ИПМ им.М.В.Келдыша РАН

Абрау-2016 • Труды конференции

В.А. Бахтин, О.Ф. Жукова, Н.А. Катаев,
А.С. Колганов, В.А. Крюков,
Н.В. Поддерюгина, М.Н. Притула,
О.А. Савицкая, А.А. Смирнов

Автоматизация распараллеливания программных комплексов

Рекомендуемая форма библиографической ссылки

Бахтин В.А., Жукова О.Ф., Катаев Н.А., Колганов А.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Савицкая О.А., Смирнов А.А. Автоматизация распараллеливания программных комплексов // Научный сервис в сети Интернет: труды XVIII Всероссийской научной конференции (19-24 сентября 2016 г., г. Новороссийск). — М.: ИПМ им. М.В.Келдыша, 2016. — С. 76-85. — doi:10.20948/abrau-2016-31

Размещена также презентация к докладу

Автоматизация распараллеливания программных комплексов

В.А. Бахтин[1,2], О.Ф. Жукова[1], Н.А. Катаев[1], А.С. Колганов[1,2], В.А. Крюков[1,2], 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. Опыт использования системы САПФОР для распараллеливания Фортран-программ показал, что для распараллеливания больших программ и программных комплексов необходимо развивать систему в следующих направлениях – автоматическое определение требуемых преобразований последовательной программы и их автоматическое выполнение, распараллеливание не всей программы, а только ее времяемких фрагментов, инкрементальное распараллеливание. Кроме того, необходимо расширение возможностей системы для распараллеливания программ на языке Си. Развитие системы в указанных направлениях требует существенной корректировки ее внутренней организации.

Ключевые слова: автоматизация распараллеливания, гетерогенный вычислительный кластер, инкрементальное распараллеливание

1. Введение

Прекращение роста частоты отдельно взятого процессора привело к повышенному интересу к использованию многоядерных и многопроцессорных вычислительных систем для достижения максимальной производительности при расчете вычислительно емких задач. При этом умение писать параллельные программы, способные эффективно выполняться на таких системах, граничит с искусством и требует от программиста знания не только используемых программных моделей, но и особенностей аппаратуры, на которой будет выполняться разрабатываемая программа. Ситуация особенно усложнилась с появлением новых архитектур, таких как графические ускорители NVIDIA или сопроцессоры Intel. С целью задействовать все уровни параллелизма при разработке параллельных программ одновременно должны быть использованы различные технологии параллельного программирования (MPI, OpenMP, CUDA, OpenACC).

Еще более непростой ситуация становится при попытке распараллелить ранее написанный последовательный код: зачастую требуется значительное

изменение подлежащих распараллеливанию участков, которые в противном случае не могут быть выполнены параллельно. Разработка инструментов, способных оказывать помощь программисту при решении этих задач, а тем более, разработка инструментов, способных решать указанные проблемы в автоматическом режиме, может внести значительный вклад в развитие отрасли суперкомпьютерных вычислений и способна значительно снизить трудозатраты на распараллеливание существующих и разработку новых программ.

Автоматизированное распараллеливание полагается на возможность выявления участков программного кода, которые могут быть выполнены независимо, и, следовательно, требует выполнения точного анализа исходного кода программы и определения зависимостей по данным и по управлению, присутствующих в ней. Но наличие мощных средств анализа оказывается недостаточным для успешного распараллеливания [1]. Авторы данной статьи приходят к выводу, что в дополнение к средствам анализа также необходимо использовать другие методы, и предлагают обратить внимание на спекулятивное выполнение участков программы. Другой подход, который успешно используется в современных оптимизирующих компиляторах — преобразование программы с целью достижения поставленной цели оптимизации.

