О создании распараллеливающих компиляторов на вычислительные системы с распределенной памятью

Штейнберг Б.Я.

Южный федеральный университет

«Исследование выполнено за счет гранта Российского научного фонда № 22-21-00671, https://rscf.ru/project/22-21-00671/»

Аннотация.

В данной работе описаны условия создания оптимизирующих распараллеливающих компиляторов на вычислительные системы с распределённой памятью. Приводятся оптимизирующие преобразования программ как специфичные для систем с распределенной памятью, так и преобразования программ, которые нужны для вычислительных систем с распределенной памятью и могут улучшить компиляторы для вычислительных систем и с общей памятью. Приводятся аргументы в пользу того, что создавать распараллеливающие компиляторы на вычислительные системы с распределенной памятью следует на основе высокоуровневого внутреннего представления и с высокоуровневым выходным языком.

Ключевые слова: автоматизация распараллеливания, распределенная память, преобразования программ, размещение данных, пересылки данных.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ 1
2. Модель вычислительной системы. 3
3. Специфические для ВСРП оптимизирующие преобразования программ. 4
4. Оптимизирующие преобразования программ для ВСРП, которые полезны и для ВСОП. 5
5. Рекомендации к созданию архитектуры распараллеливающей системы на ВСРП. 6
6. Оптимизирующие компиляторы или библиотеки? 7
7. Заключение 7
8. Литература 8

1. ВВЕДЕНИЕ

В данной работе идет речь о создании распараллеливающих компиляторов на вычислительные системы с распределенной памятью. В данный момент таких автоматически распараллеливающих компиляторов не существует. В [1] отмечены проблемы создания таких компиляторов. В этой же работе описывается генерация

параллельного кода с генерацией коммуникаций, основанных на MPI и используется полиэдральная распараллеливающая система Pluto. Автоматическое распараллеливание развито только для вычислительных систем с общей памятью (см., например, [19], [33], [34], [35]).

Для вычислительных систем с распределенной памятью (ВСРП) самой длительной операцией является пересылка данных между процессорными элементами (ПЭ). Для высокопроизводительных кластеров такие пересылки могут погасить ускорение от распараллеливания и даже вызвать замедление. Для эффективного отображения на ВСРП программа должна удовлетворять таким жестким требованиям, что ускорение может быть достигнуто только для очень узкого класса программ, и разработка компилятора теряет смысл. Но в последнее время появляются многоядерные процессоры, иногда называемые «суперкомпьютер на кристалле», с десятками, сотнями и тысячами ядер, большинство из которых ориентированы на реализацию нейронных сетей [2], [3], [4], [25], [26], [27], [28], [29], [30]. Пересылка данных между процессорными ядрами на одной микросхеме требует значительно меньше времени, чем на коммуникационной сети (Ethernet, Infiniband, PCI-express…). Это означает расширение множества эффективно распараллеливаемых программ и делает целесообразным разработку распараллеливающих компиляторов.

Многие системы автоматизации создания параллельного кода для ВСРП предполагают дописывание прагм в текст последовательной программы без предварительного преобразования [5], [6], [7].

Перспективность циклических пересылок данных для микросхем с тысячами ядер отмечена в [9]. В [10] приводится много задач линейной алгебры и математической физики, для параллельного решения которых на ВСРП используются циклические пересылки. Метод размещения данных с перекрытиями [11] существенно ускоряет параллельные итерационные алгоритмы, уменьшая количество пересылок при укрупнении множеств пересылаемых данных. В [12] показан дополнительный эффект ускорения от размещений с перекрытиями для микросхем «суперкомпьютер на кристалле». В [13], [14] описаны блочно-аффинные размещения данных в распределенной памяти.

В [15] рассмотрена задача распараллеливания программного цикла на ВСРП, для минимизации межпроцессорных пересылок по тексту программы строится вспомогательный граф «операторы-переменные» и приводятся примеры реализации на основе ОРС [20].

Промышленные распараллеливающие компиляторы (GCC, ICC, MS-Compiler, LLVM) распараллеливают программы только для вычислительных систем с общей памятью (ВСОП). В [16] показано, что эти компиляторы плохо оптимизируют код, имеют большой неиспользованный потенциал оптимизирующих преобразований. Эти результаты частично подтверждаются в [17], [18]. Можно полагать, что для микросхем supercomputer-on-chip этот потенциал оптимизаций значительно больше.

