# LUNA SYSTEM FOR AUTOMATIC CONSTRUCTION OF NUMERICAL PARALLEL PROGRAMS FOR MULTICOMPUTERS

V. A. Perepelkin

Institute of Computational Mathematics and Mathematical Geophysics SB RAS,
630090, Novosibirsk, Russia

DOI: 10.24411/2073-0667-2020-10004

Usage of supercomputers for numerical simulations implies the complexity of distributed parallel programs development. Non-functional requirements for such programs include efficiency, memory and network bandwidth economy, tuning to available resources, fault tolerance and check pointing support, dynamic workload balancing, etc. To satisfy these requirements one has to concern peculiarities of application algorithm, hardware configuration and data. The development requires skills and knowledge supercomputer users are unlikely to have. To reduce the complexity of parallel programs development, debugging and modification diverse systems and tools of programs construction automation exist and evolve. Such tools accept a high level description of an application algorithm, as well as hardware configuration description, to produce and execute a parallel program, which implements the algorithm and fits into the non-functional requirements. Thus some of the programmer's burden is eliminated. Also such systems and tools are necessary to implement an active knowledge technology, where application algorithm is synthesized automatically based on application problem specification and has then to be automatically executed.

Automatic parallel programs construction system has to employ a high-level description of an application algorithm. The algorithm representation should be independent from hardware configuration and allow tuning of algorithm execution to given hardware and data. The representation should comprise independently computable parts which system should be able to dynamically map to given resources.

According to such demands the fragmented programming technology (FPT) is being developed in ICMMG SB RAS, as well as LuNA system for automatic numerical parallel programs construction (LuNA stands for Language for Numerical Algorithms). In the FPT computations are represented as the fragmented algorithm (FA) — a countable set of computational tasks called computational fragments (CFs). A CF is defined by two finite sets of input and output arguments — immutable data objects called data fragments (DFs) — and an operation on the DFs, which computes values of output DFs from values of input DFs. Execution of FA is the execution of all the CFs once their input DFs are computed. Operations are side-effect-free sequential subroutines, thus a CF can be executed on any computing node dynamically, provided its input DFs' values are transferred to the node. The job of the programming system is to dynamically map DFs and CFs to computing nodes, provide input DFs transfer to CFs and execute CFs to produce new DFs. During FA execution the system redistributes CFs and DFs in order to balance workload, employs replication to provide fault tolerance, saves DFs values for check pointing or performs other jobs to provide necessary non-functional properties of the program execution.

In order to improve non-functional properties of FA execution supplementary information called recommendations can be provided by the user. Recommendations are means to overcome the

fundamental difficulty of efficient execution of an application algorithm, represented in a high-level language, such as LuNA. Using recommendations the user can explicitly declare properties of the algorithm, which are hard to obtain automatically, but can significantly affect execution (e.g. estimated operations execution time). Also the user can employ recommendations to express his insight on how the computations should be performed in order to achieve high efficiency (instead of programming it in a low-level language, such as C+MPI). The system will follow the recommendations in cases, where good efficiency is hard to achieve automatically.

An experiment of automatic CFs mapping to resources using profiling is concerned. An application (matrices multiplication test) was run multiple times in the profiling mode. After each run the CFs mapping to computing nodes was adjusted to avoid load imbalances, which occurred in the previous execution. After few executions the execution time of the program decreased to the time of an efficient hand-made implementation of the same algorithm. This shows the possibility to automatically tune a FA execution to hardware configuration.

Another experiment shows automatic provision of dynamic load balancing on the example application of self-gravitating dust cloud simulation using Particle-in-Cell method. The application is an example of a problem, where workload distribution cannot be predicted statically and depends on input data. Necessary dynamic workload balancing was provided by LuNA system automatically based on the Rope of Beads algorithm, which preserves distributed data structure when balancing.

A number of other models and real-life applications were developed using the LuNA system. The efficiency of the constructed program is generally lower, than the one of hand-coded implementation, but reduction of labourness of program development, debugging and modification is often worth it. In future it is planned to implement other system algorithms and heuristics to improve the quality of automatically constructed programs, and to use LuNA system as a basis for active knowledge system construction.

