во-вторых, даже в одном журнале публикуются авторы разных тематических направлений.

Повышение качества определения "похожих" авторов возможно за счет предварительной оценки количества рубрик разных тематических направлений. Журналы с узкой специализацией имеют, как правило, не более четырех основных тематических направлений (не связанные между собой позиции тематического рубрикатора ГРНТИ, Scopus или Web of Science). Ниже показано распределение числа журналов по количеству основных тематических направлений, зарегистрированных в системе "ИСТИНА".

Число тематических направлений	Число журналов
1	7158
2	5359
3	4047
4	3296
5	2180
6	1540
Более 6	2114

Как видно из представленных данных, большая часть журналов (77 %) имеет только четыре основных направления. Журналы с большим числом тематических направлений исключались из расчета тематической близости авторов. Кроме того, при проведении расчетов вес тематических направлений каждого журнала делили на число общих тематических направлений в этом журнале. Таким образом, узкоспециализированные журналы оказывали большее влияние на определение тематических интересов автора.

С учетом изложенных выше фактов авторами данной статьи был разработан алгоритм автоматического определения близости авторов статей по тематическим направлениям. Разработанный алгоритм определения тематической близости авторов состоит из следующих шагов.

Шаг 1. Выделение списка журналов с числом рубрик не более четырех.

Шаг 2. Расчет веса тематических направлений каждого журнала.

Шаг 3. Расчет вектора тематических направлений для каждого автора как число журналов с данной рубрикой, в которых напечатаны статьи автора, деленное на общее число журналов с данной рубрикой.

Шаг 4. Исключение из расчета пар авторов, векторы которых ортогональны (не имеют общих тематических направлений).

Шаг 5. Сортировка пар авторов по значению косинуса угла между соответствующими им векторами.

Шаг 6. Отбор для каждого автора N наилучших пар (при оценке результатов работы использовалось $N = 30$).

Исключение из расчета журналов с широким набором классификаторов и учет веса тематических рубрик позволили существенно повысить точность получаемых результатов. Точность определения пар авторов, имеющих одинаковые общие тематические направления, составила 90 %. Правильность результатов определения на тестовой выборке проверяли вручную по списку статей каждого автора.

Основным преимуществом предлагаемого подхода является небольшая вычислительная сложность по сравнению с алгоритмами тематического анализа полнотекстовых документов, а также высокая скорость работы на больших объемах библиографических данных. В первую очередь это объясняется тем фактом, что описанный алгоритм использует только связи между объектами (статьями, авторами, журналами), доступ к которым осуществляется с использованием индекса. Алгоритм не требует использования методов морфологической и синтаксической обработки полнотекстовых документов. Вместе с тем выделить точное тематическое направление он не способен в силу описанных выше причин. Как следствие, задача поиска похожих авторов по тематике не может быть решена в рамках подобного подхода. Предлагаемый алгоритм может быть использован только для выделения общих тематических направлений и подбора журналов и конференций, соответствующих тематическим интересам каждого пользователя системы.

Одним из альтернативных подходов к классификации тематических интересов авторов является использование методов анализа текстовой информации из названий, аннотаций и ключевых слов, которые авторы указывают в описании статей, проектов, достижений, докладов на семинарах и выступлениях в СМИ. Использование полных текстов статей затруднено, поскольку издательства многих журналов не размещают полные версии статей в сети Интернет. Из перечисленных выше методов анализа тематической близости наиболее эффективным является использование набора ключевых слов [7], поскольку авторы самостоятельно отбирают слова, наиболее точно описывающие тематику статьи. Однако именно необходимость ручной работы является существенным препятствием для широкого использования такого подхода. От авторов требуется аккуратное заполнение ключевых слов на этапе ввода данных о статье в систему. Например, из 600 000 статей, зарегистрированных в системе "ИСТИНА", ключевые слова указаны менее чем для 2000. Аннотации заполнены для 100 000 статей. Таким образом, дальнейшее уточнение тематических интересов авторов возможно с использованием методов анализа текстов для аннотаций и названий статей.

Заключение

Решение описанных в статье задач идентификации авторов и определения близости тематических интересов пользователей с использованием методов анализа больших объемов библиографических данных позволяет создавать удобные инструментальные средства для сбора, верификации и обработки информации о научной продукции крупных организаций, а также предоставляет расширенные возможности по поиску, агрегации информации и составлению аналитических отчетов при принятии управленческих решений. Использование нового алгоритма определения авторов по библиографическим данным статьи позволяет повысить качество вносимых пользователем данных и увеличить точность последующего расчета наукометрических показателей. Автоматическое определение тем научных исследований позволяет оперативно готовить аналитические материалы для принятия управленческих решений в разрезе отдельных тематических направлений, а также может использоваться для предоставления пользователям наукометрической системы удобного интерфейса по поиску журналов и конференций.

Список литературы

1. **Садовничий В. А., Васенин В. А., Афонин С. А.** и др. Информационная система "ИСТИНА" как big data — инструментарий в области управления на основе анализа наукометрических данных // Материалы Всероссийской конференции с международным участием "Знания-Онтологии-Теории" (ЗОНТ-2015), 6—8 октября 2015 г. Т. 1. Новосибирск: Институт математики им. С. Л. Соболева СО РАН, 2015. С. 115—123.

2. **Афонин С. А., Бахтин А. В., Бухонов В. Ю.** и др. Интеллектуальная система тематического исследования научно-технической информации (ИСТИНА) / Под ред. В. А. Садовничего. М.: Изд-во МГУ, 2014. 262 с.

3. **Васенин В. А., Афонин С. А., Козицын А. С.** Автоматизированная система тематического анализа информации// Информационные технологии. Приложение. 2009. № 4. 32 с.

4. **Афонин С. А., Гаспарянц А. Э.** Автоматическое построение функции оценки качества в задаче разрешения неоднозначности имен авторов научных публикаций // Программная инженерия. 2015. № 10. С. 31—37.

5. **Афонин С. А., Гаспарянц А. Э.** Разрешение неоднозначности авторства публикаций при автоматической обработке библиографических данных // Программная инженерия. 2014. № 1. С. 25—28.

6. **Медицинские** предметные рубрики. URL: https://ru.wikipedia.org/wiki/Medical_Subject_Headings

7. **Афонин С. А., Лунев К. В.** Выявление тематических направлений в коллекции наборов ключевых слов // Программная инженерия. 2015. № 2. С. 29—39.

The Resolution of Ambiguities in the Identification of Authors of the Publication with the Use of Co-Authors' Graphs in Large Collections of Bibliographic Data

A. S. Kozitsin, alexanderkz@mail.ru, **S. A. Afonin,** serg@msu.ru, Lomonosov Moscow State University, Moscow, 117223, Russian Federation

Corresponding author:

Kozitsin Alexander S., Researcher, Lomonosov Moscow State University, Moscow, 117223, Russian Federation
E-mail: alexanderkz@mail.ru

Received on July 18, 2017
Accepted on August 02, 2017

This article addresses problems related to automated processing of bibliographic data in scientometric systems by means of statistical analysis of large collections of such data. Experimental results on authors identification in bibliographic data are based on the data set presented in ISTINA, a scientometric information system developed and deployed at Moscow State University. The new algorithm for authors identification, presented in this paper, shows 95 % accuracy on the considered data set. Utilization of this algorithm improves the quality of data entered into the system, thus leading to a more reliable scientometric characteristics of individual researchers and administrative units. The paper also discusses possible approaches to some related practically important problems, such as thematic search and classification of publications using coauthoring graph and thematic classification of scientific journals. Automatic discovery of researchers topics of interest allows for on-demand generation of various documents suitable for decision making in specific research areas and could be useful for supplying system users with information on relevant journals or upcoming scientific events.

Keywords: *succometrics, detection of regularities, bibliographic data, graph, citation, author, thematic analysis*

For citation:

Kozitsin A. S., Afonin S. A. The Resolution of Ambiguities in the Identification of Authors of the Publication with the Use of Co-Authors' Graphs in Large Collections of Bibliographic Data*, Programmnaya Ingeneria*, 2017, vol. 8, no. 12, pp. 556—562.

DOI: 10.17587/prin.8.556-562

References

1. **Sadovnichij V. A., Vasenin V. A., Afonin S. A., Kozitsin A. S., Golomazov D. D.** Informacionnaja sistema "ISTINA" kak big data — instrumentarij v oblasti upravlenija na osnove analiza naukometricheskih dannyh. (The information system "ISTINA" as big data is a tool in the field of management based on the analysis of scientometric data), *Materialy Vserossijskoj konferencii s mezhdunarodnym uchastiem "Znanija-Ontologii-Teorii" (ZONT-2015)*, 6—8 October 2015, vol. 1, Novosibirsk, 2015, pp. 115—123 (in Russian).

2. **Afonin S. A., Bahtin A. V., Buhonov V. Yu., Vasenin V. A., Gankin G. M., Gaspariants A. E., Golomazov D. D., Itkes A. A., Kozitsin A. S., Tumajkin I. N., Shapchenko K. A.** *Intellektual'naja sistema tematicheskogo issledovanija nauchno-tehnicheskoj informacii (ISTINA)* (Intellectual system of case research of scientific and technical information) / Eds. V. A. Sadovnichiy, Moscow, Moscow State University, 2014, 262 p. (in Russian).

3. **Vasenin V. A., Afonin S. A., Kozitsin A. S.** Avtomatizirovannaja sistema tematicheskogo analiza informacii (Automated system of thematic information analysis), *Informacionnye tehnologii, Supplement,* 2009, no. 4, 32 p. (in Russian).

