串行程序的任务 DAG 图构造算法①

孙立斌, 邓 蓉, 陈闵中

(同济大学 计算科学与技术系, 上海 201804)
(同济大学 嵌入式系统与服务计算教育部重点实验室, 上海 201804)

摘 要: 任务 DAG 图是刻画程序中各任务间依赖关系的一种手段, DAG 图上除了标有任务间的依赖关系, 还记录了各任务的计算量和任务之间的通信量, 这些信息共同构成了任务调度的依据, 国内外有许多基于任务 DAG 图的调度算法研究, 但通过分析串行程序的相关性来构造任务 DAG 图的研究却不多见. 分析了串行程序中存在的数据相关性和控制相关性, 就程序中的顺序, 分支, 循环三种基本结构进行分别讨论, 提出了一种串行程序任务 DAG 图的构造算法.

关键词: 串行程序并行化; DAG 图; 数据相关性; 控制相关性; 资源相关性

Algorithm for Constructing Task DAGs of Serial Programs

SUN Li-Bin, DENG Rong, CHENG Hong-Zhong

(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)
(Key Lab of Services Computing and Embedded Systems of Ministry of Education, Tongji University, Shanghai 201804, China)

Abstract: Task DAG is a mean of depicting dependency among tasks of a program. Apart from dependency between tasks, amount of computation and traffic between tasks are also reflected on DAG. All these information together provides a basis for scheduling. There are much research on scheduling base on DAG both domestic and abroad, but research on constructing task DAG based on analysis of correlation among serial program is rare. This paper focuses on analysis of data dependency and control dependency and discuss respectively on sequential, branch and loop structure, propose an algorithm for constructing task DAG of serial programs.

Key words: parallelization of serial programs; directed acyclic graph; data dependency; control dependency; resources dependency

1 引言

并行计算是加快程序执行速度的有效手段. 然而, 在现实生活中, 实际问题的解决方案很多都是串行的, 如何利用这些已有的串行程序, 挖掘其中潜在的并行执行潜力, 使其能够并行执行, 提高执行效率, 是并行计算领域里的一个基本问题.

要将串行程序并行化, 要经过 2 个步骤: (1)分析串行程序中存在的相关性, 找出其中哪些任务可以并行执行, 哪些不能够并行执行, 并使用一种数据结构(一般是 DAG 图)来描述任务间的这种相关性. (2)根据 DAG 图将程序划分成若干个可以并行执行的部分, 并为它们指派处理器或处理结点.

国内外有许多基于任务 DAG 图的调度算法研究, 但通过程序相关性分析来构造串行程序任务 DAG 图的研究却不多见. 文献[1]中的 DAG 图构造算法没有消除数据相关中的输出相关和反相关. 文献[2]中, 在利用变量换名消除输出相关和反相关时, 没有考虑换名对程序语义造成的影响, 算法也没有分析分支结构中存在的控制相关性. 此外, 文献[3]只是针对数据相关性进行了分析, 且不是基于 DAG 图的. 本文就程序中的顺序、分支、循环这三种基本结构分别进行讨论, 分析了其中蕴含的数据相关性和控制相关性, 在利用

① 收稿时间:2012-01-17;收到修改稿时间:2012-02-29

换名规则消除输出相关和反相关的同时，通过增加额外的语句来使换名前后程序语义不发生变化. 最后，对相关构造算法的复杂度进行了分析.

2 相关性理论

根据相关性理论的研究，任务间存在三种相关性: 数据相关性、控制相关性和资源相关性. 其中，数据相关性和控制相关性是任务本身固有的性质，而资源相关性则是由于资源不足而导致的，可以通过增加资源数量的方式来消除. 下面对这 3 种相关性进行逐一的说明.

2.1 数据相关性

数据相关性指的是顺序执行的若干条语句所涉及的变量存在着交集，设语句 S_1 于 S_2 之前执行，S_1 的输入变量集为 I_1，输出变量集为 O_1，S_2 的输入变量集为 I_2，输出变量集为 O_2. 数据相关性又可细分为以下 3 种.

