

ЭФФЕКТИВНОЕ ИСПОЛНЕНИЕ ФРАГМЕНТИРОВАННЫХ ПРОГРАММ С ПОМОЩЬЮ СРЕДСТВ ПРЯМОГО УПРАВЛЕНИЯ В СИСТЕМЕ LuNA НА ПРИМЕРЕ ЗАДАЧИ РЕДУЦИРОВАНИЯ ДАННЫХ

А. А. Ткачева

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

Новосибирский национальный исследовательский государственный университет,
630090, Новосибирск, Россия

УДК 519.685.1

Рассмотрена проблема эффективного исполнения фрагментированной программы (ФП) в системе фрагментированного программирования LuNA. Для повышения производительности исполнения ФП разрабатываются средства задания прямого управления [1, 2], которые позволяют избежать накладных расходов на организацию вычислений внутри узла мультимикрокомпьютера, вычислителя с распределенной памятью. В работе представлены разработка одного из них для случая циклических конструкций в ФП, а также описание реализации для его поддержки на уровне компиляции в системе LuNA. Представлено сравнительное тестирование производительности исполнения ФП с использованием разработанного средства и без в общей и распределенной памяти на примере задачи редуцирования данных.

Ключевые слова: параллельное программирование, фрагментированное программирование, средства прямого управления.

The problem of efficient execution of fragmented program (FP) under the fragmented parallel programming system LuNA is considered. To achieve good performance of FP execution the control flow means was designed. A compiler module, supported it, was developed. FP execution under system LuNA was compared with FP execution under control flow means on computer with shared and distributed memory.

Key words: parallel programming, control flow means, fragmented programming.

Введение. Для достижения приемлемой производительности исполнения параллельных программ численного моделирования от разработчика требуются не только знания специфики задачи, но и знания системного параллельного программирования, а именно: обеспечение динамических свойств параллельной программы, таких как динамическая балансировка загрузки, настройка на все доступные ресурсы, реализация коммуникаций на фоне вычислений и т. д.

В лаборатории синтеза параллельных программ ИВМ и МГ СО РАН разрабатывается система фрагментированного программирования LuNA [3], предназначенная для автоматизации процесса реализации задач численного моделирования на суперкомпьютере, в которой автоматизированно обеспечиваются эти свойства. В ней прикладная программа собирается из множества фрагментов данных (ФД) и фрагментов вычислений (ФВ), и

фрагментированная структура программы сохраняется в ходе вычислений. Такой подход к распараллеливанию позволяет программировать на более высоком уровне и уйти от некоторых сложностей системного параллельного программирования. Описание фрагментированного алгоритма является платформенно независимым, а настройку на конкретный вычислитель обеспечивает runtime-система LuNA. Система LuNA может исполнять ФП параллельно в многопоточном режиме в случае вычислителя с общей памятью и в многопроцессорном и многопоточном режимах совместно в случае вычислителя с распределенной памятью.

Будем называть потоковым управлением множество ограничений на порядок выполнения ФВ типа $a \leq b$ (ФВ a должен начать исполнение и завершиться раньше начала исполнения ФВ b), отражающих только информационные зависимости между ФВ, тогда под прямым управлением подразумеваются ограничения управления, не попавшие в потоковые, связанные обычно с распределением ресурсов и оптимизацией выполнения программ [1].

Базовый алгоритм исполнения runtime-системы LuNA реализует в динамике потоковое управление, что является причиной плохой производительности из-за большого количества накладных расходов. В то же время нельзя полностью исключить динамическое управление вычислениями, так как от параллельной программы требуются некоторая гибкость, например, поддержка миграции фрагментов для обеспечения динамической балансировки загрузки и настройки на все доступные ресурсы вычислителя.

Можно выделить два подхода к решению проблемы накладных расходов:

— сузить предметную область, например, PaRSEC [4] — runtime-система, ориентированная на исполнение алгоритмов линейной алгебры для плотных матриц, представленных в библиотеке DPLASMA [5], в распределенных гетерогенных системах;

— ввести в язык специализированные операторы, задающие прямое управление, например, для LISP [6] — язык функционального программирования, в нем используются операторы `for` и `while` из традиционного программирования.