4. **Afonin S. A., Gaspariants A. E.** Avtomaticheskoe postroenie funkcii ocenki kachestva v zadache razreshenija neodnoznachnosti imen avtorov nauchnyh publikacij (Automatic construction of the quality assessment function in the problem of resolving the ambiguity of names of authors of scientific publications), *Programmnaya Ingeneria,* 2015, no. 10, pp. 31—37 (in Russian).

5. **Afonin S. A., Gaspariants A. E.** Razreshenie neodnoznachnosti avtorstva publikacij pri avtomaticheskoj obrabotke bibliograficheskih dannyh. (Resolution of the ambiguity of the authorship of publications in the automatic processing of bibliographic data), *Programmnaya Ingeneria,* 2014, no. 1, pp. 25—29 (in Russian).

6. **Medical**_Subject_Headings, available at: https://ru.wikipedia.org/wiki/Medical_Subject_Headings

7. **Afonin S. A., Lunev K. V.** Vyjavlenie tematicheskih napravlenij v kollekcii naborov kljuchevyh slov (Identify the thematic areas in the collection of keyword sets. Software engineering), *Programmnaya Ingeneria,* 2015, no. 2, pp. 29—39 (in Russian).

УДК 532.517 DOI: 10.17587/prin.8.563-574

В. Г. Лущик, д-р техн. наук, вед. науч. сотр., e-mail: vgl_41@mail.ru,
М. С. Макарова, канд. техн. наук, науч. сотр., e-mail: april27_86@mail.ru,
А. Е. Якубенко, д-р физ.-мат. наук, вед. науч. сотр., e-mail: yakub@imec.msu.ru,
Научно-исследовательский институт механики МГУ имени М. В. Ломоносова

Применение трехпараметрической модели сдвиговой турбулентности для решения задач внешнего обтекания проницаемых поверхностей потоком сжимаемого газа

Представлено описание методики численного исследования задач внешнего обтекания проницаемых поверхностей сжимаемыми газовыми потоками с использованием дифференциальной трехпараметрической модели сдвиговой турбулентности. Описаны результаты многочисленных исследований с использованием предложенной методики, одной из целей проведения которых стала верификация данного метода.

Ключевые слова: *дифференциальная модель турбулентности, сжимаемый пограничный слой, ламинарно-турбулентный переход, вдув (отсос) газа, градиент давления, турбулентное число Прандтля*

Введение

Разработке методов расчета турбулентного пограничного слоя в потоке сжимаемого газа посвящено большое число работ, подробный обзор которых представлен, например, в работе [1]. В инженерной практике наиболее распространены интегральные методы расчета [2, 3], основанные на использовании интегральных уравнений импульса и энергии, замкнутых алгебраическими выражениями для коэффициентов трения и теплообмена и формпараметра. В частности, на их основе разработаны методики расчета турбулентного пограничного слоя в соплах жидкостных ракетных двигателей (ЖРД) [3].

Развитие моделей турбулентности, включающих уравнения переноса для характеристик турбулентности, является еще одним популярным направлением разработки методов расчета турбулентных потоков. Наибольшее применение нашли однопараметрические [4, 5] и двухпараметрические модели [6, 7], которые наряду с уравнениями переноса содержат алгебраические соотношения. В работах [8—12] представлены результаты исследований, направленных на разработку моделей турбулентности, не включающих алгебраические соотношения между характеристиками турбулентности, входящими в модель. Эти модели имеют как общие черты, так и существенные различия, которые рассмотрены в работе [13]. Трехпараметрические модели, включающие члены с вязкостью [8], использовались для расчета широкого класса течений несжимаемой жидкости в приближении пограничного слоя [8, 13, 14].

В настоящей работе представлена обобщенная методика исследования задач внешнего обтекания проницаемых поверхностей сжимаемыми газовыми потоками с использованием дифференциальной трехпараметрической модели сдвиговой турбулентности, предложенной авторами в работах [9, 15]. Приведены результаты более 30 лет исследований в данной области. Среди рассмотренных течений можно отметить расчеты пограничного слоя со вдувом и отсосом газа, с продольным градиентом давления во внешнем потоке, включая участок перехода к турбулентности при большом уровне внешних возмущений и тепловых потоков на стенке, а также с учетом зависимости (далее — переменности) теплофизических свойств газа набегающего потока от температуры и давления.

1. Методика расчета

В настоящем разделе представлено подробное описание вычислительной методики, приведены основные уравнения, характеризующие течение в пограничном слое, а также уравнения и константы используемой авторами модели турбулентности. Описан метод численного исследования, проведено обоснование выбора данной модели на основе сравнительного анализа ряда известных в литературе моделей турбулентности.

1.1. Основные уравнения

Расчет в приближении пограничного слоя двумерных турбулентных течений газов постоянного состава, но переменной температуры требует определения средних значений двух компонент скорости, а также давления и температуры.

Зависимость градиента давления от продольной координаты при расчете пограничных слоев считается заданной. Теплофизические свойства считаются известными функциями давления и температуры. Таким образом, для определения двух компонент скорости и температуры необходимы три уравнения. Этими уравнениями являются уравнения неразрывности, движения и энергии для температуры, которые для стационарного случая имеют следующий вид:

$$\frac{\partial \rho u}{\partial x} + \frac{\partial \rho v}{\partial y} = 0, \tag{1}$$

$$\rho u \frac{\partial u}{\partial x} + \rho v \frac{\partial u}{\partial y} = -\frac{dP}{dx} + \frac{\partial}{\partial y}\left(\eta \frac{\partial u}{\partial y} + \rho\tau\right), \quad \frac{dP}{dx} = f(x), \tag{2}$$

$$\begin{gathered}\rho u C_p \frac{\partial T}{\partial x} + \rho v C_p \frac{\partial T}{\partial y} = \\ = \frac{\partial}{\partial y}\left(\lambda \frac{\partial T}{\partial y} + \rho q_t\right) + \eta\left(\frac{\partial u}{\partial y}\right)^2 + \rho\tau \frac{\partial u}{\partial y} + \Phi,\end{gathered} \tag{3}$$

где x — направление вдоль стенки; y — нормальная координата, отсчитываемая от стенки; u и v — компоненты скорости вдоль осей x и y соответственно. Под величинами ρ, u, T и ρv подразумеваются их средние значения. Так как поперечная компонента скорости в формулах (1) — (3) встречается только в комбинации ρv, то обычно вместо средней поперечной компоненты используется среднее значение произведения ρv. Здесь ρ — плотность; P — давление, u — продольная скорость; C_p — удельная изобарная теплоемкость; T — температура; η — динамическая вязкость. Для расчетов необходимо задать зависимость ρ, η, C_p от температуры и давления; $\rho\tau$ — турбулентное трение $\rho\tau = -\rho\langle u'v'\rangle$; ρq_t — турбулентный поток теплоты $\rho q_t = -\rho\langle v'T'\rangle$.

В систему уравнений (1) — (3) входят характеристики турбулентности и Φ — приток теплоты за счет турбулентных пульсаций скорости, который равен диссипативному члену в уравнении переноса турбулентной энергии.

1.2. Модель турбулентности

Для вычисления величин τ и q_t использована трехпараметрическая модель турбулентности, разработка которой описана в работе [15], обобщенная на течение с теплообменом [16], в которой уравнения переноса записываются для энергии турбулентности $E = \frac{1}{2}\sum \langle u_i'^2 \rangle$, турбулентного напряжения сдвига $\tau = -\langle u'v'\rangle$ и квадрата завихренности турбулентности $\omega = E/L^2$ (L — масштаб турбулентности).

С учетом переменной плотности уравнения для E, τ и ω имеют следующий вид:

$$\begin{gathered}\rho u \frac{dE}{dx} + \rho v \frac{\partial E}{\partial y} = \\ = -\left(c\rho\sqrt{E}L + c_1\eta\right)\frac{E}{L^2} + \frac{\partial}{\partial y}\left(rD_E \frac{\partial E}{\partial y}\right) + \rho\tau \frac{\partial u}{\partial y},\end{gathered} \tag{4}$$

$$\begin{gathered}\rho u \frac{\partial \tau}{\partial x} + \rho v \frac{\partial \tau}{\partial y} = \\ = -\left(3c\rho\sqrt{E}L + 9c_1\eta\right)\frac{\tau}{L^2} + \frac{\partial}{\partial y}\left(D_\tau \frac{\partial \tau}{\partial y}\right) + c_2\rho E \frac{\partial u}{\partial y},\end{gathered} \tag{5}$$

$$\begin{gathered}\rho u \frac{\partial \omega}{\partial x} + \rho v \frac{\partial \omega}{\partial y} = \\ = -\left(2c\rho\sqrt{E}L + 1{,}4c_1\eta f_\omega\right)\frac{\omega}{L^2} + \frac{\partial}{\partial y}\left(D_\omega \frac{\partial \omega}{\partial y}\right) + \\ + \left[\frac{\tau}{E} + 2c_4 \operatorname{sign}\left(\frac{\partial u}{\partial y}\right)\right]\rho\omega \frac{\partial u}{\partial y},\end{gathered} \tag{6}$$

$$D_\varphi = a_\varphi \rho\sqrt{E}L + \alpha_\varphi \eta \left(\varphi = E,\ \tau,\ \omega,\ q_t\right),$$

$$f_\omega = 1 - \frac{1}{2c_1}\left(\frac{L}{E}\frac{\partial E}{\partial y}\right)^2, \quad f(\mathrm{Pr}) = \frac{1+c_5}{2}\frac{\sqrt{\mathrm{Pr}} + 1/\sqrt{\mathrm{Pr}}}{1 + c_5\sqrt{\mathrm{Pr}}}.$$