(1)若 $I_2 \cap O_1 \neq \emptyset$，则 S_1 和 S_2 流相关，记作: $S_1 \rightarrow S_2$

(2)若 $O_2 \cap O_1 \neq \emptyset$，则 S_1 和 S_2 输出相关，记作: $S_1 \nrightarrow S_2$

(3)若 $O_2 \cap I_1 \neq \emptyset$，则 S_1 和 S_2 反相关，记作: $S_1 \not\rightarrow S_2$

存在以上任何一种相关，S_1 和 S_2 都不能直接并行. 然而，输出相关和反相关可以通过变量换名的方式来消除的，而流相关，也称作真相关，是无法消除的.

2.2 控制相关性

在串行程序中，除了数据相关性外，还存在控制相关性. 若程序中有两个任务 T_1 和 T_2. T_2 是否能执行取决于 T_1 的执行结果. 那么，就称 T_2 控制依赖于 T_1. 在高级语言中，if、switch 等复合语句的内部语句块是否执行依赖于其条件部分，以下面的 if 语句快为例:

```
if(f(n)> threshold){
   c = g(n)
}
else {
   c = t(n)
}
```

设 if(f(n) > threshold)为 T_1、g(n)为 T_2、t(n)为 T_3，则 T_2 和 T_3 是否执行控制依赖于 T_1 的执行结果. 类似地，Switch 结构也是如此，只是增加了分支数目.

除了分支结构外，程序中的跳转语句也反应了一种控制依赖关系，在下面的例子中，T_1 执行完后，执行跳转指令 CALL Proc 调转到 Proc 中执行 T_2. T_2 控制依赖于 T_1.

```
T1;   CALL Proc
Proc {
 T2;
}
```

2.3 资源相关性

资源相关性指的是任务由于资源数量的限制而不能够并行地执行. 例如有 5 个任务，4 个 CPU，则同一时间只能有 4 个任务同时执行，但这种相关性不是任务间固有的，可以通过增加资源数量来消除. 本文探讨的是程序内在的相关性: 即数据相关性和控制相关性.

3 程序相关性分析

任务 DAG 图(Directed Acyclic Graph)刻画的是任务间固有的依赖关系，即数据相关性和控制相关性. 在 DAG 图中，结点代表一个任务，如果两个任务间存在依赖关系，则在相应的结点间连一条有向边，并在边上标注这两个任务的所有相关变量所占据的字节数总和. 当任务被调度到不同机器上执行时，该数字代表任务间的通信量.

下面就程序中的三种基本结构: 顺序、分支和循环分别进行讨论，通过分析其中的相关性来构造串行程序的任务 DAG 图.

3.1 顺序结构的相关性分析

在分析程序相关性、构造任务的DAG图的过程中，为了最大限度地挖掘串行程序的并行潜力，要尽可能消除其中的输出相关和反相关. 如果两个语句块在某个变量上存在输出相关或反相关，则可以通过变量换名的方式来消除，同时，还要引入额外的语句来消除换名带来的影响，保证换名前后程序的语义一致.

首先假定有 N 个语句块，它们的输入集和输出集合分别为 I_1、I_2、$I_3 \cdots I_N$ 和 O_1、O_2、$O_3 \cdots O_N$.

3.1.1 算法的具体步骤

首先在DAG图上为每个语句块建立一个结点，命名为结点 1…i. 将集合 S_{ij} 全部置空，用于保存语句块 i 和语句块 j 中存在流相关的变量. Rv 记录变量 V 的最后一次换名.

```
For i=2 to N do
   BEGIN
```

将 V 置空, 把语句块 i 的所有变量添加到 V

While VAR 集合非空

BEGIN

从 VAR 中取出一个变量 V

Case 1: V 出现在语句块 i 的输入集 I_i 中, 则语句块 i 与语句块 1…i-1 只可能存在流相关. 调用子程序 Proc1 处理

Case2: V 只在语句块 i 的输出集 Oi 中出现, 则语句块 i 与语句块 1…i-1 只可能存在输出相关和反相关, 调用子程序 Proc2 处理

Case3: V 在 I_i 和 O_i 中都出现, 则语句块 i 与语句块 1..i-1 关于变量 V 可能存在流相关、输出相关和反相关. 调用子程序 Proc3 处理

