ISSN 2220-3397

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

ПрИн
1
2020
Том 11

Рисунки к статье **М. Б. Кузьминского, А. М. Чернецова**

«СОВРЕМЕННЫЕ СРЕДСТВА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ В МОДЕЛИ РАСПРЕДЕЛЕННОЙ ПАМЯТИ»

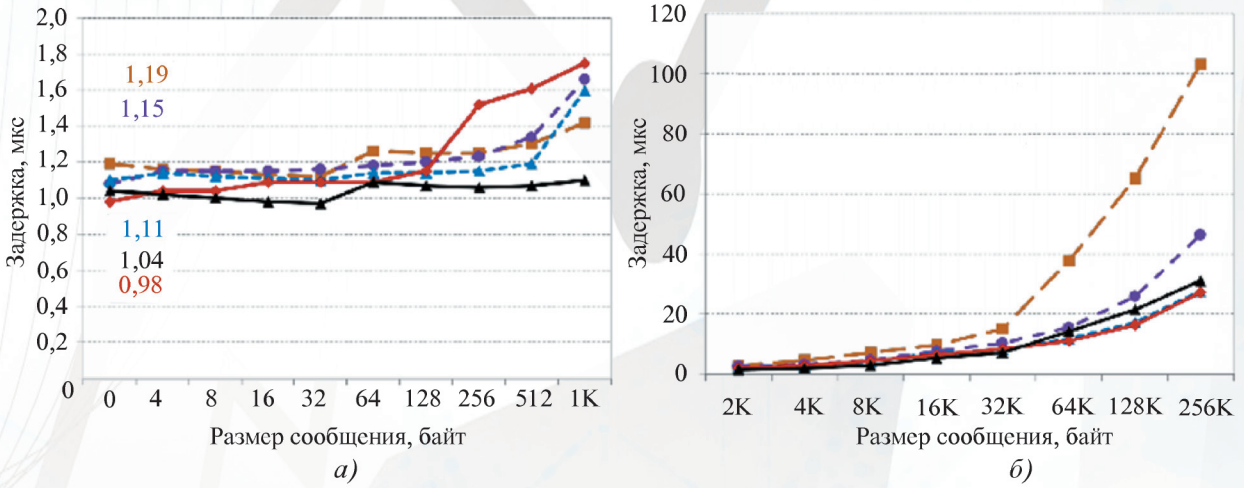

Рис. 1. Задержки односторонних MPI-операций MVAPICH2 поверх Omni-Path и Infiniband EDR:

а – короткие сообщения; *б* – длинные сообщения;

– TrueScale-QDR; – ConnectX-3-FDR; – ConnectIB-DualFDR;

– ConnectX-5-EDR; – Omni-Path

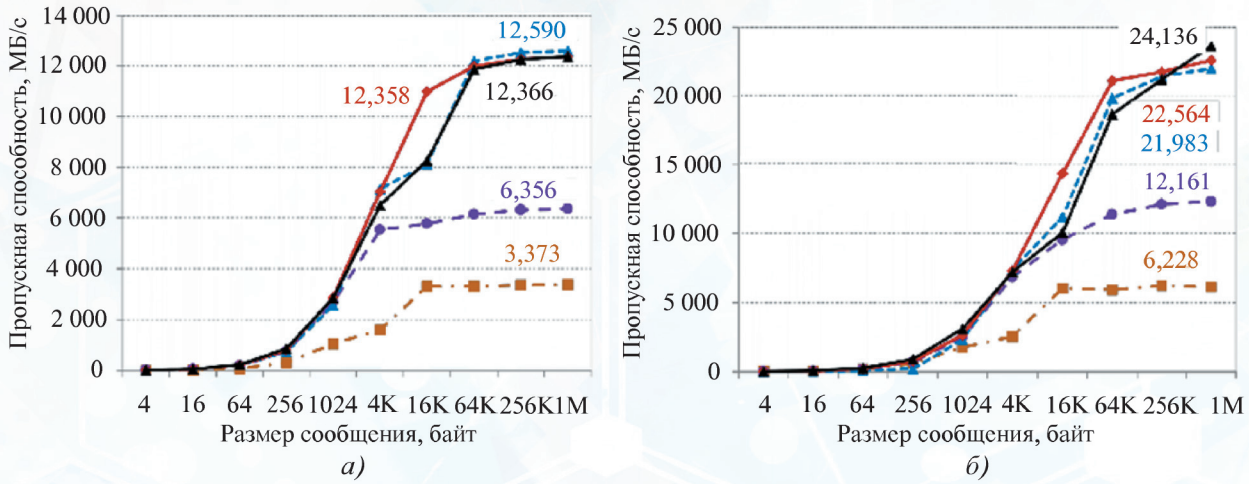

Рис. 2. Пропускная способность односторонних MPI-операций MVAPICH2 поверх Omni-Path и Infiniband EDR:

а – однонаправленный случай (короткие сообщения);

б – двунаправленный случай (длинные сообщения);

– TrueScale-QDR; – ConnectX-3-FDR; – ConnectIB-DualFDR;

– ConnectX-5-EDR; – Omni-Path

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

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

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

СОДЕРЖАНИЕ

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

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индекс по Объединенному каталогу "Пресса России" — **22765**) 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. 11
N 1
2020

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

CONTENTS

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

УДК 004.052.42 DOI: 10.17587/prin.11.3-13

И. А. Федотов, аспирант, ivan.fedotov@phystech.edu,
А. С. Хританков, канд. физ.-мат. наук, доц., anton.khritankov@acm.org,
Московский физико-технический институт

Систематический обзор исследований в области автоматической верификации кода смарт-контрактов

Смарт-контракты — это обладающий спецификой программный код, исполняемый в системах распределенного реестра (блокчейн-платформах). Смарт-контракты все чаще применяют в финансовой, юридической и других сферах, требующих высокой надежности и безопасности. Автоматическая верификация и анализ кода — важнейшее направление в обеспечении надежности и безопасности программ. В данной работе представлен краткий систематический обзор исследований в области верификации смарт-контрактов в период 2015—2019 гг.

Ключевые слова: *смарт-контракт, блокчейн, верификация, систематический обзор, Ethereum, BitCoin, проверка моделей*

Введение

Проблема устранения ошибок при разработке и использовании программ является одной из ключевых в области компьютерных наук. Ошибка может появиться в явном виде при работе программы в режиме промышленной эксплуатации. Впоследствии эта ошибка может быть использована злоумышленниками. Особенно опасна такая ситуация при разработке и сопровождении финансовых приложений, когда от корректности работы кода зависит трансфер средств.

Технология распределенных реестров блокчейн (*blockchain*) и децентрализованные приложения (*dApps*) несмотря на новизну стремительно набирают популярность при создании финансовых приложений, средств электронного голосования, в энергетике, медицине, интернете вещей [1]. Программный код смарт-контракта, исполняемый на платформе блокчейн, реализует алгоритм, в соответствии с которым финансовые средства или другие важные данные передаются от одного пользователя другому. При этом общепринятых средств для проверки и отладки программного кода пока еще не существует. Вместе с тем растет число создаваемых смарт-контрактов [2], соответственно проблема выявления и устранения ошибок становится все более актуальной.

Существует и другая особенность, которая делает ошибки в смарт-контрактах наиболее критичными. Зачастую смарт-контракты после развертывания на распределенном реестре остаются неизменными. Это обстоятельство не позволяет исправить ошибки во время работы и, следовательно, основное значение имеет выявление ошибок на уровне разработки. Уязвимости в смарт-контрактах уже приводили к атакам, которые подрывают доверие к технологии распределенного реестра. Например, печально известная атака DAO [3] привела к потере почти 50 млн долл. США. Некоторые другие атаки привели к существенной потере стоимости виртуальной валюты Ether (эфир), включая ошибку Parity Wallet, результатом которой было то, что эфир на 169 млн долл. оказался навсегда заблокирован [4]. Эти и многие другие примеры являются показателем актуальности исследования верификации кода смарт-контрактов.

Данное исследование представляет систематический обзор в области автоматической верификации кода смарт-контрактов. Его целью является исследование текущих тенденций и методов верификации программного кода смарт-контрактов, их классификация и анализ. Обзор определяет также существующие инструментальные средства (*tools*, программные средства), позволяющие проводить верификацию программного кода смарт-контрактов.

Настоящий обзор отличен от других [5] тем, что в нем не только исследуется вопрос безопасности системы распределенного реестра, но, в частности, рассматриваются методы верификации смарт-контрактов и соответствующие им инструментальные (программные) средства.

Обзор структурирован следующим образом. В разд. 1 приведено описание технологии распределенного реестра и принцип работы смарт-контрактов. В разд. 2 приведена постановка задачи исследования и определены методы исследования. Затем представлена характеристика современных методов верификации программного обеспечения. В разд. 3 представлен систематический обзор литературы, существующих на данный момент технических, а также инструментальных средств для верификации смарт-контрактов. Исследование завершается анализом и систематизацией рассмотренных статей и выводами о направлениях дальнейшего исследования.

1. Технология распределенного реестра и смарт-контрактов

Технология распределенного реестра. Блокчейн — это выстроенная цепочка блоков, при этом каждый блок содержит информацию в том числе и о пре-

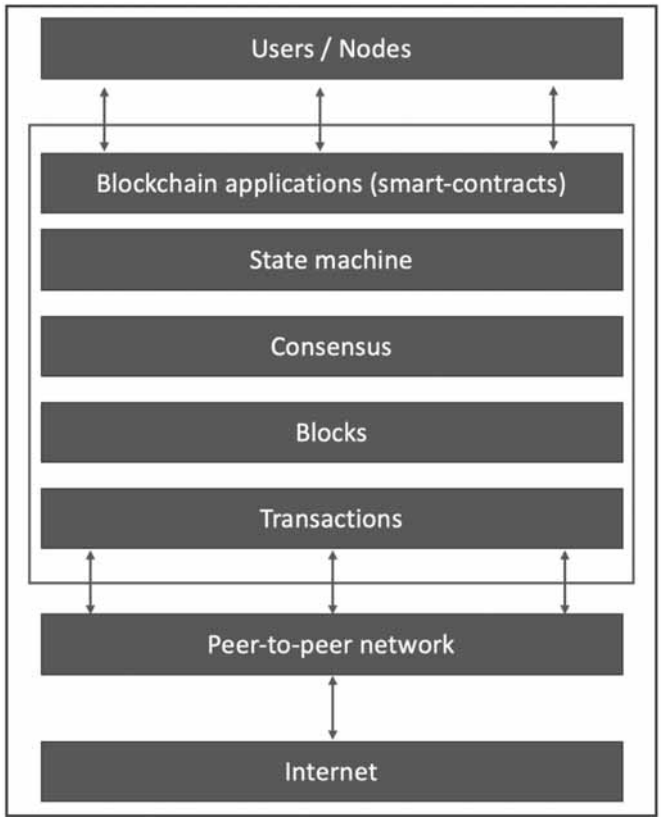

Рис. 1. Высокоуровневая схема распределенного реестра

дыдущем блоке [6]. Ядром блокчейна является распределенное одноранговое (peer-to-peer) хранилище, доступное только для записи значений, но не для их изменений. Данные в хранилище обновляются только посредством консенсуса участников сети, имеющей выход в сеть Интернет, как это представлено на рис. 1. Основой технологии является множество протоколов и криптографических методов, применяемых к сети узлов, которые взаимодействуют в целях обеспечения безопасной записи и хранения данных в распределенном хранилище.

Все изменения в сети, совершаемые посредством одной транзакции, помещаются в соответствующий блок, который можно рассматривать как страницу в памяти. Блок может быть представлен как последовательность транзакций, собранных вместе в порядке логической организации. Структура блока зависит от архитектуры блокчейна, но в целом заключает в себе набор атрибутов, необходимых для его корректного функционирования, таких как заголовок блока, указатель на предыдущий блок, временные метки, счетчик транзакций и сами транзакции. Каждый блок данных делегируется подтверждающим узлам (майнерам). Подтверждение происходит с помощью технологии, принятой в платформе, — это может быть решение криптографической задачи, либо майнеры получают априорное право подтверждать блок по определенным критериям. Такими критериями могут выступать временной штамп, обозначающий, как долго майнер участвует в консенсусе, либо количество виртуальной валюты на аккаунте майнера. Также майнеры могут назначаться выборным путем [7].

Смарт-контракты. Смарт-контракты инкапсулируют программный код, который отображает контрактные соглашения в реальном мире на блокчейн-платформы [8]. Ключевой особенностью смарт-контрактов является то, что они автоматизируют выполнение соглашений между двумя и более участниками без привлечения третьей заверяющей стороны за счет невозможности изменения порядка исполнения контракта одной из сторон после его публикации. Программный код смарт-контрактов размещается на всех подтверждающих узлах распределенного реестра.

Остановимся подробнее на рассмотрении контрактных языков [9]. Принцип их работы проиллюстрирован на рис. 2. Компиляция (A) подразумевает промежуточный уровень, который позволяет выполнить оптимизацию программы и верифика-

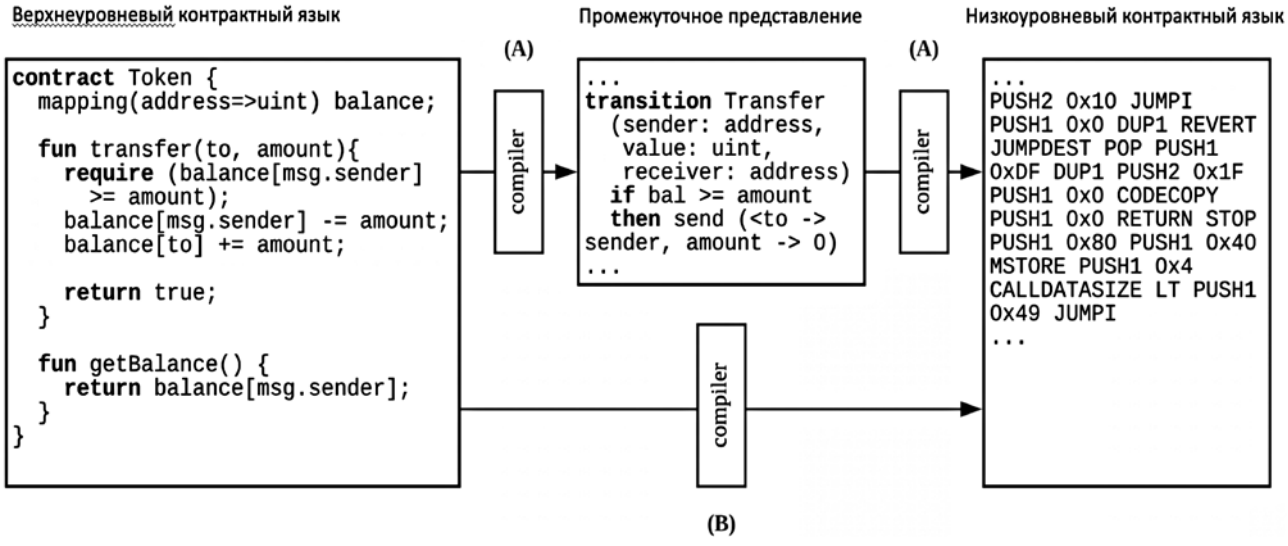

Рис. 2. Различные уровни контрактных языков

цию. Массовое распространение смарт-контракты получили недавно, и на ранних этапах компиляция происходила без промежуточного представления (B). Язык Bitcoin Script требует написания низкоуровневого кода. Примерами верхнеуровневых языков могут служить Solidity [10] и Vyper [11].

Наибольшее распространение смарт-контракты получили на платформах типа Ethereum [6]. Для их запуска используется виртуальная машина Ethereum (*Ethereum Virtual Machine*, EVM) [12], что делает возможным написание смарт-контрактов на разных языках, для которых реализован транслятор в исполняемый байт-код для виртуальной машины. Самым распространенным языком с поддержкой трансляции в байт-код EVM является Solidity.

Смарт-контракты на платформе Ethereum должны удовлетворять определенным стандартам. Стандарт ERC20 [13] определяет набор правил для токенов Ethereum, что позволяет формализовать поведение токенов в экосистеме Ethereum. Практически это означает, что смарт-контракт должен реализовывать ряд функций, чтобы удовлетворять стандарту.

2. Методология исследования

Исследование проведено согласно методу систематического обзора [14].

Стоит добавить, что на ошибки исполнения программного кода смарт-контрактов также влияют сторонние факторы, такие как внешние библиотеки, работа операционной системы, а также виртуальной машины. Предполагается, что все это работает и ошибки вышеперечисленных факторов не учитываются.

Постановка вопросов исследования. Цель исследования — получить ответы на следующие вопросы.

Q1. Какие методологии верификации кода используются?

Q2. Какие результаты известны по автоматической верификации смарт-контрактов?

Q3. Какие есть ограничения в автоматической верификации программного кода?

Q4. Какие можно выделить будущие направления в автоматической верификации смарт-контрактов?

Протокол поиска. Поиск проводился в августе-сентябре 2019 г. Для поиска статей использовались следующие поисковые системы: IEEE Xplore, Scopus, Google Scholar, Research Gate, ScienceDirect. Для проведения поиска ключевые слова были объединены в следующие поисковые строки:

- (blockchain OR ethereum OR bitcoin) AND smart-contract AND (verification OR model-checking OR validation) OR systematic review;
- smart contract AND verification tools.

Для статей, прошедших поисковый фильтр по перечисленным выше запросам, а также упомянутых в них исследований, был проведен ручной поиск, чтобы выявить статьи, которые косвенно относятся к тематике, но могут иметь существенное влияние на область (неопубликованные статьи, дипломные работы, коммерческие документы, учебные материалы). В исследовании, направленном на построение качественного систематического обзора [14], также подчеркнута важность используемых методик.

После выполнения поисковых запросов был проведен анализ статей по названию, аннотации, введению, заключению, числу цитирований и году публикаций. Далее приведены критерии включения и критерии исключения, которые были использованы при анализе и выявлении наиболее подходящих статей.

Критерии включения.

- Исследование проведено на русском или английском языке.
- Исследование относится к области компьютерных наук.
- Исследование проведено в последние четыре года. Исключением могут быть фундаментальные учебные материалы.
- Исследования, представленные в виде статьи, опубликованы в рецензируемых журналах.

Критерии исключения.

- Основной язык не является русским или английским.
- Исследование нетехнического характера.

В процессе поиска было изучено более 100 статей, коммерческих документов, документации к инструментальным средствам и учебных пособий. В результате в обзор попало 52 исследования.

3. Краткая характеристика современных методов верификации программ

Обратимся сначала к вопросу Q1. В основе **формальной верификации (*formal verification*)** лежат формальные методы, используемые для доказательства соответствия модели системы установленным спецификациям требований. Необходимо понимать, что фактически реализованная программа и требования к ней будут отличаться в той степени, в которой отличаются соответствующие им модели. Методы формальной верификации могут различаться как по модели программ (конечные автоматы, сети Петри, размеченные системы переходов), так и по модели требований (программные контракты, продукционные правила, темпоральные утверждения).

Метод проверки эквивалентности предполагает совпадение типа модели программы с типом модели спецификаций. Для сравнения моделей используется отношение эквивалентности, определенное для данного типа модели.

Другой метод формальной верификации называется **проверкой моделей (*model checking*)** [15]. В данном методе спецификации представляются логическими формулами, а программный код — структурами, интерпретирующими эти формулы. Под формальным соответствием подразумевается, что структура, представляющая код, является моделью соответствующей формулы. Для представления спецификаций может использоваться логика LTL (*Linear-time Temporal Logic*, LTL) [16], которая представляет программу как последовательность действий во времени. Такая логика позволяет представлять формулы в контексте будущего развития.

Например, условие будет верным, если выполнится ряд предшествующих условий. С помощью такого подхода можно рассматривать действие программы или алгоритма во времени.

В методе **дедуктивного анализа** [17] модель программы — это запись на языке с формализованной семантикой (может быть представлена в виде исходного кода, псевдокода или схемы программы), а модель требований — совокупность программных контрактов для модулей программы. Здесь под программным контрактом понимается совокупность логических формул: пред- и постусловия для функций, процедур, методов. Предусловия обеспечивают выполнение всех условий, при которых программа будет запущена правильно. Постусловия указывают корректный результат выполнения самой подпрограммы, т. е. что все выходные переменные и состояние программы после исполнения будут удовлетворять этим условиям. Метод дедуктивного анализа активно применяется для основных языков программирования, таких как C [18], Java [19] или C# [20]. Верификация происходит на уровне исходного кода программы. Инструментальное средство KeY [21] было разработано для дедуктивной верификации объектно-ориентированных языков. В стандартной версии KeY принимает на вход код с расширением .java и спецификации в виде JML (*Java Modeling Language*).

Гибридная верификация сочетает в себе формальные методы верификации и тестирование. Рассмотрим случай, в котором язык не имеет строгой семантики, и программа на нем не является одновременно своей моделью для верификации. Тогда одна лишь математическая проверка моделей не может полностью обеспечить отсутствие ошибок в программном коде, так как модели отличаются от реальных программ и требований. Вместе с тем тестирование кода происходит уже после написания программного кода, а значит при появлении ошибок их нужно будет исправлять в уже разработанной программе. Для наиболее эффективного исключения ошибок формальные методы используются вместе с тестированием, и такой подход называется **методом тестирования на основе моделей**.

Выявление ошибок можно проводить и без доступа к исходному коду — с помощью **журнала протоколирования.** Вся информация о работе программы записывается в журнал, в том числе и путь к ошибке с возможной причиной. При аварийном завершении программы разработчик может отследить ошибку в журнале протоколирования и выявить ее причину [22]. Отладка происходит путем повторения исполнения программы в тестовой среде.

4. Применимость методов верификации к смарт-контрактам

Обратимся к вопросу Q2 и укажем, какие из подходов, относящихся к вопросу Q1, применимы в области смарт-контрактов.

Статический анализ кода. Рассмотрим инструментальные средства статического анализа кода смарт-контрактов [23]. Средство SmartCheck [24] выполняет разбор текста исходного кода программы и представляет абстрактное синтаксическое дерево (AST) в виде документа XML с сохранением трассировки по номерам строк с исходным кодом. Строится дерево состояний смарт-контракта и делается разбор схемы запросами XPath. SmartCheck включает набор правил для проверки соответствия полученного дерева рекомендациям и ограничениям языка Solidity. Исследование [25] показало, что SmartCheck среди других инструментальных средств статической верификации является наиболее эффективным по числу найденных ошибок, но также он может выдавать ложные предупреждения.

Статический анализ кода также интегрирован в некоторые среды разработки, например Intellij Idea Solidity plugin. В приведенных инструментальных средствах учитываются синтаксические ошибки в коде смарт-контрактов.

Формальная верификация. Рассмотрим методы формальной верификации смарт-контрактов. Спецификации могут передаваться на специально созданном для реализации таких методов языке.

Инструментарий VeriSol [26] для платформы Azure Blockchain комбинирует подходы на основе семантического анализа кода и тестирования: программный код транслируется в формальный язык Boogie [27], происходит семантический анализ кода, который определяет соответствие реализации кода спецификации интерфейса по пред- и постусловиям. Примером выявляемых ошибок может служить ошибка в реализации интерфейса, при которой снятие средств со счета может произойти несколько раз до обновления баланса. Результатом подобной ошибки была кража 80 млн долл. злоумышленником [28]. После анализа в код Solidity внедряются модификаторы с проверками пред- и постусловий функции.

В другой методике [29] выполняется трансляция программного кода смарт-контракта в функциональный язык F*, предназначенный для верификации программ, после чего происходит верификация функциональной корректности спецификаций. В дополнение к этому байт-код декомпилируется и выполняются низкоуровневые проверки, такие как проверка достаточного количества эфира при вызовах.

Инструментальное средство SOLAR [30] проверяет соответствие смарт-контрактов стандарту с помощью символьного исполнения (*symbolic execution*) и пересчета заданных на python-подобном языке ограничений и инвариантов совместно с исполнением кода смарт-контракта (*concolic execution*) [31, 32]. Написав один программный контракт (стандарт), можно сделать проверку всех смарт-контрактов, которые должны ему удовлетворять. Для безопасной проверки корректности поведения смарт-контракта средство SOLAR моделирует поведение смарт-контракта и запускает его в тестовой среде с пробными транзакциями. Такой инструментарий примечателен тем, что при поиске логических ошибок он выдает существенно меньше ошибок первого рода (*false positive*), чем другие аналогичного назначения инструментальные средства.

Упомянем также специальный язык и программный каркас (*framework*) [33] описания одновремен-

ного обращения нескольких участников к смарт-контракту. Работа каркаса основана на теории игр. Каждый участник, который получает доступ к смарт-контракту, рассматривается как игрок, причем число игроков и вызовов функций в смарт-контракте ограничено. Происходит трансляция контрактов в одновременные игры (*simultaneous game*), где передвижение средств в контракте сопоставляется с действием в игре. Анализ игры помогает определить возможные состояния для смарт-контракта и выявить логические ошибки, вследствие которых средства могут быть перечислены коррумпированному участнику контракта.

Интересно рассмотреть методы верификации, основанные на технике SMT (*Satisfiability Modulo Theories*) [34]. Решающие устройства SMT применяют для доказательства выполнимости формул в различных логиках, которые могут быть использованы для моделирования работы программ. Контракты Solidity могут быть транслированы в SMT-формулы [35]. Целью трансляции является верификация свойств программ путем выполнения запросов в решающее устройство SMT.

В исследовании [36] предложено использование инструментария SPIN [37] для верификации смарт-контрактов. Поведение контракта транслируется в язык Promela (*Process Meta Language*), после чего происходит верификация кода Promela в соответствии с представленными свойствами ранее разработанными инструментами. Верифицируемые системы представляются на языке Promela, поддерживающем моделирование асинхронных распределенных алгоритмов как недетерминированных автоматов. Получив модель и определение проверяемых свойств, SPIN генерирует программу на языке Си, которая решает конкретную задачу. Свойства выражаются в виде формул линейной временной логики (*Linear temporal logic*, LTL).

Инструментарий NuSMV [38] использует похожий подход для выражения свойств, которые необходимо проверить на модели: вычислительное дерево логики (*Computation tree logic*, CTL). В модели CTL логика представлена древовидной структурой, в которой каждая ветвь может быть будущим развитием событий при выполнении определенных условий. При этом, если выполнены требования, все возможные ветви исполнения программы позволяют избежать нежелательных условий (например, деления на ноль или других ошибок). Модель написана на специальном языке для NuSMV, а свойства для проверки формализованы в CTL.

Примеры использования логических формул, таких как LTL и CTL в процессе верификации смарт-контрактов можно найти в соответствующих работах [37, 38].

Модель рассматривается на трех уровнях: поведение всего реестра на платформе Ethereum, поведение смарт-контракта и его последующая трансляция в язык NuSMV, поведение исполняемого каркаса в соответствии с переданными свойствами. Если условие не выполняется, генерируется контрпример, на основе которого можно проанализировать природу выявленного дефекта. Метод был разработан в ходе исследования применения распределенного реестра на энергетическом рынке Франции.

Дедуктивный анализ. Для смарт-контрактов представлено расширение SolidiKeY [39], в котором инструмент KeY [21] адаптирован для верификации кода смарт-контрактов. Используется метод дедуктивной верификации, который выполняется автоматически с помощью инструментальных средств, реализующих доказательства теорем. Происходит проверка на то, что пред- и постусловия соответствуют поведению смарт-контракта. Примером предусловия для смарт-контракта может быть наличие определенной суммы для транзакции, которую инициирует смарт-контракт. Соответствующим постусловием будет являться начисление суммы на аккаунт пользователя. На выходе инструментальное средство выдает информацию о том, удовлетворяет ли программный код спецификациям, и если не удовлетворяет, то по каким пунктам.

В работе [39] предлагается язык спецификаций для Solidity, учитывающий его особенности, а также возможность верифицировать программный код Solidity в соответствии с переданными спецификациями. Результатом работы SolidiKeY является информация о соответствии программного кода смарт-контракта спецификациям.

Проверка доказательств теорем. В работах [40, 41] представлен подход верификации с помощью инструментальных средств проверки доказательств Isabelle/HOL. Байт-код представляется в виде линейной последовательности блоков программы. Конструкция байт-кода может содержать скачки, которые ведут поток выполнения к другой части программы, что затрудняет представление в виде последовательного выполнения кода. В связи с этим декомпиляция происходит с помощью метода контрольных графов (*Control Flow Graphs*, CFG), который позволяет представить программу в виде базовых блоков и соединить их ребрами, соответствующими скачкам. Таким образом, в соответствии с теоремой о структурном программировании [42] программный код будет представлен в виде последовательности подпрограмм. Спецификации передаются в виде псевдокода, после составленная последовательность проверяется в соответствии со спецификациями в средстве Isabelle/HOL.

В инструментальном средстве FEther [43], использующем автоматическое средство доказательства теорем Coq [44], применяется кеширование уже проверенного кода и при последующем попадании на него повторная проверка не требуется.

Каркас VeriSolid [45] не требует кода смарт-контракта. Нужна только его модель, а также список верифицируемых свойств, что позволяет верифицировать поведение контракта на высоком уровне абстракции. Спецификации могут быть записаны на естественном языке, например, с помощью шаблонов. Принцип работы такого инструментального средства следующий: сначала строится поведенческая модель контракта (*Behaviour-Interaction-Priority*, BIP). В ней рассматриваются только возможные со-

стояния смарт-контракта — это позволяет уменьшить нагрузку при последующей верификации. Из переданных спецификаций строится вычислительное дерево логики CTL. После формальной верификации модели CTL с помощью уже имеющихся инструментальных средств [46, 47] может быть сгенерирован код смарт-контракта на языке Solidity. Такой процесс соответствует разработке по принципу "корректность в соответствии с дизайном" (*correct by design*) [48]. При обнаружении ошибок вместо генерации кода пользователю передается вывод спецификации, в котором указаны ошибки проверки.

Интересным представляется гибридный подход, в котором формальные методы сочетаются с другими методами верификации, такими как тестирование или статический анализ, что обеспечивает большее покрытие кода проверками. Для формальной проверки в инструментальном средстве верификации FSPVM-E [49], в котором реализован гибридный подход к верификации программного кода, используется средство автоматического доказательства теорем Coq и рассмотренное ранее средство FEther.

Динамическая верификация и анализ журналов работы. Методы динамической верификации также применимы к смарт-контрактам. Метод проверки корректности работы смарт-контрактов постфактум, т. е. уже после записи в реестр, в одной из работ [50] был обозначен как *query blockchain*. После развертывания контракта в сети делается запрос на корректность записанных в реестр данных, после чего происходит автоматическая проверка соответствия смарт-контракта и записанных данных.

Другой метод, реализованный в инструментальном средстве ContractFuzzer [51], подразумевает генерацию данных для смарт-контракта и анализ журнала протоколирования в тестовой среде. Таким образом, при промышленном использовании смарт-контракта ряд ошибок уже будет учтен.

Наиболее часто встречающимся при составлении обзора инструментальным средством динамической верификации смарт-контрактов было ContractLarva [52]. На вход этому средству передаются спецификации, после чего происходит мониторинг событий смарт-контракта. Код контракта разбирается, и в него внедряются проверки на события из спецификации. В базовой версии этого средства предусмотрена возможность взимания со всех участников консенсуса априорной платы, которая будет внесена в случае нарушения смарт-контракта со стороны определенного пользователя. Другим вариантом является полная остановка смарт-контракта без вычета средств с пользователя. Также имеется ряд наработок по совершенствованию инструментальных механизмов ContractLarva, которые относятся к вопросам устранения последствия нарушений смарт-контракта путем генерирования и развертывания вторичного смарт-контракта [53].

Представляет интерес подход, направленный на автоматическое восстановление спецификаций на основе анализа истории исполнения контракта в распределенном реестре [54]. Спецификации подходят как для самого контракта, так и для межконтрактного взаимодействия. При анализе работы программного кода строится граф зависимостей контрактов и состояний. Полученный результат может быть использован как для выявления ошибок, так и для построения более точной верификации.

Работа [55] посвящена моделированию межконтрактного взаимодействия с помощью анализа траекторий перевода эфира и предсказания связей смарт-контрактов в некотором тензорном пространстве. В этом пространстве одно измерение связано с аккаунтом отправителя, второе — с аккаунтом получателя средств, а третье — со временем. В пересечении в конкретном временном слоте между аккаунтом отправителя и аккаунтом получателя указывается количество переданного эфира. Для полученного тензора выполняется декомпозиция CANDECOMP/PARAFAC (CP) [56]. По полученным данным в приведенном трехмерном пространстве строится картина распределения эфира во времени между участниками, после чего с помощью временных рядов моделируется поведение в будущем. На основе этого анализа можно выявить нарушения в работе системы, а в случае ошибки может быть сгенерирован контрпример.

Аудит и экспертиза. Рассмотрим еще один метод в верификации смарт-контрактов — аудит с применением автоматического анализа кода. Инструмент аудита смарт-контрактов S-gram [57] для платформы Ethereum сочетает методы статического анализа кода и обнаружения аномалий. На этапе обучения S-gram по заданной коллекции (корпусу) смарт-контрактов для кода после разделения на токены, дополненные информацией от статического анализатора кода, создается статистическая языковая модель (*statistical language model*) [58]. Такая языковая модель представляет собой конечный автомат с частотными вероятностями переходов от одного токена к следующему. На основе языковой модели строятся взаимосвязи между критическими операциями и условиями их выполнения (например, трансфер средств и проверка адреса отправителя). На этапе анализа с помощью языковой модели рассчитывается, насколько типичен исследуемый смарт-контракт или его методы относительно корпуса, и если нетипичен — передается на дальнейшую проверку. Работа S-gram была проверена на более 1500 реальных контрактов, было найдено 95 % всех уязвимостей.

Техническая экспертиза может применяться для выявления соответствия смарт-контракта стандартам (например, ERC20 [59]).

Поддержка других языков. Одним из перспективных направлений исследования является верификация смарт-контрактов для менее распространенных языков [9, 60]. Как отмечено выше, Solidity — контрактно-ориентированный язык, в то время как для разработки распределенных приложений созданы процедурные языки (Scilla [61], Bamboo[62]), в которых функцию можно сделать атомарной. Более того, перечисленные языки позволяют разделить контракт на несколько составляющих, по сути независимых контрактов, которые, однако, имеют один и тот же адрес и в реестре являются одним целым. Такой подход позволяет представить один контракт в виде

переходов между независимыми состояниями. Исследования в верификации альтернативных языков уже ведутся [62], однако результатов, позволяющих использовать инструменты в промышленной эксплуатации, пока нет.

5. Анализ результатов исследования

В рамках данного раздела анализируются текущие, а также рассмотрены перспективные направления в верификации смарт-контрактов. Таким образом, будут даны ответы на исследовательские вопросы Q3, Q4.

Исходя из анализа содержания большого числа статей, можно сделать вывод, что наибольшее распространение получили методы формальной верификации. Это может быть связано с тем обстоятельством, что при верификации смарт-контрактов устранение вероятных ошибок важнее скорости проведения верификации. Основная причина в том, что формальные методы используют строгий математический аппарат, процент выявленных ошибок оказывается выше, нежели при тестировании, при динамическом или статическом анализе. Однако это не означает, что последними тремя методами можно пренебрегать. Как было отмечено выше, эти методы позволяют выявить ошибки, которые могут быть упущены на этапе применения формальных методов верификации.

Среди методов верификации смарт-контрактов следует выделить статический анализ кода. Различные реализации этого метода включают в себя анализ журнала протоколирования, представление и последующий анализ исходного кода в виде XML, а также в комбинации с другими методами, например, как это реализовано в инструменте VeriSol для платформы Azure Blockchain.

Отдельно стоит выделить виды ошибок смарт-контрактов и соответствующие инструментальные средства, которые помогают их выявить: логические (VeriSol, F*, SOLAR, программный каркас на основе теории игр, решающие устройства SMT, SPIN, NuSMV, ContractFuzzer, ContractLarva, извлечение спецификаций из журналов, тензорный анализ, Isabelle/HOL, FEther, VeriSolid, FSPVM-E, SolidiKeY, S-gram); синтаксические (VeriSol, ContractFuzzer, ContractLarva, SmartCheck, Intellij Idea Solidity plugin, Remix); ошибки компиляции (F*, ContractFuzzer, ContractLarva, извлечение спецификаций из журналов); среды исполнения/runtime (NuSMV, ContractFuzzer, ContractLarva, извлечение спецификаций из журналов). Интересным направлением исследования является выявление ошибок взаимодействия, которые могут возникнуть в связи с отсутствием соответствия программного обеспечения аппаратному интерфейсу или интерфейсу прикладного программирования.

Относительно методов экспертизы были выявлены инструментальные средства аудита [58]. Исследования по сквозному контролю и инспекции смарт-контрактов найдены не были. Это может быть связано с тем обстоятельством, что для сквозного контроля и инспекции необходима группа разработчиков, а полная автоматизация этих методов противоречила бы их принципам. Ввиду того что относительно других языков программирования рассматриваемые языки смарт-контрактов появились недавно, сложно найти группу разработчиков, которые были бы компетентны выполнять такого рода экспертизы.

Однако другие методы верификации имеют свои недостатки. Например, верификация по журналам протоколирования [51] определяет ошибки постфактум, когда они уже произошли и были записаны. Можно совместить данный метод с тестированием: перед развертыванием смарт-контракта в эксплуатационной среде провести несколько итераций запусков на тестовых стендах, проанализировать журнал протоколирования и устранить ошибки. Предложенный процесс также может быть автоматизирован.

При составлении обзора выяснилось, что формальная семантика применяется для языков низкого уровня (байт-код). Высокоуровневые языки программирования смарт-контрактов с полной формальной семантикой находятся в разработке [9]. Формальная семантика для языков всех уровней необходима для разработки верифицированных компиляторов. Верификация формальными методами языков высокого уровня также представляется перспективным направлением исследования.

Интересным является направление исследования по предотвращению перехода дефекта в ошибку. Имеется в виду интеграция инструментального средства верификации со средством контроля версий, с помощью которого можно было бы предотвратить миграцию ошибок на последующие релизы проекта.

Отмечая инструментальные средства разработки, следует заметить, что разрабатывать смарт-контракты можно не только на специально созданных для этого приложениях типа Remix [65]. Другие платформы, такие как Intellij Idea [64], Visual Studio, Eclipse также поддерживают расширения для редактирования исходного кода смарт-контрактов на языке Solidity.

Интересным для рассмотрения и анализа вопросом являются межконтрактные взаимодействия и рекурсивность в работе контрактов. При этом возникает вопрос о том, как очертить область, в которой рассматривать взаимодействие смарт-контрактов. Моделировать взаимодействие всех развернутых смарт-контрактов в распределенном реестре может быть сложно в зависимости от размеров среды моделирования. В данном случае можно будет построить вероятностную картину и в соответствии с ней очертить область смарт-контрактов, между которыми будет рассматриваться взаимодействие. Инструментальное средство CANDECOMP/PARAFAC позволяет выполнить моделирование межконтрактного взаимодействия.

При анализе исследуемых работ было замечено, что верификация исходного кода смарт-контрактов для языков высокоуровневого типа рассматривалась в основном для языка Solidity, либо для низкоуровневого байт-кода EVM. Необходимо провести исследования и для других языков смарт-контрактов, например, IELE, Scilla, Michelson.

Все перечисленные выше методы структурированы в таблице.

Классификация инструментальных средств верификации смарт-контрактов

Метод	Описание	Инструментальные средства
Формальная верификация	Используются формальные методы для доказательства соответствия модели системы установленным спецификациям требований	VeriSol [26], F* [29], SOLAR [30], программный каркас на основе теории игр [33], решающие устройства SMT [34, 35], SPIN [36], NuSMV [38]
Динамическая верификация	Проверка поведения программы в момент ее выполнения. Полезна при отсутствии доступа к коду программы	ContractFuzzer [51], ContractLarva [53], извлечение спецификаций из журналов [54], тензорный анализ [55]
Статический анализ кода	В отличие от динамического анализа происходит без запуска программ. Статический анализ обладает высокой степенью надежности и применяется в жизненно важных областях [63]	SmartCheck [23—25], Intellij Idea Solidity plugin [64], онлайн среда разработки Remix [65]
Проверка доказательств теорем	Использование инструментариев доказательств теорем для подтверждения соответствия спецификациям	Isabelle/HOL [40, 41], FEther [43], VeriSolid [45], FSPVM-E [49]
Дедуктивный анализ	Сопоставление предусловия с постусловием в контексте работы функций, процедур, методов для определения корректности их исполнения	SolidiKeY [39]
Аудит и анализ	Анализ результатов работы смарт-контракта на соответствие спецификациям	S-gram [57]

Заключение

В настоящем обзоре рассмотрены принципы работы технологии распределенных реестров, общие методы верификации программ и применимость этих методов к верификации смарт-контрактов. Выделены основные исследовательские области, которые включают в себя текущие методы по верификации смарт-контрактов, используемые инструментальные средства, их преимущества и недостатки. Для ответов на поставленные вопросы использовались работы, опубликованные в аккредитованных журналах за последние четыре года, а также учебные и методические пособия.

Представлены результаты анализа актуальных на момент написания статьи инструментальных средств, выявлены их сильные и слабые стороны, а также возможности усовершенствования некоторых из них. Рассмотрены принципы работы "контрактных" языков и их особенности в контексте верификации смарт-контрактов.

Выявлены перспективные направления разработки новых, а также усовершенствование уже существующих инструментальных средств верификации смарт-контрактов. Полученные результаты можно использовать при выборе направления будущих исследований в относительно новой и востребованной на практике проблемной области.