В приведенных уравнениях используются следующие константы:

$$c = 0{,}3;\ c_1 = 5\pi/4;\ c_2 = 0{,}2;\ c_3 = 0{,}04;\ c_4 = 0{,}235;\ c_5 = 0{,}25;$$

$$a_E = a_\omega = 0{,}06;\ a_\tau = a_q = 3a_E;$$
$$\alpha_E = \alpha_\tau = 1;\ \alpha_\omega = 1{,}4;\ \alpha_q = f(\mathrm{Pr}).$$

Укажем, что из уравнения (4) следует выражение для величины Φ, входящей в уравнение (3):

$$\Phi = \left(c_1\eta + c\rho\sqrt{E}L\right)E/L^2.$$

Необходимо еще определить величину турбулентной температуропроводности $a_t = -\langle \upsilon' T'\rangle/(\partial T/\partial y)$. Для этого вводим величину турбулентной вязкости $\nu_t = -\langle u'\upsilon'\rangle/(\partial u/\partial y)$ и считаем, что $\mathrm{Pr}_t = \nu_t/a_t = 0{,}9$. Это значение получено в работе [17] для логарифмического слоя в диапазоне чисел Прандтля $0{,}02 \leqslant \mathrm{Pr} \leqslant 100$. Таким образом, для определения турбулентного потока теплоты $\rho q_t = -\rho C_p\langle \upsilon' T'\rangle$, входящего в уравнение энергии (3), используется предположение, что турбулентное число Прандтля Pr_t постоянно по толщине пограничного слоя, а именно:

$$\rho q_t = \rho C_p \frac{\nu_t}{\mathrm{Pr}_t}\frac{\partial T}{\partial y} = \rho C_p \frac{\tau}{\mathrm{Pr}_t}\frac{\partial T/\partial y}{\partial u/\partial y}. \tag{7}$$

В случае, когда переменностью турбулентного числа Прандтля пренебречь нельзя, в модели дополнительно используется полученное в работе [18] уравнение для турбулентного потока теплоты:

$$\rho u\frac{\partial q_t}{\partial x}+\rho v\frac{\partial q_t}{\partial y}= \\ =-\left(3c\rho\sqrt{E}L+9c_1\eta f(\mathrm{Pr})\right)\frac{q_t}{L^2}+ \\ +\frac{\partial}{\partial y}\left(D_q\frac{\partial q_t}{\partial y}\right)+c_4\rho EC_p\frac{\partial T}{\partial y}. \quad (8)$$

В этом случае турбулентный поток теплоты находится из решения задачи, а затем по формуле (7) определяется значение турбулентного числа Прандтля Pr_t, которое при определенных условиях может значительно отличаться от значения $\mathrm{Pr}_t = 0{,}9$. Однако такая постановка задачи в статье не рассматривается. Более подробно влияние переменности Pr_t на характеристики течения и теплообмена рассмотрено в работе [19].

1.3. Обоснование выбора модели турбулентности

Авторами трехпараметрической модели [15] были проведены расчеты установившегося течения в кольцевых, плоских и круглых каналах при различных числах Рейнольдса, перехода к турбулентности в пограничном слое при большом уровне внешних возмущений, в пограничных слоях со вдувом и отсосом, а также при положительном и отрицательном градиентах давления. Во всех случаях наблюдалось удовлетворительное качественное согласование результатов расчетов и экспериментов.

Далее перечислим некоторые качественные результаты, полученные в этих расчетах [8, 13] и представленные в работе [16].

1. При развитии течения в кольцевом канале отмечено несовпадение нулевых значений $-\langle u'\upsilon'\rangle$ и $\partial u/\partial y$.

2. При расчете течения на начальном участке круглой трубы получено немонотонное изменение средней скорости на оси.

3. При расчете пограничного слоя со вдувом и отсосом получена соответствующая эксперименту зависимость для коэффициента трения, а также ламинаризация при отсосе.

4. При расчете пограничного слоя с торможением потока определялось положение точки, в которой $\partial u/\partial y$ на стенке равно нулю. По этим расчетам установлено условие безотрывного обтекания, близкое к экспериментальному.

5. Расчеты показывают, что при уменьшении числа Рейнольдса в канале течение приближается к ламинарному.

6. Расчет перехода к турбулентности в пограничном слое при небольшой интенсивности внешних возмущений (3...4 %) показывает, что для некоторого числа Рейнольдса, построенного по толщине потери импульса, трение на стенке соответствует ламинарному режиму течения, а затем энергия турбулентности и трение в пограничном слое начинают возрастать и происходит переход к развитому турбулентному режиму течения.

7. При расчете следа за тонкой пластиной вначале рассчитывался пограничный слой на пластине, а начиная с задней кромки условие прилипания на стенке заменялось равенством нулю производных. Полученные в расчете распределения скорости и турбулентных характеристик совпадают с экспериментальными данными на расстояниях от кромки, равных ~400 толщинам потери импульса.

8. Рассчитывалось течение у стенки, когда скорость в начальном сечении вблизи стенки возрастает линейно в слое конечной толщины от нуля на стенке до значения скорости в линейной части канала. В эксперименте такой профиль скорости создавался входным устройством, размер ячейки которого много меньше толщины слоя с градиентом скорости. В этом случае градиент скорости у стенки возрастает по длине, а градиент средней скорости на участке перехода к развитому турбулентному течению изменяется поперек слоя немонотонно, что соответствует экспериментальным данным.

Таким образом, предложенная в работах [9, 15] трехпараметрическая модель турбулентности прошла всестороннюю проверку при исследовании течения несжимаемой жидкости в каналах, пристеночных и свободных пограничных слоях. Проверка модели подтвердила ее работоспособность и универсальный характер в указанном классе задач. Эти результаты позволяют проводить расчеты без изменения констант, входящих в уравнения для характеристик турбулентности, которые были определены в работах [9, 15] и в последующих расчетах не менялись.

Следующим шагом на пути развития этой модели турбулентности было обобщение ее на случай течений с тепло- и массообменном [16], что позволило вплотную подойти к решению многих актуальных задач для практических приложений.

Обзор работ, посвященный вопросам разработки моделей переноса теплоты (массы) на различных уровнях замыкания, можно найти в работе [1]. Поиск оптимальных решений в направлении выбора модели переноса теплоты должен начинаться с простейших подходов к замыканию уравнения энергии. Для определения турбулентного потока теплоты с использованием гипотезы типа Буссинеска $-\langle\upsilon' T'\rangle = a_t\partial T/\partial y$ необходимо ввести турбулентное число Прандтля Pr_t в определение связи турбулентной

температуропроводности a_t с турбулентной вязкостью ν_t. Обоснованием правомочности такого подхода является то обстоятельство, что немногочисленные измерения характеристик турбулентного переноса импульса и теплоты в неизотермических потоках подтверждают, что значения величин a_t и ν_t всегда одного порядка. Что же касается турбулентной вязкости, то использование для ее определения рассчитанных на основе трехпараметрической модели турбулентности полей турбулентного трения и скорости позволяет надеяться на достоверность полученных результатов. При этом Pr_t полагается величиной, постоянной по всей толщине сдвигового слоя, хотя это предположение все чаще подвергается критике [19].

Для подтверждения работоспособности изложенного выше подхода были проведены расчеты в широком диапазоне изменения определяющих параметров — чисел Прандтля и Рейнольдса и установлены границы его применимости [16]. В работе [16] рассмотрен теплообмен при течении среды с постоянными и переменными физическими свойствами. Учет эффектов сжимаемости потребовал дополнения соответствующими членами уравнений для характеристик турбулентности.

Сравнение результатов расчета с известными экспериментальными данными в диапазоне чисел Рейнольдса $Re = 10^4...10^6$, Прандтля $Pr = 10^{-3}...10$, Маха $M = 0...4$ оказалось вполне удовлетворительным.

1.4. Сравнительный анализ различных (интегральных, алгебраических дифференциальных) моделей турбулентности

В последнее время в инженерной практике расчетов турбулентных пограничных слоев наряду с традиционными интегральными и алгебраическими методами все большее распространение получают методы, базирующиеся на дифференциальных моделях турбулентности, в основе которых лежат идеи А. Н. Колмогорова [20], связанные с использованием уравнений переноса для характеристик турбулентности.

Обзору дифференциальных моделей турбулентности посвящен ряд работ, список которых содержится в работе [21]. В большинстве таких работ представлены аналитические обзоры моделей, не содержащие результатов численного исследования на примере какого-либо течения, расчет которого выполнен по рассматриваемым моделям в рамках одной программы расчета. Исключение составляет обзор [22], в котором проанализировано восемь двухпараметрических моделей турбулентности и с их использованием проведен расчет турбулентного пограничного слоя с нулевым и положительным градиентом давления. Приведенное в работе [22] численное исследование позволило рекомендовать для использования ряд моделей, которые продемонстрировали удовлетворительное совпадение с экспериментом. Ограниченный характер обзора [22] заключается в том, что в нем рассмотрены только двухпараметрические модели, а в качестве теста выбран пограничный слой несжимаемой жидкости.

В работах [23, 24] предпринята попытка расширения круга моделей, которые используются в расчетах в настоящее время, и их сравнение при решении тестовой задачи, в качестве которой выбран турбулентный пограничный слой в сопле ЖРД. Такой выбор объясняется тем обстоятельством, что эта задача представляет собой комплексный тест, в котором существенными являются эффекты сжимаемости, обусловленные большими значениями величин теплового потока в стенку, числа Маха потока и разности температур стенки сопла и газа, а также присутствует значительный отрицательный градиент давления.

