ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № ?, 2014, с. ??–??

УДК 004.4'416

А. С. Лебедев

Оптимизация локальности данных при автоматическом распараллеливании программ

Аннотация. Рассматривается класс линейных программ и методы их анализа в рамках модели многогранников. Предлагается подход к нахождению функций размещения операций, нацеленный на оптимизацию локальности данных в программе, полученной в процессе автоматического распараллеливания. Реализация подхода иллюстрируется распараллеливанием алгоритма LU-разложения.

Ключевые слова и фразы: Автоматическое распараллеливание, оптимизация методами модели многогранников, локальность данных, размещение вычислений.

Введение

С увеличением количества вычислительных элементов в современных процессорах вычислительные системы приобретают все большую степень параллелизма. Это в равной степени справедливо по отношению как к высокопроизводительным рабочим станциям, так и к кластерным системам, построенным на основе вычислительных серверов, часто являющихся многопроцессорными NUMA-машинами.

Разработка вычислительного программного кода, способного эффективно задействовать возможности аппаратуры, всегда являлась сложной задачей. Сегодня одним из актуальных подходов решения данной задачи является автоматическое распараллеливание программ.

Работа поддержана грантом РФФИ в рамках проекта №14-37-50316 «Исследование и разработка методов автоматического распараллеливания программ для гетерогенных вычислительных систем и систем с распределенной памятью» и проекта по программе №18 фундаментальных исследований президиума РАН «Алгоритмы и математическое обеспечение для вычислительных систем сверхвысокой производительности».

Большинство вычислительно емких научных и инженерных приложений тратят значительную часть времени исполнения на вложенности циклов, часто отвечающие критериям линейности программ. Модель многогранников [1] предоставляет мощный математический аппарат, упрощающий анализ и преобразование таких программ с целью улучшения быстродействия путем повышения параллелизма и оптимизации локальности данных при вычислениях.

Методы модели многогранников широко используются в статической компиляции [2–6]. Код, написанный на языке высокого уровня (чаще всего это C или FORTRAN), оптимизируется для параллельного выполнения и компилируется для целевой архитектуры.

Распараллеливание в модели многогранников осуществляется в пять этапов.

(1) Анализ зависимостей операций по данным. Рассматриваются потоковые зависимости (чтение после записи), антизависимости (запись после чтения), зависимости через выход (запись после записи).
(2) Поиск для каждой операции функции расписания [5, 7, 8], сохраняющей логический порядок операций в соответствии с зависимостями по данным.
(3) Поиск для каждой операции функции размещения [9, 12], ставящей в соответствие операции индекс виртуального процессора.
(4) Разбиение индексного пространства виртуальных процессоров — сопоставление виртуальных процессоров физическим.
(5) Генерация параллельной программы по алгоритму [13] в соответствии с пространственно-временным преобразованием, заданным функциями расписания и размещения, и разбиением.

Множество техник разбиения [10–12] направлено на повышение локальности данных за счет уменьшения количества актов информационного обмена на гранях многогранников, на которые разделяется пространство виртуальных процессоров гиперплоскостями разбиения (фрагменты-многогранники обрабатываются физическими процессорами).

В настоящей работе предлагается рассмотреть оптимизацию локальности данных на более раннем этапе, чем разбиение, а именно осуществить поиск функций размещения, обладающих наперед заданным свойством.

В начале работы дан обзор основных понятий модели многогран-

ников, затем следует изложение эвристики и формализация подхода к оптимизации локальности данных при автоматическом распараллеливании программ для кластерных систем. Последующий разбор примера — распараллеливание алгоритма LU-разложения — проиллюстрирует целесообразность ограничений, наложенных на функцию размещения.

1. Модель многогранников

Основная задача параллельного программирования может быть выражена следующими положениями [7]:

- Задано множество операций Ω — множество преобразований над памятью, которые требуется выполнить.
- Порядок выполнения операций не является произвольным. Задано бинарное отношение Γ на Ω, описывающее граф зависимостей: если $u, v \in \Omega$ и $u\Gamma v$, то операция v не может быть выполнена до операции u. Напротив, если пара $\langle u, v\rangle$ не принадлежит транзитивному замыканию Γ, то операции u и v могут выполнены в любом порядке, и даже одновременно, если достаточно ресурсов для их параллельного выполнения.
- Построение параллельной программы заключается в задании частичного порядка, как можно более близкого к транзитивному замыканию Γ, и при этом достаточно просто описываемого для того, чтобы быть технически реализуемым.