Список литературы

1. **Srivastava A., Bhattacharya P., Singh A., Mathur A.** A Systematic Review on Evolution of Blockchain Generations // ITEE Journal. 2018. Vol. 7, N. 6. P. 1—8.
2. **Perez D., Livshits B.** Smart Contract Vulnerabilities: Does Anyone Care? Preprint. 2019. arXiv:1902.06710.
3. **Gelvez M.** Explaining the DAO exploit for beginners in Solidity. 2016. URL: https://medium.com/@MyPaoG/explaining-the-dao-exploit-for-beginners-in-solidity-80ee84f0d470.
4. **Thomson I.** Parity: The bug that put $169m of Ethereum on ice? Yeah, it was on the todo list for months. 2017. URL: https://www.theregister.co.uk/2017/11/16/parity_flaw_not_fixed.
5. **Macrinici D., Cartofeanu C., Gao S.** Smart Contract Applications within Blockchain Technology: A Systematic Mapping Study // Telematics and Informatics. 2018. Vol. 35, N. 8. P. 2337—2354.
6. **Wood G.** Ethereum: A secure decentralised generalised transaction ledger. 2016. URL: http://paper.gavwood.com
7. **Xinxin F., Qi C.** Roll-DPoS: A Randomized Delegated Proof of Stake Scheme for Scalable Blockchain-Based Internet of Things Systems // 15th EAI International Conference, 2018. P. 482—483.
8. **Grishchenko I., Maffei M., Schneidewind C.** A Semantic Framework for the Security Analysis of Ethereum Smart Contract // POST 2018: Principles of Security and Trust, 2018. P. 246—248.
9. **Harz D., Knottenbelt W.** Towards Safer Smart Contracts: A Survey of Languages and Verification Methods. 2018. Preprint arXiv:1809.09805.
10. **Zakrzewski J.** Towards Verification of Ethereum Smart Contracts: A Formalization of Core of Solidity // Verified Software. Theories, Tools, and Experiments. 10th International Conference, VSTTE 2018, Oxford, UK, 2018. P. 229—247.
11. **Ethereum** Foundation. Vyper documentation. 2018. URL: https://vyper.readthedocs.io/en/v0.1.0-beta.13/
12. **Buterin V.** A next-generation smart contract and decentralized application platform. Ethereum White Paper. 2014. URL: https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf
13. **ERC:** Token standard. URL: https://github.com/ethereum/eips/issues/20
14. **Silva R., Neiva F.** Systematic Literature Review in Computer Science — A Practical Guide. 2016. P. 1—5.
15. **Siddiqui J. H., Rauf A., Ghafoor M. A**. Advances in Software Model Checking // Advances in Computers. 2018. Vol. 108. P. 59—89.
16. **Sistla A. P., Clarke E. M.** The complexity of propositional linear temporal logics // J. of the ACM (JACM) JACM Homepage archive. 1985. Vol. 32, N. 3. P. 733—749.
17. **Кулямин В. В.** Методы верификации программного обеспечения // Всероссийский конкурс обзорно-аналитических статей по приоритетному направлению "Информационно-телекоммуникационные системы", 2008. 117 с.
18. **Kosmatov N., Prevosto V., Signoles J.** A lesson on proof of programs with Frama-C // TAP 2013: International Conference on Tests and Proofs. Springer, 2013. P. 168—177.

19. **Deductive** Software Verification. — The KeY Book / Eds. W. Ahrendt, B. Beckert, R. Bubel et al. Springer, 2016. 702 p.

20. **Hähnle R., Huisman M.** Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools // Computing and Software Science. State of the Art and Perspectives. Springer, 2019. P. 345—373.

21. **Ahrendt W., Beckert B., Bubel R.** et al. The KeY platform for verification and analysis of Java programs // In STTE'14, V. 8471 of LNCS. Springer, 2014. P. 55—71.

22. **Reynders F.** Modern API Design with ASP.NET Core 2. Apress. 2018. 236 p.

23. **Grishchenko I., Maffei M., Schneidewind C.** Foundations and tools for the static analysis of ethereum smart contracts // Computer Aided Verification. 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK. 2018. Proceedings, 2018. Part I. P. 51—78.

24. **Tikhomirov S., Voskresenskaya E., Ivanitskiy I., Takhaviev R., Marchenko E., Alexandrov Y.** SmartCheck: Static Analysis of Ethereum Smart Contracts // ACM/IEEE 1st International Workshop on Emerging Trends in Software Engineering for Blockchain, 2018. P. 9—16.

25. **Parizi R. M., Dehghantanha A., Choo K.-K. R., Singh A.** Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains // CASCON'18, 2018. URL:https://arxiv.org/pdf/1809.02702.pdf

26. **Lahiri K., Chen S., Wang Y., Dillig I.** Formal Specification and Verification of SmartContracts for Azure Blockchain. 2018. URL: https://www.microsoft.com/en-us/research/publication/formal-specification-and-verification-of-smart-contracts-for-azure-blockchain/

27. **Barnet M. Chang B.-Y. E., DeLine R., Jacobs B., Leino K. R. M.** Boogie: A modular reusable verifier for object-oriented programs // Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005, Amsterdam, The Netherlands, 2005 / Eds. F. S. de Boer, M. M. Bonsangue, S. Graf, W. P. de Roever. Revised Lectures, V. 4111 of LNCS. Springer, 2006. P. 364—387.

28. **Simonite T.** $80 million hack shows the dangers of programmable money. MIT Technology review. 2016. URL: https://www.technologyreview.com/s/601724/80-million-hack-shows-the-dangers-of-programmable-money/

29. **Bhargavan K., Delignat-Lavaud A., Fournet S.** et al. Short Paper: Formal Verification of Smart Contract // PLAS '16 Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, 2016. P. 91—96.

30. **Li A., Long F.** Detecting Standard Violation Errors in Smart Contracts. ArXiv. 2018. URL: https://arxiv.org/abs/1812.07702

31. **Godefroid P., Klarlund N., Sen K.** DART: Directed Automated Random Testing // ACM SIGPLAN Conf. on Programming language design and implementation, 2005. P. 213—223.

32. **Sen K., Marinov D., Agha G.** CUTE: A Concolic Unit Testing Engine for C // 10th European Software Engineering Conf. Lisbon, Portugal, 2005. P. 263—272.

33. **Chatterjee K., Goharshady A. K., Velner Y.** Quantitative Analysis of Smart Contracts // Programming Languages and Systems. 2018. P. 739—767.

34. **Barrett C., Tinelli C.** Satisfiability Modulo Theories // Handbook of Model Checking. 2018. P. 305—343.

35. **Alt L., Reitwiessner C.** SMT-Based Verification of Solidity Smart Contract // 8th International Symposium, ISoLA 2018. Limassol, Cyprus. 2018. P. 376—388.

36. **Bai X., Cheng Z., Duan Z., Hu K**. Formal Modeling and Verification of Smart Contracts // 7th Int. Conf. on Software and Computer Applications — ICSCA, 2018. P. 322—326.

37. **Mikk E., Lakhnech Y., Siegel M., Holzmann G. J.** Implementing statecharts in PROMELA/SPIN // 2nd IEEE Workshop on. IEEE, 1998. P. 90—101.

38. **Nehai Z., Piriou P., Daumas F.** Model-Checking of Smart Contracts // 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 2018. P. 980—987.

39. **Wolfgang J., Gordon J., Schneider G.** Smart Contracts: A Killer Application for Deductive Source Code Verification // Principled Software Development. 2018. P. 1—18.

40. **Amani S., Bégel M., Bortin M., Staples M** Towards verifying ethereum smart contract bytecode in Isabelle/HOL // 7th ACM SIGPLAN International Conference. 2018. P. 66—77.

41. **Hirai Y.** Defining the Ethereum virtual machine for interactive theorem provers // Int. Conference on Financial Cryptography and Data Security. 2017. P. 520—535.

42. **Bohm C., Jacopini G.** Flow diagrams, Turing machines and languages with only two formation rules // Comm. ACM 9. 1966. Vol. 9, N. 5.P. 366—371.

43. **Yang Z., Lei H.** FEther: An Extensible Definitional Interpreter for Smart-contract Verifications in Coq // IEEE Access. 2019. Vol. 7. P. 37770—37791.

44. **Anand A., Boulier S., Cohen C., Sozeau M., Tabareau N.** Towards Certified Meta-Programming with Typed Template-Coq // ITP 2018: Interactive Theorem Proving. 2018. P. 20—39.

45. **Mavridou A., Laszka A., Stachtiari E., Dubey A.** VeriSolid: Correct-by-Design Smart Contracts for Ethereum // 23rd Int. Conference on Financial Cryptography and Data Security, 2018. P. 446—465.

46. **Basu A., Bensalem B., Bozga M., Combaz J., Jaber M., Nguyen T. H., Sifakis J.** Rigorous component-based system design using the bip framework // IEEE Software. 2011. Vol. 28, N. 3. P. 41—48.

47. **Bliudze S., Cimatti A., Jaber M., Mover S., Roveri M., Saab W., Wang Q.** Formal verification of infinite-state BIP models // In: Proceedings of the 13th International Symposium on Automated Technology for Verification and Analysis (ATVA). Springer, 2015. P. 326—343.

48. **Zurawski R.** Volcano: Enabling Correctness by Design // Networked Embedded Systems. 2017. Chapter 19.

49. **Yang Z., Lei H., Qian W.** A Hybrid Formal Verification System in Coq for Ensuring the Reliability and Security of Ethereum-based Service Smart Contracts. URL: https://arxiv.org/vc/arxiv/papers/1902/1902.08726v1.pdf

50. **Duchmann F., Koschmider A.** Validation of Smart Contracts Using Process Mining // ZEUS. 2019. Workshop on Services and their Composition Proceedings of the 11th Central European Workshop on Services and their Composition, Bayreuth, Germany, February 14—15, 2019. P. 13—16.

51. **Jiang B., Liu Y., Chan W. K.** ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection // 33rd ACM/IEEE Int. Conference, 2018. P. 259—269.

52. **Azzopardi S., Ellul J., Peace G. J.** Monitoring Smart Contracts: ContractLarva and Open Challenges Beyond // 18th International Conference on Runtime Verification. Cyprus. 2018. P. 113—137.

53. **Colombo C., Ellul J., Pace G. J.** Contracts over Smart Contracts: Recovering from Violations Dynamically // Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice, 2018. P. 300—315.

54. **Guth F., Wüstholz V., Christakis M., Müller P.** Specification Mining for Smart Contracts with Automatic Abstraction Tuning. 2018. URL: https://arxiv.org/abs/1807.07822.

55. **Charlier J., State R., Hilger J.** Modeling Smart Contracts Activities: A Tensor based Approach // European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML-PKDD) — Workshop on MIning DAta for financial application. 2019. URL: https://arxiv.org/pdf/1905.09868.pdf

56. **Kolda T. G., Bader B. W.** Tensor Decompositions and Applications // SIAM Review, 2009. P. 455—500.

57. **Liu H., Liu C., Zhao W., Jiang Y., Sun J.** S-gram: Towards Semantic-Aware Security Auditing for Ethereum Smart Contracts // 33rd ACM/IEEE International Conference. 2018. P. 814—819.

58. **Martin J. H., Jurafsky D**. Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition. 2nd edition. Upper Saddle River: Pearson/Prentice Hall. 2008. 1044 p.

59. **Somin S., Gordon G., Altshuler Y.** Network Analysis of ERC20 Tokens Trading on Ethereum Blockchain // Unifying Themes in Complex Systems IX. Proceedings of the Ninth International Conference on Complex Systems, 2018. P. p 439—450.

60. **Parizi M., Singh A., Dehghantanha A.** Smart Contract Programming Languages on Blockchains: An Empirical Evaluation of Usability and Security // International Conference on Blockchain, 2018. P. 75—91.

61. **Sergey I., Kumar A., Hobor A.** et al. Scilla: a Smart Contract Intermediate-Level Language. URL: https://arxiv.org/pdf/1801.00687.pdf

62. **Hirai Y.** Bamboo. URL: https://github.com/pirapira/bamboo

63. **Connors E. M., Augusto Muñoz C., Schnur C., Siminiceanu R.** Static Verification of Spacecraft Procedures // AIAA Infotech@Aerospace Conference, 2009. P. 2—5.

64. **Intellij** Solidity overview. URL: https://plugins.jetbrains.com/plugin/9475-intellij-solidity

65. **Remix** documentation.URL: https://remix.readthedocs.io/en/stable/

Systematic Review of Automatic Verification of Smart-Contracts

I. A. Fedotov, ivan.fedotov@phystech.edu, **A. S. Khritankov,** anton.khritankov@acm.org,
Moscow Institute of Physics and Technology, Dolgoprudny, Moscow Region, 141701, Russian Federation

Corresponding author:

Fedotov Ivan A., PhD Student, Moscow Institute of Physics and Technology, Dolgoprudny, Moscow Region, 141701, Russian Federation
E-mail: ivan.fedotov@phystech.edu

Received on October 25, 2019
Accepted on December 09, 2019

A smart contract is a special kind of software code that runs on a blockchain platform. As other software smart contracts can contain errors and vulnerabilities that could lead to substantial adverse results and financial losses. These risks are compounded by the growing popularity of blockchain-based systems, devices, and infrastructure in finance, legal, energy and other domains. Early detection of software errors can reduce losses and automatic verification tools are a method of choice for situations where reliability and security is important.

This systematic review analyzes the state of the art in smart contracts verification in 2015—2019. The review gives a brief introduction to blockchain, smart contracts and decentralized applications (dApps), examines verification methods for smart contracts and related tools. Different verification methods are taken into account including formal verification, dynamic and static code analysis, deductive analysis, theorem provers tools, code and execution trace audit.

The following research questions have been considered:

- *What code verification methodologies are commonly applied to software verification?*
- *Which of them are applicable to smart contracts and what tools are available?*
- *What are current limitations in smart contract verification?*
- *What are possible future directions in smart contract verification?*

During the research process more than 100 academic papers and industrial reports, software tool documentation and guides were studied. As a result, 52 studies, basically in English, were included in the systematic review.

Keywords: *Smart-contract, blockchain, verification, systematic review, Ethereum, BitCoin, model checking*

For citation:

Fedotov I. A., Khritankov A. S. Systematic Review of Automatic Verification of Smart-Contracts, *Programmnaya Ingeneria,* 2020, vol. 11, no. 1, pp. 3—13.

DOI: 10.17587/prin.11.3-13

References

1. **Srivastava A., Bhattacharya P., Singh A., Mathur A.** A Systematic Review on Evolution of Blockchain Generations, *ITEE Journal,* 2018, vol. 7, no. 6, pp. 1—8.
2. **Perez D., Livshits B.** Smart Contract Vulnerabilities: Does Anyone Care? 2019. arXiv:1902.06710.
3. **Gelvez M.** Explaining the DAO exploit for beginners in Solidity, 2016, available at: https://medium.com/@MyPaoG/explaining-the-dao-exploit-for-beginners-in-solidity-80ee84f0d470
4. **Thomson I.** Parity: The bug that put $169m of Ethereum on ice? Yeah, it was on the todo list for months. 2017, available at: https://www.theregister.co.uk/2017/11/16/parity_flaw_not_fixed
5. **Macrinici D., Cartofeanu C., Gao S.** Smart Contract Applications within Blockchain Technology: A Systematic Mapping Study, *Telematics and Informatics*, 2018, vol. 35, no. 8, pp. 2337—2354.
6. **Wood G.** Ethereum: A secure decentralised generalised transaction ledger. 2016, available at: http://paper.gavwood.com
7. **Xinxin F., Qi C.** Roll-DPoS: A Randomized Delegated Proof of Stake Scheme for Scalable Blockchain-Based Internet of Things Systems,*15th EAI International Conference*, 2018, pp. 482—483.
8. **Grishchenko I., Maffei M., Schneidewind C.** A Semantic Framework for the Security Analysis of Ethereum Smart Contract, *POST 2018: Principles of Security and Trust*, 2018, pp. 246—248.
9. **Harz D., Knottenbelt W.** Towards Safer Smart Contracts: A Survey of Languages and Verification Methods. 2018. Preprint arXiv:1809.09805.
10. **Zakrzewski J.** Towards Verification of Ethereum Smart Contracts: A Formalization of Core of Solidity, *Verified Software. Theories, Tools, and Experiments. 10th International Conference, VSTTE 2018*, Oxford, UK, 2018, pp. 229—247.
11. **Ethereum** Foundation. Vyper documentation. 2018, available at: https://vyper.readthedocs.io/en/v0.1.0-beta.13/
12. **Buterin V.** A next-generation smart contract and decentralized application platform. Ethereum White Paper. 2014, available at: https://cryptorating.eu/whitepapers/Ethereum/Ethereum_white_paper.pdf
13. **ERC:** Token standard, available at: https://github.com/ethereum/eips/issues/20
14. **Silva R., Neiva F.** Systematic Literature Review in Computer Science — A Practical Guide, 2016, pp. 1—5.
15. **Siddiqui J. H., Rauf A., Ghafoor M. A**. Advances in Software Model Checking, *Advances in Computers*, 2018, vol. 108, pp. 59—89.
16. **Sistla A. P., Clarke E. M.** The complexity of propositional linear temporal logics, *J. of the ACM (JACM),* 1985, vol. 32, no. 3, pp. 733—749.
17. **Kulyamin V. V.** *Program verification methods.* All-Russian competition of review and analytical articles in the priority area "Information and telecommunication systems", 2008, 117 p. (in Russian).
18. **Kosmatov N., Prevosto V., Signoles J.** A lesson on proof of programs with Frama-C, *TAP 2013: International Conference on Tests and Proofs.* Springer, 2013, pp. 168—177.
19. ***Deductive** Software Verification. — The KeY Book/* Eds. W. Ahrendt, B. Beckert, R. Bubel et al., Springer, 2016, 702 p.

20. **Hähnle R., Huisman M.** Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools, *Computing and Software Science. State of the Art and Perspectives* Springer, 2019, pp. 345—373.

21. **Ahrendt W., Beckert B., Bubel R.** et al. The KeY platform for verification and analysis of Java programs, *In STTE'14*, vol. 8471 of LNCS. Springer, 2014, pp. 55—71.

22. **Reynders F.** *Modern API Design with ASP.NET Core 2*, Apress, 2018, 236 p.

23. **Grishchenko I., Maffei M., Schneidewind C.** Foundations and tools for the static analysis of ethereum smart contracts, *Computer Aided Verification. 30th International Conference, CAV 2018*, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, 2018, Proceedings, 2018, part I, pp. 51—78.

24. **Tikhomirov S., Voskresenskaya E., Ivanitskiy I., Takhaviev R., Marchenko E., Alexandrov Y.** SmartCheck: Static Analysis of Ethereum Smart Contracts, *ACM/IEEE 1st International Workshop on Emerging Trends in Software Engineering for Blockchain,* 2018, pp. 9—16.

25. **Parizi R. M., Dehghantanha A., Choo K.-K. R., Singh A.** Empirical Vulnerability Analysis of Automated Smart Contracts Security Testing on Blockchains, *CASCON'18*, 2018, available at: https://arxiv.org/pdf/1809.02702.pdf

26. **Lahiri K., Chen S., Wang Y., Dillig I.** Formal Specification and Verification of SmartContracts for Azure Blockchain, 2018, available at: https://www.microsoft.com/en-us/research/publication/formal-specification-and-verification-of-smart-contracts-for-azure-blockchain/

27. **Barnet M. Chang B.-Y. E., DeLine R., Jacobs B., Leino K. R. M.** Boogie: A modular reusable verifier for object-oriented programs, *Formal Methods for Components and Objects, 4th International Symposium, FMCO 2005*, Amsterdam, The Netherlands, 2005 / Eds. F. S. de Boer, M. M. Bonsangue, S. Graf, W. P. de Roever. Revised Lectures, V. 4111 of LNCS. Springer, 2006, pp. 364—387.

28. **Simonite T.** $80 million hack shows the dangers of programmable money, MIT Technology review, 2016, available at: https://www.technologyreview.com/s/601724/80-million-hack-shows-the-dangers-of-programmable-money/

29. **Bhargavan K., Delignat-Lavaud A., Fournet S.** et al. Short Paper: Formal Verification of Smart Contract, *PLAS '16 Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, 2016, pp. 91—96.

30. **Li A., Long F.** Detecting Standard Violation Errors in Smart Contracts. ArXiv. 2018, available at: https://arxiv.org/abs/1812.07702

31. **Godefroid P., Klarlund N., Sen K.** DART: Directed Automated Random Testing, *ACM SIGPLAN Conf. on Programming language design and implementation,* 2005. pp. 213—223.

32. **Sen K., Marinov D., Agha G.** CUTE: A Concolic Unit Testing Engine for C, *10th European Software Engineering Conf.*, Lisbon, Portugal, 2005, pp. 263—272.

33. **Chatterjee K., Goharshady A. K., Velner Y.** Quantitative Analysis of Smart Contracts, *Programming Languages and Systems*, 2018, pp. 739—767.

34. **Barrett C., Tinelli C.** Satisfiability Modulo Theories, *Handbook of Model Checking.* 2018, pp. 305—343.

35. **Alt L., Reitwiessner C.** SMT-Based Verification of Solidity Smart Contract*, 8th International Symposium, ISoLA 2018.* Limassol, Cyprus, 2018, pp. 376—388.

36. **Bai X., Cheng Z., Duan Z., Hu K.** Formal Modeling and Verification of Smart Contracts, *7th Int. Conf. on Software and Computer Applications — ICSCA,* 2018, pp. 322—326.

37. **Mikk E., Lakhnech Y., Siegel M., Holzmann G. J.** Implementing statecharts in PROMELA/SPIN, *2nd IEEE Workshop on. IEEE,* 1998, pp. 90—101.

38. **Nehai Z., Piriou P., Daumas F.** Model-Checking of Smart Contracts, *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData),* Halifax, NS, Canada*,* 2018, pp. 980—987.

39. **Wolfgang J., Gordon J., Schneider G.** Smart Contracts: A Killer Application for Deductive Source Code Verification, *Principled Software Development*, 2018, pp. 1—18.

40. **Amani S., Bégel M., Bortin M., Staples M.** Towards verifying ethereum smart contract bytecode in Isabelle/HOL, *7th ACM SIGPLAN International Conference*, 2018, pp. 66—77.

41. **Hirai Y.** Defining the Ethereum virtual machine for interactive theorem provers, *Int. Conference on Financial Cryptography and Data Security*, 2017, pp. 520—535.

42. **Bohm C., Jacopini G.** Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM 9,* 1966. vol. 9, no. 5, pp. 366—371.

43. **Yang Z., Lei H.** FEther: An Extensible Definitional Interpreter for Smart-contract Verifications in Coq, *IEEE Access*, 2019, vol. 7, pp. 37770—37791.

44. **Anand A., Boulier S., Cohen C., Sozeau M., Tabareau N.** Towards Certified Meta-Programming with Typed Template-Coq, *ITP 2018: Interactive Theorem Proving*, 2018, pp. 20—39.

45. **Mavridou A., Laszka A., Stachtiari E., Dubey A.** VeriSolid: Correct-by-Design Smart Contracts for Ethereum, *23rd Int. Conference on Financial Cryptography and Data Security*, 2018, pp. 446—465.

46. **Basu A., Bensalem B., Bozga M., Combaz J., Jaber M., Nguyen T. H., Sifakis J.** Rigorous component-based system design using the bip framework, *IEEE Software,* 2011, vol. 28, no. 3, pp. 41—48.

47. **Bliudze S., Cimatti A., Jaber M., Mover S., Roveri M., Saab W., Wang Q.** Formal verification of infinite-state BIP models, *In: Proceedings of the 13th International Symposium on Automated Technology for Verification and Analysis (ATVA)*,Springer, 2015, pp. 326—343.

48. **Zurawski R.** Volcano: Enabling Correctness by Design, *Networked Embedded Systems,* 2017. Chapter 19.

49. **Yang Z., Lei H., Qian W.** A Hybrid Formal Verification System in Coq for Ensuring the Reliability and Security of Ethereum-based Service Smart Contracts, available at: https://arxiv.org/vc/arxiv/papers/1902/1902.08726v1.pdf

50. **Duchmann F., Koschmider A.** Validation of Smart Contracts Using Process Mining, *ZEUS. 2019, Workshop on Services and their Composition Proceedings of the 11th Central European Workshop on Services and their Composition,* Bayreuth, Germany, February 14—15, 2019, pp. 13—16.

51. **Jiang B., Liu Y., Chan W. K.** ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection, *33rd ACM/IEEE Int. Conference*, 2018, pp. 259—269.

52. **Azzopardi S., Ellul J., Peace G. J.** Monitoring Smart Contracts: ContractLarva and Open Challenges Beyond, *18th International Conference on Runtime Verification*, Cyprus, 2018, pp. 113—137.

53. **Colombo C., Ellul J., Pace G. J.** Contracts over Smart Contracts: Recovering from Violations Dynamically, *Leveraging Applications of Formal Methods, Verification and Validation. Industrial Practice,* 2018, pp. 300—315.

54. **Guth F., Wüstholz V., Christakis M., Müller P.** Specification Mining for Smart Contracts with Automatic Abstraction Tuning. 2018, available at: https://arxiv.org/abs/1807.07822.

55. **Charlier J., State R., Hilger J.** Modeling Smart Contracts Activities: A Tensor based Approach, *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML-PKDD) — Workshop on MIning DAta for financial application,* 2019, available at: https://arxiv.org/pdf/1905.09868.pdf

56. **Kolda T. G., Bader B. W.** Tensor Decompositions and Applications, *SIAM Review.* 2009, pp. 455—500.

57. **Liu H., Liu C., Zhao W., Jiang Y., Sun J.** S-gram: Towards Semantic-Aware Security Auditing for Ethereum Smart Contracts, *33rd ACM/IEEE International Conference,* 2018, pp. 814—819.

58. **Martin J. H., Jurafsky D**. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. 2nd edition. Upper Saddle River: Pearson/Prentice Hall, 2008, 1044 p.

59. **Somin S., Gordon G., Altshuler Y.** Network Analysis of ERC20 Tokens Trading on Ethereum Blockchain, *Unifying Themes in Complex Systems IX. Proceedings of the Ninth International Conference on Complex Systems*, 2018, pp. 439—450.

60. **Parizi M., Singh A., Dehghantanha A.** Smart Contract Programming Languages on Blockchains: An Empirical Evaluation of Usability and Security, *International Conference on Blockchain*, 2018, pp. 75—91.

61. **Sergey I., Kumar A., Hobor A.** et al. Scilla: a Smart Contract Intermediate-Level Language, available at: https://arxiv.org/pdf/1801.00687.pdf

62. **Hirai Y.** Bamboo, available at: https://github.com/pirapira/bamboo

63. **Connors E. M., Augusto Muñoz C., Schnur C., Siminiceanu R.** Static Verification of Spacecraft Procedures, *AIAA Infotech@ Aerospace Conference*, 2009, pp. 2—5.

64. **Intellij** Solidity overview, available at: https://plugins.jetbrains.com/plugin/9475-intellij-solidity

65. **Remix** documentation, available at: https://remix.readthedocs.io/en/stable/

УДК 007:004.451:004.75 DOI: 10.17587/prin.11.14-20

М. Б. Кузьминский, канд. хим. наук, ст. науч. сотр., kus@free.net, Институт органической химии им. Н. Д. Зелинского РАН, Москва, **А. М. Чернецов,** канд. техн. наук, науч. сотр., an@ccas.ru, Вычислительный центр им. А. А. Дородницына ФИЦ ИУ РАН, Москва, доц., chernetsovam@mpei.ru, Национальный исследовательский университет "МЭИ", Москва

Современные средства параллельного программирования в модели распределенной памяти

Дан обзор реализаций MPI-средств распараллеливания, ориентированный в первую очередь на область высокопроизводительных вычислений. Для средств MPI приведен анализ производительности, которой удается добиться на основных реализациях MPI и при использовании современных межсоединений с удаленным прямым доступом к памяти RDMA (Intel Omni-Path, Infiniband EDR). Большое внимание в обзоре уделено односторонним RMA-коммуникациям, способствующим повышению производительности и поддержке перспективных PGAS-моделей распараллеливания. А распространенные тесты производительности SPEC MPI 2007, использующие MPI-приложения, зависят от большого числа параметров аппаратных и программных средств, являются менее актуальными для области высокопроизводительных вычислений и в настоящей работе не анализируются.

Ключевые слова: *средства параллельного программирования, Message Passing Interface (MPI), OpenMPI, MVAPICH2, Intel MPI, тестирование производительности*

Введение

Настоящая статья посвящена анализу средств распараллеливания (технологий параллельного программирования). Спектр таких средств очень широк, поэтому в работе рассмотрены только современные широко распространенные и доказавшие свою эффективность средства распараллеливания, ставшие стандартами. Учитывая тенденцию увеличения со временем числа доступных процессоров (процессорных ядер) во всех современных вычислительных системах от ПК до суперЭВМ, актуальность распараллеливания также возрастает в самых разных областях практического применения. Однако статья в первую очередь ориентируется на область высокопроизводительных вычислений (*High Performance Computing*, HPC). Применение распараллеливания давно стало одним из основных способов увеличения производительности и уменьшения необходимого времени расчета.

Хотя средства распараллеливания доступны в самых разных операционных системах, в HPC-области и на суперЭВМ в качестве основного решения стало применение кластерной архитектуры на базе операционной системы Linux. В первую очередь такие средства применяют на платформе x86-64, где используются высокоскоростные межсоединения с удаленным прямым доступом к памяти RDMA (*Remote Direct Memory Access*), не нагружающие процессоры удаленного узла — Infiniband, Ethernet с RDMA (RoCE или iWARP) или Intel Omni-Path. Приводимые в статье оценки производительности в первую очередь будут относиться к таким системам. Однако сами средства распараллеливания применяют, естественно, и для вычислительных систем с другими процессорами и межсоединениями.

Обычно модели параллельного программирования в соответствии с топологией применяемой памяти разделяют следующим образом:

- модели, использующие общую (разделяемую) память (для SMP и NUMA-систем);
- модели, использующие распределенную память (в том числе для кластеров);
- гибридные модели с сочетанием распределенной и общей памяти (последнее типично для современных кластеров с многопроцессорными узлами), например, представленные в обзорах [1, 2].

Однако основу для достижения высокой производительности на современных суперЭВМ, основывающихся на кластерной архитектуре, предоставляют средства распараллеливания с распределенной памятью, которые в настоящее время являются самыми широко распространенными. Именно эти средства распараллеливания проанализированы в настоящем обзоре. При этом OpenMP [3] или перспективные средства PGAS (*Partioned Global Address Space*) [4], как и специализированные средства распараллеливания для ускорителей, далее не рассматриваются.

Распределенная память. Модель передачи сообщений

Различные модели передачи сообщений относятся к параллельному программированию с распределенной памятью, в них процессы обмениваются данными между собой, отправляя и получая сообще-

ния. Это типично для кластеров из вычислительных узлов, соединенных коммуникационной сетью (межсоединениями), где у каждого узла имеется своя локальная память, а прямой доступ к памяти других узлов отсутствует.

Существует много различных моделей передачи сообщений, которые представляются в виде программных библиотек. Далее остановимся только на самых широко применяемых и доказавших свою эффективность средствах MPI [5]. Естественно, такие модели предполагают применение библиотечных вызовов. Применение MPI как API типу MIMD (множественный поток команд, множественный поток данных по классификации Флинна [5]). Интерфейс MPI может использоваться в любых вычислительных системах как с общей, так и с распределенной памятью.

В стандарте MPI 2.0 была обеспечена возможность динамического изменения числа процессов, расширены коллективные коммуникации и добавлены односторонние коммуникации MPI_Put/MPI_Get. Функциональные возможности последних были расширены в MPI 3.0. Стандарт MPI 3.0 позволил напрямую использовать возможности RDMA. Еще в MPI 2.0 начал поддерживаться параллельный ввод-вывод, а последняя версия стандарта MPI 3.1 содержит уже и коллективные неблокирующие операции ввода-вывода [6].

Односторонние коммуникации основываются на механизме удаленного доступа к памяти (*Remote Memory Access*, RMA). Односторонние коммуникации в MPI имеют доступ только к так называемому окну, т. е. к области памяти на цели. Цель объявляет о доступности окна для других процессов. Альтернативой использованию окон является использование виртуальной общей памяти. Модель MPI RMA позволяет заблокировать окно. Механизм RMA MPI имеет два режима — активный и пассивный. В активном режиме цель устанавливает период времени, в течение которого можно получить доступ к окну. В пассивном режиме RMA-процесс объявления цели доступ к окну не ограничивает. Возможность эффективно использовать MPI RMA в более общей модели параллельного программирования PGAS, применяющей совместно стили общей и распределенной памяти, способствовала миграции некоторых реализаций MPI в направлении PGAS.

В настоящее время существует много конкретных реализаций MPI, в том числе MPICH, MVAPICH, OpenMPI, Intel MPI, Mellanox X-MPI, IBM Spectrum MPI, HPE MPI, Cray MPI. Из этих реализаций в контексте настоящей статьи подробнее остановимся только на трех наиболее активно используемых в широко распространенных HPC Linux, а именно на кластерах с архитектурой x86-64 — OpenMPI, MVAPICH2 и на коммерческом варианте Intel MPI.

Следует отметить, что эти реализации MPI с RDMA-межсоединениями предполагают использование универсальных низкоуровневых средств API, базирующихся на драйверах. В качестве таких средств ранее применяли ibverbs из OFED (*OpenFabrics Enterprise Distribution*) и uDAPL (*User Direct Access Programming Library*) — стандартный API-интерфейс для RDMA-межсоединений Infiniband, iWARP и RoCE (см., например, [7]). Программные средства OFED имеют более широкие возможности и давно включают в себя и uDAPL [8, 9]. Эти низкоуровневые API используют не только для MPI, но и для других работающих с RDMA API-средств распараллеливания.

Несмотря на то что OFED является некоммерческим программным продуктом с доступным исходным текстом, производители межсоединений его дорабатывают для своих аппаратных средств. При этом создают в том числе и свои варианты низкоуровневых API. Например, у компании "Mellanox" имеется собственный вариант OFED для межсоединений Infiniband и RoCE, включающий низкоуровневый драйвер MLX5 [10]. Специализированной для Infiniband альтернативой таким продуктам являются свободно доступные с исходным текстом программные средства UCX (*Unified Communication X*) [11—13]. Они могут не только применяться с MPI, а используются, например, в более широком программном инструментарии Mellanox HPC-X [14], который применяют в MPI и PGAS-средствах распараллеливания. Следует также указать на библиотеку для Infiniband — UCCS (*Universal Common Communication Substrate*) [15], которая может считаться альтернативой низкоуровневой (фоновой) оптимизированной библиотеке Mellanox MXM. Она предоставляет высокопроизводительные коммуникационные примитивы, обеспечивая при этом интероперабельность сетей и поддержку разных моделей программирования, включая MPI и PGAS-средства. С помощью UCCS реализуются различные операции коммуникации, включая односторонние и двусторонние операции "точка-точка", коллективные и удаленные атомарные операции.

Относительно недавно для OFED появилась альтернатива в виде продукта OFI (*OpenFabrics Interfaces*) [16]. Библиотека libfabric, входящая в состав OFI и используемая в некоторых современных реализациях MPI, представляет API с более высоким уровнем абстракции, чем у libibverbs. Однако поставщики ibverbs (производители оборудования для соответствующих межсоединений) позволяют программным средствам, применяющим OFI, использовать libibverbs для части операций.

Рассмотрение этих низкоуровневых API выходит за рамки данной статьи, так как они напрямую не используются в распараллеливаемых приложениях. Однако следует заметить, что их применение принципиально важно для получения высокой производительности MPI, а конкретная реализация MPI может обеспечивать возможность использования различных таких API. Так, Intel MPI работает с разными низкоуровневыми API (см., например, [17]).

Потенциально важным для получения высокой производительности работы MPI при распараллеливании на общей памяти является применение XPMEM, модуля ядра Linux, который позволяет процессу отображать память другого процесса в виртуальном адресном пространстве [18]. Вместе с тем

разработчики XPMEM указывают на возможные ошибки применения его последней версии [18], поэтому продукт XPMEM можно считать пока недостаточно стабильным. Работа с XPMEM поддерживается во всех трех рассматриваемых далее реализациях MPI, и в целом XPMEM следует считать актуальным для применения с MPI в ближайшем будущем.

Производительность работы MPI зависит от огромного числа показателей и параметров: используемые межсоединения или общая память, версии MPI-реализации, используемый низкоуровневый API, применяемая тестовая система и конкретные анализируемые MPI-операции и т. д. Возможное эффективное решение для оптимизации производительности может основываться на механизмах динамически изменяемого выбора конкретной применяемой реализации. Подробнее вопросы производительности будут рассмотрены далее.

Несмотря на то что в рамках настоящей публикации мы останавливаемся только на трех реализациях MPI, нельзя не отметить также реализацию MPICH. Она в настоящее время доступна в версии 3.2.1 в исходных текстах [19] и часто используется разработчиками для создания собственных реализаций MPI. Версия MPICH активно развивается, обеспечивая поддержку работы поверх современных межсоединений, низкоуровневых средств коммуникаций и с новыми версиями MPI (см., например, [20]). Из рассматриваемых далее реализаций она применялась для MVAPICH и Intel MPI. Средства MPICH поддерживают работу с UCX, поэтому базирующиеся на MPICH реализации также могут применять UCX. Современные версии MPICH используют низкоуровневый API libfabric и имеют широкий круг поддерживаемых межсоединений. Однако с точки зрения производительности MPICH представляет интерес только для разработчиков, которые ее улучшают в своих конкретных реализациях.

Все три рассматриваемые далее реализации MPI поддерживают стандарт MPI 3.1 и работу со всеми упоминавшимися RDMA-межсоединениями 100 Гбит/с, а также работу с DAPL и OFED verbs, UCX и XPMOD. К вопросам производительности этих реализаций MPI, как и поддержки работы с ускорителями, обратимся позднее, а сейчас в кратком изложении рассмотрим их современные версии. Здесь следует отметить, что новые версии реализаций MPI появляются очень быстро, как следствие, данные о производительности быстро могут устаревать.

Текущая версия OpenMPI 4.0.1 применяет средства libfabric. Она, как и ранее, поставляется в виде открытого исходного текста [21, 22]. Это преимущество для OpenMPI, так как она может быть реализована на разных ОС, в том числе на разных дистрибутивах Linux, подстраивая создаваемые библиотеки под различные компиляторы. Для дистрибутивов имеются, естественно, и двоичные пакеты, в том числе и средства для работы с Python. В OpenMPI доступны и средства, выходящие за рамки MPI и относящиеся уже к PGAS. Как отмечено в работе [22], IBM Spectrum MPI базируется на OpenMPI. В текущей версии имеется поддержка работы на новых процессорах архитектуры, отличной от x86-64.

Другой свободно доступной в том числе в исходных текстах реализацией MPI является MVAPICH [23], в настоящее время — это семейство MVAPICH2, а его текущая версия — 2.3 [24]. В качестве низкоуровневого API MVAPICH2 применяет libibverbs. Реализация MVAPICH2 отличается очень быстрым появлением новых версий и поддержкой целого ряда разных вариантов библиотек, оптимизированных для различных целей. Так, библиотека MVAPICH2-GDR оптимизирована для работы с ускорителями Nvidia, а специальный вариант MVAPICH2-X когда-то ориентировался на Infiniband, а в настоящее время работает со всеми рассматриваемыми в статье RDMA-межсоединениями, и наиболее яркой его особенностью стала поддержка PGAS.

Продукт Intel MPI является коммерческой реализацией MPI, что традиционно означало более высокий уровень производительности, в том числе относительно MVAPICH2 и OpenMPI (см., например, [25]). В настоящее время текущая версия — это Intel MPI Library 2019 Update 4. В ней поддерживается не только Intel Omni-Path, но и все другие рассматриваемые RDMA-межсоединения [26]. А в качестве низкоуровневого API последние версии Intel MPI ориентируются на libfabric [27].

Тесты производительности при использовании MPI

Существует целый ряд различных тестов производительности (Пт), которая достигается при использовании MPI. Тесты SPEC MPI 2007 [28] (последняя версия — 2.0 [29]) предоставляют главным образом информацию о Пт вычислительных систем, а не собственно оценки быстродействия MPI-коммуникаций. Такие тесты могут быть актуальны для оценок качества масштабирования Пт с ростом числа используемых процессов/процессорных ядер. SPEC MPI можно использовать и для оценок эффективности применения разных реализаций MPI и их версий, сопоставляя одинаковые вычислительные системы.

Анализ Пт MPI-коммуникаций дают микротесты, которые показывают данные о задержках и пропускной способности при разных операциях MPI. Наиболее широко применяют тесты OMB (*Ohio State University Micro Benchmarks*) и IMB (*Intel Micro Benchmark*). Тесты OMB быстро развиваются, в настоящее время доступна уже версия 5.6 [30].

В комплексном наборе тестов OMB измеряются задержка и полоса пропускания для двухточечных, односторонних и коллективных операций MPI в широком диапазоне размеров сообщений.

Набор IMB [31] также поддерживает тестирование операций всех стандартов MPI вплоть до версии 3 и включает, в частности, компоненты для тестирования Пт ввода-вывода.

В контексте настоящего обзора сосредоточимся только на современных данных о Пт средств MPI, использующих наиболее распространенные высокоскоростные межсоединения с пропускной способностью 100 Гбит/с: Infiniband EDR; Intel Omni-Path; Ethernet

RoCE/iWARP. Единственным представляющим интерес исключением являются данные из работы [32] по тестам с применением OMB и IMB для отечественного межсоединения Ангара, с сопоставлением полученных результатов для Infiniband FDR.

В отличие от SPEC MPI 2007 данные микротестов MPI позволяют оценить достигаемые характеристики коммуникаций, включая и показатели самих межсоединений на реальном программном (а не на теоретическом аппаратном) уровне. Как уже отмечалось выше, получаемые оценки зависят от большого числа показателей. В аппаратном плане эти оценки зависят не только от показателей самих межсоединений, но и в частности от используемых процессоров/процессорных ядер и оперативной памяти. Однако все-таки в основном реально получаемые результаты зависят от выбора межсоединения.

Что касается данных о производительности MPI при коммуникациях через общую оперативную память, то в настоящей публикации они не рассматриваются, а исключением являются специальные случаи. Причина в том, что зачастую в приложениях вообще для распараллеливания внутри систем с общей памятью используется OpenMP в сочетании с MPI для распараллеливания, например, между узлами кластера. Поэтому актуальность тестов MPI внутри систем с общей памятью несколько ниже.

Как уже отмечалось выше, результаты микротестов Пт MPI, не говоря уже о Пт тестов SPEC MPI 2007, зависят не только от характеристик Пт межсоединения, но и от Пт процессорных ядер и оперативной памяти. Более того, на задержки при работе с NUMA-серверами может заметно влиять расположение сетевой платы межсоединения. Такая плата может располагаться на PCIe-шине выполняющего MPI-операцию процессора или процессора соседнего сокета материнской платы [33]. Поэтому достаточно точное сопоставление Пт межсоединений или реализаций MPI возможно только при сравнении данных тестов, выполненных на одной и той же вычислительной системе.

Одни межсоединения или реализации MPI могут давать высокую Пт на одних MPI-реализациях, а другие — на других. Кроме того, реализации MPI имеют много разных параметров, которые могут влиять на Пт. Наконец, быстрое появление новых версий реализаций MPI способствует быстрому устареванию данных об их относительной Пт.