Для проведения сравнительного анализа выбраны следующие модели турбулентности: интегральная модель [25]; алгебраическая модель для турбулентной вязкости ν_t [26]; однопараметрическая модель [5], в которой для ν_t используется уравнение переноса; группа двухпараметрических моделей [27—30], в которых используются уравнения переноса для энергии турбулентности E и скорости диссипации энергии турбулентности или "частоты" турбулентности ω (или ω^2); трехпараметрическая модель [15], в которой уравнения переноса записаны для E, ω^2 и напряжения сдвига τ.

Для развитого течения несжимаемой жидкости в круглой трубе по всем рассмотренным моделям турбулентности в пределах разброса опытных данных наблюдается соответствие эксперименту. Однако физически более обоснованными представляются модели, описанные в работах [15, 27], которые в вязком подслое дают необходимые степенные зависимости $E \sim y^2$, $L \sim y$, $\tau \sim y^{3-4}$.

Расчет сжимаемого до- и сверхзвукового пограничного слоя в сопле ЖРД показал, что для адиабатической стенки результаты расчетов по всем моделям близки между собой. В случае охлаждаемой стенки, когда существенно влияние температурного фактора, результаты расчета различаются. Полученные результаты сравнивали с результатами расчета по трехпараметрической модели [15]. Для количественной оценки моделей использовали значения величин потерь удельного импульса из-за трения ξ_f на выходе из сопла и теплового потока в стенку Q_w в критическом сечении сопла, имеющие определяющее значение для ЖРД. По этим оценкам лучшие результаты имеют однопараметрическая модель [5] и интегральная методика [25], отличие которых по ξ_f и Q_w от трехпараметрической модели [15] не превышает 3...6 %. Из двухпараметрических моделей приемлемой можно считать лишь модель, описанную

в работе [27]. Для этой модели и алгебраической модели [26] отличие не превышает 8...9 %. Что же касается двухпараметрических моделей [28, 29], то они не могут быть рекомендованы для расчета пограничного слоя в соплах ЖРД, так как их отличие по ξ_f и Q_w достигает 30...40 %.

Таким образом, проведенное авторами модели турбулентности численное исследование показало, что трехпараметрическая дифференциальная модель [15], как наиболее обоснованная в сравнении с физическим экспериментом, апробированная и достаточно универсальная для сжимаемого пограничного слоя, может быть использована для пограничного слоя сжимаемого газа с различными граничными условиями.

1.5. Особенности численного решения

Система уравнений (1)—(7) с соответствующими граничными условиями, определяемыми типом рассматриваемой задачи, при неизменных значениях констант решалась численно методом прогонки с итерациями. Расчеты проводили на неравномерной сетке. Шаг по координате y определялся соотношениями

$$y = \delta\left[1 - \operatorname{cth} Q \operatorname{th}\left(Q(1-\varsigma)\right)\right],$$

$$\Delta y = \frac{\delta Q \operatorname{cth} Q\Delta\varsigma}{\operatorname{ch}^2 Q(1-\varsigma)}\left(0 \leqslant \varsigma \leqslant 1,\ \Delta\varsigma = \text{const}\right),$$

где δ — толщина пограничного слоя, определяемая с помощью соответствующего текущей задаче граничного условия при $y = \delta$ из условия гладкого сопряжения решений. Выбирая значение Q, можно добиться того, чтобы шаг Δy вблизи стенки был малым. Шаг по x был достаточно мал в сечениях, близких к входному сечению, и увеличивался по мере продвижения вниз по потоку, что позволило проводить расчеты даже при достаточно малом уровне турбулентности во внешнем потоке (порядка 1 %).

Для расчетов по указанной численной методике была разработана подпрограмма MAIN, блок-схема которой представлена на рис. 1.

Начало программы и последующее управление итерационным процессом осуществляется с помощью основной подпрограммы MAIN. MAIN вызывает подпрограммы, выполняющие расчеты, результаты которых используются на всем протяжении счета, а именно подпрограмму GRID, которая рассчитывает и запоминает координаты узлов сетки; подпрограммы INIT и INUP, которые рассчитывают граничные условия на входе в расчетный участок; подпрограмму STEP, которая позволяет изменять шаг интегрирования и число шагов интегрирования между соседними печатями; подпрограмму CHGR, которая позволяет увеличить расчетную область в поперечном направлении в процессе расчета при неизменном числе точек разностной сетки; подпрограмму CHSF, которая рассчитывает профили искомых переменных в месте вдува или отсоса газа в пристеночном слое канала.

В этой же подпрограмме MAIN рассчитываются интегральные характеристики пристеночного слоя. Один полный цикл итераций выполняется каждый раз, когда MAIN вызывает подпрограмму SOLUTN.

Подпрограмма SOLUTN содержит шесть подциклов: в первом рассчитывается скорость, во втором — энтальпия, в третьем — концентрация, в четвертом энергия турбулентности, в пятом — турбулентное касательное напряжение, в — шестом турбулентный вихрь.

Текст программы написан на языке Fortran.

2. Результаты численного моделирования

В этом разделе представлены результаты численного моделирования с использованием предложенной методики для решения задачи о ламинарно-турбулентном переходе и течения в сверх- и дозвуковых пограничных слоях газовых потоков с переменными теплофизическими свойствами.

2.1. Ламинарно-турбулентный переход при большом (>1 %) уровне внешних возмущений и влияние на переход продольного градиента давления, числа Маха и температурного фактора

При решении задачи о ламинарно-турбулентном переходе (рис. 2) для замыкания системы уравнений (1)—(7) использовали следующие граничные условия:

На стенке со стороны газа набегающего потока $y = 0$:

$$u = 0,\ v = 0,\ T = T_w(x),\ E = \partial E/dy = \tau = q_t = 0.$$

На внешней границе пограничного слоя $y = \delta(x)$:

$$u = u_1(\text{M} = \text{const}),\ T = T_1,\ E = E(x),$$
$$\tau = q_t = 0,\ dP/dx = f(x).$$

Переход к турбулентности в пограничных слоях на стенках теплообменных аппаратов характеризуется следующими основными особенностями: повышенным уровнем турбулентности потока на входе; существенной разностью температур потока и охлаждаемых стенок; значительными градиентами давления.

Исследования перехода к турбулентности в пограничном слое при низком уровне внешних возмущений показали [31], что результаты линейной теории устойчивости согласуются с экспериментальными данными. Однако в работе [31] отмечено, что остается открытым вопрос о том, является ли

