

DEVELOPMENT AND IMPLEMENTATION OF DISTRIBUTED PORTABLE ALGORITHMS OF FRAGMENTED PROGRAMS EXECUTION ON HETEROGENEOUS MULTICOMPUTERS

A. A. Azhbaikov, V. A. Perepelkin*

Novosibirsk State University,
630090, Novosibirsk, Russian Federation,

*Institute of Computational Mathematics and Mathematical Geophysics SB RAS,
630090, Novosibirsk, Russian Federation

Parallel programming automation in numerical computations demands development of effective distributed system algorithms, capable of efficient execution of parallel programs, represented in high-level programming languages. This, in turn, demands conduction of many efficiency tests and experiments for a variety of applications and multicomputers. Of special importance are heterogeneous multicomputers, which comprise computing nodes of various configurations, connected by networks of different capabilities, since high performance computers tend to increase their heterogeneity, as well as different multicomputers tend to join into larger heterogeneous multicomputers. Nowadays there are many possibilities to aggregate various devices (such as cluster nodes, servers, personal computers, tablets and smartphones) into a single highly heterogeneous multicomputer, but there is a lack of software, suitable for conducting numerical computations on such multicomputers with an ability to vary system algorithms.

In the paper a distributed run-time environment is introduced, which is capable of execution of distributed parallel programs, written in LuNA language, on such highly heterogeneous hardware. To achieve good portability web-technologies were employed. Also, the architecture of the run-time environment supports replacement of the most of system algorithms, responsible for resources distribution, static and dynamic load balancing, computations scheduling, garbage collection, network routing etc. Thus, the environment is suitable for studying different system algorithms on highly heterogeneous multicomputers. The LuNA (Language for Numerical Algorithms) language was chosen as the basis because of the computational model, employed in the language. This model (called fragmented algorithm) allows defining computations in a portable, hardware-independent way, without pre-defined resources distribution or computations schedule. Fragmented algorithm represents computations as a set of side-effect-free micro-processes called computational fragments, which process immutable pieces of data called data fragments. In order to execute a fragmented algorithm the run-time environment has to assign fragments to computing nodes and deliver data fragments to their consumers — computational fragments. The problem of fragmented algorithm efficient execution is solved separately from the „numerical“ part of computations. The run-time environment developed is compatible with LuNA compiler (i. e. it executes LuNA-program internal representation, produced by LuNA compiler).

The run-time environment developed focuses on portability, parameterization and scalability. Portability is necessary to allow usage of wide specter of computing nodes (and, thus, support a variety of heterogeneous multicomputers). Parameterization means the ability to replace various system algorithms with user-provided ones in order to study them in the field. Scalability is implied by large-scale numerical computations, where no centralized algorithms, communications or data structures are allowed unless demanded by the application.

The paper proposes an analysis of fragmented algorithm execution in order to identify system algorithms, which are needed to cover the problem of efficient fragmented algorithm execution. According to the analysis architecture of the run-time environment is proposed, which provides necessary parametrization and scalability. For portability concerns the run-time environment was implemented in JavaScript and can be run in virtually any web-browser or under the Node.JS platform. All run-time environment instances form a peer-to-peer network using the Web Sockets technology. The topology of the network can also be controlled by a user module.

Some preliminary testing was conducted. A model problem was studied on a number of multicomputer configurations. The configurations included different nodes: personal multicore computer under OS Windows, a multicore server under Linux, a notebook, and Android smartphones. The tests conducted showed identical output to the output, produced by the original LuNA run-time system. Comparative performance tests were also conducted, which showed the expected curves of different parallelization efficiencies for different computation volume to data volume ratios. Further research should include study of a number of real applications and different system algorithms.

Key words: fragmented programming technology fragmented programming system LuNA, web-technologies.

References

1. Cloud Haskell. [Electron. Res.]: <http://haskell-distributed.github.io/> (accessed: 01.04.2019).
2. Carlton M., Van Roy P. A distributed Prolog system with AND-parallelism // Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences. Volume II: Software track. 1988.
3. Kale, Laxmikant V. and Bhatele, Abhinav. Parallel Science and Engineering Applications: The Charm++ Approach. Taylor & Francis Group, CRC Press, 2013. ISBN 978-1-4665-0412-7
4. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., Dongarra, J. PaRSEC: Exploiting Heterogeneity to Enhance Scalability // IEEE Computing in Science and Engineering. November, 2013. Vol. 15. N 6. P. 36–45.
5. X10 for High Performance Scientific Computing. Josh Milthorpe. Ph.D. Thesis, Research School of Computer Science, Australian National University, June 2015.
6. Treichler Sean, Bauer Michael, Sharma Rahul, Slaughter Elliott, and Aiken Alex. Dependent Partitioning // In Object Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016).
7. Malyshkin Victor E., Perepelkin Vladislav A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // Parallel Computing Technologies. 11th International Conference, PaCT 2011, Proceedings. LNCS 6873. Springer, 2011. P. 53–61.
8. Malyshkin. V., Perepelkin. V., Schukin G. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // Journal of Supercomputing, S.I.: Parallel Computing Technologies — 2016. Springer, 2016. P. 1–7. DOI: 10.1007/s11227-016-1781-0.
9. Boost C++ Libraries. [Electron. Res.]: <https://www.boost.org/> (accessed: 01.04.2019)
10. Message Passing Interface (MPI) Forum. [Electron. Res.]: <https://www.mpi-forum.org/> (accessed: 01.04.2019).
11. Javascript performance benchmarks. [Electron. Res.]: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/javascript.html> (accessed: 01.04.2019).

РАЗРАБОТКА И РЕАЛИЗАЦИЯ ПЕРЕНОСИМЫХ АЛГОРИТМОВ РАСПРЕДЕЛЕННОГО ИСПОЛНЕНИЯ ФРАГМЕНТИРОВАННЫХ ПРОГРАММ НА НЕОДНОРОДНЫХ ВЫЧИСЛИТЕЛЯХ

А. А. Ажбаков, В. А. Перепелкин

Новосибирский национальный исследовательский государственный университет,
630090, Новосибирск, Россия

*Институт вычислительной математики и математической геофизики СО РАН,
630090, Новосибирск, Россия

УДК 004.4'23

DOI: 10.24411/2073-0667-2019-00007

Разработка системных алгоритмов автоматизации параллельного программирования для неоднородных мультимикомпьютеров сопряжена с проведением множества экспериментальных исследований. В работе предлагается распределенная среда исполнения программ на языке LuNA в качестве платформы, пригодной для экспериментального исследования различных системных алгоритмов на мультимикомпьютерах с высокой степенью неоднородности. К таким алгоритмам относятся распределение ресурсов, планирование вычислений, сборка мусора и т. п. Разработанная среда исполнения использует web-технологии в качестве базовой платформы для обеспечения высокой переносимости, а архитектура среды допускает замену большинства системных алгоритмов. Приводятся результаты тестирования разработанной системы на ряде модельных задач.

