ISSN 2220-3397

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

Пр ИН
9
2018
Том 9

Devprom ALM –

первая российская
программная платформа
для всех стадий
разработки ПО

Devprom ALM
для поддержки учебного процесса
предоставляется бесплатно
на основании академической лицензии,
без ограничения по функциональности
и числу пользователей

Приглашаем к сотрудничеству колледжи и вузы,
обучающие студентов по направлению
"Программная инженерия".
Практические задания, выполняемые
с использованием программной
платформы Devprom ALM,
идеально свяжут теоретические основы,
полученные студентами в рамках курса,
с практическими упражнениями,
выполненными в современной программной среде,
широко используемой в индустрии

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

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

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

СОДЕРЖАНИЕ

Васенин В. А., Лаврищева Е. М., Петренко А. К., Позин Б. А. Научно-технический вклад В. В. Липаева в инженерию разработки надежных и качественных программных систем 387

Вьюкова Н. И., Галатенко В. А., Самборский С. В. Поддержка многоядерного программирования в компиляторе для процессоров Комдив под управлением ОС РВ Багет 393

Елизаров С. Г., Лукьянченко Г. А., Марков Д. С., Монахов А. М., Роганов В. А. Тестирование многоядерных систем на основе идей алгоритма RSA 404

Костенко К. И. Инженерия когнитивного синтеза для систем искусственного интеллекта 415

Пименов И. С., Саломатина Н. В. Распознавание интенций потребителей товаров и услуг в сообщениях пользователей социальных сетей 425

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 и базу данных RSCI на платформе Web of Science.
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. 9
N 9
2018

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

CONTENTS

Vasenin V. A., Lavrischeva E. M., Petrenko A. K., Posin B. A. V. V. Lipaev's Scientific and Technical Contribution to the Development of Reliable and Quality Software Systems 387

V'yukova N. I., Galatenko V. A., Samborskij S. V. Support for Multicore Programming in the Compiler for the Komdiv Processors under the "Baget" RTOS 393

Elizarov S. G., Lukyanchenko G. A., Markov D. S., Monakhov A. M., Roganov V. A. Functional and Performance Validation of Many-Core Systems Based on the Ideas of the RSA Algorithm 404

Kostenko K. I. Engineering of Cognitive Synthesis Morphisms at the Artificial Intelligence Systems 415

Pimenov I. S., Salomatina N. V. Extraction of Explicit Consumer Intentions from Social Network Messages 425

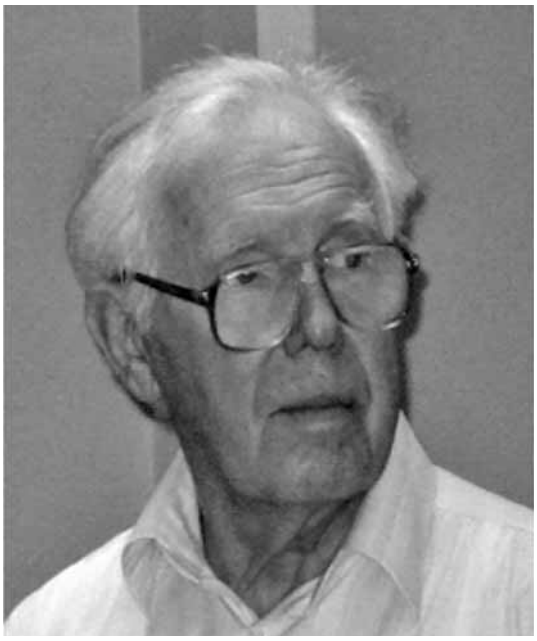
Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

В. А. Васенин[1], д-р физ.-мат. наук, проф., e-mail: vasenin@msu.ru,
Е. М. Лаврищева[2, 3], д-р физ.-мат. наук, проф., гл. науч. сотр., e-mail: lavr@ispras.ru,
А. К. Петренко[1, 2, 4], д-р физ.-мат. наук, проф., e-mail: a.k.petrenko@gmail.com,
Б. А. Позин[4, 5], д-р техн. наук, проф., техн. директор, e-mail: bpozin@ec-leasing.ru,
[1] МГУ имени М. В. Ломоносова,
[2] Институт системного программирования им. В. П. Иванникова РАН, Москва,
[3] Московский физико-технический институт,
[4] НИУ ВШЭ, г. Москва,
[5] ЗАО "ЕС-лизинг", г. Москва

Научно-технический вклад В. В. Липаева в инженерию разработки надежных и качественных программных систем

Статья посвящена памяти одного из основателей инженерии разработки комплексов программ и систем в СССР и в Российской Федерации — Владимира Васильевича Липаева, со дня рождения которого в 2018 г. исполнилось 90 лет. Более 60 лет В. В. Липаев руководил работами по созданию крупных программных комплексов в интересах оборонной промышленности и других стратегически важных отраслей национальной экономики. Одновременно с этим, опираясь на собственный опыт и международные стандарты программной инженерии, он занимался исследованиями и разработкой методологических основ создания и сопровождения больших и сложно организованных программных комплексов на всех этапах их жизненного цикла. В статье представлены краткие сведения о деятельности В. В. Липаева на этих направлениях, о ее результатах и о людях, которые его окружали.

Ключевые слова: *программная инженерия, системы и комплексы программ, проектирование, отладка, тестирование программ, надежность и качество программного обеспечения, стандарты*



Владимир Васильевич Липаев (1928—2015) — профессор, доктор технических наук, ведущий специалист Московского НИИ приборной автоматики (1954—1988 гг.), главный конструктор и председатель координационного совета Министерства радиопромышленности СССР, один из руководителей НТС "Информатизация России" (1988—1994 гг.), главный научный сотрудник Института системного программирования РАН (1995—2015 гг.). Более 30 лет он занимался исследованиями и разработками программного обеспечения для систем военно-промышленного комплекса и технологиями программирования крупных программных комплексов различного назначения. После 1995 г. В. В. Липаев опубликовал серию монографий, учебников и учебных пособий, освещающих базовые методологические принципы создания программных комплексов с учетом отечественного опыта и международных стандартов, а также обеспечения их качества и безопасности. Эти книги заложили основу обучения студентов, аспирантов и начинающих программистов в процессе их подготовки к профессиональной разработке и сопровождению сложных прикладных программных комплексов в промышленности, авиации, медицине и других отраслях экономики.

В 2018 г. исполнилось 90 лет со дня рождения одного из патриархов инженерии программ в СССР и России — Владимира Васильевича Липаева. С начала трудовой и научно-технической деятельности (1954 г.) и до 2015 г. В. В. Липаев руководил разработкой и внедрением программного обеспечения и технических средств различного назначения. В числе таких проектов разработка оборудования для специализированных ЭВМ, комплексов программ в интересах военно-промышленного комплекса (ВПК) страны, создание программных систем общего назначения для вычислительных машин БЭСМ-6, ЕС ЭМ, IBM и др.

На базе накопленного опыта на каждом этапе своей научно-производственной деятельности В. В. Липаев формулировал и передавал в виде публикаций специалистам оригинальные концепции, идеи и методы проектирования, отладки и тестирования, обеспечения надежности и качества программных средств и систем. Результаты, полученные командой программистов, которыми он руководил, представлены в научно-технических монографиях до 1992 г., а затем, после распада СССР, в других публикациях до 2015 г. В первой группе работ представлены научные и прикладные положения, которые определяют базовые фундаментальные и оригинальные методы и средства отечественной технологии программирования для разных видов ЭВМ. Во второй группе публикаций представлены методологические основы и положения программной инженерии с учетом международных стандартов ISO, IEC, IEEE и ГОСТ на всех этапах жизненного цикла программных комплексов. Эти положения определяют подходы к разработке, тестированию, обеспечению безопасности, защиты и качества таких комплексов в соответствии с требованиями заказчика. В публикациях В. В. Липаева широко представлены методы и средства создания, тестирования, испытания, оценки качества и сопровождения программных систем, представляющих интерес для разработчиков в самых разных сферах национального хозяйственного комплекса.

Разработка специализированных ЭВМ и оборудования для ВПК (1960—1992 гг.)

Научно-технические средства и специализированные ЭВМ для ВПК СССР были разработаны по Постановлению Минрадиопрома СССР в Московском НИИ приборной автоматики (МНИИПА) под руководством В. В. Липаева и Б. А. Чичигина. В этот период были созданы такие специализированные ЭВМ, как ПРА-6.0, МАПА, АРГОН, АОУ6 и др., а также военная техника и радиолокационные приборы для наведения на цели противника самолетов, подводных лодок, космических кораблей и др. (Липаев В. В. Фрагменты истории развития отечественного программирования для специализированных ЭВМ в 50—80-е годы. М.: Синтег, 2003. 126 с.).

Характеристика технических средств в ВПК (1946—1970 гг.)

В период 1946—1970 гг. под руководством В. В. Липаева был разработан комплекс программно-технических средств автоматизированного наведения истребителей-перехватчиков на цели противника. В задачу комплекса входила защита границ страны от авиации потенциального противника с помощью отечественной радиолокационной техники, а также устройств, снимающих данные с радиолокаторов о самолетах нарушителя по трассе его движения. Для решения этой задачи требовалось создавать приборы и различное физическое оборудование высокой надежности и качества. Проверку их работоспособности, поиск неисправностей и физических недоработок в конструкции оборудования проводили с использованием вероятностных марковских процессов в теории массового обслуживания и надежности. Такой подход позволил обеспечить высокую надежность, безопасность и качество военной техники, управляющей полетами самолетов в воздушном пространстве для охраны государственных границ. В 1954 г. был разработан макет, который полуавтоматически сопровождал движущуюся цель и давал параметры движения с учетом высоты цели и под углом 45° к плоскости вращения с большой точностью попадания в цель.

Радиолокационные средства входили в состав оборудования военного назначения и обеспечивали:

- обнаружение воздушных целей;
- управление средствами противовоздушной обороны;
- телекоммуникационную передачу данных на командные пункты средств поражения противоракетной техники;
- надежность систем обработки информации в режиме реального времени, а также решение ряда других задач.

Подготовленный макет сначала проверялся на полигоне Монино в 1955 г., затем был изготовлен опытный образец системы "Каскад", впоследствии вошедшей в систему "Воздух 1". Эта система прикрывала территорию СССР во многих пунктах границы и обрабатывала радиолокационную информацию для управления наведением истребителей. Опытный образец такой системы оказался надежным и эффективным, после чего его начали выпускать серийно для передачи в ряд Европейских стран для наведения на цели движущихся летающих объектов и больших кораблей ВМФ (система "Прибой"). Эта система постоянно развивалась и работает до настоящего времени в виде модифицированных приборов в системах СПЛАВ и КРЫМ.

В 1959—1970 гг. в СССР были разработаны алгоритмы и программы системы противовоздушной обороны, включающей радиолокационную обработку информации о траекториях объектов в воздушном пространстве для станции П-20 в системе "Межа" на ЭВМ "Курс-1". Первый образец системы "Межа" был изготовлен и вывезен в Капустин Яр для испытаний в максимально близких к реальным условиях движения воздушных объектов и процессах сбора данных о внешней информации системы ПВО, которые моделировались на М-20 (1965 г.). Система "Межа" выпускалась серийно на Ульяновском заводе до 1987 г. и использовалась на всех пунктах от западных границ СССР до Сахалина.

Технологии создания программных средств для ЭВМ ВПК (1979—1989 гг.)

В апреле 1979 г. Минрадиопром СССР принял решение о создании отраслевой НИР ПРОМЕТЕЙ

(ПРОектирование Математики Единая Технология) для исследования, создания и внедрения автоматизированной технологии разработки программного обеспечения комплексов, работающих в реальном времени на специализированных ЭВМ в оборонной промышленности. Главным конструктором этого проекта был назначен В. В. Липаев. Ставилась задача создать технологии и среду разработки, которая обладала бы высоким качеством, была надежна и безопасна при производстве программ реального времени для систем оборонного назначения. Такая инструментальная среда (система) должна была быть мобильной и применяться на широком классе специализированных и универсальных ЭВМ (БЭСМ, М-20 и др.). Основная цель состояла в том, чтобы на специализированных ЭВМ создать программное обеспечение, основанное на общих принципах разработки, которые используются на универсальных ЭВМ. Для этих целей в рамках ПРОМЕТЕЙ-технологии разрабатывались отдельные системы ЯУЗА, ТЕМП, РУЗА, ДВИНА, ПРОТВА, ОХТА, ПОМПА, ПРА, НАРА и др.

Система ЯУЗА (Липаев В. В., Серебровский Л. В.) для использования на ЭВМ БЭСМ-6 была создана коллективом разработчиков. Она включала 400 тысяч команд кода с трудоемкостью разработки 300 чел.-лет. В этой системе программы описывались на традиционных языках программирования (далее — язык программирования) и на автокоде (языке ассемблер), а процесс обработки программ включал:

- трансляцию программ с трех языков программирования;
- стыковку выходных данных программ с общими переменными, которые передавались между этими программами;
- автоматизированную отладку на уровне языка программирования и тестирование машинных программ на языке ассемблера на множестве контрольных данных в целях обнаружения различных дефектов и отказов в созданном программном обеспечении;
- автоматизированный выпуск документации на все компоненты системы.

Для моделирования и имитации нагрузки на системы, которые создавались на основе инструментальных средств системы ЯУЗА, использовался моделирующий комплекс ЭВМ БЭСМ-6 и АС-6. Система ЯУЗА была изготовлена в 1975 г. и передана во многие организации ВПК для создания на ее основе комплекса программ для управления радиолокационным узлом "Основа". Система ЯУЗА была изготовлена с высоким качеством и проработала почти 20 лет на 13 предприятиях оборонной промышленности более чем на 30 типах ЭВМ. Общий объем программного кода этой системы к 1985 г. составил 5 млн команд. В процессе создания системы ЯУЗА была отработана технология проектирования, тестирования, обеспечения надежности и качества такого класса систем. Общие концепции и принципы этой технологии представлены в ряде книг В. В. Липаева (Проектирование математического обеспечения АСУ, 1977; Надежность программного обеспечения АСУ, 1981; Качество программного обеспечения, 1983; Тестирование программ, 1986 и др.). Эти книги пользовались широким спросом у программистов СССР.

Система РУЗА (Липаев В. В., Штрик А. А.) выполняла функции, аналогичные тем, которые предоставляла ЯУЗА. Однако она имела ряд новых научно-технических идей для машин серии ЕС ЭВМ, бортовой машины АРГОН НИЦЭВТ с системой команд машин ЕС ЭВМ. На этих машинах разрабатывались новые интерпретаторы и кросс-системы, направленные на совершенствование процессов разработки функциональных программ для бортовых ЭВМ ВПК. Были созданы специальные средства автоматизации жизненного цикла систем реального времени, включая процессы разработки, отладки и испытания программ в режиме разделения времени. Полученные в ходе выполнения таких работ результаты опубликованы в работе Каганова Ф. А., Корепанова Б. А., Липаева В. В. и др. Автоматизация проектирования программ для управляющих и микроЭВМ на базе технологической системы РУЗА // Автоматика и телемеханика. 1984. № 7. С. 159—168. Под *системой* в контексте выполнения таких работ понималась совокупность взаимодействующих элементов, работающих совместно для достижения заданных целей. Система РУЗА была иерархически организована. В качестве элементов в ней выступали отдельные подсистемы, каждая из которых функционировала самостоятельно. Однако подсистемы имели возможность связи друг с другом для обмена данными. Так как работа системы зависела от специальной аппаратуры ЭВМ, то формировалась системотехническая технология адаптации отдельных ее модулей, охватывающая отдельные аспекты реализации функций создания и модернизации программных и технических средств подобных систем.

Модули системы специфицировались в специальном макроязыке и в языках программирования типа Автокод, Фортран и др., а данные задавались в форме, принятой для специализированных ЭВМ. Отладку отдельных модулей и подсистем проводили на входном языке методом интерпретации, а тестирование — на машинном языке с помощью наборов тестовых данных, проверяющих правильность выполнения разных функций отдельных модулей и подсистем. Модули комплексировались в подсистемы и систему в целом по методу сборки. Общие данные для взаимодействия модулей и подсистем описывали в специальных модулях — зонах глобальных переменных. Расчет временных характеристик системы осуществляли по графовой модели и таблицам исполнения команд ЭВМ с учетом времени их исполнения и вызываемых данных. В процессе разработки системы формировалась документация на отдельные модули и на весь комплекс программ.

В состав имитационного комплекса внешней среды (движения самолетов и ракет) входили специальные программы. Такие программы обеспечивали гибкую сборку необходимого набора модулей для испытаний отдельных компонентов системы ПВО, а также контроль оборудования, трактов связи, анализа сбоев и устранения их последствий. В модель воздушной обстановки вводились данные о движе-

нии самолетов и ракет, моделировалась траектория их движения с учетом времени и дефектов их обнаружения различными радиолокационными средствами. Результаты моделирования отражались на дисплеях и регистрировались на графопостроителях. Проводилась оценка характеристик качества функционирования системы объектной ЭВМ с использованием выявленных дефектов и ошибок.

Система ПРОТВА (Липаев В. В., Позин Б. А.) предназначалась для создания программ, ориентированных на специализированные ЭВМ А30 и А50, с использованием в качестве технологической базы разработки соответствующих ЕС ЭВМ, а также для сопровождения сложных компьютерных программ. Был представлен типовой технологический процесс такой разработки и сопровождения, прошедший государственные испытания. Кроме основных по функциональному назначению трансляции, загрузки и документирования на технологической ЭВМ, система поддерживала следующие процессы жизненного цикла программного обеспечения, разработанного на языках PL/1 (с ограничениями на коэффициент расширения компилятора по памяти команд) и на ассемблере:

- планирование трудоемкости и длительности создания программного обеспечения системы;
- прогнозирование и оценивание реального состояния качества в зависимости от обнаруженных ошибок;
- оценка ресурсов ЭВМ по памяти и производительности для реализованной системы;
- прогнозирование и оценка качества программных средств, описания сертификата продукта для передачи пользователям бортовых систем.

Вторая и третья версии системы содержали 16 технологических подсистем различного назначения, автоматизирующих процессы создания и сопровождения комплексов программ, в том числе и размещаемых в постоянной и полупостоянной памяти команд.

В систему ПРОТВА был включен и ряд инструментальных средств, которые позволяли использовать ее для разработки прикладных систем различного назначения на ЕС ЭВМ. Для таких комплексов программ можно было применять средства, расширяющие возможности по автоматизации выполнения различных технологических операций и подпроцессов. Концепция повышения производительности труда методами сборочного программирования, т. е. за счет функциональной и структурной унификации разноязыковых комплексов программ и повторного использования однажды написанных программ и библиотек программ в новых программах с учетом ресурсных ограничений, разрабатывалась в рамках задания на НИР ПРОМЕТЕЙ и была реализована в 11 системах, выпущенных Минрадиопромом. Эта работа была дополнена в том числе за счет автоматизации взаимодействия разноязыковых программ без учета таких ограничений. Для сборки подобных систем из разноязыковых модулей, написанных на таких языках, как Fortran, PL/1, Modula, Algol, Cobol, Prolog, использовалась система АПРОП. Эта система соответствовала концепции сборочного программирования, которая в те же годы также разрабатывалась и в Институте кибернетики АН УССР.

Практически полное отсутствие доступа к иностранной литературе требовало издания переводов лучших книг по программной инженерии на русском языке. В. В. Липаев активно участвовал в отборе, рецензировании и редактировании передовых зарубежных изданий. Так, по его рекомендации был издан перевод книги Г. Майерса "Искусство тестирования программ".

В рамках работ по проекту ПРОМЕТЕЙ сформировался следующий порядок проведения разработки и сопровождения программных систем и комплексов программ на разных этапах их жизненного цикла:

- определение целей и задач на разработку системы в техническом задании;
- проектирование структуры системы и разработка отдельных ее элементов;
- тестирование элементов и системы в целом;
- комплексирование (сборка) модулей в систему;
- испытания отдельных модулей и системы, составленной из этих модулей, на адекватном множестве данных и тестов;
- оценка надежности и разработка сертификата качества для передачи готового продукта заказчикам и потребителям;
- эксплуатация системы и ее сопровождение;
- снятие системы с эксплуатации (утилизация).

Каждая система автоматизированной разработки программного обеспечения, созданная в рамках программы ПРОМЕТЕЙ, сдавалась в Фонды алгоритмов и программ для широкого распространения среди потребителей из разных сфер национального хозяйственного комплекса. Основные разработчики программы ПРОМЕТЕЙ, комплексов ЯУЗА, РУЗА и ПРОТВА были награждены премией Совета Министров СССР (1985 г.).

Сборочное программирование, реализованное в рамках проектов под руководством В. В. Липаева, поддерживали многие научные коллективы в области программирования (например, Ершов А. П. Опыт интегрального подхода к актуальной проблематике ПО // Кибернетика. 1984. № 1).

Задачи информатизации России (1990—1994 гг.)

После распада СССР научно-технические разработки разных организаций России по технологии программирования сложных программных комплексов и систем обсуждались на научно-техническом Совете "Информатизация России" (1992 г.). В Совет входили 30 ведущих специалистов из различных институтов РАН и других научно-исследовательских организаций России. Целью деятельности Совета было создание новых фундаментальных и технических основ информатизации России. Этим Советом руководили В. В. Липаев, В. П. Иванников, В. К. Левин и др. Советом был подготовлен отчет "Научно-технические основы информатизации России" (Москва, 1992. 151 с.).

К основам информатизации страны в этом отчете были отнесены следующие положения.

1. Экономические и социальные задачи информатизации.

2. Технико-экономическое развитие рыночной экономики информатизации.

3. Научно-техническое обеспечение информатизации России, которое включало:

3.1. фундаментальные и методологические исследования и разработки информационных технологий и систем;

3.2. технологию и инженерию разработки программных систем;

3.3. обеспечение защиты информации;

3.4. стандартизацию программно-технических средств для создания систем;

3.5. концепцию отбора конкурсных проектов (РФФИ);

3.6. поддержку процессов информатизации системой высшего образования.

4. Развитие инфраструктуры информатизации России.

5. Системный проект информатизации России.

Основные положения научно-технического обеспечения информатизации страны были разработаны:

по п. 3.1 — Фундаментальные и прикладные методологические исследования и разработки информационных технологий и систем (В. П. Иванников);

по п. 3.2 — Технология проектирования программных средств (В. В. Липаев и А. А. Штрик).

В п. 3.1 (В. П. Иванников) были озвучены следующие направления исследований, которые предполагалось выполнить в ближайшие годы:

- алгебраические и логические теории формальных систем;
- анализ параллельных и распределенных систем, языки, логика, модели;
- формальные спецификации и верификация сложных систем;
- парадигмы программирования;
- формализмы представления знаний;
- модели человеко-машинных интерфейсов;
- визуализация в динамических системах и виртуальной реальности.

Важнейшим направлением по информационным технологиям значилось системное программирование, обеспечивающее интерфейс с аппаратурой вычислительных средств и систем. Оно включает: языки программирования и компиляторы; операционные системы; системы управления данными и знаниями; технологические системы поддержки процессов разработки больших программных комплексов; открытые системы, которые, по замыслу авторов, должны базироваться на международных стандартах и обеспечивать совместную работу с другими компонентами.

По п. 3.2 была представлена необходимость решения трех задач:

1) достижение мирового уровня разработок программных систем, выполнение в этой области требований мировых стандартов, введение системы аттестации и сертификации программных продуктов, реализация экономических стимулов создания высококачественных систем;

2) формирование рынка программных систем и услуг по их разработке и сопровождению, поиск источников финансирования и вложение инвестиций в рыночную структуру программного обеспечения даже при отсутствии быстрой прибыли;

3) определение основ программной инженерии как самостоятельной и прибыльной сферы деятельности, сочетающей инвестиции в нее как государственного, так и частного секторов экономики.

Фундаментальные исследования при этом должны были, по замыслу авторов, иметь достаточное для их реализации финансирование. Выход из кризиса планировалось осуществлять путем повышения производительности, применения новых CASE и технологий многократного повторного использования. Многие системы при этом должны были, по мнению авторов, развиваться в виде версий, которые следует использовать длительное время. В целях реализации этих положений было предложено:

- разрабатывать базовые типовые проблемно-ориентированные технологии для основных классов прикладных систем, которые описываются на языках Паскаль, Си, Ада и др.;
- проводить исследования и разработки методов повышения производительности систем на основе сборочного программирования, основанного на принципах многократного использования компонентов четвертого поколения;
- повышать качество создаваемых систем на основе совершенствования процессов разработки и гарантии качества, процедур аттестации и сертификации программных продуктов;
- внедрять перспективные технологии на базе искусственного интеллекта и автоматической генерации машинных программ с языков программирования.

Таким образом, по технологии проектирования программных средств в проекте информатизации России были предложены инфраструктурные решения, которые включали: систему вычислительных средств и сетей для взаимодействия информационных объектов и технологий; программные средства функционирования аппаратных комплексов, информационных и программных средств, баз данных и систем управления ими; систему образования для подготовки национальных кадров, способных решать поставленные задачи. Разработка информационно-вычислительных технологий и систем, согласно этому проекту, должна была основываться на базовых положениях международных стандартов Computer Science по безопасности, надежности и качеству создаваемых комплексов программ и систем.

В следующем отчете этого Совета, который назывался "Научно-технические аспекты развития безопасности и защиты информационных технологий" (Липаев В. В., Левин В. К., Михайлов С. Ф. и др. 1992 г.), давалось описание: математических моделей защиты информации в базах данных и комплексах программ, в каналах передачи данных и в узлах сети; методологии защиты от угроз, атак и несанкционированного доступа; способов обеспечения безопасности информационных технологий и программных систем.

Результатом деятельности Совета явилась реорганизация научных структур, сферы обучения и производства компьютерных систем в целях создания инфраструктуры информатизации в России, обеспечивающей научно-технический прогресс в науке, технике и промышленности. Как один из шагов на этом направлении в 1994 г. был создан Институт системного программирования РАН во главе с В. П. Иванниковым. В 1995 г. в этом институте В. В. Липаев начал работать

в качестве главного научного сотрудника. В его задачи входили исследования в области информатизации, касающиеся программной инженерии, а также вопросы обучения специалистов программной инженерии.

Программная инженерия. Основы стандартизации процессов разработки качественных систем (1995—2015 гг.)

В качестве главной своей задачи В. В. Липаев рассматривал совершенствование отечественной технологии создания сложных и масштабных комплексов программ, формирование методических основ для освоения этой дисциплины студентами в вузах страны, а также развитие ранее созданной в СССР технологии программирования с учетом развивающихся в мире стандартов Computer Science.

За рубежом в 1968 г. на конференции НАТО была определена Software Engineering — Программная инженерия, представляющая собой систему методов и средств планирования и разработки, эксплуатации и сопровождения программного обеспечения. С этих позиций программная инженерия, как сфера деятельности и академическая дисциплина, ориентирована на разработку прикладных и информационных систем разного назначения, которые предназначены для использования во всех областях человеческой деятельности, включая хозяйственную, промышленно-производственную, финансовую и др.

Международные стандарты ISO/IEC, IEEE рассматривают программную инженерию как базовую инженерную дисциплину. С позиции стандартов (Процессы и стандарты жизненного цикла сложных программных средств. Справочник. М.: СИНТЕГ. 2006 г.) В. В. Липаев пересмотрел и с учетом положений этих стандартов уточнил ранее сформированные концепции, принципы и методы по технологии программирования комплексов для ВПК. Он переписал ряд монографий по технологии разработки крупных программных систем, по обеспечению их надежности и качества. В них он с позиций международных и отечественных стандартов разъяснял различные аспекты технологии программирования, способы реализации функциональных возможностей, технологии обеспечения безопасности, надежности и качества программных средств и систем. В основу учебных курсов по программной инженерии им был положен свод знаний программной инженерии SWEBOK (2001—2014 гг.) и стандарт по процессам жизненного цикла по ISO/IEC 12207—2007. К основным монографиям и учебникам, написанным В. В. Липаевым по программной инженерии в этот период, относятся следующие:

- Липаев В. В. Качество программных систем. М.: Методика, 2002;
- Липаев В. В. Программная инженерия. Методологические основы. М.: ГУ-ВШЭ, 2006;
- Липаев В. В. Отечественная программная инженерия: фрагменты истории и проблемы. М.: СИНТЕГ, 2007;
- Липаев В. В. Тестирование крупных комплексов программ на соответствие требованиям. Учебник. М.: Глобус, 2008;
- Липаев В. В. Экономика производства сложных программных продуктов. М.: СИНТЕГ, 2008;
- Липаев В. В. Программная инженерия заказных программных продуктов. Учебник, М., 2014;
- Липаев В. В., Позднеев Б. М. Электронный учебно-методический комплекс "Программная инженерия". М., 2012.

