AUTOMATED CONSTRUCTION OF PARALLEL NUMERICAL PROGRAMS WITH SPECIFIED NON-FUNCTIONAL PROPERTIES BASED ON COMPUTATIONAL MODELS

V. A. Perepelkin*, I. V. Sofronov, A. A. Tkacheva*

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

Implementation of numerical algorithms as parallel programs for distributed memory multicomputers is a complex task, since the program not only has to contain the algorithm implemented, but also deals with a number of other problems of system parallel programming. Those are: organize parallel computational process, synchronize parallel threads and processes, perform communications, distribute and balance computational workload and more. Implementation of such static and dynamic properties of the program requires concerning peculiarities of both multicomputer and application algorithm, or even peculiarities of the input data. Development, debugging and modification of such a parallel program are extremely complex for scientists, who need to implement their numerical models. Moreover, the program also has to be efficient (in terms of computation time, memory consumption, etc.).

To overcome the complexity program construction automation can be useful. A programmer describes the algorithm in a high-level language, and the parallel program is constructed automatically by a programming system. In this case the complexity of parallel program development is hidden from programmer. This approach also allows constructing different parallel program for different multicomputers or input data, which is potentially, allows more efficient programs, than programs, developed for general case.

In this paper we assume, that an application algorithm is described in a high-level programming language, such as LuNA language. The description is functionally decomposed, i.e. consists of functions. Each function has a pre-defined finite number of modules that implement it. The modules differ in the non-functional properties respect (for example, they may have different execution time or memory consumption). These non-functional properties of the modules are considered known. The problem of parallel program construction is considered as the problem of choice of a module for each function. Since each module has different non-functional properties, the resulting program will also have different non-functional properties. The aim is to construct a program with desired non-functional properties.

The complexity of the problem is conditioned by the fact that the dependency between the assembled program's properties and the properties of the modules can be different for different applications or even input data. Therefore, it is important to have a general way of describing the revealed dependences and including them in some knowledge base in a form that allows their automatic usage. In this case, for various limited subject areas and classes of applied algorithms, it is possible to accumulate knowledge in such base. As a result, automatic construction of an acceptable quality program would be provided.

The main point of the proposed solution is to use the computational models (CMs) described in the theory of synthesis of parallel programs on the basis of computational models for organization of such a knowledge base. Information on how the initial description of the computational process is

represented in the form of a set of functions, which modules are available for each function, and how the program is assembled from these modules, is described in the CM.

The practical application of the approach implies that the end user has no information about the CM, its composition and structure. Their role is limited to describing the computational process in a high-level language. The CM is described by the developer of the programming system. The algorithm of creating a set of acceptable modules from the CM and the construction of the program is fixed. The advantage of using the CM for solving the problem is the possibility to put in the CM a number of alternatives for designing the program. This leads to the possibility of automatic program construction in a wide range of non-functional properties required for different execution conditions.

To investigate the effectiveness of the proposed solution an explicit finite difference method for 3D Poisson equation solution was chosen as a test application. This problem was chosen as an example, because it has a typical parallel implementation scheme used for a wide class of numerical algorithms on meshes (spatial decomposition with border exchange). Within the framework of this test the one-dimensional spatial decomposition of the three-dimensional computational domain was used. Two series of experiments were conducted: in shared and distributed computing environment. The results demonstrated the desired ability of automated program construction with different non-functional properties, preferred for different execution conditions.

Key words: automated parallel programs synthesis, computational models, program auto-tuning, fragmented programming technology, fragmented programming system LuNA, code generation.

References

1. Kale L. V., Krishnan S. CHARM++: a portable concurrent object oriented system based on C++ // OOPSLA'93 Proc. of the eighth annual conf. on Object-oriented programming systems, languages and applications. ACM. NewYork. 1993. P. 91–108.

2. OpenMP. Available at: http://www.openmp.org/

3. Valkovskii, V. A., Malyshkin V. E. Sintez parallelnykh program i system na vychislitelnih modelyakh [Synthesis of systems and programs on computation models]. Novosibirsk: Nauka, SB RAS 1988.

4. Malyshkin V. E., Perepelkin V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // In the Proc. of the 11th Intern. Conf. on Parallel Computing Technologies (PaCT 2011), LNCS. V.∼6873, Springer, 2011. P. 53–61.

5. Brusentsov L. E. Avtomaticheskaya optimizatsiya pri kompilyatsii [Automatic optimization during compilation] // Open systems, 2011, N 2. P. 53–56.

6. Brusentsov L. E. Raspredelyonnii iterativnyi poisk optsij kompilyatsii [Distributed iterative search of compiler's options] // Distributed, Informational and Computational Resources (DICR'2010): conf. proc. Novosibirsk: ICT SB RAS, 2010 (CD-ROM). reg. № 0321100051. Available at: `http://conf.nsc.ru/dicr2010/ru/reportview/29810`. (Accessed 17.10.2017)

7. Purini S., Jain L. Finding Good Optimization Sequences Covering Program Space // ACM Transactions on Architecture and Code Optimization, Vol. 9. N 4. Article 39. 2010.

8. Michael R. Jantz, Prasad A. Kulkarni Performance Potential of Optimization Phase Selection During Dynamic JIT Compilation // VEE'13, March 16–17, 2013, Houston, Texas, USA.