Ключевые слова: фрагментированное программирование, параллельное программирование, web-технологии, система LuNA.

Введение. Применение высокопроизводительных вычислительных систем, таких как вычислительные кластеры или грид-системы, для научного численного моделирования сопряжено с трудностями разработки и отладки параллельных программ, реализующих требуемые прикладные алгоритмы. Эти трудности обусловлены во многом необходимостью решать ряд проблем, не связанных напрямую с прикладным алгоритмом, но возникающих из-за необходимости организовать распределенную и при этом эффективную обработку данных (тут и далее эффективность понимается в смысле времени выполнения программы, расхода памяти, нагрузки на коммуникационную сеть и т. п.). В частности, приходится решать такие проблемы как декомпозиция данных и вычислений, синхронизация доступа к общим ресурсам, распределение и динамическое перераспределение данных и вычислений по узлам мультимикомпьютера и т. п. Решение этих проблем относится к сфере системного параллельного программирования и может быть существенно сложнее, чем программирование прикладной части программы. Особенно остро эти проблемы возникают в неоднородных вычислителях, характеризующихся различной аппаратной конфигурацией вычислительных узлов и неоднородной сетью. В условиях неоднородного вычислителя обеспечение эффективной работы мультимикомпьютера является существенно более сложной проблемой. При этом имеется тенденция к увеличению неоднородности

мультикомпьютеров. С одной стороны, однородные вычислители (например, вычислительные кластеры) объединяют вместе с образованием неоднородного вычислителя. С другой стороны, все больше различных устройств (ноутбуки, планшеты, телефоны) становятся достаточно производительными, чтобы представлять интерес для их использования в научном моделировании.

Для преодоления этих трудностей разрабатываются системы программирования [1–8], автоматизирующие часть этой работы и повышающие уровень программирования для пользователя. И хотя в общей постановке эффективная параллельная реализация алгоритма, представленного на высоком уровне абстракции, является алгоритмически труднорешаемой проблемой (что не позволяет рассчитывать на универсальное ее решение), системы программирования существенно облегчают процесс разработки и отладки параллельных программ за счет того, что вбирают в себя различные частные алгоритмы и эвристики, позволяющие конструировать эффективные параллельные программы в частных случаях. По мере развития систем программирования их область применения расширяется, а эффективность конструируемых программ — повышается.

Важную роль в процессе разработки системных алгоритмов играет их экспериментальное исследование на реальных задачах и аппаратном обеспечении. Для проведения таких экспериментов целесообразно иметь некоторую программную платформу, способную проводить вычисления на мультикомпьютерах с высокой степенью неоднородности и позволяющую независимо от вычислительной части алгоритма решать задачу распределения и перераспределения данных и вычислений, выбора управления, сборки мусора и других задач, определяющих эффективность исполнения алгоритма. Далее будем обобщенно называть эти задачи задачей управления поведением программы, имея в виду, что поведение программы — это множество допустимых способов ее исполнения, из которых требуется выбрать (и реализовать) эффективный.

В настоящее время существует и активно развивается большое количество систем программирования, автоматизирующих те или иные этапы и виды работ при разработке параллельных программ численного моделирования. Модель вычислений (то, как представляется прикладная задача) определяет конкретную постановку задачи управления поведением для системы. С точки зрения экспериментального исследования системных алгоритмов модель вычислений должна быть достаточно абстрактной и гибкой, чтобы в рамках этой модели возможно было исследовать подзадачи управления поведением, а программная реализация системы должна допускать замену системных алгоритмов управления поведением на исследуемые.

В функциональных и логических языках и системах (см., например, [1, 2]), кардинально повышающих уровень программирования, эффективная распределенная реализация вычислений практически недостижима (за исключением узкого класса прикладных задач), так как модель вычислений слишком абстрактна, а количество вариантов реализации очень обширно. Другой класс систем — это низкоуровневые средства программирования. Их использование для экспериментирования с системными алгоритмами управления поведением затруднительно ввиду отсутствия высокоуровневой модели, которую можно было бы взять за основу для постановки задачи управления поведением. Основным интересом представляет промежуточный уровень, на котором находятся системы программирования, имеющие в своей основе высокоуровневую модель вычислений (такие как Charm++ [3], PaRSEC [4], X10 [5], Legion [6] и многие другие).

Главным недостатком этих систем с точки зрения использования в качестве платформы для исследования системных алгоритмов управления поведением является то, что они создавались для того, чтобы скрыть от пользователя сложности системного программирования. Возможности же модификации системных алгоритмов ограничены или требуют непосредственной модификации исходного кода системы.

Например, объектно-ориентированная система программирования Charm++ [3] имеет в своей основе удобную модель вычислений, в которой прикладной алгоритм описывается в виде набора сериализуемых объектов, обменивающихся сообщениями. Такая модель позволяет исполнительной среде Charm++ прозрачно для пользователя перемещать объекты с одного вычислительного узла на другой, контролировать очередность обработки переданных сообщений, сохранять состояние отдельных объектов и системы в целом и т. п. При этом на среду исполнения накладываются обязательства по поиску объектов на распределенном вычислителе, обеспечению баланса нагрузки, доставке сообщений. Таким образом, модель вычислений Charm++ представляется вполне подходящей для проведения экспериментов с различными алгоритмами управления поведением, однако программная реализация не предусматривает замену системных модулей, отвечающих за управление поведением, и может выполняться лишь в форме редактирования исходного кода системы. Исключения составляют отдельные системные алгоритмы, такие как модуль динамической балансировки нагрузки на вычислитель, но в целом системные алгоритмы зафиксированы в коде системы.

Кроме того, системы программирования, используемые для высокопроизводительных вычислений, обычно переносимы лишь в классе настольных и серверных компьютеров и не могут функционировать на мультикомпьютерах, имеющих в составе еще и мобильные устройства (например, на базе ОС Android), что ограничивает класс поддерживаемых мультикомпьютеров с точки зрения их неоднородности.

Из вышесказанного видно, что существующие системы лишь в ограниченной степени подходят для использования в качестве платформы для экспериментального исследования системных алгоритмов исполнения высокоуровневых программ на неоднородных вычислителях. Среди прочих систем наиболее подходящей для обозначенных целей представляется система автоматизации конструирования параллельных программ LuNA [7, 8], т. к. в ней управление ходом исполнения алгоритма явно отделяется от содержательной части вычислений (отдельно ставится задача управления поведением, как и в системе Legion [6]). Наиболее существенным недостатком системы LuNA в плане поддержки неоднородных вычислителей является ограниченная переносимость среды исполнения (runtime-системы): привязка к C++ и библиотекам Boost [9] и MPI [10].