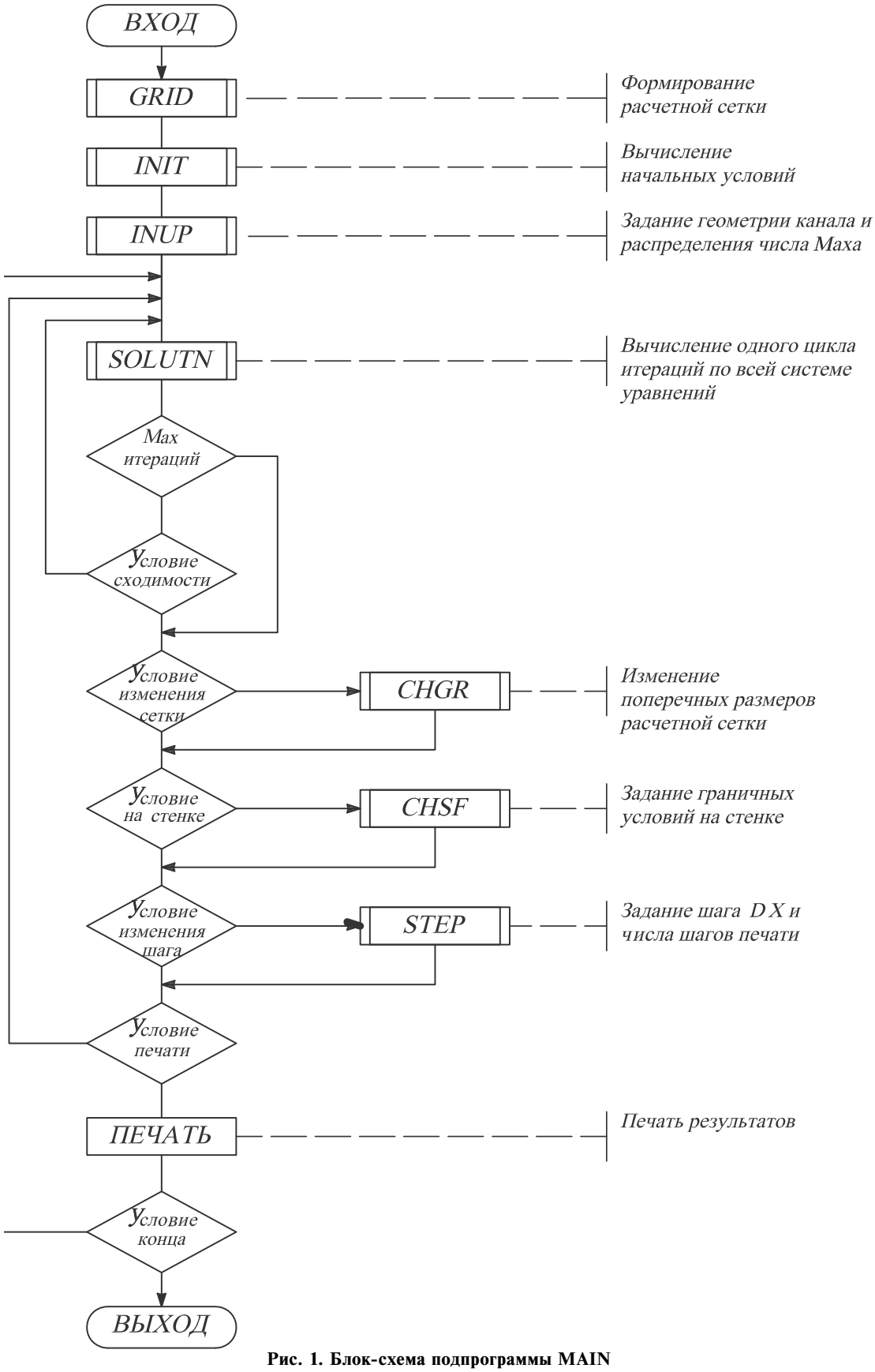


Рис. 1. Блок-схема подпрограммы MAIN

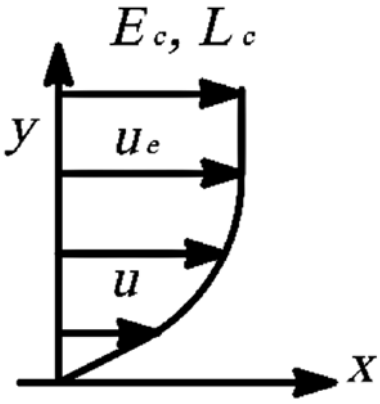


Рис. 2. Схема течения

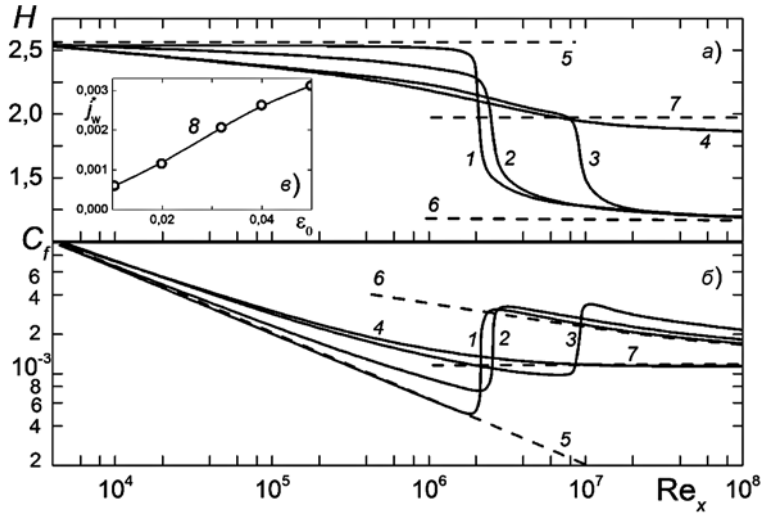


Рис. 3. Влияние параметра отсоса $u \sim v$ $j_w^0 = v_w/u_e$:
а — на зависимость формпараметра от числа Рейнольдса $H(\mathrm{Re}_x)$; *б* — на зависимость $C_f(\mathrm{Re}_x)$, а также зависимость критического значения j_w^* (*в*) от ε_0; линии *1—4* — расчет для $j_w^0 \cdot 10^3 = 0; 0{,}2; 0{,}5; 0{,}6$, соответственно при $\varepsilon_0 = 0{,}01$; линии *5—7* — зависимости Блазиуса, Кармана — Шенхерра и Шлихтинга [31] соответственно, линия и точки *8* — расчет

единственным механизм перехода к турбулентности, связанный с развитием волн неустойчивости согласно линейной теории.

Одним из альтернативных механизмов перехода к турбулентности в потоках с градиентом скорости является развитие пульсаций скорости, порожденных входными устройствами. Масштаб этих пульсаций может быть меньше поперечного размера слоя с градиентом скорости и, следовательно, в линейном приближении поток устойчив к этим пульсационным движениям. Анализ поведения турбулентности в потоке с градиентом скорости показал, что в этом случае энергия турбулентности может вначале уменьшаться, а затем возрастать [32].

Экспериментальные исследования перехода к турбулентности при высоком уровне внешних возмущений показали [31, 33], что при интенсивности внешней турбулентности выше 0,5 % переход к турбулентности не определяется развитием волн Толмина — Шлихтинга. В работах [34, 35] приведены результаты экспериментальных исследований длины участка перехода при различной интенсивности турбулентности во внешнем потоке.

В работе [36] были выполнены расчеты течения на участке перехода к турбулентности с применением трехпараметрической модели турбулентности, которая была использована без каких-либо изменений. Исследовано влияние на переход начальной интенсивности внешней турбулентности ε_0, температурного фактора θ, числа Маха M набегающего потока и параметра отсоса j_w^0. С ростом значений величин ε_0, θ, M, j_w^0 число Рейнольдса перехода как по толщине потери импульса Re_θ^*, так и по длине Re_x^* уменьшается, что соответствует как качественным соотношениям, так и эксперименту. Для проницаемой платины с отсосом показано, что существует критический параметр отсоса j_w^*, при достижении которого реализуется ламинарный режим течения в пограничном слое (рис. 3, *а*, *б*, линия *4*). Полученная расчетная зависимость свидетельствует о том, что с ростом значения величины ε_0 значение критического параметра отсоса возрастает (рис. 3, *в*)

В работе [37] приведены результаты численного исследования перехода к турбулентности в пограничном слое на пластине с отрицательным градиентом давления в потоке с большим уровнем внешних возмущений. Исследовано влияние на переход градиента давления $K = -\left(\nu/\rho u_e^3\right)\dfrac{dp}{dx}$ для значений начальной интенсивности внешней турбулентности $\varepsilon_0 = 0{,}01...0{,}05$. Показано, что число Рейнольдса перехода по толщине потери импульса Re_θ^* с ростом значения параметра K и формпараметра Польгаузена $\Lambda_\theta = K \cdot \mathrm{Re}_\theta$ (используется в литературе по исследованию устойчивости пограничного слоя с градиентом давления) изменяется очень слабо. Отличие расчетных зависимостей $\mathrm{Re}_\theta^*(\Lambda_0)$ во всем исследованном диапазоне ε_0 (точки *1—4* на рис. 4) от экспериментальных данных [34] не превосходит 23 % при $\varepsilon_0 = 0{,}05$ и 8 % при $\varepsilon_0 = 0{,}02$. При этом с ростом интенсивности внешних возмущений ε_0 критическое значение Λ_θ^* изменяется слабо, тогда как параметр K^* существенно возрастает (рис. 5), что подтверждают экспериментальные результаты [34].

В работах [38, 39] по сравнению с работой [37] был расширен диапазон изменения интенсивности турбулентности внешнего потока, а также проведено исследование влияния теплообмена на переход к турбулентности в пограничном слое. Проведено сравнение результатов расчетов с имеющимися немногочисленными экспериментальными данными и

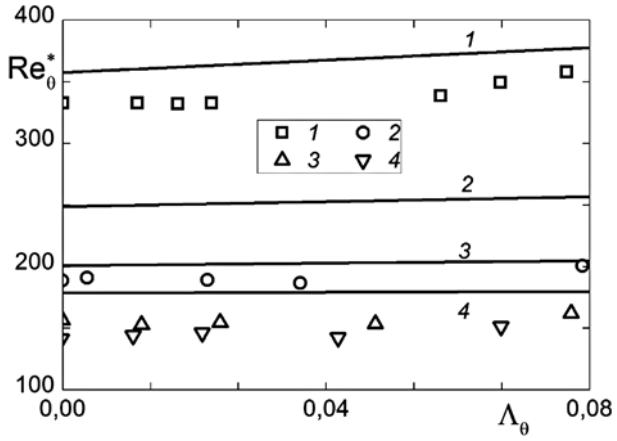


Рис. 4. Экспериментальные (линии) и расчетные (точки) зависимости нижнего критического числа Рейнольдса Re^*_θ от Λ_θ для ε_0 = 0,02; 0,03; 0,04; 0,05 (линии *1—4* соответственно)

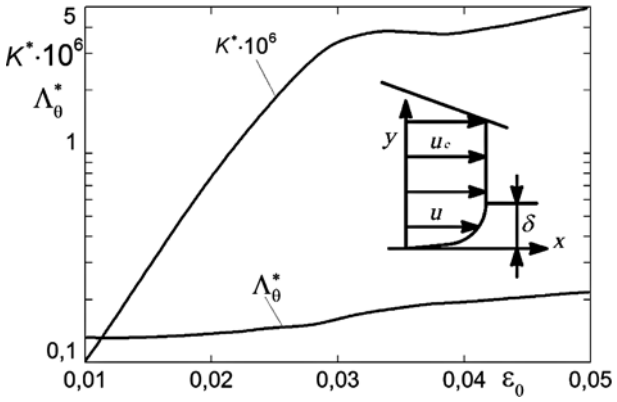


Рис. 5. Расчетная зависимость критических значений параметров K и Λ_θ от ε_0

отмечено удовлетворительное согласование расчета с экспериментом.