Например, обычно считается, что коммерческий Intel MPI показывает более высокую Пт, чем OpenMPI и MVAPICH2, что подтверждается и данными соответствующих тестов [25]. Однако имеются и результаты для новых версий OpenMPI и MVAPICH2, где на некоторых тестах Пт они показывают более высокие результаты. Считается, например, что OpenMPI традиционно является высокопроизводительной реализацией, а MVAPICH не превосходит по Пт OpenMPI на большинстве приложений [34]. Вместе с тем существует ряд работ, где показывается более высокая Пт MVAPICH2. А результаты тестов Пт MPI полезны еще и для оценок Пт новых аппаратных средств межсоединений, и вообще для эффективного написания HPC-программ.

Начнем рассмотрение данных о Пт MPI с сопоставления межсоединений Mellanox Infiniband EDR с Intel Omni-Path, представленных на рис. 1 и 2 (см. вторую сторону обложки). Эти данные получены для реализации MVAPICH2-2.3 [24]. На рис. 1, *а* видно, что по задержке при односторонних MPI-операциях для небольших сообщений Omni-Path при некоторых их длинах превосходит EDR. Для больших сообщений (что можно уже отнести и к оценке пропускной способности) иногда EDR бывает быстрее Omni-Path (см. рис. 1, *б*). Однако чаще показатели Пт MPI для EDR и Omni-Path близки (см., например, [35]). Понятно, что в других реализациях MPI эти соотношения могут быть другими.

Как отмечено выше, Пт MPI может существенно зависеть от используемых реализациями MPI программных средств, в том числе от низкоуровневых API. На рис. 3 (см. третью сторону обложки) показано, что применение UCX позволило поднять пропускную способность OpenMPI при работе с EDR до существенно более высокого уровня, чем в Intel MPI [36] (данные о версиях реализаций MPI в работе [36] не приведены). Другим важным средством, позволяющим поднять Пт работы MPI, является использование XPMEM. На рис. 4 (см. третью сторону обложки) показано, как применение XPMEM позволяет уменьшить задержки на тестах OMB для разных размеров сообщений так, что MVAPICH2-2.3 начинает превосходить Intel MPI 2017 v. 1.1.3, хотя без XPMEM он уступал Intel-реализации [24, 37].

Современные данные по задержкам и пропускной способности односторонних операций MPI_Get на кластере с Infiniband EDR в тестах OMB на реализациях OpenMPI и Intel MPI представлены на рис. 5 и 6, соответственно [38]. По Пт Intel MPI существенно опережает OpenMPI лишь на ограниченном диапазоне размеров пакетов.

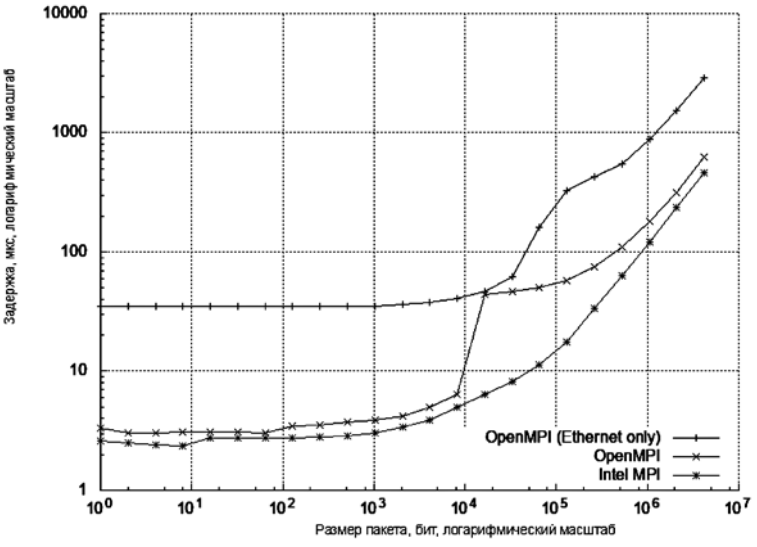

Рис. 5. Тест OMB — задержка MPI_Get

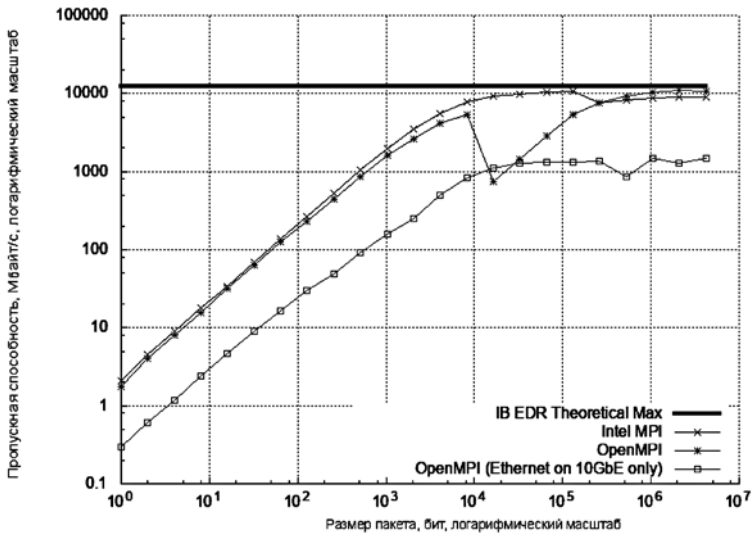

Рис. 6. Тест OMB — пропускная способность MPI_Get

В работе [39] было показано, что при работе с RoCE на микротестах IMB применение Intel MPI на коллективных MPI-операциях показывает существенно более высокую Пт (более низкие значения задержек), чем в тестах OMB с использованием MVAPICH2. При операциях между двумя процессами существенно более высокая пропускная способность достигается в тестах OMB на базе MVAPICH2. Однако в настоящее время актуальность этих данных снизилась, в том числе и потому, что версии обеих реализаций MPI уже кардинально переработаны. Интересные данные по Пт MPI при работе с несмежными данными, в том числе в кластере с межсоединением Omni-Path, с сопоставлением полученных результатов для Intel MPI и MVAPICH2 получены в работе [40].

Можно указать и на другие работы, где проводили сравнение Пт OpenMPI, MVAPICH2 и Intel MPI, например, в работе [41] для их сопоставления применяли систему тестов MadMPI, но к настоящему времени эти данные также относятся к устаревшим версиям реализаций MPI. В работе [24], например, приведены данные теста SPEC MPI 2007 с применением межсоединения Omni-Path для связи серверов с современными высокопроизводительными процессорами Intel Skylake-архитектуры. Результаты на реализациях Intel MPI 18.0.0 и MVAPICH2-2.3 обычно близки, однако на одном из приложений этого теста использование MVAPICH2 дало повышение Пт на 38 %. В работе [42] приведены результаты для Infiniband EDR, указывающие на иногда более высокую Пт при работе с короткими сообщениями бета-версии Intel MPI 2019 по сравнению с MVAPICH2-2.3.

Приведенные выше данные тестов Пт показывают, что в настоящее время правильным представляется мнение (из доступного на официальном сайте Intel доклада [43]), что эталоном для теста Пт является нужное потребителю приложение.

Заключение

В настоящей работе проведен сопоставительный анализ современных эффективных и широко используемых реализаций средств распараллеливания MPI в модели распределенной памяти. Данные об их производительности не позволяют сделать однозначный выбор в пользу какой-либо одной из реализаций. В случае базирования на производительности HPC-приложений целесообразным представляется отбор данных для конкретных приложений, а не ориентация на "интегральные" данные SPEC MPI 2007.

Список литературы

1. **Dimakopoulos V. V.** Parallel programming models // Smart Multicore Embedded Systems. Springer New York, 2014. P. 3—20.
2. **Abdulbaqi J.** Programming at Exascale: Challenges and Innovations. arXiv preprint arXiv:1809.10023. 2018, URL: https://arxiv.org/pdf/1809.10023.pdf
3. **The OpenMP** API specification for parallel programming. URL: https://www.openmp.org
4. **Sundriyal V., Gaenko A., Sosonkina M., Zhang Z.** Energy saving strategies for parallel applications with point-to-point communication phases // Journal of Parallel and Distributed Computing. 2013. Vol. 73, N. 8. P. 1157—1169.
5. **Воеводин В. В., Воеводин Вл. В.** Параллельные вычисления. СПб.: BHV-Петербург, 2002. 608 с.
6. **MPI:** A Message-Passing Interface Standard. Version 3.1. Message Passing Interface Forum. June 4, 2015. URL: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
7. **Chai L., Noronha R., Panda D. K.** MPI over uDAPL: Can High Performance and Portability Exist Across Architectures? // Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06), IEEE, 2006. P. 19—26.
8. **Libfabric** Programmer's Manual. URL: http://www.libfabric.org
9. **Кузьминский М. Б.** Infiniband: аппаратные средства и поддержка в Linux // Открытые системы. СУБД. 2008. № 7. URL: https://www.osp.ru/os/2008/07/5478314/
10. **Mellanox** OFED for Linux Release Notes. URL: http://www.mellanox.com/related-docs/prod_software/Mellanox_OFED_Linux_Release_Notes_4_5-1_0_1_0.pdf
11. **Papadopoulou N., Oden L., Balaji P.** A performance study of UCX over InfiniBand // Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE Press, 2017. P. 345—354.
12. **Unified** Communication X. URL: http://www.openucx.org
13. **Shamis P., Venkata M. G., Lopez M. G.** et al. UCX: an open source framework for HPC network APIs and beyond // 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects. IEEE, 2015. P. 40—43.
14. **Mellanox** HPC-X™ Software Toolkit User Manual. URL: http://www.mellanox.com/related-docs/prod_acceleration_software/HPC-X_Toolkit_User_Manual_v2.3.pdf
15. **Shamis P., Venkata M. G., Poole S., Welch A., Curtis T.** Designing a high performance openshmem implementation using universal common communication substrate as a communication middleware // OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools. OpenSHMEM 2014. Lecture Notes in Computer Science, vol. 8356. Springer, Cham, 2014. P. 1—13.
16. **Grun P., Hefty S., Sur S.** et al. A brief introduction to the OpenFabrics interfaces—a new network API for maximizing high performance application efficiency // 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects. IEEE, 2015. P. 34—39.
17. **Atanassov E., Barth M., Byckling M.** et al. Best Practive Guide Intel Xeon Phi v2.0. 2017. URL: http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-Intel-Xeon-Phi-1.pdf
18. **Xpmem.** URL: https://github.com/hjelmn/xpmem
19. **MPICH** User's Guide. URL: https://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1-userguide.pdf

20. **Fujita H., Cao C., Sur S.** et al. Efficient Implementation of MPI-3 RMA over OpenFabrics Interfaces // Parallel Computing. 2019. Vol. 87. P. 1—10.
21. **Open MPI:** Version 4.0. URL: https://www.open-mpi.org/software/ (дата обращения: 20.06.19).
22. **Squyres J., Bosilca G., Gabriel E., Ladd J.** Open MPI State of the Union XI Community Meeting SC18, 2018, URL: https://www.open-mpi.org/papers/sc-2018/Open-MPI-SC18-BOF.pdf
23. **MVAPICH:** MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. URL: http://mvapich.cse.ohio-state.edu/
24. **Panda D. K.** Overview of the MVAPICH Project: Latest Status and Future Roadmap, MVAPICH2 User Group (MUG) Meeting, 2018. URL: http://mug.mvapich.cse.ohio-state.edu/static/media/mug/presentations/18/panda-mug-18.pdf
25. **Introduction** to Intel® Software for Parallel Programming. URL: http://school-2010.hpc-russia.ru/files/lectures/lecture_080710_sergeev.pdf
26. **James T.** Intel MPI Library Release Notes. URL: https://software.intel.com/en-us/articles/intel-mpi-library-release-notes
27. **Gladkov D.** Intel MPI Library 2019 over libfabric. 2018. URL: https://software.intel.com/en-us/articles/intel-mpi-library-2019-over-libfabric
28. **Müller M. S., Van Waveren M., Lieberman R.** et al. SPEC MPI2007 — an application benchmark suite for parallel systems using MPI // Concurrency and Computation: Practice and Experience. 2010. Vol. 22, N. 2. P. 191—205.
29. **New** SPEC MPI2007 V2.0 benchmark adds data suite for systems with up to 2048 cores. URL: https://www.spec.org/mpi2007/press/v20release.html
30. **MVAPICH:** MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. Benchmarks. URL: http://mvapich.cse.ohio-state.edu/benchmarks/
31. **Gergana S.** Introducing Intel MPI Benchmarks (2018). URL https://software.intel.com/en-us/articles/intel-mpi-benchmarks
32. **Агарков А. А., Исмагилов Т. Ф., Макагон Д. В.** и др. Результаты оценочного тестирования отечественной высокоскоростной коммуникационной сети Ангара // Суперкомпьютерные дни в России: Тр. Междунар. конф. 26—27 сентября 2016 г., Москва. М.: МГУ. 2016. С. 626—639.
33. **Panda D. K., Subramoni H.** InfiniBand, Omni-Path, and High-Speed Ethernet: Advanced Features, Challenges in Designing HEC Systems, and Usage, IT4 Innovations'18. 2018, http://prace.it4i.cz/sites/prace.it4i.cz/files/files/iohea-01-2018-slides.pdf
34. **Жуматий С. А., Стефанов К. С.** Суперкомпьютеры: администрирование. М.: Макс-пресс, 2018. 448 с.
35. **Erwin J., Mascarenhas E.** Intel Omni-Path Architecture Performance Optimization // Intel Extreme Performance User Group (IXPUG) Conference. September 25—28 2018, Hillsboro (USA). URL: https://www.ixpug.org/resources/intel-omni-path-architecture-performance-optimization
36. **Squyres J., Bosilca G., Hursey J.** et al. Open MPI State of the Union XI Community. 2017. URL: https://www.open-mpi.org/papers/sc-2017/Open-MPI-SC17-BOF.pdf
37. **Hashmi J.** Designing Shared Address Space MPI libraries in the Many-core Era. 2018. URL: http://mvapich.cse.ohio—state.edu/static/media/talks/slide/Jahanzeb_sc18_booth.pdf
38. **UL HPC** MPI Tutorial: Building and Runnning OSU Micro-Benchmarks. URL: https://ulhpc-tutorials.readthedocs.io/en/latest/parallel/mpi/OSU_MicroBenchmarks/
39. **Kaur G., Kumar M., Bala M.** Comparing Ethernet and Soft RoCE for MPI Communication // IOSR Journal of Computer Engineering (IOSR—JCE). 2014. Vol. 16, Issue 4. P. 52—58.
40. **Eijkhout V.** Performance of MPI sends of non—contiguous data. URL: https://arxiv.org/pdf/1809.10778.pdf
41. **Denis A., Trahay F.** MPI overlap: Benchmark and analysis // Parallel Processing (ICPP), 2016, 45th International Conference on. IEEE, 2016. P. 258—267.
42. **Sur S.** Status of OpenFabrics over Verbs based Fabrics, 6th Annual MVAPICH User Group (MUG). 2018. URL: http://mug.mvapich.cse.ohio-state.edu/static/media/mug/presentations/18/sur-mug-18.pdf
43. **Vienne J.** All the things you need to know about Intel MPI Library. 2016. URL: https://www.intel.com/content/dam/www/public/us/en/documents/presentation/things-mpi-library.pdf

Modern Parallel Programming Tools in a Distributed Memory Model

M. B. Kuzminsky, kus@free.net, Zelinsky Institute of Organic Chemistry RAS, Moscow, 119991, Russian Federation, **A. M. Chernetsov,** chernetsovam@mpei.ru, National Research University "Moscow Power Engineering Institute", Moscow, 111250, Russian Federation, an@ccas.ru, Dorodnicyn Computing Centre CSC RAS, Moscow, 119333, Russian Federation

Corresponding author:

Kuzminsky Mikhail B., Ph. D., Senior Researcher, Zelinsky Institute of Organic Chemistry RAS, Moscow, 119991, Russian Federation
E-mail: kus@free.net

Received on September 02, 2019
Accepted on October 21, 2019

The paper provides an overview of the implementation of MPI parallelization tools, focused primarily on the field of high performance computing (HPC). Low-level communications software used in various MPI implementations (such as uDAPL, OFED, OFI, etc.), which significantly affect the achieved performance, is considered. For the most widely used MPI implementations in HPC (OpenMPI, MVAPICH2, Intel MPI), an analysis is made of the performance achieved in their modern versions using high-speed interconnects with remote direct access to RDMA memory (Ethernet 100 Gbit — RoCE / iWARP, Intel Omni-Path, Infiniband EDR) for communication of the most widely used computing nodes based on x86-64 processors. Among the characteristics of achieved performance, indicators of achieved throughput and message transmission delays are considered, including data in terms of achieved performance in widely used MPI microtests OMB (Ohio State University Micro Benchmarks) and IMB (Intel Micro Benchmark), taking into account its dependence from number of message and their sizes. These latency and throughput data also characterize the actual performance indicators of the interconnect hardware itself. More attention is paid in the review to one-way RMA communications, supported since MPI 2.0, which helps to increase productivity and support promising PGAS parallelization models. On the contrary, widespread performance tests using MPI applica-

tions, SPEC MPI 2007, that depend on a huge number of hardware and software parameters, not only related to interconnects and parallelization tools, are less relevant for HPC and are not analyzed in the paper.

Keywords: *parallel programming tools, MPI, OpenMPI, MVAPICH2, Intel MPI, performance testing*

For citation:

Kuzminsky M. B., Chernetsov A. M. Modern Parallel Programming Tools in a Distributed Memory Model, *Programmnaya Ingeneria,* 2020, vol. 11, no. 1, pp. 14—20.

DOI: 10.17587/prin.11.14-20

References

1. **Dimakopoulos V. V.** Parallel programming models, *Smart Multicore Embedded Systems,* Springer New York, 2014, pp. 3—20.
2. **Abdulbaqi J.** Programming at Exascale: Challenges and Innovations, arXiv preprint arXiv:1809.10023, 2018, https://arxiv.org/pdf/1809.10023.pdf
3. **The OpenMP** API specification for parallel programming, available at: https://www.openmp.org
4. **Sundriyal V., Gaenko A., Sosonkina M., Zhang Z.** Energy saving strategies for parallel applications with point-to-point communication phases, *Journal of Parallel and Distributed Computing,* 2013, vol. 73, no. 8, pp. 1157—1169.
5. **Voevodin V. V., Voevodin Vl. V.** *Parallel Computations*, Saint-Petersburg, BHV-Peterburg, 2002, 608 p. (in Russian).
6. **MPI:** A Message-Passing Interface Standard. Version 3.1. Message Passing Interface Forum. June 4, 2015, available at: https://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf
7. **Chai L., Noronha R., Panda D. K.** MPI over uDAPL: Can High Performance and Portability Exist Across Architectures? *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, IEEE, 2006, pp. 19—26.
8. **Libfabric** Programmer's Manual, available at: http://www.libfabric.org
9. **Kuzminsky M. B.** Infiniband: hardware and Linux support, *Otkrytye sistemy. SUBD,* 2008, no. 7, available at: https://www.osp.ru/os/2008/07/5478314/ (in Russian).
10. **Mellanox** OFED for Linux Release Notes, available at: http://www.mellanox.com/related-docs/prod_software/Mellanox_OFED_Linux_Release_Notes_4_5-1_0_1_0.pdf
11. **Papadopoulou N., Oden L., Balaji P.** A performance study of UCX over InfiniBand*, Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing,* IEEE Press, 2017, pp. 345—354.
12. **Unified** Communication X, available at: http://www.openucx.org
13. **Shamis P., Venkata M. G., Lopez M. G.** et al. UCX: an open source framework for HPC network APIs and beyond, *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, IEEE, 2015, pp. 40—43.
14. **Mellanox** HPC-X™ Software Toolkit User Manual, available at: http://www.mellanox.com/related-docs/prod_acceleration_software/HPC-X_Toolkit_User_Manual_v2.3.pdf
15. **Shamis P., Venkata M. G., Poole S., Welch A., Curtis T.** Designing a high performance openshmem implementation using universal common communication substrate as a communication middleware, *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools. OpenSHMEM 2014,* Lecture Notes in Computer Science, vol 8356, Springer, Cham, 2014, pp. 1—13.
16. **Grun P., Hefty S., Sur S., Goodell D., Russell R., Pritchard H., Squyres J.** A brief introduction to the OpenFabrics interfaces — a new network API for maximizing high performance application efficiency, *2015 IEEE 23rd Annual Symposium on High—Performance Interconnects,* IEEE, 2015, pp. 34—39.
17. **Atanassov E., Barth M., Byckling M.** et al. Best Practive Guide Intel Xeon Phi v2.0. 2017, available at: http://www.prace-ri.eu/IMG/pdf/Best-Practice-Guide-Intel-Xeon-Phi-1.pdf
18. **Xpmem**, available at: https://github.com/hjelmn/xpmem
19. **MPICH** User's Guide, available at: https://www.mpich.org/static/downloads/3.2.1/mpich-3.2.1-userguide.pdf
20. **Fujita H., Cao C., Sur S., Archer C., Paulson E., Garzaran M.** Efficient Implementation of MPI-3 RMA over OpenFabrics Interfaces, *Parallel Computing*, 2019, vol. 87, pp. 1—10.
21. **Open MPI:** Version 4.0, available at: https://www.open-mpi.org/software/
22. **Squyres J., Bosilca G., Gabriel E., Ladd J.** Open MPI State of the Union XI Community Meeting SC18, 2018, available at: https://www.open-mpi.org/papers/sc-2018/Open-MPI-SC18-BOF.pdf
23. **MVAPICH:** MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE, available at: http://mvapich.cse.ohio-state.edu/
24. **Panda D. K.** Overview of the MVAPICH Project: Latest Status and Future Roadmap, MVAPICH2 User Group (MUG) Meeting, 2018, available at: http://mug.mvapich.cse.ohio-state.edu/static/media/mug/presentations/18/panda-mug-18.pdf
25. **Introduction** to Intel® Software for Parallel Programming, available at: http://school-2010.hpc-russia.ru/files/lectures/lecture_080710_sergeev.pdf
26. **James T**. Intel MPI Library Release Notes, available at: https://software.intel.com/en-us/articles/intel-mpi-library-release-notes
27. **Gladkov D**. Intel MPI Library 2019 over libfabric (2018), available at: https://software.intel.com/en-us/articles/intel-mpi-library-2019-over-libfabric
28. **Müller M. S., Van Waveren M., Lieberman R., Whitney B., Saito H., Kumaran K., Baron J., Brantley W. C., Parrott C., Elken T., Feng H., Ponder C.** SPEC MPI2007—an application benchmark suite for parallel systems using MPI, *Concurrency and Computation: Practice and Experience*, 2010, vol. 22, no. 2, pp. 191—205.
29. **New** SPEC MPI2007 V2.0 benchmark adds data suite for systems with up to 2048 cores, available at: https://www.spec.org/mpi2007/press/v20release.html
30. **MVAPICH:** MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. Benchmarks, available at: http://mvapich.cse.ohio-state.edu/benchmarks/
31. **Gergana S.** Introducing Intel MPI Benchmarks (2018) available at: https://software.intel.com/en-us/articles/intel-mpi-benchmarks
32. **Agarkov A. A., Ismagilov T. F., Makagon D. V., Semenov A. S., Simonov A. S.** Results of assessment testing of the domestic high-speed communication Angara network, *Superkomp'yuternye dni v Rossii: Trudy mezhdunarodnoy konferentsii,* 26—27 September 2016, Moscow, MGU, 2016, pp. 626—639 (in Russian).
33. **Panda D. K., Subramoni H.** InfiniBand, Omni-Path, and High—Speed Ethernet: Advanced Features, *Challenges in Designing HEC Systems, and Usage, IT4 Innovations'18* (2018), available at: http://prace.it4i.cz/sites/prace.it4i.cz/files/files/iohea-01-2018-slides.pdf
34. **Zhumatiy S. A., Stefanov K. S.** *The System Administration of supercomputers,* Moscow, Maks-press, 2018, 448 p. (in Russian).
35. **Erwin J., Mascarenhas E.** Intel Omni-Path Architecture Performance Optimization, *Intel Extreme Performance User Group (IXPUG) Conference,* September 25-28, 2018, Hillsboro (USA), available at: https://www.ixpug.org/resources/intel-omni-path-architecture-performance-optimization
36. **Squyres J., Bosilca G., Hursey J., Gouaillardet G., Bernholdt D.** Open MPI State of the Union XI Community, 2017, available at: https://www.open-mpi.org/papers/sc-2017/Open-MPI-SC17-BOF.pdf
37. **Hashmi J.** Designing Shared Address Space MPI libraries in the Many-core Era, 2018, available at: http://mvapich.cse.ohio-state.edu/static/media/talks/slide/Jahanzeb_sc18_booth.pdf
38. **UL HPC** MPI Tutorial: Building and Runnning OSU Micro-Benchmarks, available at: https://ulhpc-tutorials.readthedocs.io/en/latest/parallel/mpi/OSU_MicroBenchmarks/ (accessed: 20.06.19).
39. **Kaur G., Kumar M., Bala M.** Comparing Ethernet and Soft RoCE for MPI Communication, *IOSR Journal of Computer Engineering (IOSR—JCE),* 2014, vol. 16, issue 4, ver. I, pp. 52—58.
40. **Eijkhout V.** Performance of MPI sends of non-contiguous data. arXiv preprint arXiv:1809.10778. 2018.
41. **Denis A., Trahay F.** MPI overlap: Benchmark and analysis. *Parallel Processing (ICPP), 2016, 45th International Conference on*, IEEE, 2016, pp. 258—267.
42. **Sur S.** Status of OpenFabrics over Verbs based Fabrics, 6th Annual MVAPICH User Group (MUG) 2018, available at: http://mug.mvapich.cse.ohio-state.edu/static/media/mug/presentations/18/sur-mug-18.pdf
43. **Vienne J.** All the things you need to know about Intel MPI Library (2016), https://www.intel.com/content/dam/www/public/us/en/documents/presentation/things-mpi-library.pdf

УДК 004.2+004.8 DOI: 10.17587/prin.11.21-25

В. В. Корнеев, д-р техн. наук, проф., гл. науч. сотр., korv@rdi-kvant.ru,
ФГУП "Научно-исследовательский институт "Квант", Москва

Направления повышения производительности нейросетевых вычислений

Рассмотрены подходы к повышению производительности нейросетевых вычислений. Обосновано, что основным путем повышения производительности является разработка новых эффективных алгоритмов нейросетевых вычислений.

Ключевые слова: *парадигмы нейросетевых вычислений, бинарные нейросети, тензоризованные нейросети, импульсные мозгоподобные нейросети*

Введение

При решении ряда практически важных задач, например, в области обработки речи, изображений и при анализе информации на естественных языках, практика использования нейросетей, в которых применяется машинное обучение, показывает результаты, добиться которых невозможно с помощью традиционных вычислительных методов. С 2012 по 2018 г. производительность вычислительных средств, используемых для обучения нейросетей, выросла в 300 000 раз, что соответствует росту в 2 раза каждые 3,5 месяца [1]. Тенденция к возрастанию сложности используемых нейросетей, а следовательно, к повышению необходимой для работы с ними производительности вычислительных средств сохраняется.

Как показывает опыт, повышение производительности вычислительных средств достигается во взаимодействии процессов развития элементной базы, совершенствования архитектуры и алгоритмов вычислений. В настоящей работе предпринята попытка анализа путей построения высокопроизводительных нейросетевых вычислителей, которые создаются на современной микроэлектронной базе.

Аппаратные средства для реализации традиционных нейросетевых парадигм

Традиционные нейросетевые парадигмы требуют выполнения большого числа вычислений с плавающей точкой как на стадии обучения нейросети, так и при ее применении для получения результатов решения прикладных задач — выводов нейросети. Существенный прогресс в использовании нейросетей наметился с применением графических процессоров GPGPU как массово-параллельных вычислителей. Однако время обучения нейросетей часто составляет неприемлемо длительный для практического применения срок даже для кластерных вычислителей с GPGPU в качестве сопроцессоров у универсальных процессоров вычислительных узлов.

Для преодоления этого затруднения калифорнийская инновационная фирма "Cerebras Systems" представила нейропроцессор Cerebras — Cerebras Wafer Scale Engine (WSE), ориентированный на выполнение алгоритмов машинного обучения и получение выводов нейросетей [1]. Кристалл имеет площадь 46 225 мм^2 со сторонами 21,5 см и производится на кремниевой пластине диаметром 300 мм в одном экземпляре. Для сравнения, наибольший кристалл графического процессора NVIDIA имеет площадь 815 мм^2, что в 56,7 раза меньше Cerebras при том, что при производстве NVIDIA с пластины получается около 30 годных кристаллов. Кристалл WSE содержит 1,2 трлн транзисторов, использованных для построения 400 000 вычислителей, специализированных на работу с разреженными матрицами (56 графических процессоров NVIDIA GV 100 в совокупности имеют более 300 000 ядер), 18 Гбайт распределенной по вычислителям одноуровневой локальной памяти с суммарной пропускной способностью 9 Пбайт/с и коммуникационной средой Swarm с топологией 2D с суммарной пропускной способностью 100 Пбит/с. Однако следует отметить, что потребляемая нейропроцессором Cerebras мощность близка к 15 кВт, что создает проблемы с подводом энергии и отводом теплоты.

Утверждается, что при обучении нейросетей вместо месяцев требуются минуты, а вместо недель — секунды [1]. Однако такая производительность достигается грубым увеличением используемого для вычислений объема оборудования при существенных затратах энергии. Как следствие, актуальной является задача достижения сравнимых с перечисленными выше результатов обучения и получения выводов нейросетей на вычислителях с более низкой производительностью и существенно меньшим энергопотреблением (на настольных компьютерах и на автономных мобильных устройствах).

Подходы к увеличению производительности и энергоэффективности традиционных нейросетевых парадигм

Тензоризованные нейронные сети. Так как вычисления, которые выполняются при обучении и получении выводов нейросетей, могут быть представлены операциями над матрицами весов и векторами

входов и выходов, то они могут быть выполнены в формате тензорного поезда (*Tensor-Train*, TT) [2].

Представление матрицы $\mathbf{W}(i, j)$ в TT-формате имеет вид

$$\mathbf{W}(i,\ j) = \mathbf{W}\left((i_1,\ i_2,\ ...,\ i_d);\ (j_1,\ j_2,\ ...,\ j_d)\right) =$$
$$= \mathbf{G}_1[i_1,\ j_1]\mathbf{G}_2[i_2,\ j_2]\ ...\ \mathbf{G}_d[i_d,\ j_d].$$

Здесь матрицы $\mathbf{G}_p[i_p, j_p]$ имеют одинаковый размер $r_{p-1}\times r_p$, $p = 1, ..., d$; $n_0 = n_d = 1$. Последовательность $\{r_k\}_{k=0}^{d}$ называется рангом TT, а $r = \max_{k\ =\ 0,\ ...,\ d} r_k$ — максимальным рангом. В работе [2] показано, что может быть найдена тензорная аппроксимация матриц с наименьшим значением r. Набор матриц $\left(\mathbf{G}_k[j_k]\right)_{jk=1}^{n_k}$ соответствует измерению k, что приводит к необходимости хранить только $\sum_{k=1}^{d} n_k r_{k-1} r_k$ чисел вместо $\prod_{k=1}^{d} n_k$, требуемого для хранения исходной плотной матрицы.

Формат TT позволяет существенно сократить объем памяти для хранения тензора, по сравнению с объемом, необходимым для плотных матриц, а также эффективно выполнять операции над тензорами, представленными в этом формате. Методы, предложенные в работах [3—5], существенно уменьшают число весов в сверточных нейронных сетях без значительной потери в качестве работы тензоризованной нейронной сети по сравнению с исходной сверточной глубокой нейросетью. Так, в работах [4, 5] показано, что для набора изображений CIFAR-10 [6] объем памяти сверточной нейронной сети может быть уменьшен в 80 раз с потерей 1,1 % качества работы.

Наборы матриц $\left(\mathbf{G}_k[j_k]\right)_{j_k=1}^{n_k}$, $k = 1, ..., d$ могут быть распределены по процессорам многопроцессорной системы, что обусловит эффективность параллельных вычислений. Отметим, что при этом путем выбора соответствующего значения d возможно менять объем вычислений, проводимых каждым процессором. Таким образом, получается некоторая новая схема идеального распараллеливания нейросетевых вычислений.

Бинарные нейросети. Ранее исследователями предпринимались многочисленные попытки перейти к целочисленным малоразрядным представлениям весов, входных и выходных переменных нейронов, а также к отказу от операции умножения путем ее замены, например, сдвигами. К настоящему времени в результате этих исследований сформировался ряд подходов к решению задачи снижения объема используемого оборудования и затрат энергии при применении нейросетей.

Одним из перспективных подходов служит следующий. Исходная глубокая нейронная сеть может быть подвергнута процедурам бинеаризации весов и дообучения в целях получения бинарной сети. Бинарные сети BNN, предложенные в работе [7], позволяют резко сократить объем памяти хранения весов нейронов и исключить умножение при вычислении функции состояния нейрона. Наряду с сетями BNN предложены также бинарные сети XNOR-net [8] и DoReFa-net [9]. В работе [10] показано, что посредством предложенного авторами метода (Our Method) возможно сжатие хранимых данных в 23,6 раза при достижении точности, сравнимой с традиционными нейросетевыми парадигмами. В таблице приведены результаты сравнения по требуемым объемам памяти и получаемой точности (top5 и top1 — попадание истинного значения в 5 лучших и истинное значение, соответственно) на наборе данных [11] нейросетей AlexNet, BNN [7], XNOR [8], DoReFa-net [9], Our Method [10].

Представленные результаты по продвижению новых нейросетевых парадигм без операции умножения и с ограниченным множеством значений весов входов нейронов позволяют надеяться на создание высокоэффективных специализированных нейрокомпьютеров.

СБИС Celerity. В рамках проекта DARPA CRAFT была разработана проблемно-ориентированная на создание автономных систем обработки изображений СБИС Celerity [12]. Она содержит три подсистемы:

- подсистему универсальных процессоров из пяти 64-разрядных RISC-V ядер под управлением ОС Linux;
- специализированную подсистему, программно настраиваемую на реализацию бинарной нейросети BNN;
- 496-процессорную систему из 32-разрядных процессоров RISC-V Vanilla-5, образующих массивно-параллельную подсистему.

Все подсистемы имеют доступ к разделяемой распределенной памяти с 32-разрядным адресным пространством через накристальную коммуникационную сеть 496-процессорной подсистемы с топологией решетки 32×16. Первые девять разрядов адреса памяти определяют координаты блока памяти $x = 0, ..., 31$ и $y = 0, ..., 15$ массивно-параллельной подсистемы. Остальные 22 разряда адресуют ячейки памяти каждого блока распределенной памяти. Адресное пространство $x = 31$ и $y = 0, ..., 15$ используется для создания 4 RoCC-интерфейсов с подсистемой универсальных процессоров и 4 RoCC-интерфейсов со специализированной подсистемой, реализующей бинарную нейросеть BNN.

Пример нейросети [12], реализуемой в СБИС Celerity для обработки изображений, включает шесть

Сравнение нейросетей по объему требуемой памяти и точности на наборе данных [11]

Нейросеть	Объем памяти, Мбайт	Сжатие, раз	Точность, % (top5/top1)
AlexNet	232	1	80,2/56,6
BNN	26,2	10,3	50,4/27,9
XNOR	26,2	10,3	69,2/44,2
DoReFa-net	26,2	10,3	—/49,8
Our Method	1,23	23,6	75,6/51,4

сверточных (*convolutional*) слоев, три сжимающих (*max-pooling*) слоя и три полносвязных (*dense-fully connected*) слоя. Исходное изображение представляется совокупностью 20-битных целочисленных значений, подаваемых на вход первого сверточного слоя. Все остальные слои имеют бинарные значения весов и входов/выходов. Применяемая в нейросети процедура бинаризации позволила получить на наборе данных CIFAR-10 [6] точность 89,8 %. Производительность нейросети, которой удалось добиться, составляет 60 выводов в секунду, чего достаточно для функционирования в реальном времени блока обработки изображений автономных мобильных систем.

Загрузка весов нейросети из внешней памяти не позволяет получить требуемую производительность и энергоэффективность. По этой причине веса предварительно размещаются в памяти массивно-параллельной подсистемы, в процессорах которой выполняются программы, обеспечивающие заданный поток весов в специализированную нейросетевую подсистему.

По энергоэффективности нейросеть BNN на наборе данных CIFAR-10 превосходит более чем в 100 раз графический процессор Nvidia Tesla K40 [12].

Мозгоподобные (neuromorphic) нейросети

Альтернативой традиционным нейросетевым парадигмам служат импульсные мозгоподобные (*neuromorphic*) нейросети, в которых передаваемые сигналы между нейронами могут быть представлены одним или последовательностью импульсов. Эта парадигма используется при построении энергоэффективных реализаций нейросетевых вычислителей. Можно указать по крайней мере на две таких реализации: IBM TrueNorth [13, 14, 16] и Intel Loihi [15, 16].

В нейросети IBM TrueNorth нейроны имеют выход 0/1 (нет импульса/есть импульс) и четыре фиксированных значения, которые могут принимать веса входов нейрона [13]. Соответственно, когда на вход нейрона поступает 1, вес этого входа суммируется с весами других входов, на которые тоже поступила 1. Тем самым исключается энергетически и аппаратно затратная операция умножения. Кристалл, на котором размещаются 1024 нейрона, функционирующий на уровне одного нейрона на частоте около 1 ГГц, при выполнении вычислений во всех нейронах кристалла в течение 1 мс (тактовая частота 1 кГц) потребляет 65 мВт.

В работе [14] для задач, описанных в работах [6, 17—21], представлены данные по наилучшему достигнутому результату решения любыми методами и с использованием нейросети IBM TrueNorth с произвольным числом кристаллов, а также однокристальной нейросети IBM TrueNorth. Показано, что нейросети IBM TrueNorth незначительно уступают по точности лучшим из результатов, которых на настоящее время удалось добиться. Так, при обучении и тестировании на наборе изображений CIFAR10 [6] опубликованный наилучший результат составляет 91,73 %, многокристальная нейросеть IBM TrueNorth достигает результата 89,32 %, а однокристальная — результата 83,41 %. При обучении и тестировании образцов речи TIMIT [20] соответствующие значения показателей точности составили 83,30, 81,96 и 79,16 %.

Заключение

Построение нейросетевых вычислителей для традиционной нейросетевой парадигмы путем увеличения числа вычислительных ядер ведет к повышению потребляемой энергии и проблемам теплоотвода. Корнем проблемы является необходимость выполнения операции умножения над многоразрядными операндами с плавающей точкой.

Возможными путями преодоления этого негативного обстоятельства является уменьшение числа выполняемых операций за счет использования тензоризованных нейронных сетей или переход к бинарным нейросетям и сетям с существенно малоразрядными весами и входами. Первые полученные результаты позволяют надеяться на перспективность дальнейших исследований по повышению достигаемой точности нейросетевых вычислений.

Новая нейросетевая парадигма мозгоподобных (*neuromorphic*) импульсных нейросетей также не использует операцию умножения над многоразрядными операндами с плавающей точкой и энергозатратные передачи многоразрядных данных, заменяя их передачей одиночных импульсов.

Таким образом, основным путем повышения производительности нейросетевых вычислений является разработка новых алгоритмов функционирования нейросетей при обучении и выработке выводов.

Архитектура нейросетевых вычислителей должна обеспечивать быстрый доступ множества вычислительных ядер к локальным блокам распределенной одноуровневой памяти, что характерно для высокоэффективных параллельных систем и не является спецификой нейросетевых вычислений.

Список литературы

1. **Moore S. K.** 6 Things to Know About the Biggest Chip Ever Built // IEEE Spectrum. 21 Aug. 2019. URL: https://spectrum.ieee.org/tech-talk/semiconductors/processors/4-things-to-know-about-the-biggest-chip-ever-built

2. **Oseledets I. V.** Tensor-Train Decomposition // SIAM J. Scientific Computing. 2011. Vol. 33, N. 5. P. 2295—2317.

3. **Novikov A., Podoprikhin D., Osokin A., Vetrov D. P.** Tensorizing neural networks // Advances in Neural Information Processing Systems. 2015. P. 442—450.

4. **Гарипов Т. И.** Применение тензорного разложения для сжатия сверточных нейронных сетей // Сб. тезисов XXIV Междунар. науч. конф. студентов, аспирантов и молодых ученых "Ломоносов-2017". Секция "Вычислительная математика и кибернетика". Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М. В. Ломоносова, 2017. С. 10—11.

5. **Гарипов Т. И.** Тензоризованные нейронные сети. Выпускная квалификационная работа. МГУ, 2017. URL: http://www.machinelearning.ru/wiki/images/2/22/2017_417_GaripovTI.pdf

6. **Krizhevsky A.** Learning multiple layers of features from tiny images. Master's thesis. Computer Science Department, University of Toronto, 2009. 60 p.

7. **Courbariaux M., Hubara I., Soudry D.** et al. Binarized Neural Networks: Training Neural Networks with Weights and

Activations Constrained to +1 or −1. arXiv: 1602.02830 v3[cs. LG].17 Mar 2016.
8. **Rastegari M., Ordonez V., Redmon J., Farhadi A.** Xnor-net: Imagenet classication using binary convolutional neural networks. arXiv preprint arXiv:1603.05279. 2016.
9. **Zhou S., Ni Z., Zhou X.** et al. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160. 2016.
10. **Wei Tang, Gang Hua, Liang Wang.** How to Train a Compact Binary Neural Network with High Accuracy? // Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17), 2017. P. 2625—2631.
11. **Russakovsky O., Deng J., Su H.** et al. Imagenet large scale visual recognition challenge // International Journal of Computer Vision. 2015. Vol. 115, N. 3. P. 211—252.
12. **Davidson S., Xie S., Torng C.** et al. The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips // IEEE Micro. 2018. Vol. 38, N. 2. P. 30—41.
13. **Akopyan F., Sawada J., Cassidy A.** et al. TrueNorth: Design and Tool Flow of a 65 mW1 Million Neuron Programmable Neurosynaptic Chip // IEEE transactions on computer-aided design of integrated circuits and systems. 2015. Vol. 34, N. 10. P. 1537—1557.
14. **Esser S. K., Merolla P. A., Arthur J. V.** et al. Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing. arXiv: 1603.08270 v2 [cs.NE]. 24 May 2016.
15. **Davies M., Srinivasa N., Lin T.-H.** et al. Loihi: a Neuromorphic Manycore Processor with On-Chip Learning // IEEE Micro. 2018. Vol. 38, N. 1. P. 82—99.
16. **Rajendran B., Sebastian A., Schmuker M., Srinivasa N.** Evangelos Eleftheriou. Low-Power Neuromorphic Hardware for Signal Processing Applications. arXiv:1901.03690v2 [cs.ET]. 27 Apr. 2019.
17. **Netzer Y., Wang T., Coates A.** et al. Readingdigits in natural images with unsupervised feature learning // NIPS Workshop onDeep Learning and Unsupervised Feature Learning 2011, URL: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
18. **Stallkamp J., Schlipsing M., Salmen J., Igel C.** Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition // Neural networks. 2012. Vol. 32. P. 323—332.
19. **Romberg S., Pueyo L. G., Lienhart R., Zwol R. V.** Scalable logo recognitionin real-world images // Proceedings of the 1st ACM International Conference onMultimedia Retrieval. ACM, 2011. P. 25.
20. **Garofolo J., Lamel L., Fisher W.** et al. TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1, Philadelphia: Linguistic Data Consortium, 1993. 94 p.
21. **Varga A., Steeneken H. J. M.** Assessment for Automatic Speech Recognition II:NOISEX-92: A Database and an Experiment to Study the Effect of Additive Noise onSpeech Recognition Systems // Speech Communications. 1993. Vol. 12, N. 3. P. 247—251.