Микросхемы «суперкомпьютер-на-кристалле» будут иметь большую производительность, чем многоядерные с общей памятью, и смогут выполнять новые классы программ с большим объемом вычислений. Отсутствие компиляторов на ВСРП сдерживает развитие новых высокопроизводительных систем.

В данной работе формируются и обосновываются рекомендации к созданию оптимизирующих распараллеливающих компиляторов с последовательных программ языков высокого уровня для ВСРП, в том числе, приводятся оптимизирующие преобразования программ, которые должны использоваться в таком компиляторе.

2. Модель вычислительной системы.

Будем рассматривать параллельную вычислительную систему с распределенной памятью (ВСРП), состоящую из процессоров и модулей памяти, которые соединяются коммуникационной системой, по которой можно пересылать данные. Пересылки данных по коммуникационной сети требует много времени и составляют основную задачу оптимизации программ на ВСРП.

Существует множество конкретных вычислительных систем, которые удовлетворяют условиям описанной абстрактной модели. Многообразие таких вычислительных систем определяется разнообразием коммуникационных систем и разнообразием вычислительных элементов и модулей памяти. ВСРП отличаются не только количеством модулей памяти и вычислительных элементов, но и их связями посредством коммуникационной сети, которые по длительности могут быть несимметричными. Распределенная память входит в состав вычислительных систем с программируемой архитектурой [31]. Как только будет разработан новый высокопроизводительный компьютер относительно небольших габаритов, сразу появится желание для некоторых важных задач объединить несколько таких компьютеров сетью – и появится новая ВСРП. Еще много архитектур будет придумано.

Многообразие архитектур ВСРП создает следующее требование к архитектуре преобразователей программ. Должна быть система преобразований программ, которые полезны широкому множеству ВСРП и должны быть библиотеки преобразований, ориентированные на конкретную целевую архитектуру. Это аналогично тому как устроены семейства компиляторов GCC и LLVM: есть библиотека оптимизирующих преобразований программ в промежуточном представлении (middle end) и есть генераторы кода на целевую архитектуру (back end), которые тоже содержат оптимизирующие преобразования.

3. Специфические для ВСРП оптимизирующие преобразования программ.

Отображение последовательных программ на ВСРП требует разработки дополнительных функций, которых нет в ВСОП, например: построение вспомогательного графа «операторы-переменные» для поиска минимального множества пересылок, генерация межпроцессорных пересылок, группировка пересылок, поиск оптимального размещение данных, поиск кратного оптимального размещения данных в распределенной памяти с перекрытиями.

Оптимизирующие преобразования для ВСРП направлены, в первую очередь, на минимизацию пересылок между модулями системы. Можно выделить следующие функции и преобразования:

1. Размещение данных в распределенной памяти [13], [14].
2. Поиск оптимального множества циклических пересылок с помощью ГОП [15].
3. Транспонирование матриц, размещенных в распределенной памяти и его обобщение на многомерные массивы [14].
4. Размещение массивов данных кратными с перекрытиями [11].
5. Группировка пересылок.
6. Определение оптимального количества ПЭ [14].
7. Оптимизация преобразований многомерных массивов: транспонирование и скашивание [23].

К специфическим для ВСРП преобразованиям можно отнести распознавание и вызов пересылок данных для многомерных массивов, таких как транспонирование и скашивание.

Транспонирование матриц используется и в библиотеках прикладных программ для обычных процессоров, поскольку от расположения элементов матрицы в оперативной памяти зависит скорость их считывания. Транспонирование может быть описано простой программой

```
for (int i = 0; i < n; i++) {
   for (int j = 0; j < n; j++) {
   Y(i,j) = X(j,i);
} }
```

В ВСРП транспонирование матрицы может быть полезно, если в одной части программы матрица должна быть размещена в распределенной памяти по строкам, а в другой – по столбцам. Если размерность матрицы равна количеству ПЭ и коммуникационная система поддерживает циклические пересылки данных, то транспонирование сводится к одновременным пересылкам диагональных элементов.