В настоящей работе предлагается распределенная среда исполнения LuNA-программ с существенно более высокой степенью переносимости. Переносимость достигается использованием веб-технологий как платформы, обладающей широчайшей переносимостью и при этом достаточно высокой для обозначенных целей производительностью (отставание по производительности от языка C составляет от 1,5 до 10 раз, в зависимости от типа задачи [11]). Акцент в предлагаемой системе делается на создании инструментария для исследования распределенных системных алгоритмов исполнения высокоуровневых программ. В частности, системные алгоритмы распределения ресурсов, выбора порядка выполнения вычислений, сборки мусора и т. п. являются заменяемыми модулями системы, а в систему встроены средства профилирования и оценки эффективности исполнения программы.

Рассмотрим требования, предъявляемые к среде исполнения.

— Переносимость — возможность запуска системы на разнородных вычислительных устройствах, а также одновременное использование разнородных вычислительных ресурсов в распределенных вычислениях.

— Параметризуемость системными алгоритмами распределенного исполнения параллельных программ в неоднородной среде. Исследовательская система предназначена для проведения распределенного исполнения параллельных программ под управлением пользовательских алгоритмов, которые являются предметом исследования пользователя — разработчика системных алгоритмов. Необходимо определить виды алгоритмов, которые могут выступать предметом исследований, и спроектировать архитектуру и интерфейсы исследовательской платформы таким образом, чтобы обеспечить возможность их подключения к системе, а также определить способ представления пользовательских алгоритмов. Таким образом, алгоритм работы самой исследовательской платформы использует подключенные пользовательские системные алгоритмы для выполнения реализуемых ими процессов управления распределенным исполнением программ.

— Масштабируемость — алгоритм работы системы распределенного исполнения программ не должен содержать принципиальных ограничений для одновременного использования произвольного количества вычислительных устройств. Это подразумевает отсутствие централизации в системных алгоритмах распределенной среды исполнения. Допустимы ограничения масштабируемости, вызванные конкретной реализацией пользовательских системных алгоритмов или прикладной задачей.

Оставшаяся часть статьи организована следующим образом. В разделе 1 рассматриваются модель распределенного исполнения фрагментированной программы и ее анализ. В разделе 2 предлагаются архитектура и алгоритмы среды исполнения. В разделе 3 представлены описание реализации и описание результатов экспериментального исследования. В заключении подведены итоги работы и обозначены дальнейшие направления исследования.

1. Модель исполнения фрагментированной программы.

В данном разделе формулируются определения технологии фрагментированного программирования, представляется модель исполнения фрагментированной программы в технологии фрагментированного программирования в общем случае и в условиях распределенной среды исполнения, а также выполняется анализ модели с целью выявления функций среды исполнения и их особенностей.

1.1. *Термины и определения. Фрагментированная программа.* В технологии фрагментированного программирования фрагментированная программа представляет собой описание прикладного алгоритма в виде операторного выражения. Выражение задает множество операторов, которые должны быть выполнены вычислителем в соответствии с семантикой, и множество переменных, значения которых вычисляются в ходе исполнения операторов. Значения переменных в технологии фрагментированного программирования называют *фрагментами данных*, а операторы которые их производят — *фрагментами вычислений*.

Исполнение фрагментированной программы заключается в вычислении этого выражения над некоторым входным набором данных. Вычисление выражения, то есть выполнение задаваемого им множества операторов над входными данными, описывается потенциально бесконечным *вычислительным графом*.

Вычислительный граф. Вычислительный граф программы P на входном наборе данных I — ориентированный граф $G_{P, I} = \langle V, A \rangle$, где множество вершин $V = D \cup O$,

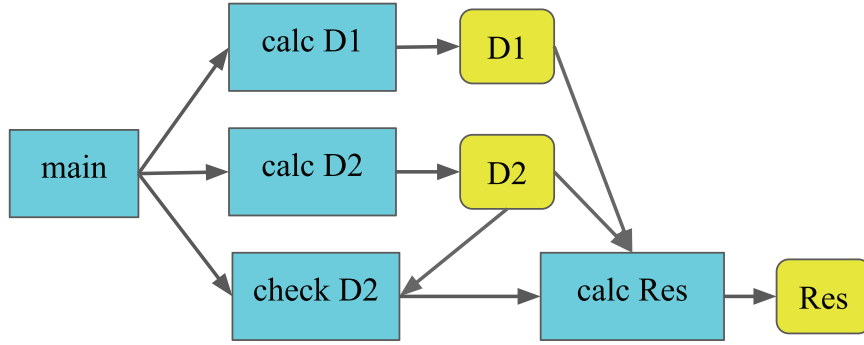


Рис. 1. Пример вычислительного графа; прямоугольники — операторы, скругленные прямоугольники — фрагменты данных

D — множество переменных единственного присваивания,
 O — множество операторов $\{op = \langle D_{in\ op}, D_{out\ op}, O_{out\ op} \rangle\}$, где
 $D_{in\ op} \subseteq D$ — множество входных переменных оператора op ,
 $D_{out\ op} \subseteq D$ — множество выходных переменных оператора op ,
 $O_{out\ op} \subseteq O$ — множество выходных операторов оператора op ,
 множество дуг $E = U_{D_{in}} \cup U_{D_{out}} \cup U_{O_{out}}$, где

$$U_{D_{in}} = \bigcup_{op \in O} (D_{in\ op}, op), U_{D_{out}} = \bigcup_{op \in O} (op, D_{out\ op}), U_{O_{out}} = \bigcup_{op \in O} (op, O_{out\ op}).$$

Исполнение оператора приводит к вычислению значения его выходных переменных. Исполнение оператора возможно, только если вычислены значения всех его входных переменных. Выходные операторы оператора op не могут быть исполнены до исполнения оператора op .

В LuNA исполнение программ начинается с оператора `main` по аналогии с процедурой `main` в традиционных языках программирования (рис. 1).

Исполнение фрагментированной программы. Будем называть *состоянием исполнения* фрагментированной программы P на начальном наборе данных I подграф $S = \langle V_S, A_S \rangle$, $V_S = D_S \cup O_S$ вычислительного графа $G_{P, I}$, такой что:

- если $d \in D_S$, то $\forall op \in O \mid d \in D_{out\ op}$ выполняется $op \in O_S$ и $(op, d) \in A_S$;
- если $o \in O_S$, то $\forall op \in O \mid o \in O_{out\ op}$ выполняется $op \in O_S$ и $(op, 0) \in A_S$.

Таким образом, состояние исполнения содержит операторы, которые были исполнены, и переменные, значения которых были вычислены, а также операторы, которые являются прямыми потомками исполненных операторов. Обозначим множество операторов, *готовых к исполнению* в состоянии исполнения S : $R_S = \{op \in O_S \mid D_{in\ op} \subseteq D_S \text{ и } op \text{ не был исполнен ранее}\}$ (для упрощения изложения не будем описывать разбиение O_S на исполненные и неисполненные операторы).

