Распараллеливание программных комплексов. Проблемы и перспективы

В.А. Бахтин[1,2], О.Ф. Жукова[1], Н.А. Катаев[1], А.С. Колганов[1,2], В.А. Крюков[1,2], М.Ю. Кузнецов[1], Н.В. Поддерюгина[1], М.Н. Притула[1], О.А. Савицкая[1], А.А. Смирнов[1]

1 Институт прикладной математики им. М.В. Келдыша РАН
2 Факультет вычислительной математики и кибернетики Московского государственного университета им М.В. Ломоносова

Аннотация. При распараллеливании реальных производственных программ вычислительного характера приходится сталкиваться со следующими проблемами, характерными для таких программ: многомодульность, многовариантность, многоязыковость. Большой объем программного кода затрудняет проведение необходимых для распараллеливания преобразований вручную и принятие решений по согласованному распределению данных и вычислений. Поэтому в системе SAPFOR [1,2] было предложено организовать для кластеров поэтапное или частичное распараллеливание. Кроме того, особое внимание было направлено на автоматическое определение требуемых преобразований последовательной программы и их автоматическое выполнение. Однако, проблемы, вызываемые многомодульностью, многовариантностью и многоязыковостью, также требуют своего решения. В статье рассматриваются возможные пути преодоления этих проблем.

Ключевые слова: автоматизация распараллеливания, гетерогенный вычислительный кластер, инкрементальное распараллеливание, многомодульность, многовариантность, многоязыковость

Parallelization of software packages. Problems and prospects

V.A. Bakhtin[1,2], O.F. Zhukova[1], N.A. Kataev[1], A.S. Kolganov[1,2], V.A. Krukov[1,2], M.U. Kuznetsov[1], N.V. Podderyugina[1], M.N. Pritula[1], O.A. Savitskaya[1], A.A. Smirnov[1]

1 Keldysh Institute of Applied Mathematics
2 The faculty of Computational Mathematics and Cybernetics of Lomonosov Moscow State University

Abstract. Parallelization of large-scale computational applications leads to the following challenges a programmer will face: the growing amount of modularity, dynamic application functionality and multi-lingual source code. In general program transformation is essential for program parallelization. However, manual program transformation is not straightforward in the case of a large amount of code. The growing size of code drastically complicates the main stages of the parallel programming: distribution of data on the processors and mapping of computations on the processors. Therefore, the incremental or step-by-step parallelization mode was introduced in SAPFOR system [1,2] to map sequential large-scale applications to parallel architectures with distributed memory. In addition, special attention was directed to the automatic finding the required transformations of the sequential program. SAPFOR also relies on the automatic execution of the found transformations. However, the three main challenges mentioned above also require their solution. This paper discusses possible ways to overcome these problems.

Keywords: automation of parallelization, heterogeneous computing cluster, incremental parallelization, multimodularity, multifunctionality, multilingual code

1. Введение

При распараллеливании реальных производственных программ вычислительного характера приходится сталкиваться со следующими распространенными проблемами, характерными для таких программ: многомодульность, многовариантность, многоязыковость [3]. Большой объем программного кода затрудняет проведение вручную необходимых для распараллеливания преобразований и принятие решений по согласованному распределению данных и вычислений. Невозможность проведения для кластеров поэтапного (инкрементального) распараллеливания, широко используемого при распараллеливании для мультипроцессоров, требует от программиста сначала принятия глобальных решений по распределению данных и вычислений с учетом свойств всей программы, а затем выполнения кропотливой работы по модификации программы и ее отладке.

Именно это представляется главной проблемой, препятствующей построению и внедрению средств автоматизации распараллеливания реальных программ, кардинально снижающих трудоемкость такого распараллеливания. Поэтому в системе SAPFOR [1,2] было предложено организовать для кластеров псевдоинкрементальное распараллеливание [4], при котором обеспечиваются основные достоинства инкрементального распараллеливания. Кроме того, особое внимание было направлено на автоматическое определение требуемых преобразований последовательной программы и их автоматическое выполнение. Однако, проблемы, вызываемые многомодульностью, многовариантностью и многоязыковостью, также требуют своего решения. Далее будут рассмотрены возможные пути преодоления этих проблем.

