A MULTI-AGENT APPROACH TO IMPROVE EXECUTION EFFICIENCY OF FRAGMENTED PROGRAMS IN LUNA SYSTEM

V. E. Malyshkin, V. A. Perepelkin

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

DOI: 10.24412/2073-0667-2023-3-55-67
EDN: LYEAKS

Development of efficient numerical parallel programs is an complex and laborious problem which impedes application of supercomputers to perform numerical simulations. Parallel programs construction languages and systems are of help by providing to a user a high-level language to describe a desired parallel program and generating the program automatically. The systems do not only reduce complexity of parallel programs development, but also provide static and/or dynamic adaptation of the program execution to particular hardware and execution conditions (e.g. workload balancing). This implies that a high-level parallel program specification may be transformed into a parallel program in diverse ways, many of which may be optimal for particular simulation. Such diversity can either be resolved statically or dynamically, the former causing less run-time overhead, while the latter preserves the ability to dynamically tune parallel program execution. Finding a good trade-off between static and dynamic decision making is a challenging problem in system parallel programming, which also depends on the computational model on which the system is based.

LuNA system [1] is a system for automatic construction of numerical parallel programs, which is being developed in ICMMG SB RAS. It is based on the theory of parallel programs synthesis based on the computational models [2] and follows the active knowledge concept [3]. Its input language LuNA comprises means to describe pieces of data and computations, called data and computational fragments correspondingly (DFs and CFs). DFs are immutable coarse-grained data objects, while CFs are side-effect free sequential procedure calls on particular DFs as input or output arguments. Such dataflow computational model allows the system to automatically execute CFs on a distributed memory machine either by generating a conventional distributed program which performs procedures invocation according to the information dependencies, or by dynamically interpreting the program on a multicomputer. The former approach lacks dynamic flexibility while the latter approach causes significant overhead.

In order to achieve a trade-off between those two options a multi-agent approach is suggested. For that a multi-agent system is defined, where each agent implements a single CF in a distributed environment, which allows an agent to consume and produce DFs and to create new agents. Each agent is controlled by an imperative sequential program in a conventional language (C++ to be specific).

This work was carried out under state contract with ICMMG SB RAS 0251-2022-0005.

The environment system supports agents with basic operations, such as storing a DF, migrating a CF to another computing node, etc. Usage of the C++ language to formulate agent programs is beneficial in sense that it is compiled into a highly efficient machine code using conventional C++ compiler. Such approach allows to make decisions on how to execute LuNA programs both statically (by generating corresponding behavior into agents' programs) and dynamically (by adding dynamic properties support modules into run-time environment and making agents' behavior depend on that modules). For example, distribution of agents to computing nodes may either be static (in this case each agent will have a particular computing node to migrate to for execution) or dynamic (in this case each agent will request a dynamic balancing module to assign a node to migrate to).

The proposed approach showed to be much more efficient than a naive distributed interpretation approach which is confirmed by works [4–5]. The fragmented structure of the program is preserved in run-time, which makes it possible to provide dynamic properties of LuNA-program execution. The efficiency achieved in practical cases appeared to be comparable with that of manual low-level programming, which makes LuNA system usable for automating construction of real numerical parallel programs at least in some application domains. The proposed approach takes advantage of well-developed tools for conventional sequential compilation to reduce the run-time overhead.

Key words: Fragmented programming, LuNA system, parallel programs construction automation, high performance computing, multi-agents approach, partial evaluation.

References

1. Malyshkin, V. E., Perepelkin, V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. // Malyshkin, V. (eds) Parallel Computing Technologies. PaCT 2011. Lecture Notes in Computer Science, 2011. V. 6873. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-23178-0_5.

2. Sintez parallelnykh programm i sistem na vychislitelnykh modelyah / Valkovsky V. A., Malyshkin V. E. Novosibirsk: Nauka. 1988. 126 p. (In Russian)

3. Malyshkin V. E. Tekhnologiya fragmentirovannogo programmirovaniya // Vestnik YuUrGU. Seriya: Vychislitelnaya matematika i informatika. 2023. N 46(305). (In Russian)

4. Malyshkin V. E. Active Knowledge, LuNA and Literacy for Oncoming Centuries // Essays Dedicated to Pierpaolo Degano on Programming Languages with Applications to Biology and Security. 2015. V. 9465. Springer-Verlag, Berlin, Heidelberg. P. 292–303.

5. Lebedev D. V., Perepelkin V. A. Chislennoe reshenie odnomernody kraevoy zadachi filtracii zhidkosti dlya sistemy "neft-voda" i ee realizaciya v sisteme fragmentirovannogo programmirovaniya LuNA // Vestnik Kazahskogo nacionalnogo universiteta im. al-Farabi, seriya "Matematika, mehanika, invormatica". N 3 (82). 2014. S. 64–73 (In Russian).