END

END

Proc1(i)

BEGIN

For J=i-1 to 1 do

if $I_i \cap O_j \neq \emptyset$ 则语句块 i 和 j 在变量 V 上存在流相关, 将 V 添加到 S_{ij} 中

END

Proc2(i)

BEGIN

语句块 i 修改了变量 V 的值, 之后的语句块 i+1…N 只受语句块 i 对 V 操作的影响, 如果语句块 i 与之前的语句块间存在反相关或输出相关, 则可以通过变量换名来消除.

For J=i-1 to 1 do

if $O_i \cap O_j \neq \emptyset$ 或 $O_i \cap I_j \neq \emptyset$

then 将语句块 i…N 中的变量 V 换名为 V’, 并修改 Rv 为 V’, RETURN

END

Proc3(i)

BEGIN

由于可能存在三种相关, 因此在换名的时候, 要确保不会丢失流相关.

For J=i-1 to 1 do

BEGIN

if $I_i \cap O_j \neq \emptyset$, 则将 V 添加到 S_{ij} 中

if $O_i \cap O_j \neq \emptyset$或 $O_i \cap I_j \neq \emptyset$, 则将语句块 i…N 中的变量 V 换名为 V’, 并修改 RV 为 V’, 此外, 还需要在语句块开始增加语句 V=V’, 确保换名不丢失流相关.

END

END

最后检查每个 S_{ij}, $1 \leq i < j \leq N$, 如果 S_{ij} 非空, 则语句块 i 与语句块 j 之间存在流相关, 在 DAG 上标连一条从结点 i 出发到结点 j 的边, 边上标注集合 S_{ij} 中变量的总字节数.

对每个变量 V, 如果 Rv 不为空, 则说明变量 V 经过了换名, 在所有语句块执行完毕后, 将变量 Rv 的值赋还给 V: V=Rv, 以保证整个程序语义正确.

3.1.2 算法复杂度分析

对于上述算法, 分析其复杂度, 大循环执行 N-1 次, 从语句块 2 开始逐个分析每个语句块, 第二个循环检查语句块 i 中的每个变量与之前的语句块是否存在相关性, 最多执行 M 次, M 是单个语句块中的最大的变量个数.

三个子程序 Proc1、Proc2、Proc3 中执行平均执行 N/2 次循环, 来检查语句块 i 和语句块 j 之间的相关性, 每次判断语句块输入输出集交集是否为空的复杂度为 O(M). 因此子程序的复杂度为 O(NM). 程序的主体循环的复杂度为 $O(N^2M^2)$. 最后, 根据 S_{ij} 建立 DAG 边的复杂度为 $O(N^2M)$, 将换名变量的值赋回给原变量的复杂度为 O(M).

根据上述分析, 可知算法的整体复杂度为 $O(N^2M^2+N^2M+M)=O(N^2M^2)$, 其中 N 是语句块数目, M 是单个语句块中变量的最大数目.

3.2 控制结构的相关性分析

在 if、switch 等复合语句中, 内部语句块是否执行依赖于条件部分的执行结果, 而 CALL 跳转语句改变了原来顺序执行的流程, 使其跳转到别处继续执行. 下面分别以 if、switch 和 CALL 语句为例, 来解释如何生成控制结构的 DAG 图.

① If 结构

```
if(cond) {
B1;
}
else {
B2;
}
```

设 T1 代表 cond 语句, T_2、T_3 分别代表 B_1 和 B_2 语

句块, 因此在 DAG 图上建立三个结点 T_1、T_2 和 T_3. 由于 T_2、T_3 是否能执行受 T_1 的控制, 所以结点轮廓用虚线来表示, 由两条从 T_1 出发的指向 T_2、T_3 的边来表示控制依赖关系.