2.2. Течение и теплообмен в пограничном слое при течении сред с постоянными и переменными физическими свойствами

Граничные условия в данной постановке (рис. 6) имеют следующий вид:

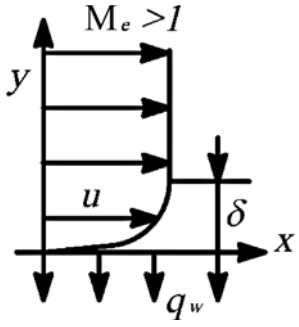


Рис. 6. Схема течения

На стенке со стороны газа набегающего потока $y = 0$:

$$u = 0,\ v = 0,\ q_w = -\left(\lambda \partial T/\partial y\right)_w = f(x)$$

$$\text{или}\ \ T = T_w(x),\ \ E = \partial E/\partial y = \tau = q_t = 0.$$

На внешней границе пограничного слоя $y = \delta(x)$:

$$u = u_1,\ \ T = T_1,\ \ E = E(x),\ \ \tau = q_t = 0.$$

Необходимость учета переменности теплофизических свойств газа теплоносителя, вызванная экстремальными условиями по температуре, давлению и скорости, появляется при расчетах в ряде технических приложений (сопла ЖРД, камеры сгорания и т. д.). Поэтому от методик требуется повышенная точность определения характеристик течения и теплообмена в пограничном слое и достаточная универсальность для учета изменения граничных условий. В работе [40] было предложено отказаться от использования многопараметрических обобщенных зависимостей и находить их непосредственно по разработанной авторами методике, позволяющей достаточно точно рассчитывать характеристики течения и теплообмена в сверхзвуковом пограничном слое с переменными теплофизическими свойствами.

Результаты расчета зависимости $C_f(\mathrm{Re}_\theta)$ в сравнении с известными зависимостями Кармана — Шенхерра, Никурадзе и Блазиуса [31] и экспериментальными данными представлены на рис. 7. Расчеты проведены при T_e = 300 K (Pr_e = 0,71 — воздух), M_e = 0,005 для двух значений интенсивности начальной турбулентности в набегающем потоке ε_0 = 0,01 и 0,05 (линии *4*

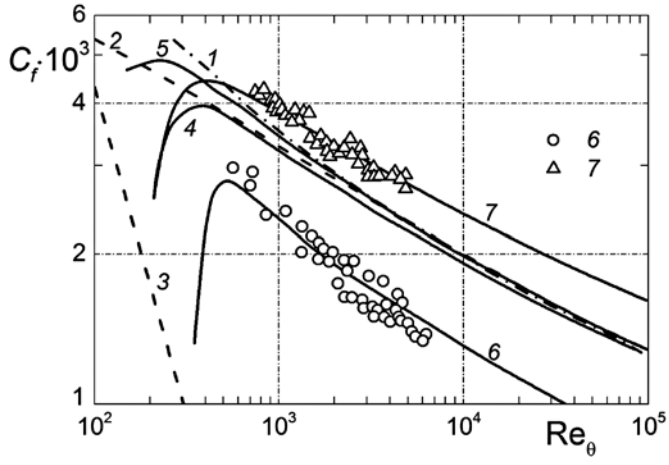


Рис. 7. Зависимость коэффициента трения $C_f(\mathrm{Re}_\theta)$:

линии *1—3* — зависимости Кармана—Шенхерра, Никурадзе и Блазиуса [31] соответственно; *4*, *5* — расчет при ε_0 = 0,01 и 0,05 соответственно для пограничного слоя несжимаемой жидкости; *6* — эксперимент (точки) и расчет для сверхзвукового (M_e = 2,5) пограничного слоя теплоизолированной стенки; *7* — эксперимент (точки) и расчет для дозвукового (M_e = 0,06) пограничного слоя на охлаждаемой стенке при значениях температурного фактора θ = 0,39...0,48 и 0,45

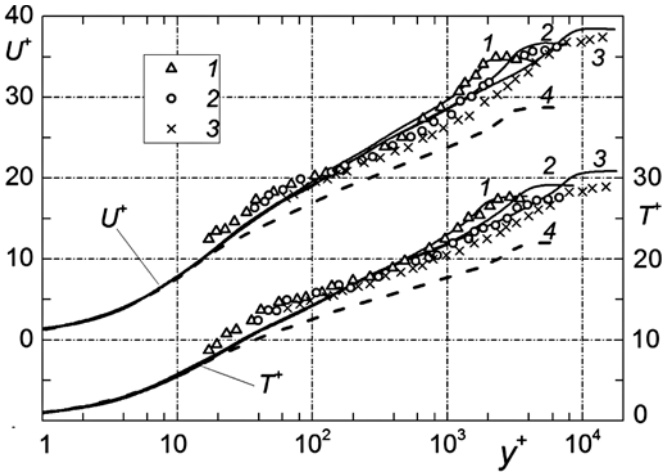


Рис. 8. Профили скорости U^+ и температуры T^+ в дозвуковом (M_e = 0,06) пограничном слое на охлаждаемой стенке:

линии *1—3* — $\mathrm{Re}_\theta^* \cdot 10^3$ = 2,7; 5,2; 10,5 соответственно; точки *1—3* — эксперимент при θ = 0,41...0,47 и 0,45 соответственно; линии *4* — расчет для теплоизолированной стенки

и *5* соответственно). Результаты расчета различаются в переходной области. При больших значениях это различие Re_θ мало и расчетные зависимости близки к зависимостям Кармана — Шенхерра и Никурадзе для турбулентного пограничного слоя [31].

Тепловые граничные условия допускают большее разнообразие, чем динамические. На поверхности обтекаемого газом тела можно задать не только постоянную и переменную температуру, но и поток тепла, т. е. градиент температуры на стенке. Частным случаем является отсутствие теплоты, т. е. $\partial T/\partial y = 0$ при $\partial y = 0$, что приводит к задаче о теплоизолированной стенке. Эта задача занимает особое место в литературе о температурных пограничных слоях (см. [3, 41]) и ей посвящено много теоретических и экспериментальных работ.

Сравнение результатов расчета для коэффициента трения $C_f(\mathrm{Re}_\theta)$ с экспериментальными данными для случая теплоизолированной стенки как в дозвуковом, так и сверхзвуковом потоке газа (M_e = 2,5), показало их хорошее согласование (линия и точки *6* на рис. 7). С ростом числа Маха коэффициент трения возрастает, что также подтверждено расчетами.

Температура стенки также оказывает значительное влияние на интегральные и локальные характеристики пограничного слоя. Так, расчет, проведенный для случая охлаждаемой стенки, показал, что охлаждение приводит к росту коэффициента трения C_f (линия и точки *7*, рис. 7).

Охлаждение стенки также меняет вид профилей скорости U^+, и температуры T^+ (рис. 8). Полученные результаты расчета для различных значений Re_θ при хорошем согласовании с экспериментом лежат выше соответствующих профилей для теплоизолированной стенки (линии *4*).

Заключение

С использованием трехпараметрической дифференциальной модели турбулентности разработана методика численного исследования задач внешнего обтекания непроницаемых и проницаемых поверхностей сжимаемыми газовыми потоками. Проведены многочисленные исследования самых различных сложных турбулентных течений, которые показали хорошую точность и универсальность модели и позволили обнаружить в исследованных течениях ряд интересных особенностей и эффектов. Данная методика была использована в расчетах перехода к развитой турбулентности в пограничном слое и каналах при большом уровне внешних возмущений. Правильно описан характер перехода и с приемлемой для практики точностью получена количественная информация как по интегральным, так и по локальным характеристикам перехода.

Для течений в пограничных слоях с разнообразными граничными условиями получены экспериментально подтвержденные результаты расчетов в широком диапазоне параметров, вплоть до экстремальных (отсос и ускорение потока — до ламинаризации, торможение — до отрыва пограничного слоя), что имеет важное значение для практики и свидетельствует об универсальности используемой модели турбулентности.

Было предложено вместо использования многопараметрических обобщенных зависимостей в инженерных приложениях определять их непосредственно по разработанной методике, позволяющей достаточно точно рассчитывать характеристики течения и теплообмена в сжимаемом пограничном слое с постоянными и переменными теплофизическими свойствами.

Работа поддержана РФФИ (грант № 17-08-00115) и Советом по грантам Президента РФ (МК-6025.2016.8).

Список литературы

1. **Cebeci T.** Analysis of Turbulent Flows with Computer Programs (Third edition). Butterworth-Heinemann, Oxford, UK, 2013.

2. **Кутателадзе С. С., Леонтьев А. И.** Тепломассообмен и трение в турбулентном пограничном слое. 2-е изд., перераб. М.: Энергоатомиздат, 1985. 319 с.

3. **Иевлев В. М.** Турбулентное движение высокотемпературных сплошных сред. М.: Наука, 1975. 256 с.

4. **Глушко Г. С.** Турбулентный пограничный слой на плоской пластине в несжимаемой жидкости // Изв. АН СССР. Механика. 1965. № 4. С. 13—23.

5. **Секундов А. Н.** Применение дифференциального уравнения для турбулентной вязкости к анализу плоских неавтомодельных течений // Изв. АН СССР. МЖГ. 1971. № 5. С. 114—127.

6. **Глушко Г. С.** Дифференциальное уравнение для масштаба турбулентности и расчет турбулентного пограничного слоя на плоской пластине // Турбулентные течения. М.: Наука, 1970. С. 37—44.

7. **Jones W. P., LaunderB. E.** The calculation no flow-Reynolds-number phenomena with a two-equation model of turbulence // Int. J. Heat and Mass Transfer. 1973. Vol. 16, N 6. P. 1119—1130.

8. **Лущик В. Г., Павельев А. А., Якубенко А. Е.** Исследование сдвиговых течений с использованием трехпараметрической модели турбулентности // 5-й Всесоюз. съезд по теоретической и прикладной механике. Алма-Ата: Наука, 1981. С. 241—242.

9. **Лущик В. Г., Павельев А. А., Якубенко А. Е.** Модель сдвиговой турбулентности // 4-й Всесоюзн. съезд по теоретической и прикладной механике. Киев: Наук. думка, 1976. С. 66.

