# DISTRIBUTED ALGORITHM FOR DATA ALLOCATION IN LUNA FRAGMENTED PROGRAMMING SYSTEM

V. E. Malyshkin*, V. A. Perepelkin*, G. A. Schukin**

Institute of Computational Mathematics and Mathematical Geophysics SB RAS,
630090, Novosibirsk, Russia
*Novosibirsk State University,
630090, Novosibirsk, Russia
**Novosibirsk State Technical University
630073, Novosibirsk, Russia

The paper presents a scalable distributed algorithm for static and dynamic data allocation in LuNA fragmented programming system. The proposed algorithm takes into account data structure of the application numerical model executed, enables static and dynamic load balancing and can be used with various network topologies. Implementation of numerical models on multicomputers with large number of computing nodes is a challenging problem in the domain of high-performance parallel computing. Effective resources allocation and balancing strategies are necessary to achieve good efficiency and scalability of parallel programs. LuNA is a fragmented programming system which is being developed in Supercomputer Software Department of Institute of Computational Mathematics and Mathematical Geophysics SB RAS. LuNA's main objective is automation of construction of parallel programs, which implement large-scale numerical models for large-scale multicomputers.

In LuNA system an application algorithm is represented as two sets: a set of immutable data pieces (data fragments, DF) and a set of side-effect free computational processes (computational fragments, CF). All these fragments can be distributed over computational nodes of a multicomputer. Each DF is produced by one CF and can be used as an input by other CFs; each CF is executed only once and requires values of all its input DFs to be gathered on a single node for its execution. Execution of a program is managed by LuNA's runtime system. The runtime system performs distribution and migration of DFs and CFs over nodes of a multicomputer and deliverance of input DFs to their corresponding CFs to provide execution of all CFs in the program.

There are several steps in processing of DFs and CFs. Each DF must initially be assigned to a node for its storage and be allocated on this node. Each CF must be assigned a node for its execution and also be allocated on the node. Efficiency of LuNA program execution (in terms of running time, memory consumption, communications amount, etc.) heavily depends on CFs and DFs distribution. To achieve suitable efficiency, during execution of a program fragments may be transferred between nodes in order to equalize workload. Such dynamic load balancing is also performed by the runtime system. The algorithm described in the paper is used as for initial data allocation, as well as for dynamic load balancing. Although its description is given in respect to data fragments only, in the same way it can also be used (and is used actually) for managing allocation of computational fragments.

First, a mapping of data fragments to a range of integer indices must be constructed. The mapping is an input for the algorithm described. The mapping is expected to map dependent („neighboring“) data fragments on the same or close indices. Construction of such a mapping is a complex task, which is out of the scope of the paper. Construction of a suitable mapping, a knowledge of algorithm structure is required, human help is expected. To create a mapping of a multi-dimensional data structures to the

one-dimensional index range space-filling curves (such as Hilbert space-filling curve) may be used to preserve data neighborhood. The mapping is a constant function during execution of a program, thus it can be computed on any node without communications.

Second, the given indices range is split into segments with number of segments equal to the number of computational nodes. Each node is given its own segment, neighboring nodes in a line topology receiving neighboring segments. Each data fragment will be allocated on the node containing index of its mapping. For that, assuming the data fragment was produced on some node, first its mapping is computed and then, if necessary, the fragment is transferred across neighboring nodes until its final node is found. Because indices are ordered and for each node its neighbors are known, direction of fragments' movement in the line of nodes can always be determined at any node. In such way data allocation requires only local communications between neighboring nodes.

For the sake of dynamic load balancing the mapping of segments of indices to nodes can be changed. First, for each index in a segment its workload is evaluated. Formulas for workload evaluation can be different, for example value of workload can be proportional to a volume of memory occupied by data on the node. Node's workload is computed as sum of its indices workloads. Second, workloads of neighboring nodes are compared periodically. If the workload difference exceeds a threshold, load re-balancing will be performed. For that neighboring nodes shift common border of their segments (from overloaded node to underloaded one), causing DFs mapped to migrate thus equalizing workload. Because order of indices is preserved during border shift, data locality is preserved too. Load balancing algorithm is distributed (executes on each node) and uses only local communications.

Usage of only local communications for data allocation and load balancing makes proposed algorithm scalable to a large number of computational nodes, because no global synchronization (that can impede scalability) is used.

Testing of the algorithm has shown its potential for scalability; performance of fragmented program for problem used was just several times worse than of usual MPI implementation for the same problem, given that MPI implementation with dynamic load balancing takes more time and effort to program instead of LuNA implementation, where data allocation and load balancing are implemented automatically.

**Key words:** scalable distributed system algorithm, dynamic data allocation, distributed algorithms with local interactions, fragmented programming technology, fragmented programming system LuNA.

## References

1. Malyshkin, V. E., Perepelkin, V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. In: PaCT 2011, LNCS, V. 6873, P. 53–61. Springer, Heidelberg, 2011.

2. Malyshkin, V. E., Perepelkin, V. A. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System // J. Supercomputing. 2012. V. 61, N 1, P. 235–248.

3. Malyshkin, V. E., Perepelkin V. A. The PIC Implementation in LuNA System of Fragmented Programming // J. Supercomputing. 2014. V. 69. N 1. P. 89–97.

4. Kraeva, M. A., Malyshkin, V. E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers // J. Future Generation Computer Systems. 2001. V. 17. N 6. P. 755–765.

5. Kireev, S. E., Malyshkin V. E. Fragmentation of Numerical Algorithms for Parallel Subroutines Library // J. Supercomputing. 2011. V. 57. N 2. P. 161–171.

6. Kraeva, M. A., Malyshkin, V. E. Algoritmy dinamicheskoi balansirovki zagruzki pri realizaczii metoda chasticz v yacheikakh na MIMD-multikomputerakh. J. Programmirovanie. 1999. N 1. P. 47–53.

7. Hu, Y. F., Blake, R. J. An Improved Diffusion Algorithm for Dynamic Load Balancing. J. Parallel Computing. 1999. V. 25. N 4. P. 417–444.

8. Corradi, A., Leonardi, L., Zambonelli F. Performance Comparison of Load Balancing Policies Based on a Diffusion Scheme / Euro-Par'97 Parallel Processing. 1997. P. 882–886. Springer, Heidelberg.

9. Anderson, J. M., Lam, M. S. Global Optimizations for Parallelism and Locality on Scalable Parallel Machines / ACM-SIGPLAN PLDI'93. 1993. P. 112–125. ACM New York, USA.

