ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

Том 8
№ 2
2017

Учредитель: Издательство "НОВЫЕ ТЕХНОЛОГИИ"

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 2
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 004.4 DOI: 10.17587/prin.8.51-57

В. В. Липаев, д-р техн. наук, проф, гл. науч. сотр., Институт системного программирования РАН, г. Москва

К разработке моделей динамической внешней среды для испытаний сложных программных продуктов

Некоторые виды программных продуктов предназначены для управления динамическими (изменяющими состояние во времени) объектами, которые могут располагаться в среде, внешней по отношению к основным программам управления. На различных стадиях разработки таких программных комплексов, в том числе при их проектировании, для тестовых испытаний и верификации создают испытательные стенды и проблемно-ориентированные комплексы программ, моделирующие функционирование объектов в динамической внешней среде.

На основе опыта автора по созданию моделей динамической внешней среды для испытаний критически важных программных управляющих комплексов представлены: подходы к разработке требований к моделям такой среды; возможное содержание компонентов, генерирующих динамические модели среды; особенности испытаний заказных программных продуктов во взаимодействии с моделями внешней среды.

Ключевые слова: *заказной программный продукт, внешняя среда управляющих программных продуктов, динамические объекты внешней среды, модели внешней среды, испытания объектов внешней среды*

Введение

При разработке программных комплексов управления сложноорганизованными целевыми системами, а также изменяющимися во времени (динамическими) ***объектами внешней среды***, с которыми подобные системы взаимодействуют, целесообразно выделять требования к комплексу управляющих программ ***и требования к модели, имитирующей среду, внешнюю по отношению к подлежащему созданию программному продукту*** [1, 2]. Такие сложные, наукоемкие программные продукты, как правило, создают по заказу промышленных предприятий и организаций. Заинтересованное участие заказчиков и разработчиков вновь создаваемого программного комплекса в формировании таких требований необходимо для обеспечения должной полноты и точности набора требований, позволяющих адекватно ***описать характеристики разрабатываемого оборудования и процессов его взаимодействия с объектами внешней среды (ВС).*** Как следствие, при определенном исходном состоянии подлежащего разработке программного продукта и с учетом множества входных данных разработчики должны ***предсказать состав и содержание выходных, близких к прогнозируемым данных управляющего программного продукта и всей динамически изменяющейся целевой системы, взаимодействующей с моделями ВС***.

Имитация конкретных динамических объектов с реальными характеристиками, адекватными характеристикам объектов ВС, является основной частью моделирующих имитационных стендов (рисунок). Качество моделей зависит от объема и глубины знаний разработчиками алгоритмов функционирования внешних объектов, характеристик их компонентов и обобщенных параметров динамических режимов функционирования целевой системы в целом. Такими динамическими объектами во внешней среде могут быть: динамические объекты воздушного, наземного и водного транспорта; операторы — пользователи объектов ВС; результаты анализа состояния аппаратуры объектов ВС; тренажеры пользователей динамических объектов ВС.

Все эти объекты находятся под управлением программных продуктов с использованием обрат-

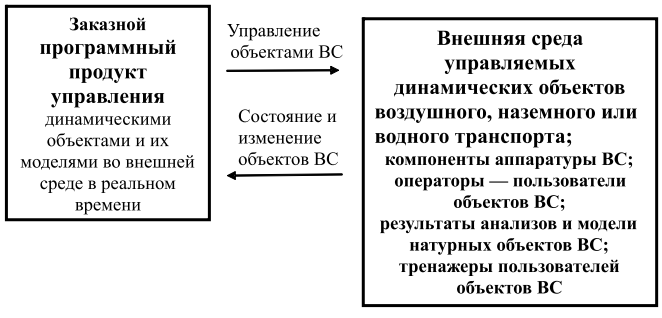


Схема функционирования в динамике программных продуктов с использованием объектов ВС

ной связи, позволяющей получить информацию о состоянии и изменениях объектов внешней среды.

Отдельного внимания в процессе разработки управляющих программных продуктов заслуживают вопросы оценки их качества [3]. С этих позиций необходимо, чтобы ***имитаторы динамической внешней среды*** создавали условия эксплуатации таких продуктов, близкие к реальным. Такая адекватность зависит от степени учета основных и второстепенных факторов, характеризующих функционирование реальных объектов и источников информации. Точность требований к моделям ВС и к их программной реализации, прежде всего, определяется алгоритмами, на которых они базируются, а также полнотой учета в них всех особенностей функционирования моделируемых объектов ВС. Кроме того, на корректность имитации ВС влияет уровень дефектов и ошибок, допущенных в реализующих ее программах. Каждый не учитываемый в имитаторе элемент или фактор моделируемой системы необходимо оценивать путем сопоставления характеристик частных имитируемых данных с результатами аналитических исследований или с данными, полученными на реально работающих системах. Следует также определять возможное влияние такого фактора на формулируемые к модели ВС требования и на точность описания модели и генерируемых ею динамических свойств. При этом нужно принимать во внимание другие составляющие, которые влияют на достоверность результатов имитационного моделирования.

Разумное ***сочетание части реальных (физических) объектов внешней среды*** (когда это возможно) ***и тех, которые моделируют на компьютерах,*** может обеспечивать создание высокоэффективных моделирующих испытательных стендов (МИС). Такие комплексные (гибридные) модели крайне необходимы для динамических испытаний качества программных продуктов, управляющих динамическими системами реального времени. Важнейшая задача формирования требований к модели ВС состоит в достижении основной цели всей системы управления, в которой применяется заказной программный продукт. Моделирующие испытательные стенды позволяют осуществлять автоматизированную генерацию тестов с помощью имитаторов на компьютерах и аналогов реальной аппаратуры. Они представляют возможности дополнять создаваемый программный продукт и систему в целом реальными данными от операторов пользователей, контролирующих и корректирующих процесс функционирования систем обработки информации.

Кроме перечисленных выше возможностей, модель ВС должна предоставлять возможность разработчику и заказчику определять характеристики качества при принятии решения о ***готовности программного продукта для его применения по назначению***. Автоматизация испытаний с использованием модели ВС способствует более быстрому, качественному и эффективному тестированию создаваемого программного комплекса. Это обстоятельство, в свою очередь, ведет к сокращению объема работ и к улучшению процесса производства программного продукта. В ходе его испытаний необходимо проверять условия того, что комплекс программ работает в соответствии с предъявляемыми к нему требованиями и ***удовлетворяет заданным критериям качества всей целевой системы.***

Разработка требований к модели динамической внешней среды

Процесс разработки модели ВС для управляющего программного комплекса реального времени целесообразно начинать с определения требований к такой модели. Далее следует верификация требований к отдельным компонентам и к модели в целом. Эти требования к модели ВС должны соответствовать требованиям к подлежащему созданию программному продукту. Для заказного комплекса программ важно разработать корректные ***требования к такому продукту и сценарии его использования, а также модели внешней среды для управляющего программного комплекса***. Для этого при анализе требований к модели ВС необходимо [4]:

- изучить требования к целевой системе и к управляющему программному продукту как его составляющей, назначение и сценарии использования создаваемого программного продукта и ВС для того чтобы более точно формализовать эти требования;
- определить наиболее ***значимые для программного продукта функции, а также функции, реализация которых связана с повышенным риском неблагоприятных событий;***
- определить требования к тестовым испытаниям для проверки корректности версии управляющего программного продукта и компонентов ВС;
- преобразовать архитектуру, требования к функциям и характеристикам программного продукта в сценарии и результаты его использования;
- по результатам анализа требований к системе и к результатам функционирования модели ВС сформировать сопроводительную документацию на комплекс программ.

Требования к модели ВС должны содержать подробный перечень тех сущностей (объектов, их атрибутов, свойств и т. п.), которые должны функционировать и подлежать испытаниям. Цели верификации модели на предмет соответствия составляющих ее сущностей предъявляемым к ним требованиям достигаются путем анализа потенциально возможных ситуаций, разработкой контрольных сценариев и процедур, а также возможных результатов последующего их выполнения. Такие сценарии, по результатам которых анализируют характеристики ***модели ВС, предназначены для поэтапной проверки внутренней непротиворечивости и полноты реализации совокупности требований к модели ВС.*** Однако следует иметь в виду, что результаты такого анализа не гарантируют полноту и корректность всех требований. Причина в том, что эти работы проводят частично

вручную и, как следствие, могут отсутствовать четкие эталоны для их проверки.

Для обеспечения высокого качества управляющего программного продукта одновременно с ***верификацией требований*** и с их модификацией следует разрабатывать и верифицировать ***сценарии***, отражающие методы и конкретные процедуры проверки выполнения этих требований. Специфицированные требования могут использоваться для проверки согласованности, внутренней непротиворечивости и степени полноты выполнения требований без исполнения программ. Для каждого требования к функциям комплекса программ, к его архитектуре, к отдельным компонентам должна быть разработана спецификация требований к тестам, обеспечивающая проверку их корректности, адекватности реалиям и возможности в последующем реализовать тестирование компонентов на соответствие такому требованию. Такая взаимная проверка корректировок функций компонентов модели ВС, отраженных в требованиях и в спецификациях тестов, обеспечивает повышение их качества, сокращение дефектов, ошибок, неоднозначностей и противоречий.

Наборы требований к модели ВС можно применять ***как эталоны и другие адекватные формы описания функций и других характеристик комплексов управляющих программ***. Для обеспечения высокого качества программных продуктов стратегию создания ***требований к модели*** ВС целесообразно строить***, основываясь на требованиях к функциям и характеристикам целевой системы***. Требования к модели могут пополняться на основе анализа логики функционирования и архитектуры комплекса программ путем построения структуры решений. Такой подход может применяться в зависимости от условий договора исполнителя работ с заказчиком или особых требований к безопасности программного комплекса. Например, покрытие тестами полной структуры программных решений для некоторых программных систем может быть необходимо в аэрокосмической отрасли [2—4], в оборонном комплексе и в других критически важных динамических системах с повышенными требованиями к их безопасности.

Компоненты генераторов динамических моделей внешней среды

Для проведения динамических испытаний комплексов программ реального времени следует составлять планы автономных сценариев функционирования модели ВС и готовить ***обобщенные исходные данные.*** Программы моделирования должны обеспечивать реализацию этих сценариев на основе конкретных значений данных (параметров), характеризующих динамические режимы функционирования каждого имитатора или реального (физического) ***объекта ВС.*** Эти данные можно вводить и преобразовывать на моделирующем компьютере как в режиме offline, на этапе подготовки к функционированию МИС, так и в ходе испытаний программы управления объектами ВС в реальном времени (online).

Аналоги компонентов функционирования системы и программные модели ВС используют для генерации динамических тестов. Кроме того, они позволяют проверять и аттестовать некоторые программные и аппаратные имитаторы объектов ВС, которые впоследствии используют при испытаниях.

Данные с рабочих мест операторов — пользователей управляющей системы должны имитировать реальные характеристики различного рода динамических воздействий на программный продукт. При этом необходимо учитывать особенности и квалификацию администратора (оператора), которому предстоит использовать программы как в реальной системе управления, так и в моделях ВС. На эту часть МИС кроме первичных исходных данных от операторов могут вводиться данные, полученные в результате обработки некоторых динамических тестов системы, которая включает и управляющие программы, и модели ВС.

Данные натурных экспериментов с объектами внешней среды можно готовить и фиксировать заранее, вне сеансов динамических испытаний программного продукта, например, при отладке аппаратной части моделируемой системы обработки информации. Эти данные могут отражать характеристики и динамику функционирования моделируемых объектов, которые трудно или опасно подключать для непосредственного взаимодействия с недостаточно проверенными управляющими программами. Кроме того, такие данные могут использоваться для проверки адекватности разрабатываемых имитаторов некоторых объектов ВС реально существующим объектам.

При динамическом моделировании в ряде случаев необходимо иметь ***эталонные (соответствующие реальным) характеристики***, которые поступают на подлежащий испытаниям управляющий программный продукт или систему в целом, включая ВС. При работе с динамическими объектами зачастую приходится создавать специальные измерительные комплексы для получения таких характеристик в процессе наблюдения, которые соответствуют реальным объектам. К числу таких характеристик относятся, например, динамические характеристики и координаты движения самолетов при испытаниях систем управления воздушным движением. Подобные измерения, как правило, проводят в режиме автономного функционирования внешних объектов, когда подлежащие определению исходные данные или динамические характеристики регистрируются при взаимодействии таких объектов с управляющим комплексом программ в реальном времени.

Повторяемость сеансов испытаний при автоматической имитации динамических объектов ВС должна обеспечиваться фиксацией всех исходных данных и применением программного формирования псевдослучайных чисел (если они используются). При надежной работе аналогов реальных

объектов и компьютера, моделирующего ВС, можно добиться высокой степени повторяемости подлежащих измерению характеристик в ходе длительных экспериментов и сценариев моделирования. Труднее обеспечивать такую повторяемость в ходе реализации сценариев динамических испытаний, в которых активно участвует оператор-администратор. В этом случае необходимо регистрировать и сохранять действия оператора в зависимости от времени, а затем повторять их в соответствии с принятым сценарием. При необходимости временная диаграмма может соблюдаться с точностью до 0,5...1 с. Однако ошибки в действиях оператора и, соответственно, во вводимых им параметрах могут отличаться в каждом сценарии моделирования.

После приемки заказчиком или пользователями в процессе функционирования всей системы и применения управляющего программного продукта должна обеспечиваться оценка его текущего качества, учитывающего взаимодействие с моделями внешней среды. Для этого в ***составе программного продукта необходимо иметь средства, обеспечивающие:***

- генерацию динамических сценариев или хранение состояний моделей ВС для контроля работоспособности, сохранности и целостности управляющего программного продукта;
- оперативный контроль и обнаружение дефектов исполнения моделирующих программ и механизмов обработки данных при использовании программного продукта по прямому назначению;
- реализацию процедур предварительного анализа выявленных дефектов моделей ВС и оперативное восстановление вычислительного процесса, программ и данных (рестарт) после обнаружения аномалий моделей динамического функционирования программного продукта или всей системы;
- мониторинг, накопление и хранение данных о выявленных дефектах, сбоях и отказах в процессе функционирования управляющего комплекса программ и в ходе обработки данных с использованием моделей ВС.

Средства генерации динамических тестов и имитации ВС совместно с поставляемым заказчику программным продуктом могут использоваться для оперативной подготовки исходных данных при проверке различных режимов функционирования программного продукта, а также при диагностике проявившихся дефектов. Минимальный состав средств генерации моделей ВС необходимо передавать пользователям для контроля использования рабочих версий в реальном времени. Он может входить в комплект поставки каждой пользовательской версии управляющего программного продукта.

Важной функцией испытательных стендов ВС является их использование в качестве ***тренажеров для операторов-пользователей***. Качество функционирования заказных комплексов программ может существенно зависеть от характеристик конкретных операторов, участвующих в эксплуатации и обработке информации. Как следствие, возникает потребность в определении этих характеристик. Более того, необходимо иметь возможность улучшать их до уровня, который обеспечивает выполнение ***заданных требований к управляющему программному продукту***. По этой причине в программу испытаний в обязательном порядке должны входить процедуры динамической тренировки операторов, измерения реальных характеристик функционирования программного продукта и времени реакции оператора на внешние воздействия. Важным направлением является использование МИС для обучения и регулярной подготовки операторов-пользователей в процессе тиражирования и эксплуатации программного продукта и его моделей ВС.

Испытания программных продуктов с динамическими моделями внешней среды

До проведения тестовых испытаний разрабатываемого программного продукта заказчик и исполнитель должны подготовить, согласовать и ***утвердить план его испытаний с учетом взаимодействия с ВС.*** С этой целью следует описать требования к таким испытаниям. Они должны ***соответствовать требованиям к системе, которая включает программный комплекс и объекты ВС в целом, а также к тестовым сценариям.*** Испытания целесообразно делить на следующие три этапа:

- автономные испытания управляющего заказного программного продукта и его воздействий на объекты ВС;
- автономные испытания динамических объектов и других компонентов моделей ВС;
- комплексные испытания управляющей системы с динамическими объектами и моделями ВС.

Первый из перечисленных этапов представляет собой традиционные испытания сложных программных продуктов, которые дополнены требованиями проверки наличия полноты и корректности (соответствия реально существующим) воздействий разрабатываемого программного комплекса на объекты ВС и механизмов обработки поступающей от этих объектов на комплекс информации. На этом этапе должен учитываться план автономных динамических испытаний объектов и других компонентов модели ВС. Следует также оценить ресурсозатраты на испытания управляющего программного продукта с динамической моделью ВС. При этом необходимо принимать во внимание следующие обстоятельства:

- ♦ испытания, в первую очередь, целесообразно проводить для проверки выполнения требований с наивысшим приоритетом, которые наиболее важны для заказчика и пользователей, либо могут привести к значительному ущербу для качества системы;
- ♦ сначала следует осуществлять испытания новых функциональных возможностей, направленных на исправление или совершенствование характеристик управляющего программного продукта;
- ♦ тестирование полезно начинать с функций, с которыми наиболее часто будет иметь дело конеч-

ный пользователь или другие субъекты, имеющие отношение к взаимодействию с объектами системы в целом.

Второй этап должен начинаться с выбора типов объектов и других компонентов для построения моделей ВС. Их целесообразно упорядочивать по приоритетам, по сложности реализации и важности. Эти факторы принимают во внимание при определении последовательности разработки, в ходе подготовки программ проведения испытаний и опытной эксплуатации целевой системы. К числу таких объектов и компонентов относят:

- программные модули;
- аппаратные средства;
- воздействия от операторов-пользователей;
- результаты взаимодействия ВС с управляющим комплексом, полученные в ходе предварительных натурных экспериментов;
- тренажеры.

При этом наиболее гибкие программные компоненты комплекса могут приспосабливать свой интерфейс к объектам и компонентам ВС, которые изменять намного сложнее.

Третий этап — испытания всей системы, которая включает заказной программный продукт, с комплексом имитационных динамических моделей ВС. Результаты испытаний на этом этапе призваны продемонстрировать заказчику выполнение всех согласованных с ним требований к проверяемой динамической системе. Планирование испытаний включает как определение требований к тестам, так и разработку процессов управления этими требованиями.

Когда план испытаний комплекса управляющих программ разработан и полностью описывает процессы тестирования, он становится руководящим документом (базовым инструментарием) для формирования ***программы динамических испытаний системы в целом***. Эта программа может уточняться посредством корректировок целей, задач и стратегий тестирования отдельных моделей объектов и других компонентов ВС в реальном времени. Эти объекты и компоненты могут быть не только программными, но и представлять собой аппаратные средства или воздействия от операторов.

Для сокращения неопределенностей и прямых ошибок при оценивании качества программного продукта необходимо до начала испытаний определить основные параметры ВС, при которых должен функционировать комплекс программ с требуемыми характеристиками. Для этого заказчик и разработчик совместно должны структурировать, описать и согласовать модель внешней среды и ее параметры в среднем, типовом режиме применения программного продукта, а также в наиболее вероятных и критических режимах, в которых должны обеспечиваться требуемые характеристики качества функционирования всей системы.

Программа испытаний является планом проведения серии экспериментов и должна разрабатываться с позиции допустимой минимизации объема тестирования в процессе проведения испытаний. Программа испытаний, методики их проведения и оценки результатов разрабатывают совместно заказчик и исполнитель. Программа содержит уточнения и детализацию требований для данного программного продукта. Ее задача — гарантировать корректную проверку всех заданных характеристик качества функционирования объектов ВС.

До начала испытаний программного продукта подлежат проверке и паспортизации средства, которые обеспечивают: получение эталонных данных; имитацию тестов от внешних объектов; фиксацию и обработку результатов тестирования ВС. При подобных испытаниях важную роль играет оценка и обеспечение близких значений ожидаемых по методике и статистической достоверности результатов испытаний. Завершение испытаний программного продукта с моделями ВС должны сопровождаться предъявлением заказчику на утверждение ***комплекта документов, содержащих результаты комплексных испытаний*** создаваемых программных продуктов и системы в целом. При разработке сложных комплексов программ для их верификации и тестирования, как правило, необходимы ***значительные ресурсы*** в течение всего их жизненного цикла. Наиболее критичным ресурсом при этом является ***допустимое время*** поэтапного выполнения этих процедур. В результате совокупность ***требований к заказному программному продукту*** может применяться ***как эталон и дополнительная адекватная форма описания содержания комплекса управляющих программ и системы, для которой этот комплекс создается.*** Такие ***параллельные взаимные проверки*** требований, текстов управляющих программ и моделей ВС способствуют выявлению и исключению вторичных дефектов и ошибок комплексов программ. В дальнейшем эти требования должны использоваться для тестирования выполнения требований к программным компонентам создаваемого комплекса и модели ВС. Параллельная и независимая разработка управляющих программ и моделей ВС, а также их реализация позволяют распараллеливать эти независимые во времени сферы деятельности разработчиков. Это обстоятельство ведет к сокращению сроков создания компонентов и комплексов программ.

Особенности описаний и реализации программ (а также мышления их разработчиков-программистов), которые проявляются при использовании требований, функций, характеристик структуры и исполнения управляющих программ, существенно ***отличаются от представлений и методов описаний таких же функций комплекса программ ВС.*** Создатели сценариев тестирования и эталонов акцентируют свою деятельность на конкретных процедурах проверки функционирования, на возможных результатах и взаимодействии объектов ВС. ***Такой подход позволяет выявлять вторичные дефекты, повышать качество путем сопоставления двух методов и результатов*** **описания** ***одних и тех же*** функций и

характеристик системы. Результат при этом достигается за счет того, что мала вероятность одинаковых ошибок в сценариях и реализациях моделей ВС и в описаниях требований к функциям и к характеристикам управляющих программ.

Важным аспектом организации испытаний заказных программных продуктов и моделей динамических объектов ВС является принятие решения о том, в каком объеме они достаточны и ***когда необходимо завершить процесс тестирования*** [5, 6]. Принятие решения об окончании тестирования включает рассмотрение вопросов стоимости и рисков неблагоприятных ситуаций, связанных с возможными нарушениями надежности функционирования тестируемой программной системы и динамических моделей ВС. В то же время необходимо отметить, что ресурсозатраты на тестирование также являются одним из ограничений, на основе которых принимается решение о продолжении работ или об их прекращении.

Заключение

Затраты на создание моделей имитации динамических систем ВС для оценивания качества динамических программных продуктов могут быть одной из существенных составляющих при создании крупных заказных программных комплексов реального времени [4]. В ряде случаев они соизмеримы с затратами на создание основных функций управляющих комплексов программ. Это обстоятельство определяется принципиальным соответствием ***сложности необходимых наборов тестов*** и ***сложности функций***, реализуемых испытываемым комплексом программ.

Список литературы

1. **Шеннон Р.** Имитационное моделирование систем — искусство и наука. Пер. с англ. М.: Мир, 1978. 424 с.
2. **Мостовой Я. А.** Имитационная математическая модель внешней среды в жизненном цикле бортового программного обеспечения управления космической платформой// Космическая оптика. 2012. Том 36, № 3. URL: http://journals.ssau.ru/index.php/computeroptics/article/view/1493/1497
3. **Липаев В. В.** Методы обеспечение качества крупномасштабных программных средств. М.: РФФИ. СИНТЕГ, 2003. 520 с.
4. **Автоматизированные** системы управления воздушным движением. Учебное пособие/ Под науч. ред. Ю. Г. Шатракова. СПБ.Политехнология, 2014. 450 с.
5. **Дастин Э., Рэшка Д., Пол Д.** Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация. Пер. с англ. М.: ЛОРИ, 2003. 567 с.
6. **Блэк Р.** Ключевые процессы тестирования. Пер. с англ. М.: ЛОРИ, 2006. 257 с.

Послесловие

Первая версия статьи по этой тематике была написана В. В. Липаевым еще в апреле 2015 г. Принимая во внимание актуальность и важное значение этой темы, учитывая опыт В. В. Липаева по руководству коллективами разработчиков сложных программных продуктов для управления динамическими объектами, я предложил ему написать эту статью. Несмотря на относительно лаконичный стиль изложения материала, в статье много положений, которые и сегодня представляют интерес на направлении разработки сложных, критически важных программных комплексов для управления объектами, динамично изменяющими свое состояние во внешней среде.

В результате обмена мнениями текст статьи дважды редактировался, готовилась ее третья версия. Однако 21 сентября 2015 г. один из патриархов инженерии программ в России ушел из жизни, и это обстоятельство надолго отодвинуло публикацию статьи.

Настоящая работа посвящается светлой памяти замечательного человека, ученого и педагога, разработчика целого ряда стратегически важных для страны комплексов программ, классика отечественной программной инженерии Владимира Васильевича Липаева.

Главный редактор
журнала "Программная инженерия",
д-р физ.-мат. наук, проф.
В. А. Васенин

On Development of Models of a Dynamic External Environment for Testing the Complex Software Products

V. V. Lipaev, Institute for System Programming of the Russian Academy of Sciences, 109004, Moscow, Russian Federation

Received on April 20, 2015
Accepted on November 20, 2016

Some kinds of software projects are aimed at managing the dynamic objects (which change their state with time). Such objects can be placed in an environment that is external to the main programs of the large target system, managing of which is their task. Such software complexes can be divided into two actively interacting parts: part that manages the main processes of the target systems and part that implements control of dynamic objects in an external environment. Examples of objects of external environment are the following models: flight of spaceships,

airport flight control points, air defense system objects, aviation on-board control system. One of the tasks when developing such software complexes is creating and testing the above noted models of dynamically managing the objects in an external environment.

Due to complexity of the full-scale modeling of such objects' dynamics in real conditions, software emulators of external environments are used during the stages of modeling and production. For this task, the modeling test benches are created. They include problem-oriented software complexes which are modeling the objects in a dynamic external environment. It is worth noting that such complexes may be significantly larger than the corresponding managing software products being tested. The first Soviet Union software models of the external environment, imitating flight of various aircraft types in an external environment, were created already in 1960s for testing the air defense system of the country.

As examples of external environment models to be tested for qualifying for the requirements for the functions and characteristics of the managing programs complexes, one can take control systems of spaceships, and also aircraft flight and air traffic control systems. For complex debugging and testing of the program complexes of such control systems, imitation of all changes of information coming from the external environment should be implemented if needed. For modeling test benches of the air traffic control centers, sources of information were radars and personnel of the aircrafts. As a result, a need for dynamically imitating the interaction of a row of diverse objects, taking into account their influences on the control object, has emerged.

Attempts to developing the requirements for models of a dynamic external environment, components generating dynamic models of an environment, and specialities of testing the customized software projects in interaction with external environment models are presented in the article, based on the author's experience in developing such models.