При скошенной форме хранения в ПЭ хранятся косые диагонали матрицы (диагонали, перпендикулярные к главной) [10]. Скошенная форма хранения позволяет обращаться одновременно как к элементам любой строки матрицы-клетки, так и к элементам любого столбца. Алгоритм приведения матрицы к скошенной форме может выглядеть следующим образом.

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
    Y(i,j) = X((j+i) mod n ,i);
} }
```

4. Оптимизирующие преобразования программ для ВСРП, которые полезны и для ВСОП.

В данной главе рассматриваются предварительные преобразования последовательных программ перед отображением на вычислительную систему, которая может иметь как распределенную память, так и общую.

Оптимизировать следует участки кода, требующие больших объемов вычислений. Такими участками являются структуры вложенных циклов или рекурсивные функции. Обычные компиляторы LLVM, GCC, ICC, MS-compiler хорошо оптимизируют базовые блоки и самые глубоко вложенные циклы [23]. Такая оптимизация ориентирована на использование регистров, АЛУ и векторизацию. Но в последнее время в архитектуре компьютеров появилась сложная структура памяти даже в ВСОП (распределенная память ВСРП и так непосильна для компиляторов сегодня). Для оптимизации использования сложной структуры памяти появилась потребность в преобразованиях циклов, которые могут содержать другие циклы (в первую очередь, тайлинг и скошенный тайлинг) [38], [23], и в преобразованиях данных (блочные размещения матриц в оперативной памяти [23], блочно-аффинные и размещения массивов и размещения с перекрытиями в распределенной памяти [23]) и т.п.

К преобразованиям циклов следует отнести слияние, разбиение, гнездование, развертка, расщепление, … Это преобразования, которые традиционно выполняются для самых глубоко вложенных циклов (innermost loop), но условия эффективного применения этих преобразований к не самым глубоко вложенным циклам недостаточно разработаны и эти преобразования компиляторами не выполняются [37]. Есть описанные в литературе преобразования, которые не выполняются в компиляторах, такие как ретайминг (сдвиги операторов между итерациями циклов) [23].

Диагональное расщепление двойного гнезда циклов. Позволяет избавиться от операции взятия остатка от деления при вычислениях с целыми числами. Встречается для быстрого вычисления сверток на циклической группе при восстановлении изображений [39], в

теории кодирования и других приложениях. В компиляторах пока не реализуется. На микросхемах с распределенной локальной памятью преобразование актуально вместе с его применениями.

Не поддерживается компиляторами оптимизация обхода дерева вариантов, который возникает при рекурсивных вызовах функций, например, для многих задач, решаемых методом ветвей и границ. Нет инструментов, поддерживающих параллельный обход ветвей такого дерева [23].

Главная проблема современных оптимизирующих компиляторов на ВСОП – выбор оптимальной цепочки преобразований [23]. Для компиляторов на ВСРП эта проблема будет еще острее. Распараллеливание дерева обхода таких цепочек может быть полезно.

5. Рекомендации к созданию архитектуры распараллеливающей системы на ВСРП.

Даже для ВСОП распараллеливающий компилятор – это программный продукт большой по объему кода, обладающий сложной логикой, требующий очень высокой квалификации разработчиков и значительного времени разработки. По этой причине при разработке компилятора на новый процессор используются большие части кода уже созданных компиляторов для прежних вычислительных систем (например, оптимизирующие преобразования LLVM). Оригинальным является генератор кода и некоторые комбинации существующих преобразований. Такой подход по экономическим причинам неизбежен (с дополнительными усложнениями) и для компиляторов на ВСРП.