10. Li, J., Chen, M. The Data Alignment Phase in Compiling Programs for Distributed-Memory Machines // J. Parallel and Distributed Computing. 1991. V. 13. N 2. P. 213–221.

11. Lee P. Efficient Algorithms for Data Distribution on Distributed Memory Parallel Computers // J. IEEE Transactions on Parallel and Distributed Systems. 1997. V. 8. N 8. P. 825–839.

12. Yu-Kwong Kwok, Ahmad, I. Design and Evaluation of Data Allocation Algorithms for Distributed Multimedia Database Systems // IEEE Journal on Selected Areas in Communications. 1997. V. 14. N 7. P. 1332–1348.

13. Iacob, N. M. Fragmentation and Data Allocation in the Distributed Environments // Annals of the University of Craiova — Mathematics and Computer Science Series. 2011. V. 38. N 3. P. 76–83.

14. Jagannatha, S., Geetha, D. E., Suresh Kumar, T. V., Rajani Kanth, K. Load Balancing in Distributed Database System using Resource Allocation Approach // J. Advanced Research in Computer and Communication Engineering. 2013. V. 2. N 7. P. 2529–2535.

15. Honicky, R. J., Miller E. L. Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution / 18th International Parallel and Distributed Processing Symposium, 2004.

16. Alicherry, M., Lakshman, T. V. Network Aware Resource Allocation in Distributed Clouds / INFOCOM 2012. P. 963–971.

17. AuYoung, A., Chun, B. N., Snoeren, A. C., Vahdat, A. Resource Allocation in Federated Distributed Computing Infrastructures // First Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure, 2004.

18. Raman, R., Livny M., Solomon, M. Matchmaking: Distributed Resource Management for High Throughput Computing // J. Cluster Computing. 1999. V. 2. N 1. P. 129–138.

19. Reddy, C., Bondfhugula, U. Effective Automatic Computation Placement and Data Allocation for Parallelization of Regular Programs / 28th ACM International Conference on Supercomputing. 2014. P. 13–22. ACM New York, USA.

20. Baden, S. B., Shalit, D. Performance Tradeoffs in Multi-tier Formulation of a Finite Difference Method // ICCS 2001, LNCS. V. 2073. P. 785–794. Springer, Heidelberg.

21. Ken-ichiro Ishikawa. ASURA: Scalable and Uniform Data Distribution Algorithm for Storage Clusters. Computing Research Repository, abs/1309.7720, 2013.

22. Chawla, A., Reed B., Juhnke, K., Syed, G. Semantics of Caching with SPOCA: A Stateless, Proportional, Optimally-Consistent Addressing Algorithm // USENIX Annual Technical Conference 2011. P. 33–33. USENIX Association.

23. Lowder, J. K., King, P. J. H. Using Space-Filling Curves for Multi-dimensional Indexing // Advances in Databases. LNCS. V. 1832. P. 20–35. Springer, Heidelberg, 2000.

24. Moon, B., Jagadish, H. V., Faloutsos, C., Saltz, J. H. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve // J. IEEE Transactions on Knowledge and Data Engineering. 2000. V. 13. N 1. P. 124–141.

# РАСПРЕДЕЛЕННЫЙ АЛГОРИТМ УПРАВЛЕНИЯ ДАННЫМИ В СИСТЕМЕ ФРАГМЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ LUNA

В. Э. Малышкин [*], В. А. Перепелкин [*], Г. А. Щукин [**]

Институт вычислительной математики
и математической геофизики СО РАН,
630090, Новосибирск, Россия
[*] Новосибирский государственный университет,
630090, Новосибирск, Россия
[**] Новосибирский государственный технический университет,
630073, Новосибирск, Россия

УДК 004.021

В статье представлен распределенный алгоритм статического и динамического распределения данных в системе фрагментированного программирования LuNA. Система LuNA предназначена для автоматизации конструирования параллельных программ, реализующих крупномасштабные численные модели на мультикомпьютерах с большим числом процессоров. Алгоритм является масштабируемым по числу процессоров, учитывает структуру данных численной модели, обеспечивает статическую и динамическую балансировку нагрузки и может быть использован для различных топологий вычислительной сети.

**Ключевые слова:** распределенный алгоритм, распределение данных, динамическая балансировка нагрузки, технология фрагментированного программирования, система фрагментированного программирования LuNA.

**Введение.** Реализация больших численных моделей для мультикомпьютеров является сложной задачей в области высокопроизводительных параллельных вычислений. Для достижения высокой эффективности и масштабируемости параллельных программ требуются эффективная стратегия распределения ресурсов и/или динамическая балансировка нагрузки. Ввиду роста размера мультикомпьютеров (числа узлов, объема памяти и т. д.) требуется создание новых системных алгоритмов для обработки данных и организации вычислений. Для упрощения конструирования параллельных программ, способных показать хорошую производительность, была разработана система фрагментированного программирования LuNA [1, 2, 3].

В системе LuNA фрагментированная программа представляется в виде двух множеств: множества иммутабельных элементов данных (фрагменты данных, ФД) и множества вычислительных процессов без побочных эффектов (фрагменты вычислений, ФВ). Каждый фрагмент данных вырабатывается каким-либо фрагментом вычислений и может быть подан на вход другим фрагментам вычислений. Каждому фрагменту вычислений нужны

Работа поддержана грантами РФФИ 14-07-00381 а и 14-01-31328 mol_a.

значения всех его входных фрагментов данных для начала исполнения, причем все фрагменты вычислений исполняются только один раз.

Фрагментированная программа в системе LuNA выполняется под управлением исполнительной системы, которая осуществляет перемещения фрагментов данных и вычислений между процессорными элементами (ПЭ) — вычислительными узлами мультикомпьютера) — для распределения фрагментов и доставки входных фрагментов данных фрагментам вычислений, а также выравнивания нагрузки. Наличие исполнительной системы позволяет исполнять одну и ту же фрагментированную программу на разных вычислительных системах без дополнительных модификаций этой программы.

Эффективность исполнения LuNA-программы на мультикомпьютере (в терминах времени выполнения, потребления памяти и т. д.) зависит от качества распределения фрагментов данных и вычислений по ПЭ. В статье авторами предлагается масштабируемый распределенный алгоритм динамического распределения данных, используемый в системе LuNA.

**1. Обзор**. Задача эффективного и масштабируемого распределения данных активно исследовалась и исследуется. Одним из подходов к распределению данных является использование масштабируемых диффузионных алгоритмов ([6, 7, 8]). Такие алгоритмы не требуют глобальных взаимодействий, но не учитывают структуру данных задачи и допускают глобальный дисбаланс.

