В ходе решения второй задачи следует учитывать возможность растекания теплоносителя из повреждения вдоль канала в обе стороны от утечки.

Интересно применение данного метода поиска повреждений на участках временно выведенных из эксплуатации. В этих случаях формируются 2 графика распределения токов утечек - перед заполнением трубопровода и сразу же после заполнения. Если графики различаются незначительно, следует повторить измерения на заполненном трубопроводе, поскольку при малой утечке для образования токопроводящего слоя между трубой и грунтом нужно значительное время. Различия в графиках следует рассматривать как влияние увлажнения в области утечки.

Поступила 21.02.2011р.

УДК 004.415.2

А.Д.Смирнов, аспирант, ИПМЭ им. Г.Е.Пухова НАНУ, г.Киев
А.А.Чемерис, к.т.н., ИПМЭ им. Г.Е.Пухова НАНУ, г.Киев

ТРАНСФОРМАЦИЯ ПРОГРАММ С АФФИННЫМ ДОСТУПОМ К ДАННЫМ

The basic approaches of parallelization and optimization software for high-level languages are examined. Analyzed access to the data arrays whose indices are linear expressions of the indexes of cycles. Provided advantages and drawbacks of methods that are generally used in software.

Введение. На текущий момент многоядерные процессоры являются основным направлением развития процессорных технологий. Сложность программирования для таких архитектур с максимальным использованием их потенциала хорошо известна. Существует три основных подхода к обеспечению эффективной эксплуатации суперкомпьютеров как машин с параллельной архитектурой: использование параллельных языков, использование программных библиотек и автоматическое распараллеливание программ. Все три направления интенсивно развиваются, но наиболее привлекательным остается третий подход, поскольку решает проблему переносимости старых последовательных программ на суперкомпьютеры.

Несмотря на то, что автоматическое распараллеливание программ — достаточно трудная задача, существующее ее решение обеспечило коммерческий успех суперкомпьютеров на быстро меняющемся рынке вычислительной техники. В настоящее время по грубой оценке в мире эксплуатируется порядка тысячи суперкомпьютеров различных

параллельных архитектур с производительностью, измеряемой в гигафлопах и даже выше.

Основная задача распараллеливающего компилятора — извлечь как можно больше скрытого параллелизма из участков повторяемости последовательной программы, определяющих основное время ее выполнения: циклов и рекурсивных процедур. Множество программ проводят свое основное время во вложенных циклах. В основном, это инженерные и научные приложения. Алгоритмы распараллеливания циклов - это весьма важный класс реструктурирующих преобразований. Они особенно привлекательны тем, что их применение не требует знания целевой архитектуры. Эти алгоритмы можно рассматривать как завершающую стадию машинно-независимой части процесса генерации параллельного кода распараллеливающим компилятором. Распараллеливание циклов позволяет обнаружить параллелизм и выявить те зависимости, которые ответственны за изначально «последовательное» состояние некоторых операций исходной программы.

Существующие алгоритмы распараллеливания циклов различаются по многим направлениям. Во-первых, они используют различные математические инструменты: графовые алгоритмы, матричные вычисления, линейное программирование. Во-вторых, они принимают на входе различные описания зависимостей по данным: графовое описание и уровни зависимостей, векторы направления, описание зависимостей в виде многогранников или аффинных выражений.

Данная работа посвящена изучению различных алгоритмов обнаружения параллелизма (в гнездах циклов), основанных на многомерном планировании, принимающих на вход описание зависимостей в виде многогранников или аффинных выражений. Рассмотренные ниже алгоритмы, среди представителей своего класса, являются намного более мощными при наличии анализа зависимостей. Однако область их применения ограничена статически управляемыми программами с аффинными зависимостями. Статические управляемые программы – это программы, у которых границы циклов и индексы массивов являются аффинными функциями.

Полигональная модель. Основу для входных данных представляет собой полигональная модель – мощная абстрактная модель для преобразований гнезд циклов, в которой каждая итерация всех операторов в цикле представлена как целочисленная точка внутри вполне определенного пространства, называемого многогранником операторов [5, 6]. С помощью этого представления для каждого оператора и точной характеристики внешних и внутренних зависимостей возможно сделать вывод о правомерности сложной трансформации цикла только математическими методами, полагаясь на методы линейной алгебры и целочисленное программирование. В конечном итоге, трансформации отображаются в сгенерированном коде, который имеет улучшенную локальность данных и

и/или циклы которые были измененые для параллельного выполнения. Полигональная модель применима для циклов, в которых обращения к данным являются аффинными функциями (линейная функция с константой) ограничивающих цикл переменных и параметров. И, хотя точная характеристика зависимостей данных доступна только для программ со статическим управлением и аффинными обращениями и границами циклов, код не с аффинным доступом к массиву данных или с динамическим управлением тоже может быть обработан, но только с введением дополнительных ограничений касательно некоторых зависимостей.