Специфические преобразования программ на ВСРП предполагают выделение их в отдельную библиотеку. Поскольку для многих архитектур ВСРП выход компилятора должен содержать участки кода, выполняемые на ВСОП, результатом таких преобразований должен быть, скорее всего, высокоуровневый язык, к которому можно применить существующий компилятор на ВСОП (например, ICC или C-to-CUDA). Поскольку самый быстрый код создается программами языков С и ФОРТРАН, то на эти языки и должна быть рассчитана распараллеливающая система на ВСРП. Такая распараллеливающая система будет совместима со всеми компиляторами. Даже для ВСОП преобразования гнезд циклов удобнее выполнять с выходным высокоуровневым языком [36].

Многообразие архитектур ВСРП предполагает создание, в первую очередь, преобразований программ полезных для многих ВСРП: размещение данных в распределенной памяти (блочно-аффинные), алгоритмы поиска минимального множества циклических пересылок и рассылок данных (broadcast).

Внутреннее (промежуточное) представление обсуждаемой распараллеливающей системы желательно делать высокоуровневым. Высокоуровневое внутреннее представление позволит создавать выходной код более близкий (узнаваемый) к исходному коду разработчика последовательной программы. К такому коду легче применять диалоговый режим уточнения зависимостей [32]. Известные распараллеливающие системы, выдающие код на высокоуровневом языке, имеют, как правило, высокоуровневое внутреннее представление: ROSE, SUIF, OPS, PLUTO.

6. Оптимизирующие компиляторы или библиотеки?

Разработчики процессоров ВСОП в качестве системного ПО выпускают компиляторы и библиотеки. Для создания высокопроизводительных приложений следует использовать библиотеки. Но библиотеки разрабатываются не для всех приложений. При выпуске новой микросхемы необходимо все библиотеки переписывать.

Для ВСРП создание библиотек требует существенно больших усилий, чем для ВСОП, поскольку исходные данные для библиотечных функций в разных случаях могут быть размещены по-разному. Рассмотрим, например, задачу перемножения матриц C= A*B. Если размерность исходных матриц равна количеству ПЭ, то в рассматриваемых на практике случаях каждая матрица может размещаться либо «по строкам», либо «по столбцам», либо «в скошенной форме». В этом случае нужно 9 библиотечных функций. Но, если размерности матриц больше, чем количество ПЭ, то матрицы могут размещаться полосами строк или полосами столбцов, что увеличивает количество библиотечных функций.

Если ВСРП используется как ускоритель некоторого класса приложений, то библиотеки могут иметь преимущество по сравнению с компилятором. Если ВСРП предполагается использовать для переноса программ с других вычислительных систем или для разработки новых приложений, то использование компилятора более предпочтительно.

Есть еще одна функция для использования высокоуровневого преобразователя программ – проектирование новых вычислительных систем. Действительно, новые вычислительные системы при проектировании подстраиваются под некоторые бенчмарки. Бенчмарки – это конкретные программные реализации некоторых полезных алгоритмов. Но алгоритмы могут быть реализованы не единственным образом. Высокоуровневый преобразователь программ может представить семейство программ, эквивалентных исходной (бенчмарку). Проектируемая вычислительная система может ориентироваться на любой экземпляр семейства.

7. Заключение

В данной статье приводятся основные элементы проекта системы преобразования программ для создания распараллеливающих компиляторов на вычислительные системы с распределенной памятью.

Существует много приложений, для которых для которых не видно предела в повышении производительности вычислительных систем: прогноз погоды и изменений климата, отслеживание угроз столкновения метеоритов с Землей, планирование экономики, создание новых лекарств, решение задач метагеномики и др. Потребность в новых высокопроизводительных программно-аппаратных комплексах будет у общества еще многие годы. Дорогой проект создания высокоуровневого преобразователя программ для ВСРП является вполне экономически целесообразным.