Keywords: *customized software project, external environment of managing software products, dynamic objects of an external environment, testing the objects of an external environment*

For citation:

Lipaev V. V. On Development of Models of a Dynamic External Environment for Testing the Complex Software Products, *Programmnaya Ingeneria,* 2017, vol. 8, no. 2, pp. 51—57.

DOI: 10.17587/prin.8.51-57

References

1. **Shannon R.** *Imitatsionnoe modelirovanie sistem — iskusstvo i nauka* (Systems simulation. The art and science), Moscow, MIR, 1978, 424 p. (in Russian).

2. **Mostovoj Ja. A.** Imitacionnaja matematicheskaja model' vneshnej sredy v zhiznennom cikle bortovogo programmnogo obespechenija upravlenija kosmicheskoj platformoj (Simulation mathematical model of the external ambience in life cycle of on-board software of management cosmic platform), *Kosmicheskaja optika*, 2012, vol. 36, no. 3, available at: http://journals.ssau.ru/index.php/computeroptics/article/view/1493/1497 (in Russian).

3. **Lipaev V. V.** *Metody obespecheniya kachestva krupnomasshtabnykh programmnykh sredstv* (Methods of ensuring the quality of large-scale software systems), Moscow, SINTEG, 2003, 520 p. (in Russian).

4. ***Avtomatizirovannye*** *sistemy upravleniya vozdushnym dvizheniem*. Uch. posobie (Automated systems of air traffic control) / Eds by Yu. G. Shatrakova, Sankt-Petersburg, Politekhnologiya, 2014, 450 p. (in Russian).

5. **Dusting E., Reshka D., Paul D.** *Avtomatizirovannoe testirovanie programmnogo obespecheniya. Vnedrenie, upravlenie I ekspluatatsiya* (Automated Software Testing. introduction, Management and Performance), Moscow, LORI, 2003, 567 p. (in Russian).

6. **Black R.** *Klyuchevye protsessy testirovaniya* (Key testing processes), Moscow, LORI, 2006, 544 p. (in Russian).

ИНФОРМАЦИЯ

УДК 004.4'2 + 004.51 DOI: 10.17587/prin.8.58-65

В. М. Баканов, д-р техн. наук, проф., e-mail: vbakanov@hse.ru, Национальный исследовательский университет "Высшая школа экономики" (НИУ ВШЭ), Москва

Программный комплекс для разработки методов построения оптимального каркаса параллельных программ

Проанализированы подходы к разработке рациональных (стремящихся к оптимальным) методов построения планов выполнения прикладных вычислительных задач на реальных параллельных вычислительных системах (каркасов выполнения), а также инструментальная программная среда для реализации таких методов. Предложены критерии и параметры оптимизации методов планирования. Представлены результаты применения некоторых разработанных стратегий построения рациональных планов выполнения параллельных программ.

Ключевые слова: *графовые представления алгоритма, анализ информационной структуры программы, ярусно-параллельная форма информационного графа, рациональные параметры выполнения параллельных программ, стратегия построения рационального плана выполнения параллельной программы*

Введение

В настоящее время и в ближайшем будущем одним из реальных путей повышения производительности вычислительных систем является подход на основе распараллеливания процессов обработки данных (далее — параллелизация). Путь повышения тактовой частоты процессов ограничивается фундаментальными законами и технологией их производства.

Вместе с тем параллелизация (возможность одновременного, параллельного выполнения различных частей одной программы на независимо работающих вычислителях) требует от специалистов по алгоритмике и программистов дополнительных усилий по выявлению в алгоритме участков, которые могут быть исполнены параллельно (один из вариантов декомпозиции). Основное требование к таким блокам (*зернам* или *гранулам*) параллелизма состоит в их независимости (ортогональности) по данным.

Обзор исследований в данной области

Анализ алгоритмов на наличие потенциала параллелизма, часто именуемый *исследованием тонкой информационной структуры алгоритма*, непрост. Поэтому говорят о скрытом (не фиксируемом при *поверхностном рассмотрении*) параллелизме. Конструктивным инструментарием при таком анализе является представление программ графовыми моделями. В нашей стране разработкой методов анализа структуры алгоритмов на основе графовых моделей занимались В. В. Воеводин и Вл. В. Воеводин [1].

Ситуация с использованием параллелизма усложняется необходимостью рационального применения ресурсов конкретной многопроцессорной вычислительной системы (МВС). Эти ресурсы (число и возможности параллельно работающих вычислителей) в каждом отдельном случае ограничены, поэтому задача планирования использования ресурсов является важной. Несмотря на наличие архитектур параллельных вычислительных систем, не нуждающихся в априорном планировании (например, потоковый вычислитель аппаратно определяет очередность выполнения операторов на этапе выполнения программ), именно априорному планированию при реализации программ на МВС в настоящее время отдается приоритет.

Априорный (до этапа выполнения) анализ структуры алгоритмов не является самодостаточным, он лишь основа для оценки эффективности выполнения программы на конкретной МВС. Учет конкретики МВС при этом во многих случаях невозможен аналитически и требует моделирования и оптимизации процесса. С этих позиций направление разработки инструментальных программных систем для решения задач эффективного использования МВС актуально.

Как правило, для представления и анализа программ используют (наряду с иными формами) информационные графы (например, проект AlgoWiki, [2]). При таком подходе для максимального повышения быстродействия оптимизации подвергают критические участки программы, не имеющие ветвлений. В этом случае информационный граф зачастую достаточен для описания алгоритма. Целью AlgoWiki является "дать исчерпывающее описание алгоритма, которое поможет оценить его потенциал применительно к конкретной параллельной вычислительной

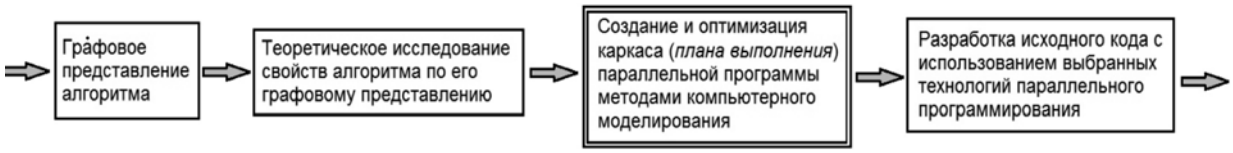


Рис. 1. Фрагмент цикла разработки эффективной параллельной программы

платформе" [2]. Следующим (перед процедурой собственно кодирования) и естественным этапом в цикле разработки параллельного приложения должно являться моделирование выполнения программы на конкретной МВС. При этом формируется оптимальный каркас (фактически план выполнения) параллельной программы (рис. 1).

При представлении программы информационным графом алгоритма (ИГА, вычислительная модель "операторы — операнды") вершины графа соответствуют преобразователям информации (операторам), а дуги — информационным связям операторов (операндам). В данном случае термин *операторы* достаточно условен, ибо под *операторами* понимают отдельные последовательности вычислений (блоки, подзадачи) любого размера в зависимости от соглашений, принятых при декомпозиции исходной программы. К свойствам ИГА относят: ацикличность, однонаправленность, детерминированность. Дуги ИГА нагружены значениями объема пересылаемых данных, вершины — значениями вычислительной сложности операторов, выраженными в машинных командах или тактах процессора. Эти данные не относятся к каркасу и конкретизируются на следующем этапе разработки параллельной программы (выбор технологии параллельного программирования и собственно кодирование) для оценки времени ее выполнения. Время пересылки данных определяется технологией обмена с учетом латентности коммуникационной среды, а время выполнения операторов — параметрами процессора.

Удобным и полезным методом выявления параллелизма является представление ИГА в ярусно-параллельной форме (ЯПФ). При таком представлении операторы, обладающие способностью выполняться независимо друг от друга (фактически параллельно), располагаются на одном уровне (*ярусе*). Формально каждый ярус ЯПФ является одним из сечений ИГА, удовлетворяющих условию независимости по данным, находящимся на этом ярусе операторов. Трудоемкость построения ЯПФ квадратичная от числа операторов.

Глубина ЯПФ, т. е. число ярусов, определяет общее время выполнения алгоритма, ширина ЯПФ — максимальное число задействованных отдельных (параллельно работающих) вычислителей, например, отдельных ядер процессора, данной МВС. В случае начала построения ЯПФ с исходных данных получаем "верхний" вариант ЯПФ (операторы располагаются на наиболее близких к начальным ярусах). При построении начиная с выходных данных получаем "нижний" вариант. Два таких представления ограничивают весь диапазон возможного расположения операторов на ярусах ЯПФ. Обычно считается, что в рамках каждого яруса операторы выполняются синхронно, а поле параллельных вычислителей — гомогенно. На рис. 2 показана ЯПФ графа простой процедуры вычисления вещественных корней полного квадратного уравнения $ax^2 + bx + c = 0$ (ЯПФ в *каноническом* виде, вследствие простоты алгоритма соответствие вершин графа вычислительным операциям не приводится).

Формально каноническая ЯПФ уже представляет собой некоторый каркас (план выполнения) отдельных частей программы — номера ярусов определяют последовательность выполнения параллельных групп операторов на данном ярусе. Однако подобное расписание реализуемо лишь на гипотетической МВС с бесконечно большим числом параллельных вычислителей (согласно *концепции неограниченного*

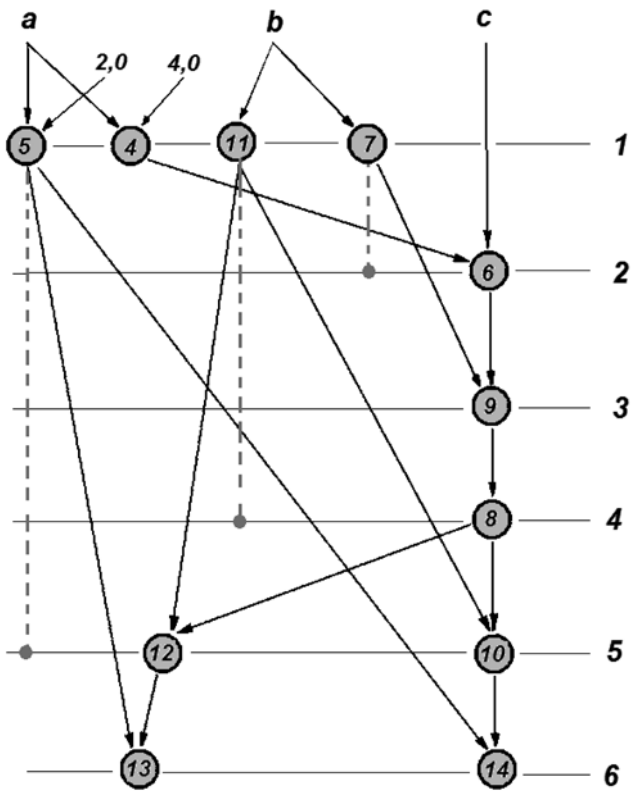


Рис. 2. Ярусно-параллельная форма процедуры вычисления вещественных корней полного квадратного уравнения (в "верхнем" варианте, крайние справа цифры — номера ярусов ЯПФ; штриховыми линиями показан диапазон возможного расположения операторов по ярусам)

параллелизма). Реальные ЯПФ информационных графов обладают большой неравномерностью в распределении числа операторов по ярусам (показательный пример — применяемый для ассоциативных операций *граф сдваивания*). Использование ресурсов МВС в таком случае очень нерационально — с точки зрения выполнения каждой программы максимум эффективности использования ресурсов обычно достигается при равномерном или близком к равномерному распределении операторов по ярусам. Вследствие стремления разработчиков к быстрейшему выполнению вычислений обычно останавливаются на построении статичного плана исполнения параллельной программы (миграция подзадач в пределах конкретного задания между вычислителями резко снижает производительность).

Применяемые при исследовании методы

Практически в любом ЯПФ имеется вариативность в расположении вершин графа (операторов) по ярусам (перемещение операторов между ярусами ограничивается соблюдением информационных зависимостей); для ЯПФ графа на рис. 2 оператор *5* может быть перемещен "вниз" на любой ярус со второго по пятый, оператор *11* — на любой ярус со второго по четвертый, оператор 7 — на второй ярус. Эта особенность ЯПФ дает возможность оптимизации ЯПФ (в смысле, например, достижения наибольшей равномерности распределения операторов по ярусам). В представленном на рис. 2 случае возможна "разгрузка" первого яруса от операторов, например, *5* и *11*; в результате программа может быть выполнена на двух (вместо четырех) параллельно работающих вычислителях за то же время.

Данная задача близка к проблематике разбиения графов и, согласно работе [3], относится к классу *NP*-полных. Для ИГА большого размера трудоемкость оптимизации путем перебора запредельно велика (задача перестановок с учетом влияния изменения конфигурации ЯПФ после каждой), существующие алгоритмы разбиения графов являются эвристическими; с учетом этого корректнее говорить о нахождении не оптимального, а приближающегося к оптимальному *рационального* решения. Параметрами оптимизации служит все множество планов выполнения операторов параллельной программы (с учетом сохранения информационных связей). В качестве критерия обычно принимают максимум скорости выполнения и/или полноту использования существующего оборудования (многокритериальная оптимизация). Для практики необходимо иметь гибкий аппарат для решения задачи планирования с разной степенью оптимизации (вследствие *NP*-полноты задачи понимая эту процедуру как последовательное повышение качества эвристик в сторону улучшения критерия оптимизации) — от "быстрых" методов, основанных, например, на использовании ЯПФ, до более трудоемких (методы "ветвей и границ", эволюционные и др.) для получения лучших решений. С этой точки зрения ценностью является получение количественных значений текущих приближений в качестве отправной точки для проведения дальнейших шагов оптимизации.

К ограничениям МВС относят: число параллельно работающих вычислителей, объемы их памяти, функциональные возможности, параметры коммуникационной среды. Выполнение программы с учетом этих ограничений достигается эквивалентными (не изменяющими информационных зависимостей в графе) преобразованиями ЯПФ-разбиений ИГА.

Будем использовать следующие параметры ЯПФ информационного графа:

- H — глубина ЯПФ графа, определяется числом ярусов в нем;
- W_i — ширина i-го яруса (число операторов на нем);
- W — ширина ЯПФ графа, определяется максимумом операторов по всем ярусам, $W = \max(W_i)$;
- $\bar{W}$ — средняя ширина ЯПФ; $\bar{W} = \frac{1}{N}\sum_{i=1}^{i=N} W_i$;

N — число ярусов ЯПФ;

- k — степень (коэффициент) неравномерности распределения операторов по ярусам ЯПФ графа; $k = \max(W_i) / \min(W_i)$;
- σ — среднее квадратичное отклонение ширин ярусов; $\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{i=N}\left(W_i - \bar{W}\right)^2}$.

Сформулируем и перечислим далее типовые постановки оптимизационной задачи.

◇ Максимально полная загрузка неизвестного заранее (но не меньшего $\bar{W}$) числа параллельно работающих вычислителей имеющейся МВС при минимальном времени выполнения задачи: $k \to 1{,}0$ при $H = H_0 = \text{const}$, где H_0 — длина критического пути, а именно минимальная из всех возможных высот ЯПФ данного графа.

◇ Наиболее полное использование доступных ресурсов (вычислительных узлов) имеющейся МВС при допущении увеличения времени выполнения (отход от канонической ЯПФ): $\max(W_i) \leqslant W_0$ при $H \geqslant H_0$, здесь W_0 — число доступных параллельно работающих вычислителей в заданной МВС; однако желательно $H - H_0 \to \min$ (минимизация времени выполнения). Для наглядности полезно представить этот процесс как подготовку связки, состоящей из не более чем заданного числа W_0 заведомо независимых (ортогональных) по операндам команд для вычислительной системы в аппаратно-программной идеологеме микропроцессорной архитектуры с явным параллелизмом команд (EPIC, *Explicitly Parallel Instruction Computing*) для параллельных вычислителей архитектуры сверхдлинного командного слова (VLIW, *Very Long Instruction Word*).

Для реализации и оптимизации эвристических алгоритмов поиска рациональных (стремящихся к оптимальным) планов выполнения параллельных частей параллельных программ разработано

специализированное программное обеспечение инструментального уровня, в котором для описания стратегий решения задачи использован встроенный скриптовый язык программирования Lua [4]. Данный встроенный язык программирования выбран по причинам его компактности (при его применении возрастание размера исполняемого файла родительского приложения не превышает ≈100 Кбайт) и простоты обучаемости для его использования (в основном Lua следует синтаксису языка C), широким функциональным возможностям (поддерживается объектно-ориентированная модель программирования и др.), бесплатности при наличии исходных текстов. Перечисленные обстоятельства подтверждают правильность выбора Lua в качестве управляющего языка для создания представлений ИГА. В целом возможностей Lua вполне достаточно для реализации наиболее трудоемких методов оптимизации задачи планирования. Клиентская часть программного инструментария (предварительное информационное сообщение приведено в работе [5]) разработана на языке программирования C++ и является 32-битным GUI-приложением Windows (рис. 3).

Набор API-вызовов разработанной системы позволяет реализовывать любой из возможных критериев оптимизации, включая их комбинации, так как выбор критерия осуществляет собственно пользователь, на основе задач, им поставленных. Частично пересекающимися по функциональным возможностям с данным проектом являются проекты V-Ray и ПАРУС (оба — разработки МГУ им. М. В. Ломоносова). Известны распространяемые по лицензии GNU с открытым кодом системы METIS и ParMETIS (университет Миннесоты, 1998—2003 гг.), функциональные возможности которых для данного случая явно избыточны — в реальных программах эффективно распараллеливать имеет смысл лишь отдельные участки кода, критичные по времени выполнения или по использованию аппаратных ресурсов. Следует отметить, что существует и ряд других пакетов, реализующих процедуры разбиения графов для последовательных и параллельных вычислителей.

Пакеты с открытым кодом обычно требуют пересборки после настройки под конкретную задачу путем изменения исходного текста. В отличие от других предлагаемая программная система представлена в виде скомпилированных исполняемых файлов и пересборки не требует. Гибкость достигается достаточным набором API-функций и, при желании, пользователь может совершать все манипуляции с ИГА средствами самого встроенного языка программирования, используя API-вызовы, например, вполне возможна работа и без явного представления ИГА в ЯПФ. Благодаря перечисленным качествам предлагаемая система (по сравнению с известными) рассчитана на более широкий круг специалистов в области исследования и оптимизации параллельного программирования.

Программный интерфейс рассматриваемой системы включает три типа API-вызовов (всего их около 30 шт., причем каждый является Lua-"оберткой" соответствующей C-функции):

- информационные вызовы, которые служат для получения информации об ИГА и его ЯПФ; именно на основании этих данных в дальнейшем принимается решение о применимости одной из стратегий обработки ИГА для решения поставленной задачи, реализуемой в дальнейшем на встроенном языке программирования;
- акционные вызовы, которые служат для реализации конкретных стратегий решения задачи построения расписания;
- вспомогательные вызовы, предназначенные для вывода рассчитанных данных в текстовом и графическом видах для обмена данными с иными приложениями, для взаимодействия с файловой системой и т. п.

Примерами информационных вызовов являются: определить общее число вершин и дуг ИГА, смежную вершину по заданной, общее число ярусов ЯПФ и число операторов на заданном ярусе, диапазон ярусов допустимого расположения каждого оператора, характер распределения ширин ярусов ЯПФ, число и параметры параллельных вычислителей и др. К числу акционных вызовов относятся: создать ЯПФ по указанному ИГА (реализовано в виде отдельного API-вызова); переместить заданный оператор на конкретный ярус ЯПФ (с учетом ненарушения информационных зависимостей), создать новый пустой ярус; уничтожить пустой ярус и др. Выполнение API-функций подробно протоколируется в файле с уникальным именем.

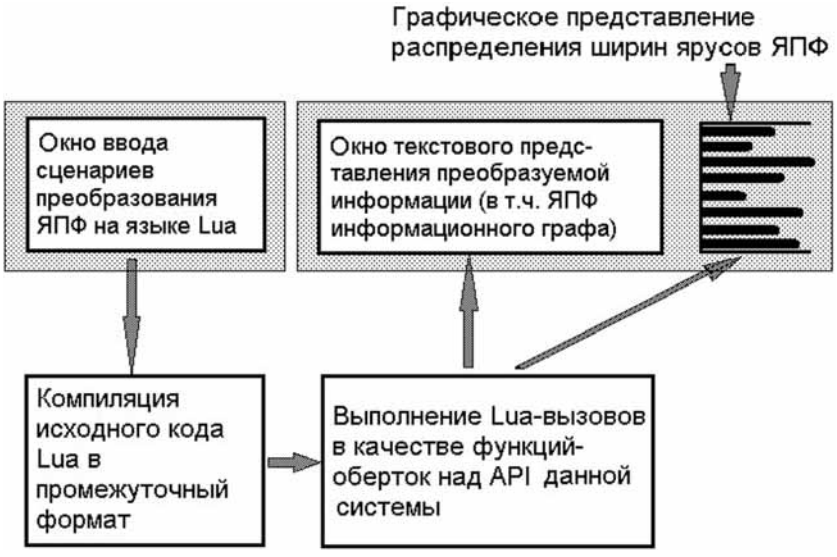


Рис. 3. Схема взаимодействия компонентов разработанной инструментальной программной системы и принципы функционирования пользовательского интерфейса

Данные расчетов выдаются как в текстовом, так и в графическом (ленточная диаграмма распределения операторов по ярусам ЯПФ) видах (см. рис. 3, 4) и могут быть сохранены в распространенных форматах.

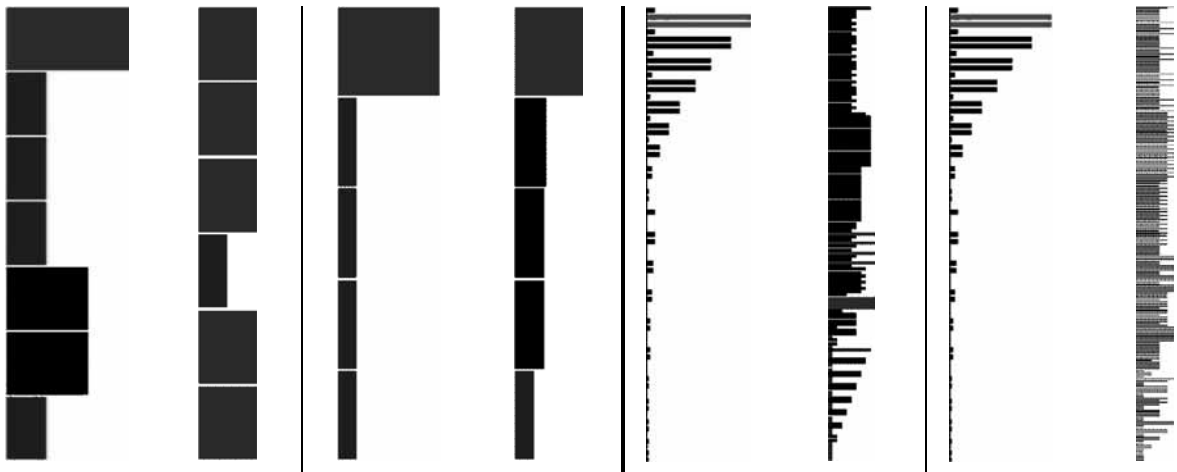
В случае применения данной системы в качестве компонента распараллеливающего компилятора, граф алгоритма априорно существует (компилятор при работе проводит анализ программы на информационные зависимости, т. е. фактически строит граф). Автор для получения корректного ИГА в течение ряда лет использует программный симулятор вычислителя потоковой (DATA-FLOW) архитектуры [6], в котором программа отлаживается и в дальнейшем экспортируется в файловый формат списка смежных вершин. Особенностью этого симулятора является возможность моделирования асинхронного режима выполнения операторов (для каждого оператора задается время исполнения в условных единицах).

Результаты, их обсуждение и перспективы

Одной из типовых является задача определения эффективных стратегий балансировки ЯПФ информационного графа алгоритма, здесь под балансировкой понимается равномерность распределения операторов по ярусам ЯПФ. На первой стадии из всех реалий конкретной МВС учитывается только число параллельных вычислителей. Разработанные методы учета гетерогенности (по параметрам объема памяти, вычислениям с различными типами данных, векторных вычислений и т. п.) вычислительного поля с возможностью выбора выполнения определенного оператора на соответствующем по параметрам вычислителе потребовали дополнения системы приблизительно 20 API-вызовами.

В качестве *параметра эффективности* собственно стратегии преобразования ЯПФ использовано число перестановок операторов с яруса на ярус (аналогично задачам сортировки) для получения заданного результата (лучшей стратегии соответствует минимум числа перестановок). При таком подходе превышение высоты ЯПФ над значением критического пути удобно рассматривать в качестве штрафной функции.

На рис. 4 в графической интерпретации (ленточные диаграммы распределения числа операторов по ярусам) приведены результаты применения различ-



Исходная ЯПФ	Преобразованная ЯПФ	Исходная ЯПФ	Преобразованная ЯПФ	Исходная ЯПФ	Преобразованная ЯПФ	Исходная ЯПФ	Преобразованная ЯПФ
(H=6; W=4; k=4; σ =1,07)	(H=6; W=2; k=2; σ =0,373)	(H=5; W=125; k=5; σ =40)	(H=5; W=85; k=3,8; δ=25,3)	(H=63; W=90; k=90; σ =21,8; $\overline{W}$=12,8)	(H=141; W=10; k=10; δ=2,69; $\overline{W}$=5,71)	(H=63; W=90; k=90; σ =21,8; $\overline{W}$=12,8)	(H=230; W=5; k=5; σ =1,15; $\overline{W}$=3,50)
Вычисление вещественных корней полного квадратного уравнения ($\overline{W}$ =1,83)		Умножение квадратных матриц пятого порядка классическим методом ($\overline{W}$ =45)		Решение системы линейных уравнений (СЛАУ) десятого порядка методом Гаусса (W_0=10)		Решение СЛАУ десятого порядка методом Гаусса (W_0=5)	
Преобразование ЯПФ согласно стратегии #01 (условие H=const)				Преобразование ЯПФ согласно стратегии #02 (условие $W<=W_0$)			

Рис. 4. Диаграммы распределения операторов по ярусам для исходных и преобразованных ЯПФ с помощью различных стратегий преобразования для разных алгоритмов (вариант гомогенного поля параллельных вычислителей)

Таблица 1

Эффективность применения стратегии балансировки ширины ЯПФ при неизменной высоте (числитель — для исходной ЯПФ, знаменатель — для преобразованной ЯПФ)

ИГА (дуг/операторов)	Число ярусов ЯПФ	Средняя ширина ЯПФ	Неравномерность ширины ЯПФ	Среднее квадратичное отклонение ширины ЯПФ	Число перестановок операторов
slau_2a (18/9)	7	1,29	2/2	0,452/0,452	0
slau_3a (56/28)	13	2,15	6/6	1,70/1,51	4
slau_5a (230/115)	25	4,60	20/20	5,42/5,11	13
slau_10a (1610/805)	63	12,8	90/90	21,8/20,7	81
mnk_10 (132/66)	16	4,13	22/22	4,79/4,78	1
mnk_20 (252/126)	26	4,85	42/42	7,52/7,51	1
korr_10 (174/88)	15	5,87	32/32	7,15/7,14	4
korr_20 (334/168)	25	6,72	62/44	11,4/7,77	22
m_matr_5 (450/225)	5	45,0	5/3,40	40/20,7	30
m_matr_10 (3800/1300)	10	190	10/6	270/137	407

ных стратегий (обе относятся к классу "быстрых" методов) преобразования ЯПФ для двух случаев. Первый из них — стратегия #01 — перемещение операторов с наиболее нагруженных на наименее нагруженные ярусы при условии сохранения высоты ЯПФ, критерий остановки алгоритма — перебор всех операторов, которые могут быть перенесены "с холмов во впадины" в целях балансировки ЯПФ. Случай второй — стратегия #02 — балансировка при условии увеличения высоты ЯПФ, критерий останова — "ужатие", т. е. снижение ширины ЯПФ до значения, не превышающего заданного, для нескольких ИГА распространенных вычислительных процедур.