9. The GNU Compiler Collection (GCC). Available at: `https://gcc.gnu.org.html` (Accessed 19.10.2017).

10. Using the GNU Compiler Collection (GCC): Optimize Options. Available at: `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html` (Accessed 18.10.2017).

11. Leather H., Bonilla E., O'Boyle M. Automatic Feature Generation for Machine Learning Based Optimizing Compilation // Proc. of the 7th annual IEEE/ACM Intern. Symposium on Code Generation and Optimization, 2009. P. 81–91.

12. Ashouri A. H., Palermo G., Silvano C. Auto-tuning Techniques for Compiler Optimization. // DATE'2016- PhD Forum, IEEE/ACM, Dresden, Germany, 2016.

13. Hashimoto M., Terai M., Maeda T. and Minami K. Extracting Facts from Performance Tuning History of Scientific Applications for Predicting Effective Optimization Patterns. // Proc. of the 12th Working Conf. on Mining Software Repositories, 2015. P. 13–23.

14. 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, S.I.: Parallel Computing Technologies. 2016. Springer, 2016. P. 1–7.

15. Malyshkin V. E., Perepelkin V. A., Tkacheva A. A. Control Flow Usage to Improve Performance of Fragmented Programs Execution. PaCT 2015. LNCS, Vol. 9251, Springer, 2015. P. 86–90.

16. Tkacheva, A. A. Effectivnoe ispolnenie fragmentirovannykh programm s pomoshh'yu sredstv pryamogo upravlenia v sisteme LuNA na primere zadachi redutsirovaniya dannykh [Efficient execution of fragmented program with a control flow means under the fragmented parallel programming system LuNA in data reduction task] / A. A. Tkacheva // Problems of Informatics. 2016. N 2 (31). P. 21–29.

17. Tkacheva A. A. The Algorithm of Control Program Generation for Optimization of LuNA Program Execution. PaCT 2017. LNCS, V. 10421. Springer, 2017. P. 365–371.

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

19. Mezhvedomstvennyj Supercomp'yuternyj Tsentr Rossijskoj Akademii Nauk [Joint Supercomputer Center of the Russian Academy of Sciences] Available at: `http://www.jscc.ru/scomputers.html`

АВТОМАТИЗАЦИЯ КОНСТРУИРОВАНИЯ ЧИСЛЕННЫХ ПАРАЛЛЕЛЬНЫХ ПРОГРАММ С ЗАДАННЫМИ НЕФУНКЦИОНАЛЬНЫМИ СВОЙСТВАМИ НА БАЗЕ ВЫЧИСЛИТЕЛЬНЫХ МОДЕЛЕЙ

В. А. Перепелкин*, И. В. Софронов, А. А. Ткачева*

Институт вычислительной математики и математической геофизики СО РАН,
630090, Новосибирск, Россия
*Новосибирский национальный исследовательский государственный университет,
630090, Новосибирск, Россия

УДК 00.004.4'24

В работе рассматривается проблема автоматизации конструирования параллельных программ численного моделирования с заданными нефункциональными свойствами. Задача конструирования программы рассматривается как задача выбора для каждой функции, входящей в исходное описание алгоритма решения задачи, реализующий ее модуль из числа заранее заданных, каждый из которых обладает разными (известными) нефункциональными свойствами. В качестве математического аппарата для формального описания связи между свойствами отдельных модулей и свойствами результирующей программы предлагается использовать вычислительные модели, описанные в теории структурного синтеза параллельных программ и систем на вычислительных моделях. На базе системы фрагментированного программирования LuNA, ориентированной на автоматическое конструирование параллельных программ численного моделирования, проведен эксперимент по конструированию параллельной программы с заданными нефункциональными свойствами.

Ключевые слова: автоматизация синтеза параллельных программ, кодогенерация, автоматическая настройка программ, технология фрагментированного программирования, система фрагментированного программирования LuNA.

Введение. В области научного численного моделирования с применением суперкомпьютеров стоит проблема разработки параллельной программы (ПП), реализующей заданный прикладной численный алгоритм. Необходимым требованием к ПП является ее эффективность (тут и далее эффективность понимается в смысле времени выполнения ПП, расхода памяти, нагрузки на коммуникационную подсистему и т. п.). Для этого ПП должна обеспечивать настройку на доступные ресурсы, равномерность распределения нагрузки по вычислительным узлам во времени, осуществление коммуникаций на фоне вычислений и другие свойства. Таким образом, ПП должна учитывать особенности вычислителя, прикладного алгоритма и входных данных задачи. Разработка, отладка и модификация такой ПП становятся чрезвычайно сложными для пользователей суперкомпьютеров, особенно, если вычислитель является неоднородным (состоит из разных вычислительных узлов), гетерогенным (содержит GPU и другие спецвычислители), а прикладная задача характеризуется динамикой, не позволяющей заранее спланировать вычисления, распределение данных по узлам и/или коммуникации.

В свете обозначенной проблемы актуальной является задача автоматизации конструирования ПП, при которой пользователь ограничивается описанием параллельного вычислительного процесса на относительно высоком уровне абстракции, не вдаваясь в подробности решения обозначенных проблем из области системного параллельного программирования, а требуемая ПП конструируется автоматически. В частности, при таком подходе открывается возможность конструировать разные ПП для разных вычислителей и/или классов входных данных. Такая частная ПП выигрывает в эффективности у ПП, рассчитанной на общий случай.