В системе LuNA используются оба подхода. Предметная область ограничена алгоритмами численного моделирования, и ведется работа по разработке средств задания прямого управления для системы LuNA. В статье [7] были представлены результаты по использованию предикатной сети Петри в качестве средства задания прямого управления. В работе будет сделан акцент на разработку средств прямого управления для циклических конструкций в ФП. Для этого язык LuNA был расширен необходимыми языковыми конструкциями, и была реализована их программная поддержка в системе LuNA в виде программного модуля. Его задачей является исполнение некоторого подмножества ФВ, входящих в ФП, под прямым управлением без необходимости дополнительных проверок ФВ на готовность, с заранее заданным распределением ресурсов, а значит, более эффективным с точки зрения накладных расходов на управление вычислениями и времени исполнения ФП.

Обзор существующих решений. Прямое управление в той или иной мере используется для повышения производительности исполнения параллельных программ, которые имеют декларативное представление алгоритма в языках и системах программирования.

В системах параллельного программирования SMP Superscalar [8], пакете ProActive Parallel Suite [9] задать прямое управление можно с помощью приоритетов. Изменение приоритета той или иной операции влияет на порядок срабатывания операций. Но этот способ, несмотря на свою универсальность, не позволяет описывать сложное поведение, и

с увеличением размера задачи теряется читабельность и увеличивается вероятность ошибок управления. В функциональных языках параллельного программирования, например, Haskell [10], Sisal [11], для повышения производительности исполнения программ отдельно выделены языковые конструкции для обозначения параллельного цикла (все итерации могут исполняться независимо) и последовательного цикла (итерации цикла исполняются последовательно, друг за другом). Это позволяет без дополнительных накладных расходов определить порядок срабатывания множества операций, описанных конструкцией `for`.

Также, кроме прямого управления, связанного с заданием порядка исполнения вычислений, есть конструкции, задающие прямое управление для коммуникаций, например, язык нотаций Charisma [12] в системе параллельного программирования Charm++ [13].

Так как алгоритмы численного моделирования в большинстве своем описываются с помощью циклов типа `for` и `while`, то использование конструкций для обозначения параллельного и последовательного циклов представляется перспективным способом задания прямого управления в ФП.

Описание цели работы. В системе фрагментированного программирования LuNA фрагментированная программа представляется в виде множества фрагментов данных (ФД) единственного присваивания и фрагментов вычислений (ФВ) единственного срабатывания, для каждого ФВ задается множество входных и выходных ФД.

Интерпретация ФП

ФД — это некоторый буфер памяти заданного размера, которому в ходе вычислений присваивается некоторое значение.

Исполнение ФВ состоит в вычислении значений его выходных ФД из значений входных ФД. ФВ вычисляется некоторой функцией без побочных эффектов. Исполнение ФП заключается в исполнении всех его ФВ. Порядок исполнения ФВ определяется информационными зависимостями между ФВ.

Базовый алгоритм исполнения ФП в LuNA:

- 1) Из множества невыполненных ФВ выбирается множество готовых к исполнению ФВ (будем говорить, что ФВ готов к исполнению, если значения всех его входных ФД вычислены).
- 2) Из множества готовых к исполнению ФВ выбирается один, несколько, все или ни одного ФВ и назначается на исполнение.
- 3) По мере исполнения ФВ их выходные ФД становятся вычисленными. При этом у некоторых ФВ все входные ФД оказываются вычисленными, и такие ФВ переходят во множество готовых к исполнению ФВ.
- 4) Исполнившиеся ФВ исключаются из множества невыполненных ФВ.
- 5) Пункты 2–3 повторяются, пока множество готовых к исполнению ФВ не окажется пустым и ни один ФВ не исполняется.

В базовом алгоритме исполнения ФП можно выделить следующие виды накладных расходов:

- накладные расходы на организацию вычисления внутри узла мультикомпьютера (выбор множества готовых к исполнению ФВ среди всех ФВ и т. д.);
- накладные расходы, связанные с динамическим распределением ресурсов;
- накладные расходы, связанные с организацией вычислений между узлами мультикомпьютера.

