УДК 004.4'24

А.А. Чемерис, Киев

ОПРЕДЕЛЕНИЕ ЗАВИСИМОСТЕЙ В КОМПЬЮТЕРНЫХ ПРОГРАММАХ

Abstract. The information dependencies in the programs based on high-level languages is considered. The classification of dependencies is presented. Particular attention is paid to the information dependencies in cyclic program section.

Введение

Использование параллельных вычислительных систем позволяет значительно увеличить производительность вычислений. Однако, эффективное их использование возможно только при разработке эффективных параллельных программ и приложений. Очевидно, что создание такого программного обеспечения зависит от уровня программиста. И даже программы, созданные программистом высокого уровня, могут быть оптимизированы в плане получения более высокого быстродействия, в частности, за счет выявления участков кода, которые возможно выполнять параллельно.

При разработке средств компилирования компьютерных программ значительное внимание уделяется средствам оптимизации кода. Это касается как последовательных программ, так и параллельных. Тем не менее, средствам автоматического распараллеливания алгоритмов уделяется особое внимание. Основываясь на некотором логическом представлении алгоритмов либо на структурном уровне в виде некоторой диаграммы, например, используя UML, или на системном уровне, используя код на языке высокого уровня, средства автоматического распараллеливания проводят, в общем виде, три стадии обработки алгоритма. На первой стадии проводится анализ алгоритма, на второй разрабатываются правила трансформации данного алгоритма, а на третьей – генерация кода для параллельной вычислительной системы.

Для первой стадии - анализа алгоритма, с целью оптимизации или распараллеливания, требуется собрать необходимую информацию о структуре программы, а именно, о выполняемых командах программы и их взаимодействии между собой. В качестве модели для выявления параллелизма и возможностей оптимизации программных приложений параллельных компьютеров в данной работе выбран граф зависимостей.

В данной статье определяются основные понятия, которые используются в методах распараллеливания циклических участков программ, написанных на языке высокого уровня. При этом рассматриваются зависимости, как для любых участков программ, так и характерные только для циклических.

Виды зависимостей

Прежде чем выполнять преобразования в программах, связанные с оптимизацией или распараллеливанием, требуется определить зависимости между операциями и данными в алгоритме. Зависимости возникают в том случае, когда определена необходимость обращения к одной и той же ячейке памяти в различные моменты времени. Так, если оператор S1 обращается к ячейке памяти x и потом S2 обращается к x, то говорим, что S2 зависит от S1 по x.

Известно, что в программах выделяют четыре типа зависимостей, связанных с доступом к одним и тем же ячейкам памяти [1].

Рассмотрим два оператора S_i и S_j ($S_i \prec S_j$) и в соответствии с [2] определим типы зависимостей в программе. Здесь используется знак '$\prec$', который определяет лексикографический порядок операторов S_i и S_j, т.е. оператор S_i предшествует в программе оператору S_j. Обе команды производят операции чтения-запись с памятью компьютера.

Определение. [3] Лексикографический порядок (упорядоченная последовательность) n-мерного декартового произведения $A^n = A \otimes A \otimes ... \otimes A$ определяется следующим образом: если $a = (a_1, a_2, ..., a_n)$ и $b = (b_1, b_2, .., b_n)$, то $a \prec b$, если $a_1 < b_1$ или $a_1 = b_1$, $a_2 = b_2$, …, $a_k = b_k$ и $a_{k+1} < b_{k+1}$, где $1 \le k \le n-1$.

Зависимость по данным (Data – Flow Dependence) между операторами Si и Sj (S$i \prec$ Sj) возникает в том случае, когда оператор программы Sj считывает данные, которые вычисляются в Si.

$$Si: \ X = F_1(\ In(Si)\ );$$
$$Sj: \ Y = F_2(\ In(Sj)\ );\ X \in In(Sj)\ \&\ j \ge i+1;$$

где In(S) - множество входных переменных оператора S. Зависимость по данным определяется так называемым порядком «запись перед чтением» [4]. В соответствии с обозначениями, принятыми в [5] запишем данную зависимость как Si δ Sj , что интерпретируется как «чтение переменной в Sj зависит от записи этой переменной в Si».

Зависимости по выходу (Output Dependence) двух операторов Si и Sj (S$i \prec$ Sj) возникает в том случае, когда запись одной и той же переменной осуществляется в обоих операторах, т.е.

$$Si: \ X = F_1(\ In(Si)\ );$$
$$Sj: \ X = F_2(\ In(Sj)\ );\ j \ge i+1.$$

Эта зависимость предохраняет от использования при выполнении какого-либо текущего оператора значений неправильных значений X. Обозначается зависимость по выходу как Si δ^0 Sj.