Состояние исполнения $S' = \langle V_{S'}, A_{S'} \rangle$ получается из состояния исполнения S *исполнением оператора* $op \in R_S$:

$$V_{S'} = V_S \cup D_{out\ op} \cup O_{out\ op},$$

$$A_{S'} = A_S \cup A_{outop} \cup A_{rest}, \text{ где } A_{outop} = \{(op, d) \mid d \in D_{out\ op} \ d \in O_{out\ op}\}, A_{rest} = \{(d_1, d_2) \mid ((d_1 \in A_{outop} \ d_2 \in V_S) \ (d_1 \in V_S \ d_2 \in A_{outop})) \ (d_1, d_2) \in A\}.$$

То есть состояние исполнения S было расширено выходными операторами и переменными оператора op , а также всеми дугами, которыми они связаны с вершинами S в вычислительном графе.

Пусть $S_0 = \langle \{main\}, \emptyset \rangle$ — начальное состояние, $S_n = G_{P, I}$ — конечное состояние.

Потенциально бесконечную последовательность состояний $\{S_0, \dots, S_n, \dots\}$, где S_{i+1} получено из S_i исполнением оператора из R_{S_i} , будем называть *исполнением фрагментированной программы P на наборе входных данных I* . Вопросы корректности LuNA-программ в данной статье не рассматриваются, будем считать, что такая последовательность существует.

Отметим, что фрагментированная программа P и набор входных данных I не определяют последовательность состояний исполнения однозначно — зачастую множество R_{S_i} содержит более одного оператора, готового к исполнению, они могут быть использованы для перехода в следующее состояние в разном порядке. Это соответствует недетерминированности порядка исполнения операторов параллельной программы.

1.2. Исполнение фрагментированной программы в распределенной среде.

Чтобы добавить в модель исполнения фрагментированной программы детали исполнения на мультикомпьютере, приведем краткую общую характеристику архитектуры мультикомпьютера и примерный сценарий исполнения программы на нем и уточним модель исполнения для условий мультикомпьютера.

Общая архитектура мультикомпьютера. Требование масштабируемости подразумевает отказ от централизованного управления узлами мультикомпьютера. Система распределенного исполнения программ должна состоять из множества самостоятельных узлов, каждый из которых находится под управлением экземпляра среды исполнения LuNA, а разработка алгоритма работы среды исполнения сводится к разработке алгоритма работы экземпляра среды исполнения, запускаемого на узлах мультикомпьютера. Каждый такой экземпляр обладает локальной памятью (не общей) и имеет возможность обмениваться сообщениями с некоторым подмножеством узлов вычислительной сети. Граф связности узлов должен быть связным, но не обязательно полным.

Примерный сценарий распределенного исполнения фрагментированной программы. В начальный момент времени оператор `main` загружен на один из узлов мультикомпьютера, узлы мультикомпьютера связаны между собой, образуя некоторую сетевую топологию, LuNA-программа загружена на каждый узел мультикомпьютера. Узел-лидер выполняет оператор вызова подпрограммы `main` и распределяет результирующие операторы по узлам вычислительной сети в соответствии с используемым системным алгоритмом для последующего исполнения. В случае выполнения оператора, присваивающего значения переменным, результирующие фрагменты данных аналогичным образом назначаются на хранение в локальную память одного или нескольких узлов мультикомпьютера. При наличии у оператора входных зависимостей по данным для каждой входной переменной определяется узел хранения ее значения и направляется запрос на получение значения. Исполнение программы заканчивается, когда выполнены все операторы, составляющие вычислительный граф исполняемого фрагментированного алгоритма с выбранными входными данными.

Формализуем понятие мультикомпьютера как связный граф $M = \langle N, C \rangle$, N — множество вычислительных узлов мультикомпьютера, $C = \{(n_1, n_2) \mid n_1, n_2 \in N, n_1 \neq n_2\}$ — двунаправленные каналы связи между узлами мультикомпьютера.

Будем считать, что исполнение $\{S_0, \dots, S_n, \dots\}$ фрагментированной программы P на наборе входных данных I на мультикомпьютере M соответствует множеству отображений $Q = \{Q_0, \dots, Q_n\}$, определяющих хранение состояний исполнения на узлах мультикомпьютера, где $Q_i : V_{S_i} \rightarrow 2^N$ такое, что:

— если S_{i+1} получено из S_i исполнением оператора op , то $\exists n \in N \mid n \in Q_i(op) \quad n \in Q_i(d) \quad \forall d \in D_{in \ op}$;

— если $\exists i \in [0, n] \quad \exists d \in D_{S_i} \mid Q_i(d) = \emptyset$, то $\forall j > i$ выполняется $Q_j(d) = \emptyset$.

Такое определение фиксирует следующие факты:

— Для исполнения программы на мультикомпьютере на каждом состоянии исполнения все невыполненные операторы, т. е. еще не послужившие для перехода в другое состояние исполнения, и их входные переменные должны находиться в памяти одного или нескольких узлов мультикомпьютера;

— Операторы и переменные могут мигрировать, т. е. между состояниями исполнения узлы хранения одного и того же оператора или переменной могут отличаться;

— Для перехода в очередное состояние исполнения путем исполнения оператора, необходимо, чтобы существовал узел мультикомпьютера, в памяти которого одновременно находится данный оператор и все входные переменные этого оператора;

— Память для хранения оператора или переменной может быть освобождена — этому соответствует отображение в пустое подмножество, такой оператор или переменная больше не могут появиться в памяти мультикомпьютера после освобождения памяти.

1.3. *Анализ функционала среды исполнения в процессе исполнения фрагментированной программы.*

Проанализируем приведенную модель исполнения фрагментированной программы на мультикомпьютере.

— Отображение операторов и переменных для каждого состояния исполнения на узлы мультикомпьютера соответствует распределению нагрузки на память мультикомпьютера;

— Отображение оператора $op \in R_S$ для состояния S на один узел со всеми $d \in D_{in \ op}$ определяет узел исполнения оператора op , что соответствует осуществлению балансировки вычислительной нагрузки мультикомпьютера;

— Отображение операторов и переменных в пустое множество соответствует распределенной „сборке мусора“;

— Отображение оператора или переменной более чем на один узел мультикомпьютера соответствует избыточному хранению, что имеет значение в обеспечении отказоустойчивости, повышения доступности данных и т. п.;

— Выбор путей миграции операторов и переменных соответствует балансировке нагрузки на каналы связи между узлами мультикомпьютера.

Таким образом, алгоритм работы распределенной среды исполнения, задавая множество отображений Q , может осуществлять перечисленные процессы для обеспечения эффективного (в смысле использования времени и ресурсов) исполнения параллельной программы.

Поскольку множество отображений Q задается алгоритмом работы среды исполнения, оно может формироваться в динамике, т. е. отображение Q_i может быть вычислено на основании $S_j \mid j < i$ с использованием дополнительной системной информации, статической или собранной средой исполнения в динамике. Множество факторов может быть учтено при принятии решения, в частности, статические и динамические характеристики вычислительной сети, такие как граф связности узлов мультикомпьютера, латентность и

пропускная способность каналов связи, объем памяти и производительность узлов мультимпьютера, количество вычислительных ядер, загруженность каналов связи и вычислительных узлов в данный момент и статистика за предыдущий отрезок времени, длина такого отрезка времени.