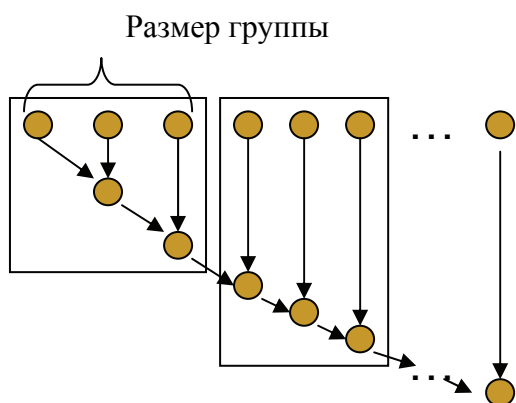


Рис. 1. Схема информационных зависимостей между ФВ во ФП для последовательного алгоритма редуцирования данных

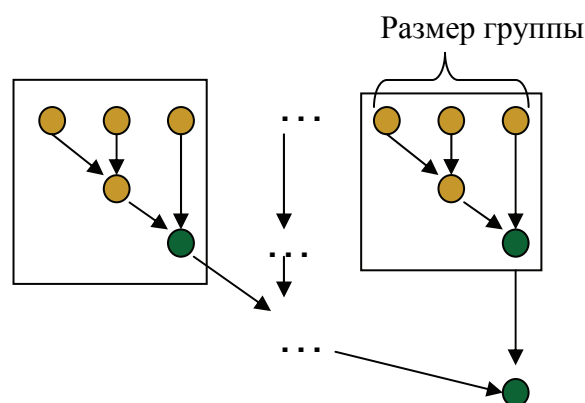


Рис. 2. Схема информационных зависимостей между ФВ во ФП для параллельного алгоритма редуцирования данных

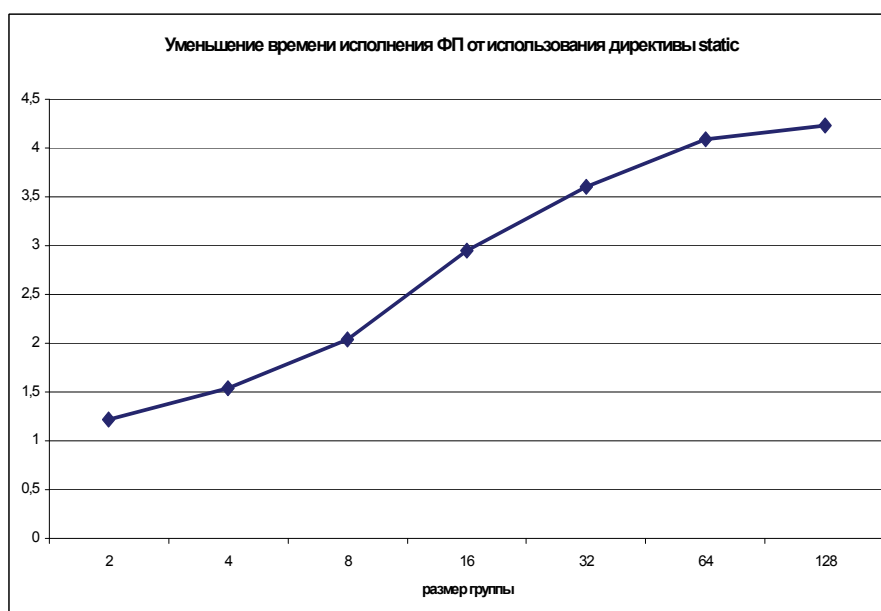


Рис. 3. Уменьшение времени исполнения ФП от использования директивы `static` в зависимости от размера группы (размер ФД $10 \times 10 \times 10$, количество редуций — 200)

В общем случае runtime-система LuNA, используя базовый алгоритм исполнения ФП, большую часть решений принимает в динамике, что влечет большое количество накладных расходов. Для повышения эффективности исполнения ФП целесообразно сократить количество решений, принимаемых в динамике, а именно: убрать динамическое управление вычислениями там, где это возможно, оставив некоторую степень недетерминизма для возможности параллельного исполнения на мультикомпьютере и обеспечения динамических свойств параллельного исполнения программы.

Целью было разработать средства задания прямого управления во ФП, направленные на уменьшение накладных расходов на организацию вычислений внутри узла, реализовать их в виде программного модуля и включить этот модуль в систему фрагментированного программирования LuNA.