10. **Hanjalic K., Launder B. E.** A Reynolds stress model of turbulence and its application to thin shear flows // J. Fluid Mech. 1972. Vol. 52, N 4. P. 609—638.

11. **Hanjalic K., Launder B. E.** Contribution towards a Reynolds stress closure for low-Reynolds-number turbulence // J. Fluid Mech. 1976. Vol. 74, N 4. P. 593—610.

12. **Павельев А. А.** Развитие решеточной турбулентности в потоке с постоянным градиентом скорости // Изв. АН СССР. МЖГ. 1974. № 1. С. 38—47.

13. **Лущик В. Г., Павельев А. А., Якубенко А. Е.** Исследование перехода к турбулентности в пограничном слое при большой интенсивности внешних возмущений с помощью трехпараметрической модели // Проблемы современной механики. Ч. 1. М.: Изд-во МГУ, 1983. С. 127—138.

14. **Ерошенко В. М., Ершов А. В., Зайчик Л. И.** Расчет турбулентного течения несжимаемой жидкости в круглой трубе с отсосом газа через пористые стенки // Изв. АН СССР. МЖГ. 1982. № 4. С. 87—93.

15. **Лущик В. Г., Павельев А. А., Якубенко А. Е.** Трехпараметрическая модель сдвиговой турбулентности // Изв. АН СССР. МЖГ. 1978. № 3. С. 13—25.

16. **Лущик В. Г., Павельев А. А., Якубенко А. Е.** Трехпараметрическая модель турбулентности. Расчет теплообмена // Изв. АН СССР. МЖГ, 1986. № 2. С. 40—52.

17. **Гиневский А. С., Иоселевич В. А., Колесников А. В.** и др. Методы расчета турбулентного пограничного слоя // Итоги науки и техники. Сер. Механика жидкости и газа. М.: ВИНИТИ, 1978. Т. 11. С. 155—304.

18. **Лущик В. Г., Павельев А. А., Якубенко А. Е.** Уравнение переноса для турбулентного потока тепла. Расчет теплообмена в трубе // Изв. АН СССР. МЖГ. 1988. № 6. С. 42—50.

19. **Kays W. M.** Turbulent Prandtl number — where are we? // Trans. ASME. J. Heat Transfer. 1994. Vol. 116. P. 284—295.

20. **Колмогоров А. Н.** Уравнения турбулентного движения несжимаемой жидкости // Изв. АН СССР. Сер. Физ. 1942. Т. 6, № 1—2. С. 56—58.

21. **Лущик В. Г., Павельев А. А., Якубенко А. Е.** Уравнения переноса для характеристик турбулентности: модели и результаты расчетов // Итоги науки и техники. Сер. Механика жидкости и газа. М.: ВИНИТИ, 1988. Т. 22. С. 3—61.

22. **Patel V. C., Rodi W., Scheuerer G.** Turbulence models for near-wall and low Reynolds number flows: A review // AIAA Journal, 1985. Vol. 23, N 9. P. 1308—1319.

23. **Лущик В. Г., Павельев А. А., Якубенко А. Е.** Турбулентные течения. Модели и численные исследования (обзор) // Изв. РАН. МЖГ. 1994. № 4. С. 4—27.

24. **Лущик В. Г., Якубенко А. Е.** Сравнительный анализ моделей турбулентности для расчета пристенного пограничного слоя // Изв. РАН. МЖГ. 1998. № 1. С. 44—58.

25. **Авдуевский В. С.** Метод расчета пространственного турбулентного пограничного слоя в сжимаемом газе // Изв. АН СССР. ОТН. Механика и машиностроение. 1962. № 4. С. 3—13.

26. **Cebeci T., Smith A. M. O.** Analysis of turbulent boundary layers. N. Y.: Acad. Press, 1974. 404 p.

27. **Chien K. Y.** Predictions of channel and boundary layer flows with a low-Reynolds-number model // AIAA Journal. 1982. Vol. 20, N 1. P. 33—38.

28. **Launder B. E., Sharma B. I.** Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc // Lett. Heat and Mass Transfer. 1974. Vol. 1, N 2. P. 131—138.

29. **Vilcox D. C.** Reassessment of the scale determining equation for advanced turbulence models // AIAA Journal. 1988. Vol. 26, N 11. P. 1299—1310.

30. **Vilcox D. C., Rubesin W. M.** Progress in turbulence modeling for complex flow fields including effects of compressibility. NASA Tech. Paper. 1980. № 1517. 70 p.

31. **Шлихтинг Г.** Теория пограничного слоя / Пер. с нем. М.: Наука, 1974. 712 с.

32. **Павельев А. А.** О переходе к турбулентности в струях // Турбулентные течения. М.: Наука, 1974. С. 185—193.

33. **Поляков Н. Ф.** Ламинарный пограничный слой в условиях "естественного" перехода к турбулентному течению // Развитие возмущений в пограничном слое. Новосибирск: Наука, 1979. С. 23—67.

34. **Abu-Ghannam B. J., Shaw R.** Natural transition of boundary layers — the effects of turbulence, pressure gradient, and flow history // J. Mech. Eng. Science. 1980. Vol. 22, N 5. P. 213—228.

35. **Коляда В. В., Павельев А. А.** О смене механизмов перехода к турбулентности в пограничном слое // Изв. АН СССР. МЖГ, 1986. № 3. С. 179—181.

36. **Лущик В. Г., Павельев А. А., Решмин А. И., Якубенко А. Е.** Влияние граничных условий на переход к турбулентности в пограничном слое на пластине при большом уровне внешних возмущений // Изв. РАН. МЖГ. 1999. № 6. С. 111—119.

37. **Лущик В. Г., Павельев А. А., Якубенко А. Е.** Переход к турбулентности в пограничном слое на пластине при отрицательном градиенте давления во внешнем потоке // Изв. РАН. МЖГ. 2004. № 2. С. 91—102.

38. **Лущик В. Г., Макарова М. С., Якубенко А. Е.** Влияние уровня турбулентности набегающего потока на характеристики течения пограничного слоя на пластине // Вестник Нижегородского университета. 2011. № 4. Ч. 3. С. 945—947.

39. **Лиознов Г. Л., Лущик В. Г., Макарова М. С., Якубенко А. Е.** Влияние турбулентности набегающего потока на течение и теплообмен в пограничном слое на пластине // Изв. РАН, МЖГ. 2012. № 5. С. 40—42.

40. **Лущик В. Г., Якубенко А. Е.** Сверхзвуковой пограничный слой на пластине. Сравнение расчета с экспериментом // Изв. РАН. МЖГ. 1998. № 6. С. 64—78.

41. **Леонтьев А. И., Лущик В. Г., Макарова М. С.** Коэффициент восстановления температуры в пограничном слое на проницаемой пластине // ТВТ. 2017. Т. 55, № 2. С. 255—261.

Application of the Three-Parameter Model of Shear Turbulence for Solving Problems of External Flowing on Permeable Surfaces by a Compressible Gas Flow

V. G. Lushchik, vgl_41@mail.u, **M. S. Makarova,** april27_86@mail.ru,
A. E. Yakubenko, yakub@imec.msu.ru, Lomonosov Moscow State University, Institute of mechanics, Moscow, 119192, Russian Federation

Corresponding author:

Makarova Mariia S., Researcher, Lomonosov Moscow State University, Institute of mechanics, Moscow, 119192, Russian Federation,
E-mail: april27_86@mail.ru

Received on September 27, 2017
Accepted on October 09, 2017

In this paper, a technique for numerical investigation of problems of external flowing on permeable surfaces by compressible gas flows using a differential three-parameter model of shear turbulence is described. The results of numerous studies using the proposed technique are presented, one of the goals of that presentation is the verification of this method.

Keywords: *differential turbulence model, compressible boundary layer, laminar-turbulent transition, gas injection (suction), pressure gradient, turbulent Prandtl number*

Acknowledgements: *This work was supported by Russian Foundation for Basic Research (project no 17-08-00115) and the Grant Council of the President of the Russian Federation (MK-6025.2016.8).*

For citation:

Lushchik V. G., Makarova M. S., Yakubenko A. E. Application of the Three-Parameter Model of Shear Turbulence for Solving Problems of External Flowing on Permeable Surfaces by a Compressible Gas Flow, *Programmnaya Ingeneria,* 2017, vol. 8, no. 12, pp. 563—574.

DOI: 10.17587/prin.12.563-574

References

1. **Cebeci T.** *Analysis of Turbulent Flows with Computer Programs* (3rd edition), Butterworth-Heinemann, Oxford, UK, 2013.

2. **Kutateladze S. S., Leontiev A. I.** *Teplomassoobmen i trenie v turbulentnom pogranichnom sloe* (Heat and Mass Transfer and Friction in a Turbulent Boundary Layer) (2nd edition), Moscow, Energoatomizdat, 1985, 319 p. (in Russian).

3. **Ievlev V. M.** *Turbulentnoe dvizhenie vysokotemperaturnyh sploshnyh sred* (Turbulent motion of high-temperature continuous media), Moscow, Nauka, 1975, 256 p. (in Russian).

4. **Glushko G. S.** Turbulentnyj pogranichnyj sloj na ploskoj plastine v neszhimaemoj zhidkosti (A turbulent boundary layer on a flat plate in an incompressible fluid), *Izv. AN SSSR, Mechanics*, 1965, no. 4. pp. 13—23.

5. **Sekundov A. N.** Application of a differential equation for turbulent viscosity to the analysis of plane non-self-similar flows, *J. Fluid Dyn.,* 1971, vol. 6, no 5, pp. 828—840.