Эффективность обработки данных иллюстрируется данными табл. 1 (стратегия #01) и табл. 2 (стратегия #02 при ограничениях ширины ЯПФ разного размера). Информационные графы slau_2a, slau_3a, slau_5a и slau_10a — процедуры решения СЛАУ порядков 2, 3, 5, 10, соответственно, классическим методом Гаусса; mnk_10 и mnk_20 — процедуры линейной аппроксимации по методу наименьших квадратов для 10 и 20 точек; korr_10 и korr_20 — процедуры

Таблица 2

Эффективность применения стратегии получения ЯПФ не более заданной ширины совместно с балансировкой ширин ярусов (для ширины ЯПФ не более 5)

ИГА (дуг/операторов)	Число ярусов ЯПФ	Средняя ширина ЯПФ	Неравномерность ширины ЯПФ	Среднее квадратичное отклонение ширины ЯПФ	Число перестановок операторов
slau_2a (18/9)	7/7	1,29/1,29	2/2	0,452/0,452	0
slau_3a (56/28)	13/15	2,15/1,87	6/3	1,7/0,806	6
slau_5a (230/115)	25/39	4,6/2,95	20/5	5,42/1,36	70
slau_10a (1610/805)	63/230	12,8/3,5	90/5	21,8/1,15	1234
mnk_10 (132/66)	16/21	4,13/3,14	22/5	4,79/1,28	27
mnk_20 (252/126)	26/35	4,85/3,6	42/5	7,52/1,18	67
korr_10 (174/88)	15/23	5,87/3,83	32/5	7,15/1,2	51
korr_20 (334/168)	25/41	6,72/4,1	62/5	11,4/1,06	124
m_matr_5 (450/225)	5/64	45/3,52	5/1,33	40/0,5	451
m_matr_10 (3800/1300)	10/544	190/3,49	10/1,33	270/0,5	6152

Таблица 3

Зависимость трудоемкости преобразований ЯПФ (в единицах числа перестановок операторов с яруса на ярус) стратегии #02 получения ЯПФ не более заданной ширины W_0

ИГА (дуг/операторов/ярусов)	Ширина W_0					
	10	8	6	4	2	1
slau_10a (1610/805/63)	926	1038	1135	1340	1601	1920
m_matr_10 (3800/1300/10)	5232	5232	5828	6152	6964	7776
korr_20 (334/168/25)	91	91	121	160	212	283
mnk_20 (252/126/26)	51	61	61	79	127	183
e313_o211_t33 (313/211/33)	52	60	103	136	207	288
e451_o276_t31 (451/276/31)	85	98	177	207	312	425
e2367_o1402_t138 (2367/1402/138)	496	613	969	1181	1730	2334
e17039_o9858_t200 (17039/9858/200)	14 579	16 129	17 418	20 250	23 521	27 507

определения коэффициента парной корреляции для 10 и 20 точек; m_matr_5 и m_matr_10 — процедуры умножения квадратных матриц порядков 5 и 10 классическим методом (гомогенное поле параллельных вычислителей). Неравномерность распределения числа операторов по ярусам ЯПФ оценивали неравномерностью распределения операторов по ярусам k, дополнительно определяли среднее квадратичное отклонение ширин ярусов.

Следует заметить, что потенциал балансировки ЯПФ при неизменной его высоте ограничен (см. табл. 1), поэтому больший практический интерес представляют стратегии с увеличением высоты ЯПФ (см. табл. 2).

Табл. 3 иллюстрирует зависимость трудоемкости преобразований ЯПФ (в единицах перестановок операторов с яруса на ярус) от заданного предела ширины ЯПФ для графов ранее рассмотренных распространенных алгоритмов и нескольких графов, сгенерированных программой-графогенератором (для данной стратегии преобразования ЯПФ рост трудоемкости близок к полиномиальной функции от степени "ужатия" ширины ЯПФ).

Как видно из данных табл. 1—3, ранее описанные (не являющиеся излишне изощренными) стратегии позволяют снизить неравномерность ширин ЯПФ при заданных ограничениях, однако с разной эффективностью для различных ИГА. В целом наблюдается повышение эффективности стратегии при увеличении сложности (и, соответственно, вариативности) ИГА. Особенно интересна и практически важна задача априорного (до проведения собственно преобразований ЯПФ) предсказания уровня эффективности стратегий (задача распознавания ситуации). Большего эффекта можно добиться путем совершенствования стратегий и разумного их совместного применения.

Заключение

Использование встроенного высокоуровневого языка программирования для моделирования методов планирования задач для параллельных вычислительных систем позволило добиться эффективности и гибкости в реализации поставленной цели, что было бы затруднительно при использовании иных подходов. Получены количественные параметры текущего приближения при совершенствовании стратегий оптимизации выполнения параллельных программ. Результаты исследований применимы для анализа алгоритмов (в плане выявления эффективности выполнения параллельных программ), при разработке распараллеливающих компиляторов и в процессе обучения специалистов в области технологий параллельного программирования.

Список литературы

1. **Воеводин В. В., Воеводин Вл. В.** Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 608 с.
2. **AlgoWiki.** Открытая энциклопедия свойств алгоритмов. URL: http://algowiki-project.org (дата обращения 01.09.2016).
3. **Гери М., Джонсон Д.** Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. 416 с.
4. **Иерузалимски Р.** Программирование на языке Lua. М.: ДМК Пресс, 2014. 382 с.
5. **Баканов В. М.** Программный инструментарий анализа информационной структуры алгоритмов по их информационным графам // Параллельные вычислительные технологии (ПаВТ'2016): Тр. междунар. науч. конф. Архангельск, 28 марта — 01 апреля 2016 г. Челябинск: Издательский центр ЮУрГУ, 2016. С. 432—441.
6. **Баканов В. М.** Управление динамикой вычислений в процессорах потоковой архитектуры для различных типов алгоритмов // Программная инженерия. 2015. № 9. С. 20—24.

Program Complex for Development Methods Construction of Optimal Frame Parallel Program

V. M. Bakanov, vbakanov@hse.ru, Higher School of Economics, National Research University, 101000, Moscow, Russian Federation

Corresponding author:

Bakanov Valery M., Professor, Higher School of Economics, National Research University, 101000, Moscow, Russian Federation
E-mail: vbakanov@hse.ru

Received on October 10, 2016
Accepted on October 28, 2016

Approaches to the problem of the development of rational (seeking optimal) planning techniques (drawing up the framework, the skeleton program in terms of parallelism) to perform the tasks of parallel computing systems with homogeneous and heterogeneous field calculators. The informational graphs are used as a formalization of the algorithm. The possible criteria and parameters of the optimization planning methods with the use of stacked parallel form (SPF) information graph algorithm, or without it are described. Two basic strategies are formulated — preserving run time (without increasing the height of the original SPF) or an increase in the height of SPF. The first one is implemented on the basis of statements on location variability tiers SPF (while retaining the same information dependencies as in the original graph), the second one — by adding tiers in SPF and the transfer of the operators "top-down" on the written stage. The data on the developed for the implementation of such methods software system (the program stand) are provided. To implement methods (strategies) of rational development of parallel programs, the built-in high-level Lua scripting language is used. The confirmation of its effectiviness in this capacity is provided. The information on the set of API calls of the system is provided. A criterion of computational complexity of parallel programs execution plan building procedures is suggested. We give qualitative (in the form of strip-chart) and quantitative results of the application of some of the proposed strategies for building a rational planning of parallel programs with respect to the common algorithms of data processing.

Keywords: *graph representations of the algorithm, analysis of the information structure of the program, longline-parallel form information graph, rational parameters of parallel programs, integrated lua scripting language, strategy of building a rational plan for the parallel program execution*

For citation:

Bakanov V. M. Program Complex for Development Methods Construction of Optimal Frame Parallel Program, *Programmnaya Ingeneria,* 2017, vol. 8, no. 2, pp. 58—65.

DOI: 10.17587/prin.8.58-65

References

1. **Voevodin V. V., Voevodin Vl. V.** *Parallel'nye vychislenija* (Parallel computing), St. Petersburg, BHV-Petersburg, 2002, 608 p. (in Russian).
2. **AlgoWiki.** Open Encyclopedia properties of algorithms, available at: http: // algowiki-project.org (reference date 01.09.2016).
3. **Gary M., Johnson D.** *Vychislitel'nye mashiny i trudnoreshaemye zadachi* (Computers and Intractability), Moscow, Mir, 1982, 416 p. (in Russian).
4. **Ierusalimschy R**. *Programmirovanie na jazyke Lua* (Programming in Lua), Moscow, DMK Press, 2014, 382 p. (in Russian).
5. **Bakanov V. M.** Programmnyj instrumentarij analiza informacionnoj struktury algoritmov po ih informacionnym grafam (Software tool analyzes information structure algorithms for their information graphs), *Parallel Computing Technologies (PaVT'2016): Proceedings of the International Scientific Conference,* Arkhangelsk, 28.03—01.04.2016, Chelyabinsk, Publishing Center South Ural State University, 2016, pp. 432—441 (in Russian).
6. **Bakanov V. M.** Upravlenie dinamikoj vychislenij v processorah potokovoj arhitektury dlja razlichnyh tipov algoritmov (Computing the dynamics of management processors streaming architecture for the different types of algorithms), *Programmnaya Ingeneria,* 2015, no. 9, pp. 20—24 (in Russian).

УДК 519.688 DOI: 10.17587/prin.8.66-75

М. Е. Куляс, аспирант, e-mail: KuliasMY@mpei.ru,
ФГБОУ ВО "НИУ "Московский энергетический институт"

О синтезе программ умножения длинных целых чисел*

Предложены два варианта комбинированных рекурсивных алгоритмов умножения длинных целых чисел, сочетающих асимптотически быстрый метод Карацубы, а также метод сдвигов и сложений на нижних уровнях рекурсии. Представлены варианты рекурсивной и последовательной (линейной) программной реализации предложенных алгоритмов, результаты экспериментального исследования их эффективности. Даны оценки объема требуемой памяти.

Ключевые слова: *умножение длинных чисел, метод Карацубы, последовательная программа, рекурсия, сложность вычислений*

Введение

В связи с постоянно возрастающими потребностями современного общества в электронном документообороте, в решении задач по кодированию и по защите информации создают и совершенствуют необходимые для этого программные библиотеки современной компьютерной алгебры. Актуальной становится задача повышения эффективности алгоритмов и методов реализации вычислений в таких библиотеках [1, 2].

Арифметику длинных целых чисел и полиномов применяют при реализации многих алгоритмов вычисления и проверки цифровой подписи, систем шифрования с открытыми ключами, кодов коррекции ошибок. Еще одной областью использования длинной арифметики является разработка математического программного обеспечения для микропроцессоров с маленькой разрядностью машинного слова (8- и 16-битные микроконтроллеры и процессоры), которые широко используют в бесконтактных банковских и транспортных картах, различных автономных датчиках, электронных устройствах персональной идентификации. Как правило, такие устройства имеют очень ограниченный объем доступной памяти, небольшую тактовую частоту процессорного устройства и весьма скромные возможности распараллеливания вычислений. Это связано с необходимостью снижения энергопотребления и массогабаритных показателей. Традиционные подходы к разработке математических библиотек (с использованием многочисленных операций ветвления, циклов, рекурсивных вызовов, динамического выделения памяти, многопоточности) для таких портативных устройств могут не дать желаемого эффекта по скорости вычислений, или они вообще неприменимы (например, в силу ограничения на объем доступной памяти). По этим причинам необходим пересмотр существующих подходов к проектированию соответствующих программных средств.

Одним из подходов к увеличению скорости вычислений может служить разработка линейной (последовательной) программы. Такая программа состоит из последовательности арифметических и логических операций, а также операций обращения к памяти. При этом отсутствуют рекурсивные вызовы, циклы и ветвления, что потенциально может увеличить скорость вычислений. Такое ускорение возможно в силу отсутствия накладных расходов на организацию циклов и рекурсии. Не происходит также сброса конвейера процессора при неправильном предсказании переходов [3] на участках ветвления в программе. Следует заметить, что размер последовательной программы может быть больше ее итеративного или рекурсивного аналога, а скорость вычислений напрямую зависит от объема доступной памяти и эффективности работы механизмов кэширования процессора (если таковые имеются). Например, частичное разворачивание циклов и линеаризацию функций применяют в таких популярных математических библиотеках, как GMP [4] и MPIR [5]. В криптографической библиотеке NaCl, часто используемой в микроконтроллерах, дополнительно сокращено число ветвлений и обращений к памяти [6].

Целью работы, результаты которой представлены в статье, является разработка линейных программ умножения длинных целых чисел и экспериментальное определение критериев их применимости. Операция умножения входит в состав более сложных алгоритмов вычислений в конечных алгебраических структурах и, как правило, определяет быстродействие таких алгоритмов. В работе [7], например, предложены варианты автоматического синтеза последовательных программ умножения по формуле Карацубы [8] с оценкой числа базовых операций и объема требуемой памяти. В настоящей работе предпринята попытка улучшить результаты, полученные

* *Работа подготовлена при финансовой поддержке РФФИ, проект №14-01-00671-А.*

авторами работы [7], за счет использования комбинированных алгоритмов умножения [9] и более эффективного использования памяти.

Алгоритмы умножения длинных целых чисел

Под длинными числами будем понимать числа, разрядность которых превышает разрядность машинного слова. Для представления таких чисел предлагается использовать позиционную систему счисления по основанию $p = 2^w$, где параметр w часто выбирают равным разрядности машинного слова (32 или 64-бита для современных процессоров):

$$A=(a_{s-1},a_{s-2},\ \ldots,\ a_0)_p=\sum_{i=0}^{s-1}a_i p^i.$$

При этом n-битное число A удобно хранить в виде массива целых, беззнаковых, w-битных чисел размерности $s = [n/w]$. Элементы a_i массива A могут принимать значения от 0 до $p - 1$. Знак числа при этом предлагается хранить и анализировать (в процессе выполнения арифметических операций) отдельно.

В современных версиях языка Си имеются расширенные целочисленные типы данных двойной точности (например, целочисленный беззнаковый тип uint128_t в 64-битной операционной системе), позволяющие хранить результат умножения двух машинных слов. К сожалению, умножить два 128-битных числа типа uint128_t стандартными средствами языка Си уже не получается. Результатом такого умножения будет усеченное до 128 бит значение произведения младших разрядов сомножителей, что приводит к необходимости использования соответствующих алгоритмов длинной арифметики.

Рассматривая алгоритмы умножения, остановимся на методе сдвигов и сложений [1], а также на асимптотически более быстром методе Карацубы [1, 8]. Эти методы являются наиболее эффективными при реализации операции умножения длинных целых чисел с диапазоном значений, представляющим наибольший практический интерес в наши дни. Существуют асимптотически еще более быстрые алгоритмы, основанные на быстром преобразовании Фурье, но они имеют большую мультипликативную константу в оценке сложности и показывают существенное преимущество по сравнению с методом сдвигов и сложений и методом Карацубы только для чисел в несколько десятков тысяч десятичных знаков.

По алгоритму сдвигов и сложений произведение двух чисел $A=\sum_{i=0}^{s-1}a_i p^i$ и $B=\sum_{i=0}^{s-1}b_i p^i$ (для определенности будем полагать их одинаковой разрядности) вычисляют по формуле

$$AB=\sum_{i=0}^{s-1}a_i p^i\sum_{j=0}^{s-1}b_j p^j=\sum_{i=0}^{s-1}\sum_{j=0}^{s-1}a_i b_j p^{i+j}.$$

Основу этого метода (алгоритм 1) составляют два вложенных цикла. Во внешнем цикле загружается значение текущего разряда одного операнда. Во внутреннем цикле выполняется операция умножения (с накоплением) текущего разряда одного операнда на все разряды второго операнда.

Алгоритм 1. Умножение длинных целых чисел методом сдвигов и сложений

ВХОД: два массива A и B с операндами размером s элементов каждый, $A = (A[s - 1], A[s - 2], ..., A[0])$, $B = (B[s - 1], B[s - 2], ..., B[0])$, где $A[i]$, $B[i]$ принимают значения от 0 до $p - 1$; $p = 2^w$; $0 \leqslant i \leqslant s - 1$.

ВЫХОД: массив P из $2s$ элементов, содержащий результаты операции умножения $P = AB = (P[2s - 1], P[2s - 2], ..., P[0])$.

Шаг 1. Установить $U \leftarrow 0$.

Шаг 2. Для всех i от 0 до $s - 1$ выполнить:

  Шаг 2.1. $(U, V)_p \leftarrow A[i]B[0] + U$.

  Шаг 2.2. $P[i] \leftarrow V$.

Шаг 3. Установить $P[s] \leftarrow U$.

Шаг 4. Для всех i от 0 до $s - 1$ выполнить:

  Шаг 4.1. Установить $U \leftarrow 0$.

  Шаг 4.2. Для всех j от 0 до $s - 1$ выполнить:

    Шаг 4.2.1. $(U, V)_p \leftarrow P[i + j] + A[i]B[j] + U$.

    Шаг 4.2.2. $P[i + j] \leftarrow V$.

  Шаг 4.3. Установить $P[i + s] \leftarrow U$.

Шаг 5. Вернуть P.

Разрядности промежуточных вычислений определяются операциями, указанными в шаге 4.2.1 алгоритма 1: $(U, V)_p \leftarrow P[i + j] + A[i]B[j] + U$. Здесь $(U, V)_p$ означает запись числа по основанию p, т. е. $(U, V)_p = V + U2^w$. Максимальное значение $(U, V)_p$ ограничено $p^2 - 1$, поскольку значения $P[i + j]$, $A[i]$, $B[j]$ и U не больше $p - 1$.

Асимптотическая сложность алгоритма оценивается как $O(s^2)$ [1]. Суммарно необходимо выполнить s^2 инструкций умножения w-битных чисел. Следует отметить, что кроме $2w$-битного компонента $(U, V)_p$ алгоритм не требует дополнительной памяти (для хранения результатов промежуточных вычислений). Для эффективной реализации этого метода требуется процессорная инструкция умножения целых w-битных чисел с сохранением $2w$-битного результата.

При рассмотрении метода умножения Карацубы будем полагать, что n (число бит, необходимых для представления операндов) является степенью двойки. В этом случае любое n-битное целое число A можно записать по основанию $2^{n/2}$ (как сумму двух $n/2$-битных чисел):

$$A=(A_1, A_0)_{n/2}=A_0+A_1 2^{n/2}.$$

Тогда по формуле Карацубы [8] произведение двух таких чисел вычисляется через три операции умножения $n/2$-битных чисел и несколько дополнительных операций сложения/вычитания, имеющих линейную сложность, и определяется формулой:

$$\begin{aligned}AB=&A_0B_0\left(1+2^{n/2}\right)+\\&+(A_1-A_0)(B_0-B_1)2^{n/2}+A_1B_1\left(2^n+2^{n/2}\right).\end{aligned}\qquad(1)$$

При практической реализации вычислений по этой формуле следует принять во внимание тот факт, что средний компонент в формуле (1) может принимать как положительные, так и отрицательные значения. Таким образом, если $M = (A_1 - A_0)(B_0 - B_1)$ больше 0, тот этот компонент входит со знаком "+", иначе со знаком "–", и операция сложения заменяется на вычитание. Такое ветвление вносит большую задержку в вычисления даже на современных процессорах, поскольку в силу случайного характера знака компонента M аппаратные блоки предсказания переходов не могут эффективно определить нужную ветвь. В результате неправильного предсказания перехода конвейер процессора вынужден перезапуститься, а подсистема кэш-памяти — заново загрузить необходимый код программ и данных. Кроме того, для программных приложений в области криптографии критически важно избегать таких ветвлений, так как они могут привести к утечке секретной информации вследствие того, что времена обработки различных ветвей различны.

При рекурсивной реализации вычислений по формуле (1) необходимо где-то хранить знаки сомножителей $(A_1 - A_0)$ и $(B_0 - B_1)$. В работе [7], например, было предложено задействовать для этого дополнительный объем памяти. Чтобы избежать дополнительного выделения памяти и избавиться от нежелательных ветвлений, предлагается вместо определения разностей $(A_1 - A_0)$ и $(B_0 - B_1)$ вычислять их абсолютные разности $|A_1 - A_0|$ и $|B_0 - B_1|$ с одновременным определением бита знака t компонента $M_{abs} = |A_1 - A_0||B_0 - B_1|$. Такая операция реализуется следующим образом:

- вычисляются s разрядов разности $(A_1 - A_0)$ и t_a — бит заема из старшего разряда (бит знака);
- из значения t_a формируется w-битная маска $mask_a = -t_a$;
- выполняется операция "исключающее или" со всеми разрядами разности $(A_1 - A_0)$ и маской $mask_a$; если $t_a = 0$ (то есть $A_1 \geqslant A_0$), то операция "исключающее или" не меняет значения разности $(A_1 - A_0)$; если же $t_a = 1$, то все биты маски устанавливаются в "1", и в результате описанной операции получается значение $|A_1 - A_0|$, записанное в формате дополнения до единицы;
- чтобы иметь представление $|A_1 - A_0|$ в формате дополнения до двойки, необходимо дополнительно прибавить значение t_a с корректной обработкой возникающих при этом переносов;
- аналогичным образом вычисляются значения $|B_0 - B_1|$ и t_b;
- результирующий знак произведения $M_{abs} = |A_1 - A_0||B_0 - B_1|$ определяется по значениям t_a и t_b: $t = t_a$ xor t_b, где xor — операция "исключающее или".

Чтобы избавиться от нежелательного ветвления, можно провести операцию "исключающее или" со всеми разрядами M_{abs} и маской $mask_t = -t$ с прибавлением бита маски t (аналогично описанному методу). Для увеличения скорости вычислений эту операцию можно совместить с операцией сложения согласно формуле (1).

Более детальный анализ формулы (1) позволяет дополнительно сократить число операций сложения, так как некоторые слагаемые в ней вычисляются дважды [10].

Запишем слагаемые A_0B_0 и A_1B_1 по основанию $n/2$ и введем соответствующие обозначения:

$$(A_0B_0) = L_0 + 2^{n/2}L_1;\ (A_1B_1) = \\ = H_0 + 2^{n/2}H_1;\ T = L_1 + H_0,$$

тогда формулу (1) можно преобразовать в следующую:

$$\begin{aligned} AB = (L_0 + 2^{n/2}L_1)(1 + 2^{n/2}) + (-1)^t M_{abs}2^{n/2} + \\ + (H_0 + 2^{n/2}H_1)(2^n + 2^{n/2}) = 2^{3n/2}H_1 + \\ + 2^n(L_1 + H_0 + H_1) + 2^{n/2}(L_1 + H_0 + L_0 + \\ + (-1)^t M_{abs}) + L_0 = 2^{3n/2}H_1 + 2^n(T + H_1) + \\ + 2^{n/2}(T + L_0 + (-1)^t M_{abs}) + L_0. \end{aligned} \quad (2)$$

Таким образом, заранее вычисленное значение T позволяет сократить $n/2$ операций сложения.

Формулу (2) можно применять рекурсивно с глубиной рекурсии не более чем $\log_2(n/w)$ уровней, где w — разрядность машинного слова. Рекурсивный вариант реализации данного метода для максимальной глубины вложенности (базой рекурсии в этом случае является умножение двух процессорных слов с разрядностью w бит каждое) имеет большую мультипликативную константу в оценке сложности и на практике оказывается хуже метода сдвигов и сложений [9]. Существенно повысить скорость вычислений по методу Карацубы можно, используя комбинированный алгоритм умножения. Суть такого метода заключается в том, что на этапе прямого хода рекурсии (по методу Карацубы) происходит понижение разрядов сомножителей до некоторого значения n_0 (называемого порогом рекурсии), по достижению которого произведение n_0-битных операндов вычисляется методом сдвигов и сложений. Это позволяет уменьшить мультипликативную константу в оценке сложности метода (для некоторого диапазона значений входных данных). Асимптотическую сложность такого метода оценивают [2] как $O\left(n_0^2 3^{\log(n/n_0)}\right)$.

Для алгоритмического описания метода введем перечисленные далее обозначения функций, выполняющих элементарные арифметические операции над длинными целыми числами.

Функция mul_base(P, A, B, s) — умножение длинных целых чисел, представленных массивами A, B размерности s элементов каждый, по методу сдвигов и сложений. Здесь P, A, B — адреса произведения и операндов соответственно.

Функция adiff(R, A, B, s) — вычисление абсолютной разности $(-1)^t R = |A - B|$ длинных чисел, представленных массивами A, B размерности s элементов каждый. Функция также вычисляет и возвращает знак t разности $(A - B)$. Здесь R, A, B — адреса результата и операндов соответственно.

Функция adc(R, A, B, c, s) — вычисление суммы двух длинных чисел $(c_{s+1}, R) = A + B + c$, пред-

ставленных массивами A, B размерности s элементов каждый, с учетом переноса c. Функция также вычисляет и возвращает значение бита переноса c_{s+1} в старший $(s+1)$-й разряд. Здесь R, A, B — адреса результата и операндов соответственно, c — значение бита переноса в 0-й разряд.