Предлагаемое решение. В качестве решения предлагается задавать прямое управление в виде *монолитного ФВ*. *Монолитным ФВ* будем называть такой ФВ, который с точки зрения runtime-системы атомарный, но внутри себя включает исполнение нескольких ФВ, причем порядок исполнения ФВ статически фиксирован и не противоречит потоковым информационным зависимостям, но при этом лишен недетерминизма исполнения.

Средство прямого управления в виде монолитного ФВ целесообразно использовать для тех подмножеств ФВ, информационная зависимость между которыми представлена как цепь. В ФП это будет, например, эквивалентно `for`-описателю, итерации которого исполняются последовательно. В отличие от базового алгоритма runtime-системы LuNA, который принимал бы решение о выборе следующего на исполнение ФВ на каждой итерации, использование прямого управления позволит принимать решение только один раз на стадии компиляции.

Описание реализации. Для того чтобы некоторое выбранное подмножество ФВ исполнялось под прямым управлением, язык LuNA был расширен аннотациями (директивами) для `for`-описателей и подпрограмм. Для задания прямого управления в виде монолитного ФВ используется директива *static*. На уровне компилятора был разработан модуль на языке python, который на стадии компиляции для выбранного подмножества ФВ генерирует монолитный ФВ как функцию без побочных эффектов на языке C. Разработанная реализация обладает ограничением на количество входных и выходных ФД монолитного ФВ, максимальный размер — 250. Это связано с ограничением компилятора языка C на количество аргументов функции.

Исследование производительности исполнения ФП. Для исследования того, насколько применение предлагаемых средств позволяет уменьшить накладные расходы, в качестве приложения был выбран алгоритм редуцирования данных, а именно две его версии: последовательная (рис. 1) и параллельная (рис. 2). Этот алгоритм был взят как типичный случай, в котором следует использовать разработанное средство прямого управления и на котором оно даст хороший выигрыш по производительности исполнения ФП.

Ограничения на количество параметров монолитного ФВ влияют на размер подмножества ФВ, которое можно представить в виде монолитного ФВ. Поэтому кроме степени фрагментации алгоритма, изменение которой значительно влияет на производительность исполнения ФП, появляется еще один изменяемый параметр — размер подмножества ФВ, который будем называть размером группы.

Тестирование в общей памяти. Параметры вычислителя: 6-ядерный компьютер (Intel Xeon CPU X5600 2,8 Ghz). Количество потоков — 4.

На рис. 3 представлен график зависимости уменьшения времени исполнения ФП при использовании директивы *static* от размера группы по сравнению с базовым алгоритмом исполнения runtime-системы LuNA. Видно, что с увеличением размера группы выигрыш от использования прямого управления увеличивается.

Для того чтобы понять, за счет чего это происходит, было выполнено профилирование исполнения ФП профилировщиком LuNA, который фиксирует события и время их происхождения. На основе полученной информации время исполнения ФП можно разделить на следующие категории:

CF — время чистых вычислений;

FOR — время на организацию вычислений внутри `for`-описателя;

calc_sum — время на организацию вычислений внутри подпрограммы, реализующей алгоритм редуцирования данных;