Многогранники в полигональной модели могут быть представлены как расписания, распределения или разбиения, либо любые другие асбстракции удобные для решения задачи. Делая данное допущение, предполагается, что новые циклы будут удовлетворять прежним условиям и в преобразованном политопе. Основываясь на свойствах гиперплоскостей на разных уровнях, возможно применение таких преобразований циклов как разбиение, свертывание/развертывание, обозначение возможности параллельного выполнения.

Основные работы, основанные на представлении гиперплоскостей как расписаний и распределений, принадлежат Фотрие [3,4], в то время как Лим и Лам[1] представляют их в виде пространств и временных разбиений.

Расписания и распределения. Расписание определяет момент, когда должна выполняться итерация. Расписание может определить для каждой итерации временную точку, и кодогенератор сможет создать код, который будет сканировать эти точки в определенном порядке. В данном контексте расписания подразумевают аффинный доступ, поскольку генерация кода в данных случаях была очень хорошо изучена. Поэтому, в расписании неявно подразумевается последовательное расположение временных точек.

$$\theta_{S_j}(\vec{t}) - \theta_{S_i}(\vec{s}) \geq 1,\ \left\langle \vec{s}, \vec{t} \right\rangle \in P_e s_i - s_j \qquad (1)$$

Несколько таких θ с установленным порядком определяют многомерное расписание. Распределения определяют где будут выполнены итерации. Оно также неявно параллельно и может быть всегда разбито, то есть группу итераций можно назначить для выполнения на конкретном процессоре, поскольку все, что выполняется в одной и той же временной точке независимо друго от друга.

Если бы коммуникационные затраты были равны нулю или ничтожно малы по сравнению с затратами на вычисления и не было бы разных уровней памяти использование оптимальных или близких к оптимальным аффинных расписаний было бы решением. К сожалению, современные вычислительные архитектуры не подпадают под данное описание. Поэтому, необходимо уменьшение коммуникационных затрат и улучшение локальности в памяти с помощью разбиения циклов. Для пространства итераций, изображенном на рис. 2, аффинное расписание определяется формулой:

$$\theta\begin{pmatrix} i \\ j \end{pmatrix} = 2 * i + j \qquad (2)$$

Этот подход имеет несколько ограничений. Использование расписаний и распределений не подходит для описания внешнего параллелизма и разбиения циклов, поскольку расписания и распределения прямо подразумевают внешний последовательный порядок и внутренние параллельные циклы, другим словом они идут рука об руку с внутренним параллелизмом (рис. 1).

Внутренние циклы всегда могут быть без труда разделены(пространственно разделение), но они не могут дать достаточного уровня параллелизма. Одним из решений этой проблемы является распределения которые минимизируют передачу данных, хотя, это может все еще требовать высокого уровня сихронизации или слишком частой синхронизации.

```
for (t1=0; i<N; i++) {
    for (t2=0; i<N; i++) {
        forall (p1=0; i<N; i++) {
            forall (p2=0; i<N; i++) {
                S1
            }
        }
    <barrier>;
    }
}
```

Рис. 1. Типичное решение с использованием расписаний и распределений

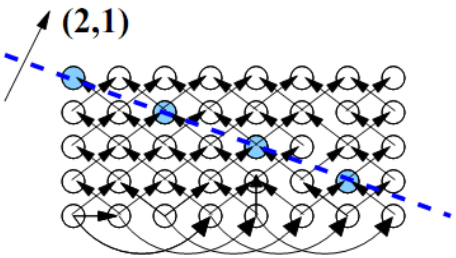

Рис. 2. Мелкозернистое аффинное расписание

Еще одним из недостатков является ухудшение локальности данных, не смотря на то, что расписание дает крупнозернистый параллелизм, поскольку реиспользование данных будет происходить во внешних циклах (зависимости удовлетворяются на внешних уровнях) и невозможно разбиение циклов без внесения изменений в распределения. Это может

свести на нет все преимущества параллелизации из-за ограниченной скорости передачи данных в памяти. Невозможность разбиения по измерениям расписания может также повлиять на переиспользование регистров при ригистровом разбиении.

Использование расписаний и распределений обычно приводит к расписаниям с минимальной размерностью и максимальным количеством параллельных циклов. Это не позволяет находить решения соответствующие высокоразмерным расписаниям. Такие не найденные расписания могут быть неоптимальными согласно заданным критериям выбора расписаний, но все же они позволяют более быстрое выполнения программы благодаря лучшему разбиению и крупнозернистому параллелизму. Данный недостаток не может быть устранен.