В этих монографиях и учебниках В. В. Липаевым даны определения основных положений программной инженерии с учетом российского опыта и международных стандартов ISO/IEC.

Наряду с развитием методов и инструментальных средств, нацеленных на повышение качества программ, в последние годы все более серьезное внимание в стране уделяется развитию и внедрению стандартов обеспечения информационной безопасности. Работы В. В. Липаева по анализу и популяризации этой группы стандартов в настоящее время приобретают особое звучание. Задел, который был создан В. В. Липаевым в этом направлении, актуален и сейчас.

Дополнительной возможностью для пропаганды своих идей, принципов и положений инженерии программ стал для В. В. Липаева основанный в 2009 г. журнал "Программная инженерия". С первых дней существования журнала В. В. Липаев вошел в состав редакционной коллегии и со свойственной ему энергией включился в совершенствование его тематики, в редактирование поступающих в журнал материалов, в подготовку своих статей. За 2009—2015 гг. им было подготовлено и опубликовано в "Программной инженерии" 15 статей. Последнюю из запланированных им статей он закончить не смог по состоянию здоровья. Она, доработанная по его рукописи уже после смерти, была издана в феврале 2017 г. (Липаев В. В. К разработке моделей динамической внешней среды для испытаний сложных программных продуктов // Программная инженерия. 2017. Т. 8, № 2. С. 51—57). Этот факт — еще одно яркое свидетельство преданности В. В. Липаева своей профессии, верности стране и делу, которому он без остатка посвятил всю свою жизнь.

* * *

Научно-техническое наследие В. В. Липаева — работающие в стране системы, монографии, учебники и учебные пособия по проектированию, реализации и обеспечению надежности и качества комплексов программ.

В последних работах В. В. Липаева представлен ряд концепций и идей на перспективу. В их числе разработка и реализация функциональных требований к сложным заказным программным продуктам на всех этапах их жизненного цикла, проведение работ по обеспечению надежности, качества и безопасности современных программных систем и данных. Работы В. В. Липаева по анализу и популяризации этой группы требований в настоящее время приобретают особое звучание.

Владимир Васильевич Липаев — незаурядная и многогранная личность, настоящий патриот своей страны, оставивший глубокий и яркий след в отечественной программной инженерии. Его соратники по работе и ученики, настоящие и будущие специалисты в области создания больших и сложноорганизованных программных систем всегда будут хранить память об этом замечательном человеке.

УДК 004.4'23 DOI: 10.17587/prin.9.393-403

Н. И. Вьюкова, ст. науч. сотр., e-mail: niva@niisi.ras.ru, **В. А. Галатенко,** д-р физ.-мат. наук, ст. науч. сотр., e-mail: galat@niisi.ras.ru, **С. В. Самборский,** ст. науч. сотр., e-mail: sambor@niisi.ras.ru, Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук", Москва

Поддержка многоядерного программирования в компиляторе для процессоров Комдив под управлением ОС РВ Багет

Представлен обзор аппаратных средств поддержки параллелизма в процессорах Комдив, а также программного обеспечения для эффективного использования этих средств при разработке приложений. Основное содержание статьи посвящено поддержке параллельного и конкурентного программирования в компиляторе с языков Си и Си++ для процессоров Комдив под управлением операционной системы реального времени Багет.

Ключевые слова: *процессор Комдив, параллельное и конкурентное программирование, Си++, OpenMP*

Введение

Создание отечественных микропроцессоров, обеспечивающих высокую производительность на инженерных и научно-технических вычислениях, является важной предпосылкой для развития IT-отрасли, а также других высокотехнологичных отраслей экономики нашей страны [1]. В ФГУ ФНЦ НИИСИ РАН за последние 20 лет разработано семейство микропроцессоров архитектуры Комдив, близкой к архитектуре MIPS, и налажено производство элементной базы для них. Микропроцессоры Комдив первого поколения предназначались преимущественно для встраиваемых систем и систем управления технологическими процессами, в том числе для аэрокосмической отрасли. В последние годы ведется работа по созданию на базе микропроцессоров Комдив вычислительных средств различных классов (серверы, рабочие станции, планшеты, маршрутизаторы) для широкого спектра применений. При этом особое внимание уделяется обеспечению таких характеристик, как надежность, доверенность [2], пригодность к эксплуатации в жестких условиях, высокая производительность, оптимальное соотношение между производительностью и энергопотреблением [3].

Производительность зависит от тактовой частоты процессора, а также от возможностей параллельного исполнения операций. Возможности наращивания тактовой частоты имеют технологические ограничения, поэтому в архитектуре микропроцессоров Комдив применяют различные виды параллелизма: суперскалярность, многоядерность, специализированные сопроцессоры, ориентированные на решение определенных классов задач. Разрабатываемый в настоящее время процессор 1890ВМ118 [2] имеет два ядра, которые могут функционировать в режиме симметричного доступа к памяти. Для обеспечения высокой производительности также используются многопроцессорные комплексы с высокоскоростными соединениями посредством сети RapidIO.

Наличие аппаратного параллелизма само по себе не приводит автоматически к увеличению производительности прикладных программ. Для этого необходимы программные инструментальные средства, библиотеки, компиляторы с языков высокого уровня, позволяющие с разумными трудозатратами создавать высокоэффективное программное обеспечение (ПО), использующее аппаратный параллелизм. Поэтому в ФГУ ФНЦ НИИСИ РАН параллельно с развитием архитектуры процессоров Комдив ведется создание и портирование прикладных библиотек и инструментальных средств разработки ПО. В разд. 1 настоящей статьи приведен краткий обзор средств поддержки аппаратного параллелизма процессоров Комдив и существующего программного обеспечения для эффективного использования этих средств. Основной материал статьи (разд. 2, 3) посвящен вопросам поддержки программирования для многоядерных систем с симметричной памятью в разрабатываемом в настоящее время компиляторе с языков Си и Си++ для процессоров Комдив под управлением операционной системы реального времени (ОС РВ) Багет. В заключении приводятся выводы о результатах работы и сформулированы направления дальнейших исследований и разработок.

1. Программные средства поддержки параллелизма процессоров Комдив

В этом разделе представлен обзор программных инструментальных средств и библиотек, применя-

емых при разработке ПО для процессоров Комдив. Рассмотрены средства программирования для сопроцессора цифровой обработки сигналов СР2, для сопроцессора векторной и комплексной арифметики CPV, а также для многопроцессорных комплексов.

Сопроцессор цифровой обработки сигналов СР2 представляет собой SIMD-устройство, ориентированное на выполнение задач цифровой обработки сигналов. Сопроцессорами СР2 оснащены процессор Комдив128-РИО и двухъядерный процессор Комдив128-М. Пиковая производительность сопроцессора СР2 составляет 8 Гфлопс вещественных операций одинарной точности для тактовой частоты 200 МГц.

Сопроцессор содержит одну управляющую секцию со своей памятью команд и четыре вычислительных секции, каждая из которых имеет свою память данных. Вычислительные команды выполняются одновременно во всех четырех секциях. Для обмена данными между СР2 и системной памятью процессора используется канал DMA.

Сопроцессор СР2 поддерживает параллелизм как на уровне команд (VLIW, широкое командное слово), так и SIMD-параллелизм, позволяя обрабатывать одновременно четыре потока данных. Поддерживаются вычисления с вещественными и комплексными числами одинарной точности. Система команд СР2 включает инструкции (такие как комплексное умножение с накоплением и вычитанием), которые совмещают выполнение до 10 операций. Все это создает высокий потенциал производительности, однако его использование при создании прикладных программ сопряжено со значительными трудностями.

Особенности архитектуры сопроцессора и его система команд (отсутствие ветвлений, ограниченный аппаратный стек) не позволяют реализовать для него компиляторы с языков высокого уровня, поэтому программы для СР2 разрабатываются на ассемблере. При этом необходимо учитывать особенности СР2, такие как отсутствие задержек по неготовности данных, возможные конфликты при попытке записи значения двумя командами в один регистр на одном и том же такте, а также ряд других, подробное описание которых можно найти в работе [4]. Определенные сложности представляет также организация взаимодействия между процессором и сопроцессором при пересылке входных данных и возврате результатов. Подробное описание методики программирования для СР2 представлено в работе [5]. В работе [6] описаны методы оптимизации программ для СР2. В работе [7] представлен экспериментальный инструмент автоматической конвейеризации циклов в программах для СР2.

Для создания приложений, использующих сопроцессор СР2, разработаны библиотеки прикладных программ для этого сопроцессора. В частности, разработана библиотека цифровой обработки сигналов (БЦОС) [8], содержащая базовые функции работы с СР2, включая инициализацию СР2, работу с памятью, копирование, инициализацию и преобразование данных, а также ряд основных операций линейной алгебры (сложение и умножение векторов и матриц, скалярное произведение, транспонирование матриц и т. п.), операции обработки сигналов (быстрое преобразование Фурье, свертки). Помимо этого создаются и пополняются специализированные библиотеки композитных функций, ориентированных на выполнение различных прикладных задач.

Сопроцессор векторной и комплексной арифметики CPV. Впервые сопроцессор векторной и комплексной арифметики был реализован в микропроцессоре Комдив64-РИО [9]. Он поддерживал операции над значениями, состоящими из двух вещественных компонент одинарной точности. Усовершенствованный вариант сопроцессора векторной и комплексной арифметики (CPV) был в дальнейшем включен в архитектуру процессоров 1890ВМ8Я и 1890ВМ118. Сопроцессор CPV, ориентированный на обработку изображений и сигналов, выполняет операции над векторами шириной 128 разрядов, содержащими 4 вещественных значения одинарной точности или 2 значения двойной точности. В отличие от сопроцессора СР2, программирование для этого сопроцессора возможно не только на языке ассемблера, но и на языках Си, Си++.

В Си-компиляторе для процессоров Комдив, разработанном в НИИСИ РАН, поддерживается автовекторизация циклов для сопроцессора CPV. Согласно результатам измерений [10] для трех алгоритмов из пакета NAS Parallel Benchmarks автовекторизация дает прирост производительности примерно 25 %.

Более эффективного использования сопроцессора CPV можно добиться применением векторных расширений языка Си, поддерживаемых компилятором. Использование встроенных (*intrinsic*) функций позволяет включить в программу на языке Си любую команду, поддерживаемую сопроцессором.

Инструментальные средства директивной векторизации циклов на уровне исходного кода [11] позволяют расширить возможности автовекторизации с помощью использования директив, которые могут включать параметры выравнивания данных, а также спецификации операций редукции.

Сопроцессор CPV также может быть использован при разработке ПО путем применения оптимизированных версий прикладных библиотек и алгоритмов [12, 13].

Многопроцессорные комплексы. Особенностью архитектуры Комдив является наличие в процессоре контроллеров и коммутатора высокоскоростных каналов RapidIO. Такая архитектура позволяет создавать высокопроизводительные мультипроцессорные вычислительные комплексы, включающие до нескольких сотен процессоров. Подобные комплексы успешно применяют для решения задач цифровой обработки сигналов и других научно-технических вычислений [14].

Основой создания приложений для многопроцессорных комплексов является разработанная в НИИСИ РАН библиотека параллельной обработ-

ки сигналов (БПОС) [15]. Библиотека обеспечивает унифицированное окружение для программ цифровой обработки сигналов, выполняемых на многопроцессорных вычислительных комплексах. Она может применяться как для комплексов, состоящих из нескольких процессоров, так и для систем из сотен процессоров.

Библиотека БПОС предоставляет разработчику программные интерфейсы на языке Си, скрывающие особенности аппаратуры и используемой операционной системы, а также особенности другого общего программного обеспечения. Разработанная на базе БПОС программа может быть без изменений запущена как на эмуляторе, работающем на инструментальной ЭВМ под управлением ОС общего назначения, так и на целевом многопроцессорном комплексе, функционирующем под управлением ОС РВ.

Для конфигурирования высокоскоростных каналов (ВСК) передачи данных, посредством которых осуществляется взаимодействие между процессорами, применяется пакет поддержки, включающий конфигуратор ВСК и библиотеку ВСК [16].

Пример использования многопроцессорного комплекса для реализации алгоритма MG (многосеточный метод) из набора тестов NBP рассмотрен в работе [14]. Вопросы совместного использования сопроцессоров CP2, CPV и параллельных вычислений на многопроцессорных комплексах на примере реализации трех алгоритмов из набора тестов NPB рассмотрены в работе [10].

2. Поддержка параллельного и конкурентного программирования в компиляторе для процессоров Комдив

Разрабатываемый в настоящее время процессор 1890ВМ118 является первым в линейке двухъядерных процессоров Комдив, в котором будет поддерживаться режим симметричного доступа к памяти. В связи с этим обстоятельством была поставлена задача обеспечить в кросс-компиляторе для процессоров Комдив, функционирующих под управлением ОС РВ Багет, поддержку средств параллельного и конкурентного программирования, присутствующих в стандартах языка C++ версии 11 и более поздних, а также в стандарте прикладного программного интерфейса (ППИ) OpenMP [17].

В настоящей статье рассматривается применение следующих средств [18]:

- библиотека потоков <thread>;
- библиотека атомарных типов <atomic>;
- библиотека асинхронного запуска задач <future>;
- директивы распараллеливания, атомарных операций и запуска задач стандарта OpenMP;
- экспериментальная реализация параллельной версии библиотеки алгоритмов Си++.

Кросс-компилятор разрабатывается на основе инфраструктуры GCC (Gnu Compiler Collection) версии 7.3, сконфигурированной для целевой платформы "архитектура Комдив/ОС РВ Багет". При сборке стандартной библиотеки шаблонов языка Си++ (*Standard Template Library*, STL) использовалась реализация стандартной библиотеки языка Си, поставляемая в составе ОС РВ Багет.

Для тестирования перечисленных выше средств параллельного и конкурентного программирования были использованы наборы тестов, поставляемые с исходными текстами GCC. В дальнейшем планируется также провести замеры производительности с использованием набора тестов [19].

В настоящей работе представлены результаты тестирования и сравнение производительности для небольшого набора тестов, использующих различные последовательные и параллельные реализации нескольких простых алгоритмов. Цели проводимого тестирования:

- проверка корректности взаимодействия между реализациями библиотек STL и libgomp (библиотека времени выполнения для ППИ OpenMP), с одной стороны, и реализацией стандартной библиотеки языка Си, предоставляемой ОС РВ Багет, с другой стороны;
- выявление специфики работы параллельных программ в среде ОС РВ Багет;
- предварительная оценка эффективности распараллеливания программ для двухъядерных процессоров 1890ВМ118.

Поскольку процессор 1890ВМ118 в настоящее время находится в стадии тестирования и отладки, и версия ОС РВ Багет, поддерживающая параллелизм многоядерных процессоров с симметричной памятью, еще не выпущена, то тестирование проводилось на следующих платформах:

- одноядерный процессор Комдив (1890ВМ8Я) под управлением ОС РВ Багет;
- эмуляция двухъядерного процессора Комдив, функционирующего под управлением ОС Linux, с помощью эмулятора vmips;
- ЭВМ архитектуры ×86_64 с 24 ядрами под управлением ОС Linux (Fedora Core 25).

3. Результаты тестов

В этом разделе представлены описания тестовых задач, которые были разработаны для оценки работоспособности средств распараллеливания в компиляторе и оценок потенциального прироста производительности при использовании многоядерных процессоров Комдив под управлением ОС РВ Багет.

3.1. Набор тестовых задач

В работе были использованы три перечисленные далее группы небольших тестовых задач.

1. Задачи, использующие атомарные операции. В эту группу вошли две задачи: в первой из них (Atomic) число атомарных операций невелико относительно общего объема вычислений, во второй (Atomic1) — атомарные операции применяются более интенсивно.

2. Вычислительные задачи обработки массивов. В эту группу вошли две задачи (Calc, Calc1) вычислений с массивами, отличающиеся шаблонами доступа к памяти.

3. Задача слияния двух упорядоченных массивов (Merge).

Все перечисленные задачи разрабатывались только для экспериментов с целью сравнения разных подходов к распараллеливанию. Следует однако отметить, что ни одна из них, за исключением задачи слияния, не имеет практического смысла.

Для каждой задачи сравниваются несколько альтернативных программных реализаций на языке Си++: последовательная реализация, не использующая многопоточность, параллельная реализация с использованием OpenMP и аналогичная параллельная реализация с применением стандартной библиотеки потоков <thread> или библиотеки <future>.

Для задачи Calc1 рассмотрены также реализации с блочной группировкой вычислений для улучшения локальности данных и несколько реализаций с применением функции `transform` из стандартной библиотеки алгоритмов.

Для задачи Merge помимо простейшего последовательного алгоритма рассмотрены алгоритм бинарного слияния, несколько вариантов параллельных алгоритмов с использованием OpenMP и асинхронного запуска задач с применением библиотеки <future>, а также использование функции `merge` из STL.

Программы пропускались на каждой из трех перечисленных в предыдущем разделе платформ с варьированием числа потоков выполнения: 1, 2, 4, а для платформы ×86_64/Linux число потоков варьировалось до 16: 1, 2, 4, 8, 16. Размеры массивов и число повторов циклов в тестовых задачах подбирали индивидуально для каждой платформы. При анализе результатов тестирования оценивали изменение времени выполнения программы относительно последовательной реализации той же задачи на той же платформе.

3.2. Задачи, использующие атомарные операции

В этом подразделе представлены результаты тестовых задач Atomic и Atomic1, в параллельных реализациях которых применяются атомарные операции.

Задача Atomic. Последовательная реализация задачи представлена на листинге 1.

Листинг 1. Последовательная реализация задачи Atomic

```
int *res;
...
res = new int[N + 1]();
...
for (int i=1; i <= N; i++) {
   for (int j=1; j <= N; j++)
     if (i%j==0)
        res[j]++;
}
```

Параллельная реализация с использованием OpenMP отличается от последовательной наличием директив `#pragma omp` (Листинг 2).

Листинг 2. Параллельная реализация задачи Atomic средствами OpenMP

```
#pragma omp parallel for
for (int i=1; i <= N; i++) {
   for (int j=1; j <= N; j++)
      if (i%j==0) {
      #pragma omp atomic update
      res[j]++;
      }
}
```

В параллельной реализации с использованием библиотеки потоков <thread> осуществляется запуск потоков, где `n`-й поток вычисляет свою часть внешнего цикла, для `i` от `N*(n-1)/Nth+1` до `N*n/Nth`. Для обеспечения атомарности операции `res[j]++` тип переменной `res` объявлен как массив атомарных значений: `atomic<int> *res`.

В табл. 1 приведены результаты измерений времени выполнения (в секундах) на платформах 1890ВМ8Я/ОС РВ Багет (1 ядро), Комдив/Linux на vmips (2 ядра), ×86_64/Linux (24 ядра). В строке **Seq** показано время работы последовательной реализации. В строке **Par** указано число потоков, используемое в параллельных реализациях. В строках **OpenMP** и **<thread>** показано время выполнения для параллельных реализаций средствами OpenMP и средствами библиотеки C++ <thread> для разного числа потоков.

Таблица 1

Результаты измерений для задачи Atomic, с

Платформа	1890ВМ8Я/Багет, 1 ядро			Комдив/Linux на vmips, 2 ядра			×86_64/Linux, 24 ядра				
Seq	17,377			2,8960			24,590				
Par	**1**	**2**	**4**	**1**	**2**	**4**	**1**	**2**	**4**	**8**	**16**
OpenMP	12,395	12,395	12,395	2,5130	1,2611	1,2860	24,448	12,390	6,3914	3,3303	2,3649
<thread>	12,418	12,433	12,479	2,5744	1,3142	1,3317	24,626	12,631	6,4352	3,2734	2,3072

Из табл. 1 видно, что для платформы 1890ВМ8Я/ОС РВ Багет время работы последовательной реализации превышает время работы параллельных реализаций с числом потоков 1. Это связано со спецификой аппаратного предсказания переходов в процессоре 1890ВМ8Я. Последовательная реализация может быть ускорена путем применения встроенной функции компилятора `__builtin_expect` в условном операторе: `if (__builtin_expect(i%j==0, 0))`; в этом случае компилятор будет генерировать код с учетом того, что вероятность выполнения условия `i%j==0` невелика. Более радикальный способ улучшить предсказание переходов — применение оптимизации по профилю, однако для этого необходима поддержка сбора соответствующего профиля в ОС РВ Багет.

На рис. 1 показано время выполнения параллельных реализаций для разного числа потоков, нормированное по времени работы последовательной реализации, для платформ Комдив/Linux на vmips и ×86_64/Linux. Значение переменной Nth на этом и на всех последующих рисунках соответствует числу используемых потоков. Производительность возрастает приблизительно пропорционально числу используемых потоков (пока оно не превышает число ядер). Эффективность реализаций средствами библиотеки потоков и средствами OpenMP практически одинакова.

Задача Atomic1. Эта задача отличается от предыдущей главным образом тем, что атомарные операции инкрементации значений в памяти используются здесь значительно более интенсивно. Последовательная реализация задачи представлена на листинге 3.

Листинг 3. Последовательная реализация задачи Atomic1

```
int *res;
...
res = new int[N+1]();
...
for (int i=1; i <= N; i++) {
   for (int j=1; j<= N; j++)
      if (j<=i)
         res[j]++;
}
```

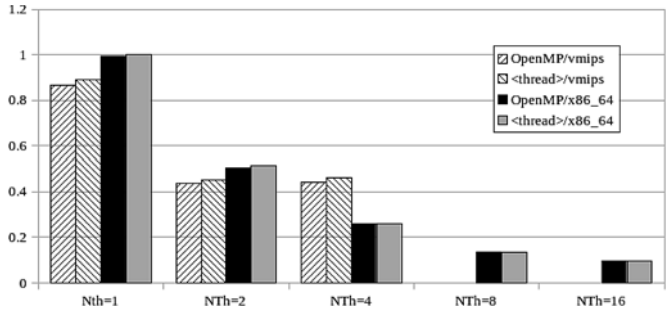


Рис. 1. Время работы параллельных реализаций задачи Atomic, нормированное по времени работы последовательной реализации

Параллельные реализации этой задачи с использованием OpenMP и библиотеки потоков аналогичны соответствующим реализациям для предыдущей задачи, но в директиве `#pragma omp parallel for` был использован параметр планирования `schedule(guided, (N/(Nth*8)))`.

В табл. 2 приведены результаты измерений времени выполнения (в секундах) на трех платформах. Данные в табл. 2 организованы так же, как и в табл. 1. Можно заметить, что время выполнения параллельных реализаций с одним потоком значительно превышает время выполнения последовательной реализации. Это объясняется тем обстоятельством, что атомарные операции `res[j]++` существенно дороже неатомарных. Для платформы ×86_64 время выполнения однопоточного варианта параллельных реализаций почти в 5 раз больше, чем время выполнения последовательной реализации. Здесь отметим, что даже при 8 потоках программа работает медленнее, чем последовательная. Для процессора 1980ВМ8Я коэффициент замедления вследствие использования атомарных операций значительно ниже, примерно 2.1.

На рис. 2 показано время выполнения параллельных реализаций задачи Atomic1, нормированное по времени работы для последовательной реализации, для платформ Комдив/Linux на vmips и ×86_64/Linux. Для этой задачи распараллеливание средствами OpenMP несколько выигрывает по сравнению с распараллеливанием с помощью библиотеки потоков благодаря применению подходящего параметра планирования. С ростом числа потоков выполнение ускоряется, однако ускорение не пропорционально числу потоков. Этот факт объясняется тем, что потоки конфликтуют при выполнении атомарных операций.

Таблица 2

Результаты измерений для задачи Atomic1, с

Платформа	1890ВМ8Я/Багет, 1 ядро			Комдив/Linux на vmips, 2 ядра			×86_64/Linux, 24 ядра				
Seq	8,7144			2,6845			5,18458				
Par	1	2	4	1	2	4	1	2	4	8	16
OpenMP	18,244	18,244	18,244	3,7450	2,03537	1,9311	26,014	14,496	7,6712	4,2269	3,1651
<thread>	18,286	18,316	18,335	3,7554	2,3819	2,0471	26,165	19,232	11,536	6,4861	3,6916

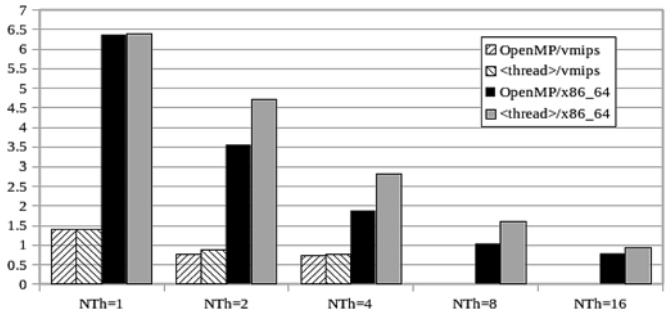


Рис. 2. Время работы параллельных реализаций задачи Atomic1, нормированное по времени работы последовательной реализации

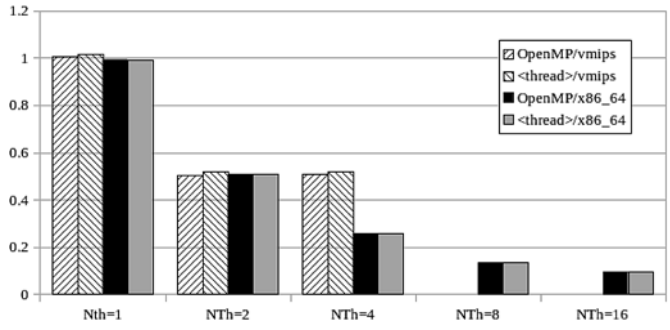


Рис. 3. Время работы параллельных реализаций задачи Calc, нормированное по времени работы последовательной реализации

3.3. Задачи обработки массивов

В этом подразделе рассмотрены результаты измерений для двух задач (Calc, Calc1) вычислений с массивами, отличающихся шаблонами доступа к памяти.

Задача Calc. В этой задаче, последовательная реализация которой представлена на листинге 4, в каждой итерации внешнего цикла вычисляется `i`-й элемент массива `res`. Для его вычисления используется внутренний цикл, в котором суммируются значения функции `asin` с разными аргументами.

Листинг 4. Последовательная реализация задачи Calc

```
double res[N]={0.0};
...
for (int i=0; i<N; i++) {
   res[i]=0.0;
   for (int j=1; j<= N; j++)
      res[i] += asin(i/((double)N*j*j));
}
```

Поскольку итерации внешнего цикла независимы, то для его распараллеливания средствами OpenMP достаточно добавить директиву `#pragma omp parallel for` перед заголовком внешнего цикла. Параллельная реализация с использованием библиотеки потоков Си++ организована так же, как и для задачи Atomic.

В табл. 3 показаны результаты измерений времени выполнения (в секундах) для последовательной (строка **Seq**) и параллельных (строки **OpenMP**, **<thread>**) реализаций задачи Calc для трех целевых платформ.

На рис. 3 показано время выполнения параллельных реализаций задачи Calc для разного числа потоков, нормированное по времени работы для последовательной реализации, для платформ Комдив/Linux на vmips и ×86_64/Linux.

Задача Calc1. Эта задача отличается от предыдущей структурой зависимостей по данным. Последовательная реализация представлена на листинге 5.

Листинг 5. Последовательная реализация задачи Calc1

```
double *src;
double *res;
...
src = new double[N];
for (int i=0; i<N; i++) src[i] =
sin((double)i);
res = new double[N]();
...
for (int i=1; i <= N; i++) {
   double k=1+1/(double)i;
   for (int j=0; j<= N-1; j++)
      res[j] += k*src[j];
}
```

Распараллеливание по внешнему циклу в данном случае неэффективно, поскольку для этого пришлось бы описать `res` как массив атомарных элементов `atomic<double>`. Поэтому здесь применялись другие способы распараллеливания. В табл. 4 показаны результаты измерения времени

Таблица 3

Результаты измерений для задачи Calc, с

Платформа	1890ВМ8Я/Багет, 1 ядро			Комдив/Linux на vmips, 2 ядра			×86_64/Linux, 24 ядра				
Seq	30,109			49,071			41,023				
Par	**1**	**2**	**4**	**1**	**2**	**4**	**1**	**2**	**4**	**8**	**16**
OpenMP	30,085	30,085	30,085	49,438	24,808	25,017	40,747	20,829	10,576	5,4796	3,9316
<thread>	30,859	30,869	30,885	49,968	25,384	25,487	40,723	20,853	10,535	5,4571	4,0010

Таблица 4

Результаты измерений для задачи Calc1, с

Платформа	1890ВМ8Я/Багет, 1 ядро			Комдив/Linux на vmips, 2 ядра			×86_64/Linux, 24 ядра				
Seq	15,298			3,2994			5,1982				
Transf	29,006			3,3013			12,932				
Int_transf	14,369			3,2997			5,0389				
Par	**1**	**2**	**4**	**1**	**2**	**4**	**1**	**2**	**4**	**8**	**16**
OpenMP	15,381	15,393	15,795	3,6778	2,0786	4,6481	5,1732	2,9272	1,5728	0,8794	0,9652
OpenMP tasks	14,369	11,089	11,095	3,3002	1,6670	1,6984	5,0530	2,6428	1,3280	0,6388	0,4746
<thread>	14,388	11,078	10,281	3,3086	1,6690	1,6984	5,2500	2,7293	1,4293	0,6063	0,5090
P_transf	29,004	29,004	29,004	3,3002	1,6684	1,7015	13,031	6,5181	3,3986	1,7761	1,2250

выполнения (в секундах) для следующих реализаций задачи Calc1.