Много общего имеют задачи распределения данных в распределенных базах данных ([12, 13, 14, 15]) и облачных сервисах ([16, 17, 18]). Из-за относительного малого числа объектов для распределения и низкой частоты их миграции эти системы используют централизованные алгоритмы. Минусом таких алгоритмов является то, что они не масштабируются ни на большое число фрагментов данных и вычислений, ни на мультикомпьютеры с большим числом ПЭ.

Хорошая эффективность может быть достигнута при распределении данных определенной структуры, например, сеток ([19, 20]). К сожалению, область применения таких алгоритмов довольно мала.

Стоит упомянуть алгоритмы статического анализа, используемые в компиляторах ([9, 10, 11]). Их ограничением является статическое принятие решений (во время компиляции), т. е. они не могут быть использованы для случая, когда решение о распределении данных должно быть принято во время исполнения программы.

Алгоритмы из [21, 22] не учитывают структуру данных задачи и не решают проблему поиска данных. В [15] представлен относительно масштабируемый алгоритм распределения данных, но использующий глобальные коммуникации (т. е. коммуникации между узлами, расположенными неограниченно далеко друг от друга в смысле сетевой топологии).

**2. Требования к алгоритму распределения данных**. Чтобы обеспечить высокую эффективность и масштабируемость параллельной программы, алгоритм распределения данных должен:

— обеспечивать равномерную загрузку ПЭ (статическая и динамическая балансировка нагрузки) в терминах объема данных и/или вычислений;

— минимизировать объем коммуникаций между ПЭ с помощью учета структуры данных задачи;

— подстраиваться под поведение моделируемого явления;

— быть децентрализованным и использовать преимущественно локальные коммуникации.

**3. Распределенный алгоритм распределения данных.**

3.1. *Особенности распределения данных в системе LuNA.* В системе LuNA каждому фрагменту данных и вычислений назначается единственный ПЭ, называемый *резиденцией* этого фрагмента. Резиденция ФД–ПЭ, на котором он хранится, резиденция ФВ–ПЭ, на котором он будет исполнен. При порождении фрагмента данных или вычислений исполнительная система LuNA доставляет его на его резиденцию. Резиденция фрагмента может быть изменена в процессе исполнения программы, что приводит к перемещению этого фрагмента со старого ПЭ на новый.

Одна из задач исполнительной системы LuNA заключается в доставке значений входных фрагментов данных требующим их фрагментам вычислений на нужные ПЭ (резиденции ФВ). Для каждого ФВ известны (или могут быть вычислены) идентификаторы его входных ФД, причем идентификаторы всех ФД уникальны в пределах исполняемой программы. Исполнительная система LuNA предоставляет средства для получения резиденции ФД по его идентификатору. Когда резиденция ФД получена, на этот ПЭ может быть отправлен запрос на пересылку значения требуемого ФД.

Динамическая балансировка нагрузки на ПЭ в системе LuNA заключается в смене резиденций фрагментов с целью выравнивания нагрузки ПЭ. Смена резиденций приводит к миграции ФД и ФВ, в результате чего нагрузка может быть выровнена.

3.2. *Формулировка задачи распределения данных.* Задача распределения данных формулируется следующим образом. Пусть $D$ — множество всех фрагментов данных в программе, и $P$ — множество всех доступных ПЭ мультикомпьютера. Требуется построить функцию $r : (D, T) \to P$, которая динамически назначает каждому фрагменту данных $df \in D$ его резиденцию $p \in P$ в произвольный момент времени $t \in T$ ($T$ — множество всех моментов времени исполнения программы; можно считать $T = [0; 1]$, где 0 — начало исполнения программы, а 1 — конец).

Для примера, пусть в момент времени $t = 0$ существует начальное распределение фрагментов данных $r_1 : D \to P$. В момент времени $t' \in (0; 1)$ была произведена балансировка нагрузки, вследствие чего было получено новое распределение фрагментов данных $r_2 : D \to P$, которое не изменялось до самого конца исполнения программы. Результирующая функция $r$ тогда будет выглядеть как:

$$r(df, t) = \begin{cases} r_1(df), t < t' \\ r_2(df), t \geqslant t' \end{cases}$$

3.3. *Предлагаемый алгоритм распределения данных.* Так как автоматическое построение функции $r$, которое бы обеспечивало хорошее распределение данных, затруднительно, то целесообразно участие человека. Для этого построение $r$ разбивается на два этапа. На первом этапе пользователь/программист самостоятельно определяет вспомогательную функцию $v : D \to V$, где $V$ — конечное множество узлов виртуального мультикомпьютера (виртуальные узлы). Виртуальный мультикомпьютер имеет одномерную топологию (линейная или кольцо), и все его узлы гомогенны. Функция $v$ назначает каждому фрагменту данных постоянную виртуальную резиденцию (виртуальный узел). На втором этапе исполнительной системой LuNA автоматически определяется функция $p : (V,T) \to P$,

которая отображает множество виртуальных узлов на множество реальных ПЭ. В итоге резиденция фрагмента данных $df$ в момент времени $t$ вычисляется как $r(df, t) = p(v(df), t)$.

При построении функции $p$ исполнительной системой учитывается реальная топология вычислительной системы, т. е. два соседних в линейной топологии виртуальных узла назначаются на один и тот же или соседние узлы в реальной топологии. Такое отображение может быть реализовано для таких топологий, как N-мерный тор или решетка, толстое дерево и т. д. Если на каждый виртуальный узел приходится примерно одинаковая нагрузка, размещение на каждом ПЭ одинакового числа виртуальных узлов позволяет получить сбалансированное распределение.

Для построения функции $v$ пользователь должен решить упрощенную задачу распределения данных. Упрощение вытекает из статичности распределения данных по виртуальным узлам, гомогенности виртуальных узлов и линейности топологии виртуального мультикомпьютера. Построение функции $v$ пользователем позволяет ему передать свои знания об эффективном распределении данных исполнительной системе.

Для того, чтобы функция $r$ удовлетворяла требованиям из секции 3, функция $v$ должна:

— минимизировать нарушения отношения соседства, т. е. разницу между резиденциями ФВ и его входных ФД в линейке виртуальных узлов;

— минимизировать дисбаланс нагрузки, т. е. разницу между объемом данных или вычислений среди всех виртуальных узлов.