Все вышеперечисленные проблемы можно попытаться устранить путем поиска распределений с конкретным свойством, которое позволяет производить разбиение циклов вдоль размерностей расписания. Однако, иногда представляется невозможным найти прямое решение.

Методы, основанные на разбиениях. Данные методы используют кодогенератор как средство для сканирования пространства итераций в другом порядке и для последующей маркировке циклов как параллельных или последовательных, или для раскрытия и перестановки циклов в более позднее время. Основной недостаток данных методов заключается в отсутствии способа нахождения оптимального преобразования. Критериев, на которых основывается поиск решений с максимальной степенью параллелизма и минимальной степенью синхронизации, недостаточно для нахождения наилучшего решения. Например, на рис. 3 показано разбиение, полученное таким способом.

$$\begin{aligned} \phi^1(i,j) &= i \\ \phi^2(i,j) &= 3i + j \end{aligned} \qquad (3)$$

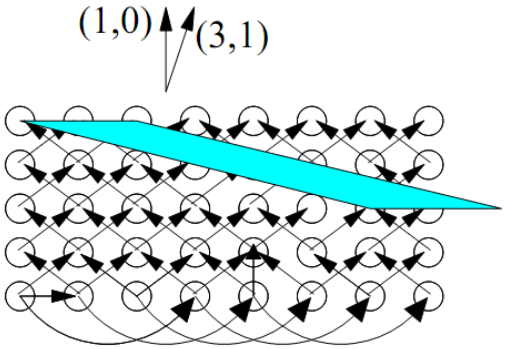

Рис. 3. Аффинное разбиение найденное без целевой функции может приводить к неудовлетворительным решениям

Хотя желаемое разбиение для данного случая показано на рис. 4

$$\phi^1(i,j) = i$$
$$\phi^2(i,j) = i + j \qquad (4)$$

Максимизация параллелизма и минимизация синхронизации. Этот алгоритм, представленный в [1], позволяет находить оптимальное аффинное отображение, которое максимизирует уровень параллелизма и минимизирует степень синхронизации. Это довольно мощный алгоритм, поскольку аффинные разбиения охватывают множество методов преобразований программ, включая разбиение, слияние и реиндексация циклов, унимодулярные преобразования, и перестановку операторов.

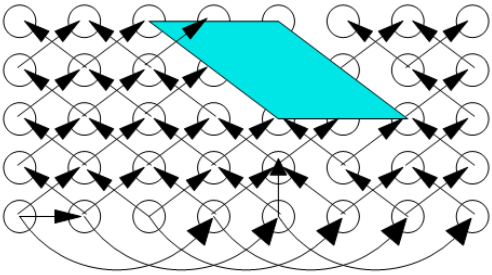

Рис. 4. Хорошая комбинация ϕ s: (1, 0) и (1, 1)

Проблема максимизиации параллелизма и минимизации синхронизации сформулирована на основе принципов, исключающих использование таких неточных абстракций как вектора зависимости данных. Определены два класса фундаментальных ограничений аффиннных преобразований: ограничения пространственного разбиения и временного разбиения, где второе – это релаксация первого. Проблема поиска всех степеней параллелизма, в его крупнозернистой форме, может быть сведена к поиску решения для этих ограничений. Более того, поиск этих решений является достаточно простым благодаря двум идеям. Первая – использование леммы Фаркаша для сведения множества ограничений в систему линейных неравенств. Вторая – используя степень параллелизма как метрику, получить оптимальное распределение с помощью простых алгоритмов, которые решают линейные неравенства.

Данный алгоритм не слишком отключается от других освещенных в литературе. Однако, модификация метода, представленная в [1], позволяет минимизировать степень синхронизации и находить временные разбиения с максимальной степенью параллелизма. Более того, для достижения конвейерного действительное разбиение может быть составлено из строк многомерного отображения.

Алгоритм планирования статически управляемых программ с аффинными зависимостями. Метод разработан Фотрие и описан в [3,4]. Основное достоинство метода – он не ограничен однородными преобразованиями. Однако, если, как часто бывает, граф потока данных имеет множество однообразных зависимостей и несколько неоднородных, один просто тест позволяет значительно уменьшить сложность процесса получения решения. Также были использованы другие методы для снижения трудоемкости алгоритма: исключение некоторых множителей Фаркаша, принимая во внимание преимущество только положительных или отрицательных экземпляров переменных. Ответ находится решением параметризированной программы относительно небольшого размера, которое выполняется пакетом PIP (www.piplib.org). Данный метод так же может решать задачи с неограниченными доменами. Можно использовать данный метод для поиска ограниченного расписания с задержкой или лучшего вогнутого расписания. В последнем случае получаем минимальную задержку в решении. Если граф потока данных равномерный, то задержка лучшего вогнутого расписания асимптотически оптимальна.