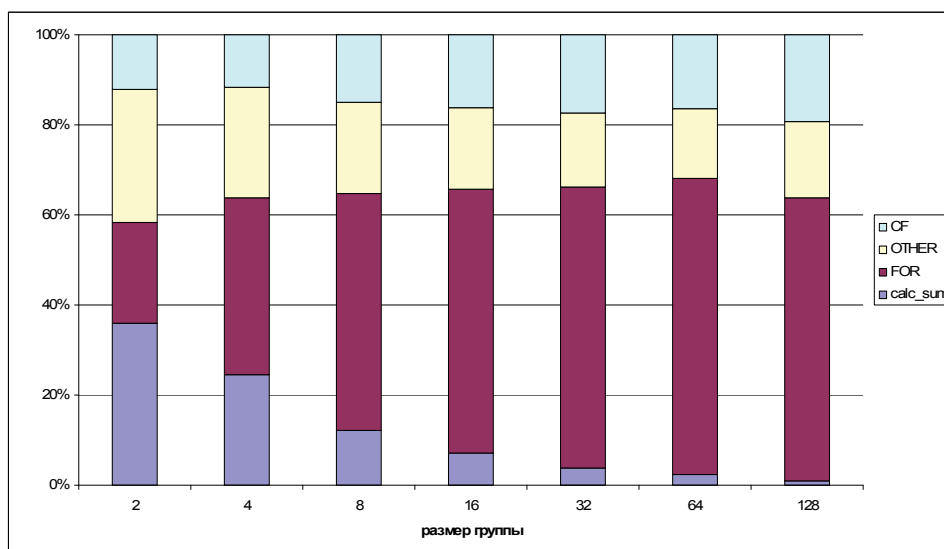


Рис. 4. Процентное соотношение времени исполнения ФП базовым алгоритмом runtime-системы LuNA в зависимости от размера группы

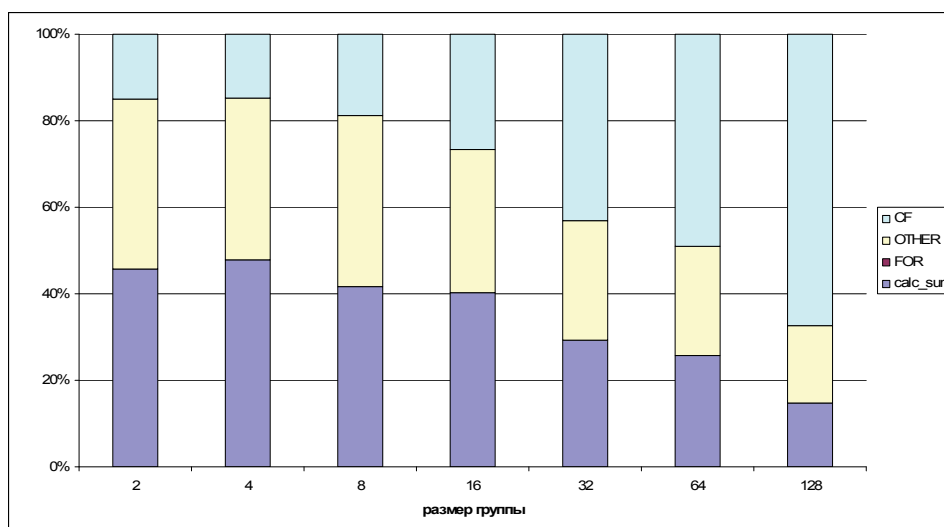


Рис. 5. Процентное соотношение времени исполнения ФП с использованием директивы *static* в зависимости от размера группы

OTHER — другие накладные расходы.

На рис. 4 и рис. 5 представлено процентное соотношение времени исполнения ФП по категориям в зависимости от размера группы. Видно, что в базовом алгоритме runtime-системы с ростом размера группы увеличиваются накладные расходы на организацию вычислений внутри for-описателя, и чем больше этих накладных расходов, тем больше будет выигрыш от использования директивы *static*.

Рассмотрим теперь изменение уменьшения времени исполнения ФП от использования директивы *static* в зависимости от размера ФД (рис. 6). С ростом размера ФД выигрыш уменьшается. Это объясняется тем, что чем больше размер ФД, тем больше его время обработки, и, начиная с некоторого значения, доля накладных расходов на организацию