Представленный список требований неполон, но может быть использован как для оценки качества имеющейся функции $v$, так и для построения новой функции $v$. В частности, важно минимизировать не только общий дисбаланс нагрузки, но и дисбаланс нагрузки в каждый момент времени. Для примера рассмотрим одномерную явную конечно-разностную схему. Множество фрагментов данных: $D = \{df_{i,t}\}$, где $i$ — координата по пространству, а $t$ — по времени. Обе функции $v(df_{i,t}) = i$ и $v'(df_{i,t}) = t$ обеспечивают одинаковую нагрузку для всех виртуальных узлов, но в первом случае вычисления могут выполняться параллельно, тогда как во втором случае возможно только последовательное исполнение из-за зависимостей по данным (рис. 1).

3.4. *Динамическая балансировка нагрузки.* Предложенный алгоритм распределения данных поддерживает динамическую балансировку нагрузки. Распределенный алгоритм балансировки нагрузки основан на известном диффузионном подходе. Предполагается, что нагрузка каждого ПЭ периодически измеряется или оценивается каким-либо способом, и полученная оценка рассылается всем его соседним ПЭ. Когда разница нагрузок каких-либо двух соседних ПЭ превосходит заданный порог, вызывается процедура балансировки, которая перемещает часть нагрузки с перегруженного узла на недогруженный.

В предлагаемом алгоритме перемещение нагрузки производится через изменение отображения части виртуальных узлов с одних ПЭ на другие. Смена отображения приводит к переназначению резиденций ФД и ФВ для этих виртуальных узлов и, в свою очередь, миграции этих фрагментов на новый ПЭ. При балансировке должно сохраняться условие, что соседние виртуальные узлы располагаются на одном или соседних ПЭ. Это, очевидно, всегда возможно. Благодаря этому соседство ФД и ФВ сохраняется в процессе балансировок и „перемешивания“ данных, характерного для классической балансировки нагрузки диффузионного типа, не происходит независимо от числа балансировок.

3.5. *Пример распределения и балансировки данных для двумерной сетки.* На рис. 2 представлена сетка, декомпозированная на $8 \times 8$ фрагментов данных по двум координа-

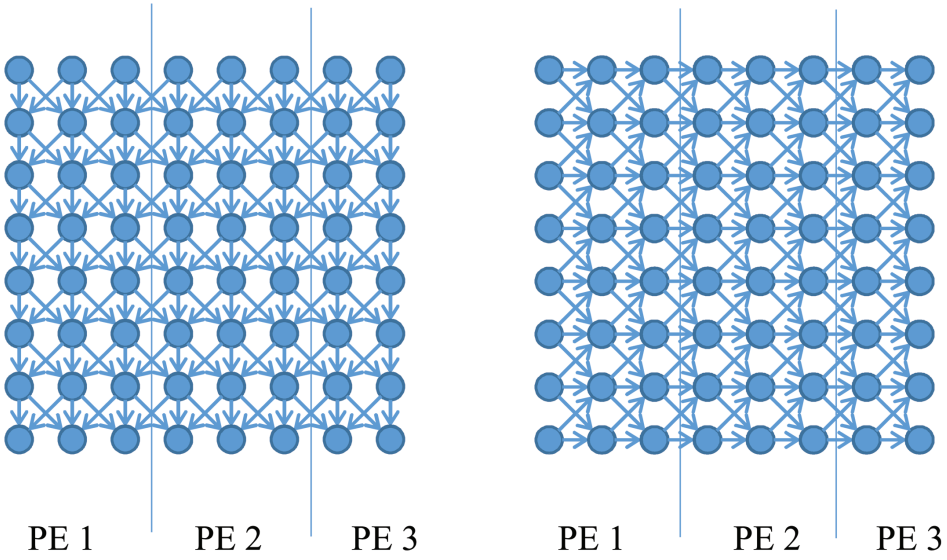


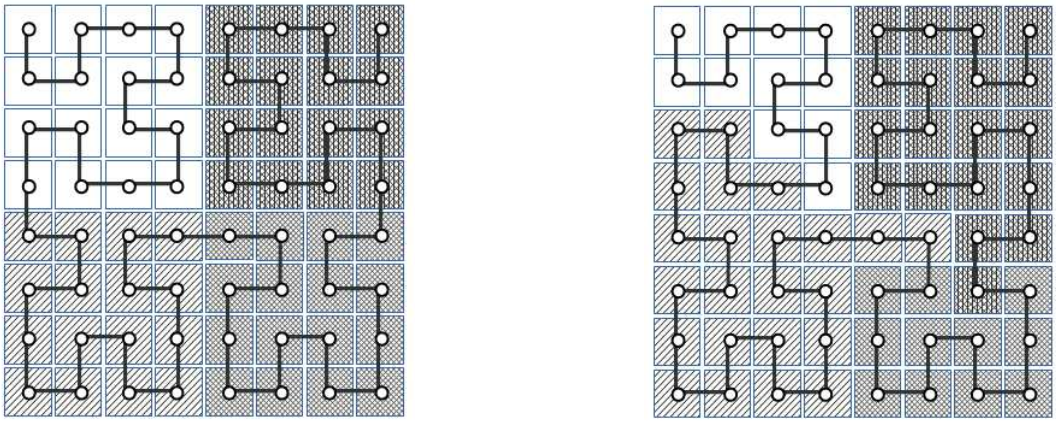

Рис. 1. Два различных распределения фрагментов данных для явной КРС

там. Каждый фрагмент данных содержит в себе блок ячеек сетки. Если сетка используется для решения уравнения Пуассона с помощью пятиточечной конечно-разностной схемы, соседями каждого фрагмента данных являются прилегающие фрагменты данных по вертикали и горизонтали.

Для распределения фрагментов данных по ПЭ сначала провоизводится их отображение на линейку виртуальных узлов. В данном случае для отображения используется пространственная кривая Гильберта: двумерная координата фрагмента данных в сетке преобразуется в одномерный индекс на кривой, которому соответствует один из виртуальных узлов (рис. 2, а). Получившееся распределение сохраняет отношение соседства для половины фрагментов данных (из четырех возможных соседей в двумерной сетке два всегда находятся рядом на кривой), для оставшейся половины их соседи находятся, как правило, рядом в той или иной окрестности на кривой (свойство кривой Гильберта).

Затем производится отображение линейки виртуальных узлов на реальные ПЭ, в данном случае 4 ПЭ (рис. 2, а, доставшиеся каждому ПЭ фрагменты данных показаны разными цветами). Начальное отображение может быть осуществлено с расчетом одинакового числа виртуальных узлов на ПЭ. Для этого множество индексов виртуальных узлов (64 индекса в данном случае) делится на сегменты по равному числу индексов в каждом (16 индексов), смежные ПЭ в линейке узлов получают смежные сегменты индексов и соответствующие этим индексам виртуальные узлы.