6. Akhmed-Zaki D. Zh., Lebedev D. V., Perepelkin V. A. Implementation of a Three-Phase Fluid Flow ("Oil-Water-Gas") Numerical Model in the LuNA Fragmented Programming System // In Proc 13th International Conference on Parallel Computing Technologies. LNCS 9251. Springer, 2015. P. 489-497. DOI: 10.1007/978-3-319-21909-7_47.

7. Lebedev D. V., Perepelkin V. A. Reshenie trekhmernogo modelnogo uravneniya teploprovodnosti v sisteme fragmentirovannogo programmirovaniya LuNA // Parallelnye vychislitelnye tekhnologii (PaVT'2016): trydy mezhdunarodnoy nauchnoy konferencii (28 marta – 1 aprelya 2016 g., g. Arkhangelsk). Chelyabinsk: Izdatelskiy centr YuUrGU, 2016. S. 784 (In Russian).

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

9. Malyshkin V. E., Perepelkin V. A., Schukin G. A. Distributed Algorithm of Data Allocation in the Fragmented Programming System LuNA // In Proc 13th International Conference on Parallel Computing Technologies. LNCS 9251. Springer, 2015. P. 80–85. DOI: 10.1007/978-3-319-21909-7_8.

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

11. Akhmed-Zaki, D., Lebedev, D., Malyshkin, V., Perepelkin, V. Automated construction of high performance distributed programs in LuNA system // 15th International Conference on Parallel Computing Technologies, PaCT 2019; Almaty; Kazakhstan. LNCS 11657. Springer, 2019. P. 3–9. DOI: 10.1007/978-3-030-25636-4_1.

12. Kudryavtsev A. A., Malyshkin V. E., Nushtayev Yu. Yu., Perepelkin V. A., Spirin V. A. Effektivnaya fragmentirovannaya realizatsiya kraevoy zadachi filtratsii dvuhfaznoy zhidkosti // Problemy Informatiki. 2023. N 2. S. 45–73. DOI: 10.24412/2073-0667-2023-2-45-73. (In Russian)

13. Belyaev, N., Kireev, S. LuNA-ICLU Compiler for Automated Generation of Iterative Fragmented Programs. // Malyshkin, V. (eds) Parallel Computing Technologies. PaCT 2019. Lecture Notes in Computer Science, V. 11657. Springer, Cham. DOI: 10.1007/978-3-030-25636-4_2.

МУЛЬТИАГЕНТНЫЙ ПОДХОД К ПОВЫШЕНИЮ ЭФФЕКТИВНОСТИ ИСПОЛНЕНИЯ ФРАГМЕНТИРОВАННЫХ ПРОГРАММ В СИСТЕМЕ LUNA

В. Э. Малышкин, В. А. Перепелкин
Институт вычислительной математики и математической геофизики СО РАН, 630090, Новосибирск, Россия,
Новосибирский национальный исследовательский государственный университет, 630090, Новосибирск, Россия,
Новосибирский государственный технический университет, 630073, Новосибирск, Россия

УДК 004.4'242
DOI: 10.24412/2073-0667-2023-3-55-67
EDN: LYEAKS

Применение систем параллельного программирования и систем автоматического конструирования параллельных программ предоставляет возможности к статической и/или динамической адаптации исполнения параллельной программы к особенностям вычислителя и хода вычислений, но при этом возникает проблема снижения накладных расходов, возникающих из-за работы исполнительной системы (динамических системных алгоритмов). Существенное снижение таких накладных расходов возможно за счет переноса как можно большего количества работы по конструированию и адаптации параллельной программы на этап трансляции, но при этом важно сохранять возможность динамической настройки исполнения параллельной программы в части, где настройка должна по существу выполняться динамически (например, для осуществления динамической балансировки нагрузки на вычислительные узлы). В работе предлагается технологичный подход к переносу существенного объема работ по конструированию и исполнению параллельных программ в системе LuNA, который сохраняет возможность обеспечения динамических свойств исполнения программы. Предложенный подход позволил существенно снизить накладные расходы на исполнение LuNA-программ по сравнению с традиционным подходом распределенной динамической интерпретации LuNA-программ.

Ключевые слова: фрагментированное программирование, система LuNA, автоматизация конструирования параллельных программ, высокопроизводительные вычисления, мультиагентный подход, частичные вычисления.