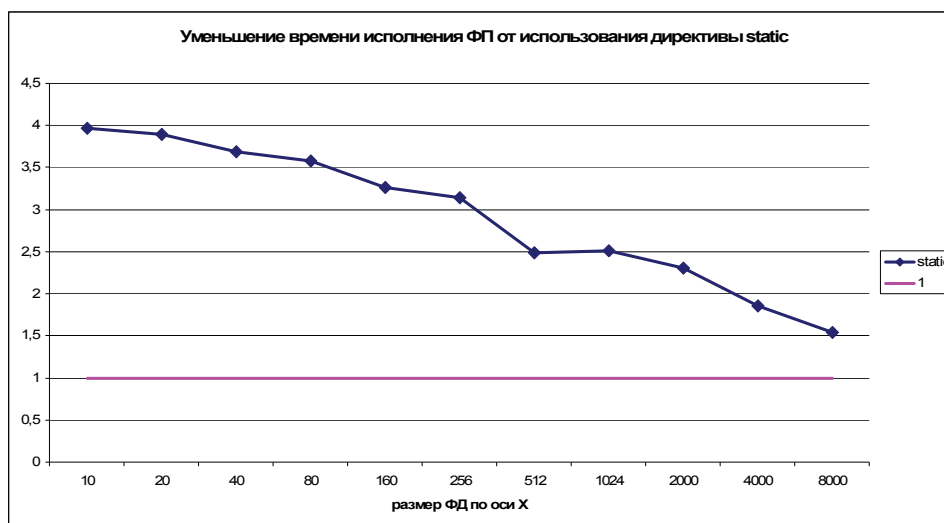


Рис. 6. Уменьшение времени исполнения ФП от использования директивы *static* в зависимости от размера ФД по оси X (количество редуций — 200, размер группы — 64, размер ФД по осям Y и Z — 10)

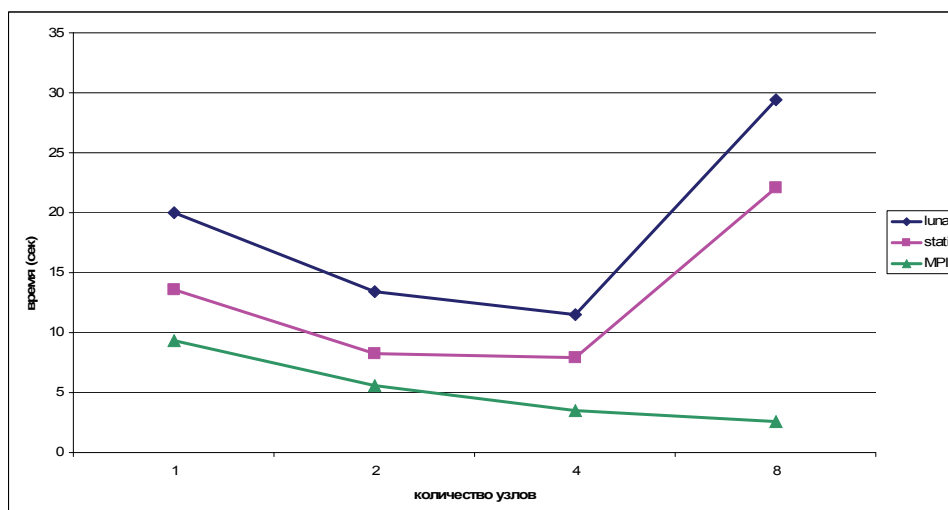


Рис. 7. Время исполнения ФП параллельного алгоритма редуцирования данных в зависимости от количества узлов (размер ФД 40x40x40, количество ФВ 10000, размер группы — 250, количество редуций — 10, количество потоков — 4)

его вычисления значительно меньше доли чистых вычислений, поэтому выигрыш от использования прямого управления нивелируется.

Для параллельного алгоритма редуцирования данных были получены аналогичные зависимости от размера группы и размера ФД, особенностью является то, что нужно учитывать влияние параллелизма на долю выигрыша от использования директивы *static*.

Из тестирования в общей памяти можно сделать вывод, что разработанное средство прямого управления в виде монолитного ФВ позволяет уменьшить накладные расходы на организацию вычислений внутри узла за счет уменьшения накладных расходов на старт очередного ФВ. Размер выигрыша зависит от доли накладных расходов по сравнению с долей чистых вычислений: чем она существенней, тем больше выигрыш.


Тестирование в распределенной памяти. Проводилось на кластере НГУ [14]. Цель тестирования заключалась в определении, насколько на производительность в распределенной памяти влияет уменьшение накладных расходов внутри узла [15]. Для этого исполнение ФП параллельного алгоритма редуцирования данных в базовом алгоритме исполнения ФП runtime-системы LuNA и с использованием директивы *static* сравнивалось с аналогичной реализацией в MPI в зависимости от количества узлов вычислителя.

Из рис. 7 видно, что с ростом количества узлов разница между исполнением ФП в системе LuNA и аналогичной реализацией в MPI возрастает. При этом использование директивы *static* позволяет уменьшить разрыв лишь на некоторую константу, что говорит о том, что в распределенной памяти более существенны накладные расходы, связанные с организацией вычислений между узлами мультимпьютера.