Approaches to Improving the Performance of Neural Network Computing

V. V. Korneev, korv@rdi-kvant.ru, Research and Development Institute "Kvant", Moscow, 125438, Russian Federation

Corresponding author:

Korneev Victor V., Principal Researcher, Research and Development Institute "Kvant", Moscow, 125438, Russian Federation
E-mail: korv@rdi-kvant.ru

Received on October 22, 2019
Accepted on November 09, 2019

Approaches to improving the performance of neural network computing are considered. The construction of neural network calculators for the traditional neural network paradigm by increasing the number of computational cores leads to increased energy consumption and heat dissipation problems. The root of the problem is the need to perform a multiplication operation on multi-bit floating-point operands.

A possible way to overcome this negative circumstance is to reduce the number of operations performed through the use of tensorized neural networks or transition to binary neural networks and networks with significantly low-bit weights and inputs.

The new neural network paradigm of neuromorphic neural networks also does not use multiplication operations on multi-bit floating-point operands.

Thus, the main way to improve the performance of neural network computing is the development of new algorithms. The architecture of neural network computers should provide fast access of many computational cores to local blocks of distributed single-level memory, which is typical for high-performance parallel systems, and is not specific to neural network computing.

Keywords: *neural network computing padigms, binary neural networks, tensorizing neural networks, neuromorphic neural networks*

For citation:

Korneev V. V. Approaches to Improving the Performance of Neural Network Computing, *Programmnaya Ingeneria,* 2020, vol. 11, no. 1, pp. 21—25.

DOI: 10.17587/prin.11.21-25

References

1. **Moore S. K.** 6 Things to Know About the Biggest Chip Ever Built, *IEEE Spectrum*, 21 Aug. 2019, available at: https://spectrum.ieee.org/tech-talk/semiconductors/processors/4-things-to-know-about-the-biggest-chip-ever-built.

2. **Oseledets I. V.** Tensor-Train Decomposition, *SIAM J. Scientific Computing,* 2011, vol. 33, no. 5, pp. 2295—2317.

3. **Novikov A., Podoprikhin D., Osokin A., Vetrov D. P.** Tensorizing neural networks, *Advances in Neural Information Processing System,* 2015, pp. 442—450.

4. **Garipov T. I.** Application of tensor decomposition for convolutional neural networks compression, *Sb. tezisov XXIV Mezhdunar. nauch. konferencii studentov, aspirantov i molodyh uchenyh "Lomonosov-2017". Sekciya "Vychislitel'naya matematika i kibernetika"*, Izdatel'skij otdel fakul'teta vychislitel'noj matematiki i kibernetiki MGU im. M. V. Lomonosova, 2017, pp. 10—11 (in Russian).

5. **Garipov T. I.** Tensorized neural networks. Vypusknaya kvalifikacionnaya rabota, MGU, 2017, available at: http://www.machine-learning.ru/wiki/images/2/22/2017_417_GaripovTI.pdf (in Russian).

6. **Krizhevsky A.** Learning multiple layers of features from tiny images. Master's thesis. Computer Science Department, University of Toronto, 2009, 60 p.

7. **Courbariaux M., Hubara I., Soudry D., El-Yaniv R., Bengio Y.** Binarized Neural Networks: Training Neural Networks with Weights and Activations Constrained to + 1 or − 1. arXiv: 1602.02830 v3[cs.LG]. 17 Mar. 2016.

8. **Rastegari M., Ordonez V., Redmon J., Farhadi A.** Xnor-net: Imagenet classication using binary convolutional neural networks. arXiv preprint arXiv:1603.05279. 2016.

9. **Zhou S., Ni Z., Zhou X., Wen H., Wu Y., Zou Y.** Dorefa-net: Training low bitwidth convolutional neural networks with low bit width gradients. arXiv preprint arXiv:1606.06160. 2016.

10. **Wei Tang, Gang Hua, Liang Wang.** How to Train a Compact Binary Neural Network with High Accuracy? *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17),* 2017, pp. 2625—2631.

11. **Russakovsky O., Deng J., Su H., Krause J., Satheesh S., Ma S., Huang Z., Karpathy A., Khosla A., Bernstein M., Berg A. C., Fei-Fei Li.** Imagenet large scale visual recognition challenge*, International Journal of Computer Vision,* 2015, vol. 115, no. 3, pp. 211—252.

12. **Davidson S., Xie S., Torng C., Al-Hawaj K., Rovinski A., Ajayi T., Vega L., Zhao C., Zhao R., Dai S., Amarnath A., Veluri B., Gao P., Rao A., Gai Liu, Gupta R. K., Zhang Z., Dreslinski R. G., Batten C., Taylor M. B.** The Celerity Open-Source 511-Core RISC-V Tiered Accelerator Fabric: Fast Architectures and Design Methodologies for Fast Chips, *IEEE Micro,* 2018, vol. 38, no. 2, pp. 30—41.

13. **Akopyan F., Sawada J., Cassidy A., Alvarez-Icaza R., Arthur J., Merolla P., Imam N., Nakamura Y., Datta P., Nam G.-J., Taba B., Beakes M., Brezzo B., Kuang J. B., Manohar R., Risk W. P., Jackson B., Modha D. S.** TrueNorth: Design and Tool Flow of a 65 mW1 Million Neuron Programmable Neurosynaptic Chip, *IEEE transactions on computer-aided design of integrated circuits and systems*. 2015, vol. 34, no. 10, pp. 1537—1557.

14. **Esser S. K., Merolla P. A., Arthur J. V., Cassidy A. S., Appuswamy R., Andreopoulos A., Berg D. J., McKinstry J. L., Melano T., Barch D. R., di Nolfo C., Datta P., Amir A., Taba B., Flickner M. D., Modha D. S.** Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing. arXiv: 1603.08270 v2 [cs.NE]. 24 May 2016.

15. **Davies M., Srinivasa N., Lin T.-H., Chinya G., Cao Y., Choday S. H., Dimou G., Joshi P., Imam N., Jain S., Liao Y., Lin C.t-K., Lines A., Liu R., Mathaikutty D., McCoy S., Paul A., Tse J., Venkataramanan G., Weng Y.-H., Wild A., Yang Y., Wang H.** Loihi: a Neuromorphic Manycore Processor with On-Chip Learning*, IEEE Micro*, 2018, vol. 38, no. 1, pp. 82—99.

16. **Rajendran B., Sebastian A., Schmuker M., Srinivasa N.** Evangelos Eleftheriou. Low-Power Neuromorphic Hardware for Signal Processing Applications. arXiv:1901.03690v2 [cs.ET]. 27 Apr. 2019.

17. **Netzer Y., Wang T., Coates A., Bissacco A., Wu B., Ng A. Y.** Readingdigits in natural images with unsupervised feature learning, *NIPS Workshop onDeep Learning and Unsupervised Feature Learning 2011*, URL: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

18. **Stallkamp J., Schlipsing M., Salmen J., Igel C.** Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition, *Neural networks,* 2012, vol. 32, pp. 323—332.

19. **Romberg S., Pueyo L. G., Lienhart R., Zwol R. V.** Scalable logo recognitionin real-world images, *Proceedings of the 1st ACM International Conference onMultimedia Retrieval,* ACM, 2011, p. 25.

20. **Garofolo J., Lamel L., Fisher W., Fiscus J., Pallett D., Dahlgren N., Zue V.** TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1, Philadelphia: Linguistic Data Consortium, 1993, 94 p.

21. **Varga A., Steeneken H. J. M.** Assessment for Automatic Speech Recognition II:NOISEX-92: A Database and an Experiment to Study the Effect of Additive Noise onSpeech Recognition Systems, *Speech Communications*, 1993, vol. 12, no. 3, pp. 247—251.

ИНФОРМАЦИЯ

УДК 004.451 DOI: 10.17587/prin.11.26-33

А. Н. Годунов, канд. физ.-мат. наук, зав. отд., nkag@niisi.ras.ru,
В. А. Солдатов, канд. техн. наук, ст. науч. сотр., nkvalera@niisi.ras.ru,
И. И. Хоменков, вед. инженер, nkigor@niisi.ras.ru, Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук" (ФГУ ФНЦ НИИСИ РАН), Москва

Передача сообщений в коммуникационной среде RapidIO для семейства операционных систем реального времени Багет

Для операционных систем реального времени семейства Багет рассмотрены средства передачи сообщений в коммуникационной среде RapidIO. Описаны алгоритм работы драйвера контроллера сообщений RapidIO и его взаимосвязь с операционной системой. Приведено сравнение по быстродействию с интерфейсом MPI в операционной среде LINUX.

Ключевые слова: *ОСРВ Багет, RapidIO, передача сообщений, контроллер сообщений, драйвер, MESSAGE*

Введение

Работы по созданию отечественной микропроцессорной техники в ФГУ ФНЦ НИИСИ РАН ведутся более 30 лет. В 1998 г. был разработан первый в России 32-разрядный RISC-процессор со встроенным сопроцессором плавающей арифметики. С тех пор было разработано свыше 20 различных микропроцессоров архитектуры КОМДИВ32 и КОМДИВ64 [1]. Разработанные электронные изделия используются как во встроенных системах, так и для создания ЭВМ общего и специального назначения. В последние годы появились системы на кристалле со встроенными сетевыми контроллерами и коммутаторами RapidIO и PCI Express (процессоры 1890ВМ6Я, 1890ВМ7Я, 1890ВМ8Я, 1890ВМ9Я), которые можно использовать для построения многопроцессорных систем.

Параллельно с разработкой электронного оборудования в институте также осуществляется и разработка операционных систем жесткого реального времени (ОСРВ), которые могут быть использованы на этом оборудовании. Первое издание ОСРВ Багет 2.0 было выпущено в 2002 г. [2]. С тех пор было разработано еще пять изданий ОСРВ Багет 2.х. В 2004 г. состоялся выпуск первого издания ОСРВ Багет 3.0 [3]. К настоящему времени выпущено еще четыре издания ОСРВ Багет 3.х [4]. В 2019 г. была закончена разработка ОСРВ Багет 4.0.

Все ОСРВ семейства Багет базируются на принципах соответствия международным стандартам и мобильности (под мобильностью понимается возможность выполнения программного обеспечения на различных аппаратных платформах). Во всех ОСРВ семейства Багет поддерживается стандарт POSIX [5]. Однако, если в ОСРВ Багет 2.х реализованы только потоки управления стандарта POSIX, то в последующих системах — процессы этого стандарта и спецификация ARINC 653 [6]. Операционная система Багет 4.0 отличается от остальных ОСРВ семейства Багет поддержкой многоядерных процессоров.

Операционные системы Багет 2.х в основном применяют в управляющих системах с изолированной архитектурой на 32-разрядных процессорах. На более мощных ЭВМ, как правило, используется ОСРВ Багет 3.х. Для работы на многоядерных процессорах предназначена ОСРВ Багет 4.0. При этом существует значительный спектр оборудования, на которое может быть установлена любая из ОСРВ семейства Багет. В настоящее время параллельно поддерживается развитие всех ОСРВ семейства Багет, поэтому значительное количество исходного кода является общим для всех ОСРВ.

С появлением электронных систем с коммутаторами RapidIO возникла необходимость в создании программных средств, обеспечивающих взаимодействие между отдельными узлами системы в коммуникационной среде RapidIO [7, 8]. Для решения этой задачи потребовалось разработать интерфейсы для пользователей и создать универсальный драйвер контроллера сообщений RapidIO. Такой драйвер должен выполняться под управлением всех ОСРВ семейства Багет и призван обслуживать различные интерфейсы взаимодействия между узлами многопроцессорной системы. В разделе статьи, посвященной решению этой задачи, вначале описаны базовые понятия коммуникационной среды RapidIO. Далее перечислены интерфейсы пользователя, которые реализованы в различных ОСРВ семейства Багет для взаимодействия между узлами системы. Основная часть этого раздела посвящена описанию алгоритма драйвера контроллера сообщений, а также приведено сравнение по быстродействию с интерфейсом MPI в операционной среде LINUX.

Коммуникационная среда RapidIO

Коммуникационная среда RapidIO [7, 8] представляет собой сеть с коммутацией пакетов и состоит из узлов, соединенных между собой каналами связи. У каждого узла сети есть один или несколько портов — конечных точек соединительных каналов. Среди узлов сети будем рассматривать:

- оконечное устройство, которое характеризуется уникальным в пределах системы идентификатором RapidIO (*RioID*); каждый процессор описываемого оборудования имеет оконечное устройство;
- коммутатор — устройство, которое всегда имеет более одного порта и выполняет маршрутизацию, направляя пакет, пришедший через один порт в другой, в соответствии с таблицей маршрутизации.

Протокол ввода/вывода RapidIO, поддерживаемый аппаратурой, использует пакеты типа запрос/ответ (*request/response operations*). Отправитель, который является инициатором обмена, посылает получателю пакет-запрос. Получатель в свою очередь отсылает обратно отправителю пакет-подтверждение, который содержит сообщение либо об успешном выполнении (*DONE*), либо о возникновении ошибки в процессе выполнения полученного запроса (*ERROR*). Во всех типах операций, поддерживаемых протоколом RapidIO, направление передачи задается значением *RioID*.

Все операции протокола можно разделить на отдельные группы:

- *MAINTENANCE* — операции, предназначенные для чтения или записи регистров устройств коммуникационной среды, которые выполняются через контроллер ввода/вывода;
- *DOORBELL* — операции передачи сообщений длиной 2 байта через контроллер дверных звонков;
- *MESSAGE* — операции передачи сообщений через контроллер почтовых ящиков или сообщений.

Операции некогерентного ввода/вывода (*NREAD, NWRITE, NWRITE_R, SWRITE*) драйвером сообщений ОСРВ Багет не используются, поэтому они не рассматриваются.

MAINTENANCE. Регистры оконечного устройства локально доступны для процессора, к которому относится это устройство. Для доступа к регистрам остальных *RIO*-устройств, расположенным в сети, необходимо использовать служебные пакеты *MAINTENANCE*. Служебные пакеты отличаются от всех остальных типов пакетов RapidIO, наличием поля *hopCount* (число шагов). Значение, лежащее в этом поле, определяет максимальное число устройств, которые могут быть пройдены пакетом прежде, чем он достигнет цели. Направление к цели, как и для всех остальных типов пакетов, задается значением *RioID*. При приеме служебного пакета проверяется значение поля *hopCount* и, если оно равно 0, это означает, что пакет предназначен данному устройству, в этом случае осуществляется чтение или запись значения заданного регистра устройства. Ответ отправляется в тот порт, откуда пришел служебный пакет. Если значение поля *hopCount* не равно 0, то число уменьшается на единицу и пакет направляется в выходной порт в соответствии с таблицей маршрутизации для заданного *RioID*.

DOORBELL. В каждом оконечном устройстве содержится контроллер дверных звонков, предназначенный для передачи сообщений длиной 2 байта, которые обычно используются для синхронизации взаимодействия между различными оконечными устройствами.

MESSAGE. В каждом оконечном устройстве для передачи данных по коммуникационной среде RapidIO содержится контроллер сообщений или почтовых ящиков. Сообщение представляет собой массив данных длиной от 8 байт до 4 Кбайт. Если длина данных превосходит 4 Кбайта, такое сообщение разбивается на более короткие сообщения (письма). Сообщения, которые помещаются в 256 байт, будем называть короткими. Остальные сообщения, длина которых превосходит 256 байт, при передаче сегментируются. Разбивка большого сообщения на сегменты при отправке и сборка большого сообщения из сегментов при приеме осуществляются аппаратурой. Сегменты передаются по RapidIO специальным типом пакета-запроса — *MESSAGE*. На каждый сегмент должен быть послан ответ, который также передается специальным типом пакета, отличным от пакета ответа на другие типы запросов, — *MESSAGE RESPONSE*.

В атрибутах передаваемого сообщения указываются номер почтового ящика и порядковый номер письма. Прием сообщений может осуществляться одновременно в девять очередей, одна из которых является общей, а остальные — выделенными. Выбор очереди осуществляется контроллером по номеру почтового ящика и *RioID* источника (выделенная очередь), если прием нельзя осуществить ни в одну из восьми выделенных очередей, то прием осуществляется в общую очередь. Номер почтового ящика используется для приема сообщений в различные выделенные очереди.

Программное обеспечение

Для обмена данными между процессорами по коммуникационной среде RapidIO все ОСРВ семейства Багет предоставляют пользователям следующие возможности:

- каналы передачи данных с очередью сообщений и без очереди сообщений [9];
- передачу сообщений по шине RapidIO с собственным интерфейсом (библиотека mp);
- эмуляцию сети Ethernet на шине RapidIO;
- передачу коротких сообщений для мониторинга RapidIO;
- передачу больших массивов данных память-память.

Каналы передачи данных, реализованные в соответствии со спецификацией ARINC 653 [6], являются универсальным средством, которым можно пользоваться как в ARINC-процессах, так и в POSIX-процессах. Каналы должны быть предварительно описаны при конфигурировании системы. Если один из участников обмена данными по каналам будет

перезапущен, то это может вызвать лишь потерю некоторого числа сообщений между ним и другими участниками обмена. В операционных системах Багет 2.x использование каналов возможно только для межпроцессорного обмена.

Средства передачи сообщений библиотеки mp используют интерфейс, разработанный специально для этой библиотеки, и отличаются высокой скоростью передачи данных за счет отсутствия копирования пересылаемых данных в памяти. Функции библиотеки mp можно использовать только после их инициализации. При инициализации указываются *RioID* всех оконечных устройств, которые будут обмениваться сообщениями. Передача сообщений проводится через порты. Каждый порт может либо передавать сообщения, либо принимать их. Порт-отправитель может отправлять сообщения только одному порту-получателю, а порт-получатель может получать сообщения только от одного порта-отправителя. Таким образом, взаимодействующие порты образуют пары (порт-отправитель, порт-получатель). Отправляемые и получаемые сообщения размещаются в буферах, каждый из которых идентифицируется соответствующим описателем. Каждый порт имеет свой набор буферов. После создания порта прикладная программа должна выделить память под буферы, описатели буферов и связать их между собой.

Передача сообщений прикладной программой проводится за четыре шага:

- вначале с помощью соответствующей функции надо получить указатель на описатель буфера для отправки сообщения;
- далее следует получить адрес буфера, связанного с этим описателем;
- после переноса сообщения в буфер надо вызвать функцию синхронизации кеш-памяти и оперативной памяти;
- на последнем шаге следует вызвать функцию, которая установит сообщение в очередь на отправку.

Прием сообщений прикладной программой также проводится за четыре шага:

- вначале надо получить указатель на описатель буфера, содержащего принятое сообщение;
- далее следует получить адрес буфера, связанного с этим описателем;
- после надо вызвать функцию синхронизации оперативной памяти и кеш-памяти и обработать лежащее в буфере сообщение;
- на последнем шаге описатель буфера (вместе с буфером) следует вернуть операционной системе, в результате чего указанный буфер снова может быть использован для приема сообщений.

Использование библиотеки mp возможно только в POSIX-процессах. Для операционных систем Багет 2.x данный интерфейс пока не реализован.

Эмуляция сети Ethernet в коммуникационной среде RapidIO позволяет использовать интерфейс сокетов семейства протоколов TCP/IP, применяемого для передачи данных по сети. Реализация выполнена на основе коротких сообщений RapidIO. Таким образом, MTU (*Maximum Transmission Unit*) такой сети равно 256 (размер короткого пакета RapidIO). Напомним, что параметр MTU определяет максимальный размер пакета, который может быть передан по сети без фрагментации. Реализовано для всех ОСРВ семейства Багет.

Передача коротких сообщений позволяет пользовательской программе с помощью вызова соответствующей функции передать любому оконечному устройству сообщение длиной не более 256 байт. На принимающем устройстве передаваемое сообщение будет принято функцией, которую задаст пользователь. Если пользователь не определил функцию приема сообщений, то текст принятого сообщения будет выведен на консоль. Функция приема сообщений может выполняться только в привилегированном POSIX-процессе.

Передача больших массивов данных позволяет с помощью вызова соответствующей функции передать большой массив информации с одного оконечного узла на другой или другие. Это средство разработано специально для загрузчика, но может вызываться и из потоков, выполняемых в привилегированном POSIX-процессе.

Драйвер контроллера сообщений

Драйвер контроллера сообщений обеспечивает взаимодействие операционной системы с аппаратурой и является нижним уровнем поддержки средств передачи сообщений, которые были перечислены выше. Взаимодействие драйвера с операционной системой осуществляется с помощью функций драйвера и обработчиков событий (callback-функций).

Функции для связи с драйвером:

- *boardRioSendMsg* — запрос к драйверу на передачу сообщения;
- *boardRioSend* — передача массива данных память-память, может вызываться только из системного привилегированного процесса; используется загрузчиком при передаче образа с одного оконечного устройства на другие;
- *boardRioAlert* — передача короткого сообщения, которая выполняется в контексте вызвавшего ее потока, прием таких сообщений на принимающем устройстве осуществляется обработчиком событий, установленным функцией *boardRioSetRxHook* для коротких сообщений.

Для установки обработчиков событий драйвера предназначены следующие функции:

- *boardRioSetRxHook* — обработчик (*RxHook*), установленный этой функцией, вызывается драйвером после получения сообщения;
- *boardRioSetTxHook* — обработчик (*TxHook*) вызывается драйвером после отправки сообщения;
- *boardRioSetGetbufHook* — обработчик (*GetbufHook*) вызывается драйвером, чтобы получить буфер для приема сообщения;
- *boardRioSetGetmsgHook* — обработчик (*GetmsgHook*) вызывается драйвером для получения текста сообщения на отправку.

Аргументами функций установки обработчиков событий драйвера являются:

- указатель на функцию, которую требуется вызвать при наступлении события;

• тип программного обеспечения, которое устанавливает этот обработчик (*CHAN_CHAN* – каналы, *LIBMP_CHAN* библиотека mp, *NET_CHAN* — эмуляция сети, *ALERT_CHAN* — короткое сообщение).

Драйвер контроллера сообщений обеспечивает поддержку нескольких логических связей между оконечными устройствами. Максимально возможное число логических связей между двумя оконечными устройствами задается при конфигурировании операционной системы. У драйвера имеется массив указателей на описания оконечных устройств, с которыми уже установлена или может быть установлена логическая связь. Описание каждого оконечного устройства заводится динамически по мере необходимости.

При старте драйвера в привилегированном системном процессе проводится инициализация необходимых структур, создаются семафоры для синхронизации доступа к описаниям из разных потоков и запускаются два потока управления: один поток предназначен для передачи сообщений (поток-передатчик), а второй — для приема (поток-приемник). Приоритет потока-приемника выше, чем приоритет потока-передатчика, который, в свою очередь, выше, чем приоритет пользовательского потока, из которого будут вызываться средства передачи сообщений.

Рассмотрим взаимодействие между передатчиком и приемником более подробно. Инициатором операции обмена всегда является передатчик сообщений. Как видно на блок-схеме алгоритма потока-передатчика, представленной на рис. 1, этот поток выполняет циклический алгоритм, который начинается с проверки состояния именованного семафора (*txQueSem*). Этот семафор при старте потока-передатчика находится в закрытом состоянии и может быть освобожден программным обеспечением более высокого уровня при вызове функции драйвера *boardRioSendMsg()* или из потока-приемника, если этот поток получит сообщение *ACK_SYN*, которое уведомляет о том, что оконечное устройство-партнер становится готовым для приема сообщений. Такое сообщение посылается партнерами при начальном старте или в случае, когда ранее партнер послал сообщение об отказе приема, а позже причина, по которой был послан отказ, исчезла.

Каждое оконечное устройство-партнер с точки зрения потока-передатчика имеет свое состояние, которое устанавливается либо потоком-приемником по последнему сообщению, принятому от этого оконечного устройства, либо потоком-передатчиком, если была получена ошибка или отказ в ответ на запрос на передачу сообщения. Поле состояния в описании оконечного устройства-партнера может иметь следующие значения:

• *MS_NONE* — начальное состояние;

• *MS_ACK* — очередь настроена, оконечное устройство готово к приему сообщения в ответ на запрос;

• *MS_ACK_SYN* — оконечное устройство стало готово к приему сообщения;

• *MS_NACK_NOBUF* — у некоторых логических связей устройства нет буферов для приема сообщений;

• *MS_NACK_NOTREADY* — на оконечном устройстве требуемый протокол не инициализирован;

• *MS_NACK_BUSY* — в данный момент идет прием от другого оконечного устройства (нет свободных выделенных очередей);

• *MS_EIO* — были получены фатальные ошибки передачи, например, *time out* при передаче.

Если состояние оконечного устройства-партнера *MS_EIO*, т. е. был сбой, то драйвер может время от

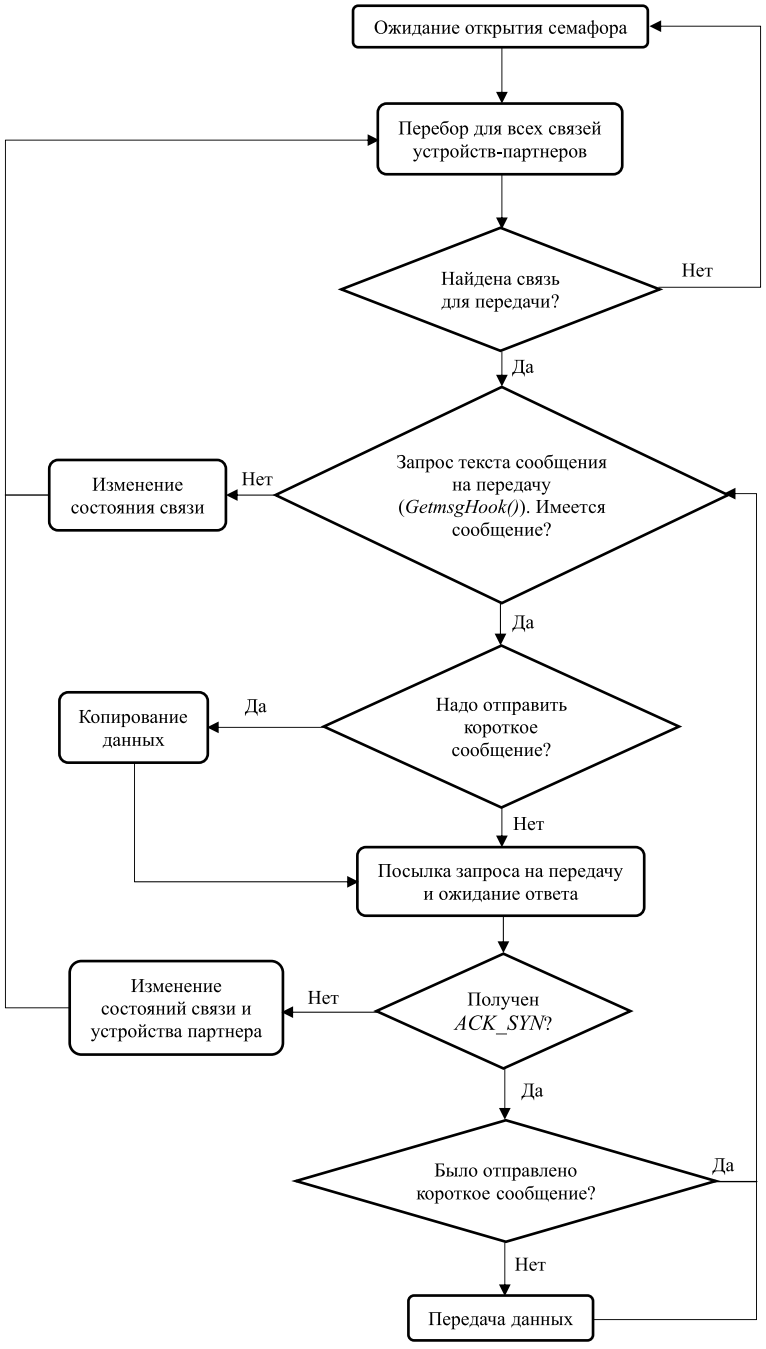

Рис. 1. Блок-схема алгоритма потока передачи сообщений

времени (время задается при конфигурировании) проверять оконечное устройство-партнер на готовность. Для этого с помощью операции *MAINTENANCE* делается попытка чтения пользовательского регистра *Tag* устройства-партнера и, если регистр прочитан и содержит соответствующее значение, то драйвер снимает с оконечного устройства состояние *MS_EIO* и переводит его в состояние *MS_NONE*.

По умолчанию это время равно бесконечности, и состояние *MS_EIO* может снять только само оконечное устройство-партнер, послав короткое сообщение *ACK_SYN*. Такое может произойти, если устройство-партнер было отключено, а затем его заново включили.

Если состояние оконечного устройства *MS_NACK_NOTREADY* или *MS_NACK_BUSY*, то отправка сообщений не проводится до смены состояния устройства-партнера. Смена состояния проводится по приходу короткого сообщения *ACK_SYN* от этого оконечного устройства. При других состояниях оконечного устройства драйвер может отправлять сообщения, используя доступные логические связи с этим оконечным устройством.

В описании каждой логической связи с оконечным устройством также имеется поле состояния, которое может принимать следующие значения:

- *RS_NONE* — начальное состояние связи;
- *RS_REQ_SEND* — был отправлен запрос;
- *RS_ACK_RECEIVED* — был получен *ACK*;
- *RS_NACK_RECEIVED* — был получен *NACK*;
- *RS_DATA_RECEIVED* — были получены данные;
- *RS_EMPTY* — нечего отправлять;
- *RS_NACK_ACK* — после *NACK* был получен *ACK*;
- *RS_REQ_RESTART* — после *REQ* был получен *ACK_SYN*, т. е. партнер был перезагружен.

Такое большое число состояний сделано в первую очередь для более точной диагностики состояния каждой связи, так как у операционной системы имеются информационные команды, позволяющие следить за работой драйвера.

После того как семафор потока-передатчика сообщений будет освобожден, драйвер начинает обход описаний оконечных устройств-партнеров, с которыми связано данное оконечное устройство. Внутри описания каждого оконечного устройства-партнера драйвер последовательно обходит описания всех логических связей, установленных с этим устройством, находит логические связи, по которым требуется передача сообщений, и выполняет передачу этих сообщений. Отправка драйвером одного сообщения осуществляется в соответствии с алгоритмом (см. рис. 1) в последовательности, описанной ниже.

Для получения текста сообщения вызывается callback-функция *GetmsgHook()*. Если сообщение не получено, то состояние найденной логической связи изменяется на *RS_EMPTY* и продолжается перебор связей.

Получив текст сообщения, поток-передатчик посылает запрос на передачу (короткое сообщение *REQ*), в котором указан протокол передачи (каналы, библиотека mp и т. д.) и по какой логической связи нужно будет отправлять сообщение. Также в этот пакет могут быть помещены передаваемые данные (если эти данные целиком помещаются в оставшуюся часть пакета). Для запроса на передачу больших сообщений можно было бы использовать операцию *DOORBELL*, но при таком подходе не было замечено никакого преимущества по сравнению с использованием коротких сообщений.

После отправки запроса на передачу некоторое время ожидается ответ, который в случае отсутствия ошибок приходит практически мгновенно.

Если за заданный диапазон времени ответ не поступил, то в описание оконечного устройства-партнера записывается состояние *MS_EIO* и передача этого сообщения прекращается.

Если пришел ответ, содержащий *NACK*, то это означает, что прием в данный момент не может быть осуществлен. В этом же ответе содержится и причина отказа в приеме сообщения, которая может принимать следующие значения:

- нет буферов для приема сообщений (*MS_NACK_NOBUF*);
- нет свободных выделенных очередей (*MS_NACK_BUSY*);
- на оконечном устройстве-партнере драйвер не настроен для приема данного типа сообщений (*MS_NACK_NOTREADY*).

Полученная причина отказа записывается в описание соответствующей логической связи и в описание оконечного устройства-партнера, причем только последние две причины являются основанием для прекращения посылки сообщений по всем логическим связям устройства-партнера. Обработка передачи сообщения по данной логической связи прекращается.

Если пришел ответ *ACK*, тогда, если сообщение было коротким, оно уже было отправлено; в противном случае поток-передатчик запускает передачу большого сообщения с использованием прямого доступа к памяти (*DMA*).

По окончании передачи очередного сообщения осуществляется поиск следующей логической связи, готовой к передаче. Если такая связь не найдена, то поток-передатчик переходит к началу цикла, а именно — к проверке состояния именованного семафора (*txQueSem*).

Рассмотрим работу приемника сообщений. Как видно на блок-схеме алгоритма потока-приемника, представленной на рис. 2, этот поток выполняет циклические действия, которые начинаются с проверки состояния именованного семафора (*rxSem*). При старте потока-приемника этот семафор находится в закрытом состоянии, но может быть освобожден обработчиком прерываний от аппаратуры. Причин для возникновения прерывания может быть две, а именно окончание приема в выделенную очередь или прием пакета в общую очередь.

Если прерывание произошло по поводу окончания приема в выделенную очередь, то поток-приемник вызывает callback-функцию *RxHook()*, соответствующую типу программного обеспечения, для которого закончился прием сообщения. После этого организуется цикл по всем партнерам, которым ранее было отправлено сообщение *NACK_BUSY*. Каждому такому

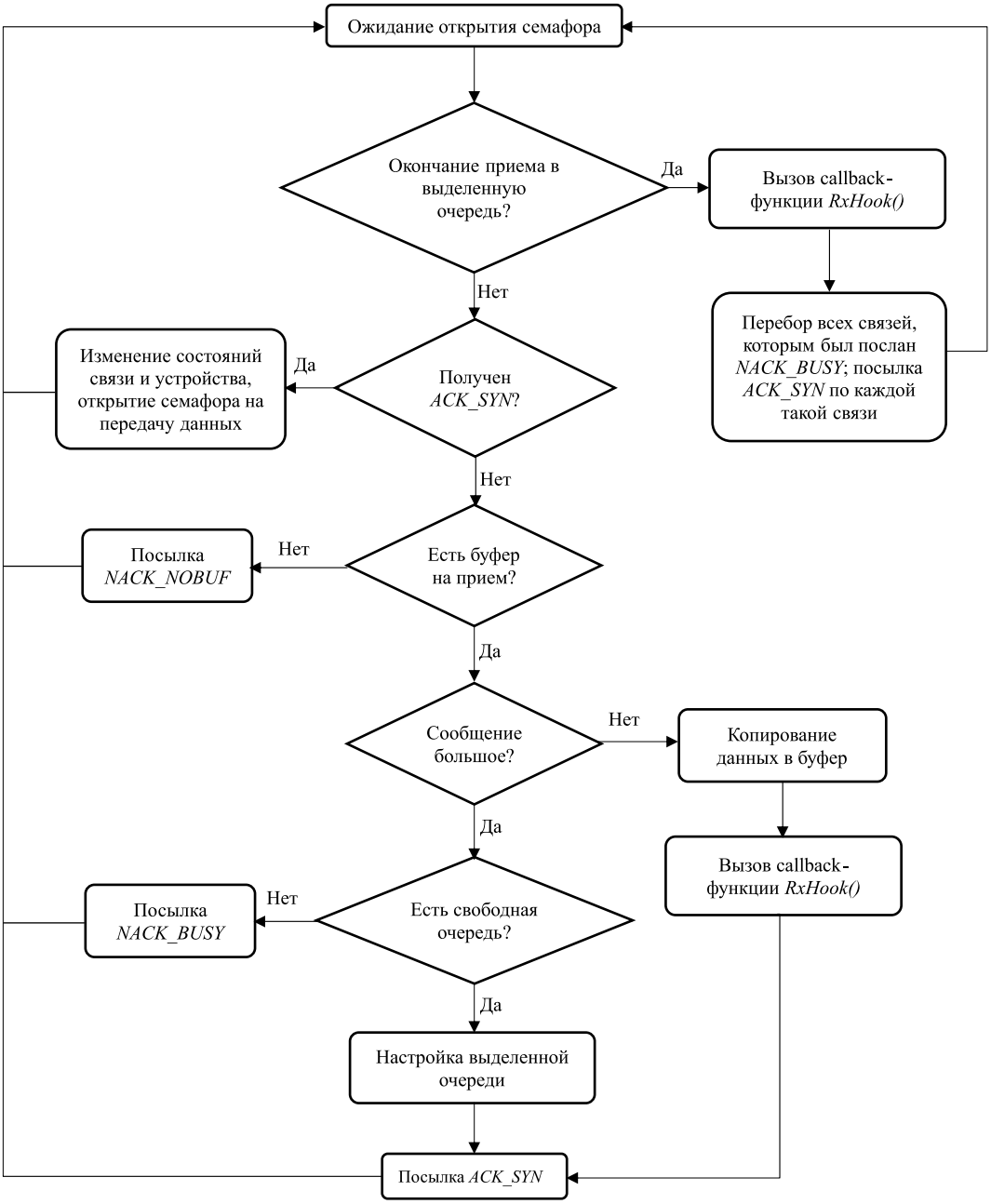

Рис. 2. Блок-схема алгоритма потока приема сообщений

партнеру посылается сообщение *ACK_SYN,* чтобы он мог повторить посылку непринятого ранее сообщения.

Каждый пакет *MESSAGE* содержит заголовок с информацией, определяющий его назначение. Поэтому если прерывание произошло по поводу приема сообщения в общую очередь, поток-приемник проводит его обработку в соответствии с принятым заголовком. Вначале проверяется это сообщение *ACK_SYN* или запрос на прием сообщения. В случае, когда получено сообщение *ACK_SYN*, изменяется состояние партнера и открывается семафор на передачу сообщений по соответствующей логической связи.

Если получен запрос на прием сообщения, тогда вызывается callback-функция *GetbufHook(),* которая возвращает адрес буфера для приема сообщения. Если свободных буферов больше нет, то поток приема сообщений посылает сообщение *NACK_NOBUF.* В случае отсутствия свободного буфера для приема сообщения функция выделения буферов *GetbufHook* должна запомнить устройство-партнер, чтобы в дальнейшем при появлении свободного буфера была возможность вызвать специальную функцию драйвера *boardRioReqAck()*, которая отправит указанному партнеру сообщение *ACK_SYN.*

После получения буфера для приема сообщения проверяется, это запрос на прием большого сообщения или нет. Если сообщение короткое, тогда поток приема сообщений проводит копирование данных

в буфер, вызывает callback-функцию *RxHook()* и посылает сообщение *ACK*. В случае запроса на прием большого сообщения проверяется наличие свободной выделенной очереди для приема сообщения. Если очередь не найдена, то драйвер посылает сообщение *NACK_BUSY*. Если свободная очередь найдена, то драйвер проводит настройку выделенной очереди на прием большого сообщения и посылает сообщение *ACK-SYN*. По окончании обработки прерывания поток приема сообщения переходит к началу цикла, а именно — к проверке состояния именованного семафора (*rxSem*).

Описанный алгоритм работы драйвера обеспечивает возобновление приема/передачи сообщений в случае сбоев оборудования, а также при включении прибора-партнера после его выключения. Этот же алгоритм позволяет реализовать механизм "холодного резервирования", когда к одному коммутатору общего прибора через разные порты подсоединены два аналогичных прибора (основной и резервный). Предполагается, что основной прибор всегда взаимодействует с одним из них. Далее выключается основной (соответственно резервный) прибор и включается резервный (соответственно основной). Программа начальной инициализации коммуникационной среды RapidIO, которая выполняется при старте включенного прибора, корректирует таблицу маршрутизации коммутатора общего прибора, и передача сообщений между приборами восстанавливается.

Тестирование и измерение быстродействия

Для тестирования разработанного драйвера контроллера сообщений были созданы два теста: один из них осуществляет передачу сообщений по каналам с очередью сообщений, а второй — средствами библиотеки mp. Оба теста для заданного числа оконечных устройств выполняют прием и передачу сообщений каждого оконечного устройства со всеми остальными с максимально возможной скоростью.

На установке, состоящей из 20 процессоров, оба теста проработали без сбоев в течение всего запланированного времени (72 ч). Эти же тесты, выполняемые на двух процессорах, использовали для измерения скорости передачи сообщений, которая существенно зависит от размера буферов и их числа. Скорость передачи данных по каналам оказалась примерно в 2 раза меньше, чем скорость передачи данных средствами библиотеки mp. Этот факт объясняется отсутствием копирования сообщений из буферов, с которыми работает драйвер контроллера сообщений. Однако у средств передачи сообщений библиотеки mp нестандартный и более сложный пользовательский интерфейс, пользователю нужно самому осуществлять управление буферами и обеспечивать синхронизацию кеш-памяти и оперативной памяти.

Измеренная скорость передачи данных средствами библиотеки mp для двух буферов размером 2 Мбайта составила 94,75 % от скорости пропускной способности канала RapidRIO. Для сравнения можно привести результат, полученный в работе [11], когда при тестировании передачи сообщений по интерфейсу MPI (*Message Passing Interface*) в операционной среде LINUX была достигнута скорость передачи 84 % от скорости пропускной способности канала RapidRIO.

Заключение

Схема взаимодействия драйвера с программным обеспечением более высокого уровня универсальна, что позволяет обслуживать различные протоколы, реализующие интерфейсы для пользовательских программ, как стандартные (каналы, эмуляция Ethernet), так и специальные (библиотека mp и др.). Передача сообщений по каждому каналу связи независима от других соединений. Обеспечиваются восстановление работоспособности после сбоев и холодное резервирование оборудования.

Проведенные тестирование и измерение быстродействия показали, что драйвер контроллера сообщений коммуникационной среды RapidIO отличается высокой надежностью и быстродействием.