Антизависимость (Antidependece) между операторами в программе Si и Sj (S$i \prec$ Sj) возникает в том случае, если чтение переменной из памяти

производится раньше ее записи в память компьютера, т.е.

$$Si: \ X = F1(\ In(Si)\ );\ Y \in In(Si)$$

$$Sj: \ Y = F2(\ In(Sj)\ );\ j \geq i+1.$$

Антизависимость предотвращает изменение S*i* до выполнения S*j*, что может привести к неправильным значениям. Записываем эту зависимость в виде $Si\ \delta^{-1}\ Sj$.

Зависимость четвертого типа есть зависимость по управлению. Она определяется операторами программы, которые изменяют порядок выполнения операторов в зависимости от некоторого условия. Так, например, во фрагменте кода на языке С

```
S1:        if( x == 0 )
S2:            y = ToDo( arg );
```

значение оператора S_1 определяет, будет ли выполняться оператор S_2 или нет.

Таким образом, в программах определены приведенные выше типы зависимостей между операторами. Эти зависимости характерны для любых участков программ. Однако, для циклических участков определены некоторые специфические типы зависимостей. Классификация зависимостей приведена на рис. 1.

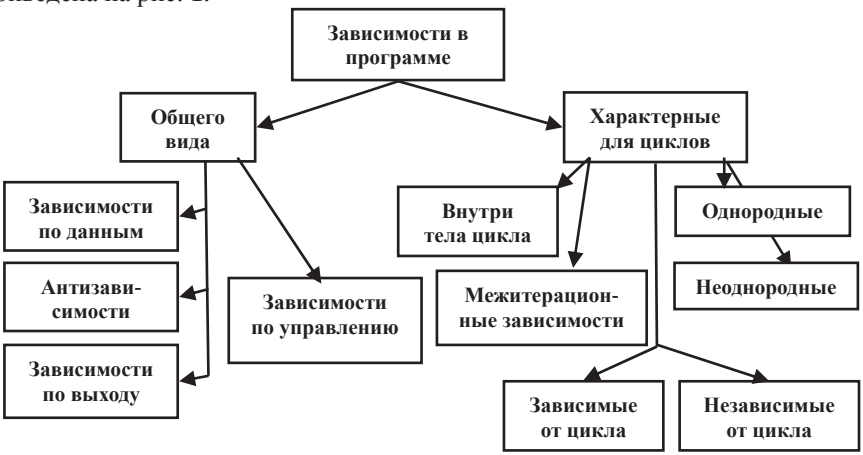

Рис. 1. Классификация зависимостей в программах

Представление зависимостей в виде графов

При анализе и оптимизации программы используется графовые модели представления зависимостей в программах. Существуют как более простые виды представления - графы информационных зависимостей или потоковый граф , так и более сложные, в частности, решетчатый граф, граф программы на основе сети Петри и др.

Определение. Развернутым графом зависимостей алгоритма (*extended dependence graph*) называется ориентированный граф. полученный следующим образом: каждая операция, *S*(*J*) порождает вершину *V*(*J*) графа, каждая зависимость *S*(*I*)—>*S**(*J*) порождает дугу (*V*(*I*), *V**(*J*)).

Развернутый граф зависимостей дает полную количественную информацию о зависимостях алгоритма.

Анализируя код программ на языках высокого уровня будем различать входные и выходные переменные. Данные определения переменных основываются на представлении оператора как вершины графа, которая имеет как входные, так и выходные дуги. Связи в графе определены соответствующими переменными, за которыми закреплены определенные ячейки в памяти. Обращение к одной и той же ячейки памяти определяет из разных мест в программе определяет конкретную зависимость в программе.

Большее число вычислений в программах сосредоточено в циклах, которые оперируют с индексными переменными, представляющими *n*-мерные массивы данных – вектора, матрицы и т.д. Распараллеливание циклов является наиболее существенным ресурсом для увеличения производительности вычислений, и далее рассматриваем циклические участки программ.

Зависимости в циклах.

В общем случае рассматриваем гнездо циклов, каждый их которых определяет свою индексную переменную. Множество индексных переменных определяет индексный вектор гнезда циклов $I = (I_1, I_2, ..., I_n)$.

Для любого цикла, в котором индекс цикла *I* изменяется от значения *L* к *U* с шагом *S*, номер итерации *i* равен значению (*I* - *L* + *S*) / *S*, где *I* – это значение индекса для этой итерации [5].

Для гнезда из *n* циклов вектор итерации *I* для самого внутреннего цикла есть вектором, содержащим целое число итераций для каждого цикла в порядке вложенности циклов. Другими словами, номер итерации многомерного вложенного цикла определяется в соответствии с формой $I = (i_1, i_2, ..., i_n)$, где $i_k, 1 \le k \le n$, представляет собой номер итерации цикла для уровня вложенности *k*.