② Switch 结构

```
switch(cond){
case c1: B1;
…
case cn: Bn;
}
```

与 if 结构类似, 设 T_1 代表 cond 语句, $T_2 \cdots T_{n+1}$ 分别代表 $B_1 \cdots B_n$ 语句块, 因此在 DAG 图上建立 n+1 个结点 $T_1 \cdots T_{n+1}$. 由于 $T_2 \cdots T_{n+1}$ 是否能执行受 T_1 的控制, 所以结点轮廓用虚线来表示, 由 n 条从 T_1 出发的指向 $T_2 \cdots T_n$ 的边来表示控制依赖关系.

③ 跳转指令 CALL

```
T1;
CALL Proc;
Proc { T2;  }
```

调转语句改变了程序原先顺序执行的流程, 因此在 DAG 图中, 建立两个表示 T_1 和 T_2 的结点, 并连一条从 T_1 到 T_2 的边代表这种依赖关系.

3.3 循环结构的相关性分析

在程序执行时, 大量的时间是花费在循环结构上. 因此, 如果能挖掘循环结构中潜在的并行性, 就能够大大提高程序执行的效率. 有之前的分析可知, 流相关是限制程序并行化的决定因素, 输出相关和反相关都是可以消除的, 同样, 在循环结构中, 我们可以消除这两种相关, 只保留流相关. 对于循环结构, 不但要考虑循环结构体内部的相关性, 还要考虑循环迭代之间的相关性.

根据循环迭代间是否存在相关性, 循环结构的相关性可进一步分为循环交叉依赖和循环独立依赖两种[4]. 循环交叉依赖指的是循环的不同迭代之间存在流相关, 循环体内部可能没有流相关. 而循环独立依赖的每次迭代之间是独立的, 但循环体内部依赖可能存在流相关.

下面是一个循环交叉依赖的例子, 循环的不同迭代关于 a 数组存在流相关, 同时在循环体内部在 b 数组上同样存在流相关, 这样的循环不能够并行执行

```
for(i=0; i<N; i++)
{
  a[i]=a[i-1] + b[i];
  b[i]= b[i] + c[i]
}
```

要判断循环体内是否有流相关, 可以由之前顺序结构相关性的分析方法, 两条语句之间不存在流, 当且仅当 $I_2 \cap O_1 = \varnothing$, 其中 I_2、O_1 是语句的输入输出集. 而对于循环交叉依赖, 则可以通过 gcd 和 Banerjee 不等式[5]来判断.

对于循环独立依赖, 可选取合适的粒度, 分割循环下标, 分解为多个循环, 在 DAG 图上, 表现为若干个独立的结点. 对于循环交叉依赖, 则可以采取循环体分裂的方法, 将循环体中具有流相关的语句分在一组, 将循环按组数分裂成若干个独立循环, 这些循环可以并行地调度执行.

3.4 生成串行程序的任务 DAG 图

之前的 3.1、3.2 和 3.3 节分别介绍了如何分析程序中顺序、循环和分支结构的相关性. 基于上述内容, 容易得出串行程序任务DAG图的构造方法, 具体步骤如下:

step1: 根据处理器执行速率等条件选定一个合适的粒度, 即 DAG 图中每个结点包含的语句数.

step2: 对于程序中的循环结构, 首先计算其语句数, 如果大于选定的粒度, 则分析是否存在循环交叉依赖和循环独立依赖, 并利用 3.3 节中的方法将其循环结构转换为 DAG 图.

step3: 对程序中的分支结构, 按照3.2节中的方法将其转换为 DAG 图中的结点.

step4: 分析完循环和分支结构后, 按照选定的粒度将程序划分为一个个的程序块, 最后利用 3.1 节中的算法分析顺序结构, 生成完整的程序 DAG 图.

