第二届“数学中国杯”数学建模网络挑战赛

承 诺 书

我们仔细阅读了第二届“数学中国杯”数学建模网络挑战赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与队外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其他公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们将受到严肃处理。

我们允许数学中国网站(www.madio.net)公布论文，以供网友之间学习交流，数学中国网站以非商业目的的论文交流不需要提前取得我们的同意。

我们的参赛报名号为：1176

参赛队员 (签名) ：

队员 1：谢旭

队员 2：肖良

队员 3：马琼敏

参赛队教练员 (签名)：

参赛队伍组别：大学组

第二届“数学中国杯”数学建模网络挑战赛

编 号 专 用 页

参赛队伍的参赛号码：（请各个参赛队提前填写好）：

1176

竞赛统一编号（由竞赛组委会送至评委团前编号）：

竞赛评阅编号（由竞赛评委团评阅前进行编号）：

2009年 第二届“数学中国杯”
数学建模网络挑战赛

题　　目　串行算法并行化的研究

关 键 词　数据相关 控制相关 依赖方向向量 仿真测试 负载平衡

摘　　要

并不是任意的串行程序都可以转化为并行程序。本文主要对常规的串行程序的三种基本结构：顺序、分支、循环进行分析，探究它们可并发执行的判据，并把可并发的程序段分离出来；然后给出了一个估算循环结构计算量的公式，再根据估算的各个并发程序段的计算量，运用0—1规划，分配到两个处理器上。

两个语句不能并发执行主要是因为它们对同一个存储单元进行了读操作和写操作，如果两个语句有了这种关系，我们称它们数据相关。为了有效地判定语句之间的这种数据相关性，我们借鉴了Bernstein准则，提出了简单的顺序结构程序并行化的算法，将简单的顺序结构程序分成若干可以并发执行的程序段。

分支结构中为了判断各语句能否并行，不仅要判定控制依赖性，还要判定数据相关性。为了直观地判断，我们基于图论的思想提出了控制依赖关系图，用于判定分支之间是否有控制依赖性。对没有控制依赖的分支进而判定数据相关性，只要不存在数据相关，它们就可以并行执行。

对于循环结构，我们定义了依赖方向向量，并就依赖方向向量的数值特征给出了循环结构中各层循环是否可并行化的判据，并推导出了循环结构的计算量公式。

到此为止，我们已经得到了若干可以并发执行的程序段，设它们的计算量依次为 $L_1, L_2, ..., L_n$。下面的工作是将它们分配到两个处理器上，使两处理器的计算量大致相等，避免出现某个处理器忙而另一个处理器闲的状态。为此，我们运用了0—1规划来求取最佳的分配方案，使得处理器的利用率最高。

最后，我们对上面的三种算法进行了计算机仿真测试，发现对简单的顺序结构采用并行化处理时程序的运行时间没有明显的缩短，这主要是因为我们的顺序结构程序并行化算法适合于规则的、运行时间大体相等的代码，对于复杂、不规则的代码难以保证负载平衡；对if-else结构，并行化的效果取决于具体的程序，对于分枝定界法等易并行任务有较好的效果；对循环结构并行化效果最明显，并行处理后的程序比处理前的程序大致要节约一半时间。

参赛队号　1176

所选题目　A

参赛密码 _____
（由组委会填写）

Abstract

In transforming a serial algorithm into a parallel structure, we need to check the validity of doing so in advance. We discuss the criterion of it through analyzing three structures of a regular serial program: sequence, embranchment and cycle. We develop a descriptive model and use it into separating and transforming the serial part into the parallel form. Finally, we put forward an equation to estimate the calculating time of each CPU, apply it to get the result and use 0-1 layout to distribute different tasks into two CPUs.

The reason of failure of paralleling executing two program sentences is mainly because they visit the same memory unit for reading or writing a data. Then they possess the data dependence. We turn to Bernstein Criterion to judge the data dependence of two sentences. Then we devise a simple method to parallel a serial structure.

The execution of every branch of an embranchment structure relies on the test result of the function if( ),it is called control dependence. Two sentences could be paralleling executed if they don't have control relied relativity. In benefit for intuitionistic judgment, we introduce control dependence graph based on the theory of graphs. Only if two sentences are data independent and control independent can them be paralleling executed.