Функция add_if_flag(R, A, B, *flag*, s) — вычисление суммы двух длинных чисел $(c_{s+1}, R) = (A + B)$*flag*, представленных массивами A, B размерности s элементов каждый, с учетом значения флага *flag* (*flag* $\in \{0, 1\}$). Функция также вычисляет и возвращает значение бита переноса c_{s+1} в старший $(s+1)$-й разряд. Здесь R, A, B — адреса результата и операндов соответственно.

Функция sub(R, A, B, s) — вычисление разности двух длинных чисел $R = (A - B)$, представленных массивами A, B размерности s элементов каждый. Здесь R, A, B — адреса результата и операндов соответственно.

Рекурсивная реализация алгоритма вычисления произведения длинных целых чисел по формуле (2) представлена далее. Для сокращенного обозначения этого метода будем использовать аббревиатуру SKML (*Subtractive Karatsuba's MuLtiplication method*), поскольку средний член в формуле (1) определяется через произведение разностей $A_1 - A_0$ и $B_0 - B_1$. Существует также аддитивный вариант алгоритма Карацубы, который будет рассмотрен далее по тексту.

Алгоритм 2. Умножение длинных целых чисел по методу SKML

SKML(P, A, B, s, n_0): P, A, B — адреса произведения и операндов соответственно; s — разрядность операндов; n_0 — порог рекурсии.

ВХОД: два массива A и B с операндами, размером s элементов каждый, $A = (A[s-1], A[s-2], ..., A[0])$, $B = (B[s-1], B[s-2], ..., B[0])$, где $A[i]$, $B[i]$ принимают значения от 0 до $p-1$; $p = 2^w$; $0 \leqslant i \leqslant s-1$; порог рекурсии n_0; массив P из $6s + 2\log_2(n/n_0) - 4s/2^{\log_2(n/n_0)}$ элементов.

ВЫХОД: массив P из $2s$ элементов, содержащий результаты операции умножения $P = AB = (P[2s-1], P[2s-2], ..., P[0])$.

Шаг. 1. Если $s \leqslant n_0/w$, то

Шаг 1.1. Вычислить AB, вызвав mul_base(P, A, B, s).

Шаг 1.2. Вернуть P.

Шаг 2. Рекурсивно вычислить A_0B_0, вызвав SMKL(P, A, B, $s/2$, n_0).

Шаг 3. Рекурсивно вычислить A_1B_1, вызвав SMKL($P + s$, $A + s/2$, $B + s/2$, $s/2$, n_0).

Шаг 4. Вычислить $|A_1 - A_0|$ и t_a, вызвав $t_a =$ = adiff($P + 2s$, $A + s/2$, A, $s/2$).

Шаг 5. Вычислить $|B_0 - B_1|$ и t_b, вызвав $t_b =$ = adiff($P + 2s + s/2 + 1$, B, $B + s/2$, $s/2$).

Шаг 6. Вычислить $t = t_a$ xor t_b.

Шаг 7. Рекурсивно вычислить M_{abs}, вызвав SMKL($P + 3s + 2$, $P + 2s$, $P + 2s + s/2 + 1$, $s/2$, n_0).

Шаг 8. Вычислить $(-1)^t M_{abs}$, записать s-разрядный результат, начиная с адреса $(P + 3s + 2)$.

Шаг 9. Вычислить $(c_t, T) = L_1 + H_0$ (c_t — бит переноса), вызвав c_t = adc($P + 2s + s/2 + 1$, $P + s/2$, $P + s$, 0, $s/2$).

Шаг 10. Вычислить $(L_0 + T)$ и бит переноса c_l, вызвав c_l = adc($P + 2s$, P, $P + 2s + s/2 + 1$, 0, $s/2$).

Шаг 11. Вычислить $(H_1 + T + c_l + c_t)$ и бит переноса c_h, вызвав c_h = adc($P + 2s + s/2$, $P + s + s/2$, $P + 2s + s/2 + 1$, $c_l + c_t$, $s/2$).

Шаг 12. Сложить результат с $(-1)^t M_{abs}$ и вычислить c_m, вызвав c_m = adc($P + s/2$, $P + s/2$, $P + 2s + s/2$, 0, $s/2$).

Шаг 13. Обработать переносы, вычислив $P[s + s/2] \leftarrow$ $\leftarrow P[s + s/2] + c_h + c_m$.

Шаг 14. Вернуть P.

На верхнем уровне рекурсии задействуется $2s + 4s/2 + 2$ элементов массива P (разрядностью w бит каждый). Младшие $2s$ элементов массива P предназначены для хранения результатов операции умножения, а оставшиеся $4s/2 + 2$ элементов — для промежуточных вычислений (рис. 1).

На каждом уровне рекурсии произведения A_0B_0 и A_1B_1 занимают по s элементов каждое и записываются с нулевым смещением относительно текущего базового адреса массива P. Абсолютные разности $|A_1 - A_0|$ и $|B_0 - B_1|$ занимают по $s/2$ элементов каждая и записываются со смещениями $2s$ и $2s + s/2 + 1$ соответственно, как показано на рис. 1. Далее со смещением $3s + 2$ записывается M_{abs}. Значение T предлагается записывать в массив P со смещением $2s + s/2 + 1$, затирая при этом $|B_0 - B_1|$. $L_0 + T$ за-

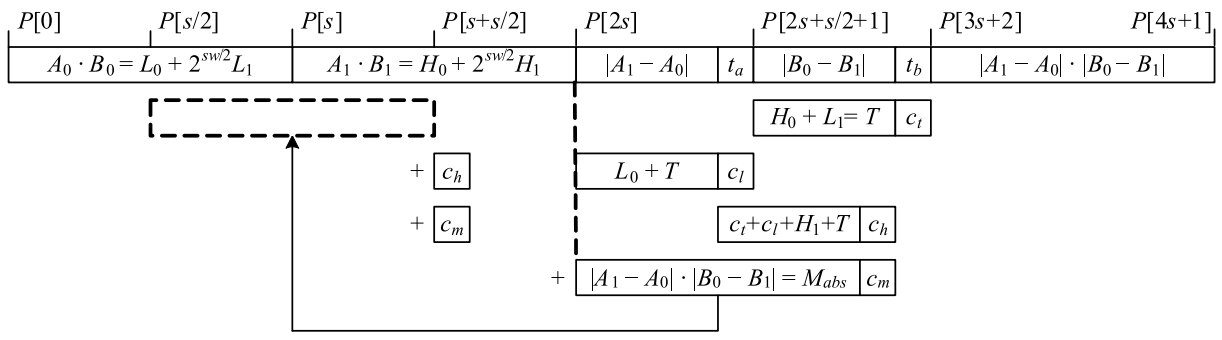


Рис. 1. Выделение памяти для рекурсивного алгоритма SKML

писывается поверх $|A_1 - A_0|$. Следующий по вложенности уровень рекурсии задействует дополнительно $4s/4 + 2$ элементов P, при этом вычисленные на верхнем уровне значения $|A_1 - A_0|$ и $|B_0 - B_1|$ становятся исходным данными для рекурсивного вычисления M_{abs}. Таким образом, на каждом уровне рекурсии задействуется дополнительно $4s/2^i + 2$ элементов массива P для хранения результатов промежуточных вычислений (i — текущий уровень рекурсии). Для корректной работы алгоритма суммарно требуется выделить массив P из $2s + \sum_{i=1}^{\log_2 n/n_0} (4s/2^i + 2) = 6s + 2\log_2(n/n_0) - 4s/2^{\log_2(n/n_0)}$ элементов.

Следует заметить, что существует вариант формулы Карацубы, в котором знаки среднего члена всегда определены [11]:

$$AB = A_0B_0(1 - 2^{n/2}) + (A_1 + A_0)(B_0 + B_1)2^{n/2} + A_1B_1(2^n - 2^{n/2}). \quad (3)$$

В случае реализации вычислений по формуле (3) не требуется обрабатывать и хранить знаки промежуточных вычислений, но сложность вычислений при этом возрастает, так как компоненты $A_1 + A_0$ и $B_0 + B_1$ занимают $n/2 + 1$ бит каждый, а их произведение требует уже $n + 2$ бит. Таким образом, нарушается условие применения рекурсии, так как разрядность сомножителей уже не является степенью двойки.

Сомножители $A_1 + A_0$ и $B_0 + B_1$ можно представить по основанию $n/2$: $A_1 + A_0 = (c, A_2)_{n/2} = c2^{n/2} + A_2$; $B_0 + B_1 = (d, B_2)_{n/2} = d2^{n/2} + B_2$. Формула (3) при этом может быть переписана в следующем виде:

$$\begin{aligned} AB &= A_0B_0(1 - 2^{n/2}) + (A_1 + A_0)(B_0 + B_1)2^{n/2} + \\ &+ A_1B_1(2^n - 2^{n/2}) = A_0B_0(1 - 2^{n/2}) + \\ &+ (c2^{n/2} + A_2)(d2^{n/2} + B_2)2^{n/2} + A_1B_1(2^n - 2^{n/2}) = \\ &= A_0B_0(1 - 2^{n/2}) + cd2^{3n/2} + cB_22^n + dA_22^n + \\ &+ A_2B_22^{n/2} + A_1B_1(2^n - 2^{n/2}). \end{aligned} \quad (4)$$

Так как слагаемые A_2 и B_2 в формуле (4) $n/2$-битные, то становится возможным рекурсивное применение данной формулы. Соответствующий алгоритм вычислений представлен далее. Для сокращенного обозначения этого метода будем использовать аббревиатуру AKML (*Additive Karatsuba's MuLtiplication method*).

Алгоритм 3. Умножение длинных целых чисел по методу AKML

AKML(P, A, B, s, n_0): P, A, B — адреса произведения и операндов соответственно; s — разрядность операндов; n_0 — порог рекурсии.

ВХОД: два массива A и B с операндами размером s элементов каждый, $A = (A[s-1], A[s-2], ..., A[0])$, $B = (B[s-1], B[s-2], ..., B[0])$, где $A[i]$, $B[i]$ принимают значения от 0 до $p - 1$; $p = 2^w$; $0 \leqslant i \leqslant s - 1$; порог рекурсии n_0; массив P из $6s + 2\log_2(n/n_0) - 4s/2^{\log_2(n/n_0)} + 1$ элементов.

ВЫХОД: массив P из $2s$ элементов, содержащий результаты операции умножения $P = AB = (P[2s-1], P[2s-2], ..., P[0])$.

Шаг 1. Если $s \leqslant n_0/w$, то

Шаг 1.1. Вычислить AB , вызвав mul_base(P, A, B, s).

Шаг 1.2. Вернуть P.

Шаг 2. Рекурсивно вычислить A_0B_0, вызвав AMKL(P, A, B, $s/2$, n_0).

Шаг 3. Рекурсивно вычислить A_1B_1, вызвав AMKL($P + s$, $A + s/2$, $B + s/2$, $s/2$, n_0).

Шаг 4. Вычислить A_2 и c, вызвав $P[2s + s/2] =$ $=$ adc($P + 2s$, $A + s/2$, A, 0, $s/2$).

Шаг 5. Вычислить B_2 и d, вызвав $P[3s + 1] =$ $=$ adc($P + 2s + s/2 + 1$, $B + s/2$, B, 0, $s/2$).

Шаг 6. Рекурсивно вычислить A_2B_2, вызвав AMKL($P + 3s + 2$, $P + 2s$, $P + 2s + s/2 + 1$, $s/2$, n_0).

Шаг 7. Вычислить $P[4s + 2] = c$ and d, где and — логическое "И".

Шаг 8. Вычислить $A_2B_2 + cd2^n + cB_22^{n/2}$, вызвав $P[4s + 2] = P[4s + 2]$ + add_if_flag($P + 3s + s/2 + 2$, $P + 3s + s/2 + 2$, $P + 2s + s/2 + 1$, $P[2s + s/2]$, $s/2$).

Шаг 9. Вычислить $M = A_2B_2 + cd2^n + cB_22^{n/2} + dA_22^{n/2}$, вызвав $P[4s + 2] = P[4s + 2]$ + add_if_flag($P +$ $+ 3s + s/2 + 2$, $P + 3s + s/2 + 2$, $P + 2s$, $P[3s + 1]$, $s/2$).

Шаг 10. Вычислить $A_0B_0 + A_1B_1$, вызвав $P[4s + 2] =$ $=$ adc($P + 3s$, P, $P + 2s$, 0, s).

Шаг 11. Вычислить $M - (A_0B_0 + A_1B_1)$, вызвав sub($P + 3s + 2$, $P + 3s + 2$, $P + 2s$, 0, $s + 1$).

Шаг 12. Вычислить $A_0B_0 + A_1B_12^n + (M - (A_0B_0 +$ $+ A_1B_1))2^{n/2}$, вызвав $P[s + s/2] = P[s + s/2] +$ $+$ adc($P + s/2$, $P + s/2$, $P + 3s + 2$, 0, s).

Шаг 13. Вернуть P.

На верхнем уровне рекурсии задействуется $2s +$ $+ (4s/2 + 2) + 1$ элементов массива P (разрядностью w бит каждый). Младшие $2s$ элементов массива P предназначены для хранения результатов операции умножения, а оставшиеся $(4s/2 + 2) + 1$ элементов — для промежуточных вычислений (рис. 2).

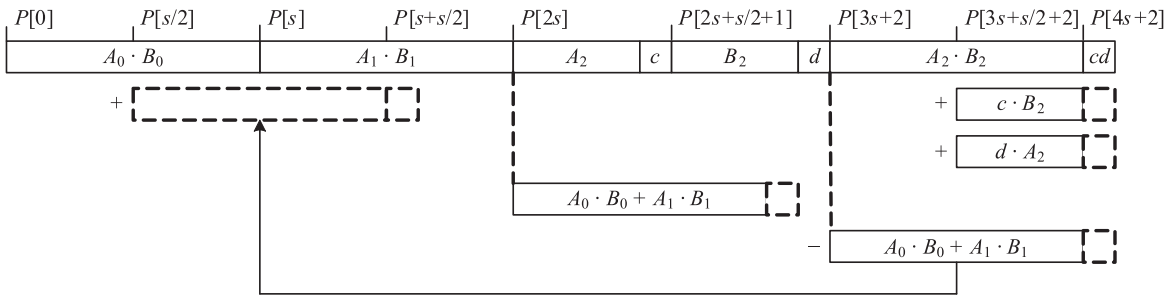


Рис. 2. Выделение памяти для рекурсивного алгоритма AKML

На каждом уровне рекурсии произведения A_0B_0 и A_1B_1 занимают по s элементов каждое и записываются с нулевым смещением относительно текущего базового адреса массива P. Компоненты $A_1 + A_0$ и $B_0 + B_1$ занимают по $s/2 + 1$ элементов каждый и записываются со смещениями $2s$ и $2s + s/2 + 1$ соответственно, как показано на рис. 2. Далее со смещением $3s + 2$ записывается A_2B_2, к которому последовательно прибавляются $cB_2 2^{n/2}$ и $dA_2 2^{n/2}$. Значение $A_0B_0 + A_1B_1$ предлагается записывать в массив P со смещением $2s$, затирая при этом A_2 и B_2. Следующий по вложенности уровень рекурсии задействует дополнительно $4s/4 + 2$ элементов P, при этом вычисленные на верхнем уровне значения $A_1 + A_0$ и $B_0 + B_1$ становятся исходным данными для рекурсивного вычисления A_2B_2. Таким образом, на каждом уровне рекурсии задействуется дополнительно $4s/2^i + 2$ элементов массива P для хранения результатов промежуточных вычислений (i — текущий уровень рекурсии). Для корректной работы алгоритма суммарно требуется выделить массив P из $2s + \sum_{i=1}^{\log_2 n/n_0} (4s/2^i + 2) + 1 =$

$= 6s + 2\log_2(n/n_0) - 4s/2^{\log_2(n/n_0)} + 1$ элементов.

При практической реализации рассмотренных алгоритмов умножения для увеличения скорости вычислений предлагается синтезировать линейную программу, построенную в виде последовательности элементарных операций (присваивание, умножение, сложение машинных слов и т. д.), без использования циклов, ветвлений и рекурсии [7]. Синтез линейных программ умножения может выполняться непосредственно по алгоритмам 2 и 3, поскольку в них заранее известен объем требуемой памяти, а схема выделения памяти разработана с учетом использования рекурсии при расчете значений промежуточных вычислений.

Для линейной программы декомпозиционная схема вычислений может строиться в автоматическом режиме. В процессе синтеза такой программы вместо непосредственных вычислений по пунктам алгоритма требуется выполнять последовательную запись соответствующих элементарных операций над текущими элементами массивов A, B и P по предложенным алгоритмам. Результатом такой записи будет линейная программа, которая для некоторых размеров разрядностей входных операндов будет работать быстрее итеративного или рекурсивного аналога, так как в ней отсутствуют накладные расходы на организацию циклов и рекурсии.

Эффективность реализации методов

Для определения критерия перехода к методу сдвигов и сложений в комбинированном методе Карацубы на тестовом стенде (Intel Core i7-2600 3,4 ГГц, Linux 3.0 64-bit, gcc 4.5.4 -O2 -m64) при фиксированных значениях параметров n и w проводили измерения среднего числа тактов процессора при различных значениях параметра порога остановки рекурсии n_0. Анализируя характер зависимости скорости метода от параметра n_0 можно определить критерий максимальной эффективности данного метода.

В результате экспериментов выяснилось, что оптимальным значением n_0 является значение, равное $16w$. Так, при $w = 32$, $n = 4096$, $n_0 = 512$ алгоритм SKML показывает наилучшие результаты по скорости вычислений (рис. 3). Аналогичная зависимость наблюдалась и для $w = 64$ (рис. 4).

Как следует из данных рис. 3 и рис. 4, при достижении порога остановки рекурсии значения $n_0 = 16w$, число процессорных инструкций минимальное для данной реализации. При дальнейшем увеличении глубины рекурсии заметен значительный рост их числа. Этот факт свидетельствует о том, что рекурсивная имплементация формулы Карацубы до максимальной глубины рекурсии (когда базой рекурсии является умножение двух процессорных слов) малоэффективна для рассматриваемого диапазона значений операндов.

Для оценки эффективности различных вариантов реализации рассматриваемых методов проводили измерения среднего числа тактов процессора при выполнении подпрограмм умножения больших целых чисел различной разрядности (n от 128 до 8192 бит) для

Рис. 3. Среднее число тактов процессора при вычислении произведения целых чисел разрядности 4096 бит при $w = 32$ бит для различных значений порога остановки рекурсии n_0 по алгоритму 2 (SKML)

Рис. 4. Среднее число тактов процессора при вычислении произведения целых чисел разрядности 8192 бит при w = 64 бит для различных значений порога остановки рекурсии n_0 по алгоритму 2 (SKML)

w = 32 (базовый тип представления данных uint32_t) и w = 64 (базовый тип представления данных uint64_t). Для метода сдвигов и сложений и метода Карацубы (алгоритмы SKML и AKML соответственно) были созданы как линейные версии программ (без использования циклов и ветвлений), так и их итеративные (метод сдвигов и сложений) и рекурсивные (метод Карацубы) аналоги. Результаты экспериментов по определению критериев применимости методов представлены на рис. 5 и рис. 6.

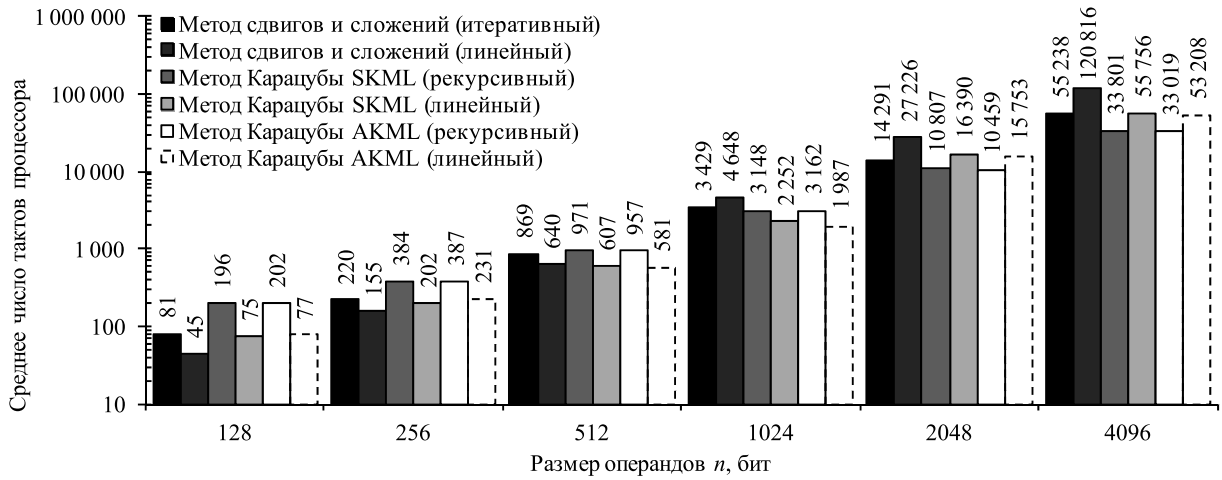


Рис. 5. Среднее число тактов процессора при вычислении произведения целых чисел различной разрядности для w = 32 бит

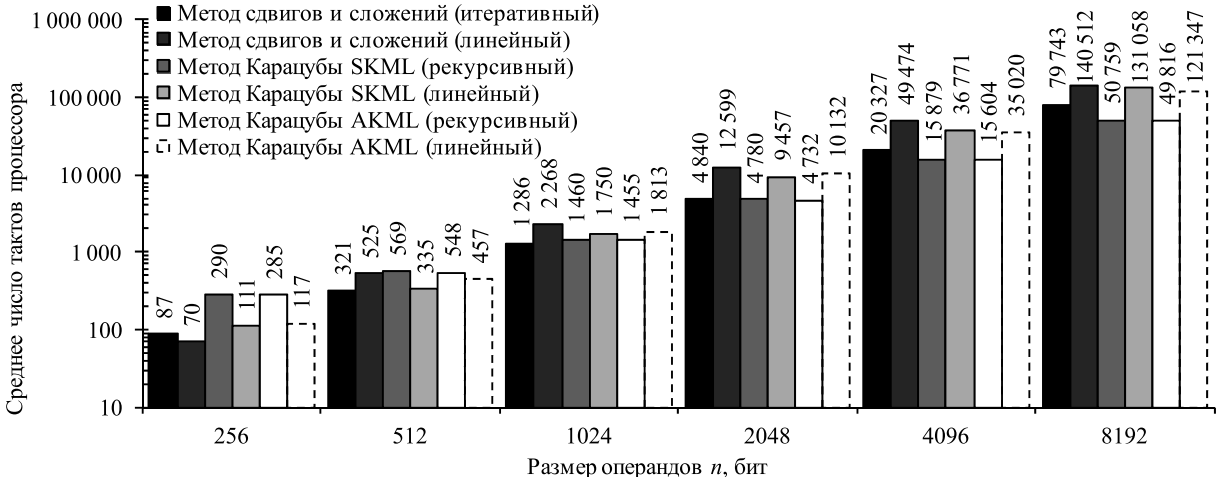


Рис. 6. Среднее число тактов процессора при вычислении произведения целых чисел различной разрядности для w = 64 бит

Таблица 1

Размер объектного кода линейных программ умножения, Кбайт

Метод	Размер операндов n, бит						
	128	256	512	1024	2048	4096	8192
w = 32 бит							
Сдвигов и сложений	0,5	2,0	8,0	35,4	163,9	691,2	—
SKML	0,7	2,1	7,3	28,1	100,3	333,2	—
AKML	0,8	2,2	7,2	26,9	97,3	319,7	—
w = 64 бит							
Сдвигов и сложений	—	0,7	4,4	15,7	75,7	291,0	789,4
SKML	—	1,1	3,5	15,8	59,0	220,9	723,5
AKML	—	1,2	4,2	16,1	62,9	210,7	669,2

Для оценки размера синтезированных линейных программ умножения в зависимости от разрядности операндов проводили измерения размера объектного кода. Соответствующие результаты представлены в табл. 1.

Для определения эффективности предложенных методов по сравнению с известными программными библиотеками длинной арифметики проводились измерения среднего числа тактов процессора при вычислении функции mpz_mul() библиотеки GMP версии 6.1.1. Данная функция вычисляет произведение двух длинных целых чисел. Библиотека GMP была скомпилирована для 64-разрядной операционной системы Linux (Intel Core i7-2600 3,4 ГГц, Linux 3.0 64-bit, gcc 4.5.4 -O2 -m64) без использования ассемблерных вставок (поскольку при реализации предложенных алгоритмов ассемблерные вставки также не использовались). Для этого при сборке использовался ключ --disable-assembly.

Дополнительно исследовали эффективность вычисления произведения двух длинных чисел при использовании популярной (C++)-библиотеки шаблонов Boost версии 1.62.0. В данной библиотеке имеется пакет Multiprecision, реализующий базовые операции длинной арифметики (использовался встроенный в библиотеку тип данных cpp_int).

Результаты этих измерении представлены в табл. 2.

Производительность предложенных алгоритмов и библиотек оценивали по методике [12], рекомендованной компанией Intel. В современных процессорах Intel имеется встроенный 64-битный счетчик тактов, который обнуляется по сбросу процессора и увеличивается на 1 на каждом такте. Текущие значения счетчика можно получить, используя ассемблерную инструкцию rdtsc (Read Time Stamp Counter). Если между последовательными вызовами rdtsc поместить

Таблица 2

Эффективность выполнения операции умножения в библиотеках GMP и Boost (среднее число тактов процессора)

Библиотека	Размер операндов n, бит					
	256	512	1024	2048	4096	8192
GMP 6.1.1	303	1024	3835	11 974	37 078	118 759
Boost 1.62.0	398	1521	5870	23 891	92 205	364 847

пользовательский код, то можно оценить число тактов, используемых процессором для его выполнения.

В процессе измерений для чистоты эксперимента в процессоре были отключены технологии Hyper-Threading и Turbo Boost. Непосредственно перед первым считыванием значения счетчика тактов программный код записывался в подсистему кеш-памяти путем многократного его выполнения (10^7 вызовов). Далее измеряли среднее число тактов процессора для 10^8 вызовов соответствующих функций умножения длинных целых чисел.

По результатам анализа экспериментальных данных (см. рис. 5 и рис. 6) можно сделать вывод, что при небольших размерах операндов наиболее эффективной является линейная реализация метода сдвигов и сложений (для w = 32). Начиная с n = 512 бит более эффективной становится линейная реализация метода Карацубы. При этом алгоритмы SKML и AKML показывают примерно одинаковую производительность вычислений. При n = 2048 лидирует рекурсивная реализация метода Карацубы, поскольку размер линейной программы (см. табл. 1) при этом превышает размер кэш-памяти L1 процессора (32 Кбайт для указанного процессора), и возникают дополнительные накладные расходы на его обновление.