Имея на каждом ПЭ информацию о принадлежащем ему сегменте индексов виртуальных узлов, поиск фрагмента данных выполняется следующим образом: на текущем ПЭ по двумерной координате фрагмента вычисляется индекс его виртуального узла; если индекс находится внутри сегмента индексов текущего ПЭ, искомый ФД находится на этом ПЭ; иначе можно однозначно определить направление (соседний ПЭ) в линейке ПЭ, куда должен быть передан запрос на поиск ФД, после чего алгоритм повторяется на соседнем ПЭ. Аналогичным образом осуществляется доставка созданного ФД на его резиденцию.



а) до балансировки б) после балансировки

Рис. 2. Распределение фрагментов данных с помощью кривой Гильберта

В случае дисбаланса нагрузки на ПЭ инициируется процедура балансировки, как описано выше. Получившееся в итоге распределение может выглядеть как на рис. 2,б.

3.6. *Особенности предложенного алгоритма.* Так как динамическая балансировка нагрузки в алгоритме реализована через смену резиденций, единицей балансировки является группа фрагментов данных и вычислений, принадлежащая одному виртуальному узлу. Из этого следует, что число виртуальных узлов должно быть в несколько раз больше числа ПЭ для обеспечения достаточно мелкой гранулярности балансировки.

К плюсам алгоритма можно отнести:

— распределенность и использование исключительно локальных коммуникаций (речь идет о коммуникациях системных алгоритмов; если дальние коммуникации присущи самой задаче, то они, естественно, будут присутствовать независимо от алгоритма распределения данных);

— возможность подстройки под различные топологии вычислительной сети, т. к. линейная виртуальная топология может быть легко вложена во многие другие топологии;

— динамическая балансировка нагрузки;

— учет отношения соседства данных и вычислений;

— константное время на распределение и поиск фрагмента данных в лучшем случае (0 или 2 сообщения), т. е. при условии, что производитель и потребитель ФД отображены на один или соседние виртуальные узлы.

К минусам алгоритма можно отнести:

— пользователь/программист ограничен линейной топологией для выражения знания о желательном способе распределения ресурсов, что позволяет лишь частично отразить более сложно соседство ФД и ФВ (пример с Гильбертовой кривой показывает принцип сведения реального соседства задачи к линейному);

— линейная зависимость времени от числа узлов для распределения и поиска фрагментов данных в худшем случае (т. е. когда ФД и ФВ отображены на виртуальные узлы без учета их соседства).

Для преодоления недостатков алгоритм можно усовершенствовать путем использования других топологий (двух- или трехмерных) для сети виртуальных узлов, лучше учитывающих особенности конкретной вычислительной задачи. Линейное время на распределе-

ние и поиск фрагментов данных можно избежать как использованием лучшего отображения на виртуальные узлы, так и применением кэширования границ несмежных сегментов индексов виртуальных узлов на несоседних ПЭ для сокращения числа коммуникаций. Кэширование подразумевает, что возможности физической топологии недоиспользованы, (например, в сегменте кластера фактически может быть топология „полный граф“), и кэширование позволит использовать прямые коммуникации.

**4. Тесты**. Для исследования эффективности предложенного алгоритма была использована фрагментированная реализация явной конечно-разностной схемы (КРС) для решения уравнения Пуассона [5] в трехмерном пространстве с начальными краевыми условиями первого рода (уравнение 1).

$$\begin{aligned} \nabla^2\phi &= f, \\ \phi\Big|_G &= F \end{aligned} \tag{1}$$

Трехмерная регулярная сетка была разбита на фрагменты данных по двум координатам, каждый фрагмент данных содержал в себе блок ячеек. Значения в ячейках сетки вычислялись в ходе итерационного процесса по формуле 2:

$$\phi_{ijk}^{m+1} = \frac{\frac{\phi_{i+1jk}^m+\phi_{i-1jk}^m}{h_x^2} + \frac{\phi_{ij+1k}^m+\phi_{ij-1k}^m}{h_y^2} + \frac{\phi_{ijk+1}^m+\phi_{ijk-1}^m}{h_z^2} - f_{ijk}}{\frac{2}{h_x^2}+\frac{2}{h_y^2}+\frac{2}{h_z^2}}, \tag{2}$$

начиная с некоторого начального приближения $\phi_0$ (здесь в формуле $h$ — шаг между ячейками сетки по соответствующей координате, $m$ — номер итерации, $ijk$ — глобальный индекс ячейки в сетке). Так как значение функции в ячейке сетки на следующей итерации зависит от значений функции в соседних ячейках с предыдущей итерации, в каждый фрагмент данных были добавлены т. н. теневые грани, которые хранили копии значений из соседних ячеек в соседних фрагментах. Перед каждой итерацией значения в теневых гранях обновлялись путем обмена значениями границ между смежными фрагментами (по двум координатам), после чего расчет значения функции для каждого фрагмента данных мог проводиться параллельно.

Эксперименты проводились на кластере МВС-100К Объединенного Суперкомпьютерного Центра РАН (по два процессора Intel Xeon E5450 и 8 Гб оперативной памяти на узел кластера; коммуникационная сеть Infiniband DDR). Использовались компилятор GCC 5.2.0 и коммуникационная библиотека MPICH 3.1.4.

Сравнивались две версии системы LuNA: *hilbert* и *cartesian*, которые различались в способе отображения данных задачи (сетки фрагментов) на виртуальные узлы. Первая использовала пространственную кривую Гильберта, вторая — декартовое построчное отображение. Для сравнения с обеими версиями в качестве эталона использовалась обычная MPI-реализация этой же задачи.