Seq: последовательная реализация, представленная на листинге 5.

Transf: последовательная реализация посредством однократного вызова стандартного алгоритма `transform` для вычисления массива `res`. В качестве аргумента используется следующая функция:

```
auto fun=[](double v){
   double r=0.0;
   for (int i=1; i <= N; i++)
      double k=1+1/(double)i; r+=k*v;
   return r;
};
```

Увеличение времени работы по сравнению с последовательной реализацией объясняется многократными перевычислениями выражения `k=1+1/(double)i`, содержащего дорогостоящую операцию деления.

Int_transf: последовательная реализация посредством вызова стандартного алгоритма `transform` в цикле по `i`, где в качестве аргумента используется функция

```
auto fun2=[k](double s, double r)
   { return r + k * s;};
```

OpenMP: распараллеливание по внутреннему циклу средствами OpenMP, реализуется добавлением директивы `#pragma omp parallel for` перед внутренним циклом.

OpenMP tasks: распараллеливание средствами OpenMP с разбиением массива `res` на фрагменты для улучшения локальности данных. Для вычисления каждого фрагмента запускается задача OpenMP (листинг 6).

<thread>: распараллеливание по внутреннему циклу средствами библиотеки потоков аналогично **OpenMP tasks**.

P_transf: то же, что **Transf**, но с применением параллельной версии библиотеки STL.

*Листинг 6. Реализация **OpenMP tasks** задачи Calc1*

```
#pragma omp parallel
{
   #pragma omp single
   for (int n=1; n <= NTH; n++) {
      #pragma omp task untied firstprivate(n)
      for (int i=1; i <= N; i++) {
         double k=1+1/(double)i;
         for (int j=N*(n-1)/NTH; j<=N*n/NTH-1; j++)
            res[j] += k*src[j];
      }
   }
   #pragma omp barrier
   }
```

На рис. 4 представлено время работы параллельных реализаций задачи Calc1, нормированное по времени работы последовательной реализации, для платформ Комдив/Linux на vmips и ×86_64/Linux.

Из таблицы 4 и рис. 4 видно, что реализации **OpenMP tasks** и **<thread>** эффективнее, чем простое распараллеливание **OpenMP**, благодаря улучшению локальности данных. Заметим, что эти реализации при увеличении числа потоков дают ускорение даже для одноядерной платформы 1890ВМ8Я/Багет.

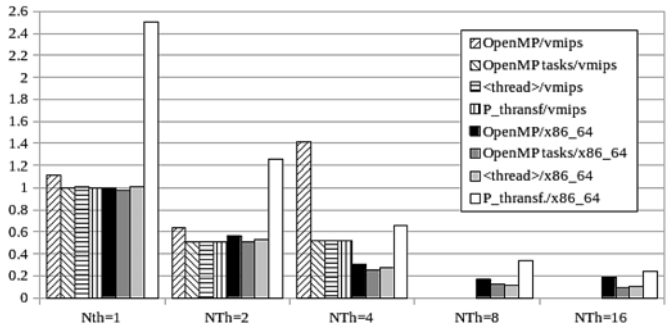


Рис. 4. Время работы параллельных реализаций задачи Calc1, нормированное по времени работы последовательной реализации

3.4. Задача слияния отсортированных массивов

В данном подразделе рассмотрены различные реализации задачи слияния двух монотонно возрастающих массивов целых чисел без совпадающих значений. Простейшая последовательная реализация этой задачи представлена на листинге 7.

Листинг 7. Последовательная реализация задачи слияния двух упорядоченных массивов

```
seqmerge _ (const size _ t na, const val _ t a[],
     const size _ t nb, const val _ t b[],
     val _ t r[]) {
   size _ t ia=0, ib=0, i=0;
   while (ia<na && ib<nb) {
      if (a[ia] < b[ib])
         r[i + +] = a[ia++];
      else if (a[ia] > b[ib])
         r[i + +] = b[ib++];
      else
         error("Duplicate");
   }
   if (ia<na) memcpy(&r[i], &a[ia],
      (na-ia)*sizeof(r[0]));
   if (ib<nb) memcpy(&r[i], &b[ib],
      (nb-ib)*sizeof(r[0]));
}
```

Эта задача интересна тем, что для нее нет очевидного способа распараллеливания. Неочевидный **рекурсивный подход** заключается в том, что выбирается середина большего массива и методом бинарного поиска находится соответствующая позиция в меньшем массиве. Затем рекурсивно выполняются две задачи: слияние первых частей двух массивов и слияние вторых частей. При этом общая асимптотика ухудшается: вместо O(n) в последовательной реализации получаем O($n$log(n)); но за счет параллельного выполнения подзадач слияния возможно ускорение.

Во всех рассмотренных далее параллельных реализациях применяется асинхронный запуск задач, без явного использования потоков. В табл. 5 приведены результаты измерений времени выполнения (в секундах) для перечисленных ниже реализаций.

Seq: последовательная реализация, приведенная на листинге 7.

Bin: последовательное исполнение приведенного выше рекурсивного алгоритма с оптимизацией: если сумма длин массивов меньше порогового значения, то применяется последовательное слияние.

S_merge: вызов стандартного библиотечного алгоритма `merge`.

M_task2: рекурсивный алгоритм с асинхронным запуском двух подзадач для слияния первых и вторых частей входных массивов, реализованный средствами OpenMP.

M_task1: то же, что **M_task2**, но асинхронный запуск выполняется только для одной подзадачи, а вторая выполняется в рамках текущей задачи. Это позволяет сократить накладные расходы на запуск задач.

M_task: оптимизированный вариант **M_task1**, где асинхронный запуск применяется только, если сумма длин массивов больше некоторого порогового значения.

Во всех приведенных выше методах используется последовательное слияние, если размеры входных массивов меньше некоторого порогового значения.

M_leaf: принципиально другой подход: здесь асинхронные задачи не создаются рекурсивно. Вместо этого в рекурсивном алгоритме слияния асинхронные задания запускаются только для "листьев", т. е. для последовательных слияний подмассивов, размер которых меньше порогового значения, а также для копирований памяти, когда один из сливаемых массивов

Таблица 5

Результаты измерений для задачи слияния массивов, с

Платформа	1890ВМ8Я/Багет 1 ядро			Комдив/Linux на vmips, 2 ядра			×86_64/Linux, 24 ядра				
Seq	4,4049			2,4060			19,150				
Bin	4,5601			2,3205			20,301				
S_merge	3,7658			1,6489			19,003				
Par	**1**	**2**	**4**	**1**	**2**	**4**	**1**	**2**	**4**	**8**	**16**
M_task2	4,6641	4,6688	4,6698	2,3576	2,2550	1,2205	19,474	23,301	12,328	12,543	14,181
M_task1	4,6192	4,6257	4,6276	2,3290	2,2237	1,1899	19,435	21,871	13,615	12,533	14,140
M_task	4,5602	4,5610	4,5605	2,3174	2,2023	2,2054	19,407	19,334	11,416	13,568	13,011
M_leaf	4,4074	4,4119	4,4166	2,3537	1,2185	1,2312	19,080	19,035	13,316	12,273	15,009
F_task2	4,5710	4,5849	4,6086	2,3180	2,2181	1,2352	19,569	15,475	14,023	13,840	13,785
F_task1	4,5662	4,5833	4,5980	2,3165	2,2065	1,2158	19,458	17,640	11,350	10,715	11,042
F_task	4,5638	4,5749	4,5994	2,3162	2,2033	1,2156	19,521	17,418	12,169	10,566	10,874
F_leaf	4,5761	4,5893	4,5977	2,3484	1,5406	1,3070	19,218	22,154	16,473	14,167	14,265
P_merge	3,8210	4,7349	4,7459	1,8829	1,7790	1,3653	20,999	15,866	10,995	9,5190	9,3096

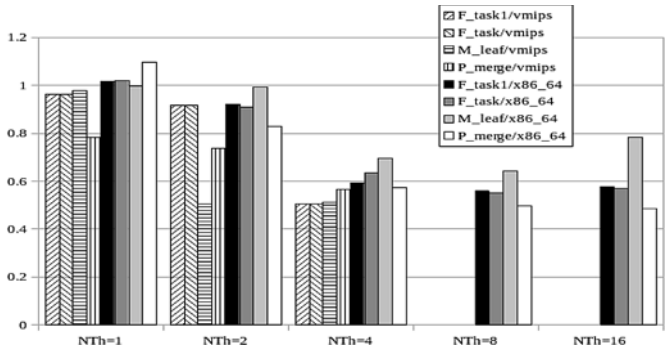


Рис. 5. Время выполнения некоторых параллельных реализаций задачи слияния массивов

пуст. Ожидание окончания запущенных задач происходит на верхнем уровне, после выхода из рекурсии.

Параллельные реализации **F_task2**, **F_task1**, **F_task**, **F_leaf** аналогичны описанным выше четырем алгоритмам, но для создания асинхронных задач применяется вызов `async` библиотеки <future> языка Си++.

P_merge: вызов алгоритма `merge` из параллельной версии библиотеки STL.

В параллельных реализациях **F_task2, F_task1, F_task, F_leaf** средствами библиотеки <future> предусмотрена проверка ограничений на общее число одновременно работающих задач. Если заданный максимум достигнут, то асинхронный запуск задачи не проводится. Вместо этого выполняется синхронное выполнение задачи в рамках текущего потока. Это сделано по той причине, что в ОС РВ Багет существуют ограничения на число одновременно используемых ресурсов (потоков, мьютексов, условных переменных), которые задействованы в реализации асинхронного запуска задач. Такой подход целесообразен и для целевых систем, не имеющих подобных ограничений, так как бесконтрольное создание большого числа задач приводит к росту накладных расходов на их планирование. Заметим, что в реализациях аналогичных алгоритмов средствами OpenMP ограничение на общее число работающих потоков обеспечивается автоматически.

На рис. 5 представлено время выполнения наиболее эффективных параллельных реализаций задачи слияния (**F_task1**, **F_task**, **M_leaf**, **P_merge**), нормированное по времени выполнения последовательной реализации для целевых платформ Комдив/Linux на vmips и ×86_64/Linux.

Результаты измерений показывают, что стандартные алгоритмы не уступают по производительности соответствующим последовательным и параллельным реализациям. Для платформы Комдив/Linux дополнительное ускорение можно получить, используя число потоков, превышающее число ядер.

Заключение

Результаты проведенного тестирования подтверждают работоспособность представленных средств параллельного и конкурентного программирования в компиляторе с языков Си и Си++ для процессоров Комдив под управлением ОС РВ Багет. Были также проведены измерения времени выполнения для набора задач на платформах Комдив/Linux на vmips и ×86_64/Linux, для того чтобы косвенно оценить потенциальное увеличение производительности в результате распараллеливания программ для многоядерных процессоров Комдив под управлением ОС РВ Багет. Результаты этих измерений показывают, что для простых циклов последовательной обработки массивов рост производительности может быть пропорционален числу ядер. Для более сложных алгоритмов, таких как задача слияния массивов, ожидаемое ускорение не столь высоко, но также может быть значительным.

По результатам измерений для задачи Atomic1 интенсивное использование атомарных операций может приводить к значительным потерям производительности. Однако для процессоров Комдив эти потери ниже, чем для процессоров ×86_64/Linux. Соотношение ускорения в результате распараллеливания и потерь от использования атомарных операций должно оцениваться в каждом конкретном случае.

Измерения для задач Atomic, Atomic1, Calc, Calc1 показывают, что ручное распараллеливание простых циклов средствами библиотеки потоков <thread> не имеет преимуществ по эффективности кода перед распараллеливанием средствами OpenMP, которое значительно проще в реализации и предоставляет дополнительные возможности, в частности, опции планирования.

Использование функции асинхронного запуска задач библиотеки <future> при разработке программ для ОС РВ Багет затруднено тем обстоятельством, что при этом необходимо учитывать ограничения системы по числу доступных ресурсов (потоков, мьютексов, условных переменных), которые задаются при конфигурировании системы. При реализации рекурсивных алгоритмов, в которых число возможных параллельных задач сложно заранее оценить, необходимо либо обеспечить принудительное ограничение на число создаваемых асинхронных задач, либо отдать предпочтение использованию средствам запуска задач OpenMP.

Направления дальнейшей работы по развитию средств параллельного программирования для многоядерных процессоров в компиляторе включают, в первую очередь, поддержку параллельной версии библиотеки алгоритмов, вошедшую в стандарт языка С++ версии 17. В настоящее время компилятор содержит лишь экспериментальную реализацию параллельной версии библиотеки, интерфейс которой не вполне соответствует спецификациям С++17.

Результаты тестов Atomic и Calc1 показывают значение комплексной оптимизации программ, которая должна дополнять их распараллеливание. В частности, важным резервом повышения эффективности как параллельных, так и последовательных программ является поддержка оптимизации по профилю. Результаты теста Atomic показывают, что использование данной оптимизации может дать увеличение производительности до 1,5 раз. В настоящее время ведется разработка средств поддержки профилирования в ОС РВ Багет для целей оптимизации кода. В задаче Calc1 дополнительное увеличение производительности было достигнуто за счет блочной обработки мас-

сива. Поэтому целесообразно включить в будущие версии компилятора поддержку автоматического выделения блоков в циклах обработки массивов.

Планируется также реализовать в компиляторе поддержку векторизации кода с задействованием команд сопроцессора CPV, что позволит, в частности, применять векторизацию циклов совместно с их распараллеливанием с помощью SIMD-директив OpenMP.

Наконец, важным направлением дальнейших исследований и разработок является создание средств для эффективного комплексного применения параллелизма гетерогенных многопроцессорных систем, включающих многоядерные процессоры со специализированными сопроцессорами.

Публикация выполнена в рамках государственного задания по проведению фундаментальных научных исследований по теме (проекту) "38. Проблемы создания глобальных и интегрированных информационно-телекоммуникационных систем и сетей, развитие технологий и стандартов GRID. Разработка средств распараллеливания программ для отечественных многоядерных микропроцессоров (0065-2018-0010)".

Список литературы

1. **Бетелин В. Б.** О проблеме импортозамещения и альтернативной модели экономического развития России // Стратегические приоритеты. 2016. № 1 (9). С. 11—21.

2. **Бобков С. Г., Аряшев С. И., Зубковский П. С., Морев С. А., Рогаткин Б. Ю.** Высокопроизводительный микропроцессор 1890ВМ118 с архитектурой Комдив для создания доверенных систем // Программные продукты и системы. 2017. Т. 30, № 3. С. 345—352.

3. **Бобков С. Г., Аряшев С. И., Зубковский П. С.** Арифметические сопроцессоры микропроцессоров с архитектурой КОМДИВ // 6-й Московский суперкомпьютерный форум. URL: https://www.osp.ru/netcat_files/userfiles/MSKF_2015/Aryashev_cor.pdf (дата обращения 13.08.2018).

4. **Аристов М. С., Осипов А. С., Щегольков А. М.** О некоторых практических вопросах программирования сопроцессора обработки сигналов микропроцессора КОМДИВ128-РИО // Труды научно-исследовательского института системных исследований Российской академии наук. 2015. Т. 5, № 2. С. 130—137.

5. **Аристов М. С.** Методы оптимизации программ для процессора КОМДИВ128-РИО // Труды научно-исследовательского института системных исследований Российской академии наук. 2014. Т. 4, № 2. С. 33—39.

6. **Аристов М. С., Базаева С. Е., Самборский С. В., Сударева О. Ю., Щегольков А. М.** Руководство по разработке и оптимизации программ для специализированного сопроцессора СР2 в составе микропроцессора КОМДИВ128-РИО. М.: НИИСИ РАН, 2015. 121 с.

7. **Вьюкова Н. И., Самборский С. В., Галатенко В. А.** Программная конвейеризация циклов для ускорителя плавающей арифметики в составе процессора КОМДИВ128-РИО // Программные продукты и системы. 2013. № 4. С. 35—43.

8. **Программа** "Библиотека цифровой обработки сигналов (БЦОС). Справочник по функциям с интерфейсом cp2m программы БЦОС". М.: НИИСИ РАН, 2014.

9. **Бобков С. Г., Аряшев С. И., Барских М. Е., Зубковский П. С., Ивасюк Е. В.** Высокопроизводительные расширения архитектуры универсальных микропроцессоров для ускорения инженерных расчетов // Информационные технологии. 2014. № 6. С. 27—37.

10. **Богданов П. Б., Сударева О. Ю.** Применение отечественных специализированных процессоров семейства Комдив в научных расчетах // Информационные технологии и вычислительные системы. 2016. № 3. С. 45—65.

11. **Вьюкова Н. И., Галатенко В. А., Самборский С. В.** Использование векторных расширений современных процессоров // Программные продукты и системы. 2016. Т. 29, № 4. С. 147—157.

12. **Бурцев А. А.** Применение векторного сопроцессора для оптимизации функций библиотеки линейной алгебры. URL: http://2014.nscf.ru/TesisAll/2_Prikladnoe_PO/02_209_BurtsevAA.pdf (дата обращения 13.08.2018).

13. **Бурцев А. А.** Оптимизация алгоритмов быстрого преобразования Фурье для специализированного векторного сопроцессора с учетом иерархической структуры памяти // Труды научно-исследовательского института системных исследований Российской академии наук. 2017. Т. 7, № 7. С. 83—95.

14. **Сударева О. Ю.** Распределенные вычисления на процессорах комдив на примере алгоритма NPB MG // Сборник научных статей по итогам Международной научно-практической конференции "Наука нового времени: сохраняя прошлое — создаем будущее". СПб.: КультИнформПресс, 2017. С. 60—63.

15. **Райко Г. О.** Библиотека параллельной обработки сигналов // Труды научно-исследовательского института системных исследований Российской академии наук. 2015. Т. 5, № 1. С. 64—69.

16. **Грингауз Т. А., Онин А. Н.** Особенности использования конфигурационных файлов при интеграции технологий параллельной обработки сигналов, приема данных по высокоскоростному каналу, подготовки запуска задач в мультипроцессорных комплексах реального времени // Труды научно-исследовательского института системных исследований Российской академии наук. 2017. Т. 7, № 7. С. 58—69.

17. **The OpenMP** API specification for parallel programming. URL: http://www.openmp.org/specifications (дата обращения 13.08.2018).

18. **V'yukova N. I., Galatenko V. A., Samboskii S. V.** Support for Parallel and Concurrent Programming in C++ //Programming and Computer Software. 2018. Vol. 44, No. 1. P. 35—42.

19. **Parsec** Benchmark Suite. URL: http://parsec.cs.princeton.edu (дата обращения 13.08.2018).

Support for Multicore Programming in the Compiler for the Komdiv Processors under the "Baget" RTOS

N. I. V'yukova, qniva@yandex.ru, **V. A. Galatenko,** galat@niisi.msk.ru,
S. V. Samborskij, sambor@niisi.msk.ru, Scientific Research Institute for System Analysis of the Russian Academy of Sciences, Moscow, 117218, Russian Federation

Corresponding author:

V'yukova Nadezhda I., Scientific Research Institute for System Analysis of the Russian Academy of Sciences, Moscow, 117218, Russian Federation,
E-mail: qniva@yandex.ru

Received on August 20, 2018
Accepted on August 28, 2018

The processors of the MIPS-compatible architecture Komdiv were being developed in the Scientific Research Institute of System Development (NIISI) of the Russian Academy of Sciences during the last 20 years. Initially they were intended mainly for embedded systems including ones used in the aerospace industry. Modern models of the Komdiv processors are used also in design of common classes of computer systems — laptops, tablets, servers, network routers.

Special attention in the architectural design of the Komdiv processors is given to such characteristics as high computational efficiency, trustability, ability to function in extreme conditions and optimal trade-off between performance and energy consumption. The last property is achieved through the use of various kinds of parallelism provided with specialized coprocessors and use of multiprocessor configurations. The paper provides a brief review of programming tools which simplify development of efficient application software using specialized coprocessors and multiprocessor configurations.

One of the recent Komdiv processors 1890VM118 is a dual core processor with uniform memory access. The main part of the paper is devoted to the support of parallel and concurrent programming in the currently being developed compiler for Komdiv processor under control of the real time operating system (RTOS) Baget. We present the results of time measurements for a number of tests parallelized with use of OpenMP and C++ libraries <thread> and <future> and discuss restrictions in use of asyncronous tasks on RTOS Baget patforms. It is shown that there are certain performance reserves which may be put to use with additional code optimizations. In the conclusion the directions for future research and development are discussed.

Keywords: *Komdiv processor, parallel and concurrent programming, C++, OpenMP.*

Acknowledgements: *The paper is published within the state assignment for conducting fundamental research on the topic (project) "38. Problems of creating global and integrated information and telecommunication systems and networks, developing the GRID technologies and standards. Developing tools for parallelizing programs for national multicore microprocessors (0065-2018-0010)."*

For citation:

V'yukova N. I., Galatenko V. A., Samborskij S. V. Support for Multicore Programming in the Compiler for the Komdiv Processors under the "Baget" RTOS*, Programmnaya Ingeneria,* 2018, vol. 9, no. 9, pp. 393—403.

DOI: 10.17587/prin.9.393-403

References

1. **Betelin V. B.** O probleme importozameshhenija i al'ternativnoj modeli jekonomicheskogo razvitija Rossii (On Import Phaseout and Alternative Model of Economic Development of Russia), *Strategicheskie prioritety*, 2016, no. 1 (9), pp. 11—21 (in Russian).

2. **Bobkov S. G., Aryashev S. I., Zubkovsky P. S., Morev S. A., Rogatkin B. Yu.** Vysokoproizvoditel'nyj mikroprocessor 1890vm118 s arhitekturoj Komdiv dlja sozdanija doverennyh sistem (High Performance Microprocessor 1890VM118 for Trusted Computing Systems), *Programmnye produkty i sistemy,* 2017, vol. 30, no. 3, pp. 345—352 (in Russian).

3. **Bobkov S. G., Aryashev S. I., Zubkovsky P. S.** Arifmeticheskie soprocessory mikroprocessorov s arhitekturoj KOMDIV (Arithmetic Coprocessors of KOMDIV Microprocessors), *6-j Moskovskij superkomp'juternyj forum*, available at: https://www.osp.ru/netcat_files/userfiles/MSKF_2015/Aryashev_cor.pdf (in Russian).

4. **Aristov M. S., Osipov A. S., Shhegol'kov A. M.** O nekotoryh prakticheskih voprosah programmirovanija soprocessora obrabotki signalov mikroprocessora KOMDIV128-RIO (On Practical Issues of Programming for a Signal Processing Coprocessor of the KOMDIV128-RIO Microprocessor), *Trudy nauchno-issledovatel'skogo instituta sistemnyh issledovanij Rossijskoj akademii nauk,* 2015, vol. 5, no. 2, pp. 130—137 (in Russian).

5. **Aristov M. S.** Metody optimizacii programm dlja processora KOMDIV128-RIO (Program Optimization Methods for the KOMDIV128-RIO Processor), *Trudy nauchno-issledovatel'skogo instituta sistemnyh issledovanij Rossijskoj akademii nauk,* 2014, vol. 4, no. 2, pp. 33—39 (in Russian).

6. **Aristov M. S., Bazaeva S. E., Samborskij S. V.**, **Sudareva O. Yu., Shhegol'kov A. M.** Rukovodstvo po razrabotke i optimizacii programm dlja specializirovannogo soprocessora SR2 v sostave mikroprocessora KOMDIV128-RIO (Guide for Development and Optimization of Programs for a Specialized Coprocessor CP2 of the KOMDIV128-RIO Processor), NIISI RAN, 2015, 121 p. (in Russian).

7. **V'yukova N. I., Samborskij S. V., Galatenko V. A.** Programmnaja konvejerizacija ciklov dlja uskoritelja plavajushhej arifmetiki v sostave processora KOMDIV128-RIO (Software Pipelining of Loops for Floating Point Calculations Accelerator of the KOMDIV128-RIO Processor), *Programmnye produkty i sistemy*, 2013, no. 4, pp. 35—43 (in Russian).

8. **Programma** "Biblioteka cifrovoj obrabotki signalov (BCOS). Spravochnik po funkcijam s interfejsom cp2m programmy BCOS (Library for Digital Signal Processing (BCOS). Reference Manual on cp2m Functions of BCOS)", NIISI RAN, 2014.

9. **Bobkov S. G., Aryashev S. I., Barskih M. E., Zubkovsky P. S., Ivasjuk E. V.** Vysokoproizvoditel'nye rasshirenija arhitektury universal'nyh mikroprocessorov dlja uskorenija inzhenernyh raschetov (High Performance Extensions of Universal Microprocessors for Speeding up Engineering Calculations), *Informacionnye tehnologii*, 2014, no. 6, pp. 27—37.

10. **Bogdanov P. B., Sudareva O. Yu.** Primenenie otechestvennyh specializirovannyh processorov semejstva Komdiv v nauchnyh raschjotah (Application of Domestic Specialized Komdiv Processors for Scientific Calculations), *Informacionnye tehnologii i vychislitel'nye sistemy*, 2016, no. 3, pp. 45—65 (in Russian).

11. **V'yukova N. I., Samborskij S. V., Galatenko V. A.** Ispol'zovanie vektornyh rasshirenij sovremennyh processorov (Application of Vector Extensions of Modern Processors), *Programmnye produkty i sistemy*, 2016, vol. 29, no. 4, pp. 147—157 (in Russian).

12. **Burcev A. A.** Primenenie vektornogo soprocessora dlja optimizacii funkcij biblioteki linejnoj algebry (Application of a Vector Coprocessor for Optimization of Functions of the Linear Algebra Library), available at: http://2014.nscf.ru/TesisAll/2_Prikladnoe_PO/02_209_BurtsevAA.pdf (in Russian).

13. **Burcev A. A.** Optimizacija algoritmov bystrogo preobrazovanija Fur'e dlja specializirovannogo vektornogo soprocessora s uchjotom ierarhicheskoj struktury pamjati (Optimization of the Fast Furje Transformation Algorithms for Specialized Vector Coprocessor with Respect to Hierarchical Memory Structure), *Trudy nauchno-issledovatel'skogo instituta sistemnyh issledovanij Rossijskoj akademii nauk*, 2017, vol. 7, no. 7, pp. 83—95 (in Russian).

14. **Sudareva O. Yu.** Raspredeljonnye vychislenija na processorah Komdiv na primere algoritma NPB MG (Distributed Calculations of Komdiv Processors through the Example of NPB MG Algorithm), *Proceedings of the International Conference "Nauka novogo vremeni: sohranjaja proshloe — sozdaem budushhee"*, 2017, pp. 60—63 (in Russian).

15. **Rajko G. O.** Biblioteka parallel'noj obrabotki signalov (Library of Parallel Signal Processing), *Trudy nauchno-issledovatel'skogo instituta sistemnyh issledovanij Rossijskoj akademii nauk*, 2015, vol. 5, no. 1, pp. 64—69 (in Russian).

16. **Gringauz T. A., Onin A. N.** Osobennosti ispol'zovanija konfiguracionnyh fajlov pri integracii tehnologij parallel'noj obrabotki signalov, prijoma dannyh po vysokoskorostnomu kanalu, podgotovki zapuska zadach v mul'tiprocessornyh kompleksah real'nogo vremeni (Specifics of Usage of Configure Files for Integration of Technologies of Parallel Signal Processing, Reception of Data through a High Speed Channel and Preparation of Task Runs in Real Time Multiprocessor Complexes), *Trudy nauchno-issledovatel'skogo instituta sistemnyh issledovanij Rossijskoj akademii nauk*, 2017, vol. 7, no. 7, pp. 58—69 (in Russian).

17. **The OpenMP** API specification for parallel programming, available at: http://www.openmp.org/specifications

18. **V'yukova N. I., Galatenko V. A., Samboskii S. V.** Support for Parallel and Concurrent Programming in C++, *Programming and Computer Software*, 2018, vol. 44, no. 1, pp. 35—42.