Введение. Применение суперкомпьютеров для научного численного моделирования связано с проблемой сложности и трудоемкости разработки численных параллельных программ. Для обеспечения необходимой эффективности требуются знания и навыки в области системного параллельного программирования (тут и далее эффективность понимается с точки зрения времени выполнения программы, расхода памяти, нагрузки на сеть и т. п.). Автоматизация параллельного программирования позволяет существенно снижать остроту этой проблемы за счет того, что часть работы перекладывается на систему автоматического конструирования параллельных программ. Для этого пользователю предоставляются более высокоуровневые декларативные средства (язык) описания требуемой

Исследование выполнено в рамках государственного задания ИВМиМГ СО РАН 0251-2022-0005.

параллельной программы. Под высокоуровневостью в данном случае понимается то, что пользователю не приходится уделять внимание деталям программы, несущественным с его точки зрения. Например, пользователю может быть безразлично, как именно устроена программа внутри, до тех пор, пока она реализует нужный ему алгоритм и работает достаточно эффективно. По высокоуровневому описанию система автоматически конструирует требуемую программу. Работа пользователя, таким образом, сводится к менее сложной и трудоемкой работе — построению высокоуровневой спецификации. Это аналогично тому, как в последовательном программировании задача получения машинного кода сводится к задаче построения высокоуровневой спецификации — программе на языке высокого уровня, а машинный код генерируется компилятором автоматически.

По заданной высокоуровневой спецификации, вообще говоря, может быть сгенерировано множество параллельных программ, каждая из которых будет этой спецификации соответствовать, но программы будут обладать разной эффективностью. Кроме того, в зависимости от конфигурации вычислителя и входных данных программы относительная эффективность параллельных программ будет отличаться. Например, одна программа может работать быстрее на небольших вычислителях, другая — на мультикомпьютерах с медленной сетью, а третья — для экспериментов с выраженным дисбалансом вычислительной нагрузки. Выбор достаточно эффективной параллельной программы из множества соответствующих спецификации — алгоритмически труднорешаемая задача системного параллельного программирования, которая успешно решается системами автоматизации программирования лишь для ограниченного круга приложений.

Множественность в выборе варианта реализации высокоуровневых конструкций входного языка может разрешаться статически, динамически или комбинированно. Статический вариант — это конструирование параллельной программы, при котором невозможно обеспечение динамических свойств. Динамический вариант — это интерпретация высокоуровневой спецификации во время исполнения, когда выбор варианта реализации осуществляется интерпретатором в зависимости от текущих обстоятельств исполнения. Его недостатком являются накладные расходы на работу интерпретатора. Комбинированный вариант является предпочтительным, т. к. позволяет часть решений принимать статически, оставляя на время выполнения принятие тех решений, которые по существу должны приниматься динамически для обеспечения динамических свойств параллельной программы.

На практике, тем не менее, далеко не всегда удается хорошо разделить процесс разрешения многовариантности реализации высокоуровневой программы на статическую и динамическую части. Часто конструируемая программа не обладает достаточной адаптивностью в динамике за счет того, что некоторые решения о способе исполнения были приняты статически. И наоборот, иногда выбор варианта реализации осуществляется динамически ценой накладных расходов на работу интерпретатора в случаях, когда удовлетворительное решение могло быть принято статически. Поиск удачных технологичных подходов, позволяющих разделять принятие решений между транслятором и исполнительной системой, является актуальной проблемой системного параллельного программирования. При этом важную роль играет модель вычислений, лежащая в основе той или иной системы автоматизации программирования, т. к. от нее в значительной степени зависит не только то, насколько возможно статически или динамически разрешать многовариантность реализации высокоуровневой программы, но и то, что в принципе возможно автоматизировать в параллельном программировании на основе этой модели. Так, например, модель вы-

числений языка C++ не позволяет автоматизировать сборку мусора ввиду возможности выполнения арифметических операций над указателями, в то время как модель вычислений языка Java допускает автоматизацию сборки мусора ввиду отсутствия арифметики над ссылками.

Система LuNA [1] является примером системы, на вход которой поступает декларативная программа, которая преобразуется в императивную частично статически, частично — динамически (т. е. исполняется в режиме полуинтерпретации) LuNA-программа представляет собой описание множества информационно зависимых задач, выполнение которых осуществляется в распределенной памяти под управлением исполнительной системы, которую можно рассматривать как совокупность распределенного портфеля задач и распределенной базы данных. Задачей исполнительной системы является управление выполнением задач в соответствии с их декларативным описанием в виде LuNA-программы. При этом исполнительная система обеспечивает динамические свойства исполнения программы, такие как динамическая балансировка нагрузки на узлы мультикомпьютера.

В системе LuNA также стоит вопрос переноса как можно большего количества работы из исполнительной системы в транслятор, если это не мешает обеспечению динамических свойств. В статье предлагается подход к формированию исполняемого представления LuNA-программы, обеспечивающий перенос большей части системной работы на этап трансляции, сохраняя при этом возможность обеспечения динамических свойств.