Граф связности вычислительной сети мультимпьютера также является одним из ключевых факторов распределенного исполнения параллельной программы, исследовательская платформа должна предоставлять возможность управлять связыванием разнородных вычислителей в вычислительную сеть.

Кроме отображения состояний исполнения на физические ресурсы мультимпьютера, среда исполнения осуществляет также исполнение операторов в рамках узла мультимпьютера. *Способ организации параллельного исполнения оператора и параллельной обработки множества операторов*, назначенных на данный узел мультимпьютера, непосредственно влияет на время распределенного исполнения фрагментированной программы.

Другой важной деталью является *стратегия исполнения операторов с большим количеством результирующих операторов* — при достаточно большом количестве выходных операторов и фрагментов данных исполнение оператора может привести к переполнению памяти мультимпьютера или скачку сетевой нагрузки. Примером такой ситуации является исполнение оператора, реализующего цикл `for` с достаточно большим диапазоном значений счетчика. Для уменьшения влияния скачка нагрузки возможно, например, производить выходное множество операторов и данных по частям, дожидаясь их использования и освобождения памяти. Данные сценарии представляют интерес для исследования, и системный алгоритм, реализующий их, должен быть представлен в архитектуре системы в виде отдельного модуля в соответствии с требованием параметризуемости.

Таким образом, выявлены основные функции среды исполнения, большая часть которых осуществляется путем управления распределением операторов и фрагментов данных по узлам мультимпьютера.

2. Предлагаемая архитектура среды исполнения LuNA и алгоритм ее работы.

Исходя из аспектов исполнения фрагментированной программы, выделенных в предыдущем разделе, была разработана модульная архитектура экземпляра среды исполнения LuNA (рис. 2).

Коммуникационный модуль. Предоставляет низкоуровневый программный интерфейс для передачи сообщений другим экземплярам среды исполнения LuNA. Не обладает представлением об устройстве вычислительной сети в целом, оперирует лишь каналами связи с соседними вычислительными узлами, которые идентифицирует по некоторому ключу-направлению, задаваемому топологией сети (например „левый сосед“, „правый сосед“, если узлы объединены в „линейку“). Позволяет реализовать различные способы передачи сообщений соседним узлам вычислительной сети и отслеживать состояние канала связи и статистику его использования.

Модуль исполнения. Данный модуль осуществляет исполнение операторов LuNA — извлекает список аргументов оператора, запрашивает фрагменты данных, соответствующие аргументам, в модуле хранилища, обрабатывает тела операторов согласно их типу, вызывает элементарную подпрограмму для атомарных операторов, передает результирующие операторы и фрагменты модулю хранилища (см. листинг 1). Модуль позволяет ре-

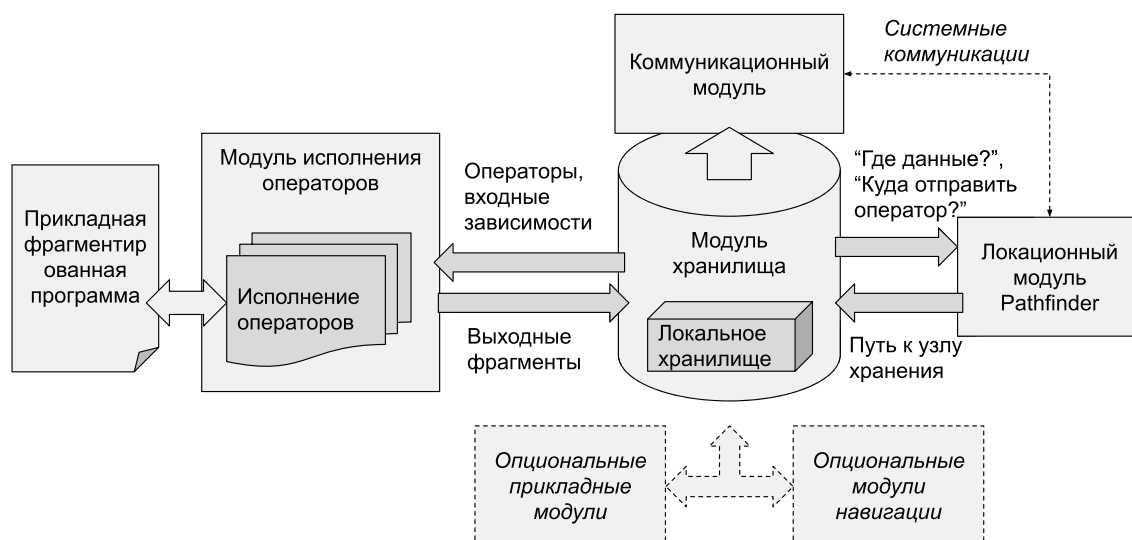


Рис. 2. Модульная архитектура экземпляра среды исполнения

ализовать различные способы работы с циклами и задействовать параллелизм на уровне одного узла мультикомпьютера.

Листинг 1. Псевдокод простого алгоритма работы модуля исполнения

```
// Основной цикл
// Работа останавливается по команде алгоритма обнаружения завершения
while (!stop) {
  if (StorageModule.IsLocalOpStorageEmpty()) WaitForOps();

  // Исполнение операторов, назначенных на текущий узел
  foreach cf in StorageModule.LocalOpStorage {
    // Запросить аргументы
    var args = [];
    foreach dfId in GetArgumentList(op) {
      args.Add(await StorageModule.GetDf(dfId));
      // Модуль хранилища выполнил сетевые запросы
    }

    // Выполнение оператора
    var newFrag = ExecOpWithArgs(op, args);

    // Сохранение результирующих фрагментов
    // в память текущего или удаленного узла
    StorageModule.Add(newFrag);
  }
}
```

Модуль хранилища. Отвечает за хранение операторов и фрагментов данных, абстрагирует распределенный характер системы. Использует модуль навигации Pathfinder

(см. далее) для определения узла хранения фрагмента и коммуникационный модуль для передачи и приема фрагментов и системных сообщений, скрывая таким образом трудности работы с распределенной памятью от модуля исполнения (см. листинг 2).

Листинг 2. Псевдокод примерного алгоритма работы модуля хранения

```
void PutOp(Op op) {
    var targetNodePath = PathfinderNavModule.GetDestination(op);
    if (IsCurrentNode(targetNodePath)) StoreInLocalStorage(op);
    else CommModule.Send(op, targetNodePath);
}

void GetOp(OpId) {
    var storageNodePath = PathfinderNavModule.GetDestination(opId);
    if (IsCurrentNode(storageNodePath)) return GetFromLocalStorage(opId);
    else {
        var opRequest = CreateRequestMessage(opId);
        CommModule.Send(opRequest, storageNodePath)
        return await WaitForAnswer();
    }
}
// Аналогично для фрагментов данных
```

В данном модуле может быть реализовано отказоустойчивое хранение, а также алгоритмы сборки мусора. При этом для обеспечения дополнительных системных коммуникаций в локационный модуль Pathfinder должна быть добавлена навигационная логика для системных сообщений.