Высокий уровень абстракции допускает различные способы реализации описанного вычислительного процесса. Например, в системах, таких как [1, 2], имеется возможность как статического планирования выполнения задач, так и их динамического выбора из общего пула по мере освобождения ресурсов. В различных ситуациях оптимальным может оказываться и тот, и другой вариант.

Таким образом, перед системой конструирования ПП встает проблема принятия решений о выборе оптимального (или, по крайней мере, приемлемого) модуля, реализующего тот или иной элемент заданного описания вычислительного процесса (функции) для построения ПП. Т. к. решения могут быть взаимосвязаны, то они должны приниматься комплексно. Как следствие, для автоматизации принятия таких решений при конструировании ПП возникает задача организации, накопления и использования некоторой базы знаний о том, как следует принимать эти решения. Обеспечение возможности накапливать и автоматически применять такие знания является одним из важнейших вопросов в сфере автоматизации конструирования ПП для суперкомпьютеров.

Работа посвящена попытке применения математического аппарата вычислительных моделей [3] для организации такой базы знаний. Проведен эксперимент по автоматизированному конструированию ПП с высокоуровневого описания численного алгоритма на языке LuNA [4].

Статья организована следующим образом. В 1-м разделе приводится обзор родственных работ. В разделе 2 приводятся необходимые термины и определения. В 3-м разделе описываются постановка задачи и предлагаемый способ организации базы знаний. В разделе 4 описываются эксперименты по конструированию ПП и их результаты.

1. Обзор родственных работ. Аналогичная проблема выбора эффективной реализации возникает в компиляторах императивных языков программирования в процессе оптимизации программы. Из доступного множества оптимизаций компилятора необходимо выбрать некоторое их подмножество так, чтобы улучшить характеристики результирующей программы относительно некоторого критерия и условий выполнения.

Одним из вариантов решения проблемы является итеративный поиск, который заключается в полном переборе всех возможных вариантов применения оптимизаций путем повторения цикла построения, запуска программы и замера производительности [5]. Такой подход плохо масштабируется относительно количества рассматриваемых вариантов и приводит к экспоненциальному увеличению времени компиляции. Существуют попытки улучшить ситуацию путем применения параллельных вычислений для поиска вариантов [6], но при реализации крупномасштабных численных моделей это не играет существенной роли, т. к. выполнение программы может занимать длительное время. Некоторым улучшением метода итеративного поиска является использование генетических и других эвристических алгоритмов [7, 8].

Для преодоления недостатков описанных подходов применяют методы, позволяющие динамически создавать наборы применяемых оптимизаций и сужать область поиска оптимизаций, чтобы не делать полный перебор. В таких подходах используется факт присутствия закономерности между выбранным набором оптимизаций и свойствами результирующей программы. После выявления этой закономерности путем анализа и/или тестирования она фиксируется в компиляторе для последующего использования. Подходы различаются способами хранения закономерностей и алгоритмами работы с ними.

Наиболее распространенным является использование фиксированных наборов оптимизаций. Наборы составляются таким образом, чтобы для большинства программ определенного класса генерировалась оптимальная программа (или близкая к ней по качеству). Выбор набора из небольшого числа заранее подготовленных лежит на пользователе. Так, например, в известном компиляторе GCC (GNU Compiler Collection) [9] используются ключи вида *–O уровень* [10]. Однако такой подход не обладает гибкостью: для более тонкой настройки программы на вычислитель требуется ручное задание дополнительных опций.

В методах машинного обучения на основе некоторой конечной выборки примеров создается модель предметной области, которая отражает зависимости, содержащиеся в этих примерах. Созданная модель должна имитировать поведение реальной системы при поступлении схожих данных. Реализация этого подхода описана в [10, 11]. Суть заключается в том, что в каждой программе выделяются характеризующие ее особенности, эти особенности представляются в виде n-мерного вектора, который подается на вход нейронной сети, и на выходе получается набор оптимизаций, которые нужно применить к программе.

В некоторых подходах (напр., [12]) для хранения существующих особенностей программ применяются онтологии. Онтологии представляют собой множество понятий некоторой предметной области, для которого определено множество бинарных отношений. С помощью онтологий можно составить описание программы, которое компьютер способен распознать и использовать для выбора варианта методами машинного обучения. Недостатком подхода является то, что обученная система работает только в рамках ситуаций, которые схожи с обучающей выборкой, иначе для эффективной работы системы нужно производить переобучение, что является длительным и сложным процессом. К тому же, собрать достаточно большую и репрезентативную выборку примеров не всегда удается.

Рассмотренные подходы обладают рядом недостатков и полностью обозначенной проблемы не решают, поэтому проработка новых подходов является актуальной задачей.

2. Термины и определения. *Вычислительные Модели.*

Приведем определение вычислительной модели из [3]. Пусть заданы:

— конечное множество $\boldsymbol{X}=\{x,\ y,\ z,\ \ldots\}$ переменных для представления вычисляемых и измеряемых величин;

— конечное множество $\boldsymbol{F}=\{a,\ b,\ c,\ \ldots\}$ символов операций арности $m\times n$, $m\geqslant 0$, $n\geqslant 0$ (m и n различные для различных символов);

— с каждым символом операции a арности $m\times n$ связан набор $\mathbf{in}(a)=(x_1,\ \ldots,\ x_m)$ входных и набор $\mathbf{out}(\mathrm{a})=(y_1,\ \ldots,\ y_n)$ различных выходных переменных.