Остальная часть статьи организована следующим образом. В разделе 1 приводятся необходимые термины и определения. В разделе 2 излагается предлагаемый подход. Итоги работы подводятся в заключении.

1. Система LuNA. Система LuNA базируется на теории синтеза параллельных программ на вычислительных моделях [2], основана на принципах, изложенных в [3] и следует концепции активных знаний [4]. Распределенный вычислительный процесс (процесс исполнения параллельной программы) в системе LuNA рассматривается как выполнение частично упорядоченного множества информационно зависимых задач, где каждая задача — это выполнение некоторой последовательной процедуры без побочных эффектов. Каждая такая процедура имеет набор входных и выходных параметров и вычисляет значения выходных параметров из значений входных. Если выходной аргумент одной задачи является входным для другой, то такие две задачи будут информационно зависимыми. В качестве значения параметра допускается любой сериализуемый объект данных. Задачи будем называть фрагментами вычислений (ФВ), а их аргументы — фрагментами данных (ФД).

Программа на языке LuNA есть параметрическое описание рекурсивно перечислимого множества ФВ и ФД. Под параметричностью в данном случае понимается то, что это множество, вообще говоря, может зависеть от значений ФД. Например, если имеет место итерационный процесс, количество итераций в котором определяется динамически в зависимости от входных данных, то и количество ФВ и ФД будет зависеть от данных. LuNA-программа состоит из конечного множества операторов, каждый из которых может описывать единичный ФВ или их множество. Множество ФД определяется косвенно через именование входных и выходных аргументов ФВ. Проиллюстрируем сказанное примером простой LuNA-программы (листинг 1).

Листинг 1. Пример LuNA-программы.

```
01: import init(name x);
02: import f(value prev, name next);
03: import g(value x, name y);
04: import print(value x);
05:
06: sub main() {
07:   df x, y, N;
08:   cf a: init(x[0]);
09:   cf e: print(x[N]);
10:   while x[i] > 0, i = 0 .. out N
11:     cf b[i]: f(x[i], x[i+1]);
12:   for i = 0 .. N-1 {
13:     cf c[i]: g(x[i], y[N-i-1]);
14:     cf d[i]: print(y[i]);
15:   }
16: }
```

В этом примере описывается ФВ `a`, который инициализирует ФД `x[0]` с помощью процедуры `init` (строка 08). Далее (строки 10–11) с помощью массового оператора `while` описывается множество ФВ `b[i]`, каждый из которых вычисляет значение очередного `x[i+1]` из `x[i]` с помощью процедуры `f`. Количество ФВ определяется динамически, в зависимости от условия оператора while (`x[i]>0`). Первое значение счетчика цикла `i`, для которого условие окажется ложным, считается значением ФД `N`. Таким образом окажется вычисленным множество ФД `x[i]` для `i` от 0 до `N`. Ниже (строки 12–15) описывается множество ФВ `c[i]` и `d[i]` с помощью массового оператора `for`. ФВ `c[i]` вычисляет значение ФД `y[N-i-1]` из `x[i]` с помощью процедуры `g`, а ФВ `d[i]` выводит значение ФД `y[i]` на экран с помощью процедуры `print`. В строке 09 описан ФВ `e`, который с помощью той же процедуры выводит на экран значение ФД `x[N]`.

Отметим, что вне зависимости от порядка следования операторов в программе исполнение ФВ будет осуществляться в порядке, определяемым информационными зависимостями. Так, в частности, ФВ `e` будет выполнен после того, как будет установлено значение ФД `N`, т. е. после того, как для некоторого `i` окажется ложным условие цикла `while`. При этом, вообще говоря, неизвестно, какой из ФВ выполнится раньше: `e` или `c[i]` (для любых значений `i`). В частности, порядок выполнения ФВ `c[i]` друг относительно друга не определен и может быть любым, включая одновременное выполнение.

Также отметим, что из листинга не видно, какого рода значения принимают ФД и какого рода вычисления над ними совершаются. Эта информация скрыта от системы и заложена в процедуры, используемые для описания ФВ (сигнатуры этих процедур описываются в строках 01–04). В частности, системе неизвестны и типы значений, присваеваемых ФД. Это могут быть как отдельные числа, так и агрегированные значения (фрагменты численных сеток, матрицы, массивы чисел и т. п.). Смысл значений ФД известен в тех процедурах, в которые эти значения подаются. Системе эта информация не нужна, для того чтобы организовать и осуществить вычислительный процесс.