На момент исследования простым и практичным способом описания параллельных программ является задание их расписания — отображения из Ω в $\mathbf{R}$, которое задает момент времени для выполнения каждой операции. Расписание θ обязательно удовлетворяет условию причинности: $u, v \in \Omega, u\Gamma v \Rightarrow \theta(v) > \theta(u)$.

Поиск расписаний в табулированном виде сопряжен со значительными вычислительными трудностями, поэтому получил распространение подход поиска функций расписания в замкнутой форме. Это ограничение приобретает смысл, когда граф зависимостей задан особой синтетической формой. Мы рассматриваем случай, когда граф зависимостей порождается последовательной программой: операции соответствуют динамическим экземплярам инструкций. Поиск функций расписания можно выполнять в виде выражений, зависящих от индексного вектора для каждой инструкции.

Модель многогранников рассматривает класс программ, для которых оба множества Ω и Γ могут быть заданы системами линейных неравенств, и полагается на поиск расписаний в аффинном виде (разработан аппарат для поиска не только одномерных расписаний, но и многомерных с аффинными компонентами). Эти программы называются линейными, и удовлетворяют следующим ограничениям:

- в программе используется любое количество простых переменных и переменных с индексами;
- единственным типом исполнительного оператора может быть оператор присваивания, правая часть которого является арифметическим выражением;
- все повторяющиеся операции описываются только с помощью циклов DO языка программирования FORTRAN (либо эквивалентными циклами for языка C); структура вложенности циклов может быть произвольной; шаги изменения параметров циклов всегда равны $+1$; если у цикла нижняя граница больше верхней, то цикл не выполняется;
- допускается использование любого количества условных и безусловных операторов перехода, передающих управление «вниз» по тексту; не допускается использование побочных выходов из циклов;
- все индексные выражения переменных, границы изменения параметров циклов и условия передачи управления задаются, в общем случае, неоднородными формами, линейными по совокупности параметров циклов и внешних переменных программы; все коэффициенты линейных форм являются целыми числами;
- внешние переменные программы всегда целочисленные; конкретные значения внешних переменных известны только во время работы программы.

Обобщенный граф зависимостей — это ориентированный мультиграф, каждая вершина которого представляет множество соответственных операций (динамических экземпляров одной и той же инструкции). Каждая операция принадлежит только одной вершине. Ребро графа представляет временное ограничение на две операции, соответствующие начальной и конечной вершине. В обобщенном графе зависимостей могут быть петли и циклы. Каждая вершина помечается описанием соответствующих операций, а каждое ребро — описанием соответствующих отношений зависимости.

Каждой вершине X соответствует ее домен — многогранник D_X на множестве $\mathbf{Q}^{p_X}$, где p_X — размерность ее итерационного пространства (количество циклов в программе, включающих X). Каждая операция представляется как $\langle i, n; X\rangle$, где $i \in D_X$ — целочисленный вектор индекса итерации, n — целочисленный вектор внешних переменных программы.

Каждому ребру e, исходящему из вершины $\sigma(e) = X$ в $\delta(e) = Y$, ставится в соответствие многогранник R_e на множестве $\mathbf{Q}^{p_X+p_Y}$ такой, что условие причинности должно быть наложено на динамические экземпляры операций $\langle i, n; X\rangle$ и $\langle j, n; Y\rangle$ тогда и только тогда, когда для составного вектора (i, j) выполняется $(i, j) \in R_e$.

Обобщенный граф зависимостей — это четверка $\langle V, E, D, R\rangle$, где V — множество вершин, E — множество ребер, D — отображение множества V на множество соответствующих доменов, R — отображение множества E на множество соответствующих многогранников зависимостей.

Уточненный граф зависимостей, ассоциированный с $\langle V, E, D, R\rangle$, — это граф со множеством вершин

$$\Omega = \bigcup_{X \in V} \{\langle i, n; X\rangle | i \in D_X\} \tag{1}$$

и множеством ребер

$$\Gamma = \bigcup_{e \in E} \{\langle (\sigma(e), x), (\delta(e), y)\rangle | x \in D_{\sigma(e)}, y \in D_{\delta(e)}, (x, y) \in R_e\} \tag{2}$$

При работе с семействами обобщенных графов зависимостей множества V и E остаются фиксированными, единственными неизвестными являются внешние переменные программы n, линейно входящие в описания многогранников D_S и R_e. Построение уточненного графа зависимостей возможно тогда и только тогда, когда значения внешних параметров программы становятся известными.