Разработанный интерфейс передачи сообщений (библиотека mp) получил практическое применение в задаче цифровой обработки сигналов [10] на многопроцессорной системе, построенной на линейке отечественного оборудования [1].

Публикация выполнена в рамках государственного задания по проведению фундаментальных научных исследований по теме "Исследование и реализация программной платформы для перспективных многоядерных процессоров" (№ 0065-2019-0002).

Список литературы

1. **Бобков С. Г.** Импортозамещение элементной базы вычислительных систем // Вестник Российской Академии Наук. 2014. Т. 84, № 11. С. 1010—1016.

2. **Безруков В. Л., Годунов А. Н., Назаров П. Е.** и др. Введение в ос2000 // Вопросы кибернетики. М.: НИИСИ РАН, 1999. С. 76—106.

3. **Годунов А. Н.** Операционная система реального времени Багет 3.0 // Программные продукты и системы. 2010. № 4. С. 15—19.

4. **Годунов А. Н., Солдатов В. А.** Операционные системы семейства Багет (сходства, отличия и перспективы) // Программирование. 2014. № 5. С. 68—76.

5. **IEEE** Standard for Information Technology — Portable Operating System Interface (POSIX®), Base Specifications, Issue 7, IEEE Std 1003.1-2017. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8277153&tag=1

6. **Годунов А. Н., Солдатов В. А.** Спецификация ARINC 653 и ее реализация в операционной системе реального времени Багет 3. // Программная инженерия. 2015. № 6. С. 3—17.

7. **RapidIO** Interconnect Specification. Part1: Input/Output Logical Specification. Revision 3.1 — RapidIO Trade Association, 2014. URL: http://www.rapidio.org/files/IO_logical.pdf

8. **RapidIO** Interconnect Specification. Part 2: Message Passing Logical Specification. Revision 3.1 — RapidIO Trade Association, 2014. URL: http://www.rapidio.org/files/msg_logical.pdf

9. **Годунов А. Н., Солдатов В. А., Хоменков И. И.** Реализация каналов спецификации ARINC 653 в операционной системе реального времени 3.0 // Программные продукты и системы. 2017. Т. 30, № 3. С. 392—400.

10. **Райко Г. О., Павловский Ю. А., Мелькановић В. С.** Технология программирования многопроцессорной обработки гидроакустических сигналов на вычислительных устройствах семейства "КОМДИВ" // Гидроакустика. 2014. № 20. С. 85—92.

11. **Кулешов А. С.** Поддержка протокола MPI в ядре ОС Linux для многопроцессорных вычислительных комплексов на базе высокоскоростных каналов RapidIO // Программные продукты и системы. 2015. № 4 (112). С. 93—98.

Message Transfer in the RapidIO Interconnect for Baget Real-Time Operating Systems Family

A. N. Godunov, nkag@niisi.ras.ru, **V. A. Soldatov,** nkvalera@niisi.ras.ru, **Homenkov I. I.,** nkigor@niisi.ras.ru, Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences" (SRISA), Moscow, 117218, Russian Federation

Corresponding author:

Godunov Alexander N., Head of Department, Federal State Institution "Scientific Research Institute for System Analysis of the Russian Academy of Sciences" (SRISA), Moscow, 117218, Russian Federation
E-mail: nkag@niisi.ras.ru

Received on October 25, 2019
Accepted on November 22, 2019

Russian CPU modules of the Baget family, interacting via the RapidIO interconnect, are designed to build up multiprocessor computing and embedded systems. The hard real-time operating systems of the Baget family (RTOS Baget 2.x, RTOS Baget 3.x and RTOS Baget 4.0) support this hardware. Messaging functions available to developers of applications running on RTOS Baget 2.x, 3.x and 4.0 may somewhat vary, but in all RTOS of the Baget family, the same RapidIO device driver is used at the lower level of software for the RapidIO interconnect. The interaction between the driver and RTOS is provided by calling a few driver functions and setting special event handlers (callback functions) to be called by driver if corresponding events occur. The RapidIO device driver allows to restore communication both in case of failures and after the partner is switched off and then switched on again. This allows to implement the "cold standby" feature when one partner-computer is switched off and replaced with another one, so that the link will be restored. Testing has shown high reliability of the developed software. Benchmarks of the data transfer rate showed that overhead costs are less than 6 % of the channel capacity.

Keywords: *RTOS BAGET, RapidIO, message transfer, message controller, driver, MESSAGE*

For citation:

Godunov A. N., Soldatov V. A., Homenkov I. I. Message Transfer in the RapidIO Interconnect for Baget Real-Time Operating Systems Family, *Programmnaya Ingeneria,* 2020, vol. 11, no. 1, pp. 26—33.

DOI: 10.17587/prin.11.26-33

References

1. **Bobkov S. G.** Import Substitution Element Base Computing Systems, *Vestnik Rossijskoj Akademii Nauk,* 2014, vol. 84, no. 11, pp. 1010—1016 (in Russian).

2. **Bezrukov V. L., Godunov A. N., Nazarov P. E., Soldatov V. A., Homenkov I. I.** Introduction to the oc2000, *Voprosy kibernetiki*, Moscow, NIISI RAN, 1999, pp. 76-106 (in Russian).

3. **Godunov A. N.** Real-time operating system Baget 3.0, *Prorammnye producty i sysmemy*, 2010, no. 4, pp. 15—19 (in Russian).

4. **Godunov A. N., Soldatov V. A.** Baget family operating systems (similarities, differences, and prospects), *Programmirovanie,* 2014, no. 5, pp. 68—76 (in Russian).

5. **IEEE** Standard for Information Technology— Portable Operating System Interface (POSIX®), Base Specifications, Issue 7, IEEE Std 1003.1-2017, available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8277153&tag=1

6. **Godunov A. N., Soldatov V. A.** ARINC Specification 653 and its implementation in RTOS Baget 3, *Programmnaya ingeneria,* 2015, no. 6, pp. 3—17 (in Russian).

7. **RapidIO** Interconnect Specification. Part 1: Input/Output Logical Specification. Revision 3.1 — RapidIO Trade Association, 2014, available at: http://www.rapidio.org/files/IO_logical.pdf

8. **RapidIO** Interconnect Specification. Part 2: Message Passing Logical Specification. Revision 3.1 — RapidIO Trade Association, 2014, available at: http://www.rapidio.org/files/msg_logical.pdf

9. **Godunov A. N., Soldatov V. A., Homenkov I. I.** Implementation of ARINC 653 specification channels in real-time operating system 3.0, *Programmnye produkty i sistemy,* 2017, vol. 30, no. 3, pp. 392—400 (in Russian).

10. **Raiko G. O., Pavlovsky Yu. A., Mel'kanovich V. S.** Programming technology for multiprocessor processing of hydroacoustic signals on computing devices of the "KOMDIV" family, *Gidroakustika*, 2014, no. 20 (2), pp. 85—92 (in Russian).

11. **Kuleshov A. S.** Support for MPI protocol in the Linux kernel for multiprocessor computing systems based on high-speed channels RapidIO, *Programmnye produkty i sistemy,* 2015, no. 4 (112), pp. 93—98 (in Russian).

УДК 004.272.44 DOI: 10.17587/prin.11.34-39

О. В. Непомнящий, канд. техн. наук, зав. кафедрой, 2955005@gmail.com,
И. Н. Рыженко, аспирант, rodgi.krs@gmail.com, Сибирский федеральный университет, Красноярск

Метод высокоуровневого синтеза и программный инструментарий для описания алгоритмов функционирования СБИС*

Рассмотрено решение задачи эффективной организации процесса проектирования сложных однокристальных систем для параллельной обработки информации. Данное решение базируется на функционально-потоковом подходе и предложенной модели вычислений с массовым параллелизмом для описания исходного алгоритма. Это обеспечивает архитектурную независимость программного описания, возможность рассмотрения максимального множества решений с учетом имеющихся ограничений и, как следствие, получение оптимального решения из доступного множества. Приведены результаты разработки программного инструментария для технологии высокоуровневого синтеза. Разработанный набор программных инструментов позволяет проводить трансляцию, отладку, оптимизацию и преобразования алгоритмов с функционально-потокового языка параллельного программирования на языки описания аппаратуры. Приведены результаты тестирования и сравнительного анализа разработанных программных инструментальных средств на группе тестовых задач.

Ключевые слова: *СБИС, параллельные вычисления, функциональное программирование, высокоуровневый синтез, синтезатор, HDL, транслятор, компилятор*

Введение

Переход к новым технологическим нормам производства однокристальных систем в условиях неуклонного роста требований к техническим характеристикам и сложности в архитектурной организации обусловливает необходимость пересмотра процесса проектирования СБИС [1]. При традиционном маршруте проектирования модель системы разрабатывается под конкретную архитектуру на основе стандартных или рекомендованных производителем библиотек, используемых для конечного воплощения в целевой кристалл или платформу [2]. Однако наиболее эффективные решения получают при использовании технологий высокоуровневого синтеза [2]. Тем не менее известные подходы, основанные на данной технологии, направлены на устранение семантического разрыва между высокоуровневым и низкоуровневым описаниями систем с использованием универсальных языков программирования или разработкой собственных языковых средств от специализированного ассемблера до высокоуровневого языка параллельного представления, например, COLAMO, SystemC, HLS и т. д. [2—4].

Выявлено, что на современном этапе развития средств поддержки высокоуровневого синтеза СБИС имеется ряд проблемных вопросов в организации процесса проектирования, а именно:

— отсутствуют методы эффективной выработки архитектурных решений для систем параллельной обработки информационных потоков, не зависящих от конечной формы реализации;

— отсутствуют инструментальные средства, обеспечивающие эффективный перенос архитектурно-независимого высокоуровневого описания решаемых прикладных задач на целевую платформу СБИС/ПЛИС;

— за редким исключением языки программирования, применяемые на современном этапе для описания систем параллельной обработки данных, либо предназначены для схемотехнического описания, либо ориентированы на традиционное последовательное программирование.

Метод высокоуровневого синтеза

Авторами предложена модель вычислений и метод архитектурно-независимого высокоуровневого синтеза СБИС [1, 5]. В основе метода лежит принцип представления алгоритмов на функционально-потоковом параллельном языке (ФППЯ) с параллелизмом на уровне операций и отсутствием явного управления вычислениями. В существующих методах высокоуровневого синтеза применяют изначально последовательные языки с явным управ-

* Работы выполнены при финансовой поддержке РФФИ в рамках научного проекта № 17-07-00288.

лением вычислениями с дальнейшим ручным или автоматическим выделением параллелизма — метод выделения параллелизма из последовательного описания алгоритма. В предлагаемом методе описание изначально формируется с максимальным параллелизмом на ФППЯ. При переходе на целевую платформу с различными ресурсами и различными вариантами параллелизма применяется сокращение/преобразование параллелизма с учетом налагаемых ограничений. Такой подход позволяет обеспечить архитектурную независимость высокоуровневого описания от ресурсных ограничений конкретной платформы СБИС/ПЛИС. Аналогичный подход применяется в некоторых методах программирования параллельных вычислительных систем. В работах [6, 7] показаны эффективность и переносимость/архитектурная независимость такого подхода в применении к высокопроизводительным параллельным вычислениям.

Инструментальные средства высокоуровневого синтеза СБИС

В рамках разработки предлагаемой авторами технологии был реализован набор программных инструментальных средств, позволяющих:

— выполнять преобразование исходного кода на ФППЯ в промежуточное представление в виде информационного и управляющего графов;

— осуществлять оптимизационные преобразования, что повышает эффективность функционально-потоковых параллельных программ (ФППП) [8];

— проводить отладку и анализ функционально-потокового параллельного кода во время выполнения, обеспечивая поиск ошибок и трассировку [9];

— осуществлять трансляцию промежуточного представления ФППП в описание СБИС на HDL-языках (языках описания аппаратуры — *Hardware Description Language*) [5].

На рис. 1 представлен состав разработанных инструментальных средств поддержки проектирования на основе предложенного метода высокоуровневого синтеза.

Набор разработанных авторами программных инструментальных средств функционирует в составе интегрированной среды разработки (оболочки) и

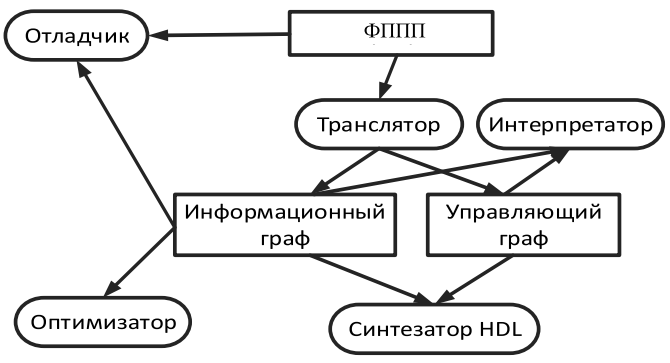

Рис. 1. Архитектура инструментальных средств поддержки проектирования

позволяет сформировать набор отлаженных функций для их реализации в виде СБИС. Оболочка предоставляет пользователю информационные ресурсы для организации всего процесса высокоуровневого синтеза СБИС на основе функционально-потокового параллельного подхода.

Транслятор представляет собой программное обеспечение, осуществляющее синтаксическую проверку исходного текста на ФППЯ "Пифагор" и его преобразование в промежуточное представление в виде информационного и управляющего графов. Сформированный информационный граф (ИГ) полностью задает структуру обработки данных программы. В состав транслятора также входит генератор управляющего графа (УГ), который принимает на вход ИГ и формирует по нему описание УГ. Промежуточные представления в виде ИГ и УГ являются входными данными для интерпретатора, оптимизатора, отладчика и синтезатора кода на языках HDL.

Для отладки функционально-потоковых программ реализован отладчик [9]. Отладчик позволяет выполнять отладку ФППП в одном из трех режимов: пошаговая отладка, отладка слоев и отладка ветвей.

Режим пошаговой отладки аналогичен режиму отладки последовательных программ. Особенности ФППЯ и модели вычисления позволяют проводить последовательную отладку и получать корректный результат независимо от порядка выполнения параллельных участков программы.

Режим отладки слоев предполагает отладку и порядок исполнения по готовности данных. Это означает, что на каждом шаге выполняются те операторы, для которых в данный момент данные готовы. Такой вариант отладки повторяет алгоритм работы интерпретатора. При отладке по слоям представляется возможность визуального контроля числа тактов конвейера (тактов исполнения) алгоритма. Это позволяет оценить степень параллелизма алгоритма (число операторов) и, как следствие, контролировать порядок работы алгоритма в параллельной системе.

Режим отладки ветвей предполагает отладку одной независимой ветви ИГ — исполнение всех операторов последовательно. В этом режиме отладчик выделяет в ФППП цепочку операторов, которые могут быть выполнены только последовательно. Отметим, что при этом у операторов нет зависимости по данным от других вершин.

Интерпретатор позволяет исполнить текст программы на ФППЯ. Входными данными для интерпретатора являются информационный и управляющий графы, а также аргумент функции верхнего уровня. Аргумент представляется в формате описания ИГ, для этого его обрабатывают транслятором.

Оптимизатор использует также входное промежуточное представление, осуществляет оптимизирующие преобразования над ИГ, результат которых сохраняется в виде промежуточного представления ИГ.

К оптимизирующим преобразованиям, осуществляемым на данном этапе, относятся:

- удаление неиспользуемого кода;
- оптимизация повторяющихся вычислений;
- непосредственная подстановка функций;

- удаление повторяющегося кода;
- оптимизация на основе эквивалентных преобразований алгебры функционально-потоковой параллельной модели.

Разработанные алгоритмы синтеза описания схемы СБИС из описания на ФППЯ позволили создать программные средства, обеспечивающие маршрут проектирования высокоуровневого синтеза на базе ФППЯ [5]. На основе этих алгоритмов был разработан транслятор (синтезатор) с ФППЯ в языки описания аппаратуры. Входными данными для синтезатора являются информационный и управляющий графы, формируемые транслятором с ФППЯ, а также аргумент функции верхнего уровня. Аргумент функции и типы данных для аргумента задаются в отдельном файле типизации (ограничений). Это позволяет отделить исходное описание алгоритма с динамической типизацией от конкретной реализации с заданными размерностями данных и получить множество реализаций из одного описания алгоритма с разной степенью параллелизма. При функционировании синтезатора в соответствии с разработанным алгоритмом синтеза HDL на первоначальном этапе осуществляется синтез графовых представлений (ИГ и УГ), в том числе происходит вычисление константных выражений, задание и вычисление типов. На втором этапе выполняется оптимизация полученных представлений посредством удаления вычисленных вершин. Третьей стадией является преобразование (сокращение) параллелизма исходного алгоритма под ограничения целевой платформы. Для этого применяют разработанные алгоритмы эквивалентных преобразований [10].

На заключительной стадии работы синтезатора формируется описание однокристальной системы на HDL-языке, в том числе происходит генерация входных и выходных портов, генерация таблицы имен (регистров), связывание входных портов с входными регистрами и генерация описания модуля на HDL. Так получают описание модуля, которое состоит из схемы обработки данных и управляющей схемы. Схема обработки данных генерируется из преобразованного ИГ. Каждая стадия конвейера в полученной схеме содержит сигнал разрешения работы, к которому в дальнейшем "привязываются" выходные управляющие сигналы схемы управления. Схема управления вычислениями генерируется из модифицированного УГ, над которым проведены преобразования для изменения степени параллелизма схемы под ограничения целевой платформы.

Разработанные авторами программные инструментальные средства дали возможность реализовать набор задач, позволяющий сравнить эффективность предлагаемого метода с существующими методами синтеза СБИС.

Результаты тестирования

На основе анализа наиболее часто используемых при промышленном проектировании СБИС/ПЛИС для сравнения известных подходов были выбраны следующие методы синтеза:

— синтез из описания на поведенческом уровне с помощью HDL;

— высокоуровневый синтез на C-подобном языке (*High-Level Synthesis*, HLS) [2, 12];

— высокоуровневый синтез в среде MATLAB/Simulink с последующим автоматическим синтезом проекта в HDL [11].

В качестве критериев для сравнения использовали быстродействие и занимаемый ресурс целевой платформы. Быстродействие схемы определяет количественную меру результата, выдаваемого СБИС в единицу времени. К метрикам, определяющим быстродействие схемы, относят:

— максимальную тактовую частоту схемы;

— задержку схемы в тактах до выдачи результата (латентность).

Отметим, что сравнение этих метрик у разных схем некорректно. Причина в том, что при различных алгоритмах синтеза у конвейерных схем для одного и того же алгоритма могут быть получены результаты с более длинным конвейером (большей латентностью в тактах) и большей тактовой частотой и наоборот. Поэтому для корректного сравнения использовалась композитная метрика от появления данных на входе в схему до выдачи результата, рассчитываемая по формуле

$$Tw = Cl/F_{\max},$$

где $F_{\max}$ — максимальная тактовая частота схемы; Cl — задержка схемы в тактах (длина конвейера).

Для определения максимальной тактовой частоты схемы использовался результат оценки частоты схемы, которая получается после этапа размещения и трассировки кристалла (*Place and route*) в САПР Xilinx ISE/Vivado [12]. Для оценки задержки схемы использовалась симуляция RTL-кода с оценкой задержки схемы в тактах. Для симуляции использовалось программное обеспечение Mentor Graphics ModelSim [13].

Оценка занимаемого ресурса также проводилась в базисе ПЛИС. Использовались следующие основные группы ресурсов:

- логические ячейки;
- триггеры/регистры;
- встроенные специализированные блоки, включающие блочную память и специализированные арифметические блоки (умножители DSP).

Среди тестовых задач, используемых при сравнении методов синтеза, выделены два класса алгоритмов: обработки данных и управления. Так же как и в работах [14, 15], задачи взяты из пакета CHStonex.

Список таких задач включает:

- вычисление квадратного корня (модуль комплексного числа);
- умножение матриц;
- вычисление суммы элементов массива (последовательно);
- вычисление суммы элементов массива (параллельно);
- цифровой фильтр с конечной импульсной характеристикой (FIR-фильтр).

Результаты реализации тестовых задач по метрикам группы производительности приведены на рис. 2 и 3.

На рис. 2 приведена сравнительная диаграмма производительности методов синтеза. Числовые значения на диаграмме приведены в процентах. Из анализа диаграммы видно, что реализации в MATLAB показывают сравнимую с реализацией на HDL производительность на трех задачах из семи, метод HLS — на одной задаче из семи, функционально-потоковый параллельный метод (ФППМ) — на шести из семи задач. При этом остальные задачи показывают производительность в 1,5—2 раза меньше, чем реализация на HDL. По суммарной производительности на наборе тестовых задач методы синтеза распределились в следующем порядке (в порядке убывания):

1) HDL;
2) ФППМ;
3) MATLAB;
4) HLS.

На рис. 3 приведены диаграммы занимаемого ресурса для разных методов по логическим ячейкам.

На диаграмме, представленной на рис. 3, видно, что решения, полученные методом HLS, занимают

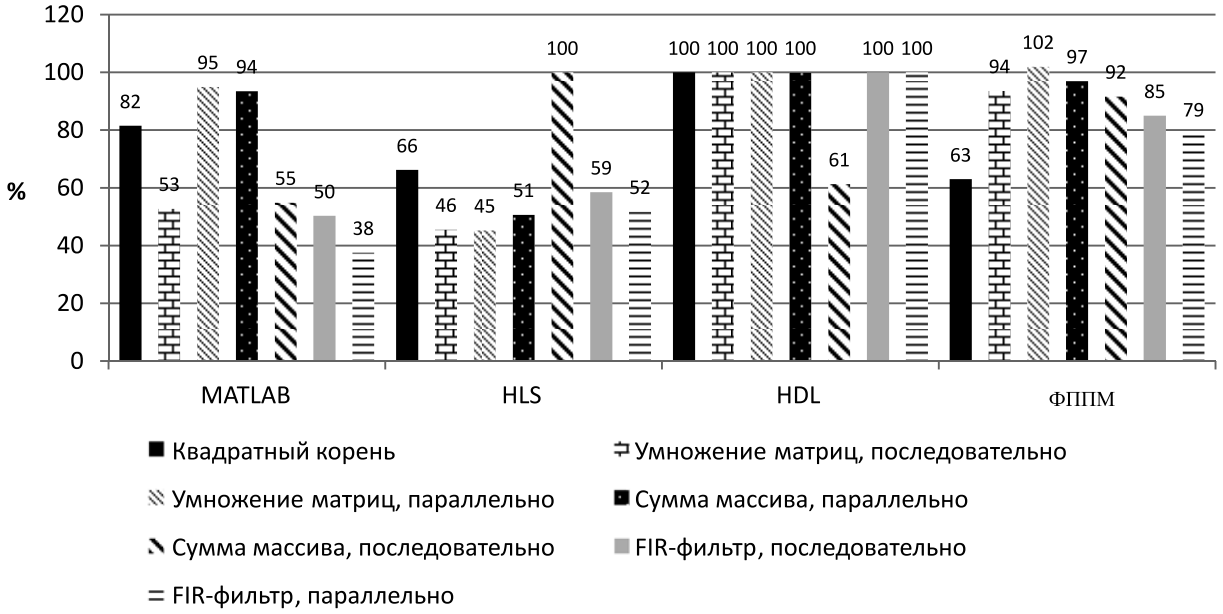

Рис. 2. Производительность методов на тестовых задачах

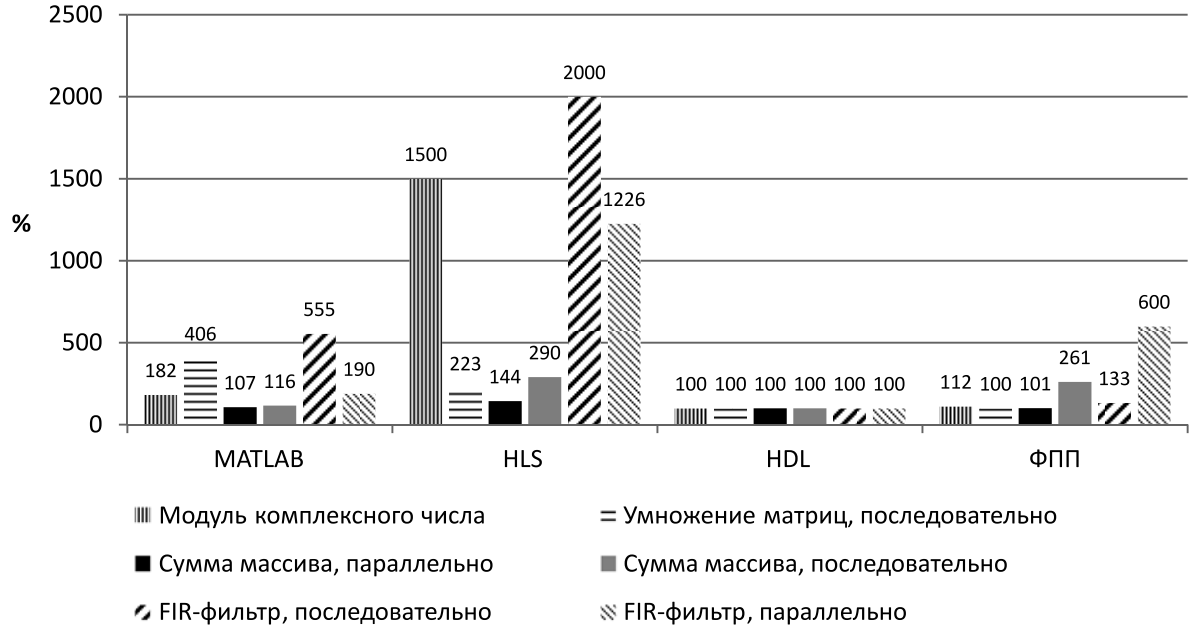

Рис. 3. Ресурс логических ячеек

самый большой ресурс, многократно превышающий остальные. Это связано с неэффективностью распараллеливания изначально последовательного описания алгоритма. Реализации тестовых задач в среде MATLAB по занимаемому ресурсу логических ячеек и триггеров в целом отличаются от HDL-реализации в 2—4 раза. Исключение составили три задачи с последовательной реализацией (сумма массива, умножение матриц и FIR-фильтр). Последовательные варианты реализации в MATLAB получены из параллельных путем изменения настроек HDL-кодера. Неоптимальное использование ресурса в последовательных вариантах показывает неэффективность преобразования параллелизма в MATLAB, так как последовательная реализация получается из одного и того же исходного описания алгоритма путем автоматического преобразования из последовательной формы описания в параллельную.

Разрабатываемый метод высокоуровневого синтеза показывает сравнимое использование ресурса при сравнимой с HDL-реализациями производительностью. Исключение составляют задачи умножения и сложения матриц параллельно. Анализ этих двух вариантов показал, что избыточное использование ресурса триггеров возникает вследствие большего числа стадий конвейера в этих задачах (четыре стадии в ФППМ против одной в HDL/MATLAB). Большее число стадий конвейера обусловливается изначальным описанием с максимальным параллелизмом на уровне операций. В таких случаях их целесообразнее объединять в одну стадию конвейера, уменьшая параллелизм схемы.

Заключение

Разработанный метод синтеза HDL-описания СБИС из промежуточного представления ФППП позволил реализовать инструментальные средства для поддержки технологии высокоуровневого синтеза. На основе предложенных алгоритмов разработан синтезатор с ФППЯ в HDL-описание СБИС. Синтезатор позволяет преобразовывать неизменное промежуточное представление алгоритма на ФППЯ в различные варианты схем СБИС с использованием различных размерностей и типов данных, а также различным уровнем параллелизма схемы. Это позволяет обеспечить архитектурную независимость и, как следствие, выбор наиболее оптимального варианта реализации в соответствии с заданием на проектирование, а также сократить временные затраты на проект.

Результаты сравнения предлагаемого метода синтеза на основе функционально-потокового параллельного описания показывают сравнимую с реализацией на HDL производительность, что превосходит при этом наиболее распространенный метод высокоуровневого синтеза HLS по эффективности использования ресурса. При этом получение различных вариантов по степени параллелизма из одного описания в предлагаемом методе не сказывается на эффективности реализации по количеству занимаемого ресурса.

Список литературы

1. **Непомнящий О. В., Шайдуров В. В., Легалов А. И., Рыженко И. Н.** Технология архитектурно-независимого, высокоуровневого синтеза сверхбольших интегральных схем // Доклады АН ВШ РФ. Новосибирск: НГТУ. 2014. Т. 57, № 3. С. 35—39.

2. **Vivado** Design Suite User Guide. High-Level Synthesis. UG902 — Xilinx — 2015. URL: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug902-vivado-high-level-synthesis.pdf

3. **Каляев И. А., Дордопуло А. И., Левин И. И.** и др. Технология программирования вычислительных систем гибридного типа // Вычислительные технологии. 2016. Т. 21, № 3. С. 33—44.

4. **Алехин В. А.** SystemC. Моделирование электронных систем. М.: Горячая линия — Телеком, 2018. 320 с.

5. **Легалов А. И., Непомнящий О. В., Рыженко И. Н.** Метод архитектурно-независимого высокоуровневого синтеза СБИС // Известия ЮФУ. Технические науки. 2018. № 8. С. 38—47.

6. **Bosilca G., Bouteiller A., Danalis A.** et al. PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability // IEEE Computing in Science and Engineering. 2013. Vol. 15, N. 6. P. 36—45.

7. **Danalis A., Bosilca G., Bouteiller A.** et al. PTG: An Abstraction for Unhindered Parallelism // Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing, 2014. P. 21—30.

8. **Васильев В. С.** Оптимизация программ функционально-потокового языка Пифагор // Перспективы развития информационных технологий. 2014. № 20. С. 7—14.

9. **Удалова Ю. В., Легалов А. И., Сиротинина Н. Ю.** Средства отладки функционально-потоковых параллельных программ // Доклады АН ВШ РФ. 2008. Т. 10, № 1. С. 96—105.

10. **Легалов А. И., Матковский И. В., Кропачева М. С.** и др. Технологические аспекты создания, преобразования и выполнения функционально-потоковых параллельных программ // Сб. тр. конф. "Научный сервис в сети Интернет: все грани параллелизма", Новороссийск, 23—28 сентября 2013. М.: Изд-во МГУ им. Ломоносова, 2013. С. 443—447.

11. **Using** Simulink to Deploy a MATLAB Algorithm on an FPGA or ASIC. URL: https://www.mathworks.com/videos/using-simulink-to-deploy-a-matlab-algorithm-on-an-fpga-or-asic-1484667134277.html

12. **Vivado** Design Suite User Guide Implementation 2018. URL: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ ug904-vivado-implementation.pdf

13. **Каршенбойм И.** Краткий курс HDL. Ч. 8. Моделирование в ModelSim SE // Компоненты и технологии. 2008. № 11. С. 139—144.

14. **Nane R., Sima V.-M., Pilato C.** et al. A Survey and Evaluation of FPGA High-Level Synthesis Tools // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. 2016. Vol. 35, N. 10. P. 1591—1604.

15. **LegUp** High-Level Synthesis.URL: http://legup.eecg.utoronto.ca/download.php

The Method of High-Level Synthesis and Software Toolkit for Description Algorithm of VLSI

O. V. Nepomnyashchiy, 2955005@gmail.com, **I. N. Ryzhenko,** rodgi@kras.ru, Siberian Federal University, Krasnoyarsk, 660041, Russian Federation

Corresponding author:

Ryzhenko Igor N., Postgraduate Student, Siberian Federal University, 660041, Krasnoyarsk, Russian Federation
E-mail: rodgi.krs@gmail.com

Received on August 20, 2019
Accepted on October 10, 2019

The article considers high-level design flow of single-chip systems for parallel data processing. The authors present solution on the base of the original functional data-flow language and the model of massive parallel processing. The language is used for description of the initial algorithm. The solution provides architectural independence of the program and allows to consider a wide variety of solutions taking constraints into account, and to select the optimal solution. The developed software toolkit for high-level design is described. The toolkit allows a developer to translate, debug, optimize programs and convert algorithm descriptions from the functional data stream language to a hardware description language. The developed software has been successfully tested on a number of test cases.

Keywords: *VLSI (Very Large Scale Integration), parallel computing, functional programming, high-level synthesis, synthesizer, HDL, translator, compiler*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project nos. 17-07-00288.*

For citation:

Nepomnyashchiy O. V., Ryzhenko I. N. The Method of High-Level Synthesis and Software Toolkit for Description Algorithm of VLSI, *Programmnaya Ingeneria,* 2019, vol. 11, no. 1, pp. 34—39.

DOI: 10.17587/prin.11.34-39

References

1. **Nepomnjaschiy O. V., Shajdurov V. V., Legalov A. I., Ryzhenko I. N.** The technology of architecturally independent high-level synthesis of VLSI, *Doklady ATVSh RF.* Novosibirsk, NGTU, 2014, vol. 57, no. 3, pp. 35—39 (in Russian).

2. **Vivado** Design Suite User Guide. High-Level Synthesis. UG902 — Xilinx — 2015, available at: http://www.xilinx.com/support/documentation/sw_manuals/xilinx2014_4/ug902-vivado-high-level-synthesis.pdf

3. **Kaljaev I. A., Dordopulo A. I., Levin I. I., Gudkov V. A., Gulenok A. A.** Tehnologija programmirovanija vychislitel'nyh sistem gibridnogo tipa, *Vychislitel'nye tehnologii*, 2016, vol. 21, no. 3, pp. 33—44 (in Russian).

4. **Alehin V. A.** SystemC. Modeling of electronic system // Gorjachaja linija — Telekom, 2018, 320 p. (in Russian).

5. **Legalov A. I., Nepomnjashhij O. V., Ryzhenko I. N.** The method of architecturally independent high-level synthesis of VLSI, *Izvestija JuFU, Tehnicheskie nauki*, 2018, no. 8, pp. 38—47 (in Russian).

6. **Bosilca G., Bouteiller A., Danalis A., Faverge M., Herault T., Dongarra J.** PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability, *IEEE Computing in Science and Engineering*, 2013, vol. 15, no. 6, pp. 36—45.

7. **Danalis A., Bosilca G., Bouteiller A., Herault T., Dongarra J.** PTG: An Abstraction for Unhindered Parallelism, *Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*, 2014, pp. 21—30.

8. **Vasil'ev V. S.** Optimizacija programm funkcional'no-potokovogo jazyka Pifagor, *Perspektivy razvitija informacionnyh tehnologij,* 2014, no. 20, pp. 7—14 (in Russian).

9. **Udalova Ju. V., Legalov A. I., Sirotinina N. Ju.** A Toolkit for debugging of Data-Driven Functional Parallel Programmes, *Doklady AN VSh RF*, 2008, vol. 10, no. 1, pp. 96—105 (in Russian).

10. **Legalov A. I., Matkovskij I. V., Kropacheva M. S., Udalova Ju. V., Vasil'ev V. M.** Technological aspects for creation and transformation of functional-dataflow parallel programs, *Nauchnyj servis v seti Internet: vse grani parallelizma,* 23—28 September 2013, Novorossijsk, Moscow, Publ. MSU, 2013, pp. 443—447 (in Russian).

11. **Using** Simulink to Deploy a MATLAB Algorithm on an FPGA or ASIC, available at: https://www.mathworks.com/videos/using-simulink-to-deploy-a-matlab-algorithm-on-an-fpga-or-asic-1484667134277

12. **Vivado** Design Suite User Guide Implementation 2018, available at: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_3/ ug904-vivado-implementation.pdf

13. **Karshenbojm I.** Short review of HDL. V. 8. Modeling in ModelSim SE*, Komponenty i tehnologii*, 2008, no. 11, pp. 139—144 (in Russian).

14. **Nane R., Sima V.-M., Pilato C., Choi J., Fort B., Canis A., Chen Y. T., Hsiao H., Srown S., Ferrandi F., Anderson J., Bertels K.** A Survey and Evaluation of FPGA High-Level Synthesis Tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2016, vol. 35, no. 10, pp. 1591—1604.

15. **LegUp** High-Level Synthesis, available at: http://legup.eecg.utoronto.ca/download.php

УДК 519.7; 004.056 DOI: 10.17587/prin.11.40-46

А. В. Галатенко, канд. физ.-мат. наук, ст. науч. сотр., agalat@msu.ru,
В. А. Плетнева, аспирант, pletnyova_va@mail.ru, МГУ имени М. В. Ломоносова

Выразимость моделей безопасности take-grant и невлияния в рамках модели CBAC*

Исследованы вопросы вложимости классических моделей безопасности take-grant и невлияния в современную модель CBAC (Concept-Based Access Control). Под вложением понимается инъективное отображение, сохраняющее свойства безопасности и функциональные возможности системы. Основными результатами, представленными в настоящей статье, являются конструктивно доказанные теоремы о вложимости моделей.

Ключевые слова: *формальные модели безопасности, модель CBAC, модель take-grant, модель невлияния, вложение моделей, сложность проверки безопасности*

Введение

Наличие формальной модели безопасности является обязательным требованием к системам с повышенным уровнем защищенности, начиная с "Оранжевой книги" [1]. К настоящему моменту разработано значительное число различных моделей, ориентированных прежде всего на поддержание конфиденциальности информации. Обзор классических результатов, таких как модели Белла—Лападула, take-grant и невлияния, можно найти в работе А. А. Грушо и Е. Е. Тимониной [2]. Из более поздних результатов можно выделить модель RelBAC [3], созданную для системы обработки наукометрической информации "ИСТИНА" [4], и модель CBAC (*Concept-Based Access Control*) [5].

В соответствии с общепринятой аксиомой (см., например, [2]) безопасность системы полностью определяется доступами. Содержательно глобальная безопасность может быть определена двумя способами. Первый из них определяется как невозможность перехода системы в заданное множество опасных состояний, например, появление доступа на чтение (нарушение конфиденциальности) или на запись (нарушение целостности) из некоторого предопределенного множества. Второй способ — требование сохранения некоторых свойств системы на протяжении жизненного цикла, например, ss-, *- и ds-свойства в модели Белла—Лападула [2] или коммутативность диаграммы функционирования в модели невлияния. Важно отметить, что свойство системы быть безопасной в общем случае неразрешимо [5, 6]. Другими словами, нет алгоритма, который бы по начальной конфигурации системы определял, существует ли последовательность доступов, переводящая систему в небезопасное состояние. Однако в ряде частных случаев ("теоремы раскрутки" для классических моделей, дополнительные ограничения в модели [5]) безопасность становится разрешимой.

* Работа поддержана грантом РФФИ 18-07-01055.

Большинство моделей безопасности может быть выражено либо на языке нагруженных графов (возможно, с нагрузкой в виде предикатов), либо на языке автоматов. Примерами графовых моделей являются модели take-grant и CBAC. По сути, автоматными являются модели Белла—Лападула и невлияния. В случае, когда требуется безопасная интеграция подсистем, основанных на различных моделях безопасности, актуальной становится задача выразимости одних моделей через другие. В качестве практических примеров можно привести слияние нескольких компаний и организацию служб типа "единого окна".

Результаты исследований, представленные в настоящей статье, посвящены изучению модели CBAC. Выбор модели обусловлен тем, что, с одной стороны, она достаточно сложна, чтобы в ней реализовывалась неразрешимость свойства безопасности, с другой стороны, она достаточно просто представима с формальной точки зрения. Далее обобщим понятие безопасности в CBAC и покажем, что в модель CBAC можно достаточно естественно вложить как графовую модель take-grant, так и автоматную модель невлияния. Так как свойство безопасности в исходных моделях разрешимо, возникают подклассы CBAC с разрешимым свойством безопасности. При этом подкласс, порожденный моделью take-grant, не удовлетворяет достаточному условию разрешимости, представленному в работе [5]. Под вложением понимается инъективное отображение систем в терминах исходной модели в системы в терминах целевой модели, сохраняющее свойство безопасности и являющееся гомоморфизмом с точки зрения функционирования. Содержательно это означает, что система-прообраз и система-образ функционируют одинаково.

Модель CBAC

Модель CBAC — графовая модель безопасности компьютерных систем, введенная в работе [5]. Ее основной сущностью является граф данных — нагруженный ориентированный граф $D=<O, A, R>$, где $O=\{o_1, ..., o_N\}$ — конечное множество объектов;

A — конечное множество имен атрибутов $R \subseteq O \times O \times A$ — множество (нагруженных) ребер. Каждому объекту поставлено в соответствие некоторое рациональное число. Значение объекта — это отображение $\mu : O \to \mathbb{Q}$. Пара (D, μ) называется конфигурацией системы. Содержательно ребра задают отношения на множестве объектов. Решение о предоставлении или отказе в доступе может быть принято в зависимости от отношений, состояния объектов, субъекта, контекста запроса.

Граф данных изменяется во времени. Время в модели дискретно и индексировано множеством натуральных чисел. В каждый момент времени на вход может поступать либо запрос на доступ (считаем, что типы доступа являются элементами некоторого конечного множества), либо запрос на выполнение одной из операций модификации графа. Запрос отправляется от имени некоторого субъекта s, элемента конечного множества субъектов S. В дальнейшем в случаях, когда субъект определен однозначно или не играет существенной роли, для краткости аргумент s будем опускать. Рассмотрим следующие операции: редактирование объекта, создание объекта или ребра, удаление объекта или ребра.

- Редактирование объекта *update*(o, x) — присвоение нового значения x объекту o.
- Создание объекта *create*(o, o', a, x) — создание нового объекта o со значением x и ребра к объекту o с меткой a из вершины o' текущего графа.
- Создание ребра *createEdge*(o_1, a, o_2) — создание ребра с меткой a между объектами o_1 и o_2.
- Удаление объекта *delete*(o) (при этом удаляются все ребра, инцидентные o) и удаление ребра с меткой l между вершинами o_1 и o_2 *deleteEdge*(o_1, l, o_2).

Правила доступа, определяющие разрешенность доступа или операции, задаются формулами первого порядка. Можно считать, что совокупность формул задает политику безопасности.

В работе [5] под безопасностью понималась недостижимость (отсутствие достижимости) опасных конфигураций, т. е. таких конфигураций (D, μ), в которых некоторый субъект s может редактировать заданный объект o. Задача в такой постановке оказалась алгоритмически неразрешимой; в качестве примера разрешимого случая авторы привели систему с ограниченным диаметром графа D. Введем более общее понятие безопасности. Пару (α, Q), где $\alpha = a_1, \ldots, a_\tau$ — последовательность запросов, $Q = \left((D_1, \mu_1), \ldots, (D_\tau, \mu_\tau)\right)$ — последовательность конфигураций, порожденных последовательностью запросов из некоторой начальной конфигурации (D_0, μ_0), $\tau \in \mathbb{N}$, назовем поведением системы. Помимо запрещения заданного множества доступов для определения безопасности разрешим накладывать ограничения на возможные поведения. Такое обобщение позволит достаточно естественно вкладывать в модель СВАС модели с мандатным разграничением доступа.