**Key words:** Fragmented programming technology, automatic parallel programs construction, high performance computing, LuNA system.

## References

1. Victor Malyshkin. Active Knowledge, LuNA and Literacy for Oncoming Centuries // 2015. LNCS, Vol. 9465. P. 292–303.

2. Ken Kennedy, Charles Koelbel, and Hans Zima. The rise and fall of High Performance Fortran: an historical object lesson. In Proceedings of the third ACM SIGPLAN conference on History of programming languages (HOPL III). Association for Computing Machinery, New York, NY, USA, 2007. 7–1–7–22. DOI: 10.1145/1238844.1238851.

3. Inkrementalnoye rasparallelivanie dlya klasterov v sisteme SAPFOR / V. A. Bakhtin, O. F. Zhukova, N. A. Kataev et al. // Trudy Vseross. nauchn. konf. Nauchnyi servis v seti INTERNET. 2017.

4. Apache Hadoop. [Electron. Res.]: https://hadoop.apache.org/.

5. Wu W., Bouteiller A., Bosilca G., Faverge M., Dongarra J. Hierarchical DAG Scheduling for Hybrid Distributed Systems // 29th IEEE International Parallel & Distributed Processing Symposium, 2014.

6. Kale, Laxmikant V. and Bhatele, Abhinav. Parallel Science and Engineering Applications: The Charm++ Approach. Taylor & Francis Group, CRC Press. 2013. ISBN: 9781466504127.

7. Abramov S. M., Kuznetsov A. A., Roganov V. A. Krossplatformennaya versiya T-sistemy s otkrytoy arkhitekturoy // Vychislitelnye metody i programmirovaniye. 2007. Vol. 8. N 1. Razdel 2. P. 175–180.

8. Datta, Kaushik & Bonachea, Dan & Yelick, Katherine. Titanium Performance and Potential: An NPB Experimental Study. 4339. 2005. P. 200–214. DOI: 10.1007/978-3-540-69330-7_14.

9. Tarek El-Ghazawi and Lauren Smith. UPC: unified parallel C // In Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC '06). Association for Computing Machinery, New York, NY, USA, 27–es. 2006. DOI: https://doi.org/10.1145/1188455.1188483

10. Kemal Ebcioglu, Vijay Saraswat, Vivek Sarkar. X10: Programming for hierarchical parallelism and non-uniform data access. International Workshop on Language Runtimes, OOPSLA 2004.

11. Andrianov A. N., Efimkin K. N., Levashov V. Y., Shishkova I. N. The Norma language application to solution of strong nonequilibrium transfer process problem with condensation of mixtures on the multiprocessor system // Computational Science — ICCS 2001, Lecture Notes in Computer Science. May, 2001. V. 2073, P. 502–510.

12. Treichler, Sean & Bauer, Michael & Aiken, Alex. Realm: An event-based low-level runtime for distributed memory architectures // Parallel Architectures and Compilation Techniques — Conference Proceedings, PACT. 2014. DOI: 10.1145/2628071.2628084.

13. Malyshkin V. E., Perepelkin V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // In: Malyshkin V. (eds) Parallel Computing Technologies // PaCT 2011. Lecture Notes in Computer Science. 2011. V. 6873. Springer, Berlin, Heidelberg.

14. Malyshkin. V., Perepelkin. V., Schukin G. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // Journal of Supercomputing, S. I.: Parallel Computing Technologies. Springer, 2017. P. 1–7. DOI: 10.1007/s11227-016-1781-0.

15. Perepelkin V. A. Optimizatsiya ispolneniya fragmentirovannykh programm na osnove profilirovaniya // Shestaya Sibirskaya konverentsiya po parallelnym i vysokoproizvoditelnym vychisleniyam: Programma i tezisy dokladov. Tomsk: Izd-vo Tom. un-ta, 2011. P. 117–122.