Пара $\boldsymbol{C}=(\boldsymbol{X},\ \boldsymbol{F})$ называется простой вычислительной моделью (далее — вычислительная модель, ВМ). Операция $a\in\boldsymbol{F}$ описывает возможность вычисления переменных $\mathbf{out}(a)$ из переменных $\mathbf{in}(a)$ с помощью некоторой процедуры.

ВМ можно представить конечным двудольным орграфом (см. пример на рис. 1).

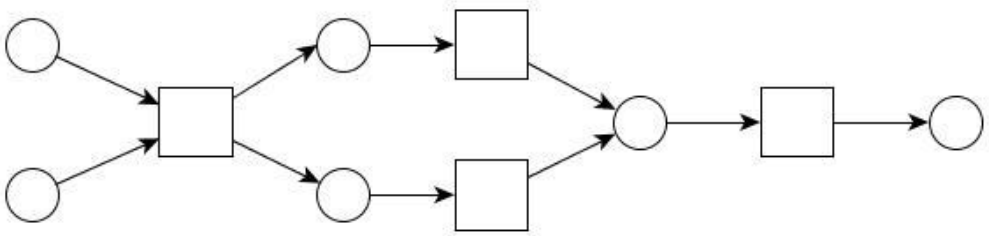

Рис. 1. Вычислительная модель; круги — переменные, квадраты — операции

ВМ в целом описывает величины некоторой предметной области и возможность получать одни величины из других.

Определим понятие задачи на ВМ в соответствии с [3]. Пусть заданы два подмножества: входных переменных $\boldsymbol{V}\subseteq\boldsymbol{X}$ и выходных переменных $\boldsymbol{W}\subseteq\boldsymbol{X}$. Пара $\boldsymbol{T}=(\boldsymbol{V},\ \boldsymbol{W})$ называется задачей. Подграф $\boldsymbol{P}\subseteq\boldsymbol{C}$ называется планом решения задачи, если он содержит $\boldsymbol{V}$ и $\boldsymbol{W}$ и все переменные из $\boldsymbol{W}$ вычислимы через операции из $\boldsymbol{V}$ (возможно, через промежуточные переменные из $\boldsymbol{P}$). Для одной и той же задачи может существовать несколько планов решения. В [3] также рассматривается процесс вывода оптимального плана для заданной задачи, при этом оптимизация осуществляется по числу операций и/или переменных. По плану решения задачи очевидным образом можно воспроизвести процесс (например, запрограммировав в виде программы) вычисления значений переменных $\boldsymbol{W}$ по заданным входным $\boldsymbol{V}$.

Применение ВМ для рассматриваемой проблемы представляется целесообразным ввиду того, что ВМ позволяет описывать процесс синтеза программы на желаемом уровне абстракции, описывать различные варианты выбора модулей для реализации отдельных частей (функций) вычислительного процесса, ставить и решать задачу синтеза оптимальной программы.

Расширим понятие ВМ понятием атрибутов. Пусть имеется конечное множество атрибутов $\boldsymbol{A}=\{A_1,\ \ldots A_k\}$. Пусть для каждого атрибута A_i задано множество его значений $\boldsymbol{D}_i$. Каждой операции и переменной ставится в соответствие значение некоторого подмножества атрибутов (возможно, пустое). Кроме того, пусть имеется функция качества плана $\boldsymbol{Q}$, которая ставит в соответствие каждому плану $\boldsymbol{P}$ его оценку качества Q_p, определяемую переменными и операциями плана и их атрибутами. Поиск оптимального плана, таким образом, может осуществляться не на основе количества операций и переменных, а в виде произвольной закономерности через задания атрибутов и функции качества. На практике может использоваться несколько различных функций качества для оптимизации конструирования программы по различным критериям.

3. Постановка задачи и предлагаемое решение. В рамках работы задача конструирования оптимальной программы рассматривается следующим образом. Предполагается, что имеется некоторое описание вычислительного процесса (например, на языке программирования высокого уровня, таком как язык LuNA [4]). Предполагается также, что это описание состоит из ряда функций, для реализации каждой из которых имеется конечный набор модулей. Каждый модуль обладает известными нефункциональными свойствами (тут и далее под нефункциональными свойствами понимается время выполнения, расход памяти, нагрузка на коммуникационную подсистему, обладание динамически-

ми свойствами и т. п. свойства, не влияющие на вычисляемые значения, но существенные с точки зрения эффективности). Требуется выбрать один из модулей для каждой функции, после чего производится компоновка выбранных модулей в результирующую программу в соответствии с описанием вычислительного процесса. Тот факт, что каждый модуль обладает различными нефункциональными свойствами приводит к тому, что результирующая программа также может обладать различными нефункциональными свойствами. Задача состоит в том, чтобы добиться желаемых свойств результирующей программы путем выбора модуля для каждой функции.

Сложность задачи в том, что характер зависимости свойств результирующей программы от свойств модулей может быть самым разным в разных предметных областях, классах прикладных алгоритмов и даже зависеть от входных данных. Поэтому важно иметь некоторый универсальный способ описания выявленных закономерностей и включения их в некоторую базу знаний в виде, допускающем их автоматическое использование. В таком случае для различных ограниченных предметных областей и классов прикладных алгоритмов возможно накопление знаний в базе, в результате которого обеспечивалось бы автоматическое конструирование программы приемлемого качества.