19. **Parsec** Benchmark Suite, available at: http://parsec.cs.princeton.edu

УДК 004.052.42, 004.272.43, 004.056.55 DOI: 10.17587/prin.9.404-414

С. Г. Елизаров, канд. физ.-мат. наук, ст. науч. сотр., e-mail: elizarov@physics.msu.ru,
Г. А. Лукьянченко, канд. физ.-мат. наук, науч. сотр., e-mail: lukyanchenko@physics.msu.ru,
Д. С. Марков, ст. науч. сотр., e-mail: markovds@maltsystem.com,
А. М. Монахов, науч. сотр., e-mail: monahov.aleksandr@physics.msu.ru,
В. А. Роганов, ст. науч. сотр., e-mail: radug-a@ya.ru, Московский государственный университет имени М. В. Ломоносова

Тестирование многоядерных вычислительных систем на основе идей алгоритма RSA

С каждым годом возрастает сложность вычислительных систем, что увеличивает вероятность появления ошибок как на этапе их разработки, так и в процессе эксплуатации. В статье рассмотрен качественный тест на основе идей алгоритма RSA для поиска проблемных мест в программном обеспечении и на аппаратном уровне many-core-систем. Разработанный тест способен реагировать на одиночные ошибки в арифметических операциях, а результат вычислений сложно предугадать и подделать без предварительных значительных вычислений.

Ключевые слова: *many-core-системы, динамическое распараллеливание вычислений, локализация ошибок в оборудовании, алгоритм RSA*

Введение

Начало 2018 г. ознаменовалось скандалом в мире информационных технологий — большинство современных процессоров Intel, выпущенных за последние 15 лет, оказались подвержены уязвимости, которая позволяет получить доступ к конфиденциальным данным компьютера [1, 2]. Несколько лет назад аналогичные уязвимости были обнаружены в линейке процессоров AMD, устранение таких уязвимостей обернулось почти полугодовой задержкой выпуска новых четырехъядерных процессоров.

С ошибками в процессорах AMD Ryzen были связаны ситуации с неожиданными сообщениями об ошибках сегментации, с которой эпизодически сталкивались пользователи различных Unix-подобных систем (например, в Linux или FreeBSD) [3]. Суть возникающей проблемы заключалась в том, что при продолжительных многопоточных тяжелых нагрузках, например, при компиляции масштабных проектов, могла появиться ошибка сегментации, спровоцированная неправильной работой процессора. В начале августа 2017 г. эта проблема была признана AMD и определена как "сложная маргинальная проблема, возникающая исключительно при определенных нагрузках в Linux". При этом было подтверждено, что она затрагивает только процессоры Ryzen, в то время как вышедшие в более поздние сроки процессоры EPYC и Ryzen Threadripper ей не подвержены.

Еще в 1994 г. в процессорах Pentium была обнаружена ошибка в модуле операций с плавающей запятой, получившая кодовое название FDIV [4]. Ошибка выражалась в некорректности при проведении деления над числами с плавающей запятой с помощью команды процессора FDIV.

Различные ошибки и уязвимости при проектировании аппаратуры выявляются достаточно часто. Сложившееся положение характеризует диаграмма, представленная на рис. 1. Эта диаграмма указывает число обнаруженных ошибок на стадии докремниевой разработки различных поколений процессора Pentium [5].

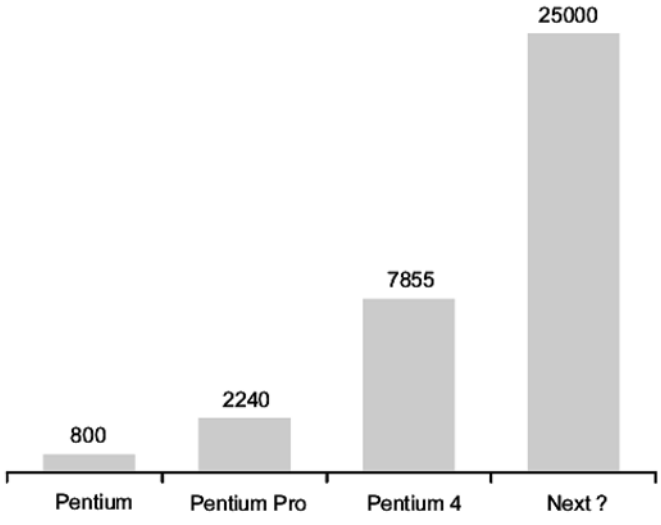


Рис. 1. Число ошибок на стадии докремниевой разработки разных поколений процессора Pentium

В силу увеличения сложности вычислительных систем возрастает объем ошибок и уязвимостей. В частности, доказательство корректности для many-core-систем на уровне математической модели устройства является сложной задачей для отработанных на настоящее время методов формальной верификации.

Помимо формальной математической модели цифрового устройства на практике нельзя также абстрагироваться и от его физической модели. На физическом уровне существует целый спектр источников потенциальных проблем различной физической природы, включая электромагнитное взаимодействие компонентов в электронной схеме, дрейф параметров кристалла при нагреве, падение напряжения на активном сопротивлении шин питания при пиковой нагрузке, диффузию материалов при старении и т. д. Исчерпывающее моделирование всех таких физических явлений для кристалла со сложной топологией представляет собой трудноразрешимую задачу.

В связи с перечисленными выше нюансами и проблемами представляют интерес подходы, позволяющие выявлять ошибки и уязвимости как на программном, так и аппаратном уровнях путем запуска программных тестов, обеспечивающих длительную загрузку системы с легко проверяемым результатом. Одним из таких подходов является построение программных тестов на основе идей криптографических алгоритмов.

1. Тестирование специализированных many-core-систем

Подходов к созданию программных тестов для ЭВМ существует много [6, 7], и каждый из этих подходов имеет свои сильные и слабые стороны. Соответственно, существует и множество различных категорий тестов, применяемых в разных случаях, на разных уровнях вычислительных систем и для разных целей. Тем не менее наиболее часто упоминаемые на настоящее время тесты не лишены серьезных недостатков, что объясняется, прежде всего, их ориентацией на универсальные вычислительные платформы предыдущих поколений. В настоящей работе рассматривается тест, базирующийся на целочисленной арифметике, лежащей в основе алгоритма несимметричного шифрования RSA [8]. Такой подход к тестированию позволяет при относительной простоте кода теста получить ряд весьма востребованных свойств на разных стадиях проверки many-core-систем.

В настоящее время создание специализированных вычислительных устройств для решения принципиально разных задач становится актуальной задачей. Например, некоторые виды обработки данных нуждаются в быстрых целочисленных вычислениях. Это приводит к тому, что универсальные и графические процессоры постепенно вытесняются в этих случаях специализированными микросхемами.

Имеющиеся, а также разрабатываемые в настоящее время many-core-системы порой не имеют большого объема оперативной памяти, в связи с чем многие тесты, разработанные для традиционных many-core-систем, оказываются к ним неприменимы. Поэтому представляют интерес тесты, способные эффективно выполняться на процессорных модулях с небольшим объемом доступной памяти без использования операции с плавающей точкой, однако обеспечивающие при этом хорошо рандомизированное покрытие в пространстве состояний вычислительной системы.

Интересным решением здесь представляется использование идей из области криптографии. В частности, при криптографических преобразованиях обеспечивается хорошее перемешивание бит. Точный, целочисленный характер криптографических преобразований позволяет при любой продолжительности счета исключить эффекты типа накопления ошибок.

1.1. Недостатки традиционных подходов к тестированию оборудования

Одним из традиционных подходов к тестированию вычислительных систем является запуск тестов на ЭВМ A с последующей проверкой результатов на заведомо исправной ЭВМ B. Однако такой подход предполагает, что имеется образцовая ЭВМ с производительностью, сравнимой с производительностью тестируемой вычислительной системы. Обозначенную проблему, на первый взгляд, можно решать следующими способами:

- организовать вычисление сложных задач с заведомо известным результатом;
- многократно вычислять одно и то же сложное выражение на всех имеющихся вычислительных ядрах системы;
- вычислять сложные, различные по форме, но эквивалентные по результату математические выражения, проводя последующую "очную ставку" для полученных ответов.

Следует однако отметить, что перечисленные способы имеют и соответствующие недостатки:

- заранее известный результат возможно получить случайным образом в силу, например, ошибок в коммуникационной среде или в случае недобросовестного тестирования;
- вычисление одного и того же выражения снижает тестовое покрытие по множеству состояний системы, в результате чего теряется преимущество many-core-систем в части возможностей самотестирования;
- ЭВМ может одинаково неверно считать на разных ядрах, что приводит к ошибочному заключению о работоспособности системы.

Среди популярных программных средств тестирования наиболее подходящими для тестирования many-core-систем являются LinPack и NPCC. Однако эти средства требовательны к объему доступной вычислительным ядрам оперативной памяти. Также в них используется плавающая арифметика,

и предполагается наличие быстрой сети между ядрами. В силу этих обстоятельств ограничена возможность их применения для некоторых видов many-core-систем.

Многие вычислительные задачи устроены так, что они совершенно не чувствительны к одиночным сбоям. Например, итерационные методы решения физических задач в случае одиночной ошибки склонны автоматически ее исправлять. Такое поведение алгоритма является ценным при эксплуатации системы, однако при тестировании требуется совсем другое поведение теста.

1.2. Корректность параллельного счета при недетерминированном распределении тестовых подзадач по ядрам

Стоит отметить, что проверка корректности процедур распараллеливания вычислений крайне важна. Динамическое распараллеливание программ приводит к тому, что распределение вычислений по ядрам системы меняется от запуска к запуску. Существенным фактором, влияющим на распределение подпрограмм по вычислительным ядрам, является различная тяжесть вычислений, а также случайные временные задержки в коммуникационной среде. Однако при корректной организации вычислений, зависимости итогового результата работы программы от распределения по вычислительным ядрам наблюдаться не должно.

Процесс тестирования many-core-системы (рис. 2), с учетом изложенного выше, должен иметь следующие свойства:

1) в случае запуска тестовой задачи утилизация вычислительной мощности должна составлять значительный процент от ее пиковой производительности;

2) многократный запуск тестовой задачи на разных множествах ядер должен выводить один и тот же верный результат, при этом время счета должно быть в первом приближении обратно пропорционально числу задействованных ядер;

3) частичные результаты вычислений при недетерминированном порядке запуска и свертки результатов вычислений должны отличаться от запуска к запуску, что свидетельствует о недетерминированном распределении подзадач по вычислительным ядрам;

4) итоговый ответ должен быть одинаков во всех случаях, что свидетельствует в пользу корректности алгоритмов распараллеливания и синхронизации вычислений.

1.3. Требования, предъявляемые к средствам тестирования many-core-систем

Тесты, ориентированные на проверку many-core-систем, в том числе специализированных, должны легко адаптироваться к нестандартным платформам. В идеале тесты должны также удовлетворять следующим требованиям:

1) быть простыми в реализации и осуществлять сложные преобразования при вычислении;

2) задействовать все основные подсистемы many-core-систем, включая счетные ядра, память, коммуникационную среду;

3) обеспечивать быструю проверку полученных в результате работы тестов;

4) хорошо масштабироваться и распараллеливаться;

5) иметь предсказуемое и легко варьируемое время выполнения;

6) иметь высокую чувствительность к одиночным сбоям: ошибки на любом этапе выполнения

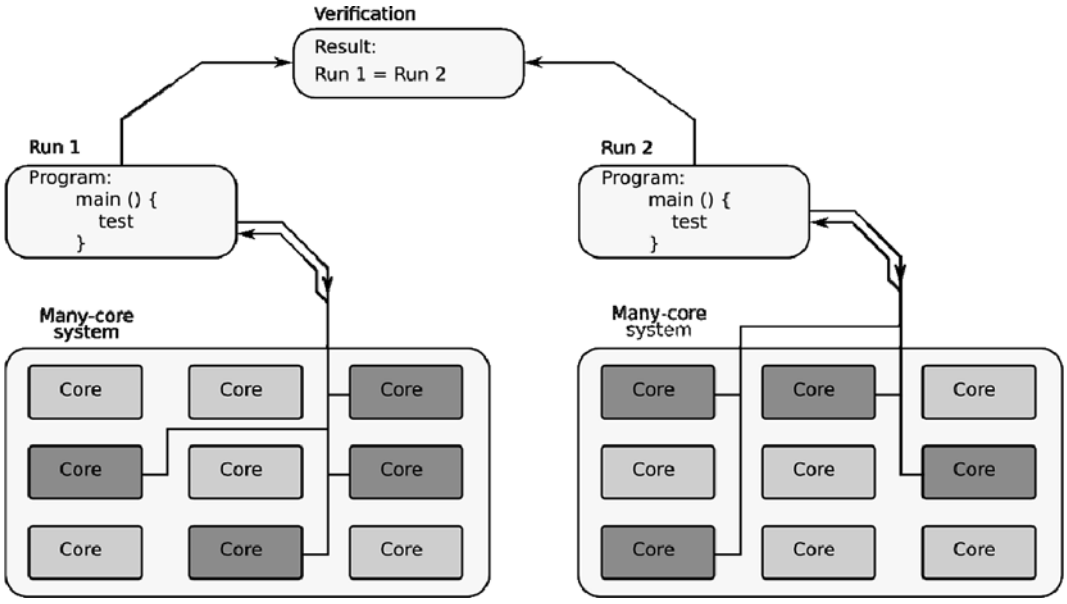


Рис. 2. Верификация выполнения теста на разных ядрах many-core-системы

программы должны с высокой вероятностью отражаться на результате теста;

7) обеспечивать помощь в локализации ошибок;

8) обеспечивать хорошее покрытие и псевдослучайное распределение получаемых в процессе счета промежуточных значений;

9) иметь труднопредсказуемый заранее результат вычислений.

Последнее требование диктуется тем обстоятельством, что запуск тестов нередко встраивают в цикл разработки с верным ответом. В принципе, существенно затруднить подделку результата можно, используя для этого обфускацию кода теста. Такой подход позволяет решить большую часть обсуждаемых вопросов, однако он предполагает безусловное доверие к программе-генератору теста и субъекту, который подобный тест предоставляет. Поэтому наибольший интерес представляют тесты, имеющие следующие свойства: их исходный код является полностью открытым; правильность результатов работы тестов мог бы легко проверить любой желающий; получить результаты тестирования можно лишь проделав значительный объем вычислений.

Поскольку близкие по постановке к перечисленным выше задачи давно являются предметом рассмотрения криптографии, логично поискать аналогии с уже существующими подходами из этой области.

2. Тестирование на основе криптоалгоритма с открытым ключом

Естественно предположить, что поиск подходов к разработке хороших с позиции перечисленных выше требований тестов целесообразно связывать с криптографическими алгоритмами с открытым ключом, такими, например, как RSA.

Действительно, если взять любой шифр с открытым ключом, то дешифрование и расшифрование (последнее отличается именно наличием информации о закрытом ключе) отличаются по вычислительной сложности на много порядков. Далее рассмотрим наиболее известный, достаточно хорошо изученный и активно используемый на практике алгоритм RSA.

2.1. Идеи, лежащие в основе алгоритма RSA

Алгоритм RSA является одним из первых предложенных алгоритмов несимметричного шифрования, в основе которого лежит операция возведения в степень по некоторому большому модулю. Выбор закрытого и открытого ключа заключается в выборе пары псевдослучайных простых чисел p и q, которые и представляют собой закрытый ключ, а в качестве открытого ключа выступает их произведение $N = p \times q$. Определяющим фактором для криптостойкости шифра RSA является сложность задачи разложения открытого ключа N на простые множители. Именно это свойство и позволяет использовать число N в качестве открытой информации.

Как будет показано далее, выбор в качестве теста непосредственно задачи разложения на множители плохо подходит для ЭВМ с малым объемом памяти. Поэтому для описываемого теста используется задача с открытой информацией (т. е. без чисел p и q) с возможностью получить без большого объема вычислений значение первоначального выражения.

2.2. Математические выражения, лежащие в основе теста на базе алгоритма быстрого возведения в степень

Криптостойкость RSA основана на задаче целочисленной факторизации, которая представляет собой задачу разложения целых чисел на большие простые множители. Поэтому представляется разумным взять задачу целочисленной факторизации за основу, а в качестве теста использовать ее решение. Однако столь прямолинейное решение имеет серьезные недостатки. Наиболее эффективный в настоящее время общий метод факторизации, который принято использовать, — это общий метод решета числового поля [9]. Недостатком данного метода является большой объем оперативной памяти, необходимой для его эффективной реализации. Другие субэкспоненциальные по длине числа подходы к факторизации, например, алгоритм Ленстры [10], устроены таким образом, что одиночная ошибка с большой вероятностью никак не повлияет на результат и окажется незамеченной. Тем не менее, используя идеи $P-1$-алгоритма Полларда [11], несложно реализовать простой и эффективный тест, вычисления в котором можно значительно ускорить, зная представление числа N в виде произведения двух простых чисел. С использованием такого простого теста появляется возможность проверки результата, полученного на тестируемой many-core-системе, с помощью обычного персонального компьютера.

В качестве базового выражения теста используется возведение числа A в большую степень K по модулю заданного большого числа N с секретным разложением на простые множители $N = p \times q$. Числа p и q являются большими случайными простыми числами, содержащими многие сотни двоичных разрядов.

Результат данного выражения получается путем использования алгоритма быстрого возведения в степень [12]. Получение результата вычисления можно ускорить путем замены большой степени K на остаток от деления на порядок мультипликативной группы кольца вычетов по модулю N, равный $(p-1) \times (q-1)$.

Полный код теста реализует параллельные вычисления множества подобных выражений и считает их свертку с помощью любой коммутативной ассоциативной операции, отличной от умножения. Умножение в рассматриваемом случае не подходит в связи с тем, что произведение одинаковых степеней, вычисляемое на векторном процессоре, равно степени произведения.

Таким образом, если известно разложение числа N на множители, то вычисление соответствующих больших степеней можно значительно ускорить.

Более интересным в контексте защищенного тестирования является обратное утверждение: если с помощью некоторого алгоритма удастся быстро вычислять произвольно большие степени числа A по модулю N, то с помощью этого же алгоритма можно легко решать считающуюся сложной задачу целочисленной факторизации [13]. Именно это свойство и позволяет получить косвенное обоснование того, что вычислительная сложность предлагаемого теста является достаточно большой, если информация о закрытом ключе недоступна.

2.3. Обоснование сложности вычисляемых выражений

Имея открытый ключ N, во многих случаях нетрудно подобрать такое большое целое число R, что R гарантированно делится на $p - 1$, но не делится на $q - 1$. В $P - 1$-алгоритме Полларда в качестве R берут факториал большого числа r, которое должно быть не меньше, чем наибольший простой делитель числа $p - 1$. Возведя произвольное число A в степень R, получим число M, сравнимое, в силу малой теоремы Ферма, с единицей по модулю p. Вычислив после этого наибольший общий делитель N и $M - 1$, с большой вероятностью получим делитель числа N.

Описанный принцип можно продемонстрировать на следующем наглядном примере для небольшого числа $N = 4\ 007\ 779$ = ("секретное" разложение) 1987×2017. При этом $p - 1 = 1986 = 2 \times 3 \times 331$, $q - 1 = 2016 = 32 \times 9 \times 7$.

В качестве числа R достаточно взять $9! = 362\ 880$, это число, как легко проверить, делится на 2016. Далее возведем, например, число 2 в степень R по модулю N и найдем соответствующий НОД с помощью небольшой программы на языке Python (лисинг 1).

Листинг 1. Поиск "секретного" делителя числа N

```
#!/usr/bin/python3
from math import gcd,factorial

def pow _ mod _ fast(a,b,m):
  r = 1; a = a % m
  while b > 0:
    if b & 1:
      r = (r*a) % m
    b // = 2
    a = (a*a) % m
   return r

N = 4007779

f9 = factorial(9)
print(f9)

x = pow _ mod _ fast(2,f9,N)
print(x)

print(gcd(x-1,N))
```

В результате выполнения программы получим значения, последним из которых является "секретный" делитель числа N (листинг *2*).

Листинг 2. Результаты работы программы по поиску "секретного" делителя числа N

```
362880
1532921
2017
```

Таким образом, альтернативные способы вычислений описанного теста плохо соотносятся с гипотезой о сложности задачи целочисленной факторизации.

2.4. Методика тестирования many-core-систем

В первую очередь, до начала тестирования many-core-системы должна быть реализована параллельная программа для получения итогового вычисляемого тестом выражения. Эта реализация должна допускать легкую замену алгоритмов генерации оснований и показателей степени, а также значения модуля, по которому считаются базовые подвыражения.

Имеющаяся параллельная реализация позволяет получить оценку для числа примитивных операций, необходимых для вычисления единичного базового выражения с известным значением показателя степени. Эта оценка должна хорошо согласовываться с наблюдаемой на практике производительностью.

Методика тестирования many-core-систем наглядно представлена на диаграмме (рис. 3) и происходит в несколько этапов, описанных далее.

Генерация открытого и закрытого ключей осуществляется на образцовой ЭВМ, в качестве которой может выступать персональный компьютер. Проводится выбор формулы для генерирования оснований и показателей с последующим вычислением правильного ответа для теста, полученного с помощью редукции показателей с использованием значения закрытого ключа. Общее число вычисляемых базовых подвыражений должно быть в несколько раз больше имеющегося на тестируемой many-core-системе числа счетных ядер, а эвристическая оценка необходимого времени для вычислений должна соответствовать промежутку времени, которое отводится на нагрузочное тестирование.

В реализацию теста для many-core-системы вносятся выбранный алгоритм генерации оснований и показателей, а также открытый ключ, после чего программа компилируется и запускается.

Полученный результат сравнивается с результатом, полученным на образцовой ЭВМ. Если результаты совпадают, то это косвенно свидетельствует об исправности функционирования аппаратуры и системного программного обеспечения. Если же результат теста оказался другим, то на образцовой ЭВМ необходимо посчитать промежуточные значения всех базовых выражений и сверить их с аналогичными значениями, рассчитанными на вычислительных блоках проверя-

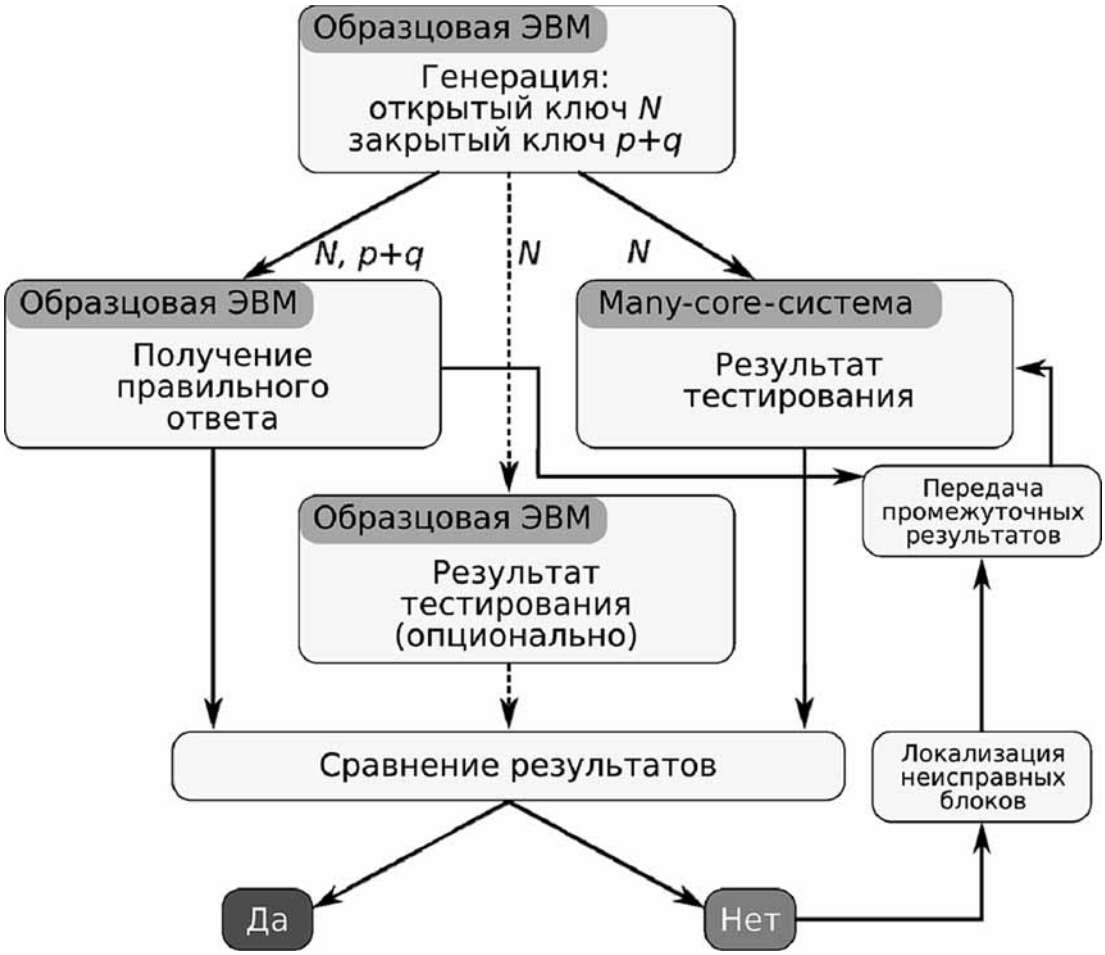


Рис. 3. Схема тестирования many-core-систем

емой many-core-системы. По результатам сравнения и определяются дефекты в вычислительных блоках.

Разработанный тест позволяет локализовать неисправные вычислительные блоки, так как при вычислениях в быстром и медленном режимах значения всех базовых выражений должны совпадать. По этой причине, если тест на many-core-системе "провалился", то можно повторить его, сохраняя в файл все промежуточные значения и номера блоков, на которых они вычислялись. Сравнивая промежуточные значения с аналогичными значениями, полученными на образцовой ЭВМ, находим блоки с неисправностями.

2.4.1. Тест на базе алгоритма быстрого возведения в степень, подготовка тестовых данных

Программа для обычного персонального компьютера, которая генерирует открытые и закрытые ключи, а также получает верный ответ, рассмотрена далее.

В начале программы задается число бит в открытом ключе *N_bits*, число вычисляемых степеней *K* и начальный аргумент для факториала (значения степени) *F*0 (листинг 3).

Листинг 3. Задание параметров программы

```
typedef mpz _ class mpz;
#define to _ C(x) ((x).get _ mpz _ t())
mp _ bitcnt _ t N _ bits = 376;
unsigned long K = 32;
unsigned long F0 = 10000;
// Два больших простых числа и их
// произведение
mpz p, q, N;
```

Получение больших простых чисел осуществляется путем генерации случайного числа с необходимым количеством двоичных разрядов и последующим нахождением ближайшего, следующего за ним простого числа (листинг 4). Нахождение простого числа осуществляется с помощью функции *mpz_nextprime()* из библиотеки GNU GMP.

Листинг 4. Нахождение большого простого числа

```
mpz getBigPrime() {
   static gmp _ randclass *rc = nullptr;
   if (!rc) {
     rc = new gmp _ randclass(gmp _ randinit _ default);
     rc->seed(time(NULL));
   }
   mpz res, rand = rc->get _ z _ bits(N _ bits/2);
   mpz _ nextprime(to _ C(res),to _ C(rand));
// Необязательная проверка на простоту числа res
assert(mpz _ probab _ prime _ p(to _ C(res),100));
   return res;
}
```

Открытый ключ получается путем перемножения двух псевдослучайных больших простых чисел (листинг 5). В качестве закрытого ключа выступает сумма двух простых чисел.

Листинг 5. Получение открытого ключа

```
void generateTestParams() {
   p = getBigPrime();
   q = getBigPrime();
   N = p * q;
   cout << N<<"\t"<<(p + q)<<"\n";
}
```

Вычислительное ядро теста состоит из цикла, в котором элементы определенной последовательности чисел возводятся в некоторую достаточно большую степень (листинг 6). Наличие информации о значении закрытого ключа позволяет многократно понизить показатель степени, взяв от нее остаток по модулю $(p - 1)(q - 1)$, что и обусловливает значительную разницу между быстрым и медленным способами получения результата.

Листинг 6. Вычислительное ядро теста

```
mpz compute(mpz N, mpz b) {
   mpz res = 0, pow, a = 2, a _ pow;
   mpz _ fac _ ui(to _ C(pow),F0);
   for (unsigned long i = 0; i < K; i++) {
   pow *= i+1;
   // Если известен закрытый ключ, используем
//его для понижения степени
   if (b)
     pow % = N - b + 1;
   // Вычисляем и аккумулируем очередную
// степень числа a _ i по модулю N
   mpz _ powm(to _ C(a _ pow), to _ C(a++),
to _ C(pow), to _ C(N));
   res ^= a _ pow & 0xFFFFFFFF;
   }
   return res;
}
```

Процедура запуска теста получает (по умолчанию — нулевой) закрытый ключ *b*, опционально проверяет соответствие закрытого ключа открытому и далее вызывает основную функцию *compute()*.

Проверка на соответствие ключей осуществляется путем решения квадратного уравнения, так как нам известны произведение и сумма *p* и *q* и проверки тождества $N = p \times q$ для полученных корней (листинг 7).