16. Malyshkin. V., Perepelkin. V. The PIC Implementation in LuNA System of Fragmented Programming // The Journal of Supercomputing, Special Issue on Parallel Computing Technologies. Springer, 2014. P. 89–97. DOI: 10.1007/s11227-014-1216-8.

17. Akhmed-Zaki, D., Lebedev, D., Perepelkin, V. Implementation of a three dimensional three-phase fluid flow („oil–water–gas“) numerical model in LuNA fragmented programming system // Journal of Supercomputing. Springer, 2017. N 73(2). P. 624–630. DOI: 10.1007/s11227-016-1780-1.

18. Daribayev B., Perepelkin V., Lebedev D., Akhmed-Zaki D. Implementation of the Two-Dimensional Elliptic Equation Model in LuNA Fragmented Programming System // 2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT). 2018. P. 1–4.

19. Kireev S. E., Perepelkin V. A. Issledovaniye proizvoditelnosti realizatsii metoda IADE v sisteme fragmentirovannogo programmirovaniya LuNA // Parallelnyye vychislitelnye tekhnologii (PaVT'2016): trydy mezhdunarodnoy nauchnoy konferentsii (28 March — 1 April 2016, Arkhangelsk). Chelyabinsk: Izdatelskiy tsentr YuUrGU, 2016, p. 780.

20. Nikolay B., Perepelkin. V. Automated GPU Support in LuNA Fragmented Programming System // Parallel Computing Technologies. PaCT 2017. Lecture Notes in Computer Science. Springer, Cham, 2017. V. 10421. P. 272–277. DOI: 10.1007/978-3-319-62932-2_26.

# СИСТЕМА LUNA АВТОМАТИЧЕСКОГО КОНСТРУИРОВАНИЯ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ ЧИСЛЕННОГО МОДЕЛИРОВАНИЯ НА МУЛЬТИКОМПЬЮТЕРАХ

В. А. Перепелкин

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

УДК 004.4'242
DOI: 10.24411/2073-0667-2020-10004

Разработка параллельных программ численного моделирования на мультикомпьютерах является сложной задачей ввиду необходимости обеспечивать нефункциональные свойства программ (производительность, расход памяти, нагрузка на сеть и т. п.), а также динамическую балансировку нагрузки, отказоустойчивость и другие свойства. В работе рассматриваются технология фрагментированного программирования и поддерживающая ее система LuNA автоматического конструирования параллельных программ с заданными нефункциональными свойствами. Прикладной алгоритм представляется в виде множества информационно-зависимых задач, что позволяет параллельно исполнять их, динамически перераспределять их по узлам мультикомпьютера, обеспечивая динамическую балансировку нагрузки на узлы, а также реализовывать другие нефункциональные свойства программы автоматически. Рассматривается возможность автоматической настройки исполнения программы на конфигурацию вычислителя на основе профилирования.

**Ключевые слова:** технология фрагментированного программирования, автоматическое конструирование параллельных программ, высокопроизводительные вычисления, система LuNA.

**Введение.** Применение суперкомпьютеров для численного моделирования наталкивается на проблему разработки эффективной параллельной программы, реализующей требуемый прикладной алгоритм. Под эффективностью понимается соответствие нефункциональным требованиям, предъявляемым к параллельной программе — производительность, расход памяти, нагрузка на сеть, полнота и равномерность нагрузки на доступные ресурсы и т. п. В ряде случаев от программы требуется динамически балансировать нагрузку на вычислительные узлы, обеспечивать отказоустойчивость, сохранять контрольные точки и прочее. Дополнительные трудности возникают в случае, если вычислитель имеет вычислительные узлы различной конфигурации, а также, если распределение нагрузки по узлам зависит от входных данных задачи и изменяется по ходу вычислений. Разработка программы, удовлетворяющей этим требованиям, затруднительна и требует специальной квалификации, которой пользователи суперкомпьютеров обычно не обладают в достаточной степени.