Распределенное исполнение LuNA-программы может быть организовано в форме распределенного интерпретатора. Пусть имеется мультикомпьютер, на каждом узле кото-

рого находится экземпляр интерпретатора, в который загружена исполняемая LuNA-программа. Каждый из экземпляров интерпретатора может хранить на своем узле значения некоторых (или всех) из вычисленных ФД (т. о., можно говорить о распределенной базе данных, хранящей ФД). Также экземпляр может хранить на своем узле задачи — команды, определяемые операторами LuNA-программы (кроме операторов описания ФД, т. е. операторы `cf`, `for` или `while`). В начальный момент времени для каждого оператора подпрограммы `main` формируется такая задача и помещается на один из узлов мультикомпьютера. Если считать «структурированными» фрагментами вычислений задачи, описываемые операторами `for` и `while`, то понятие задачи можно считать тождественным понятию ФВ. С этой оговоркой далее будем говорить о ФВ.

Далее интерпретатор работает по следующей схеме, общей для всех ФВ. Каждый ФВ должен быть выполнен. Выполнение ФВ требует готовности значений некоторых ФД (входных для данного ФВ), а результатом его выполнения является вычисление значений некоторых других ФД (входных) и/или порождение новых ФВ. Так, например, ФВ, соответствующий оператору на строке 08, не имеет входных ФД, а выходным ФД является `x[0]`, а (структурированный) ФВ, задаваемый оператором `for` на строке 12, на вход требует ФД `N` (для определения границ диапазона значений `i`), а на выход производит множество новых ФВ в количестве $2 \times$ `N` штук, по `N` штук, определяемых операторами на строках 13 и 14 при различных значениях `i` в диапазоне значений от 0 до `N-1`.

При исполнении LuNA-программы интерпретатор принимает решения о том, на каком вычислительном узле выполнить ФВ. Это решение должно приниматься с учетом текущей загруженности узлов, расположения входных ФД на узлах мультикомпьютера, того, на каких узлах будут храниться и потребляться выходные ФД и т. п. Также интерпретатор принимает решение о том, на каких узлах мультикомпьютера хранить копии ФД, в какой момент передавать их по сети и когда уничтожать их за ненадобностью (сборка мусора). В рамках имеющихся информационных зависимостей интерпретатор принимает решение о том, в каком порядке выполнять ФВ. Для выполнения ФВ на выбранном узле мультикомпьютера интерпретатор должен скопировать (или переместить) значения его входных ФД с тех узлов, на которых они находятся на узел исполнения ФВ. Решение всех этих (и других аналогичных) задач существенно влияет на эффективность выполнения LuNA-программы, а принятие этих решений динамически потребовало бы большое количество накладных расходов. В частности, это касается расходов на динамический разбор операторов LuNA-программы, расходов на работу системных алгоритмов принятия решений, расходов на межузловые коммуникации для обмена системной информацией и т. п. Ранние эксперименты по реализации системы LuNA в виде такого распределенного интерпретатора подтверждают высокую долю накладных расходов, существенно превыщающую время «полезных вычислений» — т. е. собственно вызова последовательных процедур (см. работы [5–8]).

2. Мультиагентный подход. Пусть имеется распределенная мультиагентная система, в которой агенты могут располагаться на одном из узлов мультикомпьютера и перемещаться по узлам, взаимодействовать со своим окружением. Каждый агент имеет своей целью реализацию одного ФВ, и каждый ФВ реализуется ровно одним агентом. Для этого агенту доступно описание оператора LuNA-программы, определяющей соответствующий ФВ. Взаимодействие со своим окружением агент осуществляет через системные вызовы, доступные ему на узле его текущего местоположения и исполняемые системой локально или распределенно. Примеры системных вызовов — это запрос у системы требуемого ФД

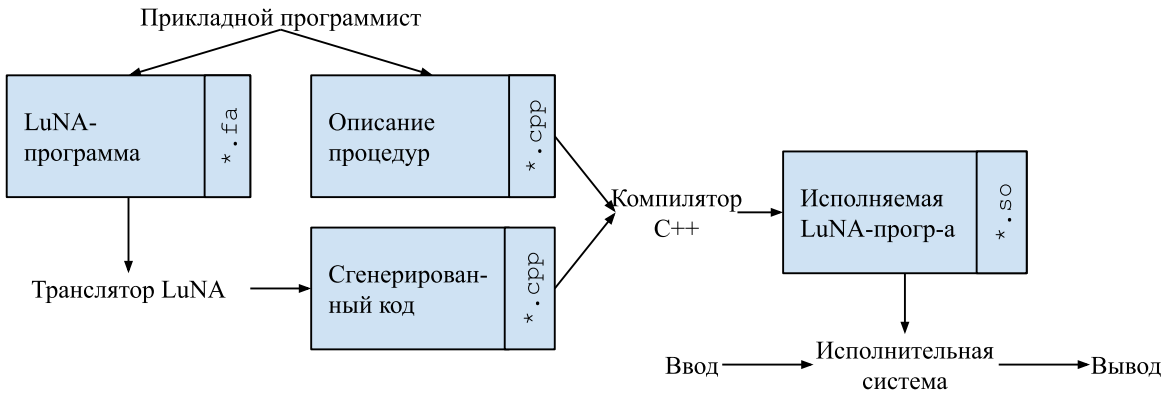

Рис. 1. Архитектура системы LuNA

по его идентификатору или запрос на миграцию агента на заданный соседний узел мультикомпьютера. Каждый агент действует в соответствии со следующим общим для всех ФВ жизненным циклом.