4 应用举例

在分析了程序的三种基本结构的 DAG 图构造方法后, 再来看一个包含了顺序、控制、循环这三种结构的程序. 我们将使用之前的方法来构造出它的 DAG 图. 并介绍如何利用DAG图在多核共享内存环境下并行执行程序, 提高效率.

```
int f(int n)
{
  int ans=0;
```

```
    for(int i=0;i<1000;i++) ans+=i;
}
int main()
{
    for(int i=1;i<2000;i++) x =x+ i; //B1
    for(int i=1;i<2000;i++) y=y+2*i; //B2
    for(int i=1;i<1500;i++) z=z+(x+y)/i; //B3
    if (f(n)<50000) { //B4
        a[0] = 0; a[1] = 1;
        for(i=2;i<2000;i++) a[i]= a[i-1]+a[i-2];
    }
    else {
        b[0] = 0; b[1] = 1; b[2] = 1;
        for(i=3;i<3000;i++) b[i]= b[i-1]+b[i-2]+b[i-3];
        }
        //B5
        for(int i=0;i<100;i++)
            for(int j=0;j<60;j++) {
                c[i][j]=x/i+y/j+a[i];
                d[i][j]=c[i][j];
            }
}
```

根据之前的算法, 生成的任务DAG图如图1所示, 采用的语句粒度为 2000 条, 其中 B5 循环块循环的每次迭代之间都不存在相关性, 能够分解为若干个独立的循环.

该 DAG 图提供了在多核共享内存环境下的任务并行调度执行的依据, 假设现有4个处理器, 首先B1、B2、B4 语句块相互独立, 可以被指派到 3 个 CPU 上执行, 当 B4 执行完毕后, B4 的结果决定了将执行 B4.bran2, 而不会执行 B4.bran1, 此时图上与 B4.bran1 相关的结点和边都可以删去, B1、B2 执行完后, 为 B3 分配一个处理器执行, 当 B3 执行完后, 为 B5.1、B5.2 和 B5.3 分配 3 个处理器并行执行.

5 结论

本文首先介绍了串行程序并行化的一般步骤, 接着引出了程序并行执行的条件——相关性理论, 并详细讨论了数据相关性和控制相关性, 利用换名的方法消除了数据相关性中的输出相关和反相关, 给出了包含顺序、分支和循环这三种结构的程序的任务 DAG 图构造算法.

最后举了一个构造出的DAG图实例, 并说明如何根据其让程序在多核共享内存的环境下并行执行.

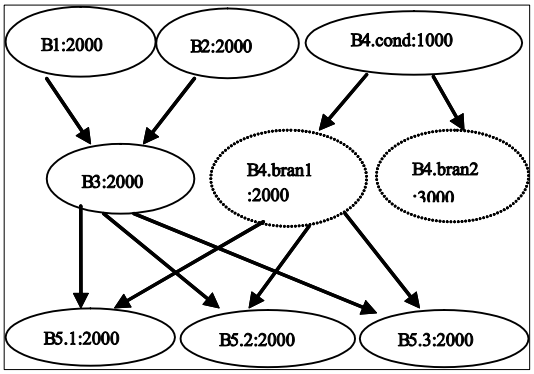

图 1 串行程序的任务 DAG 图

参考文献

1 Smotherman M, Krishnamurthy S, Aravind PS, Hunnicutt D. Efficient DAG construction and heuristic calculation for instruction scheduling. Proc. of the 24th International Symposium on Microarchitecture. Albuquerque: IEEE Computer Society 1991,93–102.

2 郭龙,陈闵中,叶青.构造串行程序对应的并行任务 DAG 图. 计算机工程与应用,2007,43(1):41–43.

3 闫昭,刘磊.基于数据依赖关系的程序自动并行化方法.吉林大学学报,2010,48(1):94–98.

4 曾国荪,陆鑫达,王景村.异构计算环境中循环级并行性提取及调度.计算机工程,2000,26(3):52–55.

5 Bacon DF, Graham SL, Shap OL. Compiler transformations for high performances computing. ACM Computing Surveys, 1994,26(4):345–420.

6 Kumar M, Patnaik LM. Automatic Loop Parallelization. ACM Computing Surveys, 1997,40(6):301–400.

7 Lilja DJ. Exploiting the parallelism available in loops. Computer, 1994,27(2):13–26.

8 Hwang YS, Saltz JH. Identifying parallelism in programs with cyclic graphs. Journal of Parallel and Distributed Computing, 2003,63(3):337–355.

9 Briš R. Parallel simulation algorithm for maintenance optimization based on directed Acyclic Graph. Reliability Engineering and System Safety, 2008,93(6):74–88.