РАСПАРАЛЛЕЛИВАЮЩИЙ ТРАНСЛЯТОР ЯЗЫКА ОПИСАНИЯ ЦИКЛОВ

А.А. Новокрещенов

Нижегородский государственный технический университет им. Р.Е. Алексеева
E-mail: andrew.novokreshchenov@gmail.com

Циклические конструкции заключают в себе значительный объем вычислений (особенно в программах, обрабатывающих большие массивы данных) и являются хорошим источником параллелизма. Более того, параллелизм, заключенный в циклах, относительно просто идентифицируется и хорошо масштабируется в пространстве процессоров [1]. Эти обстоятельства объясняют повышенный интерес к алгоритмам и программным средствам для распараллеливания циклических конструкций, о чем свидетельствует большое количество работ в данной области, а также появление средств для распараллеливания циклов в современных компиляторах [2]. С другой стороны, даже современные алгоритмы распараллеливания далеко не всегда могут сгенерировать эффективную параллельную программу. Поэтому представляется полезной реализация средств распараллеливания в виде полуавтоматических трансляторов, выходом которых является не исполняемый код для конкретной параллельной архитектуры, а детальная информация о параллелизме, скрытом в последовательной программе [3]. Такой подход позволяет разработчику участвовать в процессе распараллеливания программы, что может сделать параллельный код более эффективным.

В данной работе предлагается язык для описания циклов и структура распараллеливающего транслятора данного языка для мультипроцессорных систем.

Цель разработки языка заключается в создании удобного способа описания последовательных вложенностей циклов. Данный язык обладает простым синтаксисом и поддерживает только те конструкции, которые необходимы для выполнения процедуры поиска параллельности. Ниже приводится пример программы, записанной на языке описания циклов:

```
symbol      N;
array       A[N], B[N], C[2*N];
for (index i: 0 to N)
{
      for (index j: 0 to N)
      {
            if (i == 0 || j == 0)
                  C[I – k + N] = A[i] * B[-1*k + N];
            if (i != 0 && k != 0)
                  C[i – k + N] = C[i – k + N] + A[i] * B[-1*k + N];
      }
}
```

Задача транслятора языка описания циклов заключается в поиске параллельности, скрытой во входной вложенности циклов. Структура транслятора приводится на рис. 1.

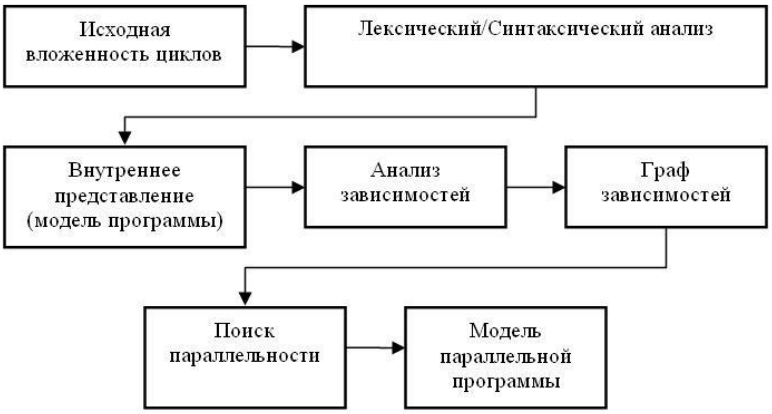

Рис. 1. Структура транслятора

Задача фазы лексического и синтаксического анализа заключается в построении внутреннего представления последовательной программы. Внутренне представление заключает в себе всю информацию, необходимую для выполнения процедур анализа зависимостей и поиска параллельности, и представляет собой множество кортежей вида:

$$L = \{< D_i, M_i >\},\ i \geq 0 \wedge i \leq N-1, \quad (1)$$

где i – порядковый номер инструкции в исходной вложенности, D_i – домен i-й инструкции, M_i – множество обращений к массивам в i-й инструкции. Домен инструкции представляет собой целочисленный параметрический многогранник, точки которого определяют все комбинации значений индексных переменных, при которых выполняется инструкция i [4]. Другими словами, домен инструкции i определяет множество экземпляров данной инструкции [1]. Множество M_i представляет собой множество кортежей вида:

$$M = \{< A_j, f_j(\vec{x}), \delta_j >\},\ j \geq 0 \wedge j \leq M-1, \quad (2)$$

где j – порядковый номер обращения к массиву в инструкции; A_j – имя массива, к которому осуществляется обращение; $f_j(\vec{x})$ – аффинная функция от индексных переменных, охватывающих инструкцию; δ_j – принимает значение «истина», если j-е обращение является обращением на запись, и «ложь», если j-е обращение является обращением на чтение. Аффинная функция $f_j(\vec{x})$ определяет соответствие между точками домена инструкции и номерами элементов массива A_j, т.е. данная функция определяет, к каким элементам массива A_j осуществляют доступ экземпляры обращения.