8. Литература

1. Bondhugula U. Automatic distributed-memory parallelization and code generation using the polyhedral framework // Technical report, ISc-CSA-TR-2011-3, 2011, 10 pp. URL: http://mcl.csa.iisc.ac.in/downloads/publications/uday11distmem-tr.pdf
2. SoC Esperanto. — URL: https://www.esperanto.ai/technology/ (Accessed 26.03.2022).
3. Процессор НТЦ «Модуль»: https://www.module.ru/products/1/25-18796-nm6407 (Дата обращения 12.07.2023).
4. Peckham O. SambaNova Launches Second-Gen DataScale System. URL: https://www.hpcwire.com/2022/09/14/sambanova-launches-second-gen-datascalesystem/ (дата обращения 20.01.2023).
5. DVM-система разработки параллельных программ. — URL: http://dvm-system.org/ru/about/ (Дата обращения 26.03.2022).
6. Kataev, N., Kolganov, A. (2021). Additional Parallelization of Existing MPI Programs Using SAPFOR. In: Malyshkin, V. (eds) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science, vol 12942. Springer, Cham. https://doi.org/10.1007/978-3-030-86359-3_4
7. Kwon D., Han S., Kim H. MPI backend for an automatic parallelizing compiler // Proceedings Fourth International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99). — 06.1999. — pp. 152–157. — DOI: 10.1109/ISPAN.1999. 778932.
8. Optimizing parallelizing system www.ops.rsu.ru (Дата обращения 08.01.2021)
9. Корнеев В.В. Параллельное программирование // Программная инженерия. 2022, т. 13, № 1, с. 3–16.
10. И. В. Прангишвили, С. Я. Виленкин, И. Л. Медведев. Параллельные вычислительные системы с общим управлением. М., Энергоатомиздат, 1983, 312 с.
11. Ammaev S., Gervich L., Steinberg B. Combining parallelization with overlaps and optimization of cache memory usage // International Conference on Parallel Computing Technologies. _ Springer. 2017. _ P. 257–264. DOI: 10.1007/978-3-319-62932-2-24.
12. Gervich L.R., Steinberg B.Ya Automation of the Application of Data Distribution with Overlapping in Distributed Memory // Bulletin of the South Ural State University. Ser. Mathematical Modelling, Programming & Computer Software (Bulletin SUSU MMCS), 2023, vol. 16, no. 1, pp. 59–68

13. Штейнберг Б. Я. Блочно-аффинные размещения данных в параллельной памяти¿, Информационные технологии, 2010, №6, с. 36–41
14. Штейнберг Б. Я. Оптимизация размещения данных в параллельной памяти, Ростов-на-Дону, Изд-во Южного федерального университета, 2010, ISBN 978-5-9275-0687-3, 255 с.
15. Krivosheev N.M., Steinberg B.Ya. Algorithm for searching minimum inter-node data transfers. // Procedia Computer Science, 10th International Young Scientist Conference on Computational Science, YSC 2021, 1-3 July 2021, pp. 306-313.
16. Gong Z., Chen Z., Szaday Z., Wong D., Sura Z., Watkinson N., Maleki S., Padua D., Veidenbaum A., Nicolau A. // An empirical study of the effect of source-level loop transformations on compiler stability / Proceedings of the ACM on Programming Languages. — 11. 2018, pp. 1–29.
17. Steinberg B.Ya., Steinberg O.B., Oganesyan P.A., Vasilenko A.A., V.V. Veselovskiy, Zhivykh N.A. Fast Solvers for Systems of Linear Equations with Block-Band Matrices // East Asian Journal on Applied Mathematics 2023, Vol. 13, No. 1, pp. 47–58 doi: 10.4208/eajam
18. Vasilenko, A., Veselovskiy, V., Metelitsa, E., Zhivykh, N., Steinberg, B., Steinberg, O. (2021). Precompiler for the ACELAN-COMPOS Package Solvers. In: Malyshkin, V. (eds) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science(), vol 12942, pp. 103–116. Springer, Cham. https://doi.org/10.1007/978-3-030-86359-3_8
19. Векторизация программ // Векторизация программ: теория, методы, реализация. / Сборник переводов статей М.: Мир, 1991. с. 246–267.