Интересен тот факт, что линейная реализация метода Карацубы остается эффективнее своей рекурсивной реализации вплоть до n = 1024 бит включительно. При этом линейная реализация метода сдвигов и сложений уже менее эффективна, чем итеративная реализация при n = 1024 бит. Этот факт объясняется соотношением размера программы и кэш-памяти L1 процессора. Аналогичные выводы можно сделать и в случае, когда w = 64 бит.

По результатам анализа экспериментальных данных, представленных в табл. 2, можно сделать вывод, что реализация предложенных алгоритмов эффективнее по скорости вычислений по сравнению с аналогичными вычислениями с использованием библиотек GMP и Boost (для рассматриваемой разрядности входных данных). Например, для 512-битных чисел умножение по линейному методу SKML в 3 раза быстрее вычислений с использованием портируемой (без ассемблерных вставок) версии GMP, и в 4,5 раза быстрее вычислений с применением библиотеки Boost.

Дальнейшим направлением исследований в области эффективной реализации алгоритмов умножения

больших целых чисел может служить анализ рассмотренных алгоритмов с учетом применения техники отложенного переноса [13, 14]. В работе [13] представлена схема организации вычислений по модифицированному методу сдвигов и сложений, позволяющая сократить число дополнительных операций сложения, возникающих в процессе вычислений переносов. Недостатком этого решения является необходимость использования ассемблерной инструкции сложения двух машинных слов с учетом флага переноса (инструкция ADC). Стандартными средствами языка Си (без использования ассемблерных вставок) такую операцию реализовать трудно. Дальнейшее развитие этой идеи описано в работе [14]. Суть метода в том, что длинные числа записываются по укороченному основанию $B = 2^W$, где $W < w$. Оставшиеся $(w - W)$ бит машинного слова используются для накопления переносов в основном цикле программы (без использования дополнительных операций сложения). Недостатком такого метода является несколько увеличенный объем памяти, необходимой для хранения операндов и результатов вычислений.

Отсутствие необходимости обрабатывать на каждом шаге вычислений возникающие переносы позволяет естественным образом распараллеливать вычисления. Например, появляется потенциальная возможность увеличения скорости вычислений с помощью векторных инструкций (расширения SSE и AVX для Intel и NEON для процессоров ARM). Такие расширения позволяют распараллеливать однотипные операции, однако они сильно ограничивают переносимость кода на различные вычислительные средства.

Существуют непозиционные системы счисления (например, система остаточных классов), обладающие естественным параллелизмом как на уровне представления данных, так и на вычислительном уровне [15]. Например, в системе остаточных классов числа представляются в виде остатков от деления на набор модулей (взаимно простых). В таких системах практически отсутствуют накладные расходы на корректную обработку межразрядных переносов. Вычисления суммы, разности или произведения чисел выполняются независимо по нескольким основаниям (модулям). Это дает широкие возможности для распараллеливания вычислений. Более детальный анализ эффективности применения системы остаточных классов для умножения длинных целых чисел требует дополнительного исследования, выходящего за рамки данной статьи.

Заключение

Предложенные варианты алгоритмов умножения могут быть эффективно использованы при создании программных библиотек компьютерной алгебры. Проведенные эксперименты позволили определить критерии применимости рассматриваемых методов в зависимости от размера операндов. Показано влияние порога остановки прямого хода рекурсии в комбинированном алгоритме на скорость вычислений. Определены оптимальные значения этого параметра. Предложена эффективная схема организации вычислений без использования дополнительной памяти для хранения знаков промежуточных вычислений (алгоритм SKML). Разработаны схемы организации памяти, поддерживающие как рекурсивную, так и линейную реализацию алгоритмов SKML и AKML. Получены оценки объема памяти программ и данных, которые на практике необходимы для эффективного использования предложенных алгоритмов. Представлено сравнение эффективности вычислений по предложенным алгоритмам с аналогичными вычислениями с применением известных программных библиотек GMP и Boost.

Список литературы

1. **Кнут Д**. Искусство программирования. Т. 2, Получисленные алгоритмы. М.: Вильямс, 2004. 832 с.
2. **Brent R., Zimmermann P.** Modern computer arithmetic. Cambridge University Press, New York, 2010. 236 p.
3. **Bryant R., O'Hallaron D.** Computer system: a programmers perspective. Addison-Wesley, USA, 2010. 1080 p.
4. **Granlund T.** The GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library. version 6.1.1 (2016). URL: http://gmplib.org.
5. **Gladman B., Hart W., Moxham J.** et al. MPIR: Multiple Precision Integers and Rationals, A fork of the GNU MP package, version 2.7.2 (2015). URL: http://mpir.org.
6. **Hutter M., Schwabe P.** NaCl on 8-bit AVR Microcontrollers // Progress in Cryptology AFRICACRYPT 2013. Springer-Verlag, 2013. P. 156—172.
7. **Фролов А. Б., Винников А. М.** О машинном синтезе некоторых линейных программ // Программная инженерия. 2011. № 6. С. 24—30.
8. **Карацуба А. А., Офман Ю. П.** Умножение многозначных чисел на автоматах // ДАН СССР. 1962. Т. 145, № 2. С. 293—294.
9. **Куляс М. Е.** О программной реализации алгоритма умножения Карацубы // Вестник МЭИ. 2016. № 1. С. 26—32.
10. **Гашков С. Б.** Занимательная компьютерная арифметика. Быстрые алгоритмы операций с числами и многочленами. М.: УРСС, Либроком, 2012. 224 с.
11. **Карацуба А. А.** Сложность вычислений // Труды математического института имени В. А. Стеклова, 1995. Т. 211. С. 186—202.
12. **Paoloni G.** How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures, 2014. URL: http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf.
13. **Comba P.** Exponentiation cryptosystems on the IBM PC // IBM Systems Journal. 1990. N 29. P. 526—538.
14. **Kovtun V. Yu., Okhrimenko A. O.** Integer squaring algorithm with delayed carry mechanism // Безпека інформації. 2013. Vol. 3, N 19. P. 188—192.
15. **Акушский И. Я., Юдицкий Д. И.** Машинная арифметика в остаточных классах. М.: Сов. Радио, 1968. 439 с.

Synthesis of Computer Programs for Long Integers Multiplication

M. E. Kulyas, kuliasmy@mpei.ru, National Research University "Moscow Power Engineering Institute", 111250, Russian Federation

Corresponding author:

Kulyas Mikhail E., PhD Student, National Research University "Moscow Power Engineering Institute", 111250, Russian Federation,
E-mail: kuliasmy@mpei.ru

Received on September 25, 2016
Accepted on November 11, 2016

This article provides two variants of hybrid recursive algorithms of long integers multiplication, combining the asymptotically fast Karatsuba algorithm with shift-and-add algorithm at low levels of recursion. These algorithms can find a good use when creating computer algebra system libraries. The article introduces both the recursive and the sequential (linear) implementation of the algorithms in question. The performance of the experiments made it possible to define the applicability of the algorithms depending on the operands size. The article demonstrates the impact of forward recursion stop threshold in a hybrid algorithm on the computing speed. The optimal values of this parameter are determined. The research proposes an efficient computational scheme which doesn't use additional memory for signs of intermediate computations results (SKML algorithm). The author develops the memory organization schemes supporting both the recursive and the sequential implementation of the SKML and AKML algorithms. The research evaluates the programs memory size and the data memory size needed for efficient use of the algorithms in question. The author compares the computing efficiency of the algorithms in question against the well-known GMP and Boost libraries.

Keywords: *long integers multiplication, Karatsuba algorithm, sequential program, recursion, computational complexity*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project № 14-01-00671-A.*

For citation:

Kulyas M. E. Synthesis of Computer Programs for Long Integers Multiplication, *Programmnaya Ingeneria*, 2017, vol. 8, no. 2, pp. 66—75.

DOI: 10.17587/prin.8.66-75

References

1. **Knuth D.** *Iskusstvo programmirovaniia, T. 2, Poluchislennie algoritmy* (The art of computer programming, vol. 2, Seminumerical algorithms), Moscow, Vil'iams, 2004, 832 p. (in Russian).

2. **Brent R., Zimmermann P.** *Modern computer arithmetic*, Cambridge University Press, New York, 2010, 236 p.

3. **Bryant R., O'Hallaron D.** *Computer system: a programmers perspective*, Addison-Wesley, USA, 2010, 1080 p.

4. **Granlund T.** The GMP development team: GNU MP: The GNU Multiple Precision Arithmetic Library, version 6.1.1 (2016), available at: http://gmplib.org.

5. **Gladman B., Hart W., Moxham J.** et al. MPIR: Multiple Precision Integers and Rationals, A fork of the GNU MP package, version 2.7.2 (2015), available at: http://mpir.org.

6. **Hutter M., Schwabe P.** NaCl on 8-bit AVR Microcontrollers, *Progress in Cryptology AFRICACRYPT 2013*, Springer-Verlag, 2013, pp. 156—172.

7. **Frolov A. B., Vinnikov A. M.** O mashinnom sinteze nekotoryh linejnyh program (Automated synthesis of sequential programs), *Programmnaya Ingeneria*, 2011, no. 6, pp. 24—30 (in Russian).

8. **Karatsuba A. A., Ofman Yu. P.** Umnozhenie mnogoznachnyh chisel na avtomatah (Multiplication of multidigit numbers on automata), *Dokl. Akad. Nauk USSR*, 1962, vol. 145, no. 2, pp. 293—294 (in Russian).

9. **Kulyas M. E.** O programmnoj realizacii algoritma umnozhenia Karacuby (Software implementation of Karatsuba's multiplication algorithm), *Vestnik MEI*, 2016, no. 1, pp. 26—32 (in Russian).

10. **Gashkov S. B.** *Zanimatelnaya komputernaya arifmetika. Bystrye algoritmi operacij s chislami I mnogochlenami* (Computer arithmetic. Fast algorithms for operating on numbers and polynomials), Moscow, URSS, Librocom, 2012, 224 p. (in Russian).

11. **Karatsuba A. A.** Slojnost Vychislenij (The complexity of computations), *Proceedings of Steklov Institute of Mathematics*, 1995, vol. 211, pp. 186—202 (in Russian).

12. **Paoloni G.** How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures, 2010, available at: http://www.intel.com/content/dam/www/public/ us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf.

13. **Comba P.** Exponentiation cryptosystems on the IBM PC, *IBM Systems Journal*, 1990, no. 29, pp. 526—538.

14. **Kovtun V. Yu., Okhrimenko A. O.** Integer squaring algorithm with delayed carry mechanism, *Information security*, 2013, vol. 3, no. 19, pp. 188—192.

15. **Akushskij I. Ja., Judickij D. I.** *Mashinnaja arifmetika v ostatochnyh klassah* (Computer arithmetic with residue number system), Moscow, Sov. Radio, 1968, 440 p. (in Russian).

УДК 519.722 DOI: 10.17587/prin.8.76-87

А. М. Кукарцев, ст. преподаватель, e-mail: amkukarcev@yandex.ru,
А. А. Кузнецов, д-р физ.-мат. наук, проф., e-mail: kuznetsov@sibsau.ru, Сибирский государственный аэрокосмический университет имени академика М. Ф. Решетнева, г. Красноярск

Об эффективном алгоритме решения уравнения действия элемента группы Джевонса над булевыми функциями*

Действие группы Джевонса над булевыми функциями интранзитивно и разбивает все множество на орбиты мощности равной, в подавляющем большинстве случаев, порядку группы $2^n n!$, где n — число аргументов функции. Вычисление действующего элемента группы, связывающего две функции в одной орбите, является сложнорешаемой математической задачей. Поэтому действие группы Джевонса над булевыми функциями используется как криптографический примитив в алгоритмах шифрования, основывающихся на управляемых операциях, где ключом является действующий элемент группы. В настоящей работе предложен эффективный алгоритм вычисления действующего элемента группы, связывающего две функции за время меньшее, чем тривиальный алгоритм (т. е. полный перебор). Алгоритм использует частотные свойства действия группы Джевонса над булевыми функциями. Приведена эмпирическая оценка сложности предлагаемого алгоритма, основывающаяся на специально разработанном спектральном анализе булевых функций.

Ключевые слова: *действие группы на множестве, частотный анализ, группа Джевонса, булевы функции, уравнения действия группы на множестве*

Введение

Среди многих способов представления информации можно выделить два основных: комбинационный и функциональный. Способы первого вида рассматривают информационный объект (ИО) как комбинацию символов некоторого алфавита. Они и соответствующие им алгоритмы обработки информации начали развиваться с начала XX века. В конце XX века, наряду с комбинационными способами, появились функциональные. В них ИО ставится в соответствие множество значений некоторой функции. Именно такой способ представления лежит в основе алгоритмов сжатия графической информации JPEG [1]. Информационный объект отображается на некоторую функцию двух аргументов. Далее функция преобразуется и определяющие ее параметры (коэффициенты Фурье) округляются, кодируются, и их коды сжимаются комбинационными методами. Такой способ сжатия приводит к потерям информации во время преобразования.

Помимо указанного существуют способы инъективного отображения информации в функции. Для этого подходят булевы функции (далее — БФ). Всякий ИО из нулей и/или единиц произвольного размера можно инъективно отобразить на ИО длины, кратной степени двойки. При этом ИО указанной длины можно биективно поставить в соответствие некоторую БФ. Для такого отображения важно задать порядок следования аргументов БФ. Представление информации в виде БФ необходимо для дальнейшего ее исследования и разработки алгоритмов преобразования, например, соединения (сложения), разложения и т. д.

При представлении информации булевыми функциями появляется естественная эквивалентность ИО. По сути, ИО ставится в соответствие функциональным элементам [2], реализующим БФ. Поэтому отрицания и/или перестановки аргументов БФ являются нейтральным преобразованием ИО, так как не меняют связи между функциональными элементами. Булевы функции, полученные путем отрицаний и/или перестановок аргументов, будут иметь одинаковый вид конъюнктивной и дизъюнктивной нормальных форм [3].

Рассмотрим две задачи. Первая — выяснить эквиваленты ли две БФ относительно группы Джевонса. Вторая — решить уравнение действия элемента группы Джевонса над БФ относительно неизвестного

* Работа выполнена при поддержке гранта Президента РФ (проект МД-3952.2015.9).

Таблица 1

Оценка времени поиска решения тривиальным алгоритмом

n	1—13	14	15	16	17	18	19
Time(n), млн лет	≈0	0,000742	0,04452	2,8495	193,7679	13 951,286	1 060 297,774

действующего элемента. Указанные задачи поставлены в работе [4]. Важно также отметить, что количество возможных отрицаний и/или перестановок аргументов БФ конечно и равно $2^n n!$, где n — число аргументов БФ. Поэтому допустимо тривиальное решение обеих задач — перебор всех отрицаний и/или перестановок аргументов. Время проверки одного варианта зависит нелинейно от числа аргументов, так как нужно анализировать каждое значение БФ (всего этих значений 2^n). Пусть скорость обработки при поиске решения указанных задач составляет миллиард значений БФ в секунду. Тогда время проверки одного варианта составит $\tau(n) = 10^{-9}2^n$ с. Всего вариантов будет (согласно порядку группы Джевонса) $d_{trivial} = 2^n n!$, откуда полное время перебора всех вариантов составит $Time(n) = \tau(n) d_{trivial} = \tau(n) 2^n n!$.

Проанализируем возможность вычисления тривиального решения по табл. 1. Из определения *Time*(n) можно заключить, что каждое следующее значение времени больше предыдущего в $\dfrac{10^{-9}2^n 2^n n!}{10^{-9}2^{n-1}2^{n-1}(n-1)!} = 4n$ раз. Начиная с $n = 19$, время, необходимое на вычисление решения тривиальным способом, превышает возраст Вселенной [5]. Поэтому требуется эффективный алгоритм решения задач, т. е. такой, который находит решение быстрее тривиального [5].

Решение указанных задач тесно связано с понятием инварианта группы, действующей на множестве БФ [6]. Определение инварианта позволяет решить первую сформулированную задачу — выяснить, связаны ли две БФ отрицаниями и/или перестановками аргументов. С. Голомб предложил полный инвариант [7], но сложность его вычисления, так же как и тривиальное решение, носит экспоненциальный характер. Э. А. Якубайтисом также был предложен инвариант [8], но он не является полным. В результате обе сформулированные задачи являются сложнорешаемыми. В силу отсутствия решения кроме тривиального, преобразования информации, заданные отрицаниями и/или перестановками аргументов БФ, используются в качестве криптографического примитива для шифрования информации алгоритмами, основывающимися на управляемых операциях, в которых элемент группы Джевонса является ключом шифрования [9].

Цель настоящей статьи: предложить эффективный алгоритм решения уравнения действия элемента группы Джевонса над булевыми функциями относительно неизвестного действующего элемента.

Определения и обозначения. Формальная постановка задачи

Пусть n — целое неотрицательное число, $k = 2^n$, i, j, — целые неотрицательные индексы (счетчики), не превосходящие k.

Определим $E = \{0,1\}$ — множество, состоящее из двух элементов. Декартово произведение этого множества на себя определим как $E^n = \underbrace{E \times \cdots \times E}_{n}$. Элементом такого множества будет бинарный вектор (далее — БВ). Для его координат будем использовать числовую нотацию L2R (*left to right*), т. е. большие координаты находятся левее. Обозначим произвольный элемент множества как $x \in E^n$, $x = \{x_{n-1}, \ldots, x_0\}$. Для векторов — аргументов БФ также будем применять нотацию L2R.

Булева функция n аргументов $f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0)$ реализует отображение $E^n \to E$:

$$y_f = \{f(11\ldots11), f(11\ldots10), \ldots, f(00\ldots01), f(00\ldots00)\},$$
$$(y_f)_j = f(x_{n-1}, x_{n-2}, \ldots, x_1, x_0), j = \sum_{i=n-1}^{0} x_i 2^i. \quad (1)$$

Пусть БФ задана через таблицу истинности [3]. Если однозначно определить порядок следования аргументов, то каждой БФ можно поставить в соответствие бинарный вектор y_f (1) длины k (столбец значений). Все множество БФ n аргументов обозначим как $B(n)$.

Группу инвертирования переменных обозначим как $E_n = (E^n, \oplus)$ [6]. Группу перестановок переменных (симметрическую группу [10]) обозначим как S_n. Группу Джевонса обозначим как D_n [6]. Структурно она является внешним полупрямым произведением, т. е. $D_n = E_n \leftthreetimes S_n$ [11]. В силу специфики целевых объектов условимся, что симметрическая группа действует на множестве чисел $[0; n-1]$. Обозначим также нейтральные элементы групп E_n и S_n как e_n и e_s соответственно.

Пусть $f, g \in B(n)$. Обозначим через $z \in E_n$ отрицание аргументов и через $\pi \in S_n$ — перестановку аргументов. Совместно отрицание и/или перестановку аргументов обозначим как $(z\pi) \in D_n$, тогда основное уравнение действия элемента группы Джевонса на БФ относительно неизвестного $(z\pi)$ обозначим как

$$f^{(z\pi)} = g. \quad (2)$$

Важно разделять групповую операцию в симметрической группе и ее действие на БВ. Эмпирически

показано, что для задач действия группы Джевонса на БФ удобно задавать гомоморфизм внешнего полупрямого произведения группы Джевонса из симметрической группы в группу автоморфизмов группы инвертирования переменных через само действие элемента на БФ. Такое действие может быть задано не единственным способом. Эмпирически показано, что можно выделить два набора действий: типа А и типа Б. Оба типа действий выражаются друг через друга. При действии элемента группы Джевонса на БВ предпочтителен тип А, но при действии на БФ — тип Б [12, 13]. Поэтому невозможно выделить какой-то один тип действий как основной, а второй как сводимый к нему. В то же время разработчик может выбирать тип действия по своему усмотрению в зависимости от решаемой задачи.

Действием типа А элемента группы $\pi \in S_n$ на БВ $x \in E^n$ будем называть вычисление результата $x' \in E^n$ по следующему правилу: в верхней строке подстановки находятся "старые" индексы, а в нижней — "новые":

$$x'_{\pi(i)} = x_i. \tag{3.А}$$

Действием типа Б элемента группы $\pi \in S_n$ на БВ $x \in E^n$ будем называть вычисление результата $x' \in E^n$ по следующему правилу: в верхней строке подстановки находятся "новые" индексы, а в нижней — "старые":

$$x'_i = x_{\pi(i)}. \tag{3.Б}$$

Отсюда получим два представления группы Джевонса для $z_0, z_1 \in E_n$ и $\pi_0, \pi_1 \in S_n$:

$$(z_0\pi_0)(z_1\pi_1) = \left(z_0 z_1^{\pi_0^{-1}} \pi_0\pi_1\right), \tag{4.А}$$

$$(z_0\pi_0)(z_1\pi_1) = \left(z_0 z_1^{\pi_0^{-1}} \pi_1\pi_0\right). \tag{4.Б}$$

Зададим правила вычисления действий элементов группы Джевонса и их композиции на булев вектор-аргумент x и БФ f из работы [13] следующим образом:

$$x^{(z\pi)} = \left(x^z\right)^\pi, \tag{5.1}$$

$$\left(x^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = x^{(z_0\pi_0)(z_1\pi_1)}, \tag{5.2}$$

$$f^{(z\pi)} = f\left(x^{(z\pi)}\right) = f\left(\left(x^z\right)^\pi\right) = \left(f^\pi\right)^z, \tag{5.3}$$

$$\left(f^{(z_0\pi_0)}\right)^{(z_1\pi_1)} = f^{(z_1\pi_1)(z_0\pi_0)}. \tag{5.4}$$

Потребуются также доказанные утверждения из работы [14] о каноническом разложении элемента группы Джевонса.

Лемма 1 (О монотонном представлении подстановки). Пусть k есть число независимых циклов, включая циклы длины 1, нетривиальной подстановки π группы S_n степени n. Тогда она может быть единственным образом представлена как произведение из $n - k$ транспозиций вида

$$\pi = \left(0, \pi_0^{-1}(0)\right)\cdots\left(i_0, \pi_{i_0}^{-1}(i_0)\right)\cdots\left(i, \pi_i^{-1}(i)\right)\cdots$$
$$\ldots\left(i_1, \pi_{i_1}^{-1}(i_1)\right)\cdots\left(n-2, \pi_{n-2}^{-1}(n-2)\right),$$

причем $0 \leqslant \ldots < i_0 < \ldots < i < \ldots < i_1 \ldots < n - 1$. В произведение включаются только транспозиции, соответствующие точкам $i : \pi(i) \neq i$ и $i < \pi_i^{-1}(i)$. Промежуточные подстановки вычисляются рекурсивно как $\pi_{i+1} = \left(i, \pi_i^{-1}(i)\right)\pi_i$, при этом $\pi_0 = \pi$.

Пусть $b_{n-1}, \ldots, b_i, \ldots, b_0, z \in E_n$, причем $b_i = \{0, \ldots, 0, \ldots, z_i, \ldots, 0, \ldots, 0\}$, т. е. содержит на позиции i значение координаты i БВ z, а на остальных — нули.

Теорема 1 (О каноническом представлении элемента группы Джевонса). *Любой элемент группы Джевонса $(z\pi) \in D_n$ представим единственным образом в виде произведения*

$$\left(b_{n-1}(n-1, j_{n-1})\right)\cdots\left(b_{i_1}(i_1, j_{i_1})\right)\cdots\left(b_i(i, j_i)\right)\cdots$$
$$\cdots\left(b_{i_0}(i_0, j_{i_0})\right)\cdots\left(b_0(0, j_0)\right), \tag{6}$$

где $0 \leqslant \ldots < i_0 < \ldots < i < \ldots < i_1 \ldots < n - 1$. Элементы симметрической группы $(i, j_i) : i \leqslant j_i$ имеют порядок не более 2, и в случае транспозиции соответствуют разложению по лемме 1 (для типа А будет π^{-1} и для типа Б — π).

В качестве примера рассмотрим каноническое представление элемента

$$(z\pi) \in D_8 : (z\pi) = \left(\{0110\ 0011\}(7, 4, 3)(6, 2, 1, 5)\right).$$

Монотонное представление подстановки уже найдено в работе [14] и равно $\pi = (1, 2)(2, 6)(3, 4)(4, 7)(5, 6)$, и соответствует каноническому разложению для типа Б, т. е. непосредственно получаем

$$(z\pi) = (e_E e_S)\left(\{0100\ 0000\}\, e_S\right)\left(\{0010\ 0000\}(5, 6)\right)\times$$
$$\times\left(e_E(4, 7)\right)\left(e_E(3, 4)\right)\left(e_E(2, 6)\right)\left(\{0000\ 0010\}(1, 2)\right)\times$$
$$\times\left(\{0000\ 0001\}\, e_S\right).$$

Для типа А сначала требуется найти монотонное представление π^{-1}. По лемме 1 получим

$$\pi^{-1} = (7, 3, 4)(6, 5, 1, 2) = (1, 5)(2, 5)(3, 7)(4, 7)(5, 6).$$

Тогда по теореме 1 получим

$$(z\pi) = (e_E e_S)\left(\{0100\ 0000\}\, e_S\right)\left(\{0010\ 0000\}(5, 6)\right)\times$$
$$\times\left(e_E(4, 7)\right)\left(e_E(3, 7)\right)\left(e_E(2, 5)\right)\left(\{0000\ 0010\}(1, 5)\right)\times$$
$$\times\left(\{0000\ 0001\}\, e_S\right).$$

Важно подчеркнуть следующее. Монотонное разложение подстановки π в общем случае не сводимо

перестановкой множителей задом наперед к монотонному разложению π^{-1}. Применительно к примеру для π^{-1} имеем: как обращение π верно

$$\pi^{-1} = \big((1,\,2)(2,\,6)(3,\,4)(4,\,7)(5,\,6)\big)^{-1} = \\ = (5,\,6)(4,\,7)(3,\,4)(2,\,6)(1,2);$$

как монотонное представление верно $\pi^{-1} = (1,\,5)(2,\,5)(3,\,7)(4,\,7)(5,\,6)$. Вычисляя оба произведения, очевидно, получим π^{-1}, но при этом первое не является монотонным представлением, а второе — является. В заключение отметим, что монотонные представления подстановок и канонические представления элементов группы Джевонса в общем случае **не одинаковы** для типа А и типа Б.

Предлагаемое решение основывается на частотных свойствах действия группы Джевонса на булевы функции. Для использования этих свойств потребуется ряд определений из работы [14].