Оптимизацию и распараллеливание программ можно свести к выполнению последовательности фаз анализа и преобразования. Выбор оптимальной последовательности фаз определяется особенностями прикладной программы, целью проводимой оптимизации, архитектурой целевой вычислительной системы, используемыми технологиями параллельного программирования. Обычно не удается построить единую оптимальную последовательность, пригодную для разных участков кода даже в рамках одной программы [2, 3]. Каждое выполняемое преобразование может характеризоваться набором параметров, таких как tile size, loop unrolling factor и др. [4, 5]. Несмотря на это, базовым подходом к построению компиляторов является использование заранее определенных последовательностей фаз, не зависящих от исходной программы и не опирающихся на ее информационную структуру [6]. Основной причиной этого являются огромные размеры пространства возможных оптимизационных последовательностей [7].

Поиск оптимальной последовательности зависит от успешного решения ряда задач [3]. Необходимо количественно оценивать, какое влияние окажет оптимизация на выбранный участок исходного кода. Качественная оценка (деградация или польза) окажется недостаточной, если применимы несколько оптимизаций. Более того нужно учитывать влияние оптимизации на программу в контексте всей оптимизационной последовательности. Требует решения и упомянутая выше проблема подбора оптимизационных параметров.

Наряду с априорным подходом, который применяется в большинстве компиляторов и при котором выбор оптимизационной последовательности определяется множеством эвристик, предсказывающих влияние оптимизаций

на программу на основе предварительного анализа исходного кода, применяется апостериорный подход [8]. Указанный подход представляет собой итерационную компиляцию и заключается в применении к участку кода множества различных оптимизационных последовательностей и выбора из них наилучшей согласно целям выполняемой оптимизации.

Итерационная компиляция предполагает возможность оценки результата применения некоторой последовательности и использование алгоритма поиска в пространстве оптимизационных последовательностей, отличного от простого перебора возможных вариантов [7]. Способ оценки эффективности оптимизационной последовательности может быть эмпирическим и опираться на результаты запусков преобразованной программы на целевой архитектуре [2, 9] или аналитическим [3]. Аналитический способ основан на построении математических моделей программы, вычислительной системы и выполняемых оптимизаций и позволяет избежать накладных расходов, вызванных большим количеством реальных запусков преобразованных программ. Для выбора очередной оцениваемой последовательности могут применяться генетические алгоритмы [7, 9], алгоритмы восхождения к вершине (hill climbing) [7], жадные алгоритмы [7]. Системы для интерактивного взаимодействие с пользователем [2] позволяют настраивать параметры поиска, предоставлять информацию об эффективности оптимизаций в графическом виде, выбирать применяемые преобразования, описывать условия применения оптимизационных фаз.

Применение интерактивных систем вносит значительный вклад в автоматизацию параллельного программирования [10-14, 20]. Некоторые системы включают в свой состав автоматически распараллеливающие компиляторы [13, 14], другие ориентированы на исследование программы статическими и/или динамическими методами и предполагают, что распараллеливание программы (преобразование исходного кода, расстановка директив параллельного программированию и др.) выполняется пользователем вручную [10].

Особенностью системы САПФОР [13] является использование автоматически распараллеливающего компилятора для систем с распределенной памятью. Взаимодействие с пользователем осуществляется на этапе подготовки последовательной программы к автоматическому распараллеливанию. Данная система переводит последовательную пользовательскую программу в параллельную программу в модели DVMH [15, 16].

При этом существующая реализация системы обладает рядом недостатков, которые препятствуют ее применению при распараллеливании крупных программных комплексов. Данная статья посвящена рассмотрению существующих недостатков системы и описанию решений по перепроектированию системы, принимаемых с целью их устранения.

Другим важным аспектом в развитии системы является идея внедрения метода инкрементального распараллеливания программных комплексов. Инкрементальное распараллеливание широко применяется при разработке программ для мультипроцессора, но при использовании для систем с распределенной памятью его применение наталкивается на значительные трудности. Необходимость распределения данных по процессорам влечет за собой накладные расходы на коммуникацию, для эффективной оптимизации которых может потребоваться рассмотрение всей программы в целом, а не отдельных ее частей. Отсутствие полного распределения данных ограничивает масштабируемость программы. В системе САПФОР предлагается ввести понятие области распараллеливания. В совокупности с выполнением итерационного распараллеливания это позволит последовательно переходить от рассмотрения отдельных небольших областей к более крупным, вплоть до целой программы, сохраняя при этом преемственность ранее принятых решений по распараллеливанию отдельных областей и при необходимости их уточняя.