В этой связи важную роль играют средства автоматизации конструирования параллельных программ, в которых прикладной алгоритм описывается специальным образом,

допускающим автоматическое конструирование параллельной программы, а сама программа конструируется и исполняется автоматически с учетом свойств прикладного алгоритма, вычислителя и входных данных задачи. Система автоматического конструирования берет на себя часть работ по конструированию программы, снимая их с пользователя. Эффективность конструируемой программы может быть как выше, так и ниже, чем у аналогичной программы, разработанной вручную. Последнее часто приемлемо за счет экономии сил пользователя по разработке, отладке и модификации программы, особенно при прототипировании. Также автоматическое конструирование и исполнение программ с заданными нефункциональными свойствами является необходимым компонентом для технологии активных знаний [1], где автоматически синтезируется алгоритм решения поставленной задачи по ее формальной постановке, после чего должна быть автоматически сконструирована программа, реализующая этот алгоритм.

В настоящее время существуют и активно развиваются различные подходы к автоматическому конструированию параллельных программ. В системах [2, 3] исследуется подход к конструированию параллельной программы на основе аннотированного последовательного кода, причем в ряде случаев аннотирование может выполняться автоматически. Преимуществом подхода является легкость получения параллельной программы, если имеется последовательный код, а недостатком — необходимость системе дополнительно решать задачу выявления параллелизма прикладного алгоритма по его последовательному представлению, а также выявлять множество переменных прикладного алгоритма после того, как оно было отображено на конкретные ресурсы памяти в последовательной программе. Обе этих задачи являются трудно решаемыми, что частично преодолевается аннотированием кода и применением эвристических методов. Хорошую эффективность удается обеспечивать при применении частных (не универсальных) моделей вычислений, в таких системах как [4, 5], но пользователь при этом ограничен кругом задач, которые могут быть выражены средствами частной модели. В системах с параллелизмом на уровне задач (напр., [6, 7]) прикладной алгоритм представляется в виде множества информационно связанных или обменивающихся сообщениями задач, что позволяет системе автоматически распределять и динамически перераспределять задачи по узлам, а также влиять на порядок выполнения задач, тем самым балансируя нагрузку, управляя распределением ресурсов. В [6] это также позволяет обеспечивать отказоустойчивость и сохранение контрольных точек. Недостатком подхода является труднорешаемость задачи обеспечения эффективного исполнения алгоритма, представленного таким образом, поэтому область применения систем ограничена классом прикладных алгоритмов, определяемых используемыми в системе эвристиками. Этот же недостаток имеют системы с общей распределенной памятью [8, 9, 10]. Одним из возможных подходов к расширению области применения системы программирования является введение средств управления исполнением прикладного алгоритма, когда, помимо собственно описания прикладного алгоритма, пользователь также предоставляет некоторую схему („управляющий слой“), описывающую на предметно-ориентированном языке, как следует исполнять прикладной алгоритм в тех или иных условиях. Этот подход применяется в системах [5, 12, 13]. При существенном изменении конфигурации вычислителя или свойств входных данных управляющий слой может изменяться для настройки программы на исполнение в новых условиях без необходимости повторной отладки описания алгоритма. Перспективность подхода обусловлена тем, что по мере развития эвристических методов эффективного исполнения прикладных алгоритмов управляющий слой может быть отброшен или сгенерирован авто-

матически, поэтому он не ограничивает переносимости и эффективности представленного алгоритма.

**1. Технология фрагментированного программирования.** Технология фрагментированного программирования (ТФП) разрабатывается в ИВМ и МГ СО РАН. В основе подхода лежит представление прикладного алгоритма в виде не более чем счетного множества вычислительных задач, называемых фрагментами вычислений (ФВ). Каждый ФВ — это триплет ?*in*, *out*, *op*?, где *in* и *out* — конечные множества входных и выходных объектов данных, называемых фрагментами данных (ФД), а *op* — операция, вычисляющая значения ФД *out* из значений ФД *in*. Такое представление будем называть фрагментированным алгоритмом (ФА). Процесс вычислений рассматривается как выполнение всех ФВ по мере готовности их входных ФД. Операции *op* реализуются последовательными процедурами без побочных эффектов, что позволяет выполнять ФВ на любом вычислительном узле по выбору системы, в том числе динамически перераспределять ФВ по узлам мультикомпьютера. ФД являются объектами единственного присваивания, которые система может дублировать, передавать по сети или сохранять на диск. Такие свойства ФВ и ФД позволяют автоматически распределять и динамически перераспределять ФВ и ФД по узлам мультикомпьютера, выбирать порядок выполнения ФВ (в рамках имеющихся информационных зависимостей), обеспечивать динамическую балансировку нагрузки на узлы, использовать репликацию для повышения доступности данных и обеспечения отказоустойчивости, сохранять контрольные точки исполнения, автоматически обеспечивать передачу сообщений по сети и синхронизацию параллельных процессов, обеспечивая требуемые нефункциональные свойства параллельной программы.