Задача процедуры анализа зависимостей заключается в установлении наличия или отсутствия зависимости между всеми парами обращений программы (1). В качестве теста на зависимость предполагается использовать омега-тест [5]. Результат фазы анализа зависимостей представляет собой сокращенный граф зависимостей с векторами расстояний [6].

Процедура поиска параллельности ориентирована на то, что в качестве целевой архитектуры используется мультипроцессорная архитектура, позволяющая эффективно выполнять большое количество потоков одновременно. Поэтому модель параллельной программы представляет собой информацию соответствия между экземплярами инструкций исходной вложенности циклов и точками k-мерного целочисленного пространства Z^k, которое представляет собой пространство потоков (процессоров). Говоря

более простым языком, для каждого экземпляра инструкций исходной программы модель параллельной программы определяет номер потока (процессора), который будет выполнять данный экземпляр. Такая постановка задачи наиболее актуальна для SIMD-архитектур [1, 7].

Информация соответствия представляется в виде набора функций вида:

$$\Pi_i(\vec{x}_i) = \mathbf{C}_i \times \begin{pmatrix} \vec{x}_i \\ \vec{n}_i \\ 1 \end{pmatrix}, i \geq 0 \wedge i \leq N-1, \quad (3)$$

где i – порядковый номер инструкции в исходной вложенности; $\vec{x}_i$, $\vec{n}_i$ – вектор индексных переменных и вектор структурных параметров циклов, охватывающих инструкцию i; $\mathbf{C}_i$ – целочисленная матрица размерности k x d, определяющая отображение домена инструкции i в пространство потоков (процессоров) размерности k.

Следует отметить, что функции вида (3) составляются для каждой инструкции исходной вложенности. Таким образом, результат выполнения процедуры поиска параллельности представляет собой набор функций вида (3).

Алгоритм поиска параллельности, используемый в данной работе, основан на двух требованиях. Первое требование заключается в том, что каждая пара зависимых экземпляров инструкций исходной вложенности должна выполняться в рамках одного потока (процессора). Более формально, для каждой дуги между вершинами i и j сокращенного графа зависимостей, помеченной вектором расстояния $\vec{v}_{i,j}$, должно выполняться условие:

$$\Delta_{i,j} = \Pi_j(\vec{x}_j) - \Pi_i(\vec{x}_i) = \vec{0}, \{\forall \vec{x}_i, \vec{x}_j \mid \vec{x}_i + \vec{v}_{i,j} = \vec{x}_j\}. \quad (4)$$

Второе требование заключается в максимизации числа потоков, т.е. в обнаружении максимально возможного параллелизма. Данное требование удовлетворяется путем назначения каждого независимого экземпляра инструкции отдельному потоку.

Учет указанных требований для всего графа зависимостей приводит к построению набора систем линейных уравнений/неравенств относительно искомых элементов матриц $\mathbf{C}_i$. Как показывает практика, если системы совместны, то они в большинстве случаев имеют бесконечное множество решений. Учитывая это обстоятельство, а также то, что системы требуется разрешить в целых числах, решение предлагается определять как точку лексикографического минимума многогранника, определяемого каждой системой уравнений/неравенств в отдельности.

При реализации транслятора для представления многогранников и выполнения операций над ними используется библиотека Polylib, в качестве реализации алгоритма анализа зависимостей используется API пакета Petit, для поиска лексикографического минимума используется библиотека Piplib [3, 8, 9].

Литература

1. Компиляторы: принципы, технологии и инструментарий / Под ред. И.В. Красиковой. М.: Вильямс, 2008. 1184 с.
2. Pop S., Bastoul C., Cohen A. Graphite: polyhedral analyses and optimization for GCC // Proceedings of the 2006 GCC Development Summit. Ottawa, Canada, 2006. P. 179-198.
3. New user interface for Petit and other extension – http://www.cs.umd.edu/projects/omega/.
4. Pouchet L., Bastoul C. Iterative optimization in the polyhedral model: part 1 // Proceedings of the International Symposium on Code Generation and Optimization, CGO'07: IEEE Computer Society, Washington, 2007. P 144-156

5. Pugh W. The omega test: a fast and practical integer programming algorithm for dependence analysis // Proceedings of the 1991 ACM/IEEE conference on Supercomputing: ACM New York, 1991. P. 4-13.
6. Касьянов В.Н., Мирзуитова И.Л. Реструктурирующие преобразования: алгоритмы распараллеливания циклов // Программные средства и математические основы информатики. Новосибирск: Институт систем информатики им. А.П. Ершова СО РАН, 2004. С. 142-188.
7. Lim A., Lam M. Maximizing parallelism and minimizing synchronization with affine transform // Proceedings of the 24th ACM SIGPLAN-SIGACT: ACM New York, 1997. P. 215-227.
8. Polylib – a library for polyhedral functions – http://icps.u-strasbg.fr/polylib/.
9. PipLib – a solver for parametric integer programming problems – http://www.piplib.org/.