Подобный подход, когда модуль, осуществляющий сетевые коммуникации, тесно связан со своим навигационным модулем, может быть использован при реализации любых других подсистем, подробнее об этом в пункте „Навигация сообщений“.

Локационный модуль (Pathfinder). Выполнение очередного оператора LuNA включает разрешение его входных зависимостей и их загрузку из распределенной памяти, а также сохранение результирующих операторов и фрагментов данных в распределенную память. Функция навигации, реализованная в данном модуле, для каждого фрагмента определяет узел мультикомпьютера, локальная память которого должна быть использована для хранения фрагмента.

Таким образом, локационный модуль может осуществлять динамическую балансировку вычислительной нагрузки между узлами мультикомпьютера, собирая дополнительную информацию о загруженности окружающих узлов и используя ее для принятия решения о назначении оператора на узел мультикомпьютера. Также здесь может быть реализована логика избыточного хранения для обеспечения отказоустойчивости, путем назначения нескольких узлов мультикомпьютера для хранения одного узла. Возможна реализация „упреждающей отправки“, когда фрагмент данных заранее отправляется на узлы вычислительной сети, на которые поступят зависящие от него операторы.

Таким образом, локационный модуль реализует отображение Q из модели исполнения фрагментированной программы на мультикомпьютере.

Отметим, что порядок исполнения операторов в среде исполнения, и, следовательно, порядок следования состояний исполнения $\{S_0, \dots, S_n, \dots\}$, определяется совокупно-

стью модулей — локационный модуль может поместить оператор на высоконагруженный узел и отложить таким образом его исполнение относительно оператора, направленного на свободный узел, а реализация модуля исполнения и модуля хранилища влияют на порядок исполнения операторов, находящихся в очереди на одном узле.

Навигация сообщений. Некоторые системные алгоритмы могут осуществлять сетевые коммуникации по собственному протоколу, например алгоритм обнаружения остановки. В ходе таких коммуникаций как правило возникают две задачи: определить, какому из соседних узлов вычислительной сети направить сообщение, чтобы оно достигло узла-получателя, либо определить узел-получатель и проложить маршрут к нему посредством других узлов, если узел-получатель не имеет прямого соединения с узлом-отправителем. Решение таких задач предполагает знание об устройстве окружающей сети и наличие логики маршрутизации. Предлагается вынести данную логику из модуля, реализующего исходный системный алгоритм в специальный отдельный модуль — модуль навигации — который выступает прослойкой между прикладным модулем и коммуникационным модулем. Это позволит упростить прикладной модуль и даст возможность использовать одну реализацию навигационного модуля нескольких прикладных. Примером модуля навигации является локационный модуль Pathfinder.

Задача модуля навигации состоит в том, чтобы собрать информацию о топологии и состоянии узлов и каналов связи некоторой подсети вычислительной сети и использовать ее для определения маршрута и/или узла назначения для прикладных сообщений (см. листинг 3 и листинг 4).

Листинг 3. Псевдокод навигации с точки зрения прикладного модуля

```
var message ;
* Сформировать сообщение message *

var path = NavModule.CalculateDestination(message);
CommModule.Send(message, path);
```

Листинг 4. Псевдокод навигации с точки зрения модуля навигации

```
var networkInfo ; // Использовать CommModule
                  // для сбора информации о сети (опционально)

networkInfo = CollectNetworkInfoImpl(CommModule);

Path CalculateDestination(message) {
    var destination = ChooseDestinationImpl(message, networkInfo);
    var gateway = ChooseGatewayImpl(destination, networkInfo);
    return new Path(destination, gateway);
}
```

Алгоритм инициализации вычислительной сети. Этап инициализации одноранговой вычислительной сети подразумевает установку каналов связи между узлами мультимедийного компьютера в соответствии с некоторой сетевой топологией. Поскольку изначально не связанные узлы не могут знать о существовании друг друга, необходим сигнальный сервер — элемент централизации, который сообщит каждому обратившемуся узлу информацию о его месте в топологии сети и поможет обменяться данными „рукопожатия“ (handshake)

для установки соединения с соседями. Сигнальный сервер параметризуется алгоритмом инициализации вычислительной сети, который при обращении очередного узла вычислительной сети назначает ему место в топологии сети. Информация об изменениях в топологии (новые связи) передается на некоторое подмножество узлов вычислительной сети, определяемое топологией, чтобы экземпляры среды исполнения на узлах могли обновить информацию о видимой подсети.

Отметим, что наличие сигнального сервера не нарушает требование масштабируемости, поскольку он используется лишь до начала распределенного исполнения программы и не участвует в передаче данных во время исполнения.

Вывод. Таким образом, все системные задачи исполнения фрагментированной программы, выявленные в ходе анализа, могут быть независимо реализованы с данной модульной декомпозицией экземпляра среды исполнения LuNA. Были рассмотрены основные виды системных алгоритмов, реализующих распределенное исполнение фрагментированной программы, разработана модульная архитектура среды исполнения, позволяющая блочно изменять реализацию системных алгоритмов, на уровне модульной архитектуры соблюдено требование параметризуемости. Архитектура также не накладывает ограничений на добавление новых видов модулей, взаимодействие модулей друг с другом принципиально не ограничено. Предлагаемое решение не накладывает ограничений на масштабируемость системы, предоставлена свобода поддержки вычислительных сетей с произвольным графом связности. Возможность подключения разных реализаций модулей на разных узлах мультикомпьютера, а также возможность реализации произвольных протоколов связи внутри модулей, дает возможность реализовать переносимую, поддерживающую разнородные устройства среду исполнения.

3. Реализация и тестирование распределенной среды исполнения.

3.1. Реализация распределенной среды исполнения фрагментированных программ на базе web-технологий.

Требования к средствам реализации среды исполнения. При выборе средств для реализации среды исполнения фрагментированных программ ключевую роль играет требование переносимости. Необходимо обеспечить возможность одновременно задействовать разнородные вычислительные устройства и сохранить при этом удобство параметризуемости — разработчик системных алгоритмов не должен реализовывать свой алгоритм несколько раз на разных языках программирования, чтобы использовать его на неоднородном оборудовании. Еще одним важным фактором при выборе технологий является удобство развертывания системы — минимизация работы по развертыванию позволит разработчику системных алгоритмов провести эксперименты на большем количестве устройств и сконцентрироваться на прикладной проблеме. Обеспечение максимальной производительности в пределах одного узла вычислительной сети не является требованием при разработке программной платформы для исследования системных алгоритмов распределенного исполнения программ, однако используемые технологии и программная реализация системы не должны препятствовать использованию системы по назначению с точки зрения производительности.