2. Многомодульность

Наличие процедур в программе, с одной стороны, упрощает ее восприятие программистом, а также ее сопровождение, но, с другой стороны, сильно усложняет процесс анализа. Одна из самых популярных оптимизаций программ – подстановка процедур.

Наличие вызова процедуры в тексте программы мешает выполнению различных оптимизаций. Например, присутствие в теле цикла вызова функции может препятствовать векторизации данного цикла. Подстановка процедур позволяет улучшить производительность получаемых программ. Данное преобразование позволяет использовать для оптимизации тела подставленной процедуры весь контекст вызывающей процедуры. Благодаря этому после подстановки может быть получен выигрыш от последующего выполнения различных внутренних оптимизаций компилятора, например, распространение констант, копий. В результате подстановки сокращаются накладные расходы на вызовы функций, улучшается локальность кода, так как вместо перехода на инструкции вызываемой процедуры, которые могут не оказаться в кэше, управление «плавно перетекает» на вставленные в результате подстановки инструкции [9].

При отображении программы на ускорители (например, на графические процессоры – ГПУ) возникает необходимость клонирования ряда процедур или функций. Например, если одна и та же функция выполняется и на центральном процессоре, и на ускорителе, то компилятор должен сгенерировать, по крайней мере, две версии функции: одна – для ЦПУ, другая – для ГПУ (например, в модели CUDA).

Похожая проблема возникает и при отображении программы на кластер. Если одна и та же процедура используется как для работы с данными, которые распределены по узлам кластера, так и с данными, которые находятся на каждом узле кластера, требуется две версии такой процедуры. Например, одна версия процедуры будет работать с локальной частью массива, используя локальную индексацию, другая – с нераспределенными данными в глобальных индексах. Такое клонирование возможно не всегда. Подстановка позволяет избежать ряда проблем, связанных с клонированием процедур.

Однако в некоторых случаях подстановка может оказать негативное влияние на производительность. Вследствие подстановки увеличивается размер кода процедуры, количество локальных переменных, размещаемых на стеке. Из-за увеличения объема исполняемого кода нужные инструкции и данные реже оказываются в кэше и выполнение программы может замедлиться. К тому же разрастание кода негативно влияет на время компиляции и размеры получаемых исполняемых файлов. Также следует учесть, что надежда на «успешность» ряда оптимизаций, которые могут быть выполнены после подстановки процедур, может не оправдаться.

Таким образом, требуется определить критерии, определяющие, какие процедуры нужно стремиться подставлять в первую очередь, какие во вторую,

а какие совсем не подставлять. Для решения данной проблемы в системе SAPFOR было решено использовать граф вызовов функций. Граф вызовов функций представляет собой неориентированный граф, узлами которого являются объявления функции или процедуры, а ребра отражают связи между этими функциями, порожденные операторами вызова процедур или функций в исходном коде программы. Граф вызовов функций строится для всего программного комплекса, который может состоять из нескольких файлов.

Для того чтобы упростить анализ и распараллеливание программы, а в некоторых случаях сделать их возможными, необходимо уметь подставлять процедуры в тексте программы там, где это нужно, а также проводить межпроцедурный анализ. Для этого каждая функция в графе вызовов процедур обладает следующими четырьмя важными флагами:

- Флаг **doNotInline**, который показывает необходимость или отмену подстановки функции. Данный флаг получает положительное значение в результате применения директивы SAPFOR, которая отменяет подстановку функции –!$SPF TRANSFORM(NO_INLINE);
- Флаг **doNotAnalyze**, который показывает необходимость или отмену анализа данной функции, например, на предмет подстановки;
- Флаг **needToInline**, показывающий требование подстановки функции. Данный флаг выставляется в положительное значение в результате межпроцедурного анализа в системе SAPFOR;
- Флаг **deadFunction**, показывающий является ли данная функция «мертвой» в рассматриваемом программном комплексе. «Мертвой» называется функция, которая объявлена, но не используется ни в одном операторе программы. Анализировать такие функции для распараллеливания не имеет смысла.