2. Проблемы системы САПФОР

2.1. Преобразования входной программы

Текущая версия системы САПФОР работает по следующей схеме. На вход ей подается последовательная программа в специальном внутреннем представлении, которое дополняется результатами требуемых анализов, таких как зависимости между витками цикла, наличие приватизируемых скалярных переменных и др. Далее, система пытается с помощью своих алгоритмов перевести данную программу в параллельную программу в модели DVMH. Так как реализация системы не предполагает проведение каких-либо преобразований, в результате ее работы может быть получена либо неэффективная параллельная программа (что в большинстве случаев и происходит), либо пользователю будет выдано очень много диагностик о невозможности ее распараллеливания. Исследуя результаты распараллеливания, пользователь сам принимает решения о том, какие преобразования кода ему необходимо выполнить, чтобы устранить причины, мешающие автоматическому распараллеливанию, и после этого цикл распараллеливания повторяется.

Таким образом, выполнение преобразований программы, пусть даже очень простых, ложится на пользователя. К таким преобразованиям можно отнести, например, объединение циклов в тесно вложенные, устранение некоторых простых зависимостей по данным внутри цикла и др.

Автоматизация выполнения преобразований последовательной программы является важным свойством системы распараллеливания, так как существующие комплексы программ содержат в себе большое количество строк кода и в большинстве случаев пользователю не под силу выполнить

преобразования вручную для устранения причин невозможности распараллеливания, даже если система САПФОР выдаст соответствующие диагностики.

Проведенные ранее исследования [17, 18] показывают, что область применения системы САПФОР может быть расширена за счет автоматизации преобразований последовательных программ, осуществляемой в двух направлениях. Во-первых, за счет автоматического устранения проблем, препятствующих распараллеливанию [17], во-вторых, за счет предоставления пользователю возможности выбора из набора элементарных преобразований, которые затем могут быть выполнены системой автоматически [18].

2.2. Инкрементальное распараллеливание больших программных комплексов

Как было отмечено выше, описываемая система автоматизированного распараллеливания ориентирована в основном на отображение уже существующих последовательных программ в эффективные параллельные. Такие программы могут содержать в себе отдельные модули, решающие определенные задачи, например, модуль ввода/вывода, модуль решения систем линейных уравнений, модуль, отвечающий за подготовительные работы. И в большинстве случаев такие программы не нуждаются в полном распараллеливании, а в некоторых случаях полное распараллеливание такой программы попросту невозможно без существенного переписывания кода.

Но полное распараллеливание часто не требуется. Обычно бывает достаточно распараллелить только модули, производящие сложные вычисления над уже подготовленными данными. С другой стороны, программа может быть настолько большой, что программисту необходимо понять, можно ли вообще получить эффективно работающую программу на конкретной архитектуре. Для выполнения такой оценки достаточно попытаться распараллелить только некоторые модули такой программы, которые могут занимать до 90% времени.

Для решения такой проблемы предлагается метод инкрементального, или частичного распараллеливания. Данный метод заключается в том, что программа изначально разбивается на так называемые области распараллеливания. Данные области могут быть построены на основе времен, полученных с помощью профилирования последовательной программы. Области распараллеливания отражают те участки кода исходной программы, которые будут рассматриваться системой САПФОР.

Можно отметить следующие достоинства такого подхода:

- Возможность распараллелить не всю программу, а ее времяемкие фрагменты. Это упрощает работу системы САПФОР и/или программиста, так как существенно сокращается объем кода программы для анализа и распараллеливания. А также позволяет потратить больше времени ЭВМ (и программиста), чтобы найти лучшие схемы распараллеливания времяемких фрагментов.