Определение. Пространством итераций цикла называется множество всех целочисленных векторов $I = (I_1, I_2, ..., I_n)$, которые удовлетворяют системе неравенств [6]

$L_i \le x_i \le U_i$, где $i = 1 \ldots n$. (1)

Неравенства (1) определяют границы циклов и в пространство итераций они ограничивают выпуклым многогранником.

Вектор расстояния зависимости

Определение. [7] Пусть два вхождения *u* и *v* имеют *d* общих циклов. Рассмотрим множество всех итераций $\boldsymbol{I}_k=(I_{k1}, I_{k2}, \cdots, I_{kn1})$ и $\boldsymbol{J}_k=(J_{k1}, J_{k2}, \cdots, J_{kn2})$, где $1 \le k \le m$, $n1 \ge d$, $n2 \ge d$, таких, что $v[\boldsymbol{J}_k]$ зависит от $u[\boldsymbol{I}_k]$. Каждый вектор $(J_{k1} - I_{k1}, J_{k2} - I_{k2}, \cdots, J_{kd} - I_{kd})$ (при фиксированном *k*) будем называть вектором расстояния зависимости *v* от *u*, а множество различных векторов $D=(J_{k1} - I_{k1}, J_{k2} - I_{k2}, \cdots, J_{kd} - I_{kd})$ будем называть множеством

векторов расстояния зависимости *v* от *u*.

Вектор направления зависимости.

Определить тип существующей зависимости по данным и возможность распараллеливания цикла по виду вектора расстояний сложно. Поэтому вводится понятие вектора направлений для цикла [8]. Компоненты вектора направлений *d* являются символами, которые определяются следующим образом:

$$\begin{aligned} &d_i == '', D_i = 0; \\ &d_i => '', D_i < 0; \\ &d_i =< '', D_i > 0; \end{aligned} \qquad (2)$$

Пусть две операции *S* и *S** выполняются в теле цикла и имеют зависимость. Направление «>» означает, что операция *S*, обращается к общему участку памяти *M* на некоторое количество итераций позже, чем операция *S**. Направление «<» означает, что операция *S*, обращается к общему участку памяти *M* на некоторое количество итераций раньше, чем операция *S**. Направление «=» означает, что операции *S* и *S** обращаются к общему участку памяти M на одной и той же итерации цикла.

Расстояние зависимости играет важную роль при анализе цикла на параллельность. Его значение позволяет определять тип возникающей зависимости по данным и возможность разбиения итерационного пространства на зоны ответственности для параллельного исполнения.

Зависимости в теле цикла и между итерациями.

Тело цикла состоит из множества операторов *S*. Зависимости между операторами существуют как внутри цикла, так и между итерациями. Представим развернутые в итерационном пространстве несколько шагов выполнения оператора цикла.

На Рис. 1 сплошными линиями представлены зависимости внутри тела цикла. Так для массива a[] существуют зависимости по данным. В то время как для массива b[] присутствуют межитерационные зависимости по данным.

В гнезде из *n* вложенных циклов множество векторов расстояния аппроксимируется целым числом *l* из интервала $[1, n] \cup \{\infty\}$, которое определяется как наибольшее целое, такое, что первые *l*-1 компонент векторов расстояний оказываются нулевыми. Зависимость на уровне $l \le n$ означает, что зависимость обнаруживается на уровне *l* гнезда циклов, т. е. на заданной итерации *l*-1 внешних циклов. В этом случае говорят, что зависимость есть межитерационной зависимостью и такие зависимости называются циклически зависимыми (*loop-carried*) на уровне *l*. Если $l = \infty$, то зависимость происходит внутри тела цикла, между двумя различными операторами. Такие зависимости называются циклически-независимыми (*loop-independent*). Значение *l* называют уровнем зависимости.

```
for( i=1; i<N; i++)
        for(j=1; j<N; j++) {
           a[i,j] = b[i,j-1] + c[j+1];
           b[i+1,j] = a[i,j] + d;  }
```

i \ j	1	2	3	4
1	a[1,1]=b[1,0]+c[2]; b[2,1] = a[1,1] + d	a[1,2]=b[1,1]+c[3]; b[2,2] = a[1,2] + d	a[1,3]=b[1,2]+c[4]; b[2,3] = a[1,3] + d	a[1,4]=b[1,3]+c[5]; b[2,4] = a[1,4] + d
2	a[2,1]=b[2,0]+c[2]; b[3,1] = a[2,1] + d	a[2,2]=b[2,1]+c[3]; b[3,2] = a[2,2] + d	a[2,3]=b[2,2]+c[4]; b[3,3] = a[2,3] + d	a[2,4]=b[2,3]+c[5]; b[3,4] = a[2,4] + d
3	a[3,1]=b[3,0]+c[2]; b[4,1] = a[3,1] + d	a[3,2]=b[3,1]+c[3]; b[4,2] = a[3,2] + d	a[3,3]=b[3,2]+c[4]; b[4,3] = a[3,3] + d	a[3,4]=b[3,3]+c[5]; b[4,4] = a[3,4] + d

Рис. 1. Зависимости внутри цикла и между итерациями

Графовые модели зависимостей в циклах

Определение развернутого графа зависимостей дано выше. Для адаптации данного графа при представлении циклических участков программ требуется развернуть цикл в соответствии значений индексных переменных, что определяет порядок выполнения операторов, составляющих тело цикла.