Начальный тест состоял из 100 итераций КРС на сетке из $400 \times 400 \times 400$ ячеек, разбитой на $8 \times 8$ фрагментов данных (рис. 3). Тест показал отставание LuNA от MPI от 3 раз в лучшем случае до 70 раз в худшем и 14 раз в среднем случае. Замедление LuNA, начиная с 32 процессов, можно объяснить возрастанием доли накладных расходов на коммуникации по сравнению с вычислениями. Также следует отметить, что ввиду иммутабельности фрагментов данных в LuNA вычисление фрагментов данных на каждой итерации приводило к порождению новых фрагментов данных и, соответственно, новому выделению

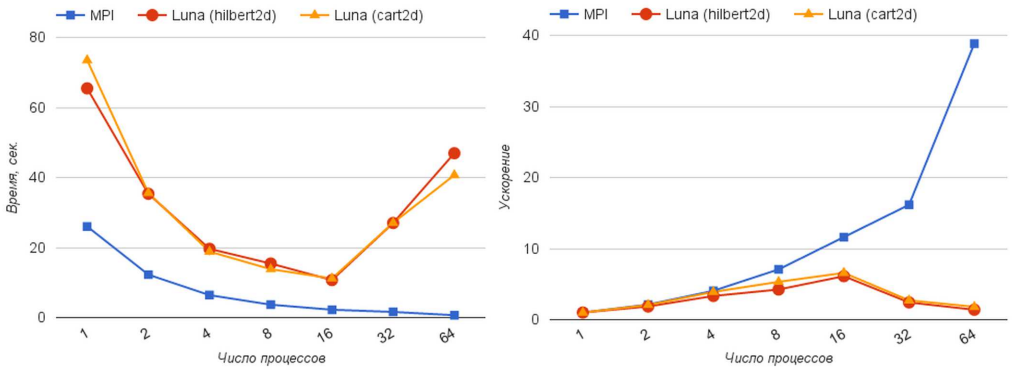



Рис. 3. Зависимость времени выполнения и ускорения от числа ПЭ (сетка $400 \times 400 \times 400$, 100 итераций)

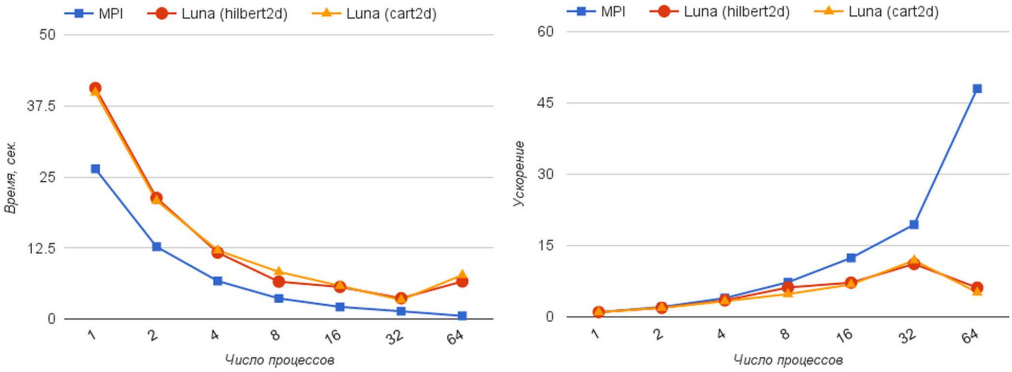



Рис. 4. Зависимость времени выполнения и ускорения от числа ПЭ (сетка $400 \times 400 \times 400$, 10 итераций, 10 повторов вычислений)

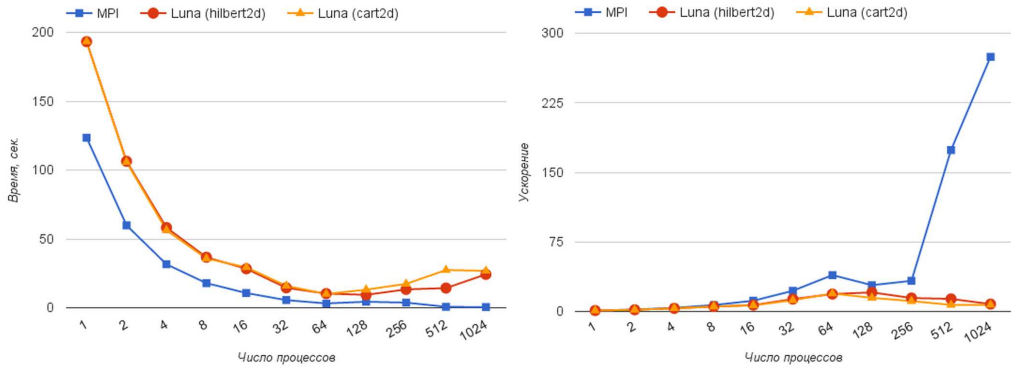



Рис. 5. Зависимость времени выполнения и ускорения от числа ПЭ (сетка $400 \times 400 \times 400$, 10 итераций, 50 повторов вычислений)

памяти (ненужные фрагменты данных при этом могли быть освобождены автоматическим сборщиком мусора), в то время как MPI-реализация переиспользовала уже выделенную память, что привело к дополнительным накладным расходам для LuNA. Вопрос оптимизации использования памяти для LuNA-программ является темой отдельного исследования и не рассматривается в этой статье.

Следующий тест (рис. 4) с уменьшением числа итераций до 10 и пропорциональным увеличением объема вычислений (путем их повтора) демонстрирует улучшение производительности LuNA-программы — разрыв с MPI-программой сокращается до 1,5, 14 и 3,8 раз в лучшем, худшем, и среднем случаях соответственно, а падение ускорения начинается

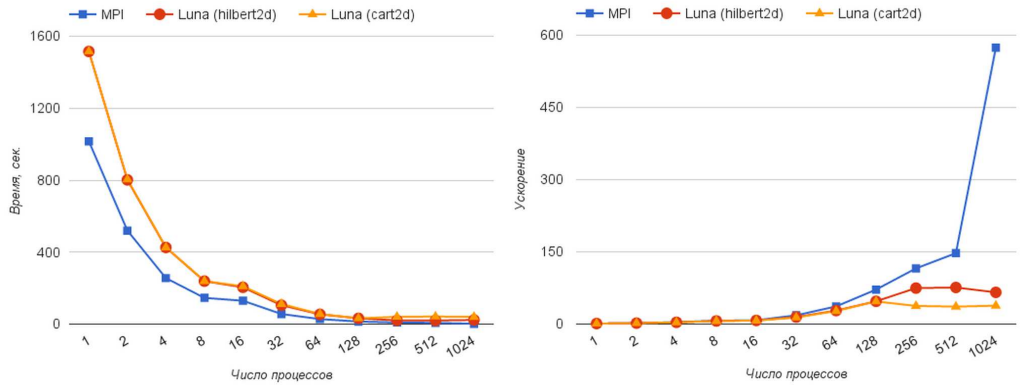



Рис. 6. Зависимость времени выполнения и ускорения от числа ПЭ (сетка $800 \times 800 \times 800$, 10 итераций, 50 повторов вычислений)