- Появляется возможность постепенного добавления новых областей распараллеливания с сохранением уже найденных решений для ранее объявленных областей или без сохранения этих решений.
- Отказ от распараллеливания сложных фрагментов программы позволяет с большей вероятностью найти хорошие решения для выделенных областей распараллеливания, поскольку эти сложные фрагменты могут приводить к тому, что сразу не найдется схем распараллеливания для всей программы, обеспечивающих приемлемое ускорение. Найденные решения для времяемких фрагментов могут быть использованы в качестве подсказки при исследовании оставшихся частей программы на следующих итерациях распараллеливания в системе САПФОР.
- Появляется возможность ручного распараллеливания некоторых фрагментов программы и учета принятых программистом решений при распараллеливании других фрагментов системой САПФОР.

2.3. Архитектура системы: более тесное взаимодействие между компонентами

Текущая реализация системы САПФОР представляет собой набор независимых компонент: анализатор, разбирающий структуру входной программы, анализатор зависимостей по данным, модуль, отвечающий за построение вариантов параллельных программ, интерактивная оболочка взаимодействия с пользователем. Все компоненты осуществляют взаимодействие через специальную базу данных. В базе данных содержится минимальная информация о распараллеливаемой программе, а также ее свойства.

В отличие от компиляторов, в которых фазы анализа и преобразования программ оперируют представлением всей программы, большинство компонент системы САПФОР оперирует с минимальным набором операторов и свойств программы, сохраненных в базе данных. Такая реализация не позволяет строить оптимизационные последовательности фаз преобразования и анализа, так как в базе данных отсутствует необходимая информация о программе.

Как следствие, итерационное распараллеливание в системе САПФОР хотя и возможно, но состоит из очень крупных итераций, каждая из которых предполагает последовательный запуск всех компонент системы и только в определенном порядке. В результате, после незначительного преобразования исходного кода распараллеливаемой программы, необходимо заново запустить все компоненты, отвечающие за анализ и распараллеливание. Время выполнения отдельной итерации может быть очень велико, но большая часть работы будет повторять действия, выполненные на предыдущих итерациях.

Существующая база данных обладает рядом других недостатков, рассмотренных в работе [19].

3. Измененная архитектура системы

Данный раздел посвящен рассмотрению измененной архитектуры системы, в которой будут устранены описанные выше недостатки. Система САПФОР реализует переход от последовательной программы к параллельной программе и отвечает за исследование программы, принятие решений по устранению проблем, препятствующих распараллеливанию, распределение данных и вычислений, оценку возможных вариантов распараллеливания программы и взаимодействие с пользователем в интерактивном режиме.

Принятие решений в системе базируется на концепции, используемой в современных оптимизирующих компиляторах: выполнении последовательности фаз анализа и преобразований. Взаимодействие между фазами осуществляется опосредовано через доступ к используемым ими данным. Данные состоят из представления программы в некоторый момент времени, например, графа потока управления и актуального в этот момент времени набора свойств программы, например, описание зависимостей между операторами в циклах программы. Выделяется два типа фаз: фазы анализа, определяющие свойства программы на основе ее представления, и фазы преобразований, изменяющие представление программы. Фазы преобразования могут аннулировать результаты фаз анализа. Необходимость доступа к результату одной фазы при выполнении другой порождает зависимость между фазами.

Процесс распараллеливания программы в системе заключается в поиске оптимизационной последовательности фаз, обеспечивающей переход от последовательной версии программы к эффективной параллельной версии. Оптимизационная последовательность фаз и совокупность данных после выполнения каждой из фаз образует представление процесса распараллеливания в системе САПФОР (ПР САПФОР). Номер фазы в последовательности и данные, доступные после ее выполнения, задают состояние ПР САПФОР.

Фазы в последовательности могут повторяться, могут выполняться над разными частями программы. В последовательность не обязательно входят все фазы, доступные в системе. Правильный порядок фаз с учетом зависимостей и необходимости соблюдения актуальности данных гарантируется специальным модулем системы, называемым менеджером фаз.