Определение 1. Алфавит A_i — множество, построенное как декартово произведение E^{2^i}.

Булевый вектор y_f может быть разбит на символы в любом из алфавитов A_i, где i пробегает все значения [0; n]. При этом разбиение начинается с первого значения y_f и символы не перекрываются. Длина y_f (число символов) в каждом из алфавитов рассчитывается как $l_i = 2^{n-i}$.

Определение 2. Символ алфавита — элемент алфавита A_i, встречаемый в векторе y_f.

Определение 3. Частота символа — число случаев встречи заданного символа из алфавита A_i в векторе y_f.

Определение 4. Частотное распределение y_f (спектр) над алфавитом A_i — отношение $Q_i\left(y_f\right) \subset A_i \times [0;\,k]$, т. е. множество пар символ—частота.

Определение 5. Спектральное распределение y_f над алфавитом A_i — отношение $R_i\left(y_f\right) \subset [0;k] \times [0;k]$ (производное множество от $Q_i(y_f)$), элементы которого показывают, как часто повторяются частоты в y_f. В элементе отношения сначала указывается частота из $Q_i(y_f)$, а затем число различных символов, обладающих такой частотой в y_f.

Далее для удобства будем подразумевать эквивалентность БФ и БВ, т. е. отношение $Q_i(y_f)$ можно обозначить как $Q_i(y_f)$. Потребуются также теоремы об инвариантности частотных спектров при действии на БФ группами E_n и S_n из работы [14].

Определение 6. Определим множество БВ c_{n-1}, ..., c_i, ..., $c_0 \in E^n$, причем каждый c_i имеет на позиции i значение 1, а на остальных — 0, т. е. все множество c_i представляет собой порождающее множество группы E_n.

Определение 7. Подгруппой инерции булевой функции f в группе Джевонса будем называть такое подмножество ее элементов, действие которых на функцию тривиально: $J_{D_n}(f) = \left\{(z\pi) \in D_n \middle| f^{(z\pi)} = f\right\}$.

Теорема 2 (Об инвариантности частотных спектров при действии E_n). *Если элемент $c_i \in E_n$ действует на БФ $f \in B(n)$, то частотные распределения БВ y_f инвариантны относительно этого действия для алфавитов $A_{i'} : i' \leqslant i$, $i \in [0; n-1]$, а спектральные распределения y_f инвариантны относительно этого же действия для алфавитов A_i.*

Теорема 3 (Об инвариантности частотных спектров при действии S_n). *Если транспозиция $(i,\,j) \in S_n$, $i, j \in [0;\,n-1] : i < j$ действует на БФ $f \in B(n)$, то частотные распределения БВ y_f инвариантны относительно этого действия для алфавитов $A_{i'} : i' \leqslant i$, а спектральные распределения y_f инвариантны относительно этого же действия для алфавитов $A_{i'}$ и $A_{j'} : j' > j$.*

Основное решение уравнения действия

Решение уравнения действия (2) строится на следующем принципе. В работе [13] показано, что отрицания и/или перестановки аргументов выполняют перестановку строк таблицы истинности (1). Как следствие, не изменяется вес БФ. Число наборов аргументов, на которых БФ принимает единичное и нулевое значение, фактически является спектром исходной функции в алфавите A_0. Откуда можно получить условие: необходимым условием решения уравнения (2) является равенство частотных спектров $Q_0(f)$ и $Q_0(g)$.

Возникает закономерный вопрос: "сколько и каких необходимых условий нужно набрать, чтобы их совокупность превратилась в достаточные условия существования решения уравнения (2)?". Как будет доказано в теореме 4, такой совокупностью условий является сохранение частотных спектров во всех допустимых алфавитах согласно теоремам 2 и 3 при последовательном действии (от $i = 0$ до $i = n - 1$) всеми множителями канонического представления по формуле (6) потенциального решения уравнения (2), затрагивающими конкретный аргумент БФ x_i.

Например, пусть при $n = 4$ для уравнения $f^{(z\pi)}(x_3,\,x_2,\,x_1,\,x_0) = g(x_3,\,x_2,\,x_1,\,x_0)$ верно $Q_0(f) = Q_0(g)$ и пусть решение есть, и оно единственно. Тогда проанализируем все возможные $(z\pi)$, которые затрагивают аргумент x_0. Согласно формуле (6) в $(z\pi)$ из уравнения (2) может присутствовать только один из множителей вида $\big(b_0(0,\,j_0)\big) : b_0 \in \{e_E,\,c_0\}, 0 \leqslant j_0 < n$, откуда имеем: $(e_E e_S)$, $\big(\{0001\}e_S\big)$, $\big(e_E(3,\,0)\big)$, $\big(e_E(2,\,0)\big)$, $\big(e_E(1,\,0)\big)$, $\big(\{0001\}(3,\,0)\big)$, $\big(\{0001\}(2,\,0)\big)$, $\big(\{0001\}(1,\,0)\big)$. Потенциально каждый из этих множителей может входить в решение (2). Согласно формулам (5.4) и (6) можно вычислить действия этих восьми множителей на f. Согласно теоремам 2 и 3 результаты таких действий дадут не более восьми различных спектров. Если множитель действительно входит в решение уравнения (2), то согласно теоремам 2 и 3 спектры результатов соответственных действий совпадают с $Q_1(g)$. В итоге можно отбраковать некоторые множители, действующие на x_0,

и вместе с ними множество заведомо ложных решений уравнения (2), которые их включают. Последовательная отбраковка множителей позволяет построить эффективный алгоритм быстрого решения уравнения (2). Для его описания потребуется ряд определений.

Определение 8. Гипотезой для значения $i : 0 \leqslant i < n$ будем называть элемент $h \in D_n$, равный произведению $i + 1$ множителей в порядке от меньших значений i' к большим канонического представления (6) решения ($z\pi$) уравнения (2), для которого верно $Q_{i+1}(f^h) = Q_{i+1}(g)$, где $i' : 0 \leqslant i' \leqslant i$ — индекс множителя.

Определение 9. Промежуточной функцией f^h, $h \in D_n$ будем называть БФ, полученную из исходной f под действием гипотезы h.

Алгоритм 1 (О вычислении нулевого действия).

Вход: число аргументов БФ n и пара БФ $f, g \in B(n)$, которые представлены в виде БВ по формуле (1), тип преобразования БВ А или Б.

Выход: множество $H_n \subseteq D_n$ всех решений уравнения (2).

Для нахождения всех решений уравнения (2) нужно выполнить следующие шаги.

Шаг 0. Проверить необходимое условие $Q_0(f) = Q_0(h)$. В случае неудачи завершить поиск решения с результатом "нет решений". В случае успеха принять, что исходное множество гипотез H_0 содержит только единичный элемент группы Джевонса $(e_E e_S)$, задать индекс первого анализируемого аргумента БФ $i = 0$ и перейти к шагу 1.

Шаг 1. Начало итерации i алгоритма. Вычислить множество элементов группы Джевонса H'_{i+1} из множества гипотез H_i (по формуле (4.А) или (4.Б) в зависимости от типа преобразования БВ) как (умножить слева на H_i)

$$H'_{i+1} = \left[\bigcup_{j_i=i}^{j_i<n} \left(e_E(i,\, j_i)\right) H_i\right] \cup \left[\bigcup_{j_i=i}^{j_i<n} \left(c_i(i,\, j_i)\right) H_i\right].$$

Мощности множеств будут связаны равенством $|H'_{i+1}| = 2(n-i)|H_i|$, т. е. происходит увеличение мощности H_i в число допустимых множителей канонического разложения (6) для конкретного i. H_i более не нужно. Перейти к шагу 2.

Шаг 2. Сравнить каждый спектр промежуточной функции $Q_{i+1}(f^h)$ для всех $h \in H'_{i+1}$ со спектром $Q_{i+1}(g)$. Определить новое множество гипотез H_{i+1}, в которое входят все те элементы из H'_{i+1}, для которых $Q_{i+1}(f^h) = Q_{i+1}(g)$. Конец итерации i алгоритма. Выполнить инкремент $i = i + 1$. H'_{i+1} более не нужно. Перейти к шагу 3.

Шаг 3. Если множество H_i пусто, то заключить "нет решений" и завершить алгоритм. Иначе сравнить i и n. Если $i = n$, то алгоритм заканчивает работу и получено множество H_n, содержащее все решения уравнения (2). Иначе ($i < n$) нужно перейти к шагу 1 алгоритма. Входными данными для шага 1 будут множество H_i и значение i.

Корректность алгоритма 1 доказывается следующей теоремой 4.

Теорема 4 (О вычислении нулевого действия). *Множество H_n совпадает с множеством всех решений уравнения $f^{(z\pi)} = g$ и может быть вычислено за n шагов последовательно для $0 \leqslant i < n$ как*

$$H'_{i+1} = \left[\bigcup_{j_i=i}^{j_i<n} \left(e_E(i,\, j_i)\right) H_i\right] \cup \left[\bigcup_{j_i=i}^{j_i<n} \left(c_i(i,\, j_i)\right) H_i\right],$$

начиная с $H_0 = \{(e_E, e_S)\}$ *и*

$$H_{i+1} = \left\{h \in H'_{i+1} \mid Q_{i+1}\left(f^h\right) = Q_{i+1}(g)\right\}.$$

Доказательство. Для доказательства теоремы сначала покажем, что пропуск шага 2 (т. е. замена множества $H_{i+1} = \left\{h \in H'_{i+1} \mid Q_{i+1}\left(f^h\right) = Q_{i+1}(g)\right\}$ на множество $H_{i+1} = H'_{i+1}$) для каждой итерации алгоритма 1 $0 \leqslant i < n$ приведет к тому, что множество H_n совпадет со всей группой Джевонса. Это следует из того, что вычисление

$$H'_{i+1} = \left[\bigcup_{j_i=i}^{j_i<n} \left(e_E(i,\, j_i)\right) H_i\right] \cup \left[\bigcup_{j_i=i}^{j_i<n} \left(c_i(i,\, j_i)\right) H_i\right]$$

фактически является перебором канонических представлений всех элементов группы Джевонса в порядке вычисления произведения, обратном (6). Поэтому либо H_n содержит все решения уравнения (2), либо решений нет вовсе. Далее вернем условия теоремы и рассмотрим множество $H_{i+1} = \left\{h \in H'_{i+1} \mid Q_{i+1}\left(f^h\right) = Q_{i+1}(g)\right\}$.

Опираясь на теоремы об инвариантности частотных спектров [14], можно заключить, что при последовательном вычислении множителей канонического произведения (6) от 0 до $n - 1$ спектр промежуточной функции на i-й итерации в алфавите A_{i+1} совпадает со спектром функции g. Действия на i-й итерации не затрагивают спектры алфавитов $A_{i'} : i' < i + 1$. Откуда условие $H_{i+1} = \left\{h \in H'_{i+1} \mid Q_{i+1}\left(f^h\right) = Q_{i+1}(g)\right\}$ выполняет роль необходимых условий существования решения (2) и позволяет исключать множители канонического произведения (и элементы, их содержащие), не удовлетворяющие (2). В результате на последней итерации будут сравниваться спектры в алфавите A_n, причем символами спектра будут являться сами булевы функции. На последней итерации алгоритма необходимые условия $H_n = \left\{h \in H'_n \mid Q_n\left(f^h\right) = Q_n(g)\right\}$ являются также и достаточными, останутся только гипотезы, удовлетворяющие (2). Если на какой-то итерации алгоритма множество H_i будет пусто, то это означает отсутствие решений уравнения (2). Что и требовалось доказать.

Вопрос корректности алгоритма 1 является не единственным основным в настоящем изложении. Интерес представляет вопрос сложности предлагаемого алгоритма по отношению к полному перебору.

Сложность алгоритма поиска решения уравнения (2) определяется числом действий элементов группы Джевонса и расчетом спектров промежуточных функций. Для расчета спектра промежуточной функции требуется время $\tau(n)$ (для оценки *Time*(*n*) из табл. 1). Примем значение $\tau(n)$ в качестве единицы сложности, потому что даже для тривиального алгоритма нужно сравнивать результат действия с эталоном, и время этого сравнения соизмеримо с $\tau(n)$. В единицах $\tau(n)$ сложность тривиального алгоритма будет $O(2^n n!)$. Сложность в единицах $\tau(n)$ предлагаемого алгоритма можно подсчитать исходя из следующего. Пусть r_i — количество промежуточных функций (равное $|H_i|$) на каждом шаге. Над каждой промежуточной функцией выполняется $2(n-i)$ действий. Откуда общее число действий (сложность) элементов группы Джевонса на БФ *n* аргументов можно подсчитать следующим образом:

$$d = \sum_{i=0}^{n-1} r_i 2(n-i). \qquad (7)$$

Покажем пример поиска решения и вычисление сложности (7) уравнения:

$$\{1000\ 0000\ 0011\ 1011\}^{(z\pi)} = \{0001\ 0100\ 1111\ 0000\}.$$

Выполнение алгоритма по шагам отражено в табл. 2.

В результате имеем следующий набор действий:

$$\left(\left(\left(f^{(\{0001\}e_S)}\right)^{(e_E(2,1))}\right)^{(\{0100\}(3,2))}\right)^{(e_E e_S)} = g,$$

откуда

$$(z\pi) = (e_E e_S)(\{0100\}(3,2))(e_E(2,1))(\{0001\}e_S).$$

Произведение нужно вычислить в зависимости от типа А или типа Б по формуле (4.А) или (4.Б) соответственно. Для типа А неизвестное будет $(z\pi) = (\{0101\}(3,1,2))$ и для типа Б — $(z\pi) = (\{0101\}(3,2,1))$. Сложность поиска решения составит

$$d = \sum_{i=0}^{3} 2(4-i) r_i =$$
$$= 2\big((4-0)1 + (4-1)1 + (4-2)2 + (4-3)1\big) =$$
$$= (4+3+4+1) = 2 \cdot 12 = 24$$

Оценка сложности решения методом спектрального анализа БФ

Оценим сложность (7) алгоритма в "худшем" случае. Для этого найдем максимально возможные значения r_i. Тогда $r_0 \equiv 1$, а остальные r_i рассчитываются рекуррентно как $r_i = 2(n-(i-1))r_{i-1}$:

$$r_0 = 1,$$
$$r_1 = 2(n-(1-1))1 = 2n,$$
$$r_2 = 2(n-(2-1))2n = 2(n-1)2n,$$
$$r_2 = 2(n-(3-1))2(n-1)2n = 2(n-2)2(n-1)2n,$$
$$\ldots$$
$$r_i = 2^i \frac{n!}{(n-i)!}.$$

В результате формула (7) в "худшем" случае примет следующий вид (максимальная теоретическая сложность):

$$d_{\max} = \sum_{i=0}^{n-1} 2^i \frac{n!}{(n-i)!} 2(n-i). \qquad (8)$$

Нетрудно видеть, что последнее слагаемое суммы (8) есть $2^n n!$ (порядок группы Джевонса и сложность тривиального алгоритма). Откуда можно заключить, что сложность предлагаемого алгоритма при отсутствии отбраковки гипотез превосходит сложность тривиального. Но такого на самом деле быть не может, потому что значения r_i зависят друг от друга, поскольку группа Джевонса действует интранзитивно. Например, для БФ $f(x_{n-1}, \ldots, x_i, \ldots, x_0) \equiv 0$ значения r_i на каждом шаге действительно максимальны, но можно модифицировать алгоритм так, что в множествах гипотез будут оставаться элементы, соответствующие только различным промежуточным функциям (это также позволит вычислить порождающее множество подгруппы инерции БФ). Поэтому допустимо принять, что $r_i \equiv 1$.

Заключим также, что $d_{\max} = d_{trivial} = 2^n n!$.

Наименьшая сложность предлагаемого решения будет при $r_i \equiv 1$ и равна (как арифметическая прогрессия):

$$d_{\min} = \sum_{i=0}^{n-1} 2(n-i) = 2n \frac{(n-0)+(n-(n-1))}{2} = n^2 + n. \qquad (9)$$

Значения r_i дают возможность оценить среднее и верхнее значения общего количества действий группы Джевонса на БФ при вычислении решения уравнения (2). На момент написания данной статьи теоретические оценки этих чисел не найдены, но есть возможность эмпирической оценки.

Для эмпирической оценки сложности алгоритма 1 специально разработан метод спектрального анализа. Определение значений r_i проводится для исходной функции *f* относительно функции *g*. Даже для небольших значений *n* это вычислительно сложно, так как нужно анализировать все возможные пары *f*, *g*. Всего таких пар для заданного *n* будет $\left(2^{2^n}\right)^2 = 2^{2^{n+1}}$. Например, для $n = 5$ будет 2^{64} пар

Таблица 2

Пример вычисления решения уравнения для $n = 4$

Итерация алгоритма i	Гипотезы	Промежуточные функции	Спектры промежуточных функций
0	$r_0 = 1$	$Q_1(g) =$	Сим. 00 01 11; Час. 4 2 2
	$(e_E e_S)$	{1000 0000 00111011}	Сим. 00 10 11; Час. 4 2 2
	$(e_E(1,0))$	{1000 0000 01011101}	Сим. 00 01 10 11; Час. 3 3 1 1
	$(e_E(2,0))$	{1000 0000 0111 0011}	Сим. 00 01 10 11; Час. 4 1 1 2
	$(e_E(3,0))$	{1001 0101 0001 0001}	Сим. 00 01 10; Час. 2 5 1
	$(\{0001\}e_S)$	{0100 0000 0011 0111}	Сим. 00 01 11; Час. 4 2 2
	$(\{0001\}(1,0))$	{0100 0000 1010 1110}	Сим. 00 01 10 11; Час. 3 1 3 1
	$(\{0001\}(2,0))$	{0100 0000 1011 0011}	Сим. 00 01 10 11; Час. 4 1 1 2
	$(\{0001\}(3,0))$	{0110 1010 0010 0010}	Сим. 00 01 10; Час. 2 1 5
1	$r_1 = 1$	$Q_2(g) =$	Сим. 0000 0001 0100 1111; Час. 1 1 1 1
	$(\{0001\}e_S)$ $(e_E e_S)$	{0100 0000 0011 0111}	Сим. 0000 0011 0100 0111; Час. 1 1 1 1
	$(\{0001\}e_S)$ $(e_E(2,1))$	{0100 0000 00011111}	Сим. 0000 0001 0100 1111; Час. 1 1 1 1
	$(\{0001\}e_S)$ $(e_E(3,1))$	{0100 0001 0011 0011}	Сим. 0000 0011 0100; Час. 1 2 1
	$(\{0001\}e_S)$ $(\{0010\}e_S)$	{0001 0000 1100 1101}	Сим. 0000 0001 1100 1101; Час. 1 1 1 1
	$(\{0001\}e_S)$ $(\{0010\}(2,1))$	{0001 0000 0100 1111}	Сим. 0000 0001 0100 1111; Час. 1 1 1 1
	$(\{0001\}e_S)$ $(\{0010\}(3,1))$	{0001 0100 1100 1100}	Сим. 0001 0100 1100; Час. 1 1 2
2	$r_2 = 2$	$Q_3(g) =$	Сим. 00010100 11110000; Час. 1 1
	$(\{0001\}e_S)$ $(e_E(2,1))$ $(e_E e_S)$	{0100 0000 00011111}	Сим. 00011111 01000000; Час. 1 1
	$(\{0001\}e_S)$ $(e_E(2,1))$ $(e_E(3,2))$	{0100 0001 0000 1111}	Сим. 00001111 01000001; Час. 1 1
	$(\{0001\}e_S)$ $(e_E(2,1))$ $(\{0100\}e_S)$	{0000 0100 1111 0001}	Сим. 00000100 11110001; Час. 1 1
	$(\{0001\}e_S)$ $(e_E(2,1))$ $(\{0100\}(3,2))$	{0001 0100 1111 0000}	Сим. 00010100 11110000; Час. 1 1
	$(\{0001\}e_S)$ $(\{0010\}(2,1))$ $(e_E e_S)$	{0001 0000 0100 1111}	Сим. 00010000 01001111; Час. 1 1
	$(\{0001\}e_S)$ $(\{0010\}(2,1))$ $(e_E(3,2))$	{0001 0100 0000 1111}	Сим. 00001111 00010100; Час. 1 1
	$(\{0001\}e_S)$ $(\{0010\}(2,1))$ $(\{0100\}e_S)$	{0000 00011111 0100}	Сим. 00000001 11110100; Час. 1 1
	$(\{0001\}e_S)$ $(\{0010\}(2,1))$ $(\{0100\}(3,2))$	{0100 00011111 0000}	Сим. 01000001 11110000; Час. 1 1
3	$r_3 = 1$	$Q_4(g) =$	Сим. 0001010011110000; Час. 1
	$(\{0001\}e_S)$ $(e_E(2,1))$ $(\{0100\}(3,2))$ $(e_E e_S)$	{0001 0100 1111 0000}	Сим. 0001010011110000; Час. 1
	$(\{0001\}e_S)$ $(e_E(2,1))$ $(\{0100\}(3,2))$ $(\{1000\}e_S)$	{1111 0000 0001 0100}	Сим. 1111000000010100; Час. 1

Примечание: серым цветом выделены совпадения спектров промежуточных функций и эталонных спектров $Q_i(g)$; Сим. — символ алфавита; Час. — частота символа.

функций. Спектральный анализ БФ позволяет снизить вычислительные затраты настолько, что значения r_i для $n = 5$ могут быть вычислены за время $O\left(2^{2^n}\right)$.

Суть спектрального анализа сводится к следующему. Зафиксируем функцию f и примем, что g — произвольна, другими словами, рассмотрим сразу все уравнения (2) с участием f. Тогда все возможные спектры f на каждой итерации алгоритма 1 будут соответствовать только некоторым возможным g (которые находятся с f в одной орбите). Отдельно стоит рассмотреть ситуацию отсутствия решения уравнения (2). При ней произойдет прерывание алгоритма 1 (пустое множество H_i) на какой-то итерации. При этом значения r_i будут также показывать число действий группы Джевонса на БФ, но их будет неполный набор, т. е. в формуле (7) верхняя граница суммы будет меньше $n - 1$. Поэтому сложность алгоритма для пары f, g из разных орбит будет меньше, чем сложность для пары f, g из одной орбиты.

Для спектрального анализа нужно выполнить итерации алгоритма 1, но не исключать гипотезы из множества H'_{i+1} (пропустить шаг 2), а разделить их на подмножества по спектрам. Далее нужно для каждого подмножества гипотез выполнить итерацию (с пропуском шага 2 и разделением по спектрам) алгоритма 1. На каждой итерации будут сформированы множества множеств гипотез со всеми своими допустимыми спектрами. Задача спектрального анализа — подсчитать их число. Множества гипотез в общем случае не равномощны. Для массивного статистического анализа (миллиарды функций) это неудобно. Тогда можно поступить следующим образом: на каждой итерации алгоритма 1 выбрать максимальное r_i из всего множества множеств гипотез. Далее обозначим эти значения как $\tilde{r}_i$. Тогда набор $\tilde{r}_i$ будет отражать максимально возможное число действий для всех возможных уравнений с участием конкретной функции f. Важно также отметить, что набор значений $\tilde{r}_i$ может превышать верхнюю границу сложности, потому что при спектральном анализе на каждом шаге алгоритма 1 выбирается наибольшее по мощности подмножество промежуточных функций, независимо от предыдущего шага. Например, пусть на шаге i для наибольших по мощности подмножеств промежуточных функций $H_{i,0}$, $H_{i,1}$ значения мощностей будут равны $r_{i,0} = 2$ и $r_{i,1} = 5$ соответственно, откуда $\tilde{r}_i = 5$. Пусть на следующем $i + 1$ шаге $H_{i,0}$ даст подмножества $H_{i+1,0,0}$ и $H_{i+1,0,1}$ со значениями мощностей $r_{i+1,0,0} = 1$ и $r_{i+1,0,1} = 10$ соответственно, а $H_{i,1}$ — подмножества $H_{i+1,1,0}$ и $H_{i+1,1,1}$ со значениями мощностей $r_{i+1,1,0} = 1$ и $r_{i+1,1,1} = 2$, откуда $\tilde{r}_{i+1} = 10$. При этом реальная сложность будет определяться парами чисел $r_{i,0} = 2$, $r_{i+1,0,1} = 10$ или $r_{i,1} = 5$, $r_{i+1,1,1} = 2$. Другими словами, значения $\tilde{r}_i$ дают грубую верхнюю оценку, но реальная сложность ее не превысит.

Следующим этапом является анализ выборки множества функций для заданного n. В результате анализа функции будут группироваться по значениям $\tilde{r}_i$, или набору таких чисел будет соответствовать некоторая доля БФ выборки. Тогда максимум для этих чисел для всей выборки покажет наихудший вариант применения алгоритма 1. Математическое ожидание $\tilde{r}_i$ по долям выборки для таких группировок покажет среднюю сложность алгоритма 1.

Перед демонстрацией результатов спектрального анализа нужно показать причины наличия значений $\tilde{r}_i$ больших единицы. Из доказательства теоремы 4 можно заключить, что если уравнение (2) имеет больше одного решения, то на последнем шаге будет ровно столько же решений. Наличие значения $\tilde{r}_i$ больше единицы может показывать, что для канонического представления (6) перебираются множители, входящие в элементы подгруппы инерции БФ группы Джевонса $J_{D_n}(f)$, потому что будут выполняться условия теоремы 4. Такой рост сложности будет устойчивым до окончания работы алгоритма 1. Например, для БФ $f(x_{n-1}, \ldots, x_i, \ldots, x_0) \equiv 0$ $J_{D_n}(f)$ совпадает с группой Джевонса $J_{D_n}(f) = D_n$, и числа r_i на каждой итерации алгоритма 1 имеют максимально допустимые значения.

Откуда можно заключить следующее. БФ с нетривиальной $J_{D_n}(f)$ могут давать экспоненциальный рост сложности алгоритма 1. При этом сложность сопоставима с порядком $J_{D_n}(f)$. Верно и другое: мощность множества БФ с нетривиальной $J_{D_n}(f)$ стремится к нулю при $n \to \infty$. Это доказано в теории Пойа [6], поэтому такими случаями можно пренебречь без потери общности. Более того, для функций с нетривиальной $J_{D_n}(f)$ нет необходимости хранить все гипотезы и можно модифицировать алгоритм 1 так, что он будет вычислять порождающее множество $J_{D_n}(f)$. Это показано в предыдущем примере для $f(x_{n-1}, \ldots, x_i, \ldots, x_0) \equiv 0$.