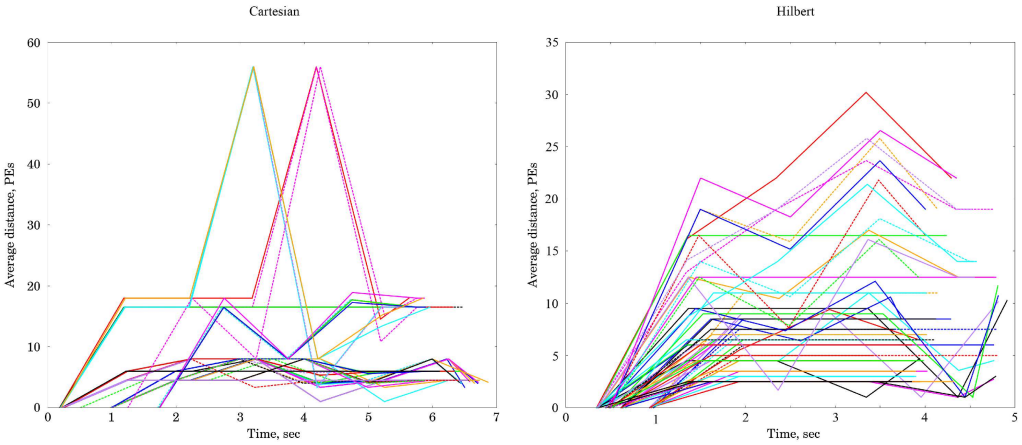



Рис. 7. Среднее расстояние перелета ФД. Разные цвета обозначают разные ПЭ

уже только с 64 процессов. Также следует отметить небольшое преимущество гильбертовой версии LuNA над декартовой.

Увеличение числа повторов, т. е. объема вычислений (а также фрагментации сетки до $32 \times 32$ фрагментов) показывает сохранение этой тенденции: отставание остается в 1,5 раза в лучшем случае, 2,25 в среднем до 64 процессов включительно, но поднимается до 8,7 и 11 раз в среднем для гильбертовой и декартовой версий соответственно до 1024 процессов (рис. 5), падение ускорения начинается только на 512 процессах.

При увеличении размера сетки до $800 \times 800 \times 800$ (рис. 6) отставание уменьшается до 2,9 и 4,3 раз в среднем для гильбертовой и декартовой версии соответственно, начиная с 256 процессов, заметно двукратное превосходство гильбертовой версии LuNA над декартовой. Также эффективность распараллеливания в лучшем случае выросла в 3,5 раза для MPI, 8,5 раз для гильбертовой версии LuNA и в 5,5 раз для декартовой.

На рис. 7 показано среднее расстояние (в ПЭ в линейке узлов) коммуникаций ФД для каждого ПЭ (разные цвета обозначают разные ПЭ). Расстояния для гильбертовой версии LuNA примерно в два раза меньше, чем для декартовой.

**Заключение**. В статье рассмотрены проблемы распределения данных для реализации больших численных моделей для суперкомпьютеров. Предложен алгоритм динамического распределения данных для системы фрагментированного программирования LuNA и

представлены его тесты производительности. Предложенный алгоритм предназначен для использования на больших мультикомпьютерах (тысячи и больше узлов), масштабируем, обеспечивает динамическую балансировку нагрузки на процессоры и учитывает структуру данных задачи. Дальнейшие исследования планируется посвятить преодолению недостатков алгоритма: частичному нарушению отношения соседства при отображении многомерных структур данных на одномерную виртуальную топологию и статичности этого отображения.

## Список литературы

1. Malyshkin, V. E., Perepelkin, V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. In: PaCT 2011, LNCS, V. 6873, P. 53–61. Springer, Heidelberg, 2011.

2. Malyshkin, V. E., Perepelkin, V. A. Optimization Methods of Parallel Execution of Numerical Programs in the LuNA Fragmented Programming System // J. Supercomputing. 2012. V. 61, N 1, P. 235–248.

3. Malyshkin, V. E., Perepelkin V. A. The PIC Implementation in LuNA System of Fragmented Programming // J. Supercomputing. 2014. V. 69. N 1. P. 89–97.

4. Kraeva, M. A., Malyshkin, V. E. Assembly Technology for Parallel Realization of Numerical Models on MIMD-Multicomputers // J. Future Generation Computer Systems. 2001. V. 17. N 6. P. 755–765.

5. Kireev, S. E., Malyshkin V. E. Fragmentation of Numerical Algorithms for Parallel Subroutines Library // J. Supercomputing. 2011. V. 57. N 2. P. 161–171.

6. Краева М. А., Малышкин В. Э. Алгоритмы динамической балансировки загрузки при реализации метода частиц в ячейках на МИМД-мультикомпьютерах // Ж. Программирование. 1999. № 1. С. 47–53.

7. Hu, Y. F., Blake, R. J. An Improved Diffusion Algorithm for Dynamic Load Balancing. J. Parallel Computing. 1999. V. 25. N 4. P. 417–444.

8. Corradi, A., Leonardi, L., Zambonelli F. Performance Comparison of Load Balancing Policies Based on a Diffusion Scheme / Euro-Par'97 Parallel Processing. 1997. P. 882–886. Springer, Heidelberg.

9. Anderson, J. M., Lam, M. S. Global Optimizations for Parallelism and Locality on Scalable Parallel Machines / ACM-SIGPLAN PLDI'93. 1993. P. 112–125. ACM New York, USA.

10. Li, J., Chen, M. The Data Alignment Phase in Compiling Programs for Distributed-Memory Machines // J. Parallel and Distributed Computing. 1991. V. 13. N 2. P. 213–221.

11. Lee P. Efficient Algorithms for Data Distribution on Distributed Memory Parallel Computers // J. IEEE Transactions on Parallel and Distributed Systems. 1997. V. 8. N 8. P. 825–839.

12. Yu-Kwong Kwok, Ahmad, I. Design and Evaluation of Data Allocation Algorithms for Distributed Multimedia Database Systems // IEEE Journal on Selected Areas in Communications. 1997. V. 14. N 7. P. 1332–1348.

13. Iacob, N. M. Fragmentation and Data Allocation in the Distributed Environments // Annals of the University of Craiova — Mathematics and Computer Science Series. 2011. V. 38. N 3. P. 76–83.

14. Jagannatha, S., Geetha, D. E., Suresh Kumar, T. V., Rajani Kanth, K. Load Balancing in Distributed Database System using Resource Allocation Approach // J. Advanced Research in Computer and Communication Engineering. 2013. V. 2. N 7. P. 2529–2535.

15. Honicky, R. J., Miller E. L. Replication Under Scalable Hashing: A Family of Algorithms for Scalable Decentralized Data Distribution / 18th International Parallel and Distributed Processing Symposium, 2004.