Модель take-grant

Модель take-grant — это модель распространения прав доступа в системе с дискреционной политикой безопасности. Подробное описание можно найти, например, в работе [2]. Система функционирует в дискретном времени, индексированном множеством натуральных чисел. Состояние системы описывается графом доступов. Множество типов доступов R_{tg} можно разделить на два подмножества. К первому из них относятся "классические" доступы на чтение r, запись w и исполнение c. Это подмножество может быть расширено произвольным образом, при этом все результаты сохранятся. Ко второму типу относится пара выделенных типов *take* (t) и *grant* (g).

Пусть $\tau \in \mathbb{N}$, $O(\tau)$ — конечное множество объектов, $S(\tau) = O(\tau)$ — множество субъектов в момент времени τ. На множестве объектов как на вершинах определен ориентированный нагруженный граф $G_\tau(V, E)$, где $V = O(\tau)$, а ребро (v_1, p, v_2) с меткой $p \in R_{tg}$ принадлежит E, если v_1 имеет доступ p к v_2. Отметим, что между парой вершин могут быть кратные ребра с попарно различными метками.

Преобразования графов доступов проводятся с помощью четырех команд, которые опишем графически:

1. *Take.* Если в исходном графе доступов G был подграф $S \xrightarrow{t} X \xrightarrow{\alpha} Y$, то в новом состоянии G', построенном по команде *take* с аргументами S, X и Y, будет подграф $S \xrightarrow{t} X \xrightarrow{\alpha} Y$, $S \xrightarrow{\alpha} Y$.

2. *Grant.* Если в исходном графе доступов G был подграф $S \xrightarrow{g} X$, $S \xrightarrow{\alpha} Y$, то в новом состоянии G', построенном по команде *grant* с аргументами S, X и Y, будет подграф $S \xrightarrow{g} X \xrightarrow{\alpha} Y$, $S \xrightarrow{\alpha} Y$.

3. *Create.* Данная команда создает новую вершину X, и в графе G' появляется подграф $S \xrightarrow{\beta} X$, где стрелка означает пучок ребер, пометки которых образуют множество β.

4. *Remove.* Данная команда исключает права доступа субъекта S к объекту X (графически: удаляются ребра с нужной пометкой), удаляет объекты (графически: удаляется объект и все инцидентные ребра).

Заметим, что доступ в графе может стать разрешенным в результате цепочки команд тогда и только тогда, когда этот доступ может стать разрешенным в результате цепочки команд, не содержащей команды *Remove*.

Пусть задано множество запрещенных доступов. Под безопасностью системы будем понимать невозможность получения запрещенного доступа путем преобразования графа G последовательностью разрешенных команд.

Свойство безопасности модели take-grant разрешимо.

Определение 1. В графе доступов G вершины P и S называются *tg*-связными, если существует путь в G, соединяющий P и S, безотносительно ориентации дуг, но такой, что каждое ребро этого пути имеет метку t или g.

Теорема (критерий безопасности системы [2]). Субъект P может получить доступ α к субъекту X тогда и только тогда, когда выполняются следующие условия:

1) существует субъект S, такой что в текущем графе G есть дуга $S \xrightarrow{\alpha} X$;

2) S *tg*-связан с P.

Из приведенного критерия безопасности очевидным образом вытекает алгоритм проверки безопасности. Приведем этот алгоритм и оценим сложность его реализации.

1. Разбиваем множество вершин V в объединение компонент *tg*-связности (например, используя поиск в ширину или в глубину; сложность такой реализации составит $O(|V|+|E|)$).

2. Согласно критерию, если из некоторой вершины *tg*-компоненты есть доступ, то такой доступ есть из любой вершины этой компоненты. Объединяем доступы внутри каждой компоненты, описывая доступы всех вершин этой компоненты. Это также можно сделать со сложностью $O(|V|+|E|)$.

3. Заметим, что проверка наличия запрещенного доступа выполняется в пункте 2.

Таким образом, сложность описания множества всевозможных доступов есть $O(|V|+|E|)$.

Выразимость в рамках модели СВАС

Пусть TG — множество графов модели take-grant SA — множество графов модели СВАС. Поставим в соответствие графу $G = (V, E)$ модели take-grant граф $D=< O, A, R >$ модели СВАС с помощью отображения $F : TG \to SA$ по следующему правилу.

1. Множество вершин графа D совпадает с множеством вершин графа G: $O = V$. Ассоциируем с O отображение $F_O : V \to O$, которое ставит в соответствие вершине $v \in V$ ее копию из O.

2. Множество имен атрибутов совпадает с множеством типов доступов модели take-grant R_{tg}.

3. Множество ребер графа D совпадает с множеством ребер графа G: $R = E$. Ассоциируем с R отображение F_R, такое что $F_R(v_1, a, v_2) = (o_1, a, o_2)$, где $o_1 = F_O(v_1)$, $o_2 = F_O(v_2)$, $a \in R_{tg}$.

4. Значение всех объектов равно 0, функция $\mu : O \to \{0\}$. При преобразованиях графа значение объекта меняться не будет, т. е. операция редактирования объекта отсутствует.

5. Операции: создание объекта или ребра; удаление объекта или ребра.

6. Правила доступа: ребро возникает в результате применения команды *take* или *grant*. Если в графе G были ребра (S, t, X) и (X, a, Y), то в образе $F(G)=D$ будут ребра $(F_O(S), t, F_O(X))$ и $(F_O(X), a, F_O(Y))$. В результате применения команды *take* в графе G' появится ребро (S, a, Y), а в образе при применении команды $F(take)$ появится ребро $(F_O(S), a, F_O(Y))$. Если в графе G были ребра (X, g, S) и (X, a, Y), то в образе $F(G) = D$ будут ребра $(F_O(X), g, F_O(S))$ и $(F_O(X), a, F_O(Y))$. В результате применения команды *grant* в графе G' появится ребро (S, a, Y), а в образе при применении команды $F(grant)$ появится ребро $(F_O(S), a, F_O(Y))$. Командам *create* и *remove* в графе системы take-grant соответствуют команды $create(o', o, a, x)$, $deleteEdge(o_1, a, o_2)$, $delete(o)$ в графе модели СВАС. Пусть α — последовательность действий в графе из TG, тогда $F(\alpha)$ — образ этих действий в графе из SA. Остальные команды считаются запрещенными. Очевидно, что условие разрешенности команды в модели СВАС может быть записано с помощью формулы первого порядка.

Доступ в текущий момент разрешен, если присутствует ребро с соответствующей пометкой.

Пусть задано множество запрещенных доступов. Под безопасностью в образе будем понимать невозможность достижения последовательностью разрешенных преобразований графа D конфигурации (D, μ), при которой вершина o может получить запрещенный доступ a к вершине o', что равносильно появлению ребра (o, a, o').

Теорема 1. Отображение F удовлетворяет следующим свойствам:

- инъективность;
- сохранение безопасности/небезопасности;
- коммутативность диаграммы для любой последовательности действий α:

$$\begin{array}{ccc} G & \xrightarrow{F} & D \\ \alpha \downarrow & & \downarrow F(\alpha) \\ G' & \xrightarrow[F]{} & D' \end{array}$$

Доказательство. Инъективность очевидна, так как при тождественном отображении разные графы переходят в разные.

Коммутативность диаграммы будем доказывать индукцией по длине последовательности команд. Пусть α — пустая последовательность, тогда диаграмма очевидно коммутативна. Предположим, что для произвольной последовательности из n команд $\alpha = \alpha_1, ..., \alpha_n$ диаграмма коммутативна. Докажем коммутативность диаграммы для последовательности $\alpha' = \alpha_1, ..., \alpha_n\alpha_{n+1}$. Обозначим через G_0 и D_0 образы графов G и D после выполнения команд $\alpha_1, ..., \alpha_n$. Заметим, что по индуктивному предположению $F(G_0) = D_0$. Команда α_{n+1} может быть одной из четырех возможных: *take*, *grant*, *create*, *remove*. Рассмотрим все случаи:

1. $\alpha_{n+1} = take$.

Пусть по команде *take* граф G_0 отобразился в граф G', в котором добавилось ребро (v_1, a, v_2), образом которого при отображении F является ребро $(F_O(v_1), a, F_O(v_2)) = (o_1, a, o_2)$. Значит, в графе G_0 были ребра (v_1, t, x) и (x, a, v_2), а в графе D_0 были ребра (o_1, t, x) и (x', a, o_2), где $o_1 = F_O(v_1)$, $o_2 = F_O(v_2)$, $x' = F_O(x)$. Значит, в графе D' при применении команды $F(take)$ появится ребро (o_1, a, o_2), т. е. $D' = F(G')$.

2. $\alpha_{n+1} = grant$.

Пусть по команде *grant* граф G_0 отобразился в граф G', в котором добавилось ребро (v_1, a, v_2), образом которого при отображении F является ребро $(F_O(v_1), a, F_O(v_2)) = (o_1, a, o_2)$. То есть в графе G_0 были ребра (x, g, v_1) и (x, a, v_2), а в графе D_0 были ребра (x', g, o_1) и (x', a, o_2), где $o_1 = F_O(v_1)$, $o_2 = F_O(v_2)$, $x' = F_O(x)$. Значит, в графе D' при применении команды $F(grant)$ появится ребро (o_1, a, o_2) и $D' = F(G')$.

3. α_{n+1} = *create*.

Создание вершины в графе G_0 равносильно созданию вершины в графе D_0. Коммутативность диаграммы в этом случае очевидна.

4. α_{n+1} = *remove*.

Удаление части (всех) доступов в графе G_0 равносильно удалению соответствующих ребер в графе D_0. Коммутативность диаграммы в этом случае очевидна. Случай удаления вершины рассматривается аналогично.

Сохранение свойства безопасности, т. е. утверждение о том, что образ отображения F безопасен тогда и только тогда, когда безопасен прообраз, очевидным образом вытекает из коммутативности диаграммы и определения безопасности.

Замечание 1. Так как разрешимость свойства безопасности при вложении сохраняется, то построен содержательный пример системы в рамках модели СВАС, для которого свойство безопасности разрешимо. При этом сложность проверки безопасности не возрастает по сравнению со сложностью проверки в исходном графе модели take-grant и оценивается как $O(|V|+|E|)$.

Модель невлияния

В данном разделе рассматривается автоматная модель невлияния, описывающая системы с многоуровневой политикой безопасности. Подробное описание модели приведено в работе [7]. Содержательно система моделируется автоматом, на котором работают два пользователя: *High* (*H*) с высоким уровнем доступа и *Low* (*L*) с низким уровнем доступа. Система безопасна, если для пользователя *Low* корректно определен подавтомат, причем пользователь *High* не может менять состояния этого подавтомата. Другими словами, *H* не может повлиять на пользователя *L*.

Приведем формальное определение модели. Система описывается автоматом без выхода $B = (S, \Sigma, \delta)$, где S — множество состояний; каждое состояние задается как вектор значений $s = (s_1, ..., s_n)$ и S — совокупность возможных векторов; Σ — входной алфавит; $\delta : S \times \Sigma \to S$ — функция переходов.

Продолжим функцию δ на множество $S \times \Sigma^*$, где Σ^* — множество всех слов, составленных из букв алфавита Σ, e — пустое слово:

- $\delta(s, e) = s$;
- $\delta\left(s,\left(x^1, ..., x^{n+1}\right)\right) = \delta\left(\delta\left(s,\left(x^1, ..., x^n\right)\right), x^{n+1}\right)$,

где $x^i \in \Sigma$.

Определение 2. Автомат называется двухуровневым, если входной алфавит Σ представляется как объединение двух непересекающихся множеств: $\Sigma = \Sigma_H \amalg \Sigma_L$, а множество состояний — как $S = S_H \times S_L$.

Определение 3. Состояния $s^1, s^2 \in S$ называются эквивалентными, пишем $s^1 \sim s^2$, если их *Low*-компоненты равны.

Таким образом, S разбивается на классы эквивалентности. Класс эквивалентности, содержащий $s \in S$, будем обозначать $[s]$. Обозначим отображение, разбивающее S на классы эквивалентности, $\pi : S \to S/_\sim$.

Вводится также функция $Z_L : \Sigma^* \to \Sigma_L^*$, которая из последовательности входов автомата оставляет только команды пользователя *Low*:

- $Z_L(x) = e$, если $x \in \Sigma_H$, $Z_L(x) = x$, если $x \in \Sigma_L$;
- $Z_L(x^1 x^2) = Z_L(x^1) Z_L(x^2)$.

Определение 4. Система, задаваемая автоматом $B = (S, \Sigma, \delta)$, называется безопасной, если:

1) отображение $\tilde{\delta} : S/_\sim \times \Sigma_L^* \to S/_\sim$, определяемое формулой

$$\tilde{\delta}([s], w_L) = \left[\delta(s, w_L)\right],$$

где $[s] \in S/_\sim$ и $w_L \in \Sigma_L^*$, корректно определено;

2) $\pi \circ \delta = \tilde{\delta} \circ (\pi \times Z_L)$.

В левой и правой частях равенств стоят отображения $S \times \Sigma^* \to S/_\sim$. Иными словами, коммутативна следующая диаграмма:

$$\begin{array}{ccc} S \times \Sigma^* & \xrightarrow{\delta} & S \\ \downarrow{\scriptstyle \pi \times Z_L} & & \downarrow{\scriptstyle \pi} \\ S/_\sim \times \Sigma_L^* & \xrightarrow[\tilde{\delta}]{} & S/_\sim. \end{array}$$

Свойство безопасности в модели невлияния разрешимо.

Теорема [7]**.** Условия безопасности 1 и 2 определения 4 выполнены тогда и только тогда, когда они выполнены с заменой Σ^* на Σ, Σ_L^* на Σ_L.

Непосредственное применение теоремы приводит к алгоритму, сложность которого квадратично зависит от мощности множества S. Приведем оптимизированный алгоритм проверки безопасности.

Без ограничения общности можно считать, что элементы вектора $s = (s_1, ..., s_n) \in S$ — натуральные числа (можно их занумеровать).

Опишем функцию, ставящую в соответствие каждому классу эквивалентности $[s]$ некоторое число. Возьмем первые $n - k$ простых чисел и для каждого $s = (s_1, ..., s_k, s_{k+1}, ..., s_n) \in S$, где $(s_1, ..., s_k) \in S_H$, $(s_{k+1}, ..., s_n) \in S_L$, вычислим значение $l_i = p_1^{s_{k+1}} \cdot \ldots \cdot p_{n-k}^{s_n}$, где $i \leqslant |S|$. Если двум состояниям поставлено в соответствие одно и то же число l_i, то они эквивалентны и принадлежат одному классу эквивалентности C_i. Обозначим сложность вычисления функции через $M(S)$. Легко увидеть, что $M(S)$ есть $o(|S|)$. Для получения набора чисел l_i достаточно $O(|S| \cdot M(S))$ операций.

Рассмотрим класс эквивалентности $C_i = \{s^1, ..., s^m\}$. Заметим, что для того чтобы проверить выполнение условия безопасности 1 определения 4, достаточно проверить пары (s^1, s^2), ..., (s^1, s^m). Если $\delta(s^1, x_L) \sim \delta(s^j, x_L)$ и $\delta(s^1, x_L) \sim \delta(s^t, x_L)$, то $\delta(s^j, x_L) \sim \delta(s^t, x_L)$, т. е. для пары (s^j, s^t) условие безопасности 1 также выполнено. Таким образом,

сложность проверки условия безопасности 1 оценивается как $O\left(|S|\cdot|\Sigma_L|\cdot M(S)\right)$.

Несложно заметить, что по теореме для проверки условия безопасности 2 определения 4 достаточно проверить, что для любого $s \in S$ и для любой буквы $x_H \in \Sigma_H$ выполнено $\delta(s, x_H) \sim s$. Таким образом, сложность проверки условия безопасности 2 оценивается как $O\left(|S|\cdot|\Sigma_H|\cdot M(S)\right)$.

В результате сложность проверки безопасности оценивается как $O\left(|S|\cdot|\Sigma|\cdot M(S)\right)$.

Выразимость в рамках модели СВАС

Изоморфные автоматы будем считать совпадающими. Рассмотрим отношения ρ_H и ρ_L на квадрате множества состояний автомата. Считаем, что $s_1\rho_H s_2$, если найдется буква алфавита $a \in \Sigma_H$, такая что $\delta(s_1, a)=s_2$; $s_1\rho_L s_2$, если найдется буква алфавита $a \in \Sigma_L$, такая что $\delta(s_1, a)=s_2$. Отождествим автоматы, у которых оба отношения совпадают. При этом общность не потеряется, так как с точки зрения функционирования отождествленные автоматы по сути совпадают.

Поставим в соответствие автомату $B=(S, \Sigma, \delta)$ граф $D=< O, A, R >$.

- Множество вершин состоит из одной вершины: $O=\{o\}$.
- Ребер в графе нет: $R = \varnothing$.
- Множество имен атрибутов пусто: $A = \varnothing$.
- Поставим в соответствие каждому состоянию $s=(s_1, \ldots, s_n)$ рациональное число $q=p_1^{s_1}\cdot\ldots\cdot p_n^{s_n}$, где $p_1, \ldots, p_n$ — различные простые числа (для определенности будем считать, что это первые простые числа). Значение объекта в текущий момент времени равно q. Ассоциируем с O отображение $G_O(s) = G_O(s_1, \ldots, s_n)= p_1^{s_1}\cdot\ldots\cdot p_n^{s_n}$. Состояния q_1, q_2 объекта o будем называть эквивалентными, если в этих состояниях показатели простых чисел, соответствующих *Low*-компонентам, совпадают (определение корректно по основной теореме арифметики).
- Операции: редактирование объекта. Доступ на редактирование объекта разрешен в следующих случаях:
  - ◇ доступ осуществляет пользователь H, текущее состояние o есть образ некоторого состояния s исходного автомата, новое состояние есть образ состояния $\delta(s, a)$ для некоторого $a \in \Sigma_H$;
  - ◇ доступ осуществляет пользователь L, текущее состояние эквивалентно образу некоторого состояния s исходного автомата, новое состояние эквивалентно образу $\delta(s, a)$ для некоторого $a \in \Sigma_L$.

В остальных случаях доступ запрещен. Очевидно, что условие разрешенности может быть записано с помощью формулы первого порядка.

Пусть значение объекта o равно q. Обозначим $edit(\alpha, q)=q'$ переход по слову $\alpha \in \Sigma^*$ для операции $update(o, q')$ (присвоение нового значения q' объекту o).

Под безопасностью в графовой модели будем понимать выполнение двух условий.

1. Для любого $\alpha \in \Sigma_L^*$ и для любых эквивалентных $q_1 \sim q_2$ выполнено: $edit(\alpha, q_1) \sim edit(\alpha, q_2)$.
2. Для любого $\alpha \in \Sigma_H^*$ и для любого q : $edit(\alpha, q) \sim q$.

Содержательно условие 1 означает, что пользователь L не может различить состояния, отличающиеся только в H-компонентах (запрет чтения вверх), а условие 2 — что пользователь H не может менять L-компоненты (запрет записи вниз).

Теорема 2. Отображение G удовлетворяет следующим свойствам:

1) инъективность;

2) сохранение безопасности/небезопасности;

3) коммутативность диаграммы для любого $\alpha \in \Sigma^*$:

$$\begin{array}{ccc} S_0 & \xrightarrow{\ \alpha\ } & \delta(S_0, \alpha) \\ \Big\downarrow G_O & & \Big\downarrow G_O \\ O_0 & \xrightarrow[G_\alpha]{} & O. \end{array}$$

Доказательство. Покажем, что отображение инъективно. Рассмотрим пару автоматов $B_1=(S_1, \Sigma_1, \delta_1)$ и $B_2=(S_2, \Sigma_2, \delta_2)$. Если $S_1 \neq S_2$, то в образе различаются разрешенные состояния объекта o и, как следствие, политики безопасности. Если $S_1=S_2$, то различается хотя бы одно из отношений ρ_H, ρ_L и, следовательно, вновь возникает команда, разрешенная в одной из систем, но запрещенная в другой.

Докажем, что диаграмма коммутативна, воспользовавшись индукцией по длине входного слова. Для пустого слова коммутативность очевидна. Предположим, что диаграмма коммутативна для произвольного входного слова длины m. Рассмотрим слово $\alpha = a_1 \ldots a_m a_{m+1}$ длины $m + 1$. Пусть $\delta(s_0, a_1 \ldots a_m)=s'$. По индуктивному предположению $G_O(s')$ совпадает с состоянием СВАС-системы, порожденным словом $G_\alpha(a_1 \ldots a_n)$ из состояния $G_O(s_0)$, и достаточно доказать, что коммутативность сохраняется для произвольных однобуквенных входных слов. Последний факт является прямым следствием определений.

Покажем, что безопасные модели переходят в безопасные.

Пусть модель $B=(S, \Sigma, \delta)$ безопасна. Проверим выполнение условия 1. Рассмотрим пару эквивалентных состояний q_1, q_2 объекта o. Воспользуемся индукцией по длине входного слова. Для пустого слова ε отношение $edit(\varepsilon, q_1) \sim edit(\varepsilon, q_2)$ очевидно. Предположим, эквивалентность сохраняется для произвольного слова длины m. Рассмотрим слово $\alpha = a_1 \ldots a_m a_{m+1}$ в алфавите Σ_L.

По определению

$$edit\left(\alpha, q_1\right) = edit\left(a_{m+1}, edit\left(a_1 \ldots a_m, q_1\right)\right),$$

$$edit(\alpha, q_2) = edit(a_{m+1}, edit(a_1...a_m, q_2)).$$

По предположению состояния $q_1' = edit(a_1...a_m, q_1)$ и $q_2' = edit(a_1...a_m, q_2)$ эквивалентны. Таким образом, достаточно проверить справедливость условия 1 для однобуквенных слов. Если q_1', q_2' эквивалентны образу некоторого состояния s исходного автомата, справедливость условия 1 вытекает из безопасности исходного автомата и правил доступа в модели CBAC. В противном случае функция *edit* становится тождественной, и справедливость условия 1 очевидна.

Проверим справедливость условия 2. Так как для тождественной функции оно очевидно, достаточно рассмотреть случай, когда q — состояние объекта o, порожденное некоторым состоянием $s \in S$. Пусть $\alpha \in \Sigma_H^*$. В силу безопасности исходной системы $\delta(s, \alpha) \sim s$. В силу коммутативности диаграммы $q \sim edit(\alpha, q)$, т. е. условие 2 также выполнено.

Покажем, что небезопасные модели переходят в небезопасные.

Пусть модель $B=(S, \Sigma, \delta)$ небезопасна, т. е. нарушается одно из условий безопасности определения 4.

1. Пусть есть такое слово $\alpha \in \Sigma_L^*$, что для $s \sim s'$: $\delta(s, \alpha) \nsim \delta(s', \alpha)$. Тогда для соответствующих значений $q_1 \sim q_2$ нарушается условие 1 безопасности: $edit(\alpha, q_1) \nsim edit(\alpha, q_2)$.

2. Пусть существуют $a \in \Sigma_H$ и состояние $s \in S$, такие что $\delta(s, a) \nsim s$. Значит, для соответствующей состоянию s конфигурации объекта o нарушается условие 2 безопасности: $edit(a, q) \nsim q$.

Замечание 2. В работе А. А. Грушо и Е. Л. Шумицкой [8] модель невлияния была обобщена на случай автомата с выходом. Можно показать, что существует вложение обобщенной модели в классическую, а также то, что композиция вложений является вложением. Из этих утверждений следует, что модель Грушо и Шумицкой также вкладывается в модель CBAC.

Заключение

Исследованы выразительные возможности модели CBAC, введенной в работе [5]. Показано, что в эту модель достаточно естественно вкладываются классические модели take-grant и невлияния. Как следствие, в рамках CBAC возникают два новых класса с разрешимым свойством безопасности.

Авторы благодарят С. А. Афонина и В. А. Васенина за внимание к работе и ценные замечания.

Список литературы

1. **Department** of Defense Trusted Computer System Evaluation Criteria (The Orange Book), Department of Defense, DOD 5200.28-STD, 1985.

2. **Грушо А. А., Тимонина Е. Е.** Теоретические основы защиты информации. М.: Яхтсмен, 1996. 176 с.

3. **Васенин В. А., Иткес А. А., Шапченко К. А., Бухонов В. Ю.** Реляционная модель логического разграничения доступа на основе цепочек отношений // Программная инженерия. 2015. № 9. С. 11—19.

4. **Васенин В. А., Афонин С. А., Голомазов Д. Д., Козицын А. С.** Интеллектуальная Система Тематического Исследования Научно-технической информации (ИСТИНА) // Информационное общество. 2013. № 1—2. С. 21—36.

5. **Afonin S., Bonushkina A.** Validation of Safety-Like Properties for Entity-Based Access Control Policies // Advances in Soft and Hard Computing. 2019. P. 259—271.

6. **Harrison M. A., Ruzzo W. L., Ullman J. D.** Protection in Operating Systems // Communications of the ACM. 1976. Vol. 19, N. 8. P. 461—471.

7. **Moskowitz I. S., Costich O. L.** A classical automata approach to noninterference type problems // Proc. Computer Security Foundations Workshop V, 1992. P. 2—8.

8. **Грушо А. А., Шумицкая Е. Л.** Модель невлияния и скрытые каналы // Дискретная математика. 2002. Т. 14, № 1. С. 11—16.

Embeddability of Take-Grant and Noninterference Security Models in CBAC Model

A. V. Galatenko, agalat@msu.ru, **V. A. Pletneva**, pletnyova_va@mail.ru, Lomonosov Moscow State University, Moscow, 119991, Russian Federation

Corresponding author:

Pletneva Vesta A., Postgraduate Student, Lomonosov Moscow State University, Moscow, 119991, Russian Federation
E-mail: pletnyova_va@mail.ru

Received on September 12, 2019
Accepted on October 24, 2019

Computer systems with high level of security require a formal proof of security in the framework of some mathematical models. There exists a sufficiently large number of such models; most of them have either a graph nature or an automata nature. In some models security is decidable in all cases, however there exist examples in which

security is undecidable, so there emerges a need in additional constraints.

Another problem consists in mutual expressibility of different security models (e.g. in case of a merge of two systems into one). A possible way of such unification is embedding one system into another. Embedding is a mapping that satisfies three properties: injectivity, preserving security/insecurity and preserving functionality.

Our research is focused on Concept-Based Access Control (CBAC) model introduced by Afonin and Bonushkina in 2019. This is a graph model with undecidable security. We constructively show that two classical security models, namely take-grant and noninterference models, can be embedded in CBAC, and complexity of security validation in original systems and in CBAC images has the same order. Thus, CBAC is rich enough to naturally reflect properties of both graph-based models and automata-based models. Since security is decidable in take-grant and noninterference, embeddings produce two new subclasses of CBAC systems with decidable security.

Keywords: *formal security models, CBAC model, take-grant model, noninterference, embedding of models, security validation complexity*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project no. 18-07-01055.*

For citation:

Galatenko A. V., Pletneva V. A. Embeddability of Take-Grant and Noninterference Security Models in CBAC Model, *Programmnaya Ingeneria*, 2020, vol. 11, no. 1, pp. 40—46.

DOI: 10.17587/prin.11.40-46

References

1. **Department** of Defense Trusted Computer System Evaluation Criteria (The Orange Book), Department of Defense, DOD 5200.28-STD, 1985.

2. **Grusho A. A., Timonina E. E.** *Theoretical Foundations of Information Security,* Moscow, Yachtsman, 1996, 176 p. (in Russian).

3. **Vasenin V. A., Itkes A. A., Shapchenko K. A., Bukhonov V. Yu.** Relational Access Control Model Based on Chains of Relations, *Programmnaya Ingeneria*, 2015, vol. 9, no. 11, pp. 11—19 (in Russian).

4. **Vasenin V. A., Afonin S. A., Golomazov D. D., Kozitsyn A. S.** Intelligent system for management of scientific and technical information (ISTINA), *Informacionnoe Obshchestvo*, 2013, no. 1—2, pp. 21—36 (in Russian).

5. **Afonin S., Bonushkina A.** Validation of Safety-Like Properties for Entity-Based Access Control Policies, *Advances in Soft and Hard Computing*, 2019, pp. 259—271.

6. **Harrison M. A., Ruzzo W. L., Ullman J. D.** Protection in Operating Systems, *Communications of the ACM*, 1976, vol. 19, no. 8, pp. 461—471.

7. **Moskowitz I. S., Costich O. L.,** A classical automata approach to noninterference type problems, *Proc. Computer Security Foundations Workshop V,* 1992, pp. 2—8.

8. **Grusho A. A., Shumitskaya E. L.** Non-interference models and covert channels, *Diskretnaya Matematika*, 2002, vol. 14, no. 1, pp. 11—16 (in Russian).

Связь 2020
32-я международная выставка «Информационные и коммуникационные технологии»

с 21 по 24 апреля 2020, Россия, Москва, ЦВК «Экспоцентр»

Темы и тренды выставки

Умный город	Программное обеспечение. Российский софт
IOT Tech	IP-технологии
5G	Телевидение и радиовещание
Телекоммуникационное оборудование, решения, услуги	AR&VR
Сети передачи данных	Искусственный интеллект
Телекоммуникационная и сетевая инфраструктура	IT-услуги
Спутниковая связь	Мобильные платежи
Радиосвязь	Интернет-технологии и услуги
Мобильная связь	Стартапы
Кабели связи, оборудование	Работа и карьера в IT и телекоммуникационных отраслях
ЦОДы. Оборудование, софт, решения, услуги	Микроэлектронные компоненты для телекоммуникаций
Системы электропитания	Навигационные системы, технологии и услуги
Smart Device Show	

Подробности: https://www.sviaz-expo.ru/

УДК 004.4'418 DOI: 10.17587/prin.11.47-53

С. С. Сосинская, канд. техн. наук, проф., sosinskaya@mail.ru,
В. В. Христюк, магистрант, just_smile08@mail.ru,
Иркутский национальный исследовательский технический университет

Фреймовая модель описания технологий комплексных деталей и ее преобразование в объектную модель

Предложен подход к описанию информационной модели разработки технологических операций типового процесса изготовления деталей. При построении типовых процессов механической обработки за основу берется комплексная деталь. Комплексная деталь применяется для краткого описания представления информации о группе деталей. Это либо основные элементы реальной, наиболее сложной детали группы, либо условная деталь, содержащая в себе информацию обо всех деталях данной группы. На основе представленной предметной области формируется объектная модель как совокупность взаимосвязанных классов, которая отображается во фреймовую модель на базе экспертной оболочки KAPPA. В связи с рядом недостатков этой экспертной оболочки фреймовая модель затем преобразуется в реляционную модель через промежуточную объектную модель. Такая система преобразований использует ряд современных программных средств и, как представляется авторам, обеспечивает более высокую степень автоматизации и в результате — представление более подробной информации, чем это делается в настоящее время на предприятиях.

Ключевые слова: *объектная модель, комплексная деталь, контекстно-свободная грамматика, реляционная модель, групповая обработка, система ANTLR, Object-Relational Mapping, объектно-ориентированный подход*

Введение

В настоящее время вся информация о выпускаемых изделиях и составляющих их узлах организуется в виде цифрового макета объектов на всех стадиях жизненного цикла их функционирования — системы *Product Lifecycle Management* (PLM). В разработке одного проекта могут участвовать специалисты, находящиеся в разных городах, а не только в разных подразделениях одного предприятия.

Представление информации в цифровом, наглядном, часто в графическом виде позволяет специалистам разных специальностей понимать друг друга, при необходимости вносить изменения и на основе накопленной информации формировать различные отчеты.

В состав PLM-системы входят следующие элементы.

- Система управления жизненным циклом изделия — набор средств и настроек для представления цифрового макета на различных этапах создания и существования изделия: конструирование, производство, обслуживание, утилизация.
- Атрибутивные данные — данные, характеризующие и описывающие элементы цифрового макета. Например, для разработанной на данном предприятии детали атрибутивными данными будут: имя и отдел разработчика, материал, вес, набор и значения контролируемых параметров.
- Технологические данные, содержащие необходимые указания для производства: используемые инструменты, материалы, технологии, средства контроля и т. д.

Однако объем информации цифровых макетов может быть сокращен за счет применения идеи типизации технологических процессов, которую впервые предложил и разработал д-р техн. наук, проф. А. П. Соколовский [1]. Созданная им методика типизации технологических процессов базируется на классификации процессов, в основе которой лежит классификация деталей. Приведем основные положения, на которых базируется его подход. Типом детали называется совокупность сходных деталей, имеющих в данных производственных условиях общий технологический процесс. Следовательно, конечная цель классификации деталей — установление типов деталей. Целью же разработки типовых технологических процессов является систематизация технологических процессов для обработки однотипных деталей. Технологический процесс разрабатывается для каждого типа детали. В пределах одного типа допускается расхождение в маршрутах операций за счет добавления или исключения некоторых операций. По методике А. П. Соколовского классификация ведется по технологическим процессам изготовления деталей. Многолетний опыт показывает, что в основном должна быть использована типизация на базе сочетания типового технологического процесса с классификацией реальных деталей. Цель типизации — стандартизовать технологический процесс и добиться, чтобы обработка сходных деталей осущест-

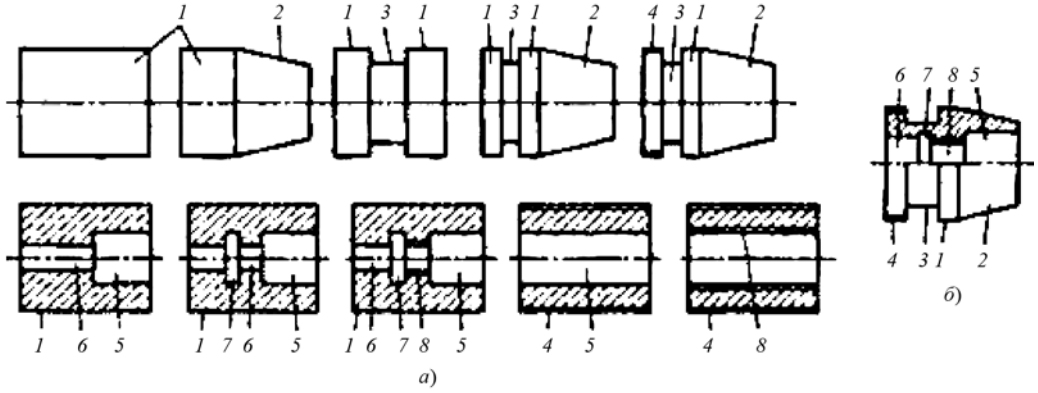

Рис. 1. Пример создания комплексной детали для группы деталей, отличающихся разнообразием обрабатываемых поверхностей:

а — реальные детали; *б* — комплексная деталь; основные поверхности: *1* — цилиндрическая поверхность; *2* — конус; *3* — канавка; *4* — наружная резьба; *5* — внутренняя цилиндрическая поверхность; *6* — отверстие малого диаметра; *7* — внутренняя канавка; *8* — внутренняя резьба

влялась с помощью общих, наиболее совершенных и эффективных методов. Типовые технологические процессы разрабатываются для всех технологических операций всех типов деталей, а также для стандартных и унифицированных узлов и отдельных изделий, выпускаемых конкретным предприятием. Разработка типовых процессов для стандартных деталей достаточно проста. Эти процессы сравнительно стабильны и характеризуются высоким уровнем оснащенности. Типовые технологические процессы, характерные для данного предприятия, охватывают детали, имеющие одинаковый технологический маршрут обработки, однотипное оборудование и технологическую оснастку.

Основой построения типовых технологических процессов является конструктивное сходство деталей. При построении типовых процессов механической обработки за основу берется комплексная деталь — это реальная, наиболее сложная в данной группе деталь, либо условная деталь, спроектированная как совокупность основных элементов всех деталей данной группы. Пример комплексной детали приведен на рис. 1. Под основными элементами понимаются поверхности, образующие конструкцию детали, а также операции, выполняемые в процессе обработки [2].

Технологический процесс проектируется для комплексной детали в целом и с небольшими подналадками применяется для изготовления любой детали данной группы [3, 4].

Статья посвящена разработке объектной модели описания технологии изготовления комплексной детали и автоматизации процессов преобразования хранимой информации из представления в формате фреймов экспертной оболочки (ЭО) KAPPA [5] в компьютерное представление иерархии классов объектной модели.

Актуальность выбранного подхода заключается, по мнению авторов, в расширении возможностей хранения необходимой информации по сравнению с существующими подходами. Прежде всего автоматизация хранения информации выполняется на всех этапах решения задачи. Кроме того, следует принимать во внимание то обстоятельство, что ЭО KAPPA не функционирует на современных версиях операционных систем и не имеет встроенных инструментов для сохранения созданных в ней моделей в реляционных базах данных.

Подход, в той или иной степени близкий к рассматриваемому, изложен в работах [6—11].

Этапы решения задачи

Авторами реализовано решение поставленной задачи в виде многоэтапной процедуры, представленной на рис. 2.

На этапе 1 на основе информации, необходимой для решения задачи, разработана объектная модель. Затем на этапе 2 реализована фреймовая модель для построения типового технологического процесса на основе программного инструмента KAPPA. На этапе 3 описана контекстно-свободная (КС) грамматика языка KAL, используемая для описания объектной модели типового технологического процесса. На этапе 4 сгенерированы классы языка C#. Эти классы содержат код лексического и синтаксического анализаторов и код для интерпретатора, конвертирующего фреймовую модель в иерархию классов языка C#. На этапе 5 классы преобразуются в таблицы реляционной базы данных.

Модель описания типового технологического процесса

Анализ предметной области задачи показывает, что необходимо хранить информацию о группах деталей, о входящих в их состав деталях, о материалах, из которых состоят детали, о технологических операциях изготовления деталей, о станках и оборудовании, на которых обрабатываются детали, и о составе конкретных групповых технологических процессов для комплексных деталей.

На основе анализа информации, необходимой для описания групповых технологических процессов,

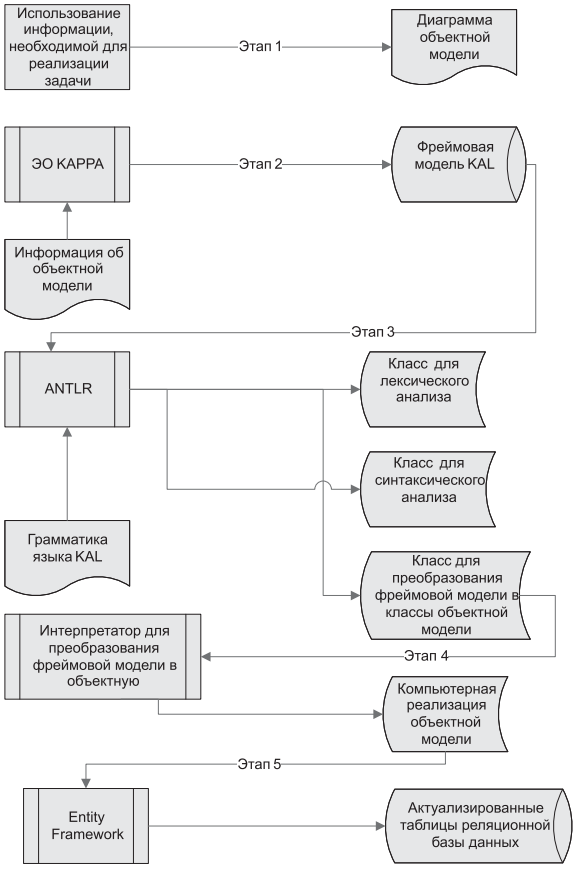

Рис. 2. Этапы решения задачи

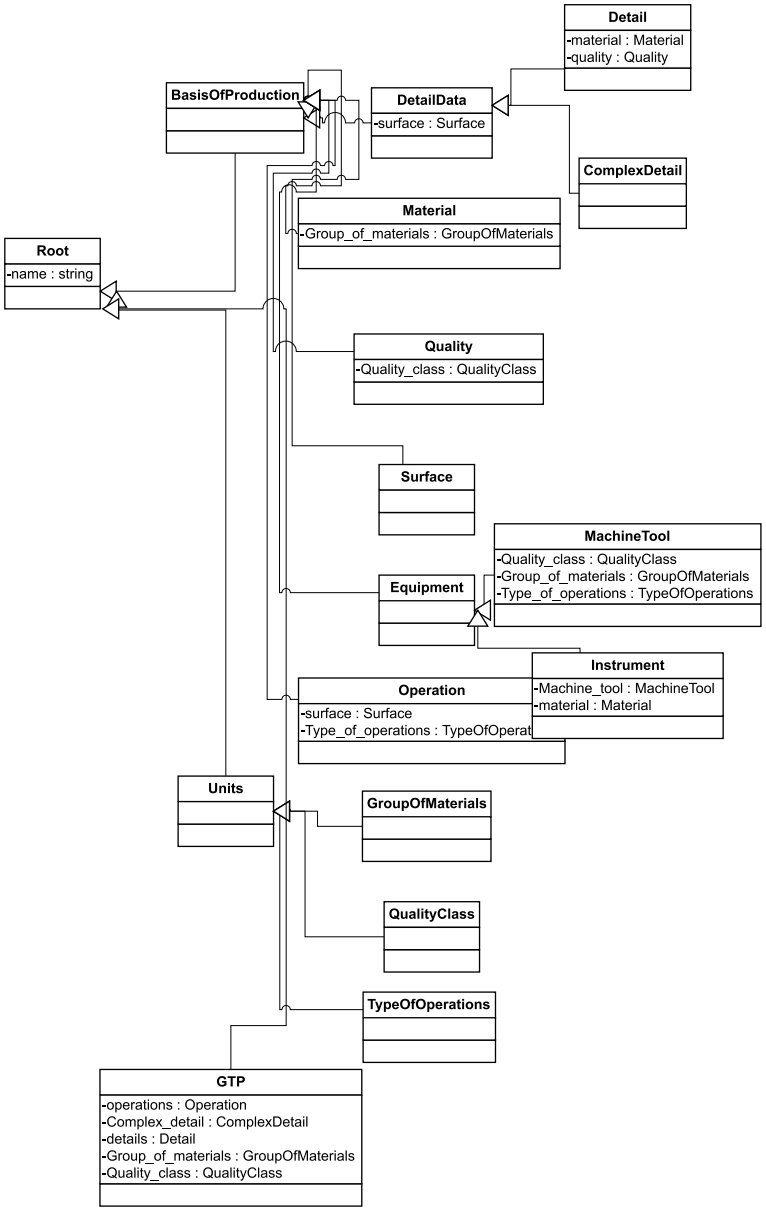

Рис. 3. Объектная модель информации для типового технологического процесса

авторами впервые была разработана объектная модель классов, образующих иерархию (рис. 3).