Применение web-технологий для реализации среды исполнения. Исходя из требований, для реализации среды исполнения LuNA были выбраны web-технологии — JavaScript Runtime Environment как окружение для запуска экземпляра среды исполнения, JavaScript в качестве языка среды исполнения и модулей, HTML для графического интерфейса при управлении системой и отображении информации о ходе исполнения.

На каждом узле мультимпьютера запускается веб-браузер или среда исполнения Node.js для выполнения JavaScript-кода среды исполнения LuNA. Одним из способов загрузки среды исполнения является загрузка веб-страницы с прикрепленным JavaScript-кодом среды исполнения и графическим интерфейсом через веб-сервер. Используя интерфейс веб-страницы, пользователь может управлять настройками вычислительной задачи, загружать прикладные программы и модули среды исполнения, а также наблюдать за состоянием вычислительной сети.

Сигнальный сервер является компонентом среды исполнения, он развертывается в зоне доступности для всех экземпляров среды исполнения и служит точкой подключения вычислителя к мультимпьютеру. Он сообщает подключившимся экземплярам среды исполнения информацию об их положении в топологии сети и передает между ними пакеты для установки p2p-соединения.

Был реализован веб-сервер на базе ASP.NET Core, совмещающий функции сигнального сервера и сервера для загрузки веб-страниц с экземплярами среды исполнения. Для связи сервера с экземплярами среды исполнения используется WebSocket, для p2p-связи между экземплярами среды исполнения — WebRTC.

Таким образом, для использования системы достаточно развернуть веб-сервер и подключиться к нему через веб-браузер с устройств, которые необходимо задействовать в качестве узлов вычислительной сети.

Характеристика web-технологий как средства реализации среды исполнения. Выбор web-технологий обусловлен тем, что они изначально разрабатывались для обеспечения платформу-независимого взаимодействия в распределенных неоднородных системах. Сохраняется тенденция обеспечения функцией просмотра веб-страниц самых разных устройств, практически каждая операционная система включает веб-браузер, возможно задействовать широчайший класс мобильных и персональных устройств, существующих и будущих, которые обладают самыми разными с точки зрения неоднородности характеристиками. Также, помимо веб-браузеров JavaScript может исполняться и на высокопроизводительном серверном ПО (Node.js), включая узлы вычислительных кластеров. Таким образом, использование web-технологий позволяет задействовать для создания экспериментального мультимпьютера множество устройств, гораздо более неоднородное, чем позволяют используемые в LuNA C++ и MPI, кроме того, это дешевле, проще и быстрее, чем использование настольных компьютеров или кластерного оборудования для этой цели.

Веб-технологии предоставляют высокую переносимость, масштабируемость, возможность задействовать разнородные вычислительные устройства, развитые средства разработки и перспективы улучшения во всех этих направлениях.

В соответствии с вышеизложенным новая среда исполнения LuNA была программно реализована. Разработаны:

- Клиентское веб-приложение, исполняемое в веб-браузере, образующее ядро системы, параметризуемое модулями;
- Сигнальный сервер, параметризуемый алгоритмами, определяющими топологию вычислительной сети;
- Простейшие реализации модулей.

3.2. Тестирование новой среды исполнения LuNA.

Чтобы убедиться, что новая реализация среды исполнения LuNA может быть использована в качестве программной платформы для экспериментальных исследований систем-

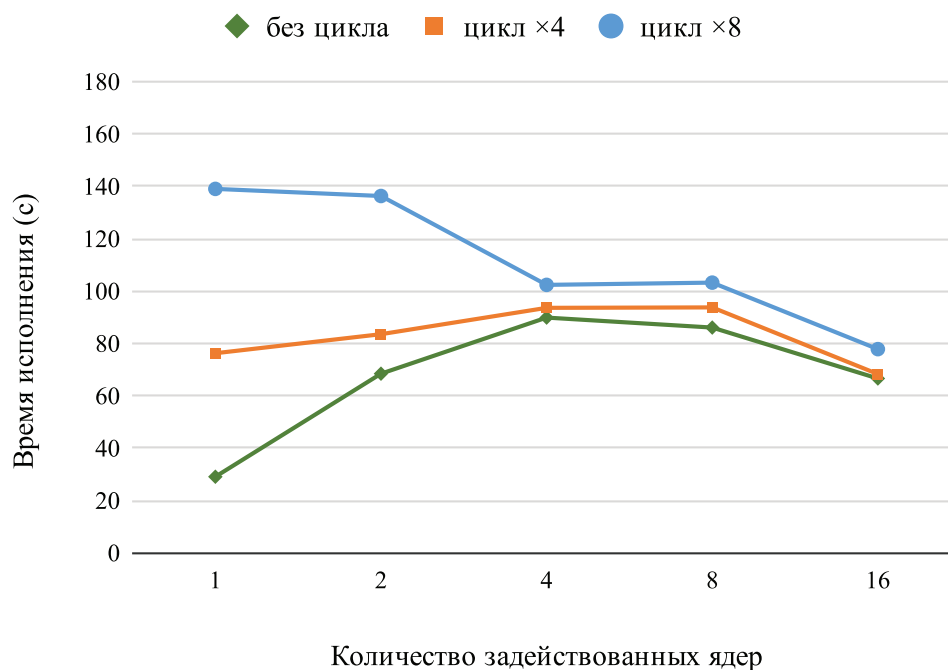


Рис. 3. Влияние реализации навигационного модуля на исполнение тестовой задачи

ных алгоритмов распределенного исполнения программ, был выполнен ряд тестовых запусков, демонстрирующих влияние используемого навигационного модуля на исполнение программы.

В качестве тестовой задачи использовалось решение уравнения Пуассона методом Якоби в трехмерной области размером $200 \times 200 \times 200$, 85 итераций, топология вычислительной сети — одномерная решетка.

Первая серия тестовых запусков демонстрирует влияние используемого навигационного модуля на исполнение программы. Для этого были выполнены запуски среды исполнения с различными реализациями локационного модуля Pathfinder, распределяющими вычислительную нагрузку на 1, 2, 4, 8 и 16 узлов мультикомпьютера из 16 имеющихся. Для наблюдения за влиянием на время исполнения программы накладных расходов на коммуникации в разных классах задач, отличающихся по объемам вычислений на единицу данных, такие классы были смоделированы увеличением времени исполнения некоторых наиболее длительно выполняемых операторов путем многократного их выполнения (4 и 8 повторов). Узлы мультикомпьютера представлялись процессами-вкладками браузера Chrome на компьютере с процессором Intel Core i5-7600K с 8Гб оперативной памяти под управлением ОС Windows 10.

На рис. 3 представлены результаты измерений времени исполнения тестовой задачи.