6. **Glushko G. S.** Differencial'noe uravnenie dlja masshtaba turbulentnosti i raschet turbulentnogo pogranichnogo sloja na ploskoj plastine (Differential equation for the scale of turbulence and calculation of the turbulent boundary layer on a flat plate), *Turbulent flows,* Moscow, Nauka, 1970, pp. 37—44 (in Russian).

7. **Jones W. P., Launder B. E.** The calculation of low-Reynolds-number phenomena with a two-equation model of turbulence, *Int. J. Heat and Mass Transfer,* 1973, vol. 16, no. 6, pp. 1119—1130.

8. **Lushchik V. G., Pavel'ev A. A., Yakubenko A. E.** Issledovanie sdvigovyh techenij s ispol'zovaniem trehparametricheskoj modeli turbulentnosti (Investigation of shear flows using the three-parameter model of turbulence), *5th All-Union Congress on Theoretical and Applied Mechanics*, Alma-Ata, Nauka, 1981, pp. 241—242 (in Russian).

9. **Lushchik V. G., Pavel'ev A. A., Yakubenko A. E.** Model' sdvigovoj turbulentnosti (The model of shear turbulence), *4th All-Union Congress on Theoretical and Applied Mechanics*, Kiev, Nauk. Dumka, 1976, pp. 66 (in Russian).

10. **Hanjalic K., Launder B. E.** A Reynolds stress model of turbulence and its application to thin shear flows, *J. Fluid Mech.,* 1972, vol. 52, no. 4, pp. 609—638.

11. **Hanjalic K., Launder B. E.** Contribution towards a Reynolds stress closure for low-Reynolds-number turbulence, *J. Fluid Mech.*, 1976, vol. 74, no. 4, pp. 593—610.

12. **Pavel'ev A. A.** Development of lattice turbulence in a stream with a constant velocity gradient, *J. Fluid Dyn.*, 1974, vol. 9, no. 1, pp. 28—34.

13. **Lushchik V. G., Pavel'ev A. A., Yakubenko A. E.** Issledovanie perehoda k turbulentnosti v pogranichnom sloe pri bol'shoj intensivnosti vneshnih vozmushhenij s pomoshh'ju trehparametricheskoj modeli (Investigation of the transition to turbulence in the boundary layer with a large intensity of external perturbations by means of a three-parameter model), *Problems of Modern Mechanics. Part 1,* Moscow, MGU, 1983, pp. 127—138 (in Russian).

14. **Eroshenko V. M., Ershov A. V., Zaichik L. I.** Calculation of turbulent flow of an incompressible fluid in a circular tube with suction through porous walls, *J. Fluid Dyn*., 1982, vol. 17, no. 4, pp. 559—564.

15. **Lushchik V. G., Pavel'ev A. A., Yakubenko A. E.** Three-parameter model of shear turbulence, *J. Fluid Dyn*., 1978, vol. 13, no. 3, pp. 350—360.

16. **Lushchik V. G., Pavel'ev A. A., Yakubenko A. E.** Three-parameter model of turbulence: Heat transfer calculations, *J. Fluid Dyn*., 1986, vol. 21, no. 2, pp. 200—211.

17. **Ginevsky A. S., Ioselevich V. A., Kolesnikov A. V., Lapin Yu. V., Pilipenko V. N., Secundov A. N.** Metody rascheta turbulentnogo pogranichnogo sloja **(**Methods for calculating the turbulent boundary layer), *Itogi Nauki i Tekhniki. Ser. Mechanics of fluid and gas,* 1978, vol. 11, pp. 155—304 (in Russian).

18. **Lushchik V. G., Pavel'ev A. A., Yakubenko A. E.** Transfer equation for turbulent heat flux. Calculation of heat transfer in a pipe, *J. Fluid Dyn*., 1988, vol. 23, no. 6, pp. 835—842.

19. **Kays W. M.** Turbulent Prandtl number — where are we? *Trans. ASME. J. Heat Transfer,* 1994, vol. 116, pp. 284—295.

20. **Kolmogorov A. N.** Uravnenija turbulentnogo dvizhenija neszhimaemoj zhidkosti (Equations of turbulent motion of an incompressible fluid), *Izv. AN SSSR. Ser. Phyzical,* 1942, vol. 6, no. 1—2, pp. 56—58 (in Russian).

21. **Lushchik V. G., Pavel'ev A. A., Yakubenko A. E.** Uravnenija perenosa dlja harakteristik turbulentnosti: modeli i rezul'taty raschetov (The transport equations for turbulence characteristics: models and calculation results), *Itogi Nauki i Tekhniki. Ser. Mechanics of fluid and gas,* 1988, vol. 22, pp. 3—61 (in Russian).

22. **Patel V. C., Rodi W., Scheuerer G.** Turbulence models for near-wall and low Reynolds number flows: A review, *AIAA Journal,* 1985, vol. 23, no. 9. pp. 1308—1319.

23. **Lushchik V. G., Pavel'ev A. A., Yakubenko A. E.** Turbulent flows. Models and numerical investigations. A review, *J. Fluid Dyn*., 1994, vol. 29, no. 4, pp. 440—457.

24. **Lushchik V. G., Yakubenko A. E.** Comparative analysis of turbulence models for calculating a near-wall boundary layer, *J. Fluid Dyn*., 1998, vol. 33, no. 1, pp. 36—47.

25. **Avduevsky V. S.** Metod rascheta prostranstvennogo turbulentnogo pogranichnogo sloja v szhimaemom gaze (A method for calculating the spatial turbulent boundary layer in a compressible gas), *Izv. AN SSSR. OTN. Mechanics and Mechanical Engineering,* 1962, no. 4, pp. 3—13 (in Russian).

26. **Cebeci T., Smith A. M. O.** *Analysis of turbulent boundary layers,* N. Y., Acad. Press, 1974, 404 p.

27. **Chien K. Y.** Predictions of channel and boundary layer flows with a low-Reynolds-number model, *AIAA Journal*, 1982, vol. 20. no. 1. pp. 33—38.

28. **Launder B. E., Sharma B. I.** Application of the energy-dissipation model of turbulence to the calculation of flow near a spinning disc, *Lett. Heat and Mass Transfer,* 1974, vol. 1, no. 2, pp. 131—138.

29. **Vilcox D. C.** Reassessment of the scale determining equation for advanced turbulence models, *AIAA Journal*, 1988, vol. 26, no. 11, pp. 1299—1310.

30. **Vilcox D. C., Rubesin W. M.** *Progress in turbulence modeling for complex flow fields including effects of compressibility*, NASA Tech. Paper, 1980, no. 1517, 70 p.

31. **Schlichting G.** *Teorija pogranichnogo sloja* (The theory of the boundary layer), Moscow, Nauka, 1974, 712 p. (in Russian).

32. **Pavel'ev A. A.** O perehode k turbulentnosti v strujah (On the transition to turbulence in jets), *Turbulent flows,* Moscow, Nauka, 1974. pp. 185—193 (in Russian).

33. **Polyakov N. F.** Laminarnyj pogranichnyj sloj v uslovijah "estestvennogo" perehoda k turbulentnomu techeniju (Laminar boundary layer under conditions of a "natural" transition to a turbulent flow) *Development of perturbations in the boundary layer*, Novosibirsk, Nauka, 1979. pp. 23—67 (in Russian).

34. **Abu-Ghannam B. J., Shaw R.** Natural transition of boundary layers-the effects of turbulence, pressure gradient, and flow history, *J. Mech. Eng. Science,* 1980, vol. 22, no. 5. pp. 213—228.

35. **Kolyada V. V., Pavel'ev A. A.** Interchange of the mechanisms of transition to turbulence in the boundary layer, *J. Fluid Dyn.,* 1986, vol. 21, no. 3, pp. 492—494.

36. **Lushchik V. G., Pavel'ev A. A., Reshmin A. I., Yakubenko A. E.** Influence of boundary conditions on the transition to turbulence in the boundary layer on a plate with a large level of external perturbations, *J. Fluid Dyn*., 1999, no. 6, pp. 111—119.

37. **Lushchik V. G., Pavel'ev A. A., Yakubenko A. E.** Transition to Turbulence in the Boundary Layer on a Plate in the Presence of a Negative Free-Stream Pressure Gradient*, J. Fluid Dyn*., 2004, vol. 39, no. 2, pp. 250—259.

38. **Lushchik V. G., Makarova M. S., Yakubenko A. E.** Vlijanie urovnja turbulentnosti nabegajushhego potoka na harakteristiki techenija pogranichnogo sloja na plastine (Influence of the level of turbulence of the incoming flow on the characteristics of boundary layer flow on a plate), *Bulletin of the University of Nizhny Novgorod*, 2011, no. 4, part 3, pp. 945—947 (in Russian).

39. **Lioznov G. L., Lushchik V. G., Makarova M. S., Yakubenko A. E.** Freestream turbulence effect on flow and heat transfer in the flat-plate boundary layer, *J. Fluid Dyn*., 2012. vol. 47, no. 5, pp. 590—592.

40. **Lushchik V. G., Yakubenko A. E.** Supersonic boundary layer on a plate. Comparison of calculation and experiment, *J. Fluid Dyn.,* 1998, vol. 33, no. 6, pp. 864—875.

41. **Leontiev A.I., Lushchik V. G., Makarova M. S.** The temperature recovery factor in a boundary layer on a permeable plate*, High Temp.,* 2017, vol. 55, no. 2, pp. 246—252.

Указатель статей, опубликованных в журнале "Программная инженерия" в 2017 г.

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *Н. В. Яшина*

Сдано в набор 12.10.2017 г. Подписано в печать 21.11.2017 г. Формат 60×88 1/8. Заказ PI1217
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**