В рассматриваемом нами случае граф зависимостей порождается последовательной программой. Пусть f и g — индексные функции ячейки памяти, к которой обращаются конфликтующие операции, тогда p_e называют глубиной зависимости:

$$(x, y) \in R_e \equiv f(x) = g(y) \wedge (x[1..p_e] = y[1..p_e]) \wedge (x[p_e + 1] < y[p_e + 1]). \tag{3}$$

Расписание для заданного обобщенного графа зависимостей есть неотрицательная функция θ из Ω в $\mathbf{N_0}$ такая, что

$$\forall e \in E : x \in D_{\sigma(e)}, y \in D_{\delta(e)}, (x, y) \in R_e \Rightarrow \theta(\delta(e), y) \geq \theta(\sigma(e), x) + 1. \tag{4}$$

Множество операций $F(t) = \{u \in \Omega | \theta(u) = t\}$ называется фронтом расписания на t. Программа с синхронным параллелизмом может быть сконструирована следующим образом:

```
1  do t = 0; L
2    doall F(t)
3    barrier
4  end do
```

Здесь L — задержка расписания.

2. Поиск размещения операций с оптимизацией локальности данных

В основе подхода лежит правило «вычислитель — владелец», означающее, что владельцем значения является тот процессор, который его вычислил. Акт коммуникации происходит тогда, когда процессору требуется значение, вычисленное другим процессором. Стоимость такой коммуникации может различаться. Ситуации в порядке увеличения стоимости: обмен данными между ядрами одного многоядерного CPU, обмен данными между ядрами разных CPU в многопроцессорном сервере, обмен данными между ядрами разных CPU на разных машинах (объединенных сетью).

Предлагается следующая эвристика для распространенного на момент исследования случая, когда вычислительная система является однородным кластером многопроцессорных серверов. Пусть M — количество узлов в кластере, доступных для проведения вычислений, S — количество процессоров в каждом узле, C — количество ядер в каждом процессоре. Занумеруем ядра физических процессоров последовательно начиная с нуля: пусть $N = mSC + sC + c$ — глобальный номер ядра (физического процессора в терминах модели многогранников), где $m = 0 \ldots (M - 1)$ — номер узла, $s = 0 \ldots (S - 1)$

— номер процессора внутри узла, $c = 0 \ldots (C-1)$ — номер ядра процессора. Зафиксируем разбиение одномерного пространства виртуальных процессоров: пусть $N(V) = P \mod MSC$, где P — номер виртуального процессора, mod — операция взятия остатка при делении целых чисел. Будем искать аффинную функцию размещения, ограниченную снизу значением 0.

Для каждой зависимости по данным $d_{X \to Y}$ установим ограничение: индексы виртуальных процессоров операций $\langle i, n; X \rangle$ и $\langle j, n; Y \rangle$ не отличаются больше, чем на заданное положительное число $\rho_{d_{X \to Y}}$. Такое ограничение позволяет управлять расстоянием использования данных в пространстве физических процессоров благодаря зафиксированному разбиению. Положительное число $\rho_{d_{X \to Y}}$ выступает параметром в следующей задаче, где может быть задано в соответствии с характеристиками S и C кластера, чтобы увеличить вероятность возникновения «ближних» (недорогих) коммуникаций:

$$d_{X \to Y} : \{\langle i, n; X \rangle \longrightarrow \langle j, n; Y \rangle : R_{d_{X \to Y}} \begin{pmatrix} i \\ j \\ n \end{pmatrix} \geq 0\}$$

$R_{d_{X \to Y}}$ — многогранник зависимостей инструкций X и Y.

$$\forall i, j : R_{d_{X \to Y}} \begin{pmatrix} i \\ j \\ n \end{pmatrix} \geq 0 \Rightarrow 0 \leq \Pi_Y \begin{pmatrix} j \\ n \end{pmatrix} - \Pi_X \begin{pmatrix} i \\ n \end{pmatrix} \leq \rho_{d_{X \to Y}}$$

Если не требуется соблюдения правила «коммуникации только вперед» (FCO-placements [12]), то ограничение может принять вид:

$$\forall i, j : R_{d_{X \to Y}} \begin{pmatrix} i \\ j \\ n \end{pmatrix} \geq 0 \Rightarrow -\rho^*_{d_{X \to Y}} \leq \Pi_Y \begin{pmatrix} j \\ n \end{pmatrix} - \Pi_X \begin{pmatrix} i \\ n \end{pmatrix} \leq \rho^*_{d_{X \to Y}}$$

Система ограничений, сформированная при рассмотрении каждой зависимости по данным, может быть сведена к линейной после применения классической техники на основе леммы Фаркаша [7]. Затем, с помощью библиотеки polylib [14] могут быть найдены лучи и линии, образующие конус решений линейной системы.