Наиболее важным анализом является проверка функции на необходимость её подстановки в местах вызова. На данный момент как система DVM [10], так и система SAPFOR имеют ограничения на вызовы пользовательских процедур или функций внутри параллельного цикла (такого цикла, перед которым стоит DVM-директива распределения вычислений), а также на использование таких циклов внутри пользовательских процедур или функций. Данное ограничение, прежде всего, связано с организацией работы с массивами в программе. Рассмотрим данные ограничения подробнее.

В модели DVMH существуют понятия «распределенный массив» и «нераспределенный массив». Распределенным массивом будем называть такой массив, который был распределен средствами DVM с помощью директив *DISTRIBUTE* или *ALIGN*. Нераспределенным будем называть такой массив, который не участвует в распределении средствами DVM. Таким образом, несмотря на то, что текстуально использование распределенных и нераспределенных массивов в программе не отличается, возникает отличие в

момент отображения программы в ее параллельную версию в модели MPI, OpenMP и CUDA с помощью DVMH-компилятора.

После того, как DVMH-программа будет преобразована в параллельную программу с использованием средств MPI, OpenMP и CUDA, а также вызовов функций системы поддержки DVMH, распределенные и нераспределенные массивы будут представлены по-разному. Таким образом, использование одного «экземпляра» процедуры, в параметры которой может быть передан как распределенный, так и нераспределенный массив, недопустимо в рамках DVMH-модели. Таким образом, требуется подстановка таких процедур в точки их вызовов в параллельных циклах, либо, как было отмечено выше, их клонирование.

3. Многовариантность

Современные программные комплексы вычислительного характера имеют достаточно сложную структуру и могут включать в себя следующие компоненты:

- Средства для постановки задачи, которые используются для построения геометрической модели физической расчетной области, генерации расчетных сеток, определения начальных и граничных условий;
- Вычислительное ядро, с помощью которого осуществляется расчет;
- Средства для сохранения, обработки и визуализации результатов расчетов.

Для решения систем уравнений в вычислительном ядре могут быть реализованы различные солверы. Ядро может быть универсальным, выполнять как 2D, так и 3D расчеты, поддерживать различные типы дискретизации, состоять из множества частей.

Например, в программном комплексе NOISEtte [11], который используется для моделирования сжимаемых турбулентных течений, вычислительное ядро состоит из множества модулей:

- Модуль точных решений для различных модельных задач;
- Модуль неявного интегрирования по времени;
- Модуль линейной алгебры;
- Модуль пространственной аппроксимации;
- Модуль вязкости;
- Турбулентный модуль, который реализует набор моделей и методов для моделирования турбулентности;
- Модуль автокоррекции, который с помощью набора критериев корректности расчёта автоматическое восстановление вычислений с коррекцией параметров;
- Модуль вычислительной инфраструктуры, который включает решатель СЛАУ малой размерности, алгоритмы сортировки;

- Другие модули.

Столь сложная структура программного комплекса серьезно усложняет процесс его автоматизированного распараллеливания. При проведении модельных расчетов многие части комплекса могут вообще не выполняться, и их распараллеливание может быть невозможным или требовать принятия общих глобальных решений, например, по распределению данных, которые противоречат требованиям других частей программы. В результате задача поиска оптимального распределения данных и вычислений становится очень трудной, а в большинстве случаев – неразрешимой.

Для решения данной проблемы в системе SAPFOR используются области распараллеливания. *Областью распараллеливания* будем называть последовательность исполняемых операторов в рамках одной области видимости (функция или процедура).