Суть предлагаемого решения заключается в использовании ВМ для организации такой базы знаний. Информация о том, как исходное описание вычислительного процесса (исходная программа на языке высокого уровня абстракции) представляется в виде множества функций (т. е. какова функциональная декомпозиция алгоритма), какие модули доступны для каждой функции и как компонуется результирующая программа из этих модулей, вносится в ВМ в виде операций и переменных. Информация о нефункциональных свойствах модулей вводится в ВМ в форме атрибутов операций и переменных.

Практическое применение подхода подразумевает, что конечный пользователь не имеет информации об атрибутах, функции качества и ВМ. Его роль ограничивается описанием вычислительного процесса на языке высокого уровня. Описание ВМ, набор и значения атрибутов составляет и поддерживает разработчик системы программирования (компилятора). Алгоритм вывода оптимального плана и конструирования результирующей программы не изменяется.

Преимуществом использования ВМ для решения данной задачи является возможность вкладывать в ВМ множество различных вариантов конструирования результирующей программы. Это обеспечивает возможность автоматического конструирования программы в широком спектре нефункциональных свойств, требуемых для тех или иных условий запуска.

4. Экспериментальное исследование. В рамках настоящей работы был проведен эксперимент по автоматизации конструирования параллельных программ на базе системы фрагментированного программирования LuNA [4]. Эта система была выбрана в качестве базовой платформы исследования, т. к. она является системой конструирования параллельных программ численного моделирования для суперкомпьютеров на основе высокоуровневого описания вычислительного процесса (на языке LuNA).

В рамках работы, для обеспечения требуемой вариативности конструирования параллельной программы, было создано расширение системы LuNA, которое конструирует реализацию заданной функции определенного вида специализированным алгоритмом, что позволило, в некоторых случаях, иметь для таких функций два модуля с разными нефункциональными свойствами. Информация об этих способах конструирования была выражена

в виде ВМ. На этой ВМ ставилась и решалась разработанным для этой цели интерпретатором задача конструирования оптимальной по времени исполнения программы.

Далее в разделе приводятся сведения о системе LuNA и описание эксперимента.

4.1. Язык и система LuNA. В системе LuNA пользователь описывает вычислительный алгоритм без привязки к ресурсам в рамках модели dataflow. Такое описание называется фрагментированным алгоритмом (ФА). Отличительными особенностями ФА являются иммутабельность (единственное присваивание) данных и отсутствие побочных эффектов у вычислительных модулей. Данные и вычисления ФА явно разделены на т. н. фрагменты данных и вычислений (ФД и ФВ соответственно). Исполнение ФА подразумевает назначение ФД и ФВ на вычислительные устройства и выбор порядка выполнения ФВ (который не должен противоречить информационным зависимостям). От того, каким образом будет осуществляться это назначение, зависят нефункциональные свойства исполнения ФА. В частности, динамическое назначение ФД может быть более точным, но приводит к появлению накладных расходов во время исполнения.

Как правило, ФА исполняется в режиме интерпретации исполнительной системой LuNA. Для повышения эффективности исполнения ФА применяются средства прямого (императивного) управления [15, 16]. Они позволяют на стадии компиляции принять ряд решений по управлению вычислениями и распределению ресурсов и зафиксировать их в виде жесткого (без динамических свойств) управляющего модуля, работающего в специализированном исполнительном окружении, называемом LuNA Framework (LuNA-FW). Работа такого модуля основана на event-driven модели (в отличие от исходной dataflow модели), что позволяет, в некоторых случаях, достигнуть более эффективного исполнения ФА.

Был разработан и реализован алгоритм автоматической генерации управляющих программ для LuNA-FW [17] для заданных функций в ФА. Алгоритм распознает и обрабатывает функции частного вида, которые можно неформально охарактеризовать как итерационные процессы на сетках. Принадлежность заданной функции этому классу проверяется очевидным алгоритмом статического анализа на стадии компиляции ФА.

Сгенерированная управляющая программа для LuNA-FW содержит то же множество ФВ над теми же наборами ФД, что и в исходном ФА, а порядок выполнения ФВ не противоречит их информационным зависимостям, что обеспечивает корректность преобразования. При этом использование event-driven модели позволяет сократить количество накладных расходов по организации вычислений внутри и между узлами, что позволяет получить лучшую производительность исполнения подпрограммы по сравнению с системой LuNA.

Таким образом, для каждой функции в ФА имеется возможность конструировать до 2-х различных реализующих модулей — LuNA-реализацию с динамическими свойствами, работу которой контролирует исполнительная система LuNA (этот модуль доступен всегда), и LuNA-FW реализацию без динамических свойств, но более эффективную в случаях, когда динамические свойства не требуются (ее возможно генерировать для подпрограмм частного вида).

Совместная работа модулей LuNA и LuNA-FW осуществляется в общем системном окружении.