В результате эмпирического анализа было показано, что также и для БФ с тривиальной $J_{D_n}(f)$ значения $\tilde{r}_i$ могут быть больше единицы. При этом, исходя из мощности орбиты, которая может быть выражена как индекс $\left[D_n : J_{D_n}(f)\right]$, на последней итерации алгоритма 1 остается ровно одна гипотеза, которая и является решением уравнения (2). Это позволяет заключить, что в процессе работы алгоритма 1 появляются ложные промежуточные решения, которые с каждой последующей итерацией отбраковываются.

При анализе всех БФ с тривиальной $J_{D_n}(f)$ для $n = 4, 5$ и некоторого множества функций с большим числом аргументов ($n > 19$) выявлена только одна причина появления ложных промежуточных решений. На каждой итерации (кроме последней) алгоритма 1 не проводится работа ни с промежуточными функциями f^h, ни с функцией g. Работа проводится с их спектрами в соответствующих итерациям алфавитах A_{i+1}. Таким образом, вместо f^h и g рассматривается множество булевых функций, которые могут

быть образованы перестановкой символов A_{i+1} в f^h и g. Другими словами, отображение БФ на спектр $A_i : i < n$ — сюръекция. Для проанализированного множества БФ выявлено, что если значение $\tilde{r}_i$ больше единицы, то среди прообразов спектров всегда присутствует БФ с нетривиальной $J_{D_n}(f)$. Более того, для такой БФ в ее подгруппе инерции присутствует элемент, действующий на аргумент x_i. Таким образом, ложные промежуточные функции ложны в рамках всего алгоритма 1, но в рамках конкретной итерации такие промежуточные функции ожидаемы, так как алгоритм 1 "не может" отличить БФ f^h или g от других прообразов с нетривиальной $J_{D_n}(f)$.

В результате можно выдвинуть три перечисленных далее предположения.

1. Появление ложных промежуточных функций обусловлено сюръекцией БФ с тривиальной и нетривиальной $J_{D_n}(f)$ на один и тот же спектр, и именно БФ с нетривиальной $J_{D_n}(f)$ являются причиной $\tilde{r}_i > 1$.

2. В первом случае доля БФ, дающих ложные промежуточные решения, при $n \to \infty$ стремится к нулю, так как доля БФ с нетривиальной $J_{D_n}(f)$ также стремится к нулю. Более того, число ложных промежуточных функций при работе алгоритма 1 должно экспоненциально падать от итерации к итерации, так как число прообразов в сюръекции уменьшается экспоненциально с ростом индекса алфавита.

3. Алгоритм 1 невозможно модифицировать так, чтобы его сложность рассчитывалась по формуле (9) в общем случае, так как причина роста сложности — сюръекция БФ с тривиальной и нетривиальной $J_{D_n}(f)$ на один и тот же спектр, но это основа самого алгоритма 1. При этом, если верно второе, то сложность алгоритма 1 должна быть достаточна для вычисления решения уравнения (2) за разумное время (см. табл. 1).

Далее показан эмпирический результат отбраковки ложных промежуточных функций для БФ четырех и пяти аргументов. Результат не противоречит первому и второму предположениям, но не доказывает их. Полученные решения уравнений (2) алгоритмом 1 для $n > 19$ указывают на непротиворечивость третьего предположения, но также не доказывает его.

Рассмотрим результаты спектрального анализа для $n = 4$. Для удобства данные сведены в табл. 3. Значение $r_0 \equiv 1$ в табл. 3 не приведено. Для остальных итераций $i = 1, 2, 3$ алгоритма имеем следующие значения $\tilde{r}_i$. В ячейках табл. 3 указан набор значений $\tilde{r}_i$ через дефис, чтобы показать, как разделяются множества гипотез при переходе от итерации к итерации алгоритма. Дополнительно указана доля таких функций по отношению к объему генеральной совокупности. Анализ проводился для 22 656 функций с тривиальной $J_{D_n}(f)$. Например, множество "1-2" мощностью 17 664, причем каждая функция имеет максимум две гипотезы $\tilde{r}_1 = 2$, формируется после нулевой итерации алгоритма и составляет больше половины генеральной совокупности. На следующей итерации из него формируются два равномощных множества "1-2-1" $(\tilde{r}_2 = 1)$ и "1-2-2" $(\tilde{r}_2 = 2)$. .Для функций множества "1-2-1" сложность возвращается в полиномиальные границы, а для функций множества "1-2-2" возврат произойдет только на третей итерации алгоритма.

На следующих итерациях алгоритма ложные промежуточные решения устраняются, потому что с ростом индекса алфавита уменьшается экспоненциально количество совпадений спектров БФ с тривиальной и нетривиальной $J_{D_n}(f)$. Задачей спектрального анализа является оценка значений $\tilde{r}_i$ для БФ с тривиальной $J_{D_n}(f)$. Анализ был проведен для всех таких функций при $n = 4, 5$. Аналог табл. 3 для $n = 5$ слишком велик для настоящего изложения, и далее показаны только итоги по нему. В результате было выявлено, что доля БФ с тривиальной $J_{D_n}(f)$, но дающих ложные промежуточные функции на разных итерациях алгоритма, крайне мала и при росте n уменьшается, что указывает на непротиворечивость второго предположения, но не доказывает его.

В табл. 4, 5 приведены средние и наихудшие значения $\tilde{r}_i$ для $n = 4$ и 5 соответственно. Также для сравнения приведены значения для тривиального алгоритма.

В табл. 6, 7 приведен расчет среднего и максимального числа действий группы Джевонса по (7) для $n = 4$ и 5 соответственно. В дополнение приведена

Таблица 3

Результаты спектрального анализа для всех БФ для $n = 4$ с тривиальной $J_{D_n}(f)$

Итерация алгоритма i	$\tilde{r}_i = 1$					$\tilde{r}_i = 2$			$\tilde{r}_i = 3$	$\tilde{r}_i = 4$
1	1-1 3072 13,56 %	—	—	—	—	1-2 17664 77,97 %	—	—	1-3 768 3,39 %	1-4 1152 5,08 %
2	1-1-1 3072 13,56 %	1-2-1 8832 38,98 %	—	—	—	1-2-2 8832 38,98 %	1-3-2 768 3,39 %	1-4-2 1152 5,08 %	—	—
3	1-1-1-1 3072 13,56 %	1-2-1-1 8832 38,98 %	1-2-2-1 8832 38,98 %	1-3-2-1 768 3,39 %	1-4-2-1 1152 5,08 %	—	—	—	—	—

Таблица 4

Значения $\tilde{r}_i$ для БФ при $n = 4$

Оценка	$\tilde{r}_0$	$\tilde{r}_1$	$\tilde{r}_2$	$\tilde{r}_3$
Средняя	1	2	1,4746	1
Худшая	1	4	2	1
Тривиальная (для сравнения)	1	8	48	192

Таблица 5

Значения $\tilde{r}_i$ для БФ при $n = 5$

Оценка	$\tilde{r}_0$	$\tilde{r}_1$	$\tilde{r}_2$	$\tilde{r}_3$	$\tilde{r}_4$
Средняя	1	2,1317	1,5060	1,0458	1
Худшая	1	8	16	3	1
Тривиальная (для сравнения)	1	10	80	480	1 920

Таблица 6

Эффективность алгоритма при $n = 4$

Оценка	Сложность	Эффективность, %
Лучшая	20	94,7917
Средняя	27,8983	92,7348
Худшая	42	89,0625
Тривиальный алгоритм (для сравнения)	384	—

минимальная оценка сложности алгоритма 1, чтобы показать, что существует объективное минимальное число действий, без которых решение уравнения (2) не может быть найдено. Приведена также оценка эффективности алгоритма 1. Под эффективностью понимается сложность предлагаемого алгоритма по отношению к тривиальному.

Анализ проводили для выборки, совпадающей с генеральной совокупностью для $n = 4$ и 5, т. е. для всех БФ указанного числа аргументов. Для больших значений ($n > 5$) анализ всех БФ вычислительно сложен. Из полученных эмпирических данных удалось сформулировать четвертое предположение о том, что для определения $\tilde{r}_i$, , показывающих среднюю оценку сложности алгоритма 1, нужно увеличивать число случайно выбираемых БФ для исследования методом спектрального анализа, пока не будет встречена первая БФ с нетривиальной $J_{D_n}(f)$.

Из данных табл. 6 и 7 можно также заключить, что эффективность алгоритма растет с увеличением числа аргументов БФ, что указывает на непротиворечивость второго предположения, но не доказывает его.

Таблица 7

Эффективность алгоритма при $n = 5$

Оценка	Сложность	Эффективность, %
Лучшая	42	99,2188
Средняя	42,2730	98,8991
Худшая	184	95,2083
Тривиальный алгоритм (для сравнения)	3 840	—

Заключение

Одним из направлений исследования ИО является анализ их свойств при представлении в виде БФ. Такой анализ в том числе включает в себя поиск решений уравнения (2) за разумное время. Предлагаемый алгоритм 1 в подавляющем большинстве случаев обеспечивает это. Исследование свойств ИО, представленных в виде БФ, также сводится к отысканию корректных алгебраических операций над множеством орбит БФ относительно группы Джевонса. Потенциально такие алгебраические операции могут основываться на решении уравнения (2).

Для обеспечения возможности работы в этой предметной области была разработана библиотека "domain operations processor" (или **dop**). Она содержит набор алгоритмов, специализированных для обработки подстановок и их действий на БВ. Библиотека распространяется под лицензией GNU LGPL (GNU Lesser General Public License). Библиотека **dop** написана на языке C++ в виде системы функций и макросов для архитектуры процессора IA-32. Библиотека является кроссплатформенной, так как не использует программный интерфейс Windows- или POSIX-подобных систем. Библиотека включает в себя: инструменты ввода-вывода объектов, обработки (групповые операции, изоморфизмы, антиизоморфизмы, действия и пр.), перечисления объектов (для построения стендов при исследовании предметной области), средства рандомизации. Состав библиотеки, включая документацию разработчика (Software Development Kit или SDK), можно получить у авторов настоящей статьи.

Библиотека **dop**, механизмы которой описаны в работах [12—14], совместно с основным алгоритмом решения (2) является исследовательской основой для анализа информационных объектов, представленных в виде БФ.

Реализации основного алгоритма на базе **dop** показали, что решение указанного в примере во введении уравнения для $n = 19$ занимает секунды в противовес тривиальному алгоритму (см. табл. 1). Важно также отметить, что материалы настоящей статьи вызывают сомнения в правильности применения таких преобразований в качестве криптографических примитивов для шифров, основывающихся на управляемых операциях.

Дальнейшие исследования по основному алгоритму выделены в отдельные направления и сводятся к проверке сформулированных предположений, преимущественно к проверке предположения о появлении ложных промежуточных функций в силу наличия сюръективного отображения БФ с тривиальной и нетривиальной $J_{D_n}(f)$ на один и тот же спектр и последующей теоретической оценке верхней границы сложности алгоритма. Также дальнейшие исследования сводятся к модификации алгоритма для вычисления порождающего множества подгруппы инерции БФ в группе Джевонса. Это позволит не только решать уравнения (2) для БФ с нетривиальной $J_{D_n}(f)$, но и решить одну из задач классификации БФ — описание подгрупп инерции эквивалентных БФ относительно группы Джевонса, — сформулированную в работе [4]. Отдельным направлением дальнейших исследований является практическая проверка предположения о минимальном числе исследуемых БФ методом спектрального анализа для определения значений $\tilde{r}_i$, показывающих среднюю сложность, и уточнение этого числа БФ для $n > 5$.

Список литературы

1. **Сэломон Д.** Сжатие данных, изображений и звука. Пер. с англ. В. В. Чепыжова. М.: Техносфера, 2004. 368 с.
2. **Яблонский С. В.** Введение в дискретную математику: Учеб. пособие для вузов — 2 изд., перераб. и доп., М.: Наука. Гл. ред. физ.-мат. лит. 1986.
3. **Марченко С. С.** Замкнутые классы булевых функций. М.: ФИЗМАТЛИТ, 2000. 128 с.
4. **Глухов М. М., Ремизов А. Б., Шапошников В. А.** Обзор по теории k-значных функций. Часть 1. Справочное пособие / Ред. Н. Р. Емельянов. Заказ № 163ф. М.: Типография в/ч 33965, 1988. 153 с.
5. **Клейнберг Дж., Тардос Е.** Алгоритмы: разработка и применение. Классика Computers Science. Пер. с англ. Е. Матвеева. СПб.: Питер, 2016. 800 с.
6. **Логачев О. А., Сальников А. А., Ященко В. В.** Булевы функции в теории кодирования и криптологии. М.: МЦМНО, 2004. 470 с.
7. **Golomb S. W.** On classification of Boolean functions // IRE, Trans. circuit theory, 1959. No. 6, Spec. Suppl. P. 176—186.
8. **Якубайтис Э. А.** Субклассы и классы булевых функций // Автоматика и вычислительная техника. 1974. № 1. С. 1—8.
9. **Шниперов А. Н.** Синтез и анализ высокоскоростных симметричных криптосистем на основе управляемых операций // Информационные технологии. 2008. № 1. С. 36—41.
10. **Супруненко Д. А.** Группы подстановок. Мн.: Навука і тэхніка, 1996. 366 с.
11. **Каргаполов М. И., Мерзляков Ю. И.** Основы теории групп. 3-е изд., перераб. и доп. М.: Наука, 1982. 288 с.
12. **Кукарцев А. М., Кузнецов А. А.** О конструктивном представлении группы Джевонса для инженерно-технических решений обработки информации // Программная инженерия. 2015. № 11. С. 25—33.
13. **Кукарцев А. М., Кузнецов А. А.** О действиях группы Джевонса на множествах бинарных векторов и булевых функций для инженерно-технических решений обработки информации // Программная инженерия. 2016. Том 7, № 1. С. 29—36.
14. **Кукарцев А. М.** О частотных свойствах действий группы Джевонса на булевых функциях // Программная инженерия. 2016. Том 7, № 11. С. 515—521.

On the Fast Solution of the Jevons Group Action Equation on Boolean Functions

A. M. Kukartsev, amkukarcev@yandex.ru, **A. A. Kuznetsov,** kuznetsov@sibsau.ru,
Siberian State Aerospace University named after academician M. F. Reshetnev, Krasnoyarsk, 660014, Russian Federation

Corresponding author:

Kukartsev Anatolii M., Senior Lecturer, Siberian State Aerospace University named after academician M. F. Reshetnev, Krasnoyarsk, 660014, Russian Federation, e-mail: amkukarcev@yandex.ru

Received on September 29, 2016
Accepted on November 14, 2016

Action of the Jevons group on Boolean functions of n arguments is intransitive and divides the set into orbits of the same cardinalities. In most cases number of elements in the orbit is equal of the group order, i.e. $2^n n!$. Calculation of the acting group element which links to two functions in the same orbit is a computationally hard mathematical problem. It refers to the problems of classification of Boolean functions relatively to group action. So this action is used as a cryptographic primitive of encryption algorithms that are based on managed operations where a key is an element of the Jevons group and Boolean functions are original encrypted data.

We propose an efficient algorithm for computing the acting group element which links (the) two functions. The time complexity of the algorithm is much less than complete enumeration. This algorithm is based on the sequential computation of factors of the acting element written in the canonical form. The presence or absence of a factor in such canonical representation in solution of the equation is determined by frequency characteristics of binary vectors which are equivalent to the unknown Boolean function. We prove the correctness of the algorithm and give the

formula for calculating its time complexity. After that an example demonstrates a solution of the equation for Boolean functions of four arguments.

We calculate the maximum and minimum theoretical estimates of the time complexity. A number of numerical experiments allowed us to formulate the assumption that complexity of the solution is $O(n^2)$ in most cases. Such assessment of complexity is characteristic for equations including Boolean functions with nontrivial inertia subgroup of the Jevons group. For the numerical experiments a spectral analysis method of Boolean functions is developed. Using this method we calculate the minimum, maximum and average estimates of the algorithm complexity for all equations with Boolean functions of four and five arguments with a trivial inertia subgroup of the Jevons group. The results lead to the assumption that the algorithm will be effective for processing of real input data.

Keywords: *action on the set, frequency analysis, the Jevons group, Boolean functions, equations of group action on the set Boolean functions*

Acknowledgements: *This work was supported by the Grant of Russian Federation President, project nos. MD-3952.2015.9*

For citation:

Kukartsev A. M., Kuznetsov A. A. On the Fast Solution of the Jevons Group Action equation on Boolean functions, *Programmnaya Ingeneria,* 2017, vol. 8, no. 2, pp. 76—87.

DOI: 10.17587/prin.8.76-87

References

1. **Salomon D.** *A guide to data compression methods*, New York, Springer-Verlag, 2006. 1092 p.

2. **Jablonskij S. V.** *Vvedenie v diskretnuju matematiku: Ucheb. posobie dlja vuzov* — 2 izd., pererab. i dop. (Introduction to Discrete Mathematics), Moscow, Nauka. Gl. red. fiz.-mat. lit., 1986 (in Russian).

3. **Marchenko S. S.** *Zamknutye klassy bulevyh funkcij* (Post's lattice), Moscow, FIZMATLIT, 2000, 128 p. (in Russian).

4. **Gluhov M. M., Remizov A. B., Shaposhnikov V. A.** *Obzor po teorii k-znachnyh funkcij. Chast' 1. Spravochnoe posobie* (Review on the Theory of k-valued functions. Part 1. A Reference Guide), Moscow, Tipografija v/ch 33965, 1988, 153 p. (in Russian).

5. **Kleinberg J., Tardos E.** *Algorithm design*, New York, Pearson Education, Addison Wesley, 2005, 857 p.

6. **Logachjov O. A., Sal'nikov A. A., Jashhenko V. V.** *Bulevy funkcii v teorii kodirovanija i kriptologii* (Boolean functions in coding theory and cryptology), Moscow, MCMNO, 2004, 470 p. (in Russian).

7. **Golomb S. W.** On classification of Boolean functions, *IRE, Trans.circuit theory, Spec.Suppl.*, 1959, no. 6, pp. 176—186.

8. **Jakubajtis Je. A.** Subklassy i klassy bulevyh funkcij (Sub class and the class of Boolean functions), *Avtomatika i vychislitel'naja tehnika*, 1974, no. 1, pp. 1—8 (in Russian).

9. **Shniperov A. N.** Sintez i analiz vysokoskorostnyh simmetrichnyh kriptosistem na osnove upravljaemyh operacij (Synthesis and analysis of high-speed symmetric cryptosystems based on controlled transactions), *Informacionnye tehnologii*, 2008, no. 1, pp. 36—41 (in Russian).

10. **Suprunenko D. A.** *Gruppy podstanovok* (Groups of permutations), Minsk, Navuka i tjehnika, 1996, 366 p. (in Russian).

11. **Kargapolov M. I., Merzljakov Ju. I.** *Osnovy teorii grupp. 3-e izd., pererab. i dop* (Basics of group theory), Moscow, Nauka, 1982, 288 p. (in Russian).

12. **Kukartsev A. M., Kuznetsov A. A.** O konstruktivnom predstavlenii gruppy Dzhevonsa dlja inzhenerno-tehnicheskih reshenij obrabotki informacii (Constructive representation of the Jevons group for engineering solutions of information processing), *Programmnaya Ingeneria*, 2015, no. 11, pp. 25—33 (in Russian).

13. **Kukartsev A. M., Kuznetsov A. A.** O dejstvijah gruppy Dzhevonsa na mnozhestvah binarnyh vektorov i bulevyh funkcij dlja inzhenerno-tehnicheskih reshenij obrabotki informacii (About Actions of the Jevons Group on Sets of Binary Vectors and Boolean Functions for Engineering Solutions of Information Processing), *Programmnaya Ingeneria*, 2016, vol. 7, no. 1, pp. 29—36 (in Russian).

14. **Kukartsev A. M.** O chastotnyh svojstvah dejstvij gruppy Dzhevonsa na bulevyh funkcijah (On Frequency Characteristics Jevons Group Action on Boolean functions), *Programmnaya Ingeneria*, 2016, vol. 7, no. 11, pp. 515—521 (in Russian).

УДК 338.364.2 DOI: 10.17587/prin.8.88-95

Д. С. Пащенко, канд. техн. наук, MBA, e-mail: denpas@rambler.ru,
независимый консультант в области разработки программного обеспечения

Географически распределенные команды: естественные и организационные особенности проектов разработки программного обеспечения

Географически распределенные команды, участвующие в выполнении проектов высокотехнологичных компаний, обладают существенными экономическими преимуществами и широко используются во всем мире. Однако такая организация проектной деятельности также имеет массу особенностей и сопутствующих рисков, способных нивелировать получаемый экономический эффект. Осенью 2016 г. менеджеры из международных компаний с офисами в странах СНГ представили автору свой опыт в управлении географически распределенными проектами разработки программного обеспечения. В данной работе обобщены результаты этих интервью, представлена классификация особенностей данного подхода к организации производственных проектов, при этом основной акцент сделан на две группы особенностей — естественные и организационные. Для каждой особенности внутри группы дано описание, указаны факторы, на которые она влияет, приведены сопутствующие риски. Проанализированы основные подходы к управлению возникающими рисками, приведены типичные управляющие воздействия и примеры из опыта автора.

Ключевые слова: *географически распределенная команда, проект разработки программного обеспечения, разработка программных продуктов, перспективные производственные практики*

Введение

Компании, разрабатывающие промышленное коммерческое программное обеспечение (ПО), в России и странах СНГ завершают стадию специализации. В ее основе повсеместная формализация процессов разработки ПО и определение основного фокуса предлагаемых решений на определенный сегмент рынка или на тип ПО [1]. При этом подбор персонала для участия в проектах таких компаний связан не только с оценкой общих навыков в IT-технологиях, но и с пониманием им предметной области. Технологические университеты региональных центров России наладили "поточный выпуск" различных профильных специалистов. Это обусловливает создание крупными IT-компаниями региональных офисов разработки и освоение практик разработки ПО географически распределенными командами. Вместе с этим в современном мире Россия и ее рынок IT-услуг давно стали частью мирового рынка [2], и этот факт уже нельзя изменить никакими административно-политическими ограничениями. Российские офисы международных компаний вовлечены в международные проекты разработки ПО, над которым одновременно могут работать специалисты из нескольких стран в различных офисах.

В США такая географически распределенная разработка давно является предметом академического и прикладного изучения. В современных работах ряда авторов [3-5] представлены пути ее оптимизации, проанализированы методы повышения качества процессов планирования и управления разработкой, предложены рекомендации по созданию общей базы знаний проекта.

В данной публикации представлены основные особенности работы географически распределенных команд, отмечены типичные проблемные вопросы и сопутствующие им риски, предложены меры реагирования и преодоления. Завершается статья набором общих рекомендаций, которые призваны на начальном этапе избежать существенных рисков, связанных с таким видом организации бизнеса в сфере разработки и внедрения ПО.

Актуальность темы исследования подтверждается следующими фактами:

- более 75 % крупнейших российских IT-компаний из самых крупных категорий по объемам

бизнеса по оценке Russoft имеют распределенные команды разработки программных продуктов [6];

- не менее 20 % крупнейших в мире компаний с бизнес-критичным производством ПО (включая IT-компании, банки, телекоммуникационные компании и т. д.) имеют локальные центры разработки в России, вовлеченные в международные проекты [6].

Фактически это означает, что в масштабах России в командах распределенной разработки работают тысячи, возможно, лучших инженеров, создающих инновационные программные продукты под брендами самых сильных "игроков" мирового рынка (Google, IBM, Intel, Samsung, Dell, Huawei, CISCO, SAP, Яндекс, Лаборатория Касперского, ABBYY и т. д.)

Анализ основных проблемных вопросов в организации и мониторинге процессов разработки ПО в распределенных командах как на международном уровне, так и внутри России и СНГ, позволяет выработать перечень типовых управляющих воздействий, повышающих эффективность разработки. В данной работе обобщен опыт автора и результаты интервью руководителей среднего звена различных российских компаний и российских офисов международных компаний, занимающихся разработкой и внедрением ПО географически распределенными командами. В работе использован опыт следующих компаний (список дан по убыванию оборота бизнеса и численности персонала): Luxoft Russia, EverNote Corp, PlayTech Ukraine, MISYS Russia, RedSys, BSC Msc, MANQRO Rus.

Особенности процессов организации разработки географически распределенными командами следует категоризировать следующим образом (по характеру их возникновения):

- естественные;
- производственные;
- организационные.

К естественным особенностям относится печень факторов природного и антропогенного характера: часовые пояса; языковой барьер; ментальность членов проектных команд. На такие особенности руководитель проектной команды повлиять не может, поэтому ему остается только с помощью корректирующих воздействий снижать негативные последствия перечисленных особенностей.

К производственным особенностям относятся изменения в ежедневных процессах разработки ПО, а именно — анализе и проектировании, в создании ПО, управлении качеством, в разработке сопроводительной документации и т. д. Наибольшее влияние на возникновение таких производственных особенностей оказывают именно риски из группы естественных ограничений.

Организационные особенности включают в себя управление персоналом, коммуникациями команды, административной, технической и технологической поддержкой производственных процессов.

В настоящей работе основное внимание направлено на рассмотрение двух типов особенностей: естественных и организационных. Рассмотрение этих особенностей сопровождается анализом сопутствующих рисков, описанием планов и отдельных мер по их преодолению. Отдельно приведены дополнительные преимущества, которые являются следствием естественных и организационных особенностей.

Естественные особенности и сопутствующие риски

К естественным особенностям относятся факторы, носящие естественный и антропогенный характер. Сопутствующие риски объективны, их можно только принять и предпринимать корректирующие воздействия по снижению негативного влияния последствий их проявления. К числу естественных особенностей относятся следующие:

- разница в часовых поясах между различными участниками и группами участников разработки, которая влияет на все коммуникации команды;
- языковой барьер, возникающий во всех командах, где язык проектов разработки ПО не является родным для членов проектных команд;
- разница в ментальных картах членов команд разработки, которая определяется социально-культурными факторами;
- разница в рабочих календарях, включающая не только различие в государственных и религиозных праздниках, но и даже разную продолжительность рабочей недели и различные рабочие дни в неделе.

Разница в часовых поясах является одним из самых серьезных факторов, влияющих на выполнение проекта на всех этапах жизненного цикла ПО — от планирования расписания проекта до ежедневных коммуникаций. Современные IT-компании довольно часто имеют такой разброс часовых поясов у локальных офисов, что управление данной особенностью в процессах разработки может как ускорить, так и замедлить разработку. К основным вопросам в сфере управления проектами разработки ПО, на которые влияет разница в часовых поясах, относятся перечисленные далее.