Класс Root является корнем любой иерархии классов.

На первом уровне иерархии представлены три класса: Units (объединения); BasisOfProduction (основа производства); GTP (групповой технологический процесс).

Класс Units является родительским для справочников: GroupOfMaterials (группы материалов), QualityClass (класс качества) и TypeOfOperations (тип операций).

Потомки класса BasisOfProduction содержат информацию, необходимую для последующего описания деталей и операций, входящих в состав группового технологического процесса.

Material (материал) — класс, содержащий информацию об используемых материалах. Каждый материал имеет название и ссылку на группу материалов, к которой он относится.

Quality (качество) — класс, содержащий информацию о требуемом качестве изготавливаемой детали. Экземпляры этого класса содержат название и ссылку на класс качества, к которому они относятся.

Surface (поверхность) — экземпляры этого класса есть поверхности, входящие в состав деталей.

Класс DetailData содержит информацию о деталях. Детали в модели могут быть двух типов: комплексная деталь (ComplexDetail) и обычная деталь (Detail). Эти классы содержат информацию о названии той или иной детали и поверхностях, из которых эти детали состоят.

Detail — дополняется информацией о материале и требуемом качестве изготавливаемой детали. Таким образом, детали, которые соответствуют одной и той же комплексной детали, могут быть отнесены в разные группы в зависимости от типа материала и класса требуемого качества.

Класс Equipment (оборудование) определяет оборудование, на котором происходит обработка деталей. Потомками этого класса являются MachineTool (станок) и Instrument (инструмент).

MachineTool (станок) — экземпляры этого класса есть станки, на которых происходит обработка деталей. Каждый станок может выполнять опреде-

ленный тип операций с требуемым классом качества для определенной группы материалов.

Instrument (инструмент) — экземплярами этого класса являются инструменты, которые могут быть применены на отдельном станке и для конкретного материала.

Operation (операция) — класс определяет действия, которые выполняются при обработке конкретной поверхности. Каждая операция относится к определенному типу.

Класс GTP содержит operations — те операции, которые используются в рамках конкретного типового технологического процесса, и информацию о группах деталей, входящих в состав комплексной детали, группах материалов и классах качества.

Разработка фреймовой модели

Предложенная объектная модель была реализована в виде фреймовой модели в ЭО KAPPA.

Концепция фреймов была разработана в 70-х гг. XX века американским ученым, профессором Массачусетского технологического института Марвином Минским для представления знаний. Минский определил фрейм как структуру данных для представления некоторого концептуального объекта. Информация, относящаяся к фрейму, содержится в составляющих его слотах. Слот может быть терминальным (листом иерархии) или представлять собой фрейм нижнего уровня. Особенностью этого подхода стало объединение в одной структуре как декларативных знаний об объектах, их свойствах и состояниях, так и процедурных знаний о поведении объектов, о методах извлечения информации и достижения целей. ЭО KAPPA ориентирована на работу с пользователем, не являющимся профессиональным программистом. Необходимость предоставления разработчику экспертной системы разнообразных инструментов реализации и интерпретации знаний обусловило включение в состав ЭО KAPPA языка высокого уровня, являющегося средством комплексирования компонентов. Компоненты предметной области представляются такими структурами, как "фрейм-прототип" и "фрейм-экземпляр". Для описания свойств фрейма используются слоты — переменные, которые могут принимать определенные значения. Одним из важнейших свойств ЭО KAPPA является возможность реализации механизма наследования. Таким образом, одни фреймы-прототипы являются потомками других в соответствии с логикой объектной модели. У каждого фрейма-прототипа есть экземпляры, которые также являются его потомками, т. е. содержат те же слоты, что и фрейм-прототип, но с различными значениями (например, экземпляры деталей). Общие для фрейма слоты могут быть определены 1 раз во фрейме-прототипе, все потомки которого будут их наследовать. Взаимоотношения между этими структурами представляются связями, в результате чего получается иерархия фреймов.

Общий вид иерархии наследования фреймов-прототипов для модели группового технологического процесса, созданной с помощью ЭО KAPPA, показан

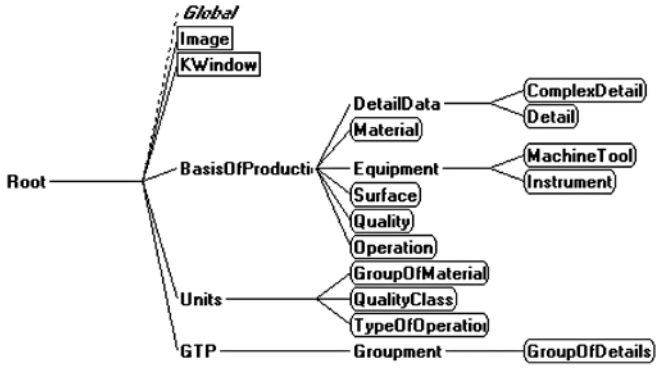

Рис. 4. Иерархия фреймов в ЭО KAPPA

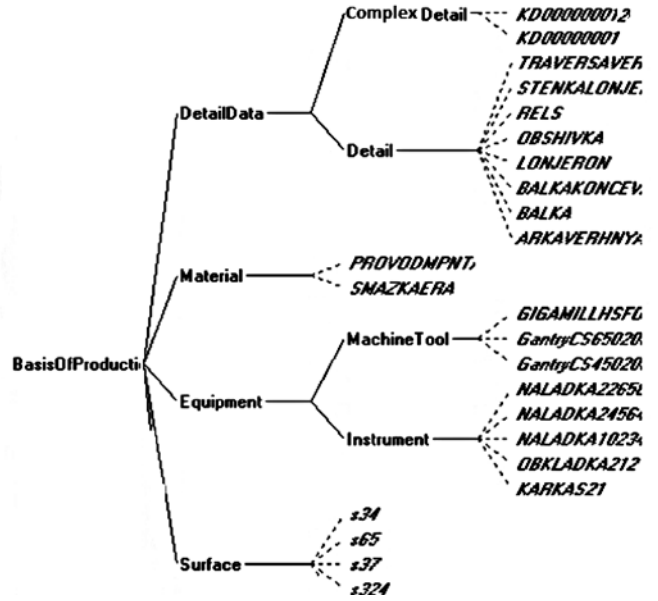

Рис. 5. Фреймовая модель с экземплярами

на рис. 4. Классы-листья обведены овальной рамкой. Имена фреймов иерархии совпадают с именами классов объектной модели, а связи соответствуют отношениям объектной модели. Фреймы Image и KWindow присутствуют в иерархии фреймов изначально и нужны для функционирования системы, фрейм Global создается для хранения информации, доступной для всего приложения, он выделен курсивом. В иерархию фреймов входит три уровня.

На рис. 5 показан фрагмент объектной модели с конкретными фреймами-экземплярами для большинства фреймов-прототипов, которые добавляются на уровне 4.

4. Разработка грамматики языка KAL с помощью инструментального средства ANTLR

Прежде чем приступить к созданию интерпретатора следует описать формальный язык. Как известно, язык — это заданный набор символов и правил, устанавливающих способы комбинации этих символов для записи правильных цепочек. Любой язык

содержит лексику, синтаксис и семантику. Лексика — алфавит, т. е. множество допустимых символов. Синтаксис — набор правил, определяющих допустимые конструкции языка. Семантика — раздел языка, определяющий содержание языка, т. е. значение предложений.

Формальной грамматикой G называется совокупность четырех объектов:

$$G = \{V_T,\ V_N,\ V_0 \in V_N,\ P\},$$

где V_T — терминальный алфавит (элементы этого алфавита называются терминальными символами; из них строятся цепочки, порождаемые грамматикой); V_N — нетерминальный, вспомогательный алфавит (элементы этого алфавита используются при построении правил вывода; они могут входить в промежуточные цепочки, но не должны входить в результат порождения); V_0 — начальный нетерминальный символ грамматики; P — множество правил вывода или порождающих правил вида $\alpha \to \beta$, где α и β — цепочки, построенные из элементов алфавита $V_T \cup V_N$, который называют полным алфавитом грамматики G. Символ $\to$ понимается как "можно заменить на".

Грамматика языка KAL, используемого при построении фреймовой модели, была описана с помощью инструментального средства ANTLR [12].

ANTLR (от ANother Tool for Language Recognition — еще один инструмент для распознавания языков) — это инструментарий для создания компиляторов или интерпретаторов языков программирования.

Создателем, основным идеологом и разработчиком ANTLR, а также ряда связанных инструментов, таких как ANTLRWorks (среда разработки для ANTLR), является профессор в области компьютерных наук университета Сан-Франциско Теренс Парр (Terence Parr).

Далее перечислены операции, используемые ANTLR:

* — встречается 0 или более раз;

\+ — встречается 1 или более раз;

? — может не быть;

() — группировка правил.

| — выбор одной альтернативы из нескольких.

Описание грамматики языка KAL средствами ANTLR, впервые выполненное авторами, представлено на рис. 6. В этой грамматике описаны правила для элементов как терминального, так и нетерминального алфавитов (терминалов и нетерминалов). Терминалы по правилам ANTLR обозначены заглавными латинскими буквами, нетерминалы — малыми буквами. Назначение терминалов поясняется на рис. 6. Правила для нетерминалов указывают, из каких терминалов и нетерминалов состоят другие нетерминалы. В частности, программа может содержать один или более операторов (присваиваний, разветвлений, циклов, вызовов функций и тел функций, слотов с их именами и типами и т. д.).

На основании этой грамматики ANTLR генерирует классы лексического и синтаксического блоков для интерпретатора преобразования фреймовой модели в набор классов.

V_T : {NUMBER (число), IDENT (идентификатор), WS (пробельные символы), COMMENT (комментарий), STRING (строка), ADD (+), SUB (-), MUL (*), DIV (/), LETTER (буква), DIGIT (цифра)}

V_N : {program, stmt, param_list, function_body, expr_variable, function, expr_stmt, expr, mult, atom, for_stmt, iter_param, if_stmt, if_for_action, usl_stmt, uslovie, operatop_sravn, logical_operation, param, frame_slot, equiv}

V_0 : program P : {

1. program → (stmt';')+
2. stmt → expr_stmt | function
3. param_list → '[' IDENT+ ']'
4. function_body → '{' (expr_variable ';')+ '}'
5. expr_variable → expr_stmt | for_stmt | if_stmt | function
6. function → IDENT '(' param? (',' param)* ')'
7. expr_stmt → frame_slot (equiv expr)?
8. equiv → '=' | '+=' | '-=' | '*=' | '/='
9. expr → mult ((ADD | SUB) mult)*
10. mult → atom ((MUL | DIV) atom)*
11. atom → STRING|NUMBER|IDENT|frame_slot|function|'(' expr ')'| 'TRUE' | 'FALSE'
12. for_stmt → 'For' IDENT iter_param if_for_action
13. iter_param → '[' NUMBER (NUMBER | frame_slot | ('(' function ')')) NUMBER ']'
14. if_stmt → 'If' usl_stmt 'Then' if_for_action ('Else' if_for_action)?
15. if_for_action → function_body | function | expr_stmt |'TRUE' | 'FALSE'
16. usl_stmt → ('(' uslovie (logical_operation uslovie)* ')') | uslovie
17. uslovie → (function | frame_slot) (operator_sravn (function | IDENT | NUMBER | 'TRUE' | 'FALSE'))?
18. operator_sravn → '==' | '#=' | '!=' | '>=' | '<=' | '>' | '<'
19. logical_operation → 'Or' | 'And'
20. param→ expr |atom | param_list | function_body
21. frame_slot → IDENT (':' IDENT)+ }

Рис. 6. Грамматика языка KAL в системе ANTLR

5. Разработка интерпретатора преобразования фреймовой модели в набор классов языка C#

Для преобразования фреймовой модели типового технологического процесса в набор классов было разработано приложение на языке C# Visual Studio, включающее классы Lexer и Parser, сгенерированные на основе грамматики ANTLR.

В грамматике ANTLR есть возможность вставить действия (Action) на целевом языке программирования непосредственно в текст грамматики. При компиляции программы этот код переносится в текст синтаксического анализатора. Действия выделяются знаками { и }.

Кроме того, в приложение был включен класс Emitter, содержащий

методы, которые во время работы синтаксического блока (Parser) генерируют код для сохранения фреймов в виде иерархии классов.

Функции класса Emitter:

— создание объектов типа ClassFromKappa на основе фреймов-прототипов;

— создание свойств класса на основе слотов фрейма-прототипа;

— создание экземпляров классов на основе фреймов-экземпляров;

— заполнение свойств экземпляров классов на основе данных слотов фреймов;

— генерация кода программы, сохраняющей данные в базе данных.

Различные методы класса Emitter отвечают за создание отдельных блоков программы. Так, метод PrintAllClasses отвечает за добавление в создаваемую программу описаний классов, созданных на основе фреймов. Метод PrintAllInstance выполняет задачу генерации кода по созданию всех экземпляров классов и т. д.

Преобразование классов в таблицы реляционной базы данных

Для преобразования сгенерированных классов в таблицы реляционной базы данных был использован инструментарий фирмы Microsoft Entity Framework [12]. Он именуется ORM (*Object-Relational Mapping*) и представляет собой библиотеку, предназначенную для реализации подхода, при котором устанавливается соответствие между объектами, используемыми в приложении, и таблицами, хранящимися в реляционных базах данных.

Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из базы данных. При этом не только извлекаются определенные строки таблиц базы данных, но и получаются объекты, связанные различными ассоциативными связями. Другим ключевым понятием является EntityDataModel. Эта модель ставит в соответствие классы сущностей и таблицы в базе данных.

После завершения работы программы-интерпретатора был получен код новой программы, содержащий описание классов, соответствующих фреймам модели, а также код для создания экземпляров этих классов и их сохранения в базе данных. После компиляции и запуска программа создает в базе реляционные таблицы на основе объектно-ориентированной модели и сохраняет строки таблиц.

На рис. 7 показан фрагмент взаимосвязанных данных реляционной модели, полученных на основе фреймов в ЭО КАРРА.

Комплексная деталь	Группа деталей	Деталь
KD-000-00-001	group5	Arka verhnja shpangouta 1 11.0201.1.801.900
KD-000-00-001	group5	Balka koncevaja 130.01.2085.0003.001/002
KD-000-00-001	group5	Lonjeron 130.11.3400.1001.000
KD-000-00-001	group5	Obshivka 11.3130.3.811.001/002
KD-000-00-001	group5	Rels 130.11.0209.1001.001/002
KD-000-00-001	group5	Stenka Lonjerona 11.1106.4.152.001/002
KD-000-00-001	group5	Traverse verhnja 130.11.0321.1003.001
KD-000-00-002	group3	Arka verhnja shpangouta 1 11.0201.1.801.900
KD-000-00-002	group3	Balka koncevaja 130.01.2085.0003.001/002
KD-000-00-002	group3	Lonjeron 130.11.3400.1001.000
KD-000-00-002	group3	Obshivka 11.3130.3.811.001/002
KD-000-00-002	group3	Rels 130.11.0209.1001.001/002
KD-000-00-002	group3	Stenka Lonjerona 11.1106.4.152.001/002
KD-000-00-002	group3	Traverse verhnja 130.11.0321.1003.001
KD-000-00-002	group4	Arka verhnja shpangouta 1 11.0201.1.801.900
KD-000-00-002	group4	Balka koncevaja 130.01.2085.0003.001/002
KD-000-00-002	group4	Lonjeron 130.11.3400.1001.000
KD-000-00-002	group4	Obshivka 11.3130.3.811.001/002
KD-000-00-002	group4	Rels 130.11.0209.1001.001/002
KD-000-00-002	group4	Stenka Lonjerona 11.1106.4.152.001/002
KD-000-00-002	group4	Traverse verhnja 130.11.0321.1003.001
KD-000-00-003	group1	Arka verhnja shpangouta 1 11.0201.1.801.900
KD-000-00-003	group1	Balka koncevaja 130.01.2085.0003.001/002
KD-000-00-003	group1	Lonjeron 130.11.3400.1001.000
KD-000-00-003	group1	Obshivka 11.3130.3.811.001/002
KD-000-00-003	group1	Rels 130.11.0209.1001.001/002
KD-000-00-003	group1	Stenka Lonjerona 11.1106.4.152.001/002
KD-000-00-003	group1	Traverse verhnja 130.11.0321.1003.001

Рис. 7. Фрагмент взаимосвязанных данных, хранимых в реляционной модели

Заключение

Описан и реализован подход описания информационных структур, связанных с разработкой типового технологического процесса. Впервые выполнена разработка объектной модели и реализованы все этапы ее преобразования в реляционную модель. Для выполнения поставленной задачи в полном объеме авторами были применены совместно несколько известных технологий, а именно фреймовая оболочка KAPPA, система ANTLR, инструмент ORM.

Список литературы

1. **Соколовский А. П.** Научные основы технологии машиностроения. М.: Машгиз, 1955. 515 с.
2. **Митрофанов С. П., Куликов Д. Д., Миляев О. Н., Падун Б. С.** Технологическая подготовка гибких производственных систем. Л.: Машиностроение, 1987. 352 с.
3. **Основы** технологий информационной поддержки изделий машиностроения: учеб. пособие / В. В. Морозов и др.; Владим. гос. ун-т. Владимир: Изд-во Владим. гос. ун-та, 2009. 252 с.
4. **Богодухов С. И., Схиртладзе А. Г., Сулейманов Р. М., Проскурин А. Д.** Технологические процессы в машиностроении: учебник / под общ. ред. проф., д-ра техн. наук С. И. Богодухова. Старый Оскол: ТНТ, 2012. 624 с.
5. **Сидоркина И. Г.** Системы искусственного интеллекта: учеб. пособие. М.: КНОРУС, 2015. 248 с.

6. **Fong J. S. P.** Information Systems Reengineering, Integration and Normalization.Springer, 2015. 391 p.
7. **Cheng Z.** Formal Verification of Relational Model Transformations using an Intermediate Verification Language. Dissertation for the degree of Doctor of Philosophy. Maynooth University 2016. URL: http://mural.maynoothuniversity.ie/7089/1/ZCHENG-THESIS-FINAL.pdf
8. **Cheng Z.** A Proposal for a Generic Translation Framework for Boogie Language // 26th European Conference on Object-Oriented Programming (Doctoral Symposium), 2012. URL: http://www.cs.nuim.ie/~zcheng/paper/ECOOP2012.pdf
9. **Cheng Z., Monahan R., Power J. F.** A sound execution semantics for ATL via translation validation // In 8th International Conference on Model Transformation. L'Aquila, Italy, Springer, 2015. P. 133—148.
10. **Cheng Z., Monahan R., Power J. F.** Verifying SimpleGT Transformations Using an Intermediate Verification Language // In 4th International Workshop on the Verification Of Model Transformation, L'Aquila, Italy. CEUR, 2015. P. 12—19. URL: http://ceur-ws.org/Vol-1530/paper3.pdf
11. **Antlr** Documentation. URL: http://www.antlr.org/
12. **EntityFramework** Documentation. URL: https://msdn.microsoft.com/enus/library/ee712907(v=vs.113).aspx

The Frame Model for Describing the Technology of Complex Detail and its Transformation into an Object Model

S. S. Sosinskaya, sosinskaya@mail.ru, **V. V. Hristyuk,** just_smile08@mail.ru, National Research Irkutsk State Technical University, Irkutsk, 664074, Russian Federation

Corresponding author:

Sosinskaya Sophia S., Professor, National Research Irkutsk State Technical University, Irkutsk, 664074, Russian Federation
E-mail: sosinskaya@mail.ru

Received on July 27, 2019
Accepted on September 05, 2019

An approach is proposed to describe the information model for the development of technological operations of a typical process for manufacturing parts. When building typical machining processes, a complex part is taken as the basis. A complex part is used to briefly describe the presentation of information about a group of parts — these are either the main elements of the real, most complex part of the group, or conditional, containing information about all the details of this group. Based on the presented subject area, an object model is formed as a set of interconnected classes, which is mapped into a frame model based on the KAPPA expert shell (ES). Due to a number of drawbacks of this ES, the frame model is then converted into a relational model through an intermediate object model. Such a transformation system uses a number of modern software tools and, as it seems to the authors, provides a higher degree of automation and, as a result, provides more detailed information than is currently done in enterprises.

Keywords: *object model, complex detail, context-free grammar, relational model, group processing, ANTLR system, Object-Relational Mapping, object-oriented approach*

For citation:

Sosinskaya S. S., Hristyuk V. V. The Frame Model for Describing the Technology of Complex Detail and its Transformation into an Object Model, *Programmnaya Ingeneria*, 2020, vol. 11, no. 1, pp. 47—53.

DOI: 10.17587/prin.11.47-53

References

1. **Sokolovskij A. P.** *Scientific foundations of engineering technology*, Moscow, Mashgiz, 1955, 515 p. (in Russian).
2. **Mitrofanov S. P., Kulikov D. D., Milyaev O. N., Padun B. S.** *Technological preparation of flexible production systems*, Leningrad, Mashinostroenie, 1987, 352 p. (in Russian).
3. ***The basics*** of information technology support for engineering products: tutorial / V. V. Morozov et al.; Vladimir, Izd-vo Vladim. gos. un-ta, 2009, 252 p. (in Russian).
4. **Bogoduhov S. I., Shirtladze A. G., Sulejmanov R. M., Proskurin A. D.** *Technological processes in mechanical engineering*: textbook /Eds S. I. Bogoduhov, Starii Oskol, TNT, 2012, 624 p. (in Russian).
5. **Sidorkina I. G.** *Artificial Intelligence Systems*: Tutorial. Moscow, KNORUS, 2015, 248 p. (in Russian).
6. **Fong J. S. P.** *Information Systems Reengineering, Integration and Normalization,* Springer, 2015, 391 p.
7. **Cheng Z.** Formal Verification of Relational Model Transformations using an Intermediate Verification Language, 2016, available at: http://mural.maynoothuniversity.ie/7089/1/ZCHENG-THESIS-FINAL.pdf
8. **Cheng Z.** A Proposal for a Generic Translation Framework for Boogie Language. *26th European Conference on Object-Oriented Programming (Doctoral Symposium),* 2012, available at: http://www.cs.nuim.ie/~zcheng/paper/ECOOP2012.pdf
9. **Cheng Z., Monahan R., Power J. F.** A sound execution semantics for ATL via translation validation, *In 8th International Conference on Model Transformation*, L'Aquila, Italy, Springer, 2015, pp. 133—148.
10. **Cheng Z., Monahan R., Power J. F.** Verifying SimpleGT Transformations Using an Intermediate Verification Language, *In 4th International Workshop on the Verification of Model Transformation*, L'Aquila, Italy, CEUR, 2015, pp. 12—19, available at: http://ceur-ws.org/Vol-1530/paper3.pdf
11. **Antlr** Documentation, available at: http://www.antlr.org/
12. **EntityFramework** Documentation, available at: https://msdn.microsoft.com/enus/library/ee712907(v=vs.113).aspx

УДК 004.89 DOI: 10.17587/prin.11.54-64

С. В. Вычегжанин, инженер-программист, vychegzhaninsv@gmail.com,
Вятский государственный университет, Киров

Программная система распознавания точки зрения автора текста на основе композиционного подхода

Рассмотрена задача автоматического анализа текстов в целях распознавания точек зрения их авторов (stance detection). Предложена структура программной системы с описанием ее подсистем и их назначения. С помощью диаграммы классов представлен вариант программной реализации системы. Эффективность разработанной системы подтверждена рядом экспериментов, проведенных с использованием трех корпусов по разным предметным областям.

Ключевые слова: *обработка естественного языка, распознавание точки зрения автора текста, программная система, диаграмма классов*

Введение

В социальных медиа, таких как социальные сети, блоги, интернет-форумы и т. п., содержится большое число текстовых сообщений, в которых пользователи свободно выражают свою точку зрения о каком-либо событии. Например, коллекция сообщений о реформе образования в России может содержать различные точки зрения о введении единого государственного экзамена (ЕГЭ) в школах, что позволяет выделить его положительные и отрицательные стороны и в целом оценить позицию общества по данному вопросу. Извлечение такой информации из огромного массива текстовых данных является ключевой проблемой, стоящей перед перспективным направлением компьютерной лингвистики, получившим название *распознавание точки зрения автора текста* (*stance detection*) [1].

Задача распознавания точки зрения (позиции) автора текста заключается в автоматическом определении по тексту, является автор сторонником целевого объекта, по отношению к которому выражено мнение, или же противником данного объекта, или же отношение автора к объекту нейтральное. Целевым объектом могут быть персона, организация, государственная политика, общественное движение, продукт и т. п.

Выделяют следующие основные классы позиций [2, 3].

1. Позиция *за* (*for*) — по тексту можно определить, что автор высказывается в поддержку целевого объекта.

2. Позиция *против* (*against*) — по тексту можно определить, что автор высказывается против целевого объекта.

3. Позиция *нейтрально* (*neutral*) — из текста можно сделать вывод о наличии нейтральной позиции, при этом она может быть указана явно ("*Я нейтрально отношусь к ...*"), либо автор приводит соображения в пользу *за* и *против*, не склоняясь ни к одной из позиций.

4. *Невозможность (отсутствие возможности) определения точки зрения* (*neither*) — из текста нельзя определить позицию автора, например, в том случае, когда не приводятся аргументы ни *за*, ни *против*; этот класс часто объединяется с классом *нейтрально.*

5. *Согласие с предыдущей точкой зрения* (*observing*) — в тексте повторяется предыдущее мнение.

Задача определения позиции автора текста тесно связана с задачей *анализа тональности* (*sentiment analysis*), для решения которой активно разрабатывают новые подходы и программные средства, например [4, 5]. Тональностью называется выраженная в тексте эмоциональная оценка (позитивная, негативная или нейтральная). Однако данные задачи имеют следующие отличия:

— позиция и тональность не всегда совпадают, например, в тексте может быть выражена позиция *за*, но тональность по отношению к объекту негативная: "*Терпеть не могу эту партию, но буду за нее голосовать — больше не за кого*";

— в задаче распознавания точки зрения автора текста целевой объект всегда присутствует, в задаче анализа тональности текста целевого объекта может не быть.

Для оценки качества классификации обычно используются значения *точности* (*precision*, *P*), *полноты* (*recall*, *R*) и *F1-меры* (*F1-measure*, *F*1), вычисляемые по следующим формулам относительно класса *s*:

$$P = \frac{TP}{TP + FP};\ \ R = \frac{TP}{TP + FN};\ \ F1 = \frac{2PR}{P + R},$$

где *TP* (*true positive*) — число текстов, которые классификатор правильно классифицировал как принадлежащие классу *s*; *FP* (*false positive*) — число текстов, которые классификатор правильно классифицировал как не принадлежащие классу *s*; *FN* (*false negative*) — число текстов, которые классификатор неправильно классифицировал как не принадлежащие классу *s*.

Существующие методы определения позиции автора текста демонстрируют недостаточно высокое качество решения задачи. По результатам соревнования, проводимого в рамках международного семинара SemEval-2016 [3], посвященного оценке систем семантического анализа, победителем стала система организаторов на основе метода опорных векторов (*Support Vector Machine*, SVM) и символьных *n*-грамм, получившая для классов позиций *за* и *против* среднее значение $F1 = 0{,}69$. Среди участников первое место заняла система MITRE, основанная на рекуррентных нейронных сетях и распределенных представлениях слов, с результатом $F1 = 0{,}68$. В работе [1] представлен сравнительный обзор исследований, в котором наибольшее значение $F1 = 0{,}74$ получено с помощью метода, основанного на SVM и использовании лексической базы данных WordNet.

С учетом изложенного выше актуальным направлением исследований является разработка методов и средств, позволяющих получить высокое качество решения рассматриваемой задачи. Примерами практического применения методов распознавания точки зрения автора текста может служить анализ мнений избирателей о политических партиях в ходе предвыборной кампании, а также маркетинговые исследования отзывов потребителей товаров и услуг в интересах производителей.

Формально постановка задачи распознавания точки зрения автора текста в социальных медиа может быть сформулирована следующим образом [1, с. 41101]: заданы целевой объект t_e и текстовый корпус C_t^e, для каждого текста $text_i \in C_t^e$ требуется определить позицию

$$s_i^{t_e} \in \{for, neutral, against\}.$$

Следует отметить, что определение позиции автора текста осложняется приведением автором доводов в пользу своей позиции и против другой позиции [6], сменой автором точки зрения на протяжении дискуссии [7], отсутствием упоминания целевого объекта в тексте [8], а также использованием иронии, сарказма и других риторических приемов [9].

В зависимости от учета связей между сообщениями выделяют два основных подхода к решению задачи распознавания точки зрения автора текста:

1) каждое сообщение рассматривается с учетом дискурса и связей с другими сообщениями, как правило, с помощью графов [9—14];

2) сообщение изучается изолированно от других сообщений [3, 6, 15].

Для решения задачи определения позиции автора текста часто используют [1]:

1) стандартное машинное обучение с учителем (*supervised learning*) [16—24];

2) слабое машинное обучение с учителем (*weakly supervised learning*) [16, 23, 25, 26].

В работах, использующих стандартное машинное обучение с учителем, исследуются техники традиционного конструирования признаков (например, извлечение лексических, синтаксических и семантических признаков) и глубокого обучения (например, сверточная нейронная сеть, рекуррентная нейронная сеть, долгая краткосрочная память). В работах [17—20] в качестве классификатора применяется SVM и исследуются текстовые признаки и признаки на основе зависимостей. В работе [20] авторы в качестве признаков используют эмоционально окрашенные слова. При этом они учитывают отношения зависимости, построенные с помощью парсера Stanford, осуществляя поиск пар зависимых слов, одно из которых эмоционально окрашенное, а другое — *за* или *против*. В работе [17] для учета семантической информации в текстах используются скрытые семантические признаки, полученные с помощью латентного размещения Дирихле или факторизацией взвешенной текстовой матрицы. Авторы работы [19] предложили подход, в основе которого лежит использование признаков, полученных с помощью программы контент-анализа текста LIWC (*Linguistics Inquiry Word Count*). В работе [18] из графа совместно встречающихся слов извлекаются признаки, которые основаны на контексте. Также существуют исследования, в которых для распознавания точки зрения автора текста используются модели, основанные на сверточных [21—23, 27, 28] и рекуррентных [16, 29, 30] нейронных сетях.

Целью слабого машинного обучения с учителем является предсказание выраженной в тексте позиции автора с использованием как размеченных, так и неразмеченных данных. В подходе, предложенном в работе [31], сначала в неразмеченном корпусе осуществляется поиск признаков, характеризующих позицию автора (например, *n*-грамм, определяющих позицию или тональность), с помощью исследования высокочастотных *n*-грамм и использования правил фильтрации для удаления нерелевантных текстов. Затем с помощью трех веб-сервисов для анализа тональности (HP Haven On Demand, IBM Alchemy и Vivekn) определяется выраженная в тексте позиция автора за счет суммирования трех предсказанных оценок тональности. Тексту присваивается метка *за*, если суммарная оценка положительная, и наоборот. Метка *нейтрально* присваивается тексту в случае, когда оценки, полученные с использованием веб-сервисов Haven и Alchemy, равны нулю. Сформированные таким образом зашумленные корпуса обучающих данных используют для обучения SVM-классификатора. Авторы работы [23] применяют двухэтапный метод предсказания позиции автора текста. Сначала осуществляется разметка неразмеченных данных на основе предположения, что некоторые выражения и хештеги могут указывать на позицию автора по отношению к определенному целевому объекту. Размеченные с использованием таких выражений и хештегов данные затем используют для обучения сверточной нейронной сети.

В работе [1] отмечено, что методы, разработанные в рамках подходов на основе слабого машинного обучения с учителем, работают хуже методов, основанных на стандартном машинном обучении с учителем. Одной из причин является малое число обучающих данных (несколько сотен текстов) в каждой предметной области.

Первой работой, посвященной распознаванию точки зрения автора в русскоязычных текстах, является [32]. В настоящей работе предлагаются структура системы распознавания точки зрения автора текста на основе композиционного подхода и вариант ее программной реализации. Работа объединяет отдельные исследования [33—35] в один общий метод, положенный в основу программной системы, и имеет следующие отличия от работы [32]:

— в работе [32] исследовали различные способы представления признаков, в настоящей статье исследуются ансамбли методов машинного обучения;

— в настоящей статье предложен подход к решению задачи классификации текстов на основе ансамблей методов отбора признаков и ансамблей классификаторов вместо подхода, использующего только отбор признаков в работе [32];

— в настоящей статье при проведении экспериментов используются три текстовых корпуса вместо одного.

Разработанная система использует ансамбли разных методов машинного обучения с учителем для определения позиции автора текста. Методы, предложенные в работах [33—35] и реализованные в системе, позволяют:

- определять оптимальное число и формировать оптимальное множество признаков для эффективного решения задачи с использованием методов машинного обучения;
- автоматически определять точку зрения автора текста; применяя наиболее релевантные признаки и преимущества отдельных методов машинного обучения, система позволяет на высоком уровне качества распознавать позицию автора текста, что подтверждается экспериментами, проведенными с использованием русскоязычных текстовых корпусов.

Общая структура системы

Разработанная программная система распознавания точки зрения автора текста состоит из нескольких компонентов и имеет структуру, представленную на рис 1. В соответствии с принципами машинного обучения с учителем массив текстовых документов, подаваемый на вход системе, делится на обучающий и контрольный корпуса. Обучающий корпус содержит размеченные тексты, принадлежащие классам позиций *за* и *против*. Контрольный корпус также содержит размеченные тексты, однако информация о разметке системе не предоставляется, а используется при итоговой оценке качества распознавания точки зрения автора текста. В процессе работы системы из обучающего корпуса документов выделяется проверочный корпус, используемый при поиске оптимального подмножества признаков и для подбора параметров классификатора.

На начальном этапе все тексты проходят первичную обработку в **подсистеме предварительной**

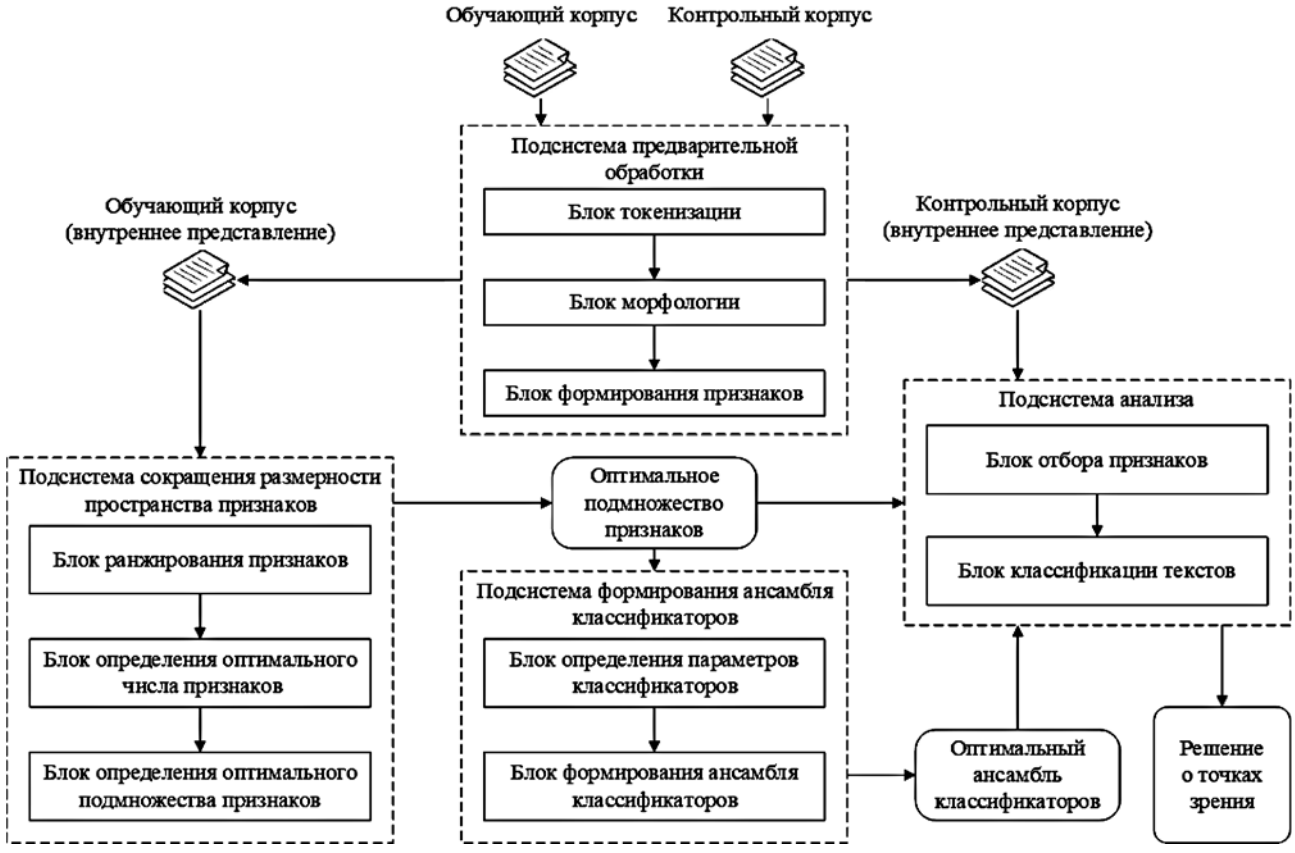

Рис. 1. Структура системы распознавания точки зрения автора текста

обработки. Сначала текст разбивается на токены, представляющие собой словоформы. Затем полученные токены передаются морфологическому анализатору для определения частей речи и выполнения нормализации словоформ. В этой же подсистеме реализованы процедура удаления слов, не несущих смысловой нагрузки (стоп-слов), и процедура формирования словаря. В блоке формирования признаков конструируется множество признаков, представляющих собой отдельные термины (леммы), и строится модель представления текста. Таким образом, в результате первичного анализа каждый документ представляется в виде двоичного вектора, компоненты которого характеризуют наличие или отсутствие термина в документе.

Подсистема сокращения размерности пространства признаков выполняет отбор наиболее релевантных признаков. В блоке ранжирования признаков с использованием заданного метода отбора признаков осуществляются их взвешивание, ранжирование и упорядочивание по убыванию ранга. В следующем блоке с помощью алгоритма, предложенного в работе [33], определяется оптимальное число признаков. Сначала строится зависимость F1-меры от числа наиболее релевантных признаков с заданным шагом. Для построения зависимости используется процедура N-кратной перекрестной проверки. Обучающий корпус текстовых документов, принадлежащих одной предметной области, делится на N равных по размеру блоков. Каждый блок поочередно объявляется контрольным и используется для оценки качества классификации, оставшиеся $N-1$ блоков используются для ранжирования признаков с последующим выбором первых k наиболее релевантных признаков и обучением классификатора. Значение F1-меры вычисляется для каждого контрольного блока и усредняется по N блокам при каждом фиксированном k. Полученная таким образом зависимость аппроксимируется функцией распределения Вейбулла, характер которой определяется подбором коэффициентов. В блоке определения оптимального подмножества признаков с помощью алгоритма, предложенного в работе [34], осуществляется поиск подмножества признаков, использование которых позволяет получить наибольшее значение F1-меры на проверочном множестве данных. Найденное подмножество признаков принимается за оптимальное и подается на вход следующей подсистеме.

Подсистема формирования ансамбля классификаторов реализует алгоритм формирования оптимального ансамбля классификаторов, предложенного в работе [35]. В первом блоке подсистемы осуществляется поиск параметров классификаторов на основании процедуры N-кратной перекрестной проверки. Во втором блоке подсистемы на основании аналогичной процедуры определяется наилучший ансамбль из предложенного набора классификаторов.

Оптимальное подмножество признаков и оптимальный ансамбль классификаторов используются **подсистемой анализа**. В блоке сокращения числа признаков формируется модель представления текстов, учитывающая только те признаки, которые содержатся в оптимальном подмножестве. В блоке агрегирования предсказаний классификаторов реализован метод мажоритарного голосования, на основании которого определяется результирующее предсказание ансамбля по формуле

$$class(d) = \underset{s \in S}{\operatorname{argmax}} \sum g\left(y_k(d),\ s\right),$$

где $y_k(d)$ — предсказание k-го классификатора для документа d; S — множество классов; $g(y, s)$ — функция, определяемая следующим образом:

$$g(y,\ s) = \begin{cases} 1,\ y = s, \\ 0,\ y \neq s. \end{cases}$$

В случае четного числа классификаторов в ансамбле и равного числа голосов в каждом классе документу присваивается метка *за*.

Итоговым результатом работы подсистемы анализа являются текстовые документы контрольного корпуса, классифицированные на m классов точек зрения авторов.

Программная реализация системы

Разработка программной системы осуществлялась с использованием принципов объектно-ориентированного программирования и унифицированного языка моделирования (*Unified Modeling Language*, UML). Общая структура иерархии классов представлена на рис. 2.

Некоторые вспомогательные классы, например, класс *Preprocessing*, реализующий подсистему предварительной обработки, на диаграмме не представлены. Для каждого класса перечислены атрибуты и методы (для простоты сигнатура методов опущена).

На диаграмме между классами представлены три вида отношений:

— отношение ассоциации (→) — отражает направленную связь. Например, класс *FeatureNumOptimization* в качестве параметра метода использует класс *Corpus*;

— отношение агрегации (—◇) — представляет отношение "целое-часть". Например, класс *FeatureSetOptimization* включает в себя класс *FeatureRanking* и может использовать реализуемые в нем функциональные возможности, т. е. выполнять ранжирование признаков с помощью определенного метода отбора признаков;

— отношение зависимости (- - →). Например, класс *EnsClfOptimization* зависит от класса *OptimalSet*, изменение спецификации которого может повлиять на работу зависимого класса, но не наоборот.

Текстовые корпуса формируются методами служебных классов (на диаграмме не представлены) и хранятся в структуре *Corpus*. Атрибут *Type* определяет тип текстового корпуса: обучающий или контрольный. Для создания корпуса используется метод *Create*. Корпус можно сохранить в файл, используя метод *Save*, или загрузить из файла с помощью метода *Load*.