Первый график, соответствующий запуску без применения зацикливания операторов, показал замедление исполнения с ростом количества задействованных вычислителей. При использовании 8-кратного зацикливания наблюдается ускорение исполнения тестовой задачи с увеличением количества вычислителей, задействованных реализацией локационного модуля, что позволяет сделать вывод о том, что в первом случае на данной конфигура-

ции среды исполнения коммуникационные расходы сопоставимы с временными затратами на выполнение полезных вычислений при данной фрагментации трехмерной области, а во втором — время передачи данных относительно мало и удается получить выгоду от параллельного исполнения.

Также выполнялись серии тестов, направленные на проверку переносимости, тестовая задача была выполнена на различных сочетаниях вычислительных устройств, таких как 4-х ядерный сервер с 32 ГБ оперативной памяти под управлением Linux, ноутбук под управлением Windows 10, а также мобильные устройства с ОС Android. Стоит отметить, что несмотря на разницу языков в новой (JavaScript) и существующей (C++) реализациях среды исполнения, количество итераций до сходимости метода в примерах совпало, а также совпало значение невязки на последнем шаге с точностью как минимум 6 знаков.

Результаты тестов свидетельствуют о функциональном состоянии реализации среды исполнения, достижении требуемого уровня переносимости, и подтверждают возможность использования замены модулей для управления исполнением параллельной программы в экспериментальных исследованиях алгоритмов динамической балансировки вычислительной нагрузки.

Заключение. В работе предлагается распределенная среда исполнения фрагментированных программ на языке LuNA, обладающая высокой степенью переносимости и предназначенная для проведения экспериментальных исследований системных алгоритмов управления поведением высокоуровневых параллельных программ. Высокая степень переносимости достигается за счет использования веб-технологий в качестве базовой платформы среды исполнения, что позволяет использовать среду исполнения на мультикомпьютерах с высокой степенью неоднородности аппаратного обеспечения, включая кластерный, серверный, персональный и мобильный классы устройств. Архитектура системы допускает замену системных алгоритмов распределения и перераспределения ресурсов, планирования вычислений, сборки мусора, маршрутизации коммуникаций и прочих функций. Проведено предварительное тестирование среды исполнения на ряде тестов, которое показало соответствие системы предъявляемым требованиям, в том числе по производительности.

Список литературы

1. CloudHaskell. [Электронный ресурс]: <http://haskell-distributed.github.io/> (дата обращения: 01.04.2019).
2. Carlton M., Van Roy P. A distributed Prolog system with AND-parallelism // Proceedings of the Twenty-First Annual Hawaii International Conference on System Sciences. Volume II: Software track. 1988.
3. Kale, Laxmikant V. and Bhatele, Abhinav. Parallel Science and Engineering Applications: The Charm++ Approach. Taylor & Francis Group, CRC Press, 2013. ISBN 978-1-4665-0412-7.
4. Bosilca, G., Bouteiller, A., Danalis, A., Faverge, M., Herault, T., Dongarra, J. PaRSEC: Exploiting Heterogeneity to Enhance Scalability // IEEE Computing in Science and Engineering. November, 2013. Vol. 15. N 6. P. 36–45.
5. X10 for High Performance Scientific Computing. Josh Milthorpe. Ph.D. Thesis, Research School of Computer Science, Australian National University, June 2015.
6. Treichler Sean, Bauer Michael, Sharma Rahul, Slaughter Elliott, and Aiken Alex. Dependent Partitioning // In Object Oriented Programming, Systems, Languages, and Applications (OOPSLA 2016).

7. Malyshkin Victor E., Perepelkin Vladislav A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // Parallel Computing Technologies. 11th International Conference, PaCT 2011, Proceedings. LNCS 6873. Springer, 2011. P. 53–61.

8. Malyshkin. V., Perepelkin. V., Schukin G. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // Journal of Supercomputing, S.I.: Parallel Computing Technologies — 2016. Springer, 2016. P. 1–7. DOI: 10.1007/s11227-016-1781-0.

9. Boost C++ Libraries. [Электронный ресурс]: <https://www.boost.org/> (дата обращения: 01.04.2019).

10. Message Passing Interface (MPI) Forum. [Электронный ресурс]: <https://www.mpi-forum.org/> (дата обращения: 01.04.2019).

11. Javascript performance benchmarks. [Электронный ресурс]: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/javascript.html> (дата обращения: 01.04.2019).



Азбиков Артем Альбертович — магистрант факультета информационных технологий, Новосибирский национальный исследовательский государственный университет, 630090, Новосибирск, Россия; e-mail:

azhbaev422@gmail.com.

Азбиков Артем Альбертович получил степень бакалавра в 2017 году в Новосибирском национальном исследовательском государственном университете на кафедре параллельных вычислений факультета информационных технологий, тема выпускной квалификационной работы — „Разработка и реализация переносимых алгоритмов распределенного исполнения фрагментированных программ“. Научные интересы — разработка системных алгоритмов распределенного исполнения параллельных программ.

Azhbaev Artem Albertovich received a bachelor's degree in Computer science in 2017 at Novosibirsk State University, 630090, Novosibirsk, Russia, e-mail: azhbaev422@gmail.com. Currently studying for the master's degree. Research interests include development of system algorithms of parallel program execution.



Перепелкин Владислав Александрович — научный сотрудник Института вычислительной математики и математической геофизики СО РАН; старший преподаватель каф. параллельных вычислений фа-

культета информационных технологий Новосибирского национального исследовательского государственного университета. Тел.: (383) 330-89-94, e-mail: perepelkin@ssd.sccc.ru.

Перепелкин Владислав Александрович в 2008 году окончил Новосибирский государственный университет с присуждением степени магистра техники и технологии по направлению „Информатика и вычислительная техника“. В настоящее время работает научным сотрудником в Институте вычислительной математики и математической геофизики СО РАН и старшим преподавателем в Новосибирском национальном исследовательском государственном университете, имеет более 20 опубликованных статей по теме автоматизации конструирования параллельных программ в области численного моделирования. Является одним из основных разработчиков экспериментальной системы автоматизации конструирования численных параллельных программ для мультикомпьютеров LuNA (от Language for Numerical Algorithms). Область профессиональных интересов включает автоматизацию конструирования параллельных программ, языки и системы параллельного программирования, высокопроизводительные вычисления.

Perepelkin Vladislav Aleksandrovich — graduated from Novosibirsk State University in 2008 with the master degree in engineering and technology in computer science. Nowadays works as a researcher in Institute of Computational Mathematics and Mathematical Geophysics (Siberian Branch of Russian Academy of Sciences), and also as a senior professor in Novosibirsk State University. Is an author of

more than 20 papers on automation of numerical parallel programs construction. Is one of main developers of fragmented programming system LuNA (Language for Numerical Algorithms). Professional interests include automation of numerical parallel programs construction, languages and systems of parallel programming, high performance computing.

Дата поступления – 13.05.2019