As for cycle structure, we define rely direction vector and develop a criterion based on the value of this rely direction vector. The total calculating time mainly depends on the calculating time of the cycle part of a program, thus we illustrate an equation $L = Q(\prod_{i=1}^{m} N_i)$, at which Q is the calculating quantity of the cycle part and N_i is the upper limit of the ith layer of the circular variable.

Up to now, we have got several pieces of program to be paralleling executed. Assume they have the calculating quantity $L_1, L_2, ..., L_n$ respectively. The remain problem is to assign them to two CPUs and t try to let these two CPUs have the same calculating quantity, avoiding that one CPU is free but in contrast the other is busy to death. Thus, we apply 0-1 layout to obtain the best allotting scheme.

Last but not the least, we carry out computer simulation test towards the above three arithmetic and come to the final conclusion. To parallel a simple sequence structure will not make a big difference in the program executing time. In fact, it is mainly because this arithmetic for sequence structure has good effects on some regular code, but is rather difficult in guaranteeing the balance of load towards complex and erratic code. The effect of paralleling an if-else structure depends. It will have awesome result on some tasks that could be easily paralleled. The outcome of paralleling a cycle structure could be the best because it saves almost half the time of the regular serial executing process.

#1176

一、问题重述

并行计算机的发展，让国防和国民经济领域的重大的科学和工程的计算速度有了几个数量级的提高。但是现代科学计算的计算复杂度巨大，串行计算已远远不能满足需求，而并行计算充分利用了 CPU 资源，因此速度快、实时性好，在现代科学计算中优势越来越明显。但是编写并行程序需要很专业的领域知识，人们往往能够编写出高效的串行程序，却对并行程序的编写不甚了解，因此如何将串行程序并行化，成为亟待解决的问题。

假设并行计算机是双核的，将串行程序并行化就是将程序拆分为两个可以并行执行的子程序部分，交给对应的两个核心处理器进行计算，并且尽量使两个核心的运算量相当。

问题：

请你们建立合理有效的模型，并依据模型对现成的串行算法进行处理。将能够使用双核心并行处理的部分分解开，并分配到两个核心上同时运行。以期达到比单核 CPU 处理更快速的目的。

具体要求：假设算法是使用 C 语言写成的，代码里只有顺序执行、分支、循环三种结构。为简单起见，我们假设只对整型变量和整型数组进行操作，不需要调用已有的库函数。程序中所有的语句只包括简单的代数运算、赋值、条件分支语句（if-else 语句），循环语句（while 语句）。不包括其他语句。

二、 问题分析

2.1 串行算法并行化的必要性

一个 CPU 在任意时刻只能处理一条指令。在执行串行程序时，按顺序执行串行算法描述的这个“动作”集合。 然而，一个串行程序中往往有很多语句不是数据相关或控制相关的。例如：语句 A：“x=a+b”与语句 B：“y=c+d”。明显看出，这两个语句的输入和输出变量均不同，彼此不相互依赖。也就是说，只要输入变量值在之前就已经确定，那么 A 与 B 的执行结果不因执行顺序的改变而改变。A 可以先于 B 执行，B 也可以先于 A 执行，如果有两个 CPU 的话 A 与 B 也可以同时执行，得到的结果都是一样的。

串行算法并行化，就是把串行算法中不相关的语句拆开来分别由并行计算机上的两个 CPU 执行，执行完之后两个 CPU 再进行通信以获得最后结果。并行化程度越高，双核系统的利用率就越高。

2.2 串行算法并行化的技术关键

将一个串行算法并行化的技术关键无非就是解决三个问题：1.判断是否可以并行化；2.怎样并行化；3.是否是最佳的并行化。

（1） 在判断不同程序语句是否可以并行化时，可以考虑它们之间的运算变量是否存在相关性。如果两个语句的数据变量彼此独立时，显然它们可以并行化处理，而不影响程序执行结果。当两个语句运算变量存在一定关联性时，就只能在一个 CPU 上按顺序运行，否则将输出错误结果。如何判断不同语句之间运算变量是否存在相关性，可以将运算变量分为输入集以及输出集，讨论它们之间的关系而得出结论。