Коммуникации — необходимо заранее планировать промежутки времени, удобные для различных участников сеансов связи. При составлении коммуникационного плана разница в часовых поясах должна быть одним из ключевых параметров планирования.

Управление процессами разработки — порядок планирования работ, позволяющий сделать удлиненный день разработки и повышающий общую производительность команды. "Идеальная разработка" распределенными командами подразумевает следующие преимущества:

— увеличенный интервал взаимодействия с бизнес-заказчиком по различным вопросам (например, планирование следующего релиза, валидация

реализованных функциональных возможностей или управление требованиями);

— передача решения задачи на удлиненный рабочий день в режиме итерационной разработки (со смещением по часовым поясам);

— передача решения задачи на удлиненный рабочий день в режиме тестирования (со смещением по часовым поясам).

После прохождения в режиме "удлиненного рабочего дня" миниэтапов работы с требованиями, разработки и тестирования собирается версия информационной системы, она проходит валидацию, и миницикл повторяется вновь. Такая "идеальная разработка" позволяет командам работать по 18...20 часов в сутки, передавая задачи и решения из одного офиса в другой.

Организация поддержки решения в эксплуатации только дневными сменами инженеров (из разных офисов), которые более эффективны и менее затратны для компании. При этом довольно часто поддержка заказчиков (или решений) одновременно осуществляется на разных континентах, когда однотипные проблемные вопросы возникают вместе с началом активного делового времени суток. В таком случае наличие смен инженеров, живущих в схожих часовых поясах с пользователями системы, становится крайне важной.

В проектных командах с распределенными по различным часовым поясам командами разработки ПО наиболее интересным управляющим воздействием является изменение начала и завершения рабочего дня таким образом, чтобы синхронизировать всю совместную работу. Инициатива составления такого ежедневного расписания должна исходить от бизнес-заказчика. Под это расписание подстраиваются команды разработчиков. При разбросе часовых поясов в 3...4 часа такая организация позволяет почти полностью устранить негативные последствия разницы в часовых поясах.

Языковой барьер в командах разработчиков чаще всего проявляется в двух случаях: различие родных языков разработчиков из разных офисов и различие родного языка бизнес-заказчика и официального языка проекта. Сопутствующие риски лежат в области появления искажений информации в самых критичных для проекта документах, поэтому руководителю проекта необходимо предпринять набор корректирующих мер. Традиционными методами минимизации негативных последствий таких рисков являются следующие управляющие воздействия:

- создание и ведение проектной документации на официальном языке проекта;
- формализация всех значимых коммуникаций (протоколы встреч, согласование результата коммуникации и т. п.);
- определение уровня проникновения официального языка проекта во все коммуникации в общей команде (бизнес-заказчик + менеджмент + команды разработки);
- взаимодействие с бизнес-заказчиком на его родном языке и транслирование значимых результатов в команды разработки на официальном языке проекта.

При таком подходе одним из ключевых факторов в подборе персонала для команды разработчиков ПО является достаточно хорошее для организации коммуникаций владение ими официально принятым языком проекта.

Рассмотрим пример из практики автора. Международный европейский вендор ПО при участии российского системного интегратора внедряет в крупном японско-российском банке решение класса "Интернет-банк". В коммуникационном плане проекта заложены следующие основные правила взаимодействия членов всех команд, участвующих в данном проекте:

- официальный язык взаимодействия проектной команды — английский;
- основная официальная документация проекта (план работ, техническое задание, план рисков и т. д.) ведется на английском языке; встречи разработчиков из различных офисов вендора и интегратора ведутся на английском языке.
- локальный интегратор взаимодействует с заказчиком (банком) на русском языке;
- часть промежуточных документов по взаимодействию ведется на русском языке, силами локального интегратора бизнес-значимые результаты транслируются на языке проекта заинтересованным участникам (в том числе — командам разработчиков);
- заказчик своими силами осуществляет перевод значимых результатов коммуникации на японский язык для заинтересованных менеджеров;
- согласование документации и результатов работ с заказчиком идет на русском языке;
- согласование документации и результатов работ внутри команды проекта и между группами разработки ведется на английском языке.

Разница в ментальных картах различных участников распределенных команд разработки носит наднациональный характер и сопряжена в большей степени с социально-культурными особенностями регионов планеты, в том числе различиями в материальном достатке. Наиболее четко данный фактор прослеживается при сравнении команд разработчиков из западных стран (США, Германия, Великобритания) и команд из крупнейших мировых экономик экспорта IT-услуг (Китай, Индия). На личном и групповом уровнях исследователи находят шесть существенных групп различий [7], которые оказывают самое серьезное влияние на реализацию задач в разработке ПО — от формулирования требований до поддержки в режиме эксплуатации.

Следует отметить, что даже внутри одной большой страны, например, такой как Россия, ментальные отличия могут быть значимым фактором воздействия на процесс производства ПО. Опыт компаний, рассматривавшийся в данном исследовании, показал, что наиболее сильное негативное влияние ментальные различия оказывают на вовлеченность сотрудников в проект. Так сотрудники из региональных офисов медленнее, чем члены столичных команд, осознают особенности автоматизации предметной области и суть новых проектов.

Выделим рекомендации, основанные на практическом опыте, для минимизации негативных факторов влияния, связанных с ментальными различиями.

- Глубокая формализация процессов и документирование на всех этапах жизненного цикла ПО вне зависимости от особенностей применяемых парадигм разработки. Данная практика существенно ограничивает использование гибких методологий в распределенных командах с сильными ментальными различиями.
- Последовательность оценки совместимости и необходимых корректировок при сравнении следующих групп участников проекта: бизнес-заказчики; команды разработчиков; менеджмент. Необходимые корректировки лежат в области формализации требований, на уровне мониторинга состояния реализации отдельных задач, в процессе валидации требований к продукту, при определении резервов времени в процессе планирования проекта.
- Детальное понимание ментальных особенностей различных команд разработчиков как с учетом исторического проектного опыта, так и общего мнения на рынке ПО.

Разница в производственных календарях у различных команд разработчиков ПО является не слишком явной, однако на практике она зачастую вносит существенный вклад в причины опоздания по срокам и снижения качества программных продуктов. В действительности разница в рабочих календарях команды разработчиков в рамках международной кооперации может быть очень значительной. К числу определяющих такую разницу обстоятельств можно отнести перечисленные далее.

- Набор государственных и религиозных праздников между Россией и всеми странами мира (кроме стран СНГ) не совпадает ни в одной дате, кроме, пожалуй, Нового года. Следует отметить, что и он празднуется в разные даты с аналогами в Китае и Индии.
- Набор выходных дней каждую неделю в мире не совпадает, а именно:
  - в большинстве стран мира рабочая неделя длится с понедельника по пятницу;
  - в некоторых странах — экспортерах IT-услуг (Индия, Южная Корея) субботы частично или полностью являются рабочими днями;
  - в Израиле, безусловном лидере экспорта IT-услуг и продуктов в Средиземноморье, основным выходным днем является суббота, рабочая неделя начинается в воскресенье и заканчивается в четверг или в пятницу после обеда;
  - в странах Ближнего и Среднего Востока, прежде всего мусульманских, основным выходным днем является пятница, рабочая неделя продолжается с субботы до среды (Алжир и Саудовская Аравия), с субботы до четверга (Иран), либо с воскресенья до четверга (Египет, Сирия, Ирак, Объединенные Арабские Эмираты).
- Рабочие часы в сутках не совпадают в различных офисах — существуют страны с 60-часовой рабочей неделей (Китай) и 30-часовой рабочей неделей (Нидерланды), безусловно, на эту разницу также накладывается различие в часовых поясах.

Сопутствующие риски связаны с ограничениями в коммуникациях команды между собою или с бизнес-заказчиком, они затрудняют определение наиболее продуктивных дней в недельном цикле разработки ПО и недель в годовом цикле. Уменьшить влияние этих рисков можно путем их анализа на различных уровнях и организации следующих корректирующих мер:

- планирование проектных работ с учетом производственных календарей и оптимизация графиков работ;
- управление коммуникациями групп внутри проектной команды, в том числе с помощью формализации типов, времени и каналов коммуникаций;
- гибкое планирование разработки и мониторинга результатов на еженедельной основе.

Организационные особенности и сопутствующие риски

К организационным особенностям относятся:

- управление коммуникациями команды;
- управление персоналом и построение команд разработки;
- административная и техническая поддержка производственных процессов на уровне локальных офисов.

Управление коммуникациями географически распределенной команды является наиболее важной областью проектного управления, обеспечивающей успех проекта. В основе таких коммуникаций лежат электронные каналы: почта, группы в месенджерах (Skype, WhatsUp, Viber), различные собрания команды в формате конференц-звонков. Современное развитие электронных каналов позволяет построить почти любые по сложности коммуникации, отправлять документы и артефакты проекта, отслеживать их получение и обработку. Электронные каналы связи имеют набор неоспоримых достоинств помимо реализации основной функции

передачи информации и получения обратной связи, а именно:

- формализация сообщения, позволяющая отследить ее движение и результаты коммуникаций внутри проекта и с внешними участниками;
- создание дополнительных горизонтальных и вертикальных связей внутри проектной команды, обеспечивающих согласованность отдельных целей участников и групп при реализации общей цели проекта.

Вместе с тем электронные каналы связи имеют и недостатки:

- ◇ дополнительные организационные усилия в микроменеджменте, когда каждый самый небольшой вопрос должен сопровождаться законченной формализованной коммуникацией вплоть до его разрешения;
- ◇ затрудненность получения обратной связи от подчиненных в вертикальных коммуникациях, особенно в ее эмоциональной составляющей;
- ◇ большой процент помех и искажений в групповых формах коммуникаций, особенно в формате конференц-звонков с большой аудиторий участников и на неродном для участников языке.

На текущий момент все географически распределенные команды имеют разветвленные и иногда дублирующиеся в различных инструментах коммуникационные связи.

Наиболее прогрессивной практикой в коммуникациях с помощью электронных каналов являются видеоконференции. Безусловно, они требуют более сложного аппаратного и программного обеспечения, однако:

- позволяют уменьшить помехи и искажения в восприятии вербальной и невербальной информации;
- повышают дисциплину участников коммуникации;
- обеспечивают визуальную обратную связь при обсуждении острых и сложных вопросов.

Все многообразие электронных средств коммуникации должно быть официально закреплено в плане коммуникаций проекта.

Пример из практики автора — это особенности коммуникационного плана в проекте внедрения в российской финансовой группе компаний портального решения для предоставления брокерских и финансовых услуг. Данный портал разрабатывался тремя офисами международной IT-компании, официальный язык проекта был русский, однако часть членов локальных команд совсем не говорила на нем. План коммуникаций проекта не только содержал типичные разделы для такого документа, но и описывал полный перечень и правила использования всех средств электронных коммуникаций.

Для проекта был создан набор групп в системе Skype, проектные и групповые почтовые рассылки, были определены форматы групповых встреч с заказчиком и внутри команд разработки. Члены проектной команды обязательно подтверждали в формате писем все договоренности, достигнутые на встречах или по телефону, использовали общие группы в Skype для проектного взаимодействия и треккинга общих задач, а также почтовые рассылки — для групповых коммуникаций. Отметим, что для миникоманд в локальных офисах создавались отдельные группы в Skype для обсуждения вопросов, ограниченных их собственной зоной ответственности.

Следующая особенность организации деятельности в географически распределенных командах — это нюансы управления персоналом, которые требуют особенного внимания к сопутствующим рискам и подразумевают учет:

— необходимости обеспечения трудовой дисциплины в разных локальных группах одной проектной команды;

— синхронизации наращивания или уменьшения состава локальных групп с необходимым объемом экспертизы по технологиям, предметной области, особенностям проектной работы в каждом конкретном случае;

— работы над развитием командного духа в проекте.

Одним из важных действий, направленным на минимизацию негативного влияния указанных выше факторов, является появление локальных тимлидов в каждой группе или локации проекта. Тимлид — это высококвалифицированный специалист, участвующий в управлении одним из направлений реализации проекта (аналитика, разработка, тестирование и т. п.) в части распределения задач среди специалистов, контроля методов их реализации и мониторинга их исполнения [8]. Такие участники проектных групп могут быть формальными и неформальными, но должны сочетать знание технологий и/или предметной области проекта, личную ответственность, минимальные лидерские качества и достаточный опыт работы в конкретной организации или проекте. Именно локальные тимлиды занимают активную позицию в решении общих проблемных вопросов проекта, отвечают за трудовую дисциплину, наличие необходимой групповой экспертизы в предметной области и технологиях разработки. Первоклассные тимлиды также участвуют в трансляции корпоративных ценностей на проектный уровень. Таким образом, тимлид несет персонализированную ответственность за результаты работы по определенному направлению проекта.

В практике российских компаний довольно часто выбор локальных тимлидов связан с их специальными навыками или опытом работы в организации. Однако без достаточно развитого чувства ответствен-

ности и лидерских качеств такой выбор является ошибочным и ведет к накапливанию серьезных проблем по мере реализации проекта. Пример из личного опыта автора: российский системный интегратор проводил проект в крупном российском банке по кастомизации и внедрению сложного решения в области информационной безопасности. В начале проекта локальным тимлидом в одной из частей команды был назначен один из самых опытных сотрудников в организации, обладающий экспертными знаниями в предметной области и в технологиях разработки, но лишенный лидерских качеств и достаточной ответственности по взятым командой обязательствам. Используемые технологии были абсолютно новыми на российском рынке, а взятые контрактные обязательства не оставляли избыточного времени для различного рода исследований и обучения, в которые команда разработки погрузилась с самого начала проекта. В результате в течение нескольких месяцев эффективные механизмы распределения и мониторинга задач среди разработчиков не были построены, а намеченные контрольные точки в проекте были упущены. Нечеткое понимание уровня ответственности по взятым интегратором обязательствам, в частности, со стороны тимлида, привело к его вынужденной ротации. Замена тимлида в этой части команды потребовала дополнительных организационных усилий, частично демотивировала команду разработки, принесла дополнительные сложности в отношения с заказчиком. Этот пример еще раз подчеркивает необходимость подбора члена команды на такую роль по всей совокупности личных и профессиональных качеств.

Вопрос трудовой дисциплины в географически распределенных командах проекта разработки ПО стоит очень остро: члены команд в небольших офисах, разделенные с руководителями проектов расстоянием, часовыми поясами, языком и ментальностью, во многом по-другому видят правила трудовой дисциплины, в том числе отработку положенных по трудовым контрактам часов и своевременность прихода на работу. Довольно часто начавшуюся тенденцию падения дисциплины в локальных командах трудно остановить, члены проектных команд теряют вовлеченность в проект, приобретают собственные цели, начинают относиться к своим обязанностям формально.

Решение данного проблемного вопроса лежит в следующих плоскостях:

— четкий и последовательный мониторинг выполненных задач;

— частые личные встречи с руководством в различных форматах в локальных офисах, включая рабочие, обучающие, мотивирующие, неформальные;

— контроль трудовой дисциплины с помощью информационных систем учета рабочего времени и благодаря усилиям локальных тимлидов.

Руководитель проекта должен координировать свои усилия с локальными формальными и неформальными тимлидами и повышать их личную заинтересованность в успехе проекта различными методами мотивации. Такие методы мотивации зачастую являются нематериальными, в том числе находятся в области согласования проектных и личных целей участников команды управления проектом [9].

Еще одна важная особенность организации разработки ПО географически распределенными командами — это существенные различия в административной и технической поддержках проектных команд в разных офисах компании. Этот фактор влияет на мотивацию членов проектной команды, оперативность взаимодействия с бизнес-заказчиками и на организацию производственных процессов. В части компаний, опыт которых изучался в рамках данной работы, материально-техническая поддержка центрального офиса (в котором, как правило, работает менеджмент) была существенно выше, чем в локальных офисах, где расположены группы разработки.

Преодоление существенных различий в административной и технической поддержках связано с работой менеджмента в локальных офисах, с работой службы управления персоналом, с усилиями тимлидом в каждом отдельном проекте. По мнению автора, в подавляющем большинстве российских компаний или российских офисах международных компаний эта сторона проблемы не попадает в поле зрения первых лиц компании, поэтому должна решаться локально.

Выводы и рекомендации

В данной работе рассмотрены основные особенности работы географически распределенных проектных команд, связанные с естественными и организационными факторами. Естественные факторы объективны, поэтому сопутствующие им риски необходимо принять и работать над минимизацией их негативного влияния на успех проекта разработки ПО. Организационные факторы и сопутствующие им риски в большей степени индивидуализированы в каждой компании, управление сопутствующими рисками более вариативно и включает возможности по уклонению от рисков. Естественные и организационные особенности оказывают существенное влияние на производственные процессы, изменяют их течение и механизмы проектного управления. Однако эта категория факторов не является предметом рассмотрения в настоящей статье.

Применение производственной функции IT-компании в виде географически распределенных команд — это объективное конкурентное преимущество, позволяющее:

— оптимизировать затраты на фонд заработной платы;

— получить доступ к специалистам с редкой или очень высокой квалификацией;

— быть "ближе" к основным заказчикам или пользователям своих информационных систем.

Вместе с тем практическая реализация такой производственной функции сопряжена не только с большими организационными усилиями, но и с длительной отладкой работы таких проектных команд, на которые влияют рассматриваемые в настоящей статье факторы и сопутствующие им риски.

Учет всего перечня таких факторов должен получать свое отражение в основных проектных документах, включая манифест проекта, рабочий план, оценку рисков, коммуникационный план. Соответствующие сопутствующие риски, планы реагирования и аварийные планы должны быть описаны в плане рисков.

Отдельно обратим внимание на ключевые особенности управления проектами с географически распределенными командами разработки, описанные в данной статье.

- ◇ При планировании работ должны учитываться специфические факторы управления персоналом и построение команд разработки, разница в часовых поясах и производственных календарях.
- ◇ При организации локальных команд необходимо выделить тимлидов и передать им часть ответственности и полномочий в работе с локальными группами разработчиков. Часто обратная связь с рядовыми сотрудниками по электронным каналам затруднена, руководитель проекта может не заметить потерю мотивации или разобщенность целей участников проекта. Именно локальные тимлиды при правильной организации их работы позволяют проекту идти в соответствии с расписанием без ущерба качеству продукта.
- ◇ В планировании коммуникаций должны приниматься во внимание не только естественные особенности вроде часовых поясов, праздников и языкового барьера, но и особенности взаимодействия через электронные каналы, необходимость отслеживания каждой коммуникации, специфические особенности групповых форм коммуникаций в чатах и конференц-звонках.
- ◇ Все механизмы постановки задач и мониторинга их исполнения должны учитывать как накладываемые ограничения (вроде доступности к коммуникациям членов команды), так и возможные характерные причины проблем (низкая трудовая дисциплина, низкая вовлеченность в проект и т. д.).

Отметим также перспективные производственные практики, которые мировые лидеры рынка создания ПО используют в географически распределенной разработке: "удлиненный день разработки" и "непрерывная поддержка решения дневными сменами инженеров". При использовании таких практик в календарные сутки вмещается максимум проектной активности, а различные работы по поддержке решений в промышленной эксплуатации выполняются более эффективными и менее затратными специалистами дневных смен.

Безусловным положительным фактором локальных команд разработки является возможность внесения в разрабатываемые продукты требований локального рынка, если IT-компания занимается на нем также деятельностью по сбыту продукции. Именно распределенные локальные офисы позволили современным лидерам IT-рынка добиться проникновения их решений на все континенты, так как позволили им соответствовать требованиям и ожиданиям каждого нового рынка.

В заключение подчеркнем, что разработка программных продуктов распределенными командами в локальных офисах де-факто становится стандартом отрасли [10]. Поэтому довольно часто в бизнес-планах IT-компаний можно найти отдельный раздел по региональной экспансии в продажах ПО и отдельный раздел по созданию локальных производственных офисов. Это означает необходимость работы с многочисленными рисками, связанными с географически распределенными командами, на различных уровнях, в том числе на уровне каждого проекта.

Список литературы

1. **Ливингстон Д.** Как все начиналось: Apple, PayPal, Yahoo! и еще 20 историй известных стартапов глазами их основателей. М.: Эксмо, 2011. 496 с.
2. **Pashchenko D.** Problem of software development process standardization — comparing CIS and CEE regions // World of New Economy. 2015. Vol. 2. P. 95—100.
3. **Espinosa J., Slaughter S., Kraut R., Herbsleb J.** Team Knowledge and Coordination in Geographically Distributed Software Development//Journal of Management Information Systems. 2007. Vol. 24 (1). P. 135—169.
4. **Cataldo M.** Dependencies in geographically distributed software development: overcoming the limits of modularity. PhD Dissertation Carnegie Mellon University USA, 2007. 188 p.
5. **Massey A. P., Montoya-Weiss M. M., Hung Y.-T.** Because TIme matters: Temporal coordination in global virtual project teams//Journal of Management Information Systems. 2003. Vol. 19, No. 4. P. 129—156.
6. **Russian** software developing industry and software exports, 9-th survey, 2012. P. 61—62. URL: http://www.russoft.ru/files/RUSSOFT_Survey_12_en.pdf
7. **Hofstede G., Hofstede G. J., Minkov M.** Cultures and organizations: Software of the Mind (3rd ed.). New York: McGraw-Hill, 2010.
8. **Kniberg H.** Scrum and XP from the Trenches. InfoQ, 2015.
9. **Уиттакер Дж., Арбон Дж., Кароло Дж.** Как тестируют в Google. СПб.: Питер, 2014. 255 с.
10. **Miranda M.** Leverage geographically-distributed development, 2016. URL: http://betanews.com/2016/01/11/leverage-geographically-distributed-development/

Research in CEE-region: Changes Implementation in Software Production

D. S. Pashchenko, denpas@rambler.ru, SlavaSoft, Moscow, 125368, Russian Federation

Corresponding author:

Pashchenko Denis S., CEO, SlavaSoft, 125368, Moscow, Russian Federation
E-mail: denpas@rambler.ru

Received on October 20, 2016
Accepted on November 21, 2016

Geographically distributed teams in projects of high-tech companies have become the de facto standard of the current business approaches. Market software development in Russia and CIS-region is developing according the global trends; the national IT leaders have a wide network of local offices with development centers. The main goal of this approach is a desire to optimize production costs and increase access to highly qualified human resources. However, projects with geographically distributed development also have a lot of specialties and associated risks that can neutralize the results of economic benefits. In this paper we are summarizing the experience of managers of international software companies with offices in the CIS countries in the management of geographically distributed software development. This experience has been presented to the author in a series of interviews in the autumn of 2016 by senior managers of those software vendors. There is a classification of specialties (and associated risks) on project level; the main focus of the paper is done on the "natural" and "organizational" groups of specialties. For each group a description and the set of major risks are given. In this paper the main approaches to the management of those risks are presented, and also examples of the own experience of the author are given.

Keywords: *geographically distributed software development, software project, software development, advanced software production practices*

For citation:

Pashchenko D. S. Research in CEE-region: Changes Implementation in Software Production, *Programmnaya Ingeneria*, 2017, vol. 8, no. 2, pp. 88—95.

DOI: 10.17587/prin.8.88-95

References

1. **Livingstoun D.** *Kak vse nachinalos': Apple, PayPal, Yahoo! i eshe 20 istori izvestnyh startapov glazami ih osnovateley* (How it's started: Apple, PayPal, Yahoo! And 20 start ups histories by the founders), Moscow, Exmo, 2011, 496 p. (in Russian).

2. **Pashchenko D.** Problem of software development process standardization — comparing CIS and CEE regions, *World of New Economy,* 2015, vol. 2, pp. 95—100.

3. **Espinosa S., Kraut H.** Team Knowledge and Coordination in Geographically Distributed, *Software Development Journal of Management Information Systems*, 2007, vol. 24-1, pp. 135—169.

4. **Cataldo M.** Dependencies in geographically distributed software development: overcoming the limits of modularity, PhD Dissertation Carnegie Mellon University USA, 2007, 188 p.

5. **Massey A. P., Montoya-Weiss M. M., Hung Y**. Because time matters: Temporal coordination in global virtual project teams, *Journal of Management Information Systems*, 2003, vol. 19, no. 4, pp. 129—156.

6. **Russian** software developing industry and software exports, 9-th survey, 2012, pp. 61—62, available at: http://www.russoft.ru/files/RUSSOFT_Survey_12_en.pdf

7. **Hofstede G., Hofstede G., Minkov M.** *Cultures and organizations: Software of the Mind* (3rd ed.), New York, McGraw-Hill, 2010.

8. **Kniberg H.** *Scrum and XP from the Trenches*, InfoQ, 2015.

9. **Witteker J., Arbon J., Karolo J**. *Kak testiruet v Google?* (How do testing software in Google?), Saint Petersburg, Piter, 2014, 255 p. (in Russian).

10. **Mike M.** *Leverage geographically-distributed development*, 2016, available at: http://betanews.com/2016/01/11/leverage-geographically-distributed-development/

Анонс выставок 2017 г.

IT FORUM 2020/ИТ-Джем 2017
12.04.2017 — 14.04.2017
9-й Международный форум
Место проведения: Россия, Нижний Новгород
Тематика: Информационные технологии, коммуникация, связь

Gadget Fair 2017
14.04.2017 — 16.04.2017
Международная выставка
Место проведения: Россия, Москва
Тематика: Информационные технологии, коммуникация, связь

Mobile & Digital Форум 2017
20.04.2017 — 23.04.2017
Международная выставка
Место проведения: Россия, Москва
Тематика: Информационные технологии, коммуникация, связь

EXPO COMM Russia 2017
25.04.2017 — 27.04.2017
Международная выставка и конференция
Место проведения: Россия, Москва
Тематика: Информационные технологии, коммуникация, связь

Связь 2017
25.04.2017 — 28.04.2017
29-я Международная выставка
Место проведения: Россия, Москва
Тематика: Информационные технологии, коммуникация, связь

Positive Hack Says 2017
23.05.2017 — 24.05.2017
7-й Международный форум
Место проведения: Россия, Москва
Тематика: Информационные технологии, коммуникация, связь

ItCOM 2017
12.10.2017 — 14.10.2017
14-я Специализированная выставка-форум
Место проведения: Россия, Красноярск
Тематика: Информационные технологии, коммуникация, связь

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ. ТЕЛЕКОММУНИКАЦИИ. БЕЗОПАСНОСТЬ 2017
01.11.2017 — 03.11.2017
7-я Межрегиональная специализированная выставка
Место проведения: Россия, Якутск
Тематика: Информационные технологии, коммуникация, связь

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

Сдано в набор 02.12.2016 г. Подписано в печать 23.01.2017 г. Формат 60×88 1/8. Заказ PI217
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**