Листинг 7. Проверка на соответствие ключей и вызов вычислительного ядра теста

```
bool runTest(mpz b = 0) {
   if (b) {
     mpz d = b*b - N*4, sq _ d;
     mpz _ sqrt(to _ C(sq _ d),to _ C(d));
     p = (b - sq _ d)/2;
     q = (b + sq _ d)/2;
     if (p*q != N) {
     cerr << "Закрытый ключ не соответствует
открытому\n";
       return false;
     } else {
       cerr << "Закрытый ключ успешно проверен\n";
     }
   }
   cout << "Ответ: 0x" << hex << compute(N,b)
<< "\n";
   return true;
}
```

Стартовая процедура генерирует открытый и закрытый ключи и запускает вышеуказанную процедуру *runTest* как с указанием, так и без указания закрытого ключа в случае необходимости (листинг 8).

Листинг 8. Стартовая процедура теста

```
void run _ rsa _ test() {

   {

     WITH _ TIMER("gen _ rsa _ test _ params");
     generateTestParams();
   }
   {
     WITH _ TIMER("fast _ rsa _ test");
     bool ok = runTest(p + q);
     assert(ok);
     }
   {
     WITH _ TIMER("slow _ rsa _ test");
     runTest();
   }
}
```

В качестве образцовой ЭВМ, используемой для проверки теста на базе алгоритма быстрого возведения в степень, использовались системы на базе процессоров Intel. Разработанный тест выводит на экран сгенерированные псевдослучайные числа *N*, $p + q$ и результат проведения вычислений в быстром

и медленном режимах (листинг 9). Выбранный секретный ключ должен иметь такую разрядность, чтобы современные суперЭВМ не могли разложить число *N* на множители за разумное время.

Листинг 9. Сгенерированные псевдослучайные числа N, p + q и результат вычислений

```
34732280776870011113890033885984743438654336447
22156868909943828921325393485466698-
-60155902725035722822506816079 27
45352278461432774906624198968918436071629125050
4052583408
Time of gen _ rsa _ test _ params : 11 ms

Ответ: 0x2b0daf20
Time of fast _ rsa _ test : 9 ms

Ответ: 0x2b0daf20
Time of slow _ rsa _ test : 1377 ms
```

Совпадения результатов, полученных при медленном и быстром (при известном закрытом ключе) способах вычислений косвенно свидетельствуют об исправности образцовой ЭВМ. Реальные вычислительные операции совершаются с разными показателями степени, что изменяет, в случае динамического распараллеливания, порядок вычислений и промежуточные результаты в используемом алгоритме быстрого возведения в степень.

Масштабирование параметров теста, т. е. увеличение общего числа операций по возведению в степень и разрядности ключей, приводит к росту времени получения правильного ответа на образцовой ЭВМ. Когда продолжительность быстрого способа получения результата достигнет нескольких часов, время медленного способа уже превысит один месяц. Ускорение времени получения результата без закрытого ключа потребует вычислительную мощность, в сотни раз превосходящую производительность образцовой ЭВМ.

2.4.2. Тестирование many-core-систем

В зависимости от аппаратной платформы необходимо проводить адаптацию базовой операции теста, т. е. реализовать алгоритм быстрого возведения в степень по большому модулю. Программная реализация должна обеспечивать возможность вычислительным ядрам many-core-систем вычислять выражения независимо и параллельно. Результаты всех базовых выражений следует аккумулировать в произвольном порядке с помощью любой подходящей коммутативной ассоциативной операции, например, с помощью операции XOR. Вычисление единичного базового выражения представлено на листинге 10 в виде псевдокода на языке Python.

Листинг 10. Вычисление единичного базового выражения

```
def pow _ mod _ fast(a,b,m):
   r = 1; a = a % m
   while b > 0:
     if b & 1:
       r = (r*a) % m
     b // = 2
     a = (a*a) % m
   return r
```

Тест не требователен к объему быстрой оперативной памяти, доступному для вычислительных ядер. Однако тестирование оперативной памяти возможно в связи с наличием программной реализации целочисленной арифметики высокой точности для чисел в сотни и тысячи разрядов, которая предполагает сохранение промежуточных результатов в RAM.

Проверка разработанной методики осуществлялась на эмуляторе моделируемой many-core-системы. В эмулятор системы добавлена возможность внесения одиночных ошибок в операцию умножения (для инструкций семейства MUL). Внесение каждой ошибки сопровождается печатью сообщения, что позволяет оценить общее число ошибок.

Адаптация базовой операции разработанного теста использует классический алгоритм быстрого возведения в степень по заданному модулю (листинг 11) с использованием фрагментов кода, реализованных на базе библиотеки BN [14].

Листинг 11. Процедура быстрого возведения в степень для эмулятора моделируемой many-core-системы

```
void pow _ mod _ faster(struct bn* a, struct bn* b,
                        struct bn* n, struct bn* res)
{
   bignum _ from _ int(res, 1); /* r = 1 */

   struct bn tmpa;
   struct bn tmpb;
   struct bn tmp;
   bignum _ assign(&tmpa, a);
   bignum _ assign(&tmpb, b);

   while (1) {

     if (tmpb.array[0] & 1) /* if (b % 2)*/
     {
       bignum _ mul(res, &tmpa, &tmp); /*r = r * a% m */
       bignum _ mod(&tmp, n, res);
     }
     bignum _ rshift(&tmpb, &tmp, 1); /* b / = 2 */
     bignum _ assign(&tmpb, &tmp);

     if (bignum _ is _ zero(&tmpb))
       break;

     bignum _ mul(&tmpa, &tmpa, &tmp);
     bignum _ mod(&tmp, n, &tmpa);
   }
}
```

С использованием этой процедуры быстрого возведения в степень основной цикл теста приобретает простой вид (листинг 12).

Листинг 12. Основной цикл теста для эмулятора моделируемой many-core-системы

```
for (int i=0; i<8; i++) {
   pow _ mod _ faster(&a,&b,&m,&r);
   bignum _ xor(&r,&acc,&acc);
   bignum _ add(&d,&a,&a);
}
```

В связи с малой скоростью программной эмуляции были выбраны небольшие значения модуля *m* и показателя степени *b* (в реальных тестах показатель степени является очень большим числом, например, факториалом большого числа, как было описано выше). В качестве операции свертки была взята операция XOR. В процессе выполнения теста проводится около 100 000 операций умножения. Период внесения ошибок эмулятором установлен и соответствует числу операций умножения. Данный программный тест реагирует на единичные ошибки в операции MUL (листинг 13).

Листинг 13. Реакция теста на ошибки

```
### mbcpu: mul bug
result = f85379831073b69
### Termination on fatal error via tracer
    from core 0x0000,
error: 7

--

### mbcpu: mul bug
result = 10d9793d706836f3
### Termination on fatal error via tracer
    from core 0x0000,
error: -53

--

### mbcpu: mul bug
### CPU#0000 error!
### EXCEPTION type 2 on core 0x0000 (IE=1,
    PC=800005CC..800005D0)

--

### mbcpu: mul bug
result = 10d9793d706836f3
### Termination on fatal error via tracer
from core 0x0000,
error: -53

--

### mbcpu: mul bug
result = bf6f1774e9c7432
### Termination on fatal error via tracer
    from core 0x0000,
error: -4
```

В большинстве запусков тест завершался с неверным результатом. Иногда происходила аварийная остановка вследствие некорректного доступа к памяти, что связано с выходом индекса за границу массива при работе с массивами посредством библиотеки BN, в которой используется операция умножения.

Заключение

Рассмотрены вопросы создания программного теста на основе идей алгоритма RSA для поиска проблемных мест на программном и аппаратном уровнях many-core-систем.

Рассмотрены недостатки стандартных методов тестирования в применении к many-core-системам. Предложен метод построения теста с учетом особенностей физического устройства современных многоядерных архитектур. Специфика предлагаемого метода позволяет локализовать проблемные вычислительные модули и блоки, используя особенности реализации предложенного программного кода при обеспечении эффективно организованной схемы для неоднородных параллельных вычислений. С учетом предложенного метода разработан тест, который способен реагировать на одиночные ошибки в арифметических операциях, результат вычислений которого сложно предугадать и подделать без предварительных значительных вычислений.

Проведена проверка предложенного метода и чувствительности разработанного теста к одиночным ошибкам на эмуляторе моделируемой many-core-системы. В результате такой проверки показана эффективность разработанного метода, пригодного также для локализации неисправностей на уровне вычислительных блоков. Для локализации проблем на низком уровне, т. е. непосредственно на уровне кода, следует использовать методы низкоуровневой отладки, частичное описание которых было представлено авторами в работе [15].

Список литературы

1. **Lipp M., Schwarz M., Gruss D., Prescher T., Haas W., Mangard S., Kocher P., Genkin D., Yarom Yu., Hamburg M.** Meltdown // CoRR. 2018. ArXiv e-prints. URL: https://arxiv.org/abs/1801.01207.

2. **Kocher P., Genkin D., Gruss D., Haas W., Hamburg M., Lipp M., Mangard S., Prescher T., Schwarz M., Yarom Yu.** Spectre attacks: exploiting speculative execution // CoRR. 2018. ArXiv e-prints. URL: https://arxiv.org/abs/1801.01203.

3. **Larabel M.** Ryzen-Test & Stress-Run Make It Easy To Cause Segmentation Faults On Zen CPUs. 2017. URL: https://www.phoronix.com/scan.php?page=news_item&px=Ryzen-Test-Stress-Run.

4. **Nicely T. R.** Pentium FDIV flaw FAQ. 2011, URL: http://www.trnicely.net/pentbug/pentbug.html.

5. **Koo H., Mishra P.** Coverage-driven functional test generation for processor validation using formal methods // UR-Korea Conference on Science, Technology and Entrepreneurship (UKC), New Jersey, August 10—13, 2006. URL: http://ukc.ksea.org/ukc2006/index.html.

6. **Chen L., Dey S.** Software-based self-testing methodology for processor cores // IEEE Transactions on CAD of Integrated Circuits and Systems. 2001. Vol. 20, No. 3. P. 369—380.

7. **Müller M. S., Juckeland G., Jurenz M. and Kluge M.** Quality assurance for clusters: acceptance-, stress-, and burn-in tests for general purpose clusters // High Performance Computing and Communications. Springer Berlin Heidelberg. 2007. P. 44—52.

8. **Bakhtiari M., Maarof M. A.** Serious Security Weakness in RSA Cryptosystem // International Journal of Computer Science Issues. 2012. Vol. 9, No. 3. P. 175—178.

9. **Lenstra A. K., Lenstra H. W.** The Development of the Number Field Sieve // Lecture Notes in Mathematics. 1993. Vol. 1554. URL: https://link.springer.com/book/10.1007 %2FBFb0091534.

10. **Parker D.** Elliptic curves and lenstra's factorization algorithm. University of Chicago: REU 2014. URL: http://math.uchicago.edu/~may/REU2014/REUPapers/Parker.pdf.

11. **Pollard J. M.** Theorems on factorization and primality testing // Mathematical Proceedings of the Cambridge Philosophical Society. 1974. Vol. 76. P. 521—528. DOI: 10.1017/S0305004100049252.

12. **Bierbrauer J., Charalambides C. A., Cohen H., Frey G., Colbourn C. J., Dinitz J. H., Furino S., Miao Yi., Yin Ji., Hankerson D., Harris G. H. A., Mollin R. A., Rosen K. H., Avanzi R. M., Doche C., Lange T., Nguyen K. and Vercauteren F.** Handbook of elliptic and hyperelliptic curve cryptography. Discrete Mathematics and Its Applications. Boca Raton, Chapman & Hall. 2006. 44 p.

13. **Манин Ю. И., Панчишкин А. А.** Введение в современную теорию чисел. М.: МЦНМО, 2013. 552 с.

14. **A small** multiple-precision integer implementation in C. URL: https://github.com/kokke/tiny-bignum-c.

15. **Елизаров С. Г., Лукьянченко Г. А., Марков Д. С., Роганов В. А.** Средства отладки программного обеспечения многоядерных процессоров // Программная инженерия. 2017. Т. 8, № 12. С. 531—542.

Functional and Performance Validation of Many-Core Systems Based on the Ideas of the RSA Algorithm

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

Corresponding author:

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

Received on June 28, 2018
Accepted on July 16, 2018

The article deals with a practical example of developing a software test based on the ideas of cryptographic algorithms for searching of problematic areas in the software and the hardware of high-performance many-core systems that contain hundreds of asynchronously interacting processing cores. Suitable cryptographic algorithms are public-key ciphers where computational complexity of hacking and decryption differ by many orders of magnitude. An example on the basis of the RSA algorithm is considered. The RSA algorithm is based on the computational complexity of integer factorization problem. A method for developing software test is proposed taking into account the specific features of processing cores, memory and the communication environment of many-core systems. Such a test can be scaled and parallelized achieving predictable and easily variable execution time. An example test was developed using the described method and it is able to respond to single errors in arithmetic operations, and the result of its calculations can be hard to forecast without preliminary significant calculations, but could be checked on any simple Personal Computer easily. The method and sensitivity of the developed test to single errors have been tested on the emulator of many-core system. The possibility to introduce single errors in the multiplication operation for testing was implemented in this system emulator. In most cases the result was wrong with indicating an incorrect operation that demonstrates the effectiveness of the developed method.

Keywords: *many-core systems, dynamic parallelization of calculations, localization of errors in equipment, RSA algorithm*

For citation:

Elizarov S. G., Lukyanchenko G. A., Markov D. S., Monakhov A. M., Roganov V. A. Functional and Performance Validation of Many-Core Systems Based on the Ideas of the RSA Algorithm*, Programmnaya Ingeneria,* 2018, vol. 9, no. 9, pp. 404—414

DOI: 10.17587/prin.9.404-414

References

1. **Lipp M., Schwarz M., Gruss D., Prescher T., Haas W., Mangard S., Kocher P., Genkin D., Yarom Yu., Hamburg M.** Meltdown, *CoRR*, 2018, ArXiv e-prints, available at: https://arxiv.org/abs/1801.01207.

2. **Kocher P., Genkin D., Gruss D., Haas W., Hamburg M., Lipp M., Mangard S., Prescher T., Schwarz M., Yarom Yu.** Spectre attacks: exploiting speculative execution, *CoRR*, 2018, ArXiv e-prints, available at: https://arxiv.org/abs/1801.01203.

3. **Larabel M.** Ryzen-Test & Stress-Run Make It Easy To Cause Segmentation Faults On Zen CPUs. August 2017, available at: https://www.phoronix.com/scan.php?page=news_item&px=Ryzen-Test-Stress-Run.

4. **Nicely T. R.** Pentium FDIV flaw FAQ. August 2011, available at: http://www.trnicely.net/pentbug/pentbug.html.

5. **Koo H., Mishra P.** Coverage-driven functional test generation for processor validation using formal methods, *US-Korea Conference on Science, Technology and Entrepreneurship (UKC),* New Jersey, August 10—13, 2006, available at: http://ukc.ksea.org/ukc2006/index.html.

6. **Chen L., Dey S.** Software-based self-testing methodology for processor cores, *IEEE Transactions on CAD of Integrated Circuits and Systems*, 2001, vol. 20, no. 3, pp. 369—380.

7. **Müller M. S., Juckeland G., Jurenz M., Kluge M.** Quality assurance for clusters: acceptance-, stress-, and burn-in tests for general purpose clusters, *High Performance Computing and Communications*, Springer Berlin Heidelberg, 2007, pp. 44—52.

8. **Bakhtiari M., Maarof M. A.** Serious Security Weakness in RSA Cryptosystem, *International Journal of Computer Science Issues*, 2012, vol. 9, no. 3, pp. 175—178.

9. **Lenstra A. K., Lenstra H. W.** The Development of the Number Field Sieve, *Lecture Notes in Mathematics*, 1993, vol. 1554, available at: https://link.springer.com/book/10.1007 %2FBFb0091534.

10. **Parker D.** *Elliptic curves and Lenstra's factorization algorithm,* University of Chicago: REU 2014, available at: http://math.uchicago.edu/~may/REU2014/REUPapers/Parker.pdf.

11. **Pollard J. M.** Theorems on factorization and primality testing, *Mathematical Proceedings of the Cambridge Philosophical Society*, 1974, vol. 76, pp. 521—528. DOI: 10.1017/S0305004100049252.

12. **Bierbrauer J., Charalambides C. A., Cohen H., Frey G., Colbourn C. J., Dinitz J. H., Furino S., Miao Yi., Yin Ji., Hankerson D., Harris G. H. A., Mollin R. A., Rosen K. H., Avanzi R. M., Doche C., Lange T., Nguyen K., Vercauteren F.** Handbook of elliptic and hyperelliptic curve cryptography, *Discrete Mathematics and Its Applications*, Boca Raton, Chapman & Hall, 2006, 44 p.

13. **Manin Yu. I., Panchishkin A. A.** *Vvedeniye v sovremennuyu teoriyu chisel* (Introduction to the modern theory of numbers), Moscow, MTsNMO, 2013, 552 p. (in Russian).

14. **A small** multiple-precision integer implementation in C, available at: https://github.com/kokke/tiny-bignum-c.

15. **Elizarov S. G., Lukianchenko G. A., Markov D. S. and Roganov V. A.** Sredstva otladki programmnogo obespecheniya mnogoyadernykh protsessorov (Debugging software for many-core processors), *Programmnaya ingeneria*, 2017, vol. 8, no. 12, pp. 531—542 (in Russian).

ИНФОРМАЦИЯ

УДК 004.82 DOI: 10.17587/prin.9.415-424

К. И. Костенко, канд. физ.-мат. наук, зав. кафедрой, e-mail: kostenko@kubsu.ru,
Кубанский государственный университет, г. Краснодар

Инженерия когнитивного синтеза для систем искусственного интеллекта

Изучается возможность адаптации концептов систем искусственного интеллекта к инвариантам математических систем формализмов представления знаний и использования таких инвариантов для конструирования сценариев решения профессиональных задач. Инварианты являются аналогами сущностей слабо формализованных когнитивных целей разных типов и реализующих такие цели процессов мышления. Процессы основаны на многообразиях знаний, определяющих содержание областей профессиональной деятельности в формате онтологий, и конструируются из морфизмов составления сложных семантических структур с помощью элементов онтологий. Такие операции формализуются как морфизмы нескольких базовых типов. Они соответствуют элементам процессов мышления и согласованы с инвариантами формализмов знаний. Семейство классов морфизмов формируется как результат адаптации элементов фундаментальных математических систем к форматам формализмов знаний и является аналогом многообразия паттернов в объектно-ориентированном проектировании. Приведен пример композиции морфизмов разных классов для решения прикладной задачи планирования производства изделий по онтологии области профессиональной деятельности.

Ключевые слова: *когнитивная цель, когнитивная операция, онтология, формализм знаний, искусственный интеллект, морфизм, синтез знания*

Введение

Основой содержания всякой области профессиональной деятельности является многообразие первичных знаний в форме объектов регулярной структуры. Комбинации таких объектов применяют для нахождения решений разных профессиональных задач. Унифицированным форматом представления содержания областей знаний является онтология в языке дескрипционной логики. Отдельную онтологию составляют иерархии классов неделимых объектов (элементарных знаний) и отношений между такими классами (семантических связей). Пары, составляющие отдельные отношения, называют простыми знаниями. Элементарные знания соответствуют именованным сущностям моделируемой области деятельности, обладающим разнообразными свойствами. Содержание всякого элементарного знания раскрывается через содержащие это знание простые знания. Решение профессиональных задач с помощью первичных знаний понимается как достижение когнитивных целей, реализуемое с помощью процессов синтеза сложных знаний из элементов онтологии. Применяемый далее формат представления синтезируемых знаний соответствует структурам конфигураций формализма абстрактного пространства знаний [1]. Знания в данном формализме (конфигурации) представляются нагруженными бинарными деревьями. Висячие вершины таких деревьев размечены элементарными знаниями (элементарными конфигурациями), а внутренние — отношениями, выполняющимися между знаниями, представляемыми левым и правым поддеревьями таких вершин. Формат структурного представления конфигураций равносилен универсальному формату алгебраических структур фрагментов знаний в произвольных формализмах знаний, конструируемых как композиции фрагментов знаний [2].

Регулярные структуры сложных знаний

Решение профессиональных задач с помощью многообразий формализованных знаний состоит в построении связных семантических структур из элементов постановок задач и онтологий областей знаний. Абстрактный уровень моделирования процесса синтеза реализует общие представления о сложных семантических структурах, конструируемых как достижения отдельных целей из системы целей области профессиональной деятельности. Этот уровень оперирует системой общих формализованных классов операций над формализованными знаниями, связанных с реализациями когнитивных целей [3]. Определения операций являются алгебраическими. Они основаны на функциональных соответствиях элементов областей определения и значений операций [4]. Алгоритмический аспект моделирования операций синтеза реализаций когнитивных

целей основан на пошаговых описаниях процессов преобразования интеллектуальных объектов иерархической структуры. Для этого применяется общий язык правил синтеза, детализирующих содержание процессов трансформаций структур знаний [2]. Такой язык не связан явно с семантикой когнитивных операций и целей. Это приводит к трудностям моделирования схем процессов мышления. Поэтому для решения профессиональных задач с помощью синтеза сложных семантических представлений оправдано использование структур знаний и операций, адаптированных к формализованным моделям мышления. Комбинации таких операций позволяют конструировать процессы достижения произвольных когнитивных целей. Примерами унифицированных структур сложных знаний являются окрестности и серии фрагментов знаний, интегрирующие сущности, связанные с заданными знаниями конкретными отношениями [2, 4].

Классификация операций над знаниями

Первичная иерархия типов абстрактных операций (далее — морфизмов) над формализованными знаниями связана с системой классов операций из разных областей математики, адаптированных с инвариантами формализмами знаний. Многообразие типов морфизмов согласовано с классификацией слабо формализованных когнитивных целей и операций, предложенной Б. Блюмом [3, 4]. Иерархия типов морфизмов включает классы теоретико-множественных морфизмов (*ST*), морфизмов структуризации (*S*) и адаптации (*A*) [4]. Часть класса *ST* составляют морфизмы селекции простых знаний (*STF*) из онтологии области знаний, применяемых для синтеза структур сложных знаний. Основные подклассы класса *S* соответствуют семействам морфизмов декомпозиции (*SD*), моделирующих замены элементарных знаний на фрагменты их структурных представлений, а также морфизмов расширения (*SE*), реализующих разные схемы вставки фрагментов знаний в синтезируемые структуры. Частями семейства морфизмов структуризации является подкласс перестановок фрагментов сложных знаний. Обозначенный последним класс *A* представлен подклассами морфизмов свертки (*AC*), моделирующих преобразования замены структурных фрагментов на значения специальных функций от этих фрагментов, а также трассирований (*AT*), соответствующих преобразованиям извлечения связных семантических подструктур из структур сложных знаний. Вторичные типы морфизмов синтеза знаний представляют преобразования построения окрестностей элементарных знаний заданной глубины (*O*), реализуемые как комбинации первичных морфизмов из классов селекции, структуризации и вставки [4]. Абстрактные определения морфизмов перечисленных классов уточняют их как семейства одноместных морфизмов. Области определения и значения таких морфизмов выбираются из фиксированного семейства видов интеллектуальных объектов, связанных с инвариантами формализмов знаний и формализма абстрактного пространства знаний.

Если семантика морфизмов некоторого класса предполагает использование многоместных наборов знаний заданных типов в качестве областей определения (значения), то область определения (значения) таких морфизмов определяется как декартово произведение соответствующих типов интеллектуальных объектов. В абстрактных формализмах знаний элементы таких произведений моделируются бинарной операцией композиции пар фрагментов представлений знаний. Поэтому абстрактную иерархию классов одноместных морфизмов в произвольных формализмах знаний дополняет единственный двуместный морфизм композиции фрагментов представлений знаний, являющийся инвариантом таких формализмов [1].

Примерами унифицированных типов областей определения и значений морфизмов когнитивных операций являются многообразия: онтологий области знаний (Δ), конечных множеств знаний (*S*) и структурных представлений знаний (Σ). При этом элементами Δ являются иерархии семейств элементарных и простых знаний (онтологии). Класс Σ составляют алгебраические структуры знаний в произвольных формализмах знаний. Они представляются бинарными деревьями, листья которых размечены фрагментами знаний, а внутренние вершины — операцией композиции фрагментов. Для абстрактных пространств знаний класс Σ составляют полные структурные представления (ПСП) конфигураций. Они также моделируются бинарными деревьями, висячие вершины которых размечены элементарными конфигурациями, а внутренние вершины — отношениями, выполняющимися между конфигурациями, представляемыми левым и правым поддеревьями таких вершин. Множества вершин (висячих вершин) структуры произвольной конфигурации абстрактного пространства знаний $z \in \Sigma$ обозначаются как $D(z)$ ($O(z)$). Разметка вершины $\alpha \in D(z)$ структурного представления $z \in \Sigma$ обозначается как $[z]_\alpha$ [1]. Дополнительные типы данных, удобные для уточнения описания морфизмов специальных типов, составляют множества предикатов на множествах объектов произвольных типов (*P*), конечных двоичных наборов в качестве вершин иерархических структур (*I*), семантических отношений между знаниями и структурами знаний (*R*).

Включение в структуры элементов областей определения и значения морфизмов над абстрактными знаниями элементов вспомогательных множеств позволяет конкретизировать специальные классы морфизмов с помощью фиксированных значений дополнительных параметров.

Иерархия классов морфизмов реализует содержательно полное абстрактное многообразие видов абстрактных инструментов обработки знаний. Такие инструменты применяют для моделирования отдельных этапов процессов достижения когни-

тивных целей разных типов [4]. Содержанием этих процессов является синтез структур знаний из элементов онтологий. Реализация конкретной когнитивной цели в форме синтезированного знания достигается применением подходящей комбинации морфизмов из заданных абстрактных классов, адаптированных к области профессиональной деятельности. Связанное с адаптацией сужение классов морфизмов реализуется уменьшением общности определений, получаемым с помощью дополнительных структурных и функциональных параметров и соотношений. Дополнительные параметры увеличивают размерность наборов начальных данных областей определений морфизмов. В уточнениях определений морфизмов такая детализация моделируется декартовыми произведениями общих и дополнительных типов данных.

Уточнение морфизмов селекции (извлечения множества интеллектуальных объектов с заданными свойствами из конкретных онтологий или структур сложных знаний) реализуется с использованием дополнительного параметра, принимающего значения из класса P. Общие форматы морфизмов селекции имеют вид $s{:}\Delta\times P\to S$ и $s{:}\Sigma\times P\to S$.

Формат класса морфизмов замены задается выражением $r{:}\Sigma\times I\times\Sigma\to S$. Обращение к таким морфизмам осуществляется с использованием записей $r(z_1, \alpha, z_2)$, где $z_1, z_2 \in \Sigma$ и $\alpha \in O(z_1)$. Применение этого морфизма состоит в замене размеченной висячей вершины α структуры z_1 на структуру z_2.

Морфизмам вставки соответствует формат $i{:}\Sigma\times I\times\Sigma\times\{0, 1\}\times R\to S$. Обращение к таким морфизмам записывается в виде $i(z_1, \alpha, z_2, \sigma, \rho)$ и обозначает результат выполнения вставки структуры z_2 в структуру z_1. Для класса структурных представлений конфигураций параметр α из последнего выражения уточняет корень поддерева, перемещаемого в вершину $\alpha\bar{\sigma}$. Структура z_2 добавляется в структуру значения морфизма с корнем в вершине $\alpha\sigma$. Вершина α размечается отношением $\rho \in R$. Для класса алгебраических структур знаний в произвольных формализмах знаний вставка связана с заменой фрагмента $(z_1)_\alpha$ на фрагмент $\big((z_1)_\alpha \circ \rho\big)\circ z_2$, если $\sigma = 0$, и фрагмент $\big(z_2 \circ \rho\big)\circ (z_1)_\alpha$, если $\sigma = 1$ [4].

Морфизмы перестановки моделируются как последовательности элементарных перестановок. Всякой элементарной перестановке соответствует формат $d{:}\Sigma\times I\times I\to\Sigma$, с обращением к морфизму с помощью выражений вида $d(z, \alpha, \beta)$, где $\alpha, \beta \in D(z)$. Результатом элементарной перестановки является структура конфигурации с переставленными поддеревьями $(z)_\alpha$ и $(z)_\beta$ с корневыми вершинами α и β. Частный случай перестановок заданного множества компонентов структурных представлений конфигураций определяется с использованием формата $d{:}\Sigma\times I^*\to\Sigma$, где I^* — перестановка префиксного семейства двоичных наборов из I^*, составленных из элементов множества $D(z)$, где $z \in \Sigma$. Она определяет перестановку семейства фрагментов структурного представления z, реализуемую данным морфизмом. Если $(\alpha_1, ..., \alpha_k) \in I^*$, а $(\beta_1, ..., \beta_k)$ — последовательность $(\alpha_1, ..., \alpha_k)$, упорядоченная лексикографически, то морфизм d заменяет фрагмент $(z)_{\beta_i}$ на $(z)_{\alpha_i}$, $i = 1, ..., k$.