Заключение. В работе представлено разработанное средство прямого управления в виде монолитного ФВ, которое позволяет уменьшить накладные расходы внутри узла по сравнению с базовым алгоритмом runtime-системы LuNA. Преимущества предлагаемого решения были показаны на примере задачи редуцирования данных. Тестирование в общей памяти показало, что разработанное средство прямого управления позволяет уменьшить накладные расходы на организацию вычислений внутри узла мультимпьютера за счет уменьшения накладных расходов на старт очередного ФВ. Размер выигрыша зависит от доли накладных расходов по сравнению с долей чистых вычислений: чем она существенней, тем больше выигрыш. Тестирование в распределенной памяти показывает, что доля накладных расходов на организацию вычислений внутри узла оказывает не столь большое влияние на время исполнения ФП по сравнению с накладными расходами на организацию вычислений между узлами мультимпьютера. Поэтому в дальнейшей работе предполагается разработка средств прямого управления для организации вычислений между узлами.

Список литературы

1. Малышкин В. Э., Корнеев В. Д. Параллельное программирование мультимпьютеров. В сер. „Учебники НГТУ“. Новосибирск: изд-во НГТУ, 2006.
2. MALYSHKIN V. E., PEREPELKIN V. A., TKACHEVA A. A. Control Flow Usage to Improve Performance of Fragmented Programs Execution. PaCT 2015. LNCS, vol. 9251, P. 86–90. Springer, CityplaceHeidelberg (2015).
3. MALYSHKIN V. E., PEREPELKIN V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem // In the Proceedings of the 11-th Conference on Parallel Computing Technologies, LNCS. Springer, 2011. V. 6873. P. 53–61.
4. BOSILCA G., BOUTEILLER A., DANALIS A., HERAULT T., LEMARINIER P., DONGARRA J. DAGuE: A Generic Distributed DAG Engine for High Performance Computing // Proceedings of the Workshops of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2011 Workshops), IEEE, Anchorage, Alaska, USA, 16-20 May, 2011. P. 1151–1158.
5. BOSILCA G., BOUTEILLER A., DANALIS A., ET AL. Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA // Proceedings of the Workshops of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2011 Workshops), IEEE, Anchorage, Alaska, USA, 16-20 May, 2011. P. 1432–1441.
6. SEIBEL P. Practical Common LISP, APRESS, 2005.
7. ТКАЧЕВА А. А. Средства задания прямого управления во фрагментированных программах и их применение на примере явного метода решения уравнения Пуассона // Труды конференции молодых ученых. Новосибирск, 2014. С. 122–133.

8. CityplaceBarcelona Supercomputing Center. SMP Superscalar (SMPSS) User's Manual, Version 2.2. [Electron. res.]. <http://www.bsc.es/media/3576.pdf> (2008).
9. CAROMEL D., LEYTON M. ProActive Parallel Suite: from active objects-skeletons-components to environment and deployment. // Euro-Par 2008 Workshops — Parallel processing. 2008. P. 423–437.
10. COUTTS D., LOEH A. Deterministic parallel programming with Haskell // Comput. Sci. country-regionplaceEng. 2012. N 14 (6). P. 36–43.
11. GAUDIOT J.-L., DEBONI T., FEO J., ET AL. The Sisal project: real world function programming // LNCS, Springer, 2001. V. 1808. P. 84–72.
12. HUANG CH., LAXMIKANT V., CHARISMA K. Orchestrating Migratable Parallel Objects // Proceedings of the 16th International Symposium on High Performance Distributed Computing (HPDC). 2007. P. 75–84.
13. PHIL MILLER. Productive parallel programming with Charm++ // Proceedings of the Symposium on High Performance Computing. 2015. P. 241–242.
14. Кластер новосибирского национального исследовательского государственного университета. [Электрон. рес.]. <http://www.nusc.ru/>
15. Малышкин В. Э. Проблемы параллельной реализации крупномасштабных численных моделей на вычислительных системах экзафлопсной производительности // Проблемы  форматики № 3 (28) 2015. С. 71–82.

*Ткачева Анастасия Александровна — младш. науч. сотрудник
Института вычислительной математики и математической геофизики СО РАН;
ассистент кафедры Параллельных вычислений факультета информационных технологий
Новосибирского национального исследовательского государственного университета;
Тел.: (383) 330-89-94,
e-mail: tkacheva@ssd.sscs.ru.*

Дата поступления — 29.04.2015