4.2. Описание эксперимента. Для исследования эффективности предложенного алгоритма была использована фрагментированная реализация явной конечно-разностной схемы для решения уравнения Пуассона в трехмерном пространстве с начальными крае-

Таблица 1

Результаты запусков различных реализаций функции, представляющей основной цикл вычислений, в зависимости от количества процессов (общая память)

	1	2	4	8	16
LuNA	581,150	289,230	152,910	83,755	71,716
LuNA-FW	301,100	168,880	85,550	86,186	109,070

выми условиями первого рода [18]:

$$\nabla^2\phi = f,$$

$$\phi|_{G} = F.$$

Значения в узлах сетки вычислялись по формуле:

$$\phi^{m+1}_{ijk} = \frac{\frac{\phi^{m}_{i+1jk}+\phi^{m}_{i-1jk}}{h^{2}_{x}} + \frac{\phi^{m}_{ij+1k}+\phi^{m}_{ij-1k}}{h^{2}_{y}} + \frac{\phi^{m}_{ijk+1}+\phi^{m}_{ijk-1}}{h^{2}_{z}} - f_{ijk}}{\frac{2}{h^{2}_{x}} + \frac{2}{h^{2}_{y}} + \frac{2}{h^{2}_{z}}}.$$

Эта задача была выбрана как пример, имеющий характерную схему параллельной реализации, применяемую для реализации широкого класса численных алгоритмов на сетках (пространственная декомпозиция с обменами на границах). В рамках настоящего тестирования применялась одномерная пространственная декомпозиция трехмерной расчетной области.

Запуск задач производился на кластере МВС-10П [19] (2 процессора Xeon E5-2690, 64 ГБ оперативной памяти, коммуникационная сеть на базе FDR Infiniband).

Параметры тестовой задачи были выбраны близкими к параметрам, используемым на практике для такого рода задач: 128 итераций; размер расчетной области 1200×300×300; размер домена 30×300×300, количество доменов: 40.

Для функции, представляющей основной цикл вычислений, имелось два реализующих модуля, обладающих различными нефункциональными свойствами: базовый dataflow (LuNA) и императивный event-driven (LuNA-FW). Для всех остальных функций, присутствующих в описании, был выбран базовый модуль LuNA.

Были проведены две серии тестов: в общей и распределенной памяти.

В табл. 1 приведены результаты запусков LuNA и LuNA-FW программ. Видно, что в зависимости от конфигурации вычислителя выбор модуля влияет на время выполнения результирующей программы: LuNA-FW быстрее выполняется на малом количестве процессов, LuNA — на большом (8 и более). Таким образом, тест является репрезентативным с точки зрения того, что имеются различные варианты реализации одного и того же алгоритма, каждый из которых является оптимальным в некоторой ограниченной области параметров исполнения.

Информация о таких нефункциональных свойствах модулей и их связи с нефункциональными свойствами результирующей программы была зафиксирована в виде ВМ, атрибутов и функции оценки качества. На основе этого описания, в соответствии с описанным выше подходом, интерпретатором ВМ автоматически выбирался оптимальный модуль (см. рис. 2). Количество процессов было параметром при оптимизации конструирования программы.

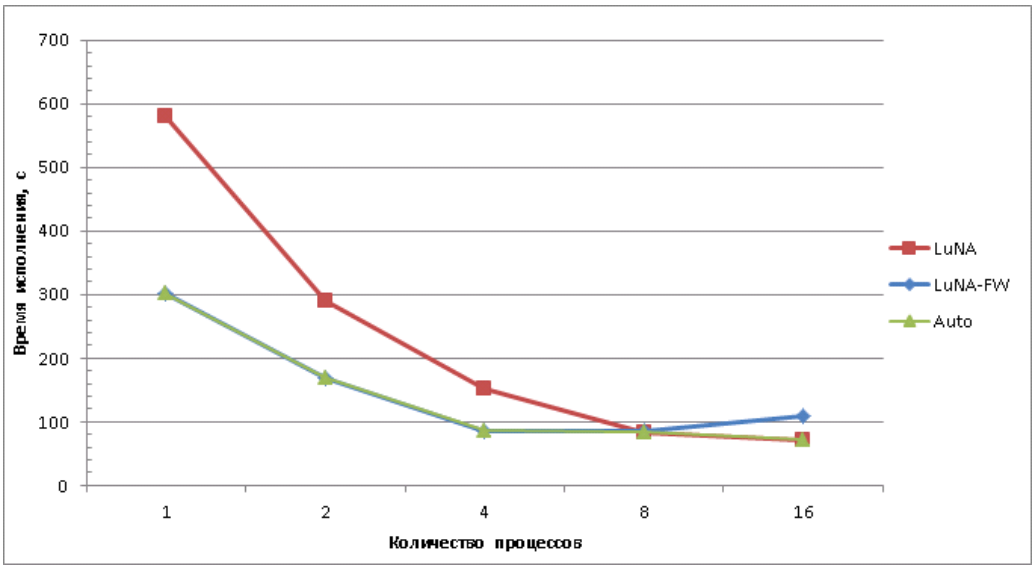

Рис. 2. Автоматический выбор оптимальной реализации (Auto) среди двух доступных вариантов конструирования параллельной программы (общая память)

Таблица 2

Результаты запусков различных реализаций функции, представляющей основной цикл вычислений, в зависимости от количества узлов (распределенная память)

	1/1	8/1	8/2	8/4
LuNA	591,966	88,143	82,236	81,101
LuNA-FW	301,103	101,259	63,808	97,749

Во втором эксперименте исследовалось поведение системы в распределенной памяти при разном количестве узлов. Количество процессов оставалось постоянным и равнялось 8, количество узлов варьировалось от одного до четырех. Для функции, представляющей основной цикл вычислений, также было доступно два модуля: LuNA и LuNA-FW. В табл. 2 приведено время работы каждой из построенных ПП при выборе того или иного модуля в разных условиях запуска. В первой строке таблицы первая цифра означает общее количество процессов, вторая — количество задействованных узлов.