Для хранения отдельного документа используется структура *Document*, содержащая следующие атрибу-

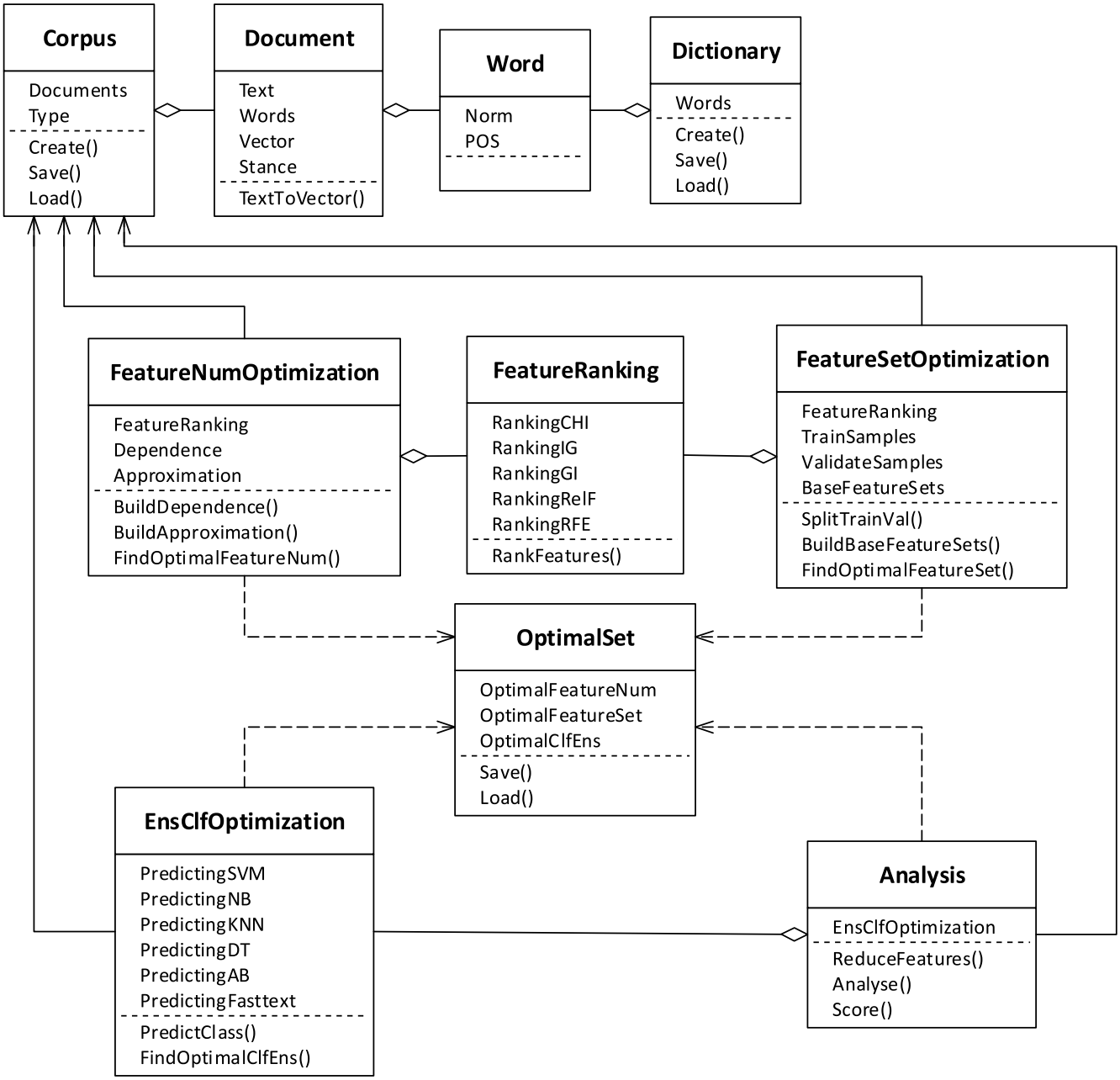

Рис. 2. Диаграмма основных классов системы

ты: *Text* — содержимое документа, которое разбивается на отдельные слова *Words*; *Vector* — представление документа в виде бинарного вектора, компоненты которого характеризуют наличие или отсутствие слова в документе; *Stance* — метка, определяющая позицию автора текста.

Слова хранятся в структуре *Word*. Для каждого слова определяются его нормальная форма *Norm* и часть речи *POS* (*part of speech*).

Структура *Dictionary* используется для хранения словаря. Формирование словаря осуществляется с помощью метода *Create*. Для сохранения словаря в файл используется метод *Save*, а для загрузки из файла — метод *Load*.

Структура *OptimalSet* используется для хранения оптимального числа признаков (*OptimalFeatureNum*), оптимального множества признаков (*OptimalFeatureSet*) и оптимального ансамбля классификаторов (*OptimalClfEns*). Оптимальные параметры могут быть сохранены в файл (метод *Save*) или загружены из файла (метод *Load*).

Реализация подсистемы сокращения пространства признаков выполнена с использованием классов *FeatureNumOptimization* и *FeatureSetOptimization*, включающих в себя класс *FeatureRanking*. В последнем из перечисленных классов осуществляются ранжирование признаков (метод *RankFeatures*) с помощью методов χ^2 (CHI), прирост информации (*Information Gain*, IG),

индекс Джини (*Gini Index*, GI), *ReliefF* (RelF), рекурсивный отбор признаков (*Recursive Feature Elimination*, RFE). В классе *FeatureNumOptimization* реализован алгоритм определения оптимального числа признаков из работы [33], состоящий из следующих трех этапов:

1) построение реальной зависимости качества классификации от числа признаков (метод *BuildDependence*);

2) построение функции Вейбулла, аппроксимирующей реальную зависимость (метод *BuildApproximation*);

3) нахождение минимального аргумента на графике функции Вейбулла, начиная с которого скорость роста функции становится мало отличимой от нуля (метод *FindOptimalFeatureNum*), этот аргумент записывается в поле *OptimalFeatureNum* класса *OptimalSet*.

В классе *FeatureSetOptimization* реализован алгоритм определения оптимального множества признаков из работы [34], состоящий из следующих этапов:

1) деление множества данных на обучающее и проверочное подмножества (метод *SplitTrainVal*);

2) формирование базовых подмножеств признаков путем деления обучающего подмножества данных на *N* блоков (например, пять) с последующим ранжированием признаков на этих блоках и выбором в каждом блоке первых *OptimalFeatureNum* признаков с наибольшим весом (метод *BuildBaseFeatureSets*);

3) нахождение всевозможных пересечений и объединений подмножеств признаков в сочетаниях по два, три, четыре и пять с последующей их оценкой на проверочном подмножестве данных (метод *FindOptimalFeatureSet*).

Подмножество признаков, для которого была получена наилучшая оценка на проверочном подмножестве данных, записывается в поле *OptimalFeatureSet* класса *OptimalSet*.

Подсистема формирования ансамбля классификаторов реализуется в классе *EnsClfOptimization*. Метод *FindOptimalClfEns* реализует алгоритм формирования оптимального ансамбля классификаторов из работы [35]. Найденный ансамбль записывается в поле *OptimalClfEns* структуры *OptimalSet*.

Подсистема анализа реализуется в классе *Analysis*. В этом классе осуществляется классификация контрольного корпуса документов по классам точек зрения авторов (метод *Analyse*) с использованием оптимального множества признаков и оптимального ансамбля классификаторов. Здесь же оценивается качество классификации в методе *Score*.

Программная реализация рассмотренных классов выполнена на языке программирования Python 3.

Экспериментальные результаты

Оценка эффективности разработанной программной системы выполнялась с использованием трех текстовых корпусов, составленных из русскоязычных сообщений пользователей интернет-форумов[1] и социальной сети "ВКонтакте"[2]. При составлении корпусов были выбраны три социально острых вопроса, в отношении которых высказываются противоположные точки зрения, а именно — вакцинация детей, ЕГЭ в школе и клонирование человека. 5000 сообщений были размечены на три класса точек зрения авторов: *за* (1550 текстов), *против* (1950 текстов), *невозможность определения точки зрения* (1500 текстов). Разметка осуществлялась с привлечением трех аннотаторов. Для оценки степени согласованности между аннотаторами использовалась статистическая мера каппа Флейса [36], значения которой оказались равны для корпуса *Вакцинация детей* — 0,87, *ЕГЭ в школе* — 0,89, *Клонирование человека* — 0,86. Согласно [37] значение каппы Флейса большее 0,8 указывает на почти полное согласие аннотаторов.

В экспериментах использовали только тексты, принадлежащие классам *за* и *против* (всего 3500 текстов[3]), т. е. решалась задача двухклассовой классификации. Характеристики текстовых корпусов приведены в табл. 1. В столбце "Общее число слов" указано число словоформ с повторами для каждого класса, в столбце "Число слов в нормальной форме" — размер словаря, содержащего слова, приведенные к нормальной форме. Для выполнения морфологического анализа текстов применяли программу *MyStem* от компании Яндекс [38].

В подсистеме формирования ансамбля классификаторов применяли шесть базовых методов классификации, обеспечивающих разнообразие в ансамбле за счет разных подходов, лежащих в их основе, а именно линейный классификатор — метод опорных векторов (*Support Vector Machine*, SVM); вероятностный классификатор — наивный байесовский классификатор (*Naïve Bayes*, NB); метрический классификатор — метод k-ближайших соседей (*k-Nearest Neighbors*, kNN); логический классификатор — дерево решений (*Decision Tree*, DT); ансамблевый классификатор — адаптивный бустинг (*Adaptive Boosting*, AB); нейросетевой классификатор — быстрый текстовый классификатор (*Fasttext*, FT). В экспериментах использовалась реализация алгоритмов машинного обучения из библиотек *scikit-learn* [39] и *fasttext* [40]. Для сравнения использовали также два простых базовых классификатора:

- первый алгоритм (*baseline* 1, BL1) относит текст к классу *против*, если в этом тексте присутствует слово "против", и к классу *за* в остальных случаях;
- второй алгоритм (*baseline* 2, BL2) относит текст к классу *за*, если в этом тексте присутствует слово "за", и к классу *против* в остальных случаях.

В качестве модели текстового представления использовалась векторная модель [41]. Тексты представлялись в виде *n*-мерного двоичного вектора, значения компонент которого характеризовали наличие или отсутствие соответствующего слова в тексте. Признаком является слово в нормальной форме.

Для оценки качества классификации использовалась F1-мера. Итоговые значения метрик получены способом макроусреднения по всем классам.

Для получения объективных оценок качества в экспериментах использовали процедуру 5-крат-

[1] http://www.woman.ru/forum, http://www.yaplakal.com, http://www.rusforum.com, http://www.kid.ru и др.

[2] https://vk.com

[3] https://tinyurl.com/y3n67c24

Таблица 1

Характеристики текстовых корпусов

Корпус	Метка текста	Число текстов	Общее число слов	Средняя длина текста, слов	Число слов в нормальной форме
Вакцинация детей	За	500	35 326	70	5100
	Против	500	34 167	68	
ЕГЭ в школе	За	600	35 410	59	5943
	Против	800	40 453	51	
Клонирование человека	За	450	18 860	42	4990
	Против	650	27 405	42	

Таблица 2

Значения *F*1-меры

Метод классификации	Корпус			Среднее значение
	Вакцинация детей	ЕГЭ в школе	Клонирование человека	
BL1	0,551	0,579	0,559	0,563
BL2	0,578	0,590	0,557	0,575
SVM	0,767	0,767	0,742	0,759
NB	0,775	0,719	0,660	0,718
kNN	0,648	0,565	0,580	0,598
AB	0,748	0,714	0,693	0,718
DT	0,638	0,702	0,644	0,661
FT	0,748	0,726	0,737	0,737
Предложенная система	**0,798**	**0,779**	**0,753**	**0,777**

ной перекрестной проверки. В результате для каждого классификатора было получено пять значений F1-меры на контрольных подмножествах данных, которые затем усредняли для получения итоговых значений оценок качества, представленных в табл. 2.

Итоговые оценки получены с использованием оптимальных множеств признаков, размеры которых составили для корпуса *Вакцинация детей* — 2652, для корпуса *ЕГЭ в школе* — 2899, для корпуса *Клонирование человека* — 2961.

Предложенная в настоящей работе система позволила повысить оценку F1-меры по сравнению с базовыми классификаторами. Превосходство над ближайшим лучшим методом (NB) для корпуса *Вакцинация детей* составило 2,3 %, для корпуса *ЕГЭ в школе* — 1,2 % (над методом SVM), для корпуса *Клонирование человека* — 1,1 % (над методом SVM). При этом в модели представления текстов для корпуса *Вакцинация детей* использовалось 52 % наиболее значимых признаков, для корпуса *ЕГЭ в школе* — 49 % признаков и для корпуса *Клонирование человека* — 59 % признаков от их первоначального количества.

Проверку статистической значимости результатов осуществляли с использованием критерия знаковых рангов Уилкоксона [42]. Расчет критерия проводили для 15 значений *F*1-меры (по пять значений для каждого корпуса), полученных с применением процедуры 5-кратной перекрестной проверки. В соответствии с критерием Уилкоксона выдвигаются две гипотезы: H_0 — отклонения оценок качества классификации носят случайный характер, и предложенный метод не влияет на качество классификации; H_1 — отклонения оценок качества не случайны. При уровне значимости α, равном 5 %, между эмпирическим ($T_{эмп}$) и критическим ($T_{кр}$) значениями *T*-критерия Уилкоксона имеет место соотношение

Таблица 3

Наилучшие сочетания классификаторов в ансамбле

№ разбиения	Корпус		
	Вакцинация детей	ЕГЭ в школе	Клонирование человека
1	SVM, NB, AB, FT	SVM, NB, DT, FT	NB, FT
2	SVM, NB, AB, DT, FT	SVM, NB, AB, FT	SVM, NB, AB, FT
3	SVM, NB, kNN, FT	SVM, NB, FT	SVM, NB, DT, FT
4	SVM, FT	SVM, NB, DT, FT	SVM, NB, DT, FT
5	SVM, NB, kNN, FT	SVM, NB, kNN, FT	SVM, NB, DT, FT

$T_{эмп} = 21 < T_{кр} = 30$. Следовательно, гипотеза H_0 отвергается, т. е. разность между показателями качества для лучшего базового классификатора и предлагаемого метода значима на уровне значимости $\alpha = 0{,}05$.

В табл. 3 представлены наилучшие сочетания классификаторов в ансамбле для каждого разбиения на контрольные и обучающие данные в процедуре 5-кратной перекрестной проверки. На основании полученных результатов можно сделать вывод, что в 100 % ансамблей присутствует метод FT, в 87 % — методы SVM и FT, в 80 % — методы SVM, NB, FT. Анализируя данные из табл. 2, можно заметить, что именно эти методы по отдельности дают лучшие оценки F1-меры в решении рассматриваемой задачи. Однако следует отметить, что в отдельных случаях в наилучшие сочетания классификаторов входят методы AB, kNN и DT, поэтому отказ от их использования привел бы к некоторому ухудшению средних оценок качества.

Заключение

Программная система, представленная в настоящей работе, демонстрирует высокое качество распознавания точки зрения автора текста, превосходящее другие методы машинного обучения. По результатам экспериментов в среднем для трех текстовых корпусов оценка F1-меры повысилась на 1,8 % по сравнению с лучшим базовым классификатором SVM.

Среди других преимуществ разработанной системы следует отметить:

— адаптированный выбор числа наиболее релевантных признаков в зависимости от характера обучающих данных;

— формирование подмножества релевантных для предметной области признаков, размер которого значительно меньшего размера первоначального множества признаков.

Предлагаемая система может использоваться в качестве самостоятельного инструмента для распознавания точек зрения авторов текстовых документов, а также может быть интегрирована в сторонние сервисы, осуществляющие поиск и рекомендацию документов на основе анализа текстовой информации.

Работа выполнена при финансовой поддержке Министерства образования и науки РФ, государственное задание ВятГУ № 34.2092.2017/4.6, проект "Разработка и исследование словарей оценочной лексики для анализа тональности текстов" (2017—2019 гг.).

Список литературы

1. **Wang R., Zhou D., Jiang M., Jiasheng S.** A Survey on Opinion Mining: from Stance to Product Aspect // IEEE Access. 2019. Vol. 7. P. 41101—41124.

2. **Ferreira W., Vlachos A.** Emergent: a novel data-set for stance classification // NAACL-HLT 2016, 2016. P. 1163—1168.

3. **Mohammad S. M., Kiritchenko S., Sobhani P.** et al. SemEval-2016 Task 6: Detecting Stance in Tweets // Proceedings of SemEval—2016. 2016. P. 31—41.

4. **Васенин В. А., Дзабраев М. Д.** Методы и средства мониторинга публикаций в средствах массовой информации // Программная инженерия. 2017. Т. 8, № 11. С. 490—503.

5. **Васенин В. А., Роганов В. А., Дзабраев М. Д.** Методы автоматизированного анализа тональности текстов в средствах массовой информации // Программная инженерия. 2016. Т. 7, № 8. С. 360—372.

6. **Somasundaran S., Wiebe J.** Recognizing Stances in Online Debates // 47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP, 2009. P. 226—234.

7. **Sridhar D., Foulds J., Huang B., Getoor L.** et al. Joint Models of Disagreement and Stance in Online Debate // Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, 2015. P. 116—125.

8. **Mohammad S. M.** Sentiment analysis: detecting valence, emotions, and other affectual states from text // Emotion Measurement, 2015. P. 231—237.

9. **Malouf R., Mullen T.** Taking sides: User classification for informal online political discourse // Internet Research. 2008. Vol. 18. P. 177—190.

10. **Agrawal R., Rajagopalan S., Srikant R., Xu Y.** Mining Newsgroups Using Networks Arising from Social Behavior // 12th International Conference on World Wide Web (WWW 2003), 2003. P. 529—535.

11. **Anand P., Walker M., Abbott R.** et al. Cats Rule and Dogs Drool!: Classifying Stance in Online Debate // 2nd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis, 2011. P. 1—9.

12. **Hasan K. S., Ng V.** Stance Classification of Ideological Debates: Data, Models, Features, and Constraints // International Joint Conference on Natural Language Processing, 2013. P. 1348—1356.

13. **Thomas M., Pang B., Lee L.** Get out the vote: Determining support or opposition from Congressional floor-debate transcripts // EMNLP, 2006. P. 327—335.

14. **Walker M. A., Anand P., Abbott R., Grant R.** Stance Classification using Dialogic Properties of Persuasion // Conference of the North American Chapter of the ACL: Human Language Technologies, 2012. P. 592—596.

15. **Sobhani P., Inkpen D., Matwin S.** From Argumentation Mining to Stance Classification // 2nd Workshop on Argumentation Mining, 2015. P. 67—77.

16. **Benton A., Dredze M.** Using author embeddings to improve Tweet stance classification // Proceedings of the EMNLP Workshop W-NUT, 4th Workshop Noisy User-Generated Text, 2018. P. 184—194.

17. **Elfardy H., Diab M. T.** CU-GWU perspective at SemEval-2016 task 6: Ideological stance detection in informal text // Proceedings of the 10th International Workshop Semantic Evaluation (SemEval), 2016. P. 434—439.

18. **Gadek G., Betsholtz J., Pauchet A.** et al. Extracting contextonyms from Twitter for stance detection // Proceedings of the 9th International Conference on Agents and Artificial Intelligence, 2017. P. 132—141.

19. **Misra A., Ecker B., Handleman T. A.** et al. NLDS-UCSC at SemEval-2016 task 6: A semi-supervised approach to detecting stance in Tweets // Proceedings of the 10th International Workshop Semantic Evaluation (SemEval), 2016. P. 420—427.

20. **Patra B. G., Das D., Bandyopadhyay S.** JU_NLP at semeval-2016 task 6: Detecting stance in Tweets using support

vector machines // Proceedings SemEval@ NAACL-HLT, 2016. P. 440—444.

21. **Vijayaraghavan P., Sysoev I., Vosoughi S., Roy D.** DeepStance at SemEval-2016 task 6: Detecting stance in Tweets using character and word-level CNNs. arXiv preprint, arXiv:1606.05694, 2016.

22. **Wei P., Mao W., Zeng D.** A target-guided neural memory model for stance detection in Twitter // Proceedings of the International Joint Conference on Neural Network (IJCNN), 2018. P. 1—8.

23. **Wei W., Zhang X., Liu X.** et al. Pkudblab at SemEval-2016 task 6: A specific convolutional neural network system for effective stance detection // Proceedings of the 10th International Workshop Semantic Evaluation (SemEval), 2016. P. 384—388.

24. **Zarrella G., Marsh A.** MITRE at SemEval-2016 Task 6: Transfer learning for stance detection. arXiv preprint, arXiv: 1606.03784, 2016.

25. **Augenstein I., Vlachos A., Bontcheva K.** USFD at SemEval-2016 task 6: Any-target stance detection on Twitter with autoencoders // Proceedings of the 10th International Workshop Semantic Evaluation (SemEval), 2016. P. 389—393.

26. **Xu C., Paris C., Nepal S., Sparks R.** Cross-target stance classification with self-attention networks. arXiv preprint, arXiv: 1805.06593, 2018.

27. **Igarashi Y., Komatsu H., Kobayashi S.** et al. Tohoku at SemEval-2016 task 6: Feature-based model versus convolutional neural network for stance detection // Proceedings of the 10th International Workshop Semantic Evaluation (SemEval), 2016. P. 401—407.

28. **Zhou Y., Cristea A. I., Shi L.** Connecting targets to Tweets: Semantic attention-based model for target-specific stance detection // Proceedings of the International Conference on Web Information System Engineering, 2017. P. 18—32.

29. **Du J., Xu R., He Y., Gui L.** Stance classification with target-specific neural attention networks // Proceedings of the International Joint Conference on Artificial Intelligence, 2017. P. 3988—3994.

30. **Sun Q., Wang Z., Zhu Q., Zhou G.** Stance detection with hierarchical attention network // Proceedings of the 27th International Conference on Computational Linguistics, 2018. P. 2399—2409.

31. **Dias M., Becker K.** INF-UFRGS-OPINION-MINING at SemEval-2016 task 6: Automatic generation of a training corpus for unsupervised identification of stance // Proceedings of the 10th International Workshop Semantic Evaluation, 2016. P. 378—383.

32. **Vychegzhanin S., Kotelnikov E.** Stance Detection in Russian: a Feature Selection and Machine Learning Based Approach // Supplementary Proceedings of the Sixth International Conference on Analysis of Images, Social Networks and Texts (AIST 2017). 2017. Vol. 1975. P. 166—179.

33. **Vychegzhanin S., Razova E., Kotelnikov E.** What Number of Features is Optimal? A New Method Based on Approximation Function for Stance Detection Task // 9th International Conference on Information Communication and Management (ICICM 2019), 2019. P. 43—47.

34. **Vychegzhanin S., Razova E., Kotelnikov E.** Selecting an optimal feature set for stance detection // 8th International Conference on Analysis of Images, Social networks and Texts (AIST 2019), 2019 [в печати].

35. **Вычегжанин С. В., Котельников Е. В.** Определение точки зрения автора текста на основе ансамблей классификаторов // Программирование. 2019. № 5. С. 10—24.

36. **Fleiss J. L.** Measuring nominal scale agreement among many raters // Psychological Bulletin. 1971. Vol. 76, N. 5. P. 378—382.

37. **Artstein R., Poesio M.** Inter-coder agreement for computational linguistics // Journal of Computational Linguistics. 2008. Vol. 34, N. 4. P. 555—596.

38. **Segalovich I.** A Fast Morphological Algorithm with Unknown Word Guessing Induced by a Dictionary for a Web Search Engine // MLMTA-2003, 2003. P. 273—280.

39. **Pedregosa F., Varoquaux G., Gramfort A.** et al. Scikit-learn: Machine Learning in Python // JMLR. 2011. Vol. 12. P. 2825—2830.

40. **Joulin A., Grave E., Bojanowski P., Mikolov T.** Bag of tricks for efcient text classication. Technical report, arXiv:1607.01759, 2016.

41. **Manning C. D., Raghavan P., Schudze H.** Introduction to Information Retrieval. Cambridge University Press, 2008. 506 p.

42. **Wilcoxon F.** Individual comparisons by ranking methods // Biometrics Bulletin. 1945. Vol. 1, N. 6. P. 80—83.

Software System for Stance Detection Based on Compositional Approach

S. V. Vychegzhanin, vychegzhaninsv@gmail.com, Vyatka State University, Kirov, 610000, Russian Federation

Corresponding author:

Vychegzhanin Sergey V., Software Engineer, Vyatka State University, Kirov, 610000, Russian Federation
E-mail: vychegzhaninsv@gmail.com

Received on September 30, 2019
Accepted on October 21, 2019

The article is devoted to the task of automatic stance detection. Stance detection is the task of automatically determining from text whether the author of the text is in favor of, against, or neutral towards a proposition or target. There is a wide range of areas where stance detection is used, including political, sociological and marketing research, the search engines, the human-computer interfaces.

The article proposes the structure of the software system for stance detection based on ensembles of methods for feature selection and ensembles of classifiers. The structure includes four main subsystems, namely text preprocessing,

feature space dimension reduction, creation of ensembles of classifiers and analysis subsystems. The system is implemented in accordance with the object-oriented approach. Main classes and their relations are described with UML class diagram.

The results of experimental research of developed system are presented. The research is performed using the text corpora composed of messages from users of the social network "VK" and online forums. These results show high quality stance detection that is superior to other machine learning methods.

The developed system can be used as a standalone application for stance detection. It is possible to integrate the system into third party services, that search and recommend documents based on information analysis.

Keywords: *natural language processing, opinion mining, stance detection, software system, class diagram*

Acknowledgments: *This work was supported by the Ministry of Education and Science of the Russian Federation and by the Vyatka State University (the project "Development and research of sentiment lexicons for text sentiment analysis" No. 34.2092.2017 / 4.6).*

For citation:

Vychegzhanin S. V. Software System for Stance Detection Based on Compositional Approach, *Programmnaya Ingeneria*, 2020, vol. 11, no. 1, pp. 54—64.

DOI: 10.17587/prin.11.54-64

References

1. **Wang R., Zhou D., Jiang M., Jiasheng S.** A Survey on Opinion Mining: from Stance to Product Aspect, *IEEE Access*, 2019, vol. 7, pp. 41101—41124.

2. **Ferreira W., Vlachos A.** Emergent: a novel data-set for stance classification, *NAACL-HLT 2016*, 2016, pp. 1163—1168.

3. **Mohammad S. M., Kiritchenko S., Sobhani P., Zhu X., Cherry C.** SemEval-2016 Task 6: Detecting Stance in Tweets, *Proceedings of SemEval-2016*, 2016, pp. 31—41.

4. **Vasenin V. A., Dzabraev M. D.** Methods and Means for Monitoring Publications in Mass Media, *Programmnaya Ingeneria*, 2017, vol. 8, no. 11, pp. 490—503 (in Russian).

5. **Vasenin V. A., Roganov V. A., Dzabraev M. D.** Methods of Automated Sentiment Analysis of Texts Published by Mass Media, *Programmnaya Ingeneria*, 2016, vol. 7, no. 8, pp. 360—372 (in Russian).

6. **Somasundaran S., Wiebe J.** Recognizing Stances in Online Debates, *47th Annual Meeting of the ACL and the 4th IJCNLP of the AFNLP*, 2009, pp. 226—234.

7. **Sridhar D., Foulds J., Huang B., Getoor L., Walker M.** Joint Models of Disagreement and Stance in Online Debate, *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, 2015, pp. 116—125.

8. **Mohammad S. M.** Sentiment analysis: detecting valence, emotions, and other affectual states from text, *Emotion Measurement*, 2015, pp. 231—237.

9. **Malouf R., Mullen T.** Taking sides: User classification for informal online political discourse, *Internet Research*, 2008, vol. 18, pp. 177—190.

10. **Agrawal R., Rajagopalan S., Srikant R., Xu Y.** Mining Newsgroups Using Networks Arising from Social Behavior, *12th International Conference on World Wide Web (WWW 2003)*, 2003, pp. 529—535.

11. **Anand P., Walker M., Abbott R., Fox Tree J. E., Bowmani R., Minor M.** Cats Rule and Dogs Drool!: Classifying Stance in Online Debate, *2nd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis*, 2011, pp. 1—9.

12. **Hasan K. S., Ng V.** Stance Classification of Ideological Debates: Data, Models, Features, and Constraints, *International Joint Conference on Natural Language Processing*, 2013, pp. 1348—1356.

13. **Thomas M., Pang B., Lee L.** Get out the vote: Determining support or opposition from Congressional floor-debate transcripts, *EMNLP*, 2006, pp. 327—335.

14. **Walker M. A., Anand P., Abbott R., Grant R.** Stance Classification using Dialogic Properties of Persuasion, *Conference of the North American Chapter of the ACL: Human Language Technologies*, 2012, pp. 592—596.

15. **Sobhani P., Inkpen D., Matwin S.** From Argumentation Mining to Stance Classification, *2nd Workshop on Argumentation Mining*, 2015, pp. 67—77.

16. **Benton A., Dredze M.** Using author embeddings to improve Tweet stance classification, *Proceedings EMNLP Workshop W-NUT, 4th Workshop Noisy User-Generated Text*, 2018, pp. 184—194.

17. **Elfardy H., Diab M. T.** CU-GWU perspective at SemEval-2016 task 6: Ideological stance detection in informal text, *Proceedings 10th International Workshop Semantic Evaluation (SemEval)*, 2016, pp. 434—439.

18. **Gadek G., Betsholtz J., Pauchet A., Brunessaux S., Malandain N., Vercouter L.** Extracting contextonyms from Twitter for stance detection, *9th International Conference on Agents and Artificial Intelligence*, 2017, pp. 132—141.

19. **Misra A., Ecker B., Handleman T., Hahn N., Walker M. A.** NLDS-UCSC at SemEval-2016 task 6: A semi-supervised approach to detecting stance in Tweets, *Proceedings of the 10th International Workshop Semantic Evaluation (SemEval)*, 2016, pp. 420—427.

20. **Patra B. G., Das D., Bandyopadhyay S.** JU_NLP at semeval-2016 task 6: Detecting stance in Tweets using support vector machines, *Proceedings SemEval@ NAACL-HLT*, 2016, pp. 440—444.

21. **Vijayaraghavan P., Sysoev I., Vosoughi S., Roy D.** DeepStance at SemEval-2016 task 6: Detecting stance in Tweets using character and word-level CNNs, arXiv preprint, arXiv:1606.05694, 2016.

22. **Wei P., Mao W., Zeng D.** A target-guided neural memory model for stance detection in Twitter, *Proceedings of the International Joint Conference on Neural Network (IJCNN)*, 2018, pp. 1—8.

23. **Wei W., Zhang X., Liu X., Chen W., Wang T.** Pkudblab at SemEval-2016 task 6: A specific convolutional neural network system for effective stance detection, *Proceedings of the 10th International Workshop Semantic Evaluation (SemEval)*, 2016, pp. 384—388.

24. **Zarrella G., Marsh A.** MITRE at SemEval-2016 Task 6: Transfer learning for stance detection, arXiv preprint, arXiv: 1606.03784, 2016.

25. **Augenstein I., Vlachos A., Bontcheva K.** USFD at SemEval-2016 task 6: Any-target stance detection on Twitter with autoencoders, *Proceedings of the 10th International Workshop Semantic Evaluation (SemEval)*, 2016, pp. 389—393.

26. **Xu C., Paris C., Nepal S., Sparks R.** Cross-target stance classification with self-attention networks, arXiv preprint, arXiv: 1805.06593, 2018.

27. **Igarashi Y., Komatsu H., Kobayashi S., Okazaki N., Inui K.** Tohoku at SemEval-2016 task 6: Feature-based model versus convolutional neural network for stance detection, *Proceedings of the 10th International Workshop Semantic Evaluation (SemEval)*, 2016, pp. 401—407.

28. **Zhou Y., Cristea A. I., Shi L.** Connecting targets to Tweets: Semantic attention-based model for target-specific stance detection, *Proceedings of the International Conference on Web Information System Engineering*, 2017, pp. 18—32.

29. **Du J., Xu R., He Y., Gui L.** Stance classification with target-specific neural attention networks, *Proceedings of the International Joint Conference on Artificial Intelligence*, 2017, pp. 3988—3994.

30. **Sun Q., Wang Z., Zhu Q., Zhou G.** Stance detection with hierarchical attention network, *Proceedings of the 27th International Conference on Computational Linguistics*, 2018, pp. 2399—2409.

31. **Dias M., Becker K.** INF-UFRGS-OPINION-MINING at SemEval-2016 task 6: Automatic generation of a training corpus for unsupervised identification of stance, *Proceedings of the 10th International Workshop Semantic Evaluation*, 2016, pp. 378—383.

32. **Vychegzhanin S., Kotelnikov E.** Stance Detection in Russian: a Feature Selection and Machine Learning Based Approach, *Supplementary Proceedings of the Sixth International Conference on Analysis of Images, Social Networks and Texts (AIST 2017)*, 2017, vol. 1975, pp. 166—179.

33. **Vychegzhanin S., Razova E., Kotelnikov E.** What Number of Features is Optimal? A New Method Based on Approximation Function for Stance Detection Task, *9th International Conference on Information Communication and Management (ICICM 2019)*, 2019, pp. 43—47.

34. **Vychegzhanin S., Razova E., Kotelnikov E.** Selecting an optimal feature set for stance detection, *8th International Conference on Analysis of Images, Social networks and Texts (AIST 2019)*, 2019 [in print].

35. **Vychegzhanin S. V., Kotelnikov E. V.** Stance detection based on ensembles of classifiers, *Programmirovanie*, 2019, no. 5, pp. 10—24 (in Russian).

36. **Fleiss J. L.** Measuring nominal scale agreement among many raters, *Psychological Bulletin*, 1971, vol. 76, no. 5, pp. 378—382.

37. **Artstein R., Poesio M.** Inter-coder agreement for computational linguistics, *Journal of Computational Linguistics*, 2008, vol. 34, no. 4, pp. 555—596.

38. **Segalovich I.** A Fast Morphological Algorithm with Unknown Word Guessing Induced by a Dictionary for a Web Search Engine, *MLMTA-2003*, 2003, pp. 273—280.

39. **Pedregosa F., Varoquaux G., Gramfort A., Michel V., Thirion B., Grisel O., Blondel M., Prettenhofer P., Weiss R., Dubourg V., Vanderplas J., Passos A., Cournapeau D., Brucher M., Perrot M., Duchesnay E.** Scikit-learn: Machine Learning in Python, *JMLR*, 2011, vol. 12, pp. 2825—2830.

40. **Joulin A., Grave E., Bojanowski P., Mikolov T.** Bag of tricks for efcient text classication, arXiv preprint, arXiv:1607.01759, 2016.

41. **Manning C. D., Raghavan P., Schudze H.** *Introduction to Information Retrieval*, Cambridge University Press, 2008, 506 p.

42. **Wilcoxon F.** Individual comparisons by ranking methods, *Biometrics Bulletin*, 1945, vol. 1, no. 6, pp. 80—83.

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

Сдано в набор 11.12.2019 г. Подписано в печать 24.01.2020 г. Формат 60×88 1/8. Заказ PI120
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**

Рисунки к статье **М. Б. Кузьминского, А. М. Чернецова**

«СОВРЕМЕННЫЕ СРЕДСТВА ПАРАЛЛЕЛЬНОГО ПРОГРАММИРОВАНИЯ В МОДЕЛИ РАСПРЕДЕЛЕННОЙ ПАМЯТИ»

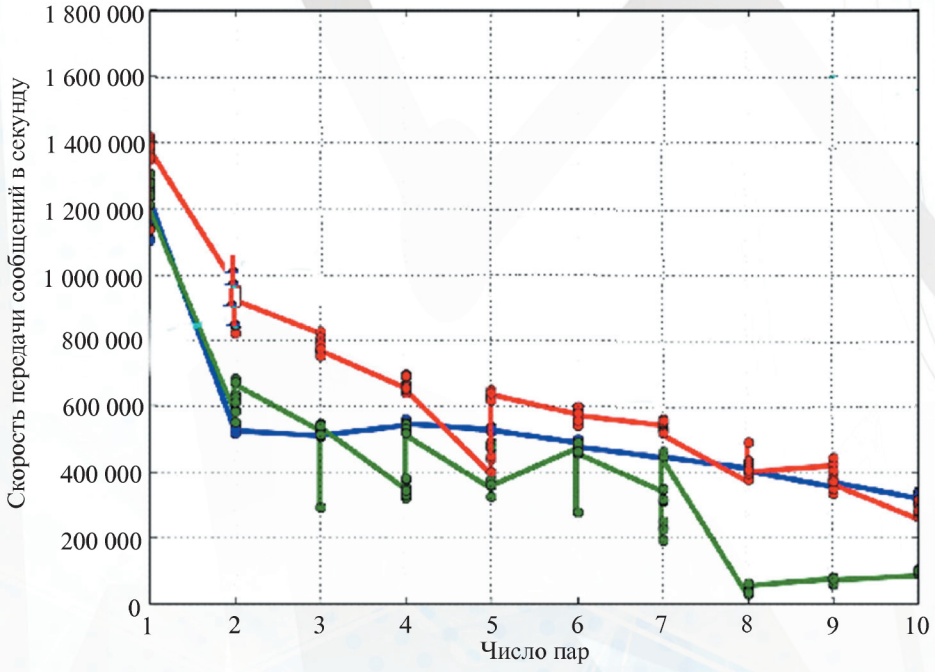

Рис. 3. Скорость операции MPI_THREAD_MULTIPLE для OpenMPI поверх Infiniband EDR:
Intel MPI; OpenMPI; OpenMPI UCX;

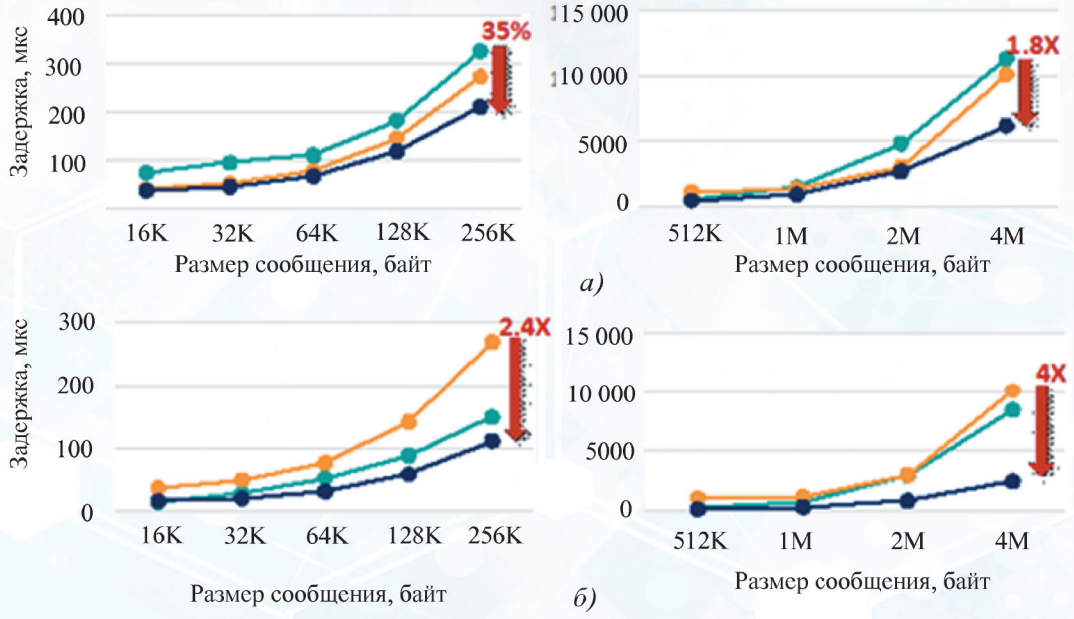

Рис. 4. Уменьшение задержек за счет использования XPMEM:
а – тест OSU_AllReduce; *б* – тест OSU_Reduce; MVAPICH2-2.3rc1;
Intel MPI 2017; MVAPICH2-XPMEM

Издательство «НОВЫЕ ТЕХНОЛОГИИ»

выпускает *научно-технические журналы*

Теоретический и прикладной научно-технический журнал

ПРОГРАММНАЯ ИНЖЕНЕРИЯ

В журнале освещаются состояние и тенденции развития основных направлений индустрии программного обеспечения, связанных с проектированием, конструированием, архитектурой, обеспечением качества и сопровождением жизненного цикла программного обеспечения, а также рассматриваются достижения в области создания и эксплуатации прикладных программно-информационных систем во всех областях челове ческой деятельности.

Подписной индекс по Объединенному каталогу «Пресса России» – 22765

Подписной индекс по Объединенному каталогу «Пресса России» – 72656

Ежемесячный теоретический и прикладной научно-технический журнал

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

В журнале освещаются современное состояние, тенденции и перспективы развития основных направлений в области разработки, производства и применения информационных технологий.

Ежемесячный междисциплинарный теоретический и прикладной научно-технический журнал

НАНО- и МИКРОСИСТЕМНАЯ ТЕХНИКА

В журнале освещаются современное состояние, тенденции и перспективы развития нано- и микросистемной техники, рассматриваются вопрос разботки и внедрения нанои микросистем в различные области науки, технологии и производства.

Подписной индекс по Объединенному каталогу «Пресса России» – 79493

Подписной индекс по Объединенному каталогу «Пресса России» – 79492

Ежемесячный теоретический и прикладной научно-технический журнал

МЕХАТРОНИКА, АВТОМАТИЗАЦИЯ, УПРАВЛЕНИЕ

В журнале освещаются достижения в области мехатроники, интегрирующей механику, электронику, автоматику и информатику в целях совершенствования технологий производства и создания техники новых поколений. Рассматриваются актуальные проблемы теории и практики автомати ческого и автоматизированного управления техническими объектами и технологическими процессами в промышленности, энергетике и на транспорте.

Научно-практический и учебно-методический журнал

БЕЗОПАСНОСТЬ ЖИЗНЕДЕЯТЕЛЬНОСТИ

В журнале освещаются достижения и перспективы в области исследований, обеспечения и совершенствования защиты человека от всех видов опасностей производственной и природной среды, их контроля, мониторинга, предотвращения, ликвидации последствий аварий и катастроф, образования в сфере безопасности жизнедеятельности.

Подписной индекс по Объединенному каталогу «Пресса России» – 79963

Адрес редакции журналов для авторов и подписчиков:
107076, Москва, Стромынский пер., 4. Издательство "НОВЫЕ ТЕХНОЛОГИИ".
Тел.: (499) 269-55-10, 269-53-97. Факс: (499) 269-55-10. E-mail: antonov@novtex.ru

ISSN 2220-3397. Программная инженерия. 2020. Том 11. № 1. 1–64. Подписной индекс по Объединенному каталогу "Пресса России" – 22765