20. Харари Ф. Теория графов. М.: «Мир», 1973, 300 с.
21. Allen, R., Kennedy, K.: Optimizing Compilers for Mordern Architetures, p. 790. Morgan Kaufmann Publisher, Academic Press, USA (2002).
22. Lamport L. The parallel execution of DO loops // Commun. ACM. - 1974.- v.17, N 2, p. 83–93.
23. Штейнберг Б. Я., Штейнберг О. Б. Преобразования программ – фундаментальная основа создания оптимизирующих распараллеливающих компиляторов // Программные системы: теория и приложения, 2021,12:1(48), с. 21–113. DOI:10.25209/2079-3316-2021-12-1-21-113 URL:http://psta.psiras.ru/read/psta2021_1_21-113.pdf
24. Nvidia compilers https://developer.nvidia.com/hpc-compilers
25. Елизаров Г.С., Конотопцев В.Н., Корнеев В.В. Специализированные большие интегральные схемы для реализации нейросетевого вывода. XXII международная конференция "Харитоновские тематические научные чтения”. "Суперкомпьютерное моделирование и искусственный интеллект": труды / Редактор Р. М. Шагалиев. – Саров: ФГУП "РФЯЦ-ВНИЭФ", 2022. pp.181–184.

26. Корнеев В.В. Направления повышения производительности нейросетевых вычислений // Программная инженерия, 2020, т. 11, № 1, с. 21–25. DOI: 10.17587/prin.11.21-25.
27. Yen I.E., Xiao Zh., Xu D. S4: a High-sparsity, High-performance AI Accelerator // arXiv:2207.08006v1 [cs.AR] 16 Jul 2022
28. Gale T., Elsen E., Hooker S. The state of sparsity in deep neural networks // arXiv preprint arXiv:1902.09574, 2019
29. Intelligence Processing Unit. https://www.graphcore.ai/products/ipu. (accessed: 20.01.2023)
30. Jia Zh., Tillman B., Maggioni M., Scarpazza D.P. Dissecting the Graphcore IPU Architecture via Microbenchmarking // Technical Report. December 7, 2019. arXiv:1912.03413v1 [cs.DC] 7 Dec 2019. 91 p.
31. Dordopulo A.I., Levin I.I., Gudkov V.A., Gulenok A.A. (2021). High-Level Synthesis of Scalable Solutions from C-Programs for Reconfigurable Computer Systems. In: Malyshkin, V. (eds) Parallel Computing Technologies. PaCT 2021. Lecture Notes in Computer Science, vol 12942. Springer, Cham. https://doi.org/10.1007/978-3-030-86359-3_7
32. Gervich L.R., Guda S.A., Dubrov D.V., Ibragimov R.A., Metelitsa E.A., Mikhailuts Y.M., Paterikin A.E., Petrenko V.V., Skapenko I.R., Steinberg B.Ya., Steinberg O.B., Yakovlev V.A., Yurushkin M.V., How OPS (Optimizing Parallelizing System) May be Useful for Clang // CEE-SECR '2017, October 20-21, 2017, St.-Peterburg, Russian Federation. Proceedings of the 13th Central & Eastern European Software Engineering Conference in Russia ACM New York, NY, USA ©2017 https://dl.acm.org/citation.cfm?id=3166094&picked=prox
33. Moldovanova O.V., Kurnosov M.G. Auto-Vectorization of Loops on Intel 64 and Intel Xeon Phi: Analysis and Evaluation International Conference on Parallel Computing Technologies PaCT 2017: Parallel Computing Technologies pp 143–150.
34. Nvidia compilers https://developer.nvidia.com/hpc-compiler
35. Peng Di, Ding Ye, Yu Su, Yulei Sui and Jingling Xue Automatic Parallelization of Tiled Loop Nests with Enhanced Fine-Grained Parallelism on GPUs. 2012. 41st International Conference on Parallel Computing.
36. Zhiyuan Li and Yonghong Song Automatic Tiling of Iterative Stencil Loops // ACM Transactions on Programming Languages and Systems, Vol. 26, No. 6, November 2004, Pages 975–1028.
37. Штейнберг Б.Я., Штейнберг О.Б., Василенко А. А.**.** Слияние циклов для локализации данных // Программные системы. Теория и приложения. №3 (Том 11), 2020 г. DOI: https://doi.org/10.25209/2079-3316-2020-11-3-17-31
38. Wolfe, M. More Iteration Space Tiling / M. Wolfe // Supercomputing. – Reno, 1989. – P. 655–664.
39. Козак, А.В., Штейнберг Б.Я., Штейнберг О.Б. Алгоритм восстановления смазанного изображения, полученного вращающейся под углом к горизонту камерой // Компьютерная оптика. – 2020. – Т. 44, № 2. – С. 229–235. – DOI: 10.18287/2412–6179-CO-598. http://www.computeroptics.smr.ru/