На рис. 3 графиком Auto отражен результат работы ПП, реализацию которых выбирал интерпретатор ВМ. Также как и в предыдущем эксперименте, по введенным данным о вычислителе принималось решение о выборе оптимального модуля для реализации функции, представляющей основной цикл вычислений. Выбор осуществлялся следующим образом: если запускается на одном узле на множестве процессов, то выбирается модуль LuNA (точка 8/1), если же на узле запускается мало процессов (в сравнении с общим количеством — восемью) и мало узлов, то выбирается модуль LuNA-FW (точка 8/2).

Заключение. Рассмотрена проблема автоматизации конструирования параллельных программ с требуемыми нефункциональными свойствами на основе имеющегося функционального описания алгоритма решения задачи численного моделирования. Описание представляет собой множество функций, связанных друг с другом зависимостями по данным, этими зависимостями описывается частичный порядок вычислений. Предложен под-

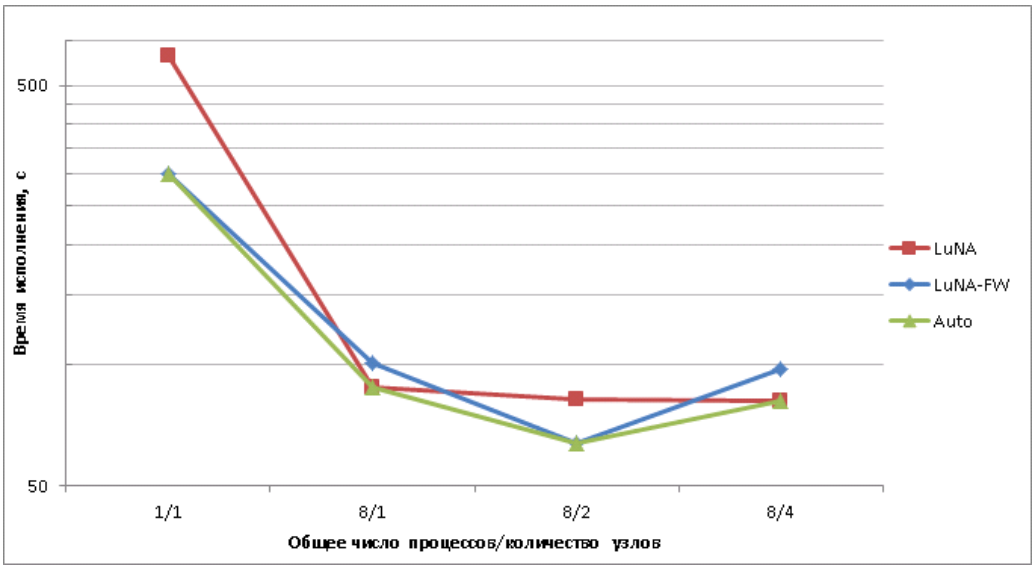

Рис. 3. Автоматический выбор реализации (зеленый график) среди двух доступных модулей (распределенная память)

ход к использованию вычислительных моделей для представления и применения знаний о том, какими модулями с различными нефункциональными свойствами каждая функция, описанная в алгоритме, может быть реализована. Разработаны системные алгоритмы и программные модули для экспериментального исследования предложенного подхода. Показана возможность автоматического выбора оптимальной реализации в частном случае на примере конструирования программы из описания алгоритма решения уравнения Пуассона явным методом в рамках системы фрагментированного программирования LuNA.

В дальнейшем планируется развивать представленный подход системными алгоритмами, обеспечивающими автоматическое конструирование эффективных параллельных численных программ в более широком классе ситуаций.

Список литературы

1. Kale L.V., Krishnan S. CHARM++: a portable concurrent object oriented system based on C++ // OOPSLA'93 Proc. of the eighth annual conf. on Object-oriented programming systems, languages and applications. ACM. New York. 1993. P. 91–108.

2. OpenMP [Электронный ресурс].: http://www.openmp.org/

3. Вальковский В. А., Малышкин В. Э. Синтез параллельных программ и систем на вычислительных моделях. Новосибирск: Наука. Сиб. отд-ние, 1988.

4. Malyshkin V. E., Perepelkin V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // In the Proc. of the 11th Intern. Conf. on Parallel Computing Technologies (PaCT 2011), LNCS. V.~6873, Springer, 2011. P. 53–61.

5. Брусенцов Л. Е. Автоматическая оптимизация при компиляции // Открытые системы, 2011, № 2. С. 53–56.

6. Брусенцов Л. Е. Распределенный итеративный поиск опций компиляции // XIII Росс. конф. с участием иностранных ученых „Распределенные, информационные и вычислительные ресурсы“ (DICR'2010): материалы конф. — электр. данные. Новосибирск: ИВТ СО РАН, 2010 (CD-ROM). № гос. регистрации 0321100051. — http://conf.nsc.ru/dicr2010/ru/reportview/29810.