1. Создание агента и инициализация его состояния (соответствующего ему оператора LuNA-программы и его параметров — значений счетчиков объемлющих циклов, доступных ему имен ФД и т. п.).

2. Миграция (перемещение) агента на узел мультикомпьютера, на котором он будет исполняться. Перемещение может осуществляться через несколько узлов-посредников.

3. Запрос и ожидание входных ФД. Может осуществляться в несколько этапов для случаев косвенной адресации (например, если требуется ФД `X[N]`, то сначала будет запрошен и получен ФД `N`, а затем уже требуемый ФД).

4. Исполнение — выполнение процедуры ФВ над входными ФД с выработкой выходных ФД, либо порождение новых агентов в случае структурированного ФВ.

5. Завершающие действия — передача системе команд на удаление ненужных более ФД и пр.

6. Самоуничтожение агента.

В предлагаемом подходе мы переходим от активного интерпретатора к пассивной исполнительной системе, обслуживающей агентов (например, доставляющей запрошенные агентами ФД). В начальный момент времени порождаются агенты, соответствующие операторам верхнего уровня в подпрограмме `main`. Все остальные агенты будут порождены в процессе деятельности этого множества агентов и агентов, порожденных ими. Исполнение завершается, когда последний агент самоуничтожается.

Каждый агент управляется императивной программой, сгенерированной автоматически LuNA-транслятором (рис. 1). Несмотря на то, что количество агентов может быть потенциально бесконечным, количество программ агентов всегда конечно, т. к. программа агента определяется соответствующим ему оператором LuNA-программы, а LuNA-программа конечна.

Программа каждого агента может рассматриваться как общий алгоритм, интерпретирующий оператор агента, но в действительности в общем алгоритме интерпретации нужды нет. Например, если у некоторого ФВ нет входных ФД, то нет нужды генерировать соответствующий код запроса входных данных в программу агента. Таким образом, программа агента содержит только действия, необходимые для реализации конкретного, а не произвольного оператора. Структурно она состоит из последовательности действий,

выполняющих последовательно все шаги жизненного цикла ФВ, изложенного выше. Каждый шаг реализуется через изменение состояния агента и с помощью системных вызовов.

Программа агента может быть сгенерирована на традиционном последовательном языке программирования, например на C++, как это сделано в современной версии системы LuNA. На листинге 2 приведен с несущественными упрощениями пример программы агента, сгенерированной для оператора `cf e` (строка 09 листинга 1).

Листинг 2. Пример сгенерированной программы агента.

```
1: void block_2(CF &self) {
2:   self.migrate(1);
3:   DF N = self.request(self.id("N"));
4:   DF x_N = self.request(self.id("x")[N]);
5:   print(x_N);
6:   self.delete_df(self.id("x")[N]);
7: }
```

На листинге 2 C++-процедура `block_2` — сгенерированная автоматически программа агента, реализующего ФВ `e`. Параметр процедуры `self` — интерфейсный объект, через который агенту доступны его состояние и системные вызовы. На строке 2 агент применяет системный вызов миграции на узел мультикомпьютера с номером 1, потому что транслятор принял решение о том, что ФВ должен выполняться на этом узле. На строке 3 агент запрашивает ФД `N` используя соответствующий системный идентификатор, хранящийся в его состоянии и инициализированный при создании агента. На строке 4 агент уже получил значение ФД `N` и запрашивает через системный вызов ФД `x[N]`, конструируя идентификатор ФД с помощью системного идентификатора, соответствующего имени `x` и значения ФД `N`. Далее на строке 5 происходит вызов процедуры `print`, определенной пользователем в другом C++-файле. На строке 6 агент отдает команду системе на удаление ФД `x[N]` (сборка мусора).

Использование традиционного последовательного языка выгодно с той точки зрения, что эта программа будет скомпилирована в эффективный оптимизированный под заданную архитектуру микропроцессора машинный код с помощью развитых оптимизирующих компиляторов. В частности, статически вычисляемые выражения будут вычислены статически, будет устранен неиспользуемый код, переменные будут по возможности отображены на регистры, вычисления будут по возможности векторизованы и выполнены многие другие оптимизации программного кода, заложенные в традиционные компиляторы. Например, если в LuNA-программе используется выражение вида `x[2+2]`, то такое выражение будет автоматически еще на этапе компиляции преобразовано в `x[4]`. В интерпретаторе это потребовало бы динамического разбора выражения и его вычисления, а в предлагаемом подходе это не придется специально выполнять даже статически в LuNA-трансляторе, потому что эта работа будет сделана традиционным C++-компилятором.