Для каждой инструкции T выписывается шаблон функции размещения операций $\langle i, n; T \rangle$, неотрицательной внутри ее домена:

$$\Pi_T \begin{pmatrix} i \\ n \end{pmatrix} = \mu_{T,0} + \sum_{k=1}^{q_T} \mu_{T,k} (a_{T,k} \cdot \begin{pmatrix} i \\ n \end{pmatrix} + b_{T,k}), \tag{5}$$

где $\mu_{T,k}$ — множители Фаркаша, а q_T неравенств $a_{T,k} \cdot \begin{pmatrix} i \\ n \end{pmatrix} + b_{T,k} \geq 0$ определяют домен D_T.

Для каждого неравенства $\Pi_{X_1} \begin{pmatrix} x_1 \\ n \end{pmatrix} - \Pi_{X_2} \begin{pmatrix} x_2 \\ n \end{pmatrix} - \Delta \geq 0$, порождаемого наложенным ограничением на расстояние использования данных относительно зависимости $d_{X \to Y}$, выписывается тождество:

$$\Pi_{X_1} \begin{pmatrix} x_1 \\ n \end{pmatrix} - \Pi_{X_2} \begin{pmatrix} x_2 \\ n \end{pmatrix} - \Delta \equiv \lambda_{R,0} + \sum_{k=1}^{q_R} \lambda_{R,k} (c_{R,k} \cdot \begin{pmatrix} x_1 \\ x_2 \\ n \end{pmatrix} + d_{R,k}), \tag{6}$$

где $\lambda_{R,k}$ — множители Фаркаша, а q_R неравенств $c_{R,k} \cdot \begin{pmatrix} x_1 \\ x_2 \\ n \end{pmatrix} + d_{R,k} \geq 0$ описывают многогранник зависимостей $R_{d_{X \to Y}}$.

После раскрытия скобок и приравнивания соответственных множителей при индексах итераций и параметрах программы получим линейную систему ограничений, где в качестве неизвестных останутся множители Фаркаша (с ограничением неотрицательности).

Лучшим решением с точки зрения стоимости коммуникаций будем считать то, которое минимизирует объем дальних коммуникаций (между физическими процессорами), обусловленных потоковыми зависимостями. Пусть $F\left(R_{d_{X \to Y}}, \Pi_X, \Pi_Y\right)$ — количество дальних коммуникаций для потоковой зависимости $d_{X \to Y}$ и размещений Π_X и Π_Y, а $w_{d_{X \to Y}}$ — размер пересылаемого элемента данных в байтах. Тогда функция стоимости примет вид:

$$W = \sum\nolimits_{d_X \xrightarrow[flow]{} Y} w_{d_{X \to Y}} \cdot F\left(R_{d_{X \to Y}}, \Pi_X, \Pi_Y\right). \tag{7}$$

Далее будем предполагать, что значения всех параметров модели являются либо числовыми константами, и тогда дальнейшие рассуждения будут применимы для статической компиляции, либо являются символьными константами, получающими значения во время исполнения программы, и тогда дальнейшие рассуждения будут применимы в случае Just-In-Time компиляции. Благодаря такому предпо-

ложению об ослаблении параметризации программы становится возможной оценка наибольшего индекса виртуального процессора.

Разделим индексное пространство виртуальных процессоров на интервалы длиной C. Согласно заданному разбиению виртуальные процессоры одного интервала будут соответствовать ядрам одного многоядерного процессора. Коммуникация будет считаться дальней, если для зависимости $d_X \xrightarrow[flow]{} Y$ значения Π_X и Π_Y для динамических экземпляров инструкций попадут в разные интервалы. Значение $F\left(d_{X\to Y}, \Pi_X, \Pi_Y\right)$ дает количество таких случаев.

Добавляя к описанию многогранника $R_{d_{X\to Y}}$ неравенства, ограничивающие Π_X и Π_Y в конкретных интервалах, получаем описание многогранника, пересечение которого с целочисленной решеткой дает однородное множество актов коммуникаций (ближних или дальних). Многогранников, характеризующих только ближние коммуникации, в общем случае меньше, чем характеризующих только дальние, по комбинаторным соображениям. Поэтому для подсчета актов дальних коммуникаций целесообразно найти количество актов ближних коммуникаций (общее количество точек с целочисленными координатами во всех многогранниках, определяемых условием попадания значений функций размещения в один интервал) и вычесть его из общего количества коммуникаций (количество точек с целочисленными координатами в многограннике $R_{d_{X\to Y}}$).