По умолчанию вся программа рассматривается как одна область распараллеливания. Если SAPFOR не справляется с распараллеливанием всей программы, то пользователь может определить свои области распараллеливания, используя специальные указания (директивы). Также в настоящее время реализуется механизм для автоматического определения областей распараллеливания. На основе информации от утилиты GNU Gcov [12], которая позволяет получить точное количество исполнений каждого оператора программы, а также статистику условных переходов (ветвлений), в область распараллеливания попадают времяемкие фрагменты программы (например, процедуры, функции и циклы, которые занимают 90% времени выполнения программы).

Каждая область имеет свой уникальный идентификатор, называемый именем области. Совокупность разных областей (внутри функции, а также в разных файлах) с одинаковым именем рассматриваются системой SAPFOR как одна область распараллеливания. Для каждой области распараллеливания с разными именами строятся разные и независимые решения по распределению данных и вычислений.

Распределенные и нераспределенные массивы в модели DVMH отличаются по использованию их в программе. Чтобы ограничить область действия распределенных средствами DVM массивов, вводится понятие массивов-копий – таких массивов, которые представляют собой копии реальных массивов в программе. До входа в область распараллеливания происходит копирование реальных массивов в массивы копии, а после выхода из нее – обратное копирование. Такой подход позволяет сохранить корректность программы и ограничить действие системы SAPFOR в рамках выделенной области распараллеливания.

Использование областей распараллеливания имеет следующие преимущества:

- Возможность распараллелить не всю программу, а ее времяемкие фрагменты – упрощает работу системы и/или программиста и

позволяет потратить больше времени ЭВМ (и программиста), чтобы найти лучшие схемы распараллеливания времяемких фрагментов.

- Отказ от распараллеливания сложных фрагментов позволяет с большей вероятностью найти хорошие решения, поскольку эти сложные фрагменты могут приводить к неверной работе алгоритмов анализа, распределения данных и вычислений или к тому, что не найдется схем распараллеливания, обеспечивающих приемлемое ускорение.
- Возможна реализация ручного распараллеливания некоторых фрагментов программы и учета принятых программистом решений при распараллеливании других фрагментов.

4. Многоязыковость

Многомодульность программного комплекса является потенциальным источником еще одной проблемы распараллеливания: разные модули могут быть написаны на разных языках программирования, например, Fortran, C или C++. Это может быть вызвано необходимостью использования сторонних библиотек, разными предпочтениями разработчиков исходного кода, направлением развития технологий программирования (например, широкое распространение объектно-ориентированного подхода). Несмотря на внешнюю схожесть, каждый язык обладает своими особенностями, которые должны учитываться во время его совместного использования с другим языком. Начиная от именования внешних символов (Fortran добавляет к именам процедур нижнее подчеркивание и переводит их в нижний регистр) и способа передачи параметров при вызове процедур (в Fortran они передаются по ссылке, а в C – по значению), заканчивая представлением объектов в памяти. Последнее является особенно существенным при автоматизации распараллеливания программ для гетерогенных кластеров. Системы автоматизации, такие как SAPFOR, должны учитывать порядок расположения многомерных массивов в памяти, чтобы корректно распределять массивы по процессорам вычислительной системы.

Еще одна важная особенность, которая должна учитываться при распараллеливании многоязыковых комплексов – невозможность выполнить подстановку процедур, написанных на одном языке программирования, в исходный код на другом языке. Эта сложность имеет большое значение не только при автоматизации распараллеливания, но и при ручном программировании, так как требование подстановки процедур может быть вызвано необходимостью устранения проблем, препятствующих распараллеливанию программы.