Предложенный подход позволяет статически принимать решения о способе исполнения LuNA-программы и формулировать их в виде программ агентов. Например, решение о том, на каком узле будет исполняться ФВ, закладывается на шаг 2 программы агента, определяющий алгоритм его миграции. То же самое касается и порядка выполнения ФВ (в рамках имеющихся информационных зависимостей), распределения ФД по узлам и т. п.

Рассмотрим пример статического принятия решения о сборке мусора (строка 6 листинга 2). Допустим, путем статического анализа программы на листинге 1 транслятор определил, что для любого ФД `x[i]` последнее потребление будет либо ФВ `e`, либо ФВ `c[i]`. Тогда в конструируемые программы агентов `e` и `c[i]` на шаге 5 будет добавлена команда удаления потребленного ФД `x[i]`. Так статическое решение будет приведено в исполнение динамически без каких-либо накладных расходов на динамический анализ возможности сборки мусора того или иного ФД `x[i]`.

При этом в предлагаемом подходе сохраняется возможность динамического принятия решений. Примером динамического принятия решения о распределении ФВ и ФД по узлам мультикомпьютера для осуществления динамической балансировки нагрузки может служить алгоритм Rope of Beads, изложенный в [14, 15]. Применительно к предлагаемому мультиагентному подходу это означает, что в системе присутствует распределенный системный модуль (алгоритм динамической балансировки нагрузки на узлы), который каждому ФВ и ФД динамически определяет узел исполнения и хранения соответственно. В программы агентов на этапах 2 и 3 вместо конкретных узлов исполнения ФВ и хранения ФД генерируется код обращения к этому модулю, который направит ФВ и запросы на ФД на узлы в соответствии со своим решением, принимаемым динамически.

Предложенный подход позволил существенно сократить накладные расходы на исполнение LuNA-программ. См., например [11]. В работах [12–13] было показано, что на основе предложенного подхода в реальной задаче была достигнута производительность, сопоставимая с ручным низкоуровневым программированием. Это показывает применимость предложенного подхода для реализации практических задач в системе LuNA.

Заключение. В работе предложен мультиагентный подход к эффективной реализации программ на языке LuNA. Подход предоставляет возможность принимать решения о способе исполнения LuNA-программы частично статически и окончательно — динамически. Это позволяет повышать эффективность исполнения LuNA-программ за счет того, что значительная часть работы по исполнению LuNA-программы переносится на этап компиляции, но при этом сохраняется возможность обеспечения динамических свойств исполнения программы за счет того, что в динамике сохраняется фрагментированная структура программы и динамически могут приниматься решения о способе исполнения LuNA-программы. Сильной стороной подхода является использование традиционных последовательных компиляторов C++ для оптимизирующей компиляции программ агентов, что существенно снижает накладные расходы на исполнение LuNA-программы. Экспериментальное исследование подтверждает преимущество подхода по сравнению с распределенной интерпретацией LuNA-программы и показывает, что предлагаемый подход позволяет обеспечивать удовлетворительную эффективность исполнения LuNA-программ, сравнимую с ручным программированием тех же приложений на основе низкоуровневых средств параллельного программирования.

Список литературы

1. Malyshkin, V. E., Perepelkin, V. A. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. // Malyshkin, V. (eds) Parallel Computing Technologies. PaCT 2011. Lecture Notes in Computer Science, 2011. V. 6873. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-642-23178-0_5.

2. Вальковский В. А., Малышкин В. Э. Синтез параллельных программ и систем на вычислительных моделях / Отв. ред. В. Е. Котов; АН СССР, Сиб. отд-ние, ВЦ. Новосибирск: Наука. Сиб. отд-ние, 1988.

3. Малышкин В. Э. Технология фрагментированного программирования // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2012. № 46 (305).

4. Malyshkin V. E. Active Knowledge, LuNA and Literacy for Oncoming Centuries // Essays Dedicated to Pierpaolo Degano on Programming Languages with Applications to Biology and Security. 2015. V. 9465. Springer-Verlag, Berlin, Heidelberg. P. 292–303.

5. Лебедев Д. В., Перепелкин В. А. Численное решение одномерной краевой задачи фильтрации жидкости для системы «нефть-вода» и ее реализация в системе фрагментированного программирования LuNA // Вестник Казахского национального университета им. Аль-Фараби, серия математика, механика информатика. № 3 (82). 2014. С. 64–73.