7. Purini S., Jain L. Finding Good Optimization Sequences Covering Program Space // ACM Transactions on Architecture and Code Optimization, Vol. 9. N 4. Article 39. 2010.

8. Michael R. Jantz, Prasad A. Kulkarni Performance Potential of Optimization Phase Selection During Dynamic JIT Compilation// VEE'13, March 16–17, 2013, Houston, Texas, USA.

9. The GNU Compiler Collection (GCC) [Electron. Res.].: https://gcc.gnu.org.html.

10. Using the GNU Compiler Collection (GCC): Optimize Options [Electron. Res.].: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html.

11. Leather H., Bonilla E., O'Boyle M. Automatic Feature Generation for Machine Learning Based Optimizing Compilation // Proc. of the 7th annual IEEE/ACM Intern. Symposium on Code Generation and Optimization, 2009. P. 81–91.

12. Ashouri A. H., Palermo G., Silvano C. Auto-tuning Techniques for Compiler Optimization // DATE'2016- PhD Forum, IEEE/ACM, Dresden, Germany, 2016.

13. Hashimoto M., Terai M., Maeda T. and Minami K. Extracting Facts from Performance Tuning History of Scientific Applications for Predicting Effective Optimization Patterns. // Proc. of the 12th Working Conf. on Mining Software Repositories, 2015. P. 13–23.

14. 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, S.I.: Parallel Computing Technologies — 2016. Springer, 2016. P. 1–7.

15. Malyshkin V. E., Perepelkin V. A., Tkacheva A. A. Control Flow Usage to Improve Performance of Fragmented Programs Execution. PaCT 2015. LNCS, Vol. 9251, Springer, 2015. P. 86–90.

16. Ткачева А. А. Эффективное исполнение фрагментированных программ с помощью средств прямого управления в системе LuNA на примере задачи редуцирования данных // Проблемы Информатики. 2016. № 2 (31). С. 21–29.

17. Tkacheva A. A. The Algorithm of Control Program Generation for Optimization of LuNA Program Execution. PaCT 2017. LNCS, V. 10421. Springer, 2017. P. 365–371.

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

19. Межведомственный Суперкомпьютерный Центр Российской Академии Наук. [Электронный ресурс].: http://www.jscc.ru/scomputers.html

Перепелкин Владислав Александрович — младш. науч. сотрудник Института вычислительной математики и математической геофизики СО РАН; старший преподаватель каф. Параллельных вычислений, факультет информационных технологий, Новосибирский национальный исследовательский государственный университет; Тел.: (383) 330-89-94, e-mail: `perepelkin@ssd.sscc.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 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, 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 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, high performance computing.

Софронов Иван Викторович — инженер Института вычислительной математики и математической геофизики СО РАН; магистрант каф. Параллельных вычислений, факультет информационных технологий, Новосибирский национальный исследовательский государственный университет; Тел.: (383) 330-89-94, e-mail: ivan.sof@bk.ru.

Софронов Иван Викторович получил степень бакалавра в 2015 году в Томском государственном университете систем управления и радиоэлектроники и степень магистра в 2017 году в Новосибирском государственном университете по специальности „Информатика и вычислительная техника“. В настоящее время является аспирантом в Институте вычислительной математики и математической геофизики СО РАН. Среди научных интересов параллельное программирование, автоматизация построения программ, теория построения компиляторов и кодогенерация.

Sofronov Ivan Viktorovich received a bachelor's degree in 2015 at the Tomsk State University of Control Systems and Radio electronics and a master's degree in 2017 at the Novosibirsk State University, both degrees in Computer science. Currently he is a post-graduate student at the Institute of Computational Mathematics and Mathematical Geophysics of the SB RAS. Research interests include parallel programming, the automation of program construction, the theory of compiler construction and code generation.

Ткачева Анастасия Александровна — младш. науч. сотрудник Института вычислительной математики и математической геофизики СО РАН; ассистент каф. Параллельных вычислений, факультет информационных технологий, Новосибирский национальный исследовательский государственный университет; Тел.: (383) 330-89-94, e-mail: tkacheva@ssd.sscc.ru.

Ткачева Анастасия Александровна получила степень бакалавра в 2011 году и степень магистра в 2013 году в Новосибирском государственном техническом университете по специальности „Прикладная математика и информатика“. С 2013 года по настоящее время работает младшим научным сотрудником и с 2014 года учится в аспирантуре в Институте вычислительной математики и математической геофизики СО РАН, а также работает ассистентом кафедры Параллельных вычислений, факультет информационных технологий, в Новосибирском национальном исследовательском государственном университете. Ее научными интересами являются высокопроизводительное вычисление, разработка системных алгоритмов параллельного программирования по реализации численных методов на суперкомпьютере.

Anastasia Aleksandrovna Tkacheva received the B.S. degree in 2011 and the M.S. degree in 2013, both in Applied Mathematics and Computer Science at Novosibirsk State Technical University, Novosibirsk, Russia. From 2013 to present, she has been working as the junior researcher at the Institute of Computational Mathematics and Mathematical Geophysics SB RAS (ICM&MG SB RAS) and also she has been working as the Assistant professor at the Parallel computing department, Novosibirsk State University. Since 2014 she's a Ph.D. student in the ICM&MG SB RAS. Her research interests include high-performance computing, system parallel algorithms development for implementation of large-scale numerical models on supercomputers.

Дата поступления — 09.06.2017