Подсчет количества точек с целочисленными координатами внутри многогранника может быть выполнен с помощью алгоритма [14], реализованного в библиотеке polylib.

3. Распараллеливание LU-разложения

3.1. Результаты экспериментов на NUMA-сервере

3.2. Результаты экспериментов на кластере с SMP-узлами

3.3. Результаты экспериментов на графическом ускорителе

Рассмотрим мультипроцессор графического ускорителя как многоядерный процессор в терминах раздела 2, а сам ускоритель как вычислительный узел. В таком случае предложенная эвристика оказывается применимой, поскольку ближние коммуникации могут быть реализованы с привлечением разделяемой памяти, которая значительно превосходит динамическую память по быстродействию.

4. Заключение

Список литературы

[1] C. Lengauer. *Loop Parallelization in the Polytope Model* // 4th Intrenational Conference on Concurrency Theory Proceedings. Lecture Notes in Computer Science: Springer-Verlag, 1993. Vol. **715**, p. 398–416. ↑2

[2] F. Irigoin, P. Jouvelot, R. Triolet. *Semantical interprocedural parallelization: An overview of the PIPS project* // Proceedings of the 5th international conference on Supercomputing: ACM, 1991. ↑2

[3] S. I. Lee, T. A. Johnson, R. Eigenmann. *Cetus — an extensible compiler infrastructure for source-to-source transformation* // Languages and Compilers for Parallel Computing: Springer Berlin Heidelberg, 2004, p. 539–553. ↑2

[4] M. Griebl, C. Lengauer. *The loop parallelizer LooPo* // Proc. Sixth Workshop on Compilers for Parallel Computers, Konferenzen des Forschungszentrums. — Jülich, 1996. Vol. **21**. ↑2

[5] U. Bondhugula, et al. *Automatic transformations for communication-minimized parallelization and locality optimization in the polyhedral model* // Compiler Construction. Springer Berlin Heidelberg, 2008, p. 132–146. ↑2

[6] T. Grosser, et al. *Polly — Polyhedral optimization in LLVM* // Proceedings of the First International Workshop on Polyhedral Compilation Techniques (IMPACT), 2011. Vol. **2011**. ↑2

[7] P. Feautrier. *Some efficient solutions to the affine scheduling problem. I. One-dimensional time* // International journal of parallel programming, 1992. Vol. **21**, no. 5, p. 313–347. ↑2, 3, 7

[8] P. Feautrier. *Some efficient solutions to the affine scheduling problem. II. Multidimensional time* // International journal of parallel programming, 1992. Vol. **21**, no. 6, p. 389–420. ↑2

[9] P. Feautrier. *Toward automatic distribution* // Parallel Processing Letters, 1996. Vol. **4**, no. 03, p. 233–244. ↑2

[10] M. Wolf, M. Lam. *A data locality optimizing algorithm* // ACM Sigplan Notices. Vol. **26**, no. 6, p. 30–44. ↑2

[11] U. Bondhugula. *Automatic distributed memory code generation using the polyhedral framework* Indian Institute of Science, Indian Institute of Science, 2011, no. IISc-CSA-TR-2011-3. ↑2

[12] M. Griebl. *Automatic Parallelization of Loop Programs for Distributed Memory Architectures*, Department of Mathematics and Informatics, University of Passau, (2004). ↑2, 7

[13] C. Bastoul. *Code generation in the polyhedral model is easier than you think* // Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques: IEEE Computer Society, 2004. ↑2

[14] V. Loechner. *PolyLib: A library for manipulating parameterized polyhedra*, 1999. ↑7, 9

Об авторе:

Артем Сергеевич Лебедев

Область научных интересов: параллельное программирование, автоматическое распараллеливание программ, системы трансляции и оптимизации программного кода, программирование параллельных вычислительных архитектур с использованием низкоуровневых механизмов оптимизации.

e-mail: tementy@gmail.com

Образец ссылки на эту публикацию:

А. С. Лебедев. *Оптимизация локальности данных при автоматическом распараллеливании программ* // Программные системы: теория и приложения: электрон. научн. журн. 2014. Т. ??, № ?, с. ??–??.

URL: http://psta.psiras.ru/read/

Artem Lebedev. *Data locality optimization during automatic code parallelization.*

Abstract. A class of linear programs is considered in terms of polyhedron model and its analysis methods. An approach to find appropriate computations' placement function to optimize data locality of automatically parallelized program is discussed. Implementation of the approach is illustrated with parallelization of LU-decomposition algorithm. (*in Russian*).

Key Words and Phrases: Automatic parallelization, polyhedral optimizations, data locality, placement of computations.