6. Akhmed-Zaki D. Zh., Lebedev D. V., Perepelkin V. A. Implementation of a Three-Phase Fluid Flow ("Oil-Water-Gas") Numerical Model in the LuNA Fragmented Programming System // In Proc 13th International Conference on Parallel Computing Technologies. LNCS 9251. Springer, 2015. P. 489-497. DOI: 10.1007/978-3-319-21909-7_47.

7. Лебедев Д. В., Перепелкин В. А. Решение трехмерного модельного уравнения теплопроводности в системе фрагментированного программирования LuNA // Параллельные вычислительные технологии (ПаВТ'2016): труды международной научной конференции (28 марта – 1 апреля 2016 г., г. Архангельск). Челябинск: Издательский центр ЮУрГУ, 2016. С. 784.

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

9. Malyshkin V. E., Perepelkin V. A., Schukin G. A. Distributed Algorithm of Data Allocation in the Fragmented Programming System LuNA // In Proc 13th International Conference on Parallel Computing Technologies. LNCS 9251. Springer, 2015. P. 80–85. DOI: 10.1007/978-3-319-21909-7_8.

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

11. Akhmed-Zaki, D., Lebedev, D., Malyshkin, V., Perepelkin, V. Automated construction of high performance distributed programs in LuNA system // 15th International Conference on Parallel Computing Technologies, PaCT 2019; Almaty; Kazakhstan. LNCS 11657. Springer, 2019. P. 3–9. DOI: 10.1007/978-3-030-25636-4_1.

12. Кудрявцев А. А., Малышкин В. Э., Нуштаев Ю. Ю., Перепелкин В. А., Спирин В. А. Эффективная фрагментированная реализация краевой задачи фильтрации двухфазной жидкости // Проблемы информатики. 2023. № 2. С. 45–73. DOI: 10.24412/2073-0667-2023-2-45-73.

13. Belyaev, N., Kireev, S. LuNA-ICLU Compiler for Automated Generation of Iterative Fragmented Programs. // Malyshkin, V. (eds) Parallel Computing Technologies. PaCT 2019. Lecture Notes in Computer Science, V. 11657. Springer, Cham. DOI: 10.1007/978-3-030-25636-4_2.

Виктор Эммануилович Малышкин — получил степень магистра математики в Томском государственном университете (1970), 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. Является одним из организаторов международной конференции PaCT (Parallel Computing Technologies), проводимых 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, технологию активных знаний.

Victor Emmanuilovich Malyshkin graduated from Tomsk State University with the master of sciences degree in 1970, defended his candidate dissertation in physical and mathematical sciences in Computing Centre of SB RAS in 1984, and defended his doctoral dissertation in technical sciences in Novosibirsk State University (1993). Currently is head of parallel programs synthesis laboratory in the Institute of computational mathematics and mathematical geophysics SB RAS.

Is also a founder and head of the chair of parallel computing in Novosibirsk State University. Is one of organizers of the PaCT (Parallel Computing Technologies) international conference, being held each odd year in Russia. Has more than 110 publications on parallel and distributed computing, parallel programs synthesis, supercomputing software and applications, parallel implementation of large-scale numerical models. Current scientific interests include parallel computing technologies, languages and systems of parallel computing, methods and tools of software implementation of large-scale numerical models, the active knowledge technology.

Перепелкин Владислав Александрович — научный сотрудник Института вычислительной математики и математической геофизики СО РАН; старший преподаватель каф. параллельных вычислений факультета информационных технологий Новосибирского национального исследовательского государственного университета. Тел.: (383) 330-89-94, e-mail: perepelkin@ssd.sscc.ru. В 2008 году окончил Новосибирский государственный университет с присуждением степени магистра техники и технологии по направлению «Информатика и вычислительная техника». Имеет более 20 опубликованных статей по теме автоматизации конструирования параллельных программ в области численного моделирования. Является одним из основных разработчиков экспериментальной системы автоматизации конструирования численных параллельных программ для мультикомпьютеров LuNA (от Language for Numerical Algorithms). Область профессиональных интересов включает автоматизацию конструирования параллельных программ, языки и системы параллельного программирования, высокопроизводительные вычисления.

Perepelkin Vladislav Aleksandrovich. Graduated from Novosibirsk State University in 2008 with the master degree in engineering and technology in computer science. Nowadays has the research position at the Institute of Computational Mathematics and Mathematical Geophysics (Siberian Branch of Russian Academy of Sciences), and also has a part time senior professor position in the Novosibirsk State University. He is an author of more than 20 papers on automation of numerical parallel programs construction. He is one of main developers of fragmented programming system LuNA (Language for Numerical Algorithms). Professional interests include automation of numerical parallel programs construction, languages and systems of parallel programming, high performance computing.

Дата поступления — 14.08.2023