Метод имеет два недостатка. Первый – метод не может находить все кусочно аффинные расписания. Причина этого – существуют программы, свободное расписание которых не вогнуто.

Программа на рис. 5 имеет следующее свободное расписание:

$$\theta(i) = \textbf{if } i > n \textbf{ then } 1 \textbf{ else } 0, \quad (5)$$

в то же время двойной алгоритм Фаркаша дает следующее:

$$\theta`(i) = i. \quad (6)$$

Более того, свободное расписание имеет задержку 1, в то же время лучшее вогнутое расписание имеет задержку $O(n)$, поэтому результат не может быть приведен к однородному виду.

```
for i = 0,2n
   x(i) = x(2n-i)
end
```

Рис. 5. Программа с не вогнутым расписанием

Что важно, существует главный граф зависимостей, у которого нет аффинного расписания. На рис. 6 дан пример программы с таким расписанием.

```
do i = 0,n
   do j = 0,i
      s = s + a(i,j)
   end do
end do
```

Рис. 6. Простая программа, не обладающая линейным расписанием

Линейная программа для этого примера не выполнима, потому что аффинное расписание имеет задержку, которая асимптотически линейна в структурных параметрах. Поэтому понятно, что программа на рис. 6 не имеет параллелизма, и мнимальное время выполнения приблизительно равно $n^2/2$.

Таблица 1

Граф потока данных программы на рис. 6

Грань	Источник	Назначение	Условие
1	(1, i, j – 1)	(1, i, j)	j ≥ 1
2	(1, i – 1, j – 1)	(1, i, j)	j < 1 ^ i ≥ 1

С другой стороны, Доулинг доказал [7], что любая программа (в которой структурные параметры имеют численные значения), имеет линейное расписание, и заключил, ошибочно, что любая программа с достаточной вложенностью циклов имеет большую степень параллелизма. В случае программы на рис. 6 расписание Доулинга имеет следующую форму:

$$\theta(i) = n \cdot i + j \,, \tag{7}$$

которая является линейной, когда дано *n.* Это расписание имеет задержку n^2, что соответствует степени параллелизма равной 1/2!

И хотя поиск расписания для примера на рис. 6 не очень продуктивен, это может принести результат, и в более сложных случаях, эта программа имеет параллелизм, но не имеет аффинного расписания. Например, задержка этой программы $O(n^2)$ при времени выполнения $O(n^3)$, дает представление о степени параллелизма в зависимости от величины *n.*

Pluto. Фреймворк для автоматической трансформации программ содержащих несовершенные вложенные циклы, с одновременной оптимизацией параллелизма и локальности. Данный подход основан на поиске хороших вариантов разбиения с помощью мощной, практической и масштабируемой функции стоимости встроенной в формулировку задачи линейного программирования. Эта функция имеет следующий вид:

$$\delta_e\left(\vec{s},\vec{t}\right) = \phi_{s_j}\left(\vec{t}\right) - \phi_{s_i}\left(\vec{s}\right), \quad \left\langle \vec{s},\vec{t} \right\rangle \in P_e \,, \tag{8}$$

где $\phi_{s_j}\left(\vec{t}\right)$, $\phi_{s_i}\left(\vec{s}\right)$ - одномерные аффинные преобразования для исходной $\vec{s}$ и целевой $\vec{t}$ итераций цикла; P_e – многогранник зависимостей, который содержит информацию касательно грани e ; e – грань графа зависимостей данных представляющая собой зависимости между вершинами S_i и S_j, являющимися представлением операторов исходной программы.

Аффинная форма $\delta_e\left(\vec{s},\vec{t}\right)$ очень важна – эта функция, которая выдает количество гиперплоскостей, которые пересекаются зависимостью e по нормали φ к гиперплоскости. Если $\delta_e\left(\vec{s},\vec{t}\right)$ используется как пространственный цикл для генерирования разбиений при

распараллеливании, то она является показателем объема коммуникаций. С другой стороны, если она используется как последовательный цикл, то она является мерой расстояния повторного использования. Верхняя граница данной функции означает, что число гиперплоскостей, которые будут обмениваться данными в результате зависимостей на границах разбиений, не превысит данное число.