Редуцированный граф цикла

Из-за наличия разного вида зависимостей, во многих случаях невозможно выполнить код параллельно без предварительного анализа и последующего сокращения зависимости. Для того чтобы облегчить получение параллельного кода программы, после получения множества зависимостей цикла, создается так называемый уменьшенный (редуцированный) граф зависимостей (reduced dependence graph - RDG) [9]. Это есть направленный граф *G=(V,E)*, структура которого определяется зависимостями в теле цикла. Вершины графа соответствуют множеству операторов в теле цикла. Далее определяются все сильно связанные компоненты, которые являются максимальным подграфом $S = (V', E')$ в котором для каждой вершины $u \in V \,\&\, u \notin V'$ не существует другой вершины $v \in V'$, между которыми существует связь $(u, v) \in E$. Т.е. между двумя вершинами подграфа S существует направленная дуга графа.

Редуцированный граф зависимостей, дуги которого помечены уровнями зависимостей, называется редуцированным уровневым графом зависимостей.

Решетчатые графы циклов

Понятие решетчатого графа подробно рассмотрено в работах Воеводина В.В. [10], [11], [2].

Если в узлах итерационного пространства, которые соответствуют точкам с координатами, равными значениям индексных переменных гнезда циклов, разместить вершины графа, соответствующие телу цикла при различных значениях индексных переменных, и связать эти вершины дугами, которые соответствуют межитерационным зависимостям, то результатом

будет решетчатый граф зависимостей гнезда циклов.

Выводы

Для эффективного распараллеливания или оптимизации программ необходимо сохранить систему зависимостей в программе. В противном случае при выполнении программы получается неправильный результат. Особенно это важно при распараллеливании циклических участков программ. Система зависимостей в программе определяет насколько качественным и эффективным будет процесс автоматического распараллеливания программ. В данной статье приводится классификация зависимостей, где особое место уделено зависимостям в циклических участках программ.

Чем полнее будет представлено множество зависимостей в программе, тем эффективнее будет распараллеливание программы или ее оптимизация. От этого шага в анализе программ зависит, будет ли распараллелена та или иная языковая конструкция. Известно, что в данное время не все циклы могут быть распараллелены. Таким образом, улучшая представление зависимостей в программе, создаются возможности не только распараллеливания, но и поддержки тех языковых конструкций, которые раньше не могли быть распараллелены.

1. *Г. Эндрюс*. Основы многопоточного, параллельного и распределенного программирования., М.: Издательский дом "Вильямс", 2003.- 512 с..
2. *Воеводин В.В., Воеводин Вл. В.,* Параллельные вычисления, СПб: БХВ-Петербург, 2002.
3. *B. Utpal,* Loop Transformations for Restructuring Compilers: The Foundations. Series on Loop Transformations for Restructuring Compilers, Boston, Massachusetts: Kluwer Academic Publishers, 1993.
4. *D. Moldovan,* Parallel Processing: From Applications to Systems., Morgan Kaufmann Publishers, Inc, 1993.
5. *Allen R., Kennedy K.,* Optimizing compilers for modern architectures: A Dependence-based Approach., Morgan Kaufmann Publishers, Inc., 2001..
6. *Р. Б. Штейнберг,* Автоматическое отображение программ на конвейерные и многоконвейерные архитектуры, Ростов-на-Дону: Диссертация на соискание ученой степени канд.техн.наук (На правах рукописи), 2012.
7. *Maydan D. E., Amarasinghe S. P., Lam M. S.,* «Data dependence and data-flow analysis of arrays.,» в In 5th Workshop on Languages and Compilers for Parallel Computing, New Haven, CT, August 1992.
8. *U. Banerjee,* Loop Transformations for Restructuring Compilers, Springer Science & Business Media, 1997.-214 p..
9. *A. Darte,* «Mathematical tools for loop transformations: from system of uniform recurrence equations to the polytope model,» Ecole Normale Superieure de Lyon, Lyon, 1997.
10. *В. В. Воеводин,* Математические модели и методы в параллельных процессах., М.: Наука.- 296 с., 1986.
11. *В. В. Воеводин,* Математические основы параллельных вычислений., М.: МГУ. - 345 с., 1991.

Поступила 21.9.2015р.