Входные данные для фаз анализа могут не зависеть от языка программирования (ЯП), это позволяет разрабатывать общие алгоритмы для языков программирования Фортран и Си. Входные данные для фаз преобразования зависят от ЯП, так как преобразование программы выполняется на уровне исходного языка. Таким образом, могут использоваться разные реализации одной и той же фазы для ЯП Фортран и Си.

Последовательность фаз не обязательно является линейной. Это означает, что некоторая фаза может создать дополнительный экземпляр менеджера фаз, подготовить для него множество фаз, которые необходимо выполнить и

запустить их на выполнение. При этом выполнение дальнейшей последовательности фаз будет продолжено после завершения вложенного менеджера фаз. Примером такой фазы может являться фаза, определяющая проблемы, мешающие распараллеливанию, и обозначающая пути поиска их решения. Так как фазы преобразования меняют код исходной программы, то привязка диагностических сообщений и другой информации, предназначенной для пользователя, должна осуществляться к коду, актуальному в данный момент.

Допускается сохранение состояний ПР САПФОР. Это позволит временно приостанавливать работу системы, например, с целью получения результатов динамического анализа. Данная возможность также необходима, когда на очередном шаге допустимо выполнение нескольких фаз преобразования, но неизвестно, выполнение какой из них наилучшим образом скажется на результате распараллеливания. Сохранение состояния ПР САПФОР в такой точке ветвления позволит исследовать разные последовательности фаз отдельно и выбрать наилучшую.

Заключение

Выполнение преобразований последовательных программ во многих случаях может повысить эффективность их распараллеливания. Требующиеся преобразования и их последовательность зависят от особенностей прикладной программы, архитектуры целевых вычислительных систем и используемых технологий параллельного программирования. Применение в процессе распараллеливания заранее фиксированной последовательности фаз анализа и преобразования, как это делается в оптимизирующих компиляторах, часто оказывается неэффективным и ограничивает возможности использования автоматически распараллеливающих компиляторов. Ручное распараллеливание программ является трудоемким процессом, хотя и позволяет учесть все указанные особенности.

Подход к автоматизации распараллеливания, предложенный авторами статьи, заключается в инкрементальном выполнении итерационного процесса распараллеливания для гетерогенных вычислительных кластеров и расширении данного процесса возможностью интерактивного взаимодействия с пользователем. Применение данного подхода в системе САПФОР наталкивается на ряд трудностей, связанных с архитектурными решениями, принятыми при проектировании системы. В статье рассмотрена обновленная архитектура системы САПФОР, которая лишена существующих недостатков и может быть использована для реализации предложенного подхода.

Литература

1. Niall Murphy, Timothy Jones, Simone Campanoni, and Robert Mullins. Limits of dependence analysis for automatic parallelization // Proceedings of the 18th