Попытка минимизировать вышеуказанную функцию приводят к объективной нелинейности в переменных цикла и коэффициентах гиперплоскости. Например, $\phi(\vec{t}) - \phi(\vec{s})$ может быть $c_1 i + (c_2 - c_3) * j$ с ограничениями $1 \le i \le N,\ 1 \le j \le N,\ i \le j$. Такая форма записи является результатом случая, когда зависимость неоднородна или существуют зависимости внутри выражений цикла. Это препятствие может быть преодолено использованием подхода с ограничивающей функцией, который позволяет применить лемму Фаркаша и привести данную задачу к задаче линейного программирования. Такой подход был впервые использован Фотрие [3,4], но в другой ситуации - для поиска расписаний с минимальной задержкой.

Решая задачу линейного программирования, получаем единственное решение с коэффициентами лучшего отображения для каждого оператора программы. Этот шаг повторяется, пока не будут найдены решения для всех размерностей домена.

Минимизируя $\phi(\vec{t}) - \phi(\vec{s})$, реализация зависимостей перемещается во внутренние циклы, от самой внешней гиперплоскости к внутренней, в то же время новые циклы будут иметь не отрицательные компоненты зависимостей, поэтому они могут быть разбиты для улучшения локальности и, если будут существовать зависимости (прямые), то может быть извлечен конвейерный параллелизм.

Выводы. Основное направление исследовательской работы видится в устранении недостатков присущих вышеперечисленным алгоритмам и расширении области их применения к различным типам программ. Другим вариантом использования сильных сторон алгоритмов есть создания программных систем, в которых будет динамически выбираться алгоритм преобразования входной полигональной модели. Основным вопросом в таком случае является критерий выбора алгоритма.

Анализаторы кода, которые генерируют полигональную модель из исходных текстов программ, на текущий момент ориентированны, в основном, на такие языки как С и Фортран. Логическим продолжением практической работы было бы создание интерфейсов для других распространенных или специфических языков, как то Java, Python, C++ и т.д.

Наиболее общим в плане обработки различных видов циклов является метод [2]. Кроме строгих теоретических обоснований, он доведен до

работающего пакета, оптимизирующего программы на уровне исходных текстов. Тем не менее не все вопросы здесь решены. В частности, не обоснованы вопросы оптимальности разбиения пространства итераций на части и некоторые вопросы при объединении циклов. Это требует дополнительных исследований в этой области.

1. *Amy W. Lim, Monica S. Lam* Maximizing parallelism and minimizing synchronization with affine partitions - Computer Systems Laboratory, Stanford University, Stanford, 12 с.
2. *Uday Kumar Reddy Bondhugula* Effective Automatic Parallelization and Locality Optimization Using The Polyhedral Model - Graduate School of The Ohio State University, 2008 –18 с.
3. *Paul Feautrier* Some efficient solutions to the affine scheduling problem Part I,II - Laboratoire MASI, Institut Blaise Pascal, 42 с.
4. *Paul Feautrier* Automatic parallelization in the polytope model, Laboratoire PRiSM, The University Of Versailles - Saint-Quentin, 27 с.
5. *Fabien Quillere, Sanjay Rajopadhye, Doran Wilde* Generation of efficient nested loops from polyhedral, Rennes, France; Provo, Ut, USA, 2000 – 29 с.
6. *Peter Faber* Code optimization in the polyhedron model – improving the efficiency of parallel loop nests, University of Passau, 2007-10 – 221 с.
7. *Michael L. Dowling* Optimal code parallelization using unimodular transformations, Parallel Computing, 1990 – 14 с.

Поступила 28.02.2011р.

УДК 683.06

Б.В.Дурняк, Є.Д.Бабинець

СПОСОБИ РОЗШИРЕННЯ МОДЕЛІ ВЗАЄМОЗВ'ЯЗКУ МІЖ ПАРАМЕТРАМИ КНИЖКИ ТА СПОЖИВАЧА ІНФОРМАЦІЙНИМИ КОМПОНЕНТАМИ

Основною ознакою інформаційної компоненти в засобах аналізу, чи засобах моделювання тих, чи інших об'єктів, що складають загальну модель процесу проектування образу книги (*МРО*), є використання текстових описів, що представляють собою інтерпретацію елементів, які останні описують [1]. У книгах, які є продуктом поліграфічного виробництва, в більшості випадків розміщається текстова інформація, яка представляє собою один з основних способів інформаційного наповнення. В цьому випадку, виникає задача визначення елемента, який представляв би собою інтерпретаційний опис такої інформаційної компоненти книжки. Для розв'язку цієї задачі, приймемо, що інтерпретаційним текстовим описом компоненти, що представляє собою текст інформаційного змісту книги,