С целью расширения класса эффективно исполняемых алгоритмов в ТФП предусмотрена возможность влияния на ход исполнения программы путем определения „рекомендаций“ — дополнительной информации об алгоритме и способе его отображения на ресурсы мультикомпьютера во времени. Управление исполнением осуществляется по следующим основным аспектам: отображение ФВ и ФД на узлы мультипроцессора, порядок выполнения ФВ и сборка мусора. Рекомендации не влияют на вычисляемые значения, но влияют на нефункциональные свойства (эффективность) исполнения ФА.

Рекомендации могут использоваться для описания важных с точки зрения эффективности, но трудно автоматически выявляемых свойств прикладного алгоритма (например, оценочные времена выполнения ФВ или асимптотическая сложность отдельных частей алгоритма), а также для описания подсказок того, как следует исполнять ФА. Примером последнего может служить алгоритм Rope of Beads [14], используемый в ТФП для конструирования динамического отображения множеств ФВ и ФД на узлы мультикомпьютера, поддерживающего динамическую балансировку нагрузки на узлы с сохранением соседства распределенных структур данных ФА. Идея этого алгоритма состоит во введении рекомендаций об отображении множества ФВ и ФД на некоторый виртуальный вычислитель с сетевой топологией „тор“. При исполнении ФА виртуальный вычислитель динамически отображается („накладывается“) на реальный с учетом сетевой топологии и относительной производительностью вычислительных узлов последнего (чем более производителен узел мультикомпьютера, тем больше соседних виртуальных узлов на него назначается). При возникновении дисбаланса на недогруженный вычислительный узел передается часть виртуальных узлов (и, соответственно, назначенных на них ФД и ФВ), вследствие чего нагрузка выравнивается. При этом „перемешивания“ ФД и ФВ не происходит, т. к. их относительное распределение по виртуальным узлам не изменяется.

Использование ТФП позволяет при разработке параллельных программ ограничиться разработкой последовательного кода (последовательных процедур, реализующих операции) и описанием примерной схемы исполнения параллельной программы на предметно-ориентированном языке (а не программировать и отлаживать это в низкоуровневых терминах с использованием коммуникационной библиотеки).

**2. Система LuNA и приложения.** Для поддержки ТФП была разработана экспериментальная система LuNA (от Language for Numerical Algorithms) автоматического конструирования параллельных программ на мультикомпьютерах. Входящий в ее состав язык LuNA позволяет описывать ФА как множество ФВ и ФД, а также рекомендации. Операции реализуются процедурами на языке C++. Система обеспечивает трансляцию LuNA-программ во внутреннее представление, которое затем исполняется исполнительной системой на мультикомпьютере. На базе системы LuNA был проведен ряд экспериментов, демонстрирующих автоматическое конструирование параллельных программ с заданными нефункциональными свойствами.

Так, в [15] была показана возможность автоматической настройки исполнения программы на вычислитель с помощью профилирования. Суть эксперимента заключалась в том, что программа (умножение плотных матриц) с произвольным распределением ФВ по узлам мультикомпьютера исполнялась в режиме профилирования. После исполнения профиль анализировался на предмет наличия дисбалансов загрузки вычислительных узлов, и при обнаружении дисбаланса некоторое количество ФВ, которые исполнялись в соответствующие моменты времени, переназначались с перегруженных узлов на недогруженные. Затем программа опять запускалась в режиме профилирования, и распределение ФВ снова корректировалось. После нескольких таких циклов время выполнения программы снижалось до величины, равной времени выполнения аналогичной программы, разработанной вручную на основе известной систолической схемы. Таким образом, настройка распределения ФВ по узлам мультикомпьютера была построена автоматически.