International Workshop on Compilers for Parallel Computing (CPC 2015, London, January 9 2015),
URL: https://www.cl.cam.ac.uk/~nedm2/cpc15paper.pdf (дата обращения 25.05.2016)
2. Kulkarni P., Zhao W., Moon H., Cho K, Whalley D., Davidson J., Bailey M., Paek Y., Gallivan K. Finding effective optimization phase sequences // Proceedings of the 2003 ACM SIGPLAN Con-ference on Languages, Tools, and Compilers for Embedded Systems. 2003. P. 12–23.
3. Zhao M., Childers B, Soffa M. Predicting the impact of optimizations for embedded systems. // Proceedings of the 2003 ACM SIGPLAN Conference on Languages, Tools, and Compilers for Embedded Systems. 2003. P. 1–11
4. Bacon D., Graham S., Sharp O. Compiler Transformations for High-Performance Computing. // ACM Computing Surveys. 1994. Vol. 26, Issue 4, P. 345-420.
5. Wolfe M.J. High Performance Compilers for Parallel Computing // Addison-Wesley, Boston. 1995. P. 570, Chapter 5.
6. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления // СПб.: БХВ-Петербург, 2004. 608 С.
7. Almagor L., Cooper K., Grosul A., Harvey T., Reeves S., Subramanian D., Torczon L., Water-man T. Finding effective compilation sequences // Proceedings of the 2004 ACM SIG-PLAN/SIGBED conference on Languages, compilers, and tools for embedded systems, Washing-ton, DC, June 11-13 2004, – ACM New York, NY, USA. 2004. P. 231-239
8. Triantifyllis S., Vachharajani M., August D. Compiler Optimization-Space Exploration // Journal of Instruction-Level Parallelism. 2005. Vol. 7. P 1-25
9. Nisbet A. GAPS: Iterative feedback directed parallelisation using genetic algorithm // Proceedings of the Workshop on Profile and Feedback-Directed Compilation, (Paris, France). 1998. P. 1-8
10. Intel Parallel Studio. URL: http://software.intel.com/en-us/intel-parallel-studio-home (дата обращения 25.05.2016)
11. Liao S., Diwan A., Bosch R., Ghuloum A., Lam M. Suif Explorer: an Interactive and Interprocedural Parallelizer // Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 1999. P. 37-48
12. Sudhakar Sah, Vinay G. Vaidya Review of Parallelization Tools and Introduction to Easypar // International Journal of Computer Applications. 2012. Vol. 56, No. 12 P. 17-29
13. Бахтин В.А., Бородич И.Г., Катаев Н.А., Клинов М.С. Ковалева Н.В., Крюков В.А., Поддерюгина Н.В. Диалог с программистом в системе автоматизации распараллеливания САПФОР // Вестник Нижегородского университета им. Н.И. Лобачевского. – Н. Новгород: Изд-во ННГУ, №5 (2), 2012. С. 242– 245.
14. Юрушкин М.В., Петренко В.В., Штейнберг Б.Я., Алымова Е.В., Абрамов А.А., Баглий А.П., Гуда С.А., Дубров Д.В., Кравченко Е.Н., Морылев Р.И., Нис З.Я., Полуян С.В., Скиба И.С., Шаповалов В.Н.,

Штейнберг О.Б., Штейнберг Р.Б. Диалоговый высокоуровневый оптимизирующий распараллеливатель (ДВОР) // Труды Международной суперкомпьютерной конференции “Научный сервис в сети Интернет: суперкомпьютерные центры и задачи”, Новороссийск, сентябрь 2010 г., – М.: Изд-во МГУ, 2010. С. 71-75.

15. Бахтин В.А., Клинов М.С., Крюков В.А., Поддерюгина Н.В., Притула М.Н., Сазанов Ю.Л. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами. – Вестник Южно-Уральского государственного университета, серия "Математическое моделирование и программирование", №18 (277), выпуск 12 – Челябинск: Издательский центр ЮУрГУ, 2012. С. 82-92
16. Коновалов Н.А., Крюков В.А., Михайлов С.Н., Погребцов А.А. Fortran DVM —- язык разработки мобильных параллельных программ // Программирование, № 1, 1995, С. 49-54.
17. Катаев Н.А., Буланов А.А. Автоматизация преобразований последовательных Фортран-программ, необходимых для их эффективного распараллеливания // Параллельные вычислительные технологии (ПаВТ'2015): труды международной научной конференции (31 марта – 2 апреля 2015 г., г. Екатеринбург) – Челябинск: Изд-во ЮУрГУ, 2015. С. 172-177.
18. Baranov M.S., Ivanov D.I., Kataev N.A., Smirnov A.A. Automated parallelization of sequential C-programs on the example of two applications from the field of laser material processing // Суперкомпьютерные дни в России: Труды международной конференции (28-29 сентября 2015 г., г. Москва). – М.: Изд-во МГУ, 2015. С. 536
19. Катаев Н.А. Особенности внутреннего представления системы САПФОР. // Труды международной научной конференции Параллельные вычислительные технологии (ПаВТ’2013, 1 - 5 апреля 2013 г., г. Челябинск) – Челябинск: Изд-во ЮУрГУ, 2013, с. 380-386.
20. ParaWise – Widening Accessibility to Efficient and Scalable Parallel Code. // Parallel Software Products White Paper WP-2004-01, 2004.