16. Alicherry, M., Lakshman, T. V. Network Aware Resource Allocation in Distributed Clouds / INFOCOM 2012. P. 963–971.

17. AuYoung, A., Chun, B. N., Snoeren, A. C., Vahdat, A. Resource Allocation in Federated Distributed Computing Infrastructures // First Workshop on Operating System and Architectural Support for the On-demand IT InfraStructure, 2004.

18. Raman, R., Livny M., Solomon, M. Matchmaking: Distributed Resource Management for High Throughput Computing // J. Cluster Computing. 1999. V. 2. N 1. P. 129–138.

19. Reddy, C., Bondfhugula, U. Effective Automatic Computation Placement and Data Allocation for Parallelization of Regular Programs / 28th ACM International Conference on Supercomputing. 2014. P. 13–22. ACM New York, USA.

20. Baden, S. B., Shalit, D. Performance Tradeoffs in Multi-tier Formulation of a Finite Difference Method // ICCS 2001, LNCS. V. 2073. P. 785–794. Springer, Heidelberg.

21. Ken-ichiro Ishikawa. ASURA: Scalable and Uniform Data Distribution Algorithm for Storage Clusters. Computing Research Repository, abs/1309.7720, 2013.

22. Chawla, A., Reed B., Juhnke, K., Syed, G. Semantics of Caching with SPOCA: A Stateless, Proportional, Optimally-Consistent Addressing Algorithm // USENIX Annual Technical Conference 2011. P. 33–33. USENIX Association.

23. Lowder, J. K., King, P. J. H. Using Space-Filling Curves for Multi-dimensional Indexing // Advances in Databases. LNCS. V. 1832. P. 20–35. Springer, Heidelberg, 2000.

24. Moon, B., Jagadish, H. V., Faloutsos, C., Saltz, J. H. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve // J. IEEE Transactions on Knowledge and Data Engineering. 2000. V. 13. N 1. P. 124–141.

**Малышкин Виктор Эммануилович** — д-р технич. наук, профессор ИВМиМГ СО РАН, зав. лабораторией синтеза параллельных программ, e-mail: malysh@ssd.sscc.ru, тел.: +7 (383) 330-89-94.

**В. Э. Малышкин** получил степень магистра математики в Томском государственном университете (1970), степень кандидата физико-математических наук в Вычислительном центре СО АН СССР (1984), степень доктора технических наук в Новосибирском государственном университете (1993). В настоящее время является заведующим лабораторией синтеза параллельных программ в Институте вычислительной математики и математической геофизики СО РАН. Он также основал и в настоящее время возглавляет кафедру Параллельных вычислений в Национальном исследовательском университете Новосибирска и кафедру Параллельных вычислительных технологий в НГТУ. Является одним из организаторов международной конференции PaCT (Parallel Computing Technologies), проводимой каждый нечетный год в России. Имеет более 110 публикаций по параллельным и распределенным вычислениям, синтезу параллельных программ, суперкомпьютерному программному обеспечению и приложениям, параллельной реализации крупномасштабных численных моделей. В настоящее время область его интересов включает параллельные вычислительные технологии, языки и системы параллельного программирования, методы и средства параллельной реализации крупномасштабных численных моделей, технология активных знаний.

**Victor Malyshkin** received his M. S. degree in Mathematics from the State University of Tomsk (1970), Ph.D. degree in Computer Science from the Computing Center of the Russian Academy of Sciences (1984), Doctor of Sciences degree from the State University of Novosibirsk (1993). He is presently the head of Supercomputer Software Department of the Institute of Computational Mathematics and Mathematical Geophysics, Russian Academy of Sciences. He also found and presently heading the Chair of Parallel Computing at the National Research University of Novosibirsk. He is one of the organizers of the PaCT (Parallel Computing Technologies) series of international conferences that are held each odd year in Russia. He published over 110 papers on parallel and distributed computing, parallel program synthesis,

supercomputer software and applications, parallel implementation of large scale numerical models. His current research interests include parallel computing technologies, parallel programming languages and systems, methods and tools for parallel implementation of large scale numerical models, active knowledge technology.

**Перепелкин Владислав Александрович** — ИВМ и МГ СО РАН, младш. науч. сотр., e-mail: perepelkin@ssd.sscc.ru, тел.: +7 (383) 330-89-94.

В 2008 году окончил Новосибирский государственный университет с присуждением степени магистра техники и технологии по направлению „Информатика и вычислительная техника“. В настоящее время работает младшим научным сотрудником в Институте вычислительной математики и математической геофизики СО РАН и старшим преподавателем в Новосибирском государственном университете. Имеет 18 опубликованных статей по теме автоматизации конструирования параллельных программ в области численного моделирования. Является одним из основных разработчиков экспериментальной системы фрагментированного программирования LuNA (от Language for Numerical Algorithms). Область профессиональных интересов включает автоматизацию конструирования численных параллельных программ, языки и системы параллельного программирования.

Graduated from Novosibirsk State University in 2008 with the master degree in engineering and technology in computer science. Nowadays works as a junior researcher in Institute of computational mathematics and mathematical geophysics (Russian Academy of Sciences), and also as a senior professor in Novosibirsk State University. Is an author of 18 papers on automation of numerical parallel programs construction. Is one of main developers of fragmented programming system LuNA (from Language for Numerical Algorithms). Professional interests include automation of numerical parallel programs construction, languages and systems of parallel programming.

**Щукин Георгий Анатольевич** — ИВМ и МГ СО РАН, младш. науч. сотр., e-mail: schukin@ssd.sscc.ru, тел.: +7 (913) 706-54-74.

Родился в Новосибирске в 1986 году. В 2009 окончил Новосибирский государственный технический университет со степенью магистра прикладной математики и информатики. Текущее место работы: Институт вычислительной математики и Математической геофизики, младший научный сотрудник. Участвует в разработке системы фрагментированного программирования LuNA, а также других проектах лаборатории синтеза параллельных программ. Профессиональные интересы: параллельное и системное программирование, языки программирования, компьютерная графика.

Born in Novosibirsk in 1986. In 2009 graduated from Novosibirsk State Technical University as master of Applied Mathematics and Informatics. Nowadays works as junior researcher at Institute of Computational Mathematics and Mathematical Geophysics SB RAS. Participates in development of LuNA fragmented programming system and other projects of Supercomputer Software Department. Professional interests: parallel and system programming, programming languages, computer graphics.

*Дата поступления — 18.08.2016*