В работе [16] были исследованы алгоритмы автоматической динамической балансировки ФВ и ФД по узлам мультикомпьютера на примере моделирования эволюции самогравитирующего протопланетного пылевого диска методом частиц-в-ячейках в трехмерной области. Это приложение является показательным примером того, как от изменения входных данных задачи изменяется распределение вычислительной нагрузки по узлам мультикомпьютера. В эксперименте показано, как с помощью алгоритма Rope of Beads выравнивается дисбаланс нагрузки с сохранением пространственной структуры данных и вычислений.

В работах [17–19] рассматриваются реализация конкретных приложений в системе LuNA и эффективность конструируемых системой LuNA программ в сравнении с теми же алгоритмами, реализованными вручную средствами коммуникационной библиотеки, реализующей стандарт MPI (Message Passing Interface). В [20] рассматривается автоматизация использования GPU в LuNA-программах. По этим экспериментам видно, что эффективность LuNA-программы обычно ниже, чем эффективность MPI-программ, что, тем не менее, часто приемлемо за счет простоты разработки, отладки и модификации LuNA-программы в сравнении с MPI-программой.

**Заключение.** Технология фрагментированного программирования и система LuNA представляют подход к автоматическому конструированию параллельных программ с заданными нефункциональными свойствами. Создан задел для проведения экспериментов с различными системными алгоритмами и эвристиками. Продемонстрирована возможность

автоматического обеспечения различных нефункциональных свойств конструируемых параллельных программ. В дальнейшем планируется расширение области практического применения системы за счет улучшения системных алгоритмов и эвристик, а также использование системы LuNA для поддержки технологий активных знаний.

## Список литературы

1. Malyshkin V. Active Knowledge, LuNA and Literacy for Oncoming Centuries // LNCS, 2015. Vol. 9465. P. 292–303.

2. Kennedy Ken, Koelbel Charles, and Zima Hans. The rise and fall of High Performance Fortran: an historical object lesson // In Proceedings of the third ACM SIGPLAN conference on History of programming languages (HOPL III). Association for Computing Machinery, New York, NY, USA, 2007. 7–1–7–22. DOI: 10.1145/1238844.1238851.

3. Инкрементальное распараллеливание для кластеров в системе САПФОР / В. А. Бахтин, О. Ф. Жукова, Н. А. Катаев и др. // Труды Всеросс. научн. конф. Научный сервис в сети ИНТЕРНЕТ. 2017.

4. Apache Hadoop [Электронный ресурс]: https://hadoop.apache.org/.

5. Wu W., Bouteiller A., Bosilca G., Faverge M., Dongarra J. Hierarchical DAG Scheduling for Hybrid Distributed Systems // 29th IEEE International Parallel & Distributed Processing Symposium, 2014.

6. Kale, Laxmikant V. and Bhatele, Abhinav. Parallel Science and Engineering Applications: The Charm++ Approach. Taylor & Francis Group, CRC Press. 2013. ISBN: 9781466504127.

7. Абрамов С. М., Кузнецов А. А., Роганов В. А. Кроссплатформенная версия Т-системы с открытой архитектурой // Вычислительные методы и программирование. 2007. Т. 8. № 1. Раздел 2. С. 175–180.

8. Datta, Kaushik & Bonachea, Dan & Yelick, Katherine. Titanium Performance and Potential: An NPB Experimental Study. 4339. 2005. P. 200–214. DOI: 10.1007/978-3-540-69330-7_14.