Морфизмами адаптации моделируются операции преобразования (применения содержания) фрагментов знаний начальных данных таких морфизмов. Общий формат морфизмов адаптации представляется конструкцией $a{:}\Sigma\to\Sigma$. Результаты применения морфизмов адаптации к произвольным начальным данным нельзя составить только как перестановки фрагментов структурных семантических представлений. Рассматриваемые далее морфизмы адаптации двух типов реализуют вычисление значений алгебраических и логических выражений, а также моделирование отдельных шагов логического вывода [4]. Реализация алгебраического (логического) вычисления возможна, если структура знания включает представление арифметического (логического) выражения, а также значения всех его параметров. Результат адаптации такого знания получается заменой составляющих его компонентов на значение выражения на заданных значениях его параметров. При моделировании логического вывода фрагментами конфигурации z представляется набор логических формул, к которым применимо правило вывода. Результат адаптации получается заменой заданных фрагментов на результат применения правила.

Синтез знания, составляющего решение отдельной задачи, реализуется как последовательность применения морфизмов из базовых классов, выбираемых по некоторой схеме. Примерами типовых схем являются построения серий знаний из содержания онтологии или из множеств сложных знаний, составленных на основе онтологии [4].

Трассирование

Трассирование является универсальным способом извлечения целостных фрагментов сложных знаний из синтезированной иерархической семантической структуры. Управление процессом трассирования в каждом конкретном случае основано на шаблоне структуры извлекаемого знания, а также требованиях к синтезированным семантическим представлениям, из которых извлекаются целостные структуры знаний. Шаблоны и требования позволяют конкретизировать отображения трассирования, размечающие элементы синтезированных структур знаний. Процессы извлечения знаний из знаний связаны с конструированием конкретных отображений трассирования, удовлетворяющих подходящим структурным и семантическим ограничениям. Трассирования являются аналогом операций селекции, переносимых с теоретико-множественных на иерархические структуры знаний, представляемых конфигурациями абстрактных пространств знаний. Частный случай трассирования конфигурации z_1 в конфигурацию z_2 связан с существованием отображения $\xi{:}I\to I$, для которого:

1) $\forall\alpha,\beta\in I\big(\alpha\subseteq\beta\to\xi(\alpha)\subseteq\xi(\beta)\big)$;

2) $\forall\alpha \in I\sigma \in \{0,1\}(\xi(\alpha) \subset \xi(\alpha\sigma) \to \xi(\alpha\sigma) = \xi(\alpha)\sigma\beta)$;

3) $\forall\alpha \in D(z_1)\backslash O(z_1)\big(\xi(\alpha) \in D(z_2)\backslash O(z_2)\big)$;

4) $\forall\alpha \in O(z_1)\big(\xi(\alpha) \in O(z_2)\big)$;

5) $\forall\alpha \in D(z_1)\backslash O(z_1)\big([z_1]_\alpha \subseteq [z_2]_{\xi(\alpha)}\big)$;

6) $\forall\alpha \in O(z_1)\big([z_1]_\alpha\, \rho_0\, [z_2]_{\xi(\alpha)}\big)$.

Первые два из приведенных условий определяют ξ как изотонное отображение, сохраняющее направление переходов для элементарных путей в $D(z_1)$, ведущих из корня в листья при изменении значений ξ. Условия 3 и 4 означают, что при трассировании конфигураций внутренние (висячие) вершины структурных представлений конфигурации z_1 соответствуют внутренним (висячим) вершинам конфигурации z_2. Заключительные условия состоят в сравнимости разметок сопоставляемых вершин в отношениях вложения отношений и сравнении элементарных знаний ρ_0. С трассированиями конфигураций связаны эндоморфизмы конфигураций, конструирующие z_1 по z_2. Эндоморфизмами представляются разные схемы извлечения знаний из знаний. Частные случаи эндоморфизмов конфигураций основаны на трассированиях растяжения и сжатия [1].

Рассмотрим пример применения композиций морфизмов абстрактных классов рассмотренной модели когнитивного синтеза в инженерии процессов решения профессиональных задач для конкретных областей деятельности.

Содержание области знаний производства изделий

Рассмотрим модель производства партии изделий конечного семейства типов $\{\tau_1, ..., \tau_k\}$ из заготовок. Изготовление каждого изделия осуществляется последовательно операциями O_1, ..., O_{d-1}, O_d. Эта последовательность — линейная, одна и та же для всех типов изделий. Для производства одного изделия типа τ_i операциями O_1, ..., O_{d-1} реализуются две последовательности промежуточных видов обрабатываемой заготовки этого изделия. При выполнении операции O_j заготовка типа τ_i из первой (второй) последовательности проходит трансформацию от промежуточного вида $R_{i,j-1}(r_{i,j-1})$ к виду $R_{i,j}(r_{i,j})$. Операция O_d осуществляет окончательную сборку изделий каждого типа τ_i, агрегирующих изделия $R_{i,d-1}$ и $r_{i,d-1}$.

Заказ на изделия формируется как список значений числа заказанных изделий разных типов: m_1 изделий типа τ_1, ..., m_k изделий типа τ_k, где m_1, ..., m_k — целые неотрицательные числа. Сведения о заказе представляются простыми знаниями вида "τ *заказано m*". Для обработки заготовок применяют разные станки. Станков одного типа может быть несколько. У каждого типа станков свой перечень типов обрабатываемых заготовок и выполняемых операций. При обработке заготовок каждого типа с помощью подходящего станка может потребоваться переналадка (внутренняя) или изменение оснастки (установка дополнительного внешнего оборудования). Во втором случае известны виды оснасток, устанавливаемых на станках разных типов. Число оснасток каждого вида фиксировано. Время переналадки (переоснастки) зависит от типа станка, значения заменяемой наладки (оснастки) или устанавливаемой наладки (оснастки).

Параметрами каждого станка являются выполняемые операции, типы обрабатываемых заготовок, указатель на использование специальных оснасток или переналадка при изменении типа обрабатываемых заготовок. Дополнительные сведения о станках включают данные производительности при обработке заготовок отдельных типов, время переналадки (переоснастки) станков при изменении типа обрабатываемых заготовок, время загрузки и выгрузки заготовок заданного типа. Первичные знания представим с помощью онтологии в формате когнитивной карты, приведенной на рис. 1.

Вершины данной карты представляют классы объектов, а отношения между классами представляются ориентированными дугами, размеченными именами таких отношений.

Специальные ограничения для работы станков включают значения максимального и минимального допустимого количества одновременно загружаемых заготовок всякого вида при выполнении соответствующей операции. Дополнительные ограничения относятся также к значению числа работников, перечням и нормативам производительности выполняемых ими операций по каждой операции, типу станка и изделия.

Задача синтеза плана по заказу и модели области знаний

Рассмотрим задачу синтеза плана исполнения заказа по онтологии, представленной на рис. 1. Исходными данными этой задачи являются количества заказанных изделий разных типов и база первичных знаний о показателях производства изделий, сведения об остатках заготовок разных типов, а также графики работы предприятия и отдельных сотрудников на достаточно продолжительный период в формате онтологии. Предполагается наличие достаточного числа еще не обрабатывавшихся заготовок для изготовления изделий каждого типа в заказанном количестве.

Содержание плана составляют последовательности выполнения операций на разных станках, обеспечивающих изготовление заказа. Построение плана осуществляется в несколько этапов. Каждый этап реализуется с использованием элементов рассмотренных ранее классов когнитивных операций. Уточним схему процесса составления плана. Он основан на конструировании сложных знаний из элементов онтологии с помощью базовых операций когнитивного синтеза.

Этап 1. Уточнение параметров заказа. На этапе 1 выполняется интеграция начальных данных постановки задачи в серию, формируемую за несколько последовательных шагов. Синтез начинается с постро-

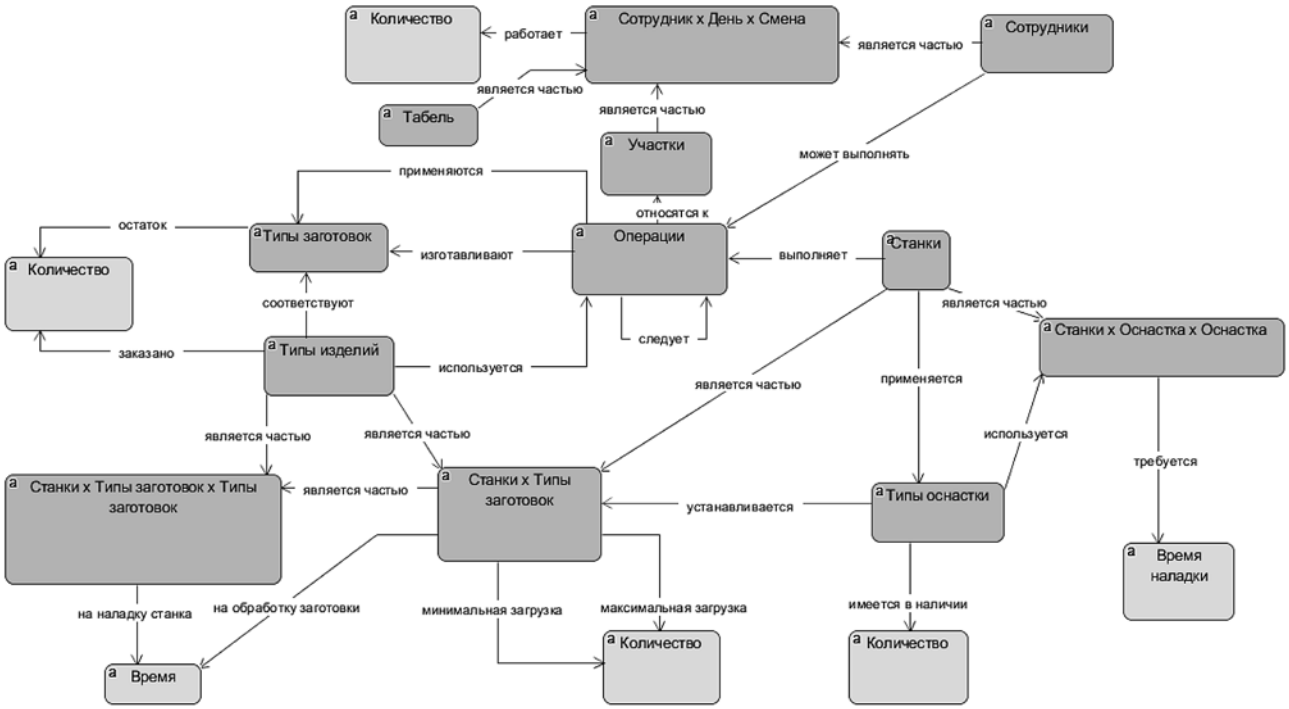


Рис. 1. Когнитивная карта содержания области деятельности

ения конфигурации, представленной элементарным знанием, соответствующим понятию плана L. Затем формируется окрестность сущности L радиуса 1, составленная простыми знаниями о заказе. Для этого последовательно применяются операции селекции элементов заказа, ранжирования отобранных элементов в неупорядоченный список и вставки списка в формируемую конфигурацию. Неупорядоченный список наименований изделий извлекается из отношения "*заказано*", связывающего классы "типы изделий" и "количество" как множество пар, в которых вторая компонента отлична от нуля. Структура получаемой в результате конфигурации приведена на рис. 2. Здесь Λ — пустая конфигурация, применяемая для маркировки признака завершения списка, а L — обозначение синтезируемого плана. Символ ⋮ обозначает отношение принадлежности параллельной (неупорядоченной) серии.

Следующий шаг процесса интеграции элементов начальных данных реализует добавление в построенную конфигурацию фрагментов, содержащих сведения о последовательностях операций в процессах изготовления изделий каждого заказанного типа. Эти фрагменты формируются как окрестности отдельных типов заказанных изделий. Они состоят из элементов класса "операции", которые связаны с типами изделий отношением "*использует*". Структура такой окрестности для отдельного типа изделий τ_i приведена на рис. 3. Встраивание построенной окрестности z'' в синте-

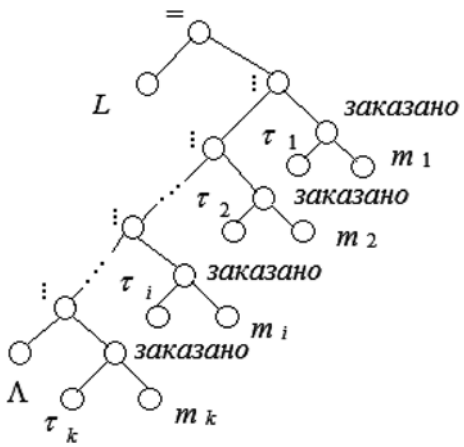


Рис. 2. Формирование списка заказанных изделий

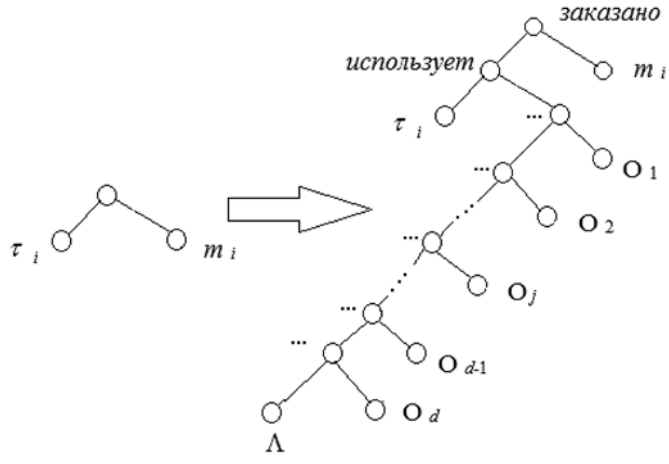


Рис. 3. Последовательность операций изготовления изделий типа τ_i

зированный на первом шаге фрагмент выполняется с помощью операций $r(z', 10^i10, z'')$ — замены висячей вершины 10^i10 конфигурации z' на z''. При этом разметки вершин вида $10^i10 \in D(z)$, где z — конфигурация на рис. 1, переносятся в вершины 10^i100. Символ $\cdots$ обозначает отношение принадлежности последовательной (упорядоченной) серии.

Для рассматриваемого в работе случая все серии операций являются одинаковыми. Операции $O_1, ..., O_d$ составляют унифицированную серию для каждого типа заказанных изделий τ_i. При этом $O_1, ..., O_{d-1}$ реализуют $d - 1$ шагов независимой обработки двух видов заготовок типа τ_i. Заключительная операция O_d осуществляет сборку изделий типа τ_i, из пар заготовок видов $R_{i,\,d-1}$ и $r_{i,\,d-1}$, обработанных операцией O_{d-1}.

Результаты следующего шага этапа 1 состоят в уточнении числа изделий каждого типа, которые должны быть произведены отдельными операциями обработки заготовок этого типа. При этом учитываются остатки изготовленных ранее заготовок видов $R_{i,\,j}$ и $r_{i,\,j}$. Значения остатков обоих видов заготовок каждого типа заказанных изделий обозначаются как $W_{i,\,j}$ и $w_{i,\,j}$. Сведения об остатках заготовок изделий таких типов и видов содержатся в онтологии. Унифицированные комбинации морфизмов из базовых классов реализуют вставки в построенные серии операций (для каждого типа заказанных изделий) дополнительных сведений (для каждой операции) о количестве недостающих заготовок, которые должны быть изготовлены этой операцией. Комбинации морфизмов начинаются с селекции из онтологии сведений об остатках заготовок двух видов для каждой операции и типа заказанных изделий. Затем морфизмы адаптации знаний реализуют вычисление серий значений $Q_{i,\,1}, ..., Q_{i,\,d}$ и $q_{i,\,1}, ..., q_{i,\,d}$ с помощью правил $Q_{i,\,j} = m_i - W_{i,\,j}$ и $q_{i,\,j} = m_i - w_{i,\,j}$, $j = 1, ..., r$. Еще одна операция класса адаптации знаний формирует серии вспомогательных значений $C_{i,\,d-j} = C_{i,\,d-j+1} - W_{i,\,d-j}$ и $c_{i,\,d-j} = c_{i,\,d-j+1} - w_{i,\,d-j}$ для возрастающих значений j.

Построение каждой из двух последних серий реализуется полностью или до первого по порядку элемента в ней, для которого значение $C_{i,\,d-j} = C_{i,\,d-j+1} - W_{i,\,d-j}$ ($c_{i,\,d-j} = c_{i,\,d-j+1} - w_{i,\,d-j}$) удовлетворяет условию $C_{i,\,d-j} \leqslant 0$ ($c_{i,\,d-j} \leqslant 0$). В таком случае остальные элементы последовательностей $\{Q_{i,j}\}$ ($\{q_{i,j}\}$) полагаются равными 0. Обозначим число ненулевых элементов в $Q_{i,\,1}, ..., Q_{i,\,d}$ и $q_{i,\,1}, ..., q_{i,\,d}$ для изделия τ_i как L_i (l_i).

Типовой фрагмент семантической структуры, реализуемой на рассматриваемом шаге этапа 1, приведен на рис. 4.

Построение этой структуры реализуется последовательностью морфизмов селекции значений остатков заготовок двух видов для морфизмов адаптации (вычисления значений по формулам $Q_{i,\,j} = m_i - W_{i,\,j}$ и $q_{i,\,j} = m_i - w_{i,\,j}$), а также вставки вычисленных значений в иерархическую структуру (см. рис. 3). Перечисленные операции многократно

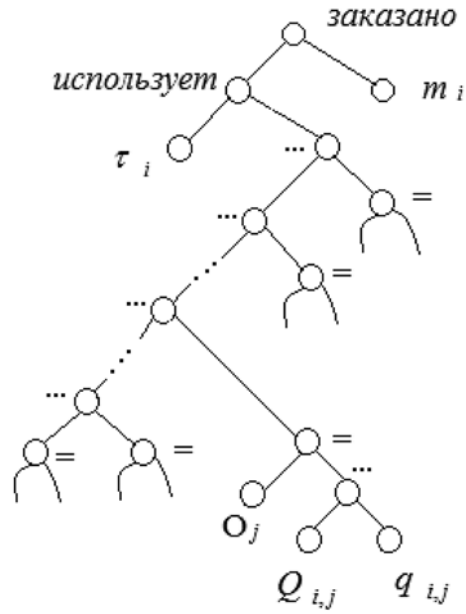


Рис. 4. Структура заказа по операциям для одного вида изделий

применяются, реализуя замену всех висячих вершин 10^i10^j10 на композиции фрагментов конфигураций $\left(O_j \circ =\right) \circ \left(\left(Q_{i,j} \circ \cdots\right) \circ q_{i,j}\right)$. Здесь $i + 1$ — порядковый номер изделия в заказе, а $j + 1$ — номер операции в последовательности операций, выполняемых при исполнении заказа на производство изделия τ_i.

На рис. 4 символ "=" обозначает отношение равенства значения параметра. Заключительная операция этапа уточнения параметров заказа по типам изделий и по выполняемым операциям состоит в удалении из полученной иерархической структуры вершин, размеченных значениями $m_1, ..., m_k$. Для этого применяется эпиморфизм конфигураций, представленный трассированием растяжения $\xi: I \to I$, определяемым соотношением

$$\xi(\alpha) = \begin{cases} 10^i10^j10\beta, & \text{если } \alpha = 10^i10^j1\beta \\ \alpha, & \text{иначе} \end{cases}.$$

Значения i и j в последнем соотношении соответствуют типу имеющейся номенклатуры изделий и типу операций обработки заготовок, общим для всех типов заготовок.

Этап 2. Интеграция знаний о ресурсах процесса исполнения заказа. Этап 2 синтеза плана обеспечивает составление и встраивание в конструируемую иерархическую семантическую структуру знаний о станках. Такую структуру составляет параллельная серия окрестностей отдельных станков. Эта серия составляет семантическую структуру z', которая добавляется к структуре, реализованной на этапе 1, с помощью морфизма вставки $i(z', \lambda, z'', 0, :)$. В новой структуре элементы $\alpha \in I$ структуры z' переносятся в элементы 0α, а элементы $\alpha \in I$ структуры z'' — в элементы 1α. Корень полученной структуры размечается специальным отношением $\perp$ [1].

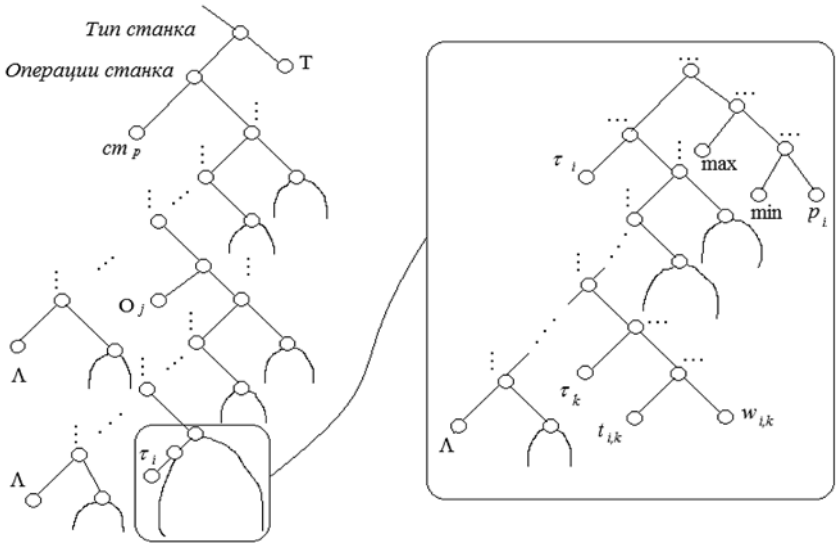


Рис. 5. Семантическая структура атрибутов станков

Содержание добавляемой структуры определяет многообразие атрибутов отдельных станков разных типов, включение которых в синтезируемую конфигурацию реализуется в форме серии окрестностей отдельных станков, аккумулирующих необходимые для составления плана сведения о каждом станке. Формат окрестности одного станка изображен на рис. 5.

Окрестность всякого станка составляет неупорядоченная серия обозначений операций, выполняемых этим станком. Окрестности отдельных операций для серии станков составляют неупорядоченные серии сведений о типах изделий, обрабатываемых на соответствующих станках, отдельно для каждого вида операций. Элементами таких сведений являются данные об используемых для этого оснастках (настройках), а также время перенастройки (переоснастки) станков с одного типа изделий на другой.

В левой части рис. 5 изображен общий вид части иерархической интегрированной структуры знаний об отдельном станке cm_p (представлен на рис. 5 отдельной вершиной). Значение T уточняет способ настройки станка cm_p с одного типа изделий на другой (переналадка или замена оснастки). Окрестность радиуса 1 станка cm_p составляет параллельная серия знаний о выполняемых на этом станке операциях отдельно для каждого типа изделий. В правой части рис. 5 изображена типовая структура элемента серии данных о станке cm_p. В ней детализирован выделенный фрагмент из левой части рис. 5. Знания об операции O_j для cm_p и типе изделия τ_i включают значения максимальной (max) и минимальной (min) однократной загрузки станка заготовками типа τ_i при выполнении операции O_j, а также производительности станка для заданных O_j и τ_i.

Остальную часть знаний о станке cm_p в правой части рис. 5 составляет параллельная серия сведений о времени перенастройки станка на переход к обработке заготовок типа τ_k. Время перенастройки станка с изделия τ_i на изделие τ_k задается как $t_{i,k}$. Значение квалификации выполняющего такую перенастройку сотрудника представлено значением $w_{i,k}$. Формирование рассматриваемой структуры знаний может быть реализовано с помощью морфизмов базовых классов конструирования и вставки окрестностей заданной структуры в неупорядоченную серию станков.

Дополнительная система знаний, добавляемая в синтезируемую структуру, содержит сведения о числе рабочих дней и смен, а также квалификации сотрудников, выполняющих разные операции управления станками. Она приведена на рис. 6. В графике работы предприятия содержатся сведения о датах рабочих дней, их количествах и планируемых выходах сотрудников. График упорядочен по рабочим дням. Каждый день работ планируется в одну или две смены. Для смен в конкретные дни уточняются списки сотрудников, работающих в этой смене. Значения времени начала и завершения первой и второй смен фиксированы и не изменяются.

В левой части рис. 6 приведена структура фрагмента знания, являющегося прямой суммой двух структур $(ST \circ \vdots) \circ SP$, содержащих график работ ST (табель) и данные о квалификации сотрудников SP (персонал) соответственно. Левые потомки корневых вершин структур ST и SP размечены словами "*Табель*" и "*Персонал*".

Фрагмент SP включает серию радиуса 1 сущности "*персонал*" в отношении параллельной серии $\vdots$, он интегрирует структуры окрестностей радиуса 1 для элементов класса "сотрудник", связанных отношением "может выполнять" с элементами класса "операции". Данный фрагмент реализуется как параллельная серия окрестностей отдельных сотрудников, составленных параллельными сериями операций, выполняемых этими сотрудниками.

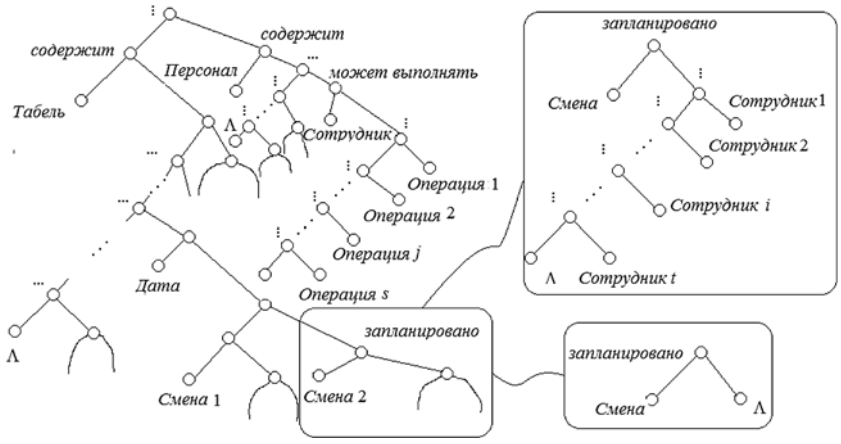


Рис. 6. Квалификация и график работы сотрудников

Фрагмент *ST* образует упорядоченная по рабочим дням серия планов выхода сотрудников в отдельные дни в две смены. Общая структура такого плана для одной смены приведена на рис. 6 справа вверху. Если в некоторый день планируется работа в одну смену, то отсутствующая вторая смена представляется типовой "пустой" структурой, изображенной в нижней правой части рис. 6.

Разнообразие профессиональных операций сотрудников включает переоснастку и переналадку станков, загрузку и выгрузку заготовок.

Этап 3. Структура плана выполнения заказа. План выполнения заказа составляется как структура конфигурации абстрактного пространства знаний, добавляемая к структуре конфигурации, составленной из фрагментов для рассмотренных этапов синтеза. План реализуется как неупорядоченная серия серий операций, выполняемых для отдельных станков при реализации заказа. Он включает описания периодов занятости станков и сотрудников при выполнении запланированных операций. Построение плана осуществляется по шагам. На очередном шаге для каждого типа изделий, производство которого еще не включено в общий план, определяется схема выполнения операций переналадки, загрузки и работы станков. Схема составляет план выполнения заказа по заданному типу изделия и дополняет уже построенный фрагмент плана. Из всех синтезированных вариантов продолжения плана отбирается один по условию минимального значения времени завершения изготовления партии изделий еще одного типа. Этим критерием реализуется условие оптимизации по атрибуту плана, выбираемого из разнообразия аналогичных критериев оптимальности, реализуемого без использования переборных методов.

Общая структура плана выполнения заказа приведена на рис. 7. Она имеет вид параллельной серии из планов работ отдельных станков в формате серий описаний последовательно выполняемых операций. Описания операций в сериях имеют фиксированный формат. Его составляют описания операций настройки (*A*), загрузки (*B*), обработки заготовок (*C*) и выгрузки заготовок изделий (*D*), реализующих полный процесс обработки партии заготовок фиксированного типа на заданном станке.