С целью учесть особенности различных языков в системе SAPFOR строится иерархическое внутреннее представление программы SAPFOR IR. На верхнем уровне программа представлена в виде абстрактного синтаксического дерева (AST), которое описывает программу на языке высокого уровня,

позволяет проводить преобразования программы и восстанавливать исходный код преобразованной программы на языке высокого уровня. Для исследования и модификации программы на данном уровне в случае C/C++ используются возможности компилятора Clang [5], а в случае Fortran применяется существенно переработанная и расширенная (для поддержания Fortran 95) библиотека Sage++ [6].

Следующий уровень основан на LLVM IR и опирается на низкоуровневое представление программы. Система компиляторных технологий LLVM [7] позволяет сделать данный уровень единым для всех поддерживаемых в SAPFOR языков программирования. Для генерации LLVM IR в случае Fortran применяется недавно появившийся фронтенд Flang [8]. Универсальность LLVM IR способствует использованию единых алгоритмов анализа сразу для всех поддерживаемых языков. При разработке SAPFOR применяются алгоритмы, уже реализованные в LLVM, а также добавляются новые алгоритмы анализа, необходимые для распараллеливания и отсутствующие в LLVM. Кроме того, LLVM позволяет выполнять преобразования программ, упрощающие анализ (подстановка констант, построение формы однократного присваивания (SSA), поворот циклов), не затрагивая исходный код на языке высокого уровня. Доступная в LLVM IR отладочная информация (metadata) используется для отображения результатов анализа программы, представленной в виде LLVM IR, на соответствующее исходному языку представление программы в виде AST. Архитектура LLVM, включающая понятия проходов анализа и преобразований, хорошо согласуется с архитектурой SAPFOR и идеей проходов, которые были предложены и рассмотрены в статье [4].

Возможностей LLVM оказывается недостаточно для принятия решений по распределению данных и вычислений при распараллеливании программы. С этой целью в SAPFOR вводятся специальные графовые представления программы, независящие от исходного языка и описывающие связи между распределяемыми объектами программы, такими как массивы. Данные графовые модели строятся на основе результатов анализа, полученных с использованием LLVM IR, и структуры программы, описанной в виде AST.

5. Заключение

Ручное распараллеливание больших программ и программных комплексов является очень трудоемким процессом. Для автоматизированного распараллеливания реальных производственных программ требуется существенное расширение возможностей системы SAPFOR для преодоления таких проблем, как многоязыковость, многовариантность и многомодульность. В статье были рассмотрены особенности реализации описанных проблем и проиллюстрированы первые результаты их решения.

Работа выполнена при поддержке РФФИ (проекты № 16-07-01067, 16-07-01014, 16-29-09550, 17-01-00820, 18-01-00851) и программы президиума РАН

№ 26 «Фундаментальные основы создания алгоритмов и программного обеспечения для перспективных сверхвысокопроизводительных вычислений».

Литература

1. Клинов М.С., Крюков В.А. Автоматическое распараллеливание Фортран-программ. Отображение на кластер // Вестник Нижегородского университета им. Н.И. Лобачевского. 2009. № 2. С. 128–134 .
2. Бахтин В.А., Клинов М.С., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н. Автоматическое отображение программ на языке фортран на кластеры с графическими процессорами // Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика. 2014. Т. 3. № 3. С. 86–96 .
3. Armstrong B. and Eigenmann R. Application of Automatic Parallelization to Modern Challenges of Scientific Computing Industries // In Proceedings of 37th International Conference on Parallel Processing. 2008. ICPP '08. 9–12 Sept. 2008. pp. 279–286 .
4. Бахтин В.А., Жукова О.Ф., Катаев Н.А., Колганов А.С., Королев Н.Н., Крюков В.А., Кузнецов М.Ю., Поддерюгина Н.В., Притула М.Н., Савицкая О.А., Смирнов А.А. Инкрементальное распараллеливание для кластеров в системе САПФОР // Научный сервис в сети Интернет: труды XIX Всероссийской научной конференции (18-23 сентября 2017 г., г. Новороссийск). — М.: ИПМ им. М.В. Келдыша, 2017. — С. 48–52. — doi: 10.20948/abrau-2017-11 .
5. Clang: a C language family frontend for LLVM. URL: http://clang.llvm.org .
6. pC++/Sage++. URL: http://www.extreme.indiana.edu/sage/ .
7. Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation // In Proceedings of the International Symposium on Code Generation and Optimization (GCO). 2004. pp. 75–87.
8. Flang: Fortran compiler targeting LLVM. URL: https://github.com/flang-compiler/flang .
9. Критерий выгодности подстановки и динамическая профилировка. URL: https://habr.com/company/intel/blog/240203/ .
10. DVM-система. URL: http://www.keldysh.ru/dvm, http://dvm-system.org/ .
11. Горобец А.В. Параллельные технологии математического моделирования турбулентных течений на современных суперкомпьютерах [Текст]: дис. д-р. физ.-мат. наук: 05.13.18: защищена 03.12.2015 / Горобец Андрей Владимирович. – М. 2015. 226 С.
    URL: http://keldysh.ru/council/3/D00202403/gorobets_a_v_diss.pdf .
12. Утилита GNU Gcov. URL: http://gcc.gnu.org/onlinedocs/gcc/Gcov.html .