9. Tarek El-Ghazawi and Lauren Smith. UPC: unified parallel C // In Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC '06). Association for Computing Machinery, New York, NY, USA, 27–es. 2006. DOI: https://doi.org/10.1145/1188455.1188483

10. Kemal Ebcioglu, Vijay Saraswat, Vivek Sarkar. X10: Programming for hierarchical parallelism and non-uniform data access. International Workshop on Language Runtimes, OOPSLA 2004.

11. Andrianov A. N., Efimkin K. N., Levashov V. Y., Shishkova I. N. The Norma language application to solution of strong nonequilibrium transfer process problem with condensation of mixtures on the multiprocessor system // Computational Science — ICCS 2001, Lecture Notes in Computer Science. May, 2001. V. 2073, P. 502–510.

12. Treichler, Sean & Bauer, Michael & Aiken, Alex. Realm: An event-based low-level runtime for distributed memory architectures // Parallel Architectures and Compilation Techniques — Conference Proceedings, PACT. 2014. DOI: 10.1145/2628071.2628084.

13. Malyshkin V. E., Perepelkin V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // In: Malyshkin V. (eds) Parallel Computing Technologies. PaCT 2011. Lecture Notes in Computer Science, vol 6873. Springer, Berlin, Heidelberg, 2011.

14. Malyshkin. V., Perepelkin. V., Schukin G. Scalable Distributed Data Allocation in LuNA Fragmented Programming System // Journal of Supercomputing, S. I.: Parallel Computing Technologies. Springer, 2017. P. 1–7. DOI: 10.1007/s11227-016-1781-0.

15. Перепелкин В. А. Оптимизация исполнения фрагментированных программ на основе профилирования // Шестая Сибирская конференция по параллельным и высокопроизводительным вычислениям: Программа и тезисы докладов. Томск: Изд-во Том. ун-та, 2011. С. 117–122.

16. Malyshkin V. E., Perepelkin V. A. The PIC Implementation in LuNA System of Fragmented Programming // The Journal of Supercomputing, Special Issue on Parallel Computing Technologies. Springer, 2014. P. 89–97. DOI: 10.1007/s11227-014-1216-8.

17. Akhmed-Zaki, D., Lebedev, D., Perepelkin, V. Implementation of a three dimensional three-phase fluid flow („oil–water–gas“) numerical model in LuNA fragmented programming system // Journal of Supercomputing. 2017. N 73(2). P. 624–630. DOI: 10.1007/s11227-016-1780-1.

18. Daribayev B., Perepelkin V., Lebedev D., Akhmed-Zaki D. Implementation of the Two-Dimensional Elliptic Equation Model in LuNA Fragmented Programming System // 2018 IEEE 12th International Conference on Application of Information and Communication Technologies (AICT). 2018. P. 1–4.

19. Киреев С. Е., Перепелкин В. А. Исследование производительности реализации метода IADE в системе фрагментированного программирования LuNA // Параллельные вычислительные технологии (ПаВТ'2016): труды международной научной конференции (28 марта — 1 апреля 2016 г., г. Архангельск). Челябинск: Издательский центр ЮУрГУ, 2016. с. 780.

20. Nikolay B., Perepelkin. V. Automated GPU Support in LuNA Fragmented Programming System // Parallel Computing Technologies. PaCT 2017. Lecture Notes in Computer Science, Springer, Cham, 2017. V. 10421. P. 272–277. DOI: 10.1007/978-3-319-62932-2_26.

**Перепелкин Владислав Александрович** — научный сотрудник Института вычислительной математики и математической геофизики СО РАН; старший преподаватель каф. параллельных вычислений факультета информационных технологий Новосибирского национального исследовательского государственного университета. Тел.: (383) 330-89-94, e-mail: perepelkin@ssd.sscc.ru. В 2008 году окончил Новосибирский государственный университет с присуждением степени магистра техники и технологии по направлению „Информатика и вычислительная техника“. Имеет более 25 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 has the research position at the Institute of Computational Mathematics and Mathematical Geophysics (Siberian Branch of Russian Academy of Sciences), and also has a part time senior professor position in the Novosibirsk State University. He is an author of more than 25 papers on automation of numerical parallel programs construction. He 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 — 28.02.2020*