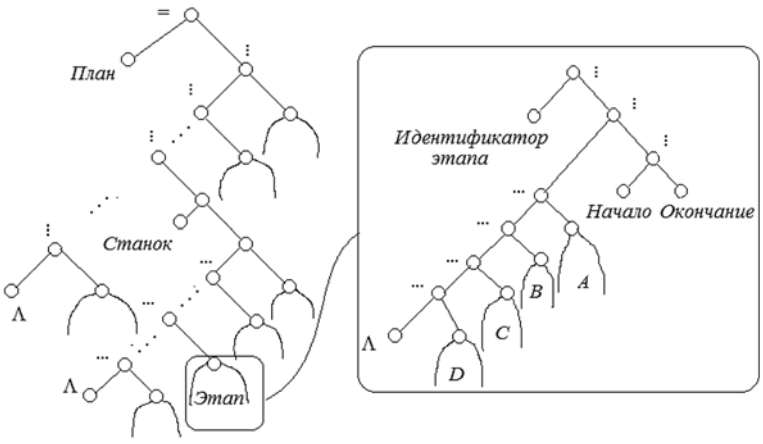


Рис. 7. Структура плана выполнения заказа

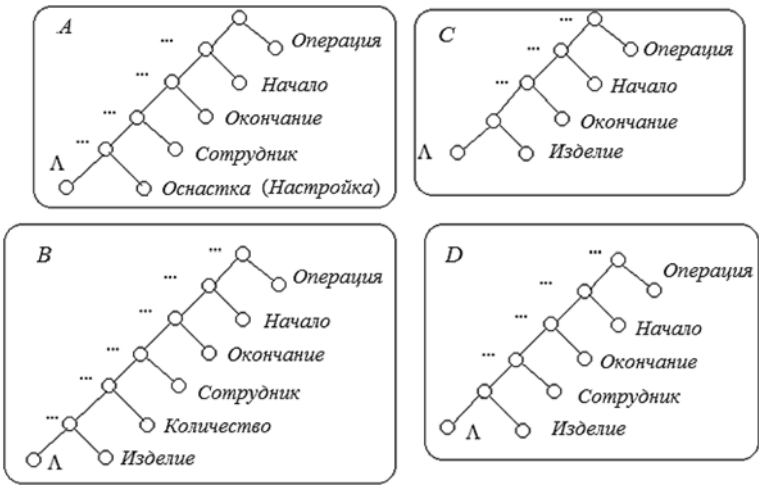


Рис. 8. Фрагменты этапов описания операции

Описание этапа выполнения операции включает сведения о количестве обрабатываемых заготовок, времени начала и завершения, типе применяемой на этапе оснастки (если нужно), а также сотрудниках, привлекаемых для работы со станком. Шаблоны структур бинарных деревьев для описаний всех четырех этапов операции изображены на рис. 8.

Процесс извлечения знаний из онтологии завершает интеграция рассмотренных фрагментов в прямую сумму $(((\mathrm{I} \oplus \mathrm{II}) \oplus \mathrm{III}) \oplus \mathrm{IV})$, где I, II, III, IV — области заказа, атрибутов станков, графика работы и персонала, а также плана работ соответственно. Корневыми вершинами этих областей являются двоичные наборы 000, 001, 01 и 1. Шаблон соответствующей конфигурации приведен на рис. 9.

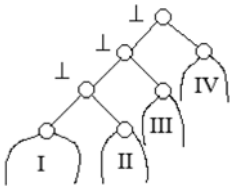


Рис. 9. Шаблон синтезируемой структуры

Здесь ⊥ обозначает "пустое" отношение, связывающее любые пары знаний в формате конфигураций абстрактного пространства знаний.

Синтез плана изготовления заказа

Построение плана выполнения заказа осуществляется с использованием элементов классов морфизмов селекции, адаптации, вставки, замены и трассирования [4]. При этом трассированием реализуется удаление вспомогательных элементов из синтезированной семантической структуры. Сам план формируется в области IV рис. 9. Начальное содержание плана представляется последовательной серией пустых планов работ отдельных станков. Последующее постро-

ние плана выполняется по шагам. Результатом каждого шага является план, дополненный схемой изготовления изделий еще одного типа. В этой области формируется серия вариантов продолжения плана. Из таких вариантов отбирается один, рассматриваемый как результат выполнения шага.

Процесс реализации одного шага построения плана начинается с конструирования параллельной серии одинаковых, уже построенных фрагментов плана по одному для каждого из типов заказанных изделий, еще не включенных в этот фрагмент. Для каждого элемента серии выполняется расширение, связанное с добавлением схемы изготовления заказанного числа изделий соответствующего типа. Из реализованных вариантов расширения построенного фрагмента плана выбирается первый в серии, выполнение которого завершается раньше других.

Расширение фрагмента плана для τ_i составляют две последовательности фрагментов планов работы станков, обеспечивающих изготовление недостающего числа младших и старших заготовок этого изделия с помощью операций O_a, ..., O_{d-1} и O_b, ..., O_{d-1}, а также операции O_d — сборки изделий типа τ_i из заготовок двух видов (младших и старших). Пусть элемент серии фрагментов плана соответствует расширению реализованной части плана схемой изготовления изделий типа τ_i, которое должно собираться из двух видов заготовок (младших и старших), изготавливаемых в больших нуля количествах q_a, ..., q_{d-1} и Q_b, ..., Q_{d-1}.

Рассмотрим такую последовательность для младшего вида, связанную с операциями O_a, ..., O_{d-1}. Пусть реализованы операции O_a, ..., O_{a+t} и $\{m', ..., m^k\}$ — все станки, которые могут выполнять операцию O_{a+t+1} по обработке младших заготовок изделия τ_i. Для каждого выбранного станка составим последовательную серию интервалов времени уже построенного графика работы этого станка, в которые для него не запланировано выполнение каких-либо операций. Интервал включается в серию, если необходимая насадка станка свободна для этого интервала, число имеющихся к началу интервала заготовок для выполнения операции O_{a+t+1} на заданном станке попадает в диапазон значений минимальной и максимальной загрузки, продолжительность промежутка времени достаточна для реализации всех необходимых действий с заготовками рассматриваемого типа и вида, имеются незанятые сотрудники для управления выполнением этих действий. Количество загружаемых деталей для каждого отобранного интервала выбирается максимально возможным. Из найденных интервалов и станков выбирается вариант с минимальным временем завершения операции O_{a+t+1}. В серию плана работ выбранного станка встраивается описание выполнения операции O_{a+t+1} на этом станке в выбранный интервал времени. Там же сохраняется значение числа изготовленных заготовок типа τ_i. Если число изготовленных операцией O_{a+t+1} младших изделий типа τ_i в составленном фрагменте плана не превосходит $c_{i,\,a+t+1}$, то планируется дополнительное выполнение этой операции. В противном случае выполняется переход к реализации операции O_{a+t+2}.

Дополнительно проводится пересчет значения числа заготовок младшего вида типа τ_i для обработки операциями O_{a+t+1} и O_{a+t+2}. Число планируемых к изготовлению заготовок для O_{a+t+1} уменьшается на значение запланированной загрузки станка. Число заготовок для O_{a+t+2} увеличивается на это же значение с момента завершения выгрузки обработанных заготовок.

Расширение плана невозможно, если число имеющихся младших заготовок типа τ_i для выполнения O_{a+t+1} меньше необходимого для загрузки любого станка из $\{m', ..., m^k\}$. В таком случае выполняется процедура изготовления недостающего числа изделий τ_i для обработки операцией O_{a+t+1}. Она состоит в нахождении ближайшей от O_{a+t+1} операции обработки изделий типа τ_i с меньшим индексом, которая может быть выполнена. После этого процесс продолжается по той же схеме, начиная с последней выполненной операции.

Построение списка фрагментов плана для старших заготовок типа τ_i осуществляется аналогично синтезу плана для младших заготовок этого типа.

Завершающая операция O_d выполняется при условии наличия достаточного числа заготовок двух видов, чтобы обеспечить загрузку соответствующих станков.

Построим серию вариантов расширения фрагмента плана для разных не включенных в план типов заготовок. Из таких расширений с помощью подходящего трассирования из серии расширений выбирается один вариант. Он соответствует плану с минимальным значением времени завершения. Трассирование завершает шаг процесса синтеза плана.

Оптимизация приведенной схемы возможна при условии использования специальных правил исполнения этапов составления плана. Такие правила связаны с дополнительными требованиями взаимозаменяемости и равномерности загрузки сотрудников, оптимизации их числа, потребности приобретения дополнительных квалификаций, уменьшения числа избыточно производимых заготовок разных типов отдельными операциями.

Заключение

Синтез сложных знаний составляет основу моделирования процессов решения профессиональных задач в системах искусственного интеллекта. Он состоит в интерпретации знаний сложной структуры как результатов процессов составления связных семантических представлений из элементов онтологий. Для моделирования операций синтеза востребован язык конструирования процессов интеграции и трансформации семантических структур знаний. В работе рассмотрены элементы такого конструирования с помощью аналогов компонентов процессов мышления. Они связаны с системой формализованных классов морфизмов, дополненных инструментами комбинирования и адаптации операций разных типов в схемы достижения когнитивных целей. Приведенный пример применения морфизмов из этих классов демонстрирует возможность использования композиций морфизмов базовых классов в качестве шаблонов описаний процессов решения профессиональных задач. Содержательная полнота классификатора морфизмов и согласованность с инвариантами класса формализмов знаний означает универсальность возможности адаптации средств данного языка к процессам синтеза сложных знаний. Морфизмы базовых классов являются аналогами паттернов в объектно-ориентированном проектировании [5]. Они составляют качественно новые

средства инженерии знаний, обеспечивающие возрастание уровня инструментов, применяемых для конструирования сложных интеллектуальных процессов и структур.

Работа выполнена при поддержке РФФИ, проект № 16-01-00214.

Список литературы

1. **Костенко К. И.** Формализмы представления знаний и модели интеллектуальных систем. Краснодар: Кубанский гос. ун-т, 2015. 300 с.

2. **Костенко К. И.** Моделирование оператора вывода для иерархических формализмов знаний // Программная инженерия. 2016. Т. 7, № 9. С. 424—431.

3. **Bloom B. S., Engelhar, M. D., Furst E. J., Hill W. H., Krathwohl D. R.** Taxonomy of educational objectives: The classification Taxonomy of educational goals. Handbook 1: Cognitive domain. — New York: David McKay, 1956.

4. **Костенко К. И.** Операции когнитивного синтеза формализованных знаний // Программная инженерия. 2018. Т. 9, № 4. С. 174—184.

5. **Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.** Приемы обьектно-ориентированного проектирования. Паттерны Проектирования. СПб.: Питер, 2016. 366 с.

Engineering of Cognitive Synthesis Morphisms at the Artificial Intelligence Systems

K. I. Kostenko, kostenko@kubsu.ru, Kuban State University, Krasnodar, 350040, Russian Federation

Corresponding author:

Kostenko Konstantin I., Head of Chair, Kuban State University, Krasnodar, 350040, Russian Federation, E-mail: kostenko@kubsu.ru

Received on June 19, 2018
Accepted on July 25, 2018

The adaptation of artificial intelligence systems concepts to knowledge representation formalisms' fundamental invariants is studied. The abstract knowledge inclusion and composition invariants of such formalisms used to be the main tools for simulating the knowledge semantic and algebraic structures. Formal analysis of these invariants provides specialists with a new application domain for developing the abstract mathematical systems general theory. This application relates to informal concept of knowledge processing operations and is based on fundamentals of linguistics, psychology, pedagogics and philosophy. Different classes of morphisms, usual for set theory, formal logic, abstract algebra, theory of algorithms and topology, are used for simulating such operations. Universal Blum's classification of cognitive operations and goals used to be the completeness and independence criteria for introduced morphisms classification. Simulating of cognitive processes non-formal invariants by abstract morphisms classification for knowledge representation formalisms is realized by morphisms compositions considered as patterns of cognitive goals realizations within formalisms of abstract knowledge spaces. Such compositions define complex knowledge synthesis processes, based on subject domain elementary and simple knowledge sets, represented by knowledge area ontology. The morphisms formal classification agrees with the different types of weakly formalized cognitive goals and the thinking processes that search for such goals realizations. An application is proposed for morphisms compositions at artificial intelligent systems based on abstract morphisms transformation and realized as morphisms homomorphic extensions into professional tasks solution processes. An example of such a process is provided for creation of the products manufacturing plan. Product of every type is made by the sequence of products processing operations performed on machines with known properties and attributes. Morphisms composition engineering originates the plan-creating process. This process uses three general types of morphisms for complex knowledge synthesis processes represented by knowledge integration, knowledge transformation and knowledge tracing.

Keywords: *cognitive goal, cognitive operation, ontology, knowledge representation formalism, artificial intelligence, morphism, knowledge synthesis*

Acknowledgements: *The Russian Foundation for Basic Research supported this work, project nos. 16-01-00214*

For citation:

Kostenko K. I. Engineering of Cognitive Synthesis Morphisms at the Artificial Intelligence Systems*, Programmnaya Ingeneria,* 2018, vol. 9, no. 9, pp. 415—424.

DOI: 10.17587/prin.9.415-424

References

1. **Kostenko K. I.** *Formalizmy predstavlenija znanij i modeli intellektualnyx sistem* (Knowledge representation formalisms and intelligent systems models), Krasnodar, Kuban state university, 2015, 300 p. (in Russian).

2. **Kostenko K. I.** Modelirovanije operatora vyvoda dlja ierarxiheskix formalizmov znanij (Simulation of Inference Operator for Hierarchical Knowledge Representation Formalisms), *Programmnaya Ingeneria*, 2016, vol. 7, no. 9, pp. 424—431 (in Russian).

3. **Bloom B. S., Engelhart M. D., Furst E. J., Hill W. H., Krathwohl D. R.** Taxonomy of educational objectives: The classification Taxonomy of educational goals. Handbook 1: Cognitive domain. New York, David McKay, 1956.

4. **Kostenko K. I.** Operacii kognitivnogo cinteza formalizovannyx znanij (Operations of Formalized Knowledge Cognitive Synthesis), *Programmnaya Ingeneria*, 2018, vol. 9, no. 4, pp. 174—184 (in Russian).

5. **Gamma E., Helm R., Johnson R., Vlissides J.** *Prijomy obyektno-orientirovannogo programmirovanija. Patterny Proeknirovanija*, Saint-Petersburg, Piter, 2016, 366 p. (in Russian).

УДК 004.912, 004.942 DOI: 10.17587/prin.9.425-432

И. С. Пименов, магистрант, e-mail: pimenov.1330@yandex.ru, Новосибирский государственный университет, г. Новосибирск, **Н. В. Саломатина**, канд. физ.-мат. наук, ст. науч. сотр., e-mail: salomatina_nv@live.ru, Институт математики им. С. Л. Соболева СО РАН, г. Новосибирск

Распознавание интенций потребителей товаров и услуг в сообщениях пользователей социальных сетей

Решается задача поиска и извлечения явно выраженных намерений (интенций) пользователей социальных сетей купить товар или воспользоваться услугой. Предлагаемый подход отличает применение правил в виде ориентированных графов, построенных автоматически на основе аннотированных экспертом текстов сообщений. Извлечение информации об объекте интенции и его свойствах проводится путем анализа текстов согласно построенным графам, аккумулирующим лексическую, грамматическую и семантическую информацию из экспертной разметки.

Ключевые слова: *интенция, маркер интенции, интенциональный блок, ориентированный граф, извлечение фактов из текстов*

Введение

В настоящее время социальные сети стали популярным сегментом интернет-маркетинга. Важной задачей маркетинга является создание баз потенциальных клиентов, которых можно обнаружить в том числе и среди пользователей сетей, оставляющих сообщения, свидетельствующие о намерении (интенции) приобрести тот или иной товар или услугу. Проблема обнаружения интенций в текстах сообщений относится к области интеллектуального (фактографического) анализа.

Структура факта включает именованные сущности, которые могут иметь некоторые свойства и иметь связи и/или взаимодействовать с другими сущностями. К самым распространенным типам фактов относят такие как: 1) *объект и его характеристика*, например, "товар" — "цена", "человек" — "должность" и т. п.; 2) *объект и действие* (назначение на должность, купля-продажа акций и пр.) [1]. Структура факта во многом зависит от конкретных анализируемых сущностей, поэтому практически для каждого нового факта приходится разрабатывать или модифицировать модель, учитывающую разнообразие способов его выражения в естественном языке. Потребительские интенции можно отнести к фактам типа 1 — объектам и их свойствам. Однако следует отметить, что объект интенции, как правило, характеризуется не одним, а несколькими разнотипными свойствами, которые могут быть как информативными, так и неинформативными. В настоящем исследовании реализован алгоритм, разделяющий свойства объекта интенции.

К часто используемым формализмам для описания фактов относят: 1) регулярные выражения, к примеру, такие как в языке Jape [2]; 2) контекстно-свободные грамматики, применяемые, например в Томита-парсер [3]; 3) шаблоны; фреймы [4, 5]; 4) решетки, графы [6]. Наглядность представления содержания и структуры факта присущи последним в большей степени.

Строятся как статистические модели извлечения сущностей и связей, так и базирующиеся на правилах (*rule-based*) (см., например, обзор [7]). Последние могут быть закодированы вручную (*hand-coded*) или извлечены из аннотированных текстов (*learning-based*). Они предполагают наличие правил маркировки объекта, включая тегирование (указание роли, грамматических характеристик), контекстные правила и пр.

В подходах, построенных на машинном обучении [8], выделяют методы, опирающиеся на обучение с учителем, *supervised machine learning*, к примеру, классификация на основе вероятностных и векторных моделей текста (наивный Байесовский классификатор, метод опорных векторов) и обучение без учителя, *unsupervised machine learning* (различные методы кластеризации, нейронные сети).

Среди комбинированных методов можно отметить итерационный подход [9, 10]. В этом случае построенные экспертом шаблоны обогащаются именами сущностей и связями между ними путем поочередного поиска в текстовых коллекциях известных сущностей с неизвестными связями и известных связей с неизвестными сущностями. В настоящей работе реализован подход, который, с одной стороны, использует экспертные знания (*rule-based*), с другой стороны, реализует автоматическое построение правил распознавания (*learning-based*). Как и многие методы, опирающиеся на знания экспертов,

он является трудоемким в подготовке обучающих коллекций, применяемых для построения модели. Однако в этом случае сравнительно небольшой объем обучающей коллекции обеспечивает хорошее качество модели.

Некоторые известные готовые программные продукты, например, Leadsift [11], Qualia [12], не проводят анализ текста, они базируются на экстралингвистической информации: отслеживании этапов развития интереса к конкретным брендам, подсчете "лайков", "эмотиконов", "хэштегов" и др., оставленных пользователем, совершение им покупок и т. п. Другие, такие как SoCeDo [13], работают с текстами на уровне ключевых слов. Имеются аналоги для русского языка, такие как LeadScanner [14], Leaderator [15], It's great [16]. Основные различия у перечисленных продуктов состоят в том, что они, во-первых, функционируют в разных сетях, к примеру: Twitter, ВКонтакте, во-вторых, работают с разным числом тематических классов (It's great с 7, Leaderator с 15 и LeadScanner с 20). Кроме того, каждый продукт имеет свой набор дополнительных возможностей, например, It's great составляет профиль пользователей, проводит мониторинг их активности, Leaderator работает с сообщениями на двух языках. Общим недостатком для всех отмеченных продуктов является поверхностностный характер осуществляемого лингвистического анализа. Результаты работы программ выдаются в виде списков сообщений, отнесенных к определенным тематическим классам, например, "аренда недвижимости", "бытовой ремонт", "медицинские услуги" и др. Извлечение объекта интенции и идентификация его свойств не предусмотрена. Поэтому в один класс попадают, например, сообщения о намерениях получить консультацию у врачей различных профилей, работающих в разных городах. Конкретизация свойств объекта позволила бы существенно уточнить классификацию.

Основной целью исследования, результаты которого представлены в настоящей статье, является разработка метода, учитывающего семантику интенции и позволяющего автоматически формировать правила сетевого типа, пригодные для извлечения объекта интенции и его свойств, с демонстрацией эффективности применения правил к тестовой коллекции в терминах полноты, точности и F-меры.

1. Задача распознавания интенций

Неформально интенция определяется как выраженный эксплицитно или имплицитно факт желания совершить какое-либо действие. Степень эксплицитности может варьироваться: "Хочу устроить фотосессию девушке на этих выходных в Москве", "Подскажите, пожалуйста, хорошего репетитора по русскому языку". Несмотря на различия в форме выражения интенции, и в том и в другом сообщении есть слова-маркеры, указывающие на присутствие интенции в тексте: "хочу" и "подскажите". В настоящей работе мы рассматриваем задачу поиска и извлечения интенций, которые вводятся в текст явно в том смысле, что они обязательно предваряются маркерами.

Для формализации определения интенции требуется ввести понятие *интенционального блока* — конкретной реализации компонента структуры интенции в тексте. Каждый интенциональный блок, являясь частью отвлеченного значения элемента структуры интенции, состоит из отдельных слов или словосочетаний и может быть охарактеризован на лексическом и грамматическом уровнях. Понятие интенции может быть формально определено как множество *допустимых последовательностей интенциональных блоков*. Допустимость определяется данными обучающей коллекции. Степень ее полноты характеризуют показатели качества распознавания. Выявление недостающих образцов конструкций и расширение коллекции сообщениями, их содержащими, проводится путем анализа ошибок распознавания тестового материала.

По оценке эксперта структура интенции может содержать четыре типа интенциональных блоков: *Mr* — маркер интенции; *Ob* — объект интенции; *Pr* — множество свойств объекта; *Em* — множество эмфатических элементов (неинформативных свойств объекта, "формул вежливости" и пр.).

Конструкция G_k любой k-й интенции обязательно включает информацию о маркере и объекте. Интенциональные блоки, содержащие свойства объекта интенции, эмфатические конструкции, относятся к факультативным.

Решение задачи распознавания интенций предполагает реализацию следующих двух этапов: этапа обучения, в ходе которого на базе размеченной экспертом коллекции конструируются правила распознавания интенции, и этапа распознавания, на котором выполняется проверка качества построенных правил на тестовом материале.

Формирование правил обнаружения и извлечения интенций требует выполнения следующих действий:

1) создания обучающей коллекции текстов сообщений, содержащих интенцию;

2) разработки системы семантических тегов и проведения на их основе разметки интенциональных блоков в текстах;

3) проведения лемматизации и морфологического анализа текстов;

4) формирования словаря маркеров интенции;

5) создания правил выделения интенциональных блоков, исследования их вариативности для выявления лексических и грамматических констант, предпочтительных для характеризации блоков;

6) построения G_k для $k = 1, ..., K$ (K — число сообщений в обучающей коллекции);

7) проведения учета допустимой позиционной комбинаторики блоков, возможно, не представленной в обучающей коллекции, но очевидной для эксперта;

8) агрегации G_k согласно близости их структуры.

Этап распознавания является существенно более простой процедурой, его описание будет дано в разд. 3.

2. Этап обучения

В процессе разметки обучающей коллекции эксперт снабжает каждый интенциональный блок текста семантическим тегом в соответствии с его ролью. Все множество тегов разметки определяется исходя из компонентов структуры интенции путем уточнения ролей каждого из типов интенциональных блоков.

Маркер интенции может выражаться как одним интенциональным блоком (снабжается тегом *mrsg*), так и несколькими (*mrpl*), каждый из которых неполон, а значение интенциональности может быть передано лишь всеми блоками в совокупности. Маркеры интенции второго типа способны принимать на себя часть значения объекта (*mrob*). Такие маркеры характеризуются высокой степенью вариативности на лексическом уровне, что отличает их как от маркеров с тегом *mrsg*, так и неполных, с абстрактной семантикой, и отражается в информационной составляющей блоков конструкции G_k. Многословный маркер может иметь вставку — элемент другого типа, например, "где *мне* купить...".

Для объекта интенции всегда может быть указана главная форма (*obmn*), характеризующая его наиболее общее значение. Указатели на объект (*obin*) являются опциональными формами, зависящими от объекта грамматически, но при этом не выражающими сам объект интенции, а лишь на него ссылающимися ("контакты кого-то", "помощь в чем-то"). Дополнительные формы (*obnd*) уточняют значение главной, находятся с ней в подчинительных отношениях и могут быть охарактеризованы как свойства без явной принадлежности.

Дифференциация свойств объекта обусловливается их собственной семантикой независимо от интенционального значения всего предложения. Наконец, эмфатические элементы группируются по их логической отнесенности либо к объекту интенции, либо к ее маркеру, либо ко всей интенциональной конструкции. Как и в случае с маркерами, данное деление необходимо для более точного определения допустимых последовательностей интенциональных блоков.

Все используемые теги разметки перечислены в табл. 1. В последней колонке приведены примеры из текстов сообщений, где маркируемые указанным тегом слова выделены курсивом.

Процедура лемматизации и приписывания грамматических характеристик каждому слову проводится с помощью свободно распространяемых модулей pymorphy2 версии 3.5.2 [17].

2.1. Формирование правил свертки

Известно, что применение сложных анализаторов на подъязыках дает больше ошибок, чем более простых, но настроенных на конкретный подъязык. Как отмечено в работе [18]: "Мощный анализатор будет лишь вносить нежелательный шум, а производительность будет падать". Эта закономерность проявляется особенно ярко на этапе синтаксического анализа.

Таблица 1

Список семантических тегов разметки

Тип блока	Тег	Значение тега	Примеры
Mr:	*mrsg*	Однословный маркер	*хочу*, *подскажите*, *нужен*
	mrpl	Маркер без объектной семантики	*кто знает*, *где купить*
	mrob	Маркер-(часть объекта)	где *сделать* УЗИ
Ob:	*obmn*	Главный объект	посоветуйте *диету*
	obnd	Дополнительные компоненты объекта	подскажите репетитора *по химии*
	obin	Указатель на объект	дайте *координаты* массажиста
Pr:	*pppl*	Локация	ищу ортопеда *в Ростове*
	ppad	Адресат услуги	нужен логопед *для школьника*
	pptm	Срок/дата	ищу фотографа *на 8 марта*
	pppr	Стоимость	хочу взять кредит *около 30 000*
	ppgl	Цель	ищу репетитора *для сдачи ЕГЭ*
	ppqn	Количество	надо *1000* друзей добавить
Em:	*emph*	Формула вежливости	подскажите, *пожалуйста*, врача
	emmd	Неинформативное расширение маркера	куда *лучше/можно* сходить
	emql	Неинформативное качество объекта	подскажите *хорошего/лучшего* врача

Правила "свертки" интенциональных блоков, содержащих более одного слова, фактически представляют собой частичный синтаксический разбор текста сообщения. Под правилом свертки подразумевается упорядоченная пара, первым элементом которой является последовательная совокупность грамматических характеристик всех слов блока, а вторым (выводом) — грамматическая характеристика слова, указанного экспертом в качестве главного. Правила свертки извлекаются из размеченного текста автоматически и демонстрируют высокую степень однозначности вывода для подъязыка потребительских интенций. В приведенных ниже примерах грамматические характеристики даны в нотации OpenCorpora, используемой в pymorphy2:

- *практикующий юрист*: *PRTF, impf, tran, pres, actv, nomn + NOUN, anim, nomn —> NOUN,anim, nomn*;
- *по уголовным делам*: *PREP + ADJF, Qual, ablt + NOUN, inan, datv —> NOUN, inan, datv*;
- *в стабильно работающую финансовую компанию*: *PREP + ADVB + PRTF, impf, intr, pres, actv, accs + ADJF, accs + NOUN, inan, accs —> NOUN, inan, accs.*

Список правил может быть автоматически расширен путем их комбинирования. Так, на основе любых правил свертки A и B, таких, что $A = <(a + b), b>$, $B = < (c + b), c>$, может быть получено новое правило $C = <(c + a + b), b>$ заменой элемента в составе посылки одного из правил (в приведенном примере — B) на посылку, выводящую этот же элемент в составе другого (в данном случае — A).

2.2. Выбор грамматических характеристик и построение словарей

Каждому элементу конструкции G_k соответствует информация о допустимом содержании на трех уровнях: лексическом, грамматическом и интенциональной семантики. Используемые для характеризации сведения зависят от интенционального значения тега: так, 7 семантических тегов из 17 (а именно, *mrsg, mrpl, obin, ppin* и все теги для эмфатических конструкций: *emph, emql, emmd*) подразумевают высокую степень абстрактности допускаемых значений и реализуются в контексте минимальным числом регулярно повторяющихся лексем. Их можно считать лексическими константами. Словари констант формируются из размеченного текста сообщений обучающей коллекции автоматически и проверяются экспертом. Элементы конструкции, снабжаемые другими десятью тегами, напротив, имеют высокую вариативность на лексическом уровне, однако, грамматические характеристики главных словоформ в блоке практически неизменны. Информационное содержание всех элементов конструкции G_k задается на уровне минимальной вариативности.

Отметим, что не все грамматические характеристики являются одинаково значимыми для различения блоков с разной интенциональной семантикой. Так, к числу несущественных грамматических тегов относятся теги, обозначающие род и число, а также специальные теги, такие как *Subx* для маркирования субстантивов, *Abbr* — для аббревиатур, *Fixd* — для неизменяемых слов. Наличие "лишних" грамматических характеристик, не влияющих на различение способов выражения интенционального значения, препятствует отождествлению блоков и приводит к усложнению агрегации G_k. Их присутствие также снижает эффективность применения правил свертки. Набор учитываемых грамматических тегов был сокращен экспертом в процессе тестирования программы распознавания.