References

1. Klinov M.S., Kriukov V.A. Avtomaticheskoe rasparallelivanie Fortran-programm. Otobrazhenie na klaster // Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo. 2009. № 2. S. 128-134 .
2. Bakhtin V.A., Klinov M.S., Kolganov A.S., Kriukov V.A., Podderiugina N.V., Pritula M.N. Avtomaticheskoe otobrazhenie programm na iazyke fortran na klastery s graficheskimi protsessorami // Vestnik Iuzhno-Uralskogo gosudarstvennogo universiteta. Seriia: Vychislitelnaia matematika i informatika. 2014. T. 3. № 3. S. 86-96 .
3. Armstrong B. and Eigenmann R. Application of Automatic Parallelization to Modern Challenges of Scientific Computing Industries // In Proceedings of 37th International Conference on Parallel Processing. 2008. ICPP '08. 9–12 Sept. 2008. pp. 279–286 .
4. Bakhtin V.A., Zhukova O.F., Kataev N.A., Kolganov A.S., Korolev N.N., Kriukov V.A., Kuznetsov M.Iu., Podderiugina N.V., Pritula M.N., Savitskaia O.A., Smirnov A.A. Inkrementalnoe rasparallelivanie dlia klasterov v sisteme SAPFOR // Nauchnyi servis v seti Internet: trudy XIX Vserossiiskoi nauchnoi konferentsii (18-23 sentiabria 2017 g., g. Novorossiisk). - M.: IPM im. M.V. Keldysha, 2017. - S. 48-52. -doi: 10.20948/abrau-2017-11 .
5. Clang: a C language family frontend for LLVM. URL: http://clang.llvm.org .
6. pC++/Sage++. URL: http://www.extreme.indiana.edu/sage/ .
7. Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation // In Proceedings of the International Symposium on Code Generation and Optimization (GCO). 2004. pp. 75–87.
8. Flang: Fortran compiler targeting LLVM. URL: https://github.com/flang-compiler/flang .
9. Kriterii vygodnosti podstanovki i dinamicheskaia profilirovka. URL: https://habr.com/company/intel/blog/240203/ .
10. DVM-system. URL: http://www.keldysh.ru/dvm, http://dvm-system.org/ .
11. Gorobets A.V. Parallelnye tekhnologii matematicheskogo modelirovaniia turbulentnykh techenii na sovremennykh superkompiuterakh [Tekst]: dis. d-r. fiz.-mat. nauk: 05.13.18: zashchishchena 03.12.2015 / Gorobets Andrei Vladimirovich. – M. 2015. 226 S.
    URL: http://keldysh.ru/council/3/D00202403/gorobets_a_v_diss.pdf .
12. GNU Gcov Tool. URL: http://gcc.gnu.org/onlinedocs/gcc/Gcov.html .