2.3. Построение графов интенций

Последовательность интенциональных блоков каждого k-го сообщения, выражающего интенцию, представляется с помощью ориентированного графа $G_k = \{V_k, E_k\}$. Вершины из множества V_k — интенциональные блоки, содержащие информацию об интенциональном теге и лексических и грамматических характеристиках, а ребра из множества E_k соединяют их в порядке следования.

Агрегация графов G_k в $G_{mr} = \mathrm{U}\ G_k$ ($mr = 1, ..., M$, M — число маркеров в словаре маркеров) проводится для каждого маркера mr посредством операции объединения: $G_{mr} = \{\mathrm{U}\ V_k, \mathrm{U}\ E_k\}$. Способ агрегации, указанный, например, в работе [6] и основанный на объединении близких по редакционному расстоянию последовательностей блоков, непригоден в данном случае вследствие высокой вариативности в позиционном положении интенциональных блоков одинаковой семантики. В результате чего сообщения, выражающие близкие по семантике интенции, по редакционному расстоянию оказываются более удаленными, чем сообщения, далекие по семантике.

На рис. 1 приведен пример графа G_{mr}, построенного для $M = 6$ аннотированных сообщений (символ "#" указывает на главное слово в блоке сообщения,

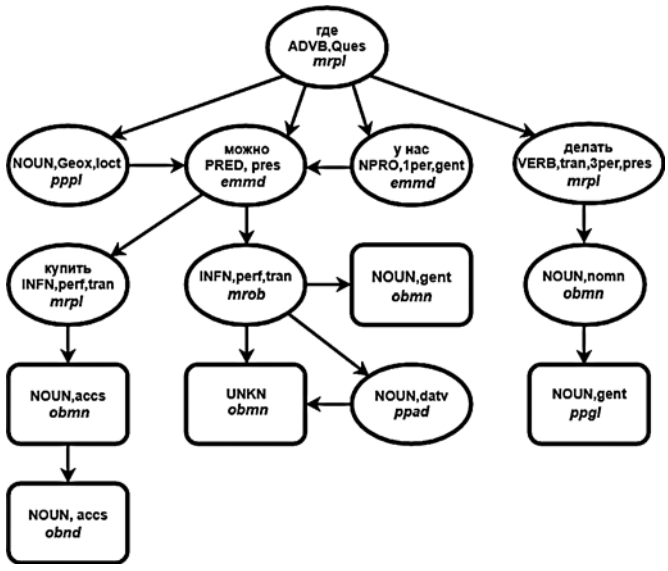


Рис. 1. Граф G_6, построенный для шести сообщений, маркированных "где"

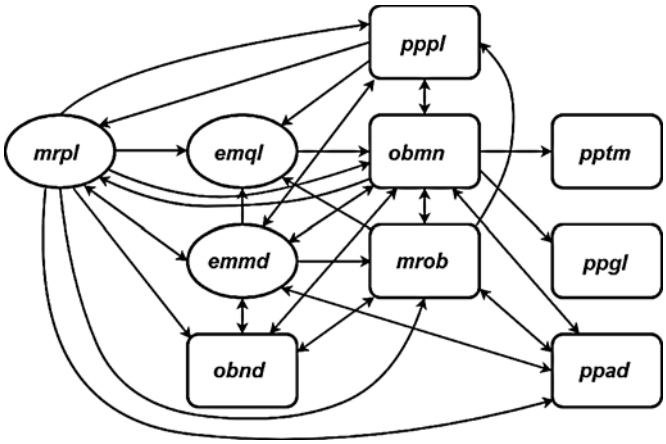


Рис. 2. Граф интенций с маркером *mrpl* = "где"

прямоугольники в графе соответствуют конечным вершинам, овалы — внутренним):

1) [где]*mrpl* [у нас#]*emmd* [можно]*emmd* [купить] *mrpl* [зарядку# на iPhone 5]*obmn*;

2) [где]*mrpl* [можно]*emmd* [купить]*mrpl* [телефон] *obmn* [в кредит#]*obnd*;

3) [где]*mrpl* [в Жлобине#]*pppl* [можно]*emmd* [сделать]*mrob* [МРТ]*obmn*;

4) [где]*mrpl* [делают]*mrpl* [массаж]*obmn* [для коррекции# фигуры]*ppgl*;

5) [где]*mrpl* [можно]*emmd* [сделать]*mrob* [ребенку] *ppad* [ЭКГ]*obmn*;

6) [где]*mrpl* [можно]*emmd* [отремонтировать] *mrob* [компьютерные колонки#]*obmn*.

Обозримость рисунка существенно снижается при увеличении числа сообщений уже до нескольких десятков. Пример графа, аккумулирующего структуры всех интенций обучающей коллекции с начальной частью маркера *mrpl* = "где" и построенного на основе информации только о тегах интенциональных блоков, приведен на рис. 2.

При всей схожести графов интенций, маркированных разными лексемами, они имеют различия в наборе вершин даже в случае, если граф содержит информацию лишь об интенциональных тегах. В частности, данный на рис. 2 граф не может иметь вершину с тегом *obin*. Число циклов в графе зависит от тега начальной вершины: графы с маркером *mrpl*, как правило, имеют большое число циклов.

Визуализация графов осуществляется посредством сторонней библиотеки PyGraphviz [19].

3. Поиск и извлечение интенций

Построенные на этапе обучения графы интенций применяются на этапе распознавания в ходе анализа произвольного сообщения (без какой-либо разметки) с целью выявления в нем интенциональных блоков.

Для каждого сообщения тестовой коллекции проводится процедура поиска интенции, включающая перечисленные далее действия.

1. Предобработка текста, в том числе — лемматизация, морфологический анализ, формирование блоков по правилам свертки, полученным при обучении, извлечение лексической и грамматической информации, характеризующей блоки.

2. Анализ лексической составляющей последовательности блоков, поиск лексем в словаре маркеров. Если маркер найден, осуществляется выбор соответствующего графа интенции, по которому будет проводиться извлечение информации, иначе выдается сообщение об отсутствии явно выраженной интенции в тексте.

3. Анализ информации последовательности блоков согласно содержанию вершин в графе и проверка достижимости конечной вершины. Если лексическая и/или грамматическая информация, соответствующая вершине графа, совпадает с той, что характеризует выделенный в сообщении блок, то осуществляется переход к следующему блоку и проводится поиск подходящей смежной вершины. При отсутствии такой вершины процедура поиска допускает переход к следующему блоку текста g раз для всего пути. На данном этапе принято ограничение $g < 3$. Если при проходе по графу ни разу не была пройдена вершина, имеющая статус конечной, то результатом поиска будет сообщение об отсутствии интенции в тексте. Иначе, исходя из интенциональных тегов, соответствующих вершинам графа, объект интенции и его свойства будут извлечены для анализируемого сообщения.

Работа программы не завершается при обнаружении первой интенции в обрабатываемом сообщении, а продолжается до разбора последнего из выделенных блоков, что обеспечивает обнаружение в тексте любого размера всех интенций, в нем содержащихся.

Для вычисления полноты R, точности P и F-меры поиска интенций использовались принятые в работе [20] оценки качества.

4. Апробация подхода

Обучающая коллекция была составлена экспертом с помощью API LeadScaner [14]. Она содержала 1000 сообщений из социальной сети ВКонтакте. Интенции потребителей относились к девяти разным коммерческим категориям, на каждую из которых приходилось немногим более 100 сообщений. Выделение интенциональных блоков коллекции было проведено в три этапа. Сначала правила свертки, полученные на первой сотне размеченных сообщений, были применены к 500 сообщениям. После коррекции экспертом, обновленный набор правил использовался для разметки оставшейся части коллекции, что существенно облегчило труд эксперта. Всего получено $r_1 = 472$ правила свертки, которые затем применялись при анализе сообщений из тестовой коллекции. Большинство правил (82 %) содержали от двух до четырех слов в посылке. Число автоматически построенных правил составило $r_2 = 3873$. Экспериментально показано, что преимущества автоматически построенных правил проявляются в редких случаях (см. показатели качества в табл. 2). Тегирование интенциональных блоков проведено полностью вручную.

Таблица 2

Показатели качества распознавания интенций в сообщениях тестовых коллекций

Тестовая коллекция, значения g, r,	P, %	R, %	F-мера, %
*C*_793, $g = 0$, $r = 472 + 0$	80	57	67
*C*_793, $g = 0$, $r = 472 + 3873$	82	56	67
*C*_793, $g = 1$, $r = 472 + 0$	80	69	74
*C*_793, $g = 1$, $r = 472 + 3873$	80	67	73
*C*_793, $g = 2$, $r = 472 + 0$	80	71	75
*C*_793, $g = 2$, $r = 472 + 3873$	80	68	74
*C*_387, $g = 0$, $r = 472 + 0$	86	68	76
*C*_387, $g = 0$, $r = 472 + 3873$	86	65	74
*C*_387, $g = 1$, $r = 472 + 0$	82	78	80
***C*_387, $g = 1$, $r = 472 + 3873$**	**83**	**78**	**80**
*C*_387, $g = 2$, $r = 472 + 0$	82	78	80
***C*_387, $g = 2$, $r = 472 + 3873$**	**83**	**78**	**80**

Основная часть словаря маркеров сформирована на текстах, относящихся к первым 3—4 категориям. На данный момент словарь содержит 35 элементов, из них около половины встретились в обучающей коллекции один раз. Число однословных (*mrsg*) и многословных (*mrpl*) маркеров примерно одинаково. Самые частотные маркеры — "подскажите", "посоветуйте", "порекомендуйте" — встречаются в четверти всех примеров из обучающей коллекции и маркируют схожие по структуре интенции, это позволяет объединить соответствующие им графы в один. Группа из четырех наиболее частотных маркеров покрывает уже 63 % коллекции, что демонстрирует регулярность их употребления. Тем не менее при появлении новых тем требовалось, хотя и незначительное, пополнение словаря маркеров. Практически все однословные маркеры (*mrsg*) являются лексическими константами, встречающимися лишь в одной грамматической форме. В составных маркерах (*mrpl*), где первые элементы можно считать лексическими константами, а вариативность вторых ограничена, в основном, на грамматическом уровне, она все же возможна. Четыре грамматических формы являются общими для всех составных маркеров: глагол третьего лица настоящего времени, инфинитив, вопросительное местоимение, глагол-связка "есть", в то же время каждому отдельному маркеру соответствует не более двух из них.

Графы с начальной вершиной, соответствующей самым частотным маркерам, как правило, являются самыми сложными, содержащими большое число вершин и ребер. Семантика маркера тоже играет роль — не все маркеры допускают присутствие интенциональных блоков любого типа. Так, конфигурация графа с маркером "сколько" допускает присутствие лишь 5 из 17 возможных тегов в своем составе. Максимальное разнообразие тегов встречается в графовой конструкции с маркерами "нужно" и "необходимо". Среднее число ребер и вершин в построенных графах отражает разнообразие способов выражения интенции: оно составляет 112 и 38 соответственно.

Для тестирования построенных графов специалистом, непосвященным в цели и задачи проводимого исследования и незнакомым с системой интенциональной разметки, были созданы две представленные далее тестовые коллекции.

1. Коллекция *C*_793. Она состояла из 793 сообщений сети ВКонтакте и содержала потребительские интенции.

2. Коллекция *C*_387. Она включала 387 сообщений. Часть из них содержала интенции, не относящиеся к потребительским, часть не содержала интенций вовсе, но маркер в тексте сообщения присутствовал в той грамматической форме, в которой он встречался в интенциях потребителей. Материал был подобран вручную из Национального корпуса русского языка (основного и газетного подкорпусов) [21] с помощью встроенной поисковой системы.

Результаты экспериментов на тестовых коллекциях — значения полноты R, точности P и F-меры — приведены в табл. 2 (g — число переходов к следующему блоку без анализа содержимого текущего, $r = r_1 + r_2$). В ходе экспериментов выяснилось, что использование автоматически сгенерированных правил оправдано в редких случаях (см. выделенные строки с *C*_387, $g = 1, 2$) и в целом мало влияет на точность распознавания, немного понижая полноту. Возможность пропуска одного блока повышает не только полноту на 10...13 %, но и точность на 2...4 %. Существенное повышение полноты обеспечивается нивелированием различного рода ошибок пользователей и работы морфологического анализатора. Допущение двух пропусков $g = 2$ не дает существенного изменения показателей качества по сравнению с вариантом $g = 1$. Средняя точность распознавания на двух тестовых коллекциях для $g = 1$ и $r = 472$ составила 81 %, полнота — 74 %, F-мера — 77 %.

Ошибки распознавания, выявленные на тестовой коллекции, связаны с вопросами, обусловленными разными уровнями сложности. Ошибок из разряда *false positives* немного, они могут быть отнесены к концептуальным вопросам и являются самыми трудноразрешимыми. Они вызваны принципиальной языковой сложностью выражения интенции как психологической категории. Так, рекламное сообщение, сформулированное в форме вопроса от покупателя, будет по структуре своей совершенно неотличимо от выражения реальной интенции: "Где купить костюм на выпускной? Приходите к нам!".

Почти половина (48 %) ошибок типа *false negatives* сопряжена с неполнотой покрытия графами способов выражения интенции, в частности, с отсутствием вершин для дублированных эмфатических элементов и многословных определительных конструкций. Процедура достраивания графов по размеченным интенциональными тегами сообщениям, содержащим отсутствующие конструкции, включена в программную реализацию.

Доля ошибок, вызванных неверной работой морфологического анализатора, несколько меньше — 32 %. Некорректное применение правил свертки вызывает ошибки в 12 % случаев. Вопросы омонимии, возникающие на этапе выделения интенциональных блоков, требуют проведения контекстного анализа.

Орфографические и синтаксические ошибки пользователей социальных сетей (их 8 %) являются естественным "шумом", на который, возможно, есть смысл настраивать содержимое вершин графа, например, целесообразно указывать варианты лексем и грамм, не предусмотренные корректным синтаксисом сообщения. Эти ошибки носят регулярный характер и не создают трудностей, связанных с омонимией.

Следует отметить, что указанные в табл. 2 оценки качества гарантированы для выявления объекта интенции. При извлечении некоторых свойств возможно снижение полноты и точности на несколько процентов. Успешное распознавание интенций, не относящихся к потребительским, свидетельствует о высокой степени универсальности построенных правил для распознавания явно выраженных интенций.

Заключение

Разработан и реализован оригинальный подход к поиску и извлечению интенций (намерений) потребителей из сообщений пользователей социальных сетей, отличающийся учетом семантической структуры факта интенции. Полученные на этапе обучения правила представлены в виде ориентированных графов, аккумулирующих информацию о возможных способах выражения интенций в русском языке. Содержащаяся в вершинах разноуровневая информация об интенциональных блоках сообщения характеризует объект интенции и его разнотипные свойства.

Результаты тестирования показывают, что разработанный метод дает возможность установить, содержится ли интенция в тексте, и, если она содержится, то извлечь объект интенции и его свойства с высокими показателями полноты и точности, часто вне зависимости от того, имеет интенция покупательскую специфику или нет. Извлекаемая на этапе распознавания информация может быть использована для уточнения классификации сообщений потребителей товаров и услуг.

Программа распознавания коммерческих интенций в сообщениях пользователей социальных сетей зарегистрирована в фонде электронных ресурсов ОФЭРНиО (04.07.2018 г., № 23691).

Работа выполнена при поддержке Российской академии наук (программа базовых исследований, проект № 0314-2016-0015).

Список литературы

1. **Пивоварова Л. М.** Фактографический анализ текста в системе поддержки принятия решений // Вестник СПбГУ. Сер. 9. 2010. № 4. С. 190—197.

2. **Cunningham H., Maynard D., Tablan V.** JAPE: a Java Annotation Patterns Engine (Second Edition). Technical report CS-00-10, University of Sheffield, Department of Computer Science, 2000. URL: http://gate.ac.uk/gate/doc/papers.html

3. **Томита-**парсер. URL: https://tech.yandex.ru/tomita

4. **Большакова Е., Баева Н., Бордаченкова Е.** и др. Лексико-синтаксические шаблоны в задачах автоматической обработки текстов // Компьютерная лингвистика и интеллектуальные технологии: Труды Международной конференции Диалог'2007. Москва: Изд-во РГГУ, 2007. Т. 2. С. 70—75.

5. **Андреев А. М., Березкин Д. В., Симаков К. В.** Модель извлечения фактов из естественно-языковых текстов и метод ее обучения // Восьмая Всероссийская научная конференция RCDL'2006. URL: http://rcdl.ru/doc/2006/paper_25_v3.pdf

6. **Navigli R., Velardi P.** LearningWord-Class Lattices for Definition and Hypernym Extraction. URL: http://www.aclweb.org/anthology/P10-1134

7. **Sarawagi S**. Information extraction // Foundations and Trends in Databases. 2008. URL: https: //www.cis.uni-muenchen.de/~fraser/information_extraction_2017_ lecture/sarawagi.pdf

8. **Tang J., Hong M., Zhang D., Liang B., Li J.** Information Extraction: Methodologies and Applications // Emerging Technologies of Text Mining. 2007. P. 1—33. URL: http://keg.cs.tsinghua.edu.cn/jietang/publications/Tang-et-al-Information_Extraction.pdf

9. **Brin S.** Extracting patterns and relations from the World Wide Web // Proceedings of the 1998 Int. Workshop on the Web and Databases. New York, USA, 1998. P. 172—183.

10. **Лукашевич Н. В.** Итерационное извлечение шаблонов описания событий по новостным кластерам // Труды конференции RCDL—2012 "Электронные библиотеки: перспективные методы и технологии, электронные коллекции" (Переславль-Залесский), 2012. С. 353—359.

11. **Leadsift.** URL: http://leadsift.com/index.html

12. **Qualia.** URL: http:// qualia-media.com/

13. **SoCeDo.** URL: http://socedo.com/

14. **Leadscanner.** URL: https://leadscanner.ru

15. **Leaderator.** URL: https://leaderator.pro/

16. **It's great.** URL: http://smm.tools/social/38-its-great.html

17. **Pymorphy2.** URL: https://pymorphy2.readthedocs.io/en/latest/

18. **Кормалев Д. А.** Приложения методов машинного обучения в задачах анализа текста // Программные системы: теория и приложения. Переславль-Залесский, 2004. С. 35—48.

19. **PyGraphviz.** URL: https://pygraphviz.github.io/

20. **Агеев М., Кураленок И.** Официальные метрики РОМИП'2004 // РОМИП 2004. Пущино, 2004. URL: http://docplayer.ru/40704470-Oficialnye-metriki-romip-2004.html.

21. **Национальный** корпус русского языка. URL: http://www.ruscorpora.ru/search-main.html

Extraction of Explicit Consumer Intentions from Social Network Messages

I. S. Pimenov, pimenov.1330@yandex.ru, Novosibirsk State University, Novosibirsk, 630090, Russian Federation, **N. V. Salomatina**, salomatina_nv@live.ru, Sobolev institute of mathematics, Novosibirsk, 630090, Russian Federation

Corresponding author:

Salomatina Natal'ya V., PhD, Senior Researcher, Sobolev institute of mathematics, Novosibirsk, 630090, Russian Federation,
E-mail: salomatina_nv@live.ru

Received on August 20, 2018
Accepted on August 21, 2018

The problem of automatic extraction of facts from Russian texts was approached in this paper. The facts under examination were the intentions of social network users to purchase certain goods or use certain services. The utilized approach is based on the semantic tagging of user messages by an expert and the automatic construction of rules. A training set for expert annotation consisted of messages from the "VKontakte" social network, selected through the LeadScanner API. The invented system of semantic tags allowed distinguishing between various intentional blocks: objects, their different properties and emphatic constructions. Pre-processing of the training set included lemmatization and grammatical tagging with PyMorphy2. Then, on the material of the training set, a directed graph was constructed. Each node in this graph would correspond to an intentional block, including information about its expertly-assigned intentional tag, grammatical and/or lexical properties of its main word. The edges of the graph would connect the intentional blocks that could be found in adjacent positions across all the messages of the training set. Extraction of intention objects and their properties was achieved by test set analysis in accordance to the constructed graph. Test set included both messages containing non-consumer intentions or no intentions at all. The results of the testing stage show that the approach used allows ascertaining if a particular message expresses intention, and, if it does, extracting the intention object along with its relevant properties. The precision and recall of intention extraction was 81 % and 74 % respectively. The data extracted can be used for further refinement of message classification.

Keywords: *intention, intention marker, intentional block with annotation, directed graph, fact extraction*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project no. 0314-2016-0015.*

For citation:

Pimenov I. S., Salomatina N. V. Extraction of Explicit Consumer Intentions from Social Network Messages, *Programmnaya Ingeneria,* 2018, vol.9, no. 9, pp. 425—432.

DOI: 10.17587/prin.9.425-432

References

1. **Pivovarova L. M.** Faktograficheskij analiz teksta v sisteme podderzhki prinyatiya reshenij (Fact extraction analysis of the text in the decision support system), *Vestnik SPbGU*, 2010, Ser. 9, no. 4, pp. 190—197 (in Russian).
2. **Cunningham H., Maynard D., Tablan V.** JAPE: a Java Annotation Patterns Engine (Second Edition). Technical report CS-00-10, 2000, University of Sheffield, Department of Computer Science, available at: http://gate.ac.uk/gate/doc/papers.html
3. **Tomita-**parser, available at: https://tech.yandex.ru/tomita
4. **Bol'shakova E., Baeva N., Bordachenkova E.** et al. Leksiko-sintaksicheskie shablony v zadachah avtomaticheskoj obrabotki tekstov (Lexical-syntactic patterns in natural language processing tasks), *Komp'yuternaya lingvistika i intellektual'nye tekhnologii: Trudy Mezhdunarodnoj konferencii Dialog'2007*, 2007, Moscow, Izd-vo RGGU, 2007, vol. 2, pp. 70—75 (in Russian).
5. **Andreev A. M., Berezkin D. V., Simakov K. V.** Model' izvlecheniya faktov iz estestvenno-yazykovyh tekstov i metod ee obucheniya (The model of fact extraction from natural language texts and the learning method), *Vos'maya Vserossijskaya nauchnaya konferenciya RCDL'2006*, 2006, available at: http://rcdl.ru/doc/2006/paper_25_v3.pdf (in Russian).
6. **Navigli R., Velardi P.** LearningWord-Class Lattices for Definition and Hypernym Extraction, available at: http://www.aclweb.org/anthology/P10-1134
7. **Sarawagi S.** Information extraction, Foundations and Trends in Databases 2008, available at: https: //www.cis.uni-muenchen.de/~fraser/information_extraction_2017_lecture/sarawagi.pdf
8. **Tang J., Hong M., Zhang D., Liang B., Li J.** Information Extraction: Methodologies and Applications, *Emerging Technologies of Text Mining*, 2007, pp. 1—33, available at: http://keg.cs.tsinghua.edu.cn/jietang/publications/Tang-et-al-Information_Extraction.pdf
9. **Brin S.** Extracting patterns and relations from the World Wide Web, *Proceedings of the 1998 Int. Workshop on the Web and Databases*, 1998, New York, USA, pp. 172—183.
10. **Lukashevich N. V.** Iteracionnoe izvlechenie shablonov opisaniya sobytij po novostnym klasteram (Iterative Pattern Extraction Using News Clusters), *Trudy konferencii RCDL—2012 "EHlektronnye biblioteki: perspektivnye metody i tekhnologii, ehlektronnye kollekcii"* (Pereslavl'-Zalesskij), 2012, pp. 353—359 (in Russian).
11. **Leadsift**, available at: http://leadsift.com/index.html
12. **Qualia**, available at: http:// qualia-media.com/
13. **SoCeDo**, available at: http://socedo.com/
14. **Leadscanner,** available at: https://leadscanner.ru
15. **Leaderator**, available at: https://leaderator.pro/
16. **It's great**, available at: http://smm.tools/social/38-its-great.html
17. **Pymorphy2**, available at: https://pymorphy2.readthedocs.io/en/latest/
18. **Kormalev D. A.** Prilozheniya metodov mashinnogo obucheniya v zadachah analiza teksta (Applications of Machine Learning to Text Analysis), *Programmnye sistemy: teoriya i prilozheniya*, 2004, Pereslavl'-Zalesskij pp. 35—48 (in Russian).
19. **PyGraphviz**, available at: https://pygraphviz.github.io/
20. **Ageev M., Kuralenok I.** Oficial'nye metriki ROMIP'2004 (Oficial metriks of ROMIP—2004), 2004, Pushchino, available at: http://docplayer.ru/40704470-Oficialnye-metriki-romip-2004.html (in Russian).
21. **Nacional'nyj** korpus russkogo yazyka, available at: http://www.ruscorpora.ru/search-main.html (in Russian).

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

Сдано в набор 02.10.2018 г. Подписано в печать 21.11.2018 г. Формат 60×88 1/8. Заказ PI918
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**

МЕЖДУНАРОДНАЯ АККРЕДИТАЦИЯ ОБРАЗОВАТЕЛЬНОЙ ПРОГРАММЫ БАКАЛАВРИАТА «ПРОГРАММНАЯ ИНЖЕНЕРИЯ» ВЫСШЕЙ ШКОЛЫ ЭКОНОМИКИ

Бакалаврская программа «Программная инженерия», реализуемая департаментом программной инженерии на факультете компьютерных наук НИУ «Высшая школа экономики», получила международную аккредитацию в Совете по аккредитации в области техники и технологий (ABET).

Любая внешняя независимая оценка качества и уровня подготовки выпускников крайне важна. В последние годы все большую роль в формировании этой оценки приобретает мнение работодателей, которое находит отражение в профессионально-общественной аккредитации образовательных программ теми или иными профессиональными сообществами. Если государственная аккредитация в первую очередь оценивает соответствие учебного процесса утвержденным образовательным стандартам, то профессионально-общественная аккредитация сосредоточивает свое внимание на сформированности у выпускников компетенций, требуемых профессиональными стандартами.

Бакалаврская программа «Программная инженерия» в апреле 2017 года получила профессионально-общественную аккредитацию в Ассоциации предприятий компьютерных и информационных технологий (АПКИТ) на соответствие требованиям профессиональных стандартов по специальностям «Программист» и «Системный программист» сроком на 6 лет. Таким образом, она стала первой и пока единственной образовательной программой в России, аккредитованной АПКИТ на максимальный срок.

Следующим этапом стала подготовка к международной аккредитации программы. Была выбрана самая авторитетная в мире профессиональная организация, занимающаяся оценкой качества бакалаврских и магистерских программ в области инженерных наук и компьютерных технологий в университетах – ABET (Accreditation Board for Engineering and Technology), существующая с 1932 года и базирующаяся в США.

Аккредитация ABET не только подтверждает высокое качество программ, но и дает их выпускникам серьезные конкурентные преимущества на международном рынке труда. Работодатели всего мира прекрасно знают, что выпускники образовательной программы, имеющей аккредитацию ABET, не нуждаются в подтверждении их квалификации.

Бакалаврская программа «Программная инженерия» факультета компьютерных наук НИУ «Высшая школа экономики» получила аккредитацию ABET в августе 2018 года после трехлетнего процесса рассмотрения заявки, включавшего, в том числе, визит экспертов.

В настоящее время аккредитацию в ABET имеют 35 бакалаврских программ по программной инженерии, большинство из которых реализуется в США.

Бакалаврская программа «Программная инженерия» департамента программной инженерии факультета компьютерных наук НИУ «Высшая школа экономики» – единственная образовательная программа в России, имеющая столь авторитетную международную аккредитацию.

2019

Январь

Пн	Вт	Ср	Чт	Пт	Сб	Вс
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Февраль

Пн	Вт	Ср	Чт	Пт	Сб	Вс
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

Март

Пн	Вт	Ср	Чт	Пт	Сб	Вс
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

Апрель

Пн	Вт	Ср	Чт	Пт	Сб	Вс
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

Май

Пн	Вт	Ср	Чт	Пт	Сб	Вс
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

Июнь

Пн	Вт	Ср	Чт	Пт	Сб	Вс
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

Июль

Пн	Вт	Ср	Чт	Пт	Сб	Вс
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

Август

Пн	Вт	Ср	Чт	Пт	Сб	Вс
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

Сентябрь

Пн	Вт	Ср	Чт	Пт	Сб	Вс
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Октябрь

Пн	Вт	Ср	Чт	Пт	Сб	Вс
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Ноябрь

Пн	Вт	Ср	Чт	Пт	Сб	Вс
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

Декабрь

Пн	Вт	Ср	Чт	Пт	Сб	Вс
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

ISSN 2220-3397. Программная инженерия. 2018. Том 9. № 9. 385–432. Подписной индекс по Объединенному каталогу «Пресса России» – 39795