（2） 如果已经判断两个语句可以进行并行化处理，那么只需要将它们做上标志，由

两个 CPU 来分别处理。相当于可以建立两个队列，存放两个 CPU 所要处理的代码。当两个语句可以进行并行化处理时，就将它们分别送入这两个队列之中，由此实现并行化。

（3） 最佳的并行化算法应该尽量使两个CPU中语句的运算量大致相等，以提高效率。若有很多个可以并行执行的语句，但是存放进两个队列时确分布不均，造成一个 CPU 一直有任务，而另外一个 CPU 处理完一些语句之后就空等待，势必造成资源浪费。理想的情况应该是并行化后，两个 CPU 运算量相等，使得程序的运行时间相对串行算法可节省一半。因此本题目也要对并行化后两个 CPU 的运算量进行合理的配置，使得处理器的利用率最高。

2.3 串行算法不同结构的代码并行化方法

题目中规定代码里只有顺序、分支、循环三种结构。因此可分别研究对这三种结构的并行化方法，从而实现对整个串行算法的并行化。

（1） 顺序结构的程序可以根据数据相关性判断，当不同语句数据不相关时，直接分列为两部分并行化。

（2） 分支结构的各个分支要进行控制相关性判断。控制依赖关系导致程序流程的变化，如：语句 S:if（i==0）；语句 T：j=a+b；语句 T 可能执行也可能不执行，取决于语句 S 的测试结果，即语句T控制依赖于语句 S。当判断两个语句没有控制依赖性时，就可以并行化处理。

（3） 循环结构的并行化必须先求解循环体中的依赖性向量，然后根据依赖性向量的数值特征判定某层循环是否可以并行化（这一部分要涉及一些新的概念，因此我们将在模型建立中对它进行详述）。

2.4 两个 CPU 的计算量以及并行化优劣指标分析

一个程序的计算量主要是它里面循环的计算量，因此计算量的估计主要针对循环结构。对一个串行程序并行化处理之后，应该尽量使两个核心的运算量相当，两个核心的计算量应该接近总的可并行的计算量的一半。只有这样才能使负载均衡，不让其中一个 CPU 空闲等着。

三、模型假设

1. 假设串行算法用类C语言写成的，代码里只有顺序执行、分支、循环三种结构。
2. 假设代码中只有整型以及整型数组，并且不需要调用已有的库函数。
3. 假设程序中只有简单的代数运算，赋值运算，条件分支语句（if-else 语句），循环语句（while 语句）。
4. 假设不考虑各个CPU速度差异，并且不考虑并行执行带来的额外计算量。

（注：上述是对本文的全局性假设，对文中建模讨论过程中我们可能引入的新的局部假设）

四、模型建立

4.1 顺序结构并行化算法

4.1.1 数据相关性定义

设有程序段 P。用 I 表示 P 程序段中操作所要读取的存储单元集，用 O 表示要写入的存储单元集。如：P:x=y+1;那么 I(P)={y},O(P)={x}。

数据相关性就是程序段之间输入输出集具有公共的存储单元。对于顺序结构的程序，两条语句数据相关等价于它们可以并发执行。

4.1.2 数据相关性判据

判断数据是否相关，也就是判断程序是否能够并行执行，可以用 Bernstein 准则：若语句 P1 和语句 P2 满足以下条件：

（1）$I_1 \cap O_2 = \varnothing$ ，即 P1 的输入变量集与 P2 的输出变量集不相交；

（2）$I_2 \cap O_1 = \varnothing$，即 P2 的输入变量集与 P1 的输出变量集不相交；

（3）$O_1 \cap O_2 = \varnothing$，即 P1 与 P2 的输出变量集不相交。

则 P1、P2 的数据不相关，可以并行执行。

两个程序段能够并行执行的根本原因就在于它们所访问的存储单元不同，不会彼此干扰到对方的运算结果。在第一个条件中，P1 的输入变量集与 P2 的输出变量集不相交，表明 P1 的输入变量存储单元和 P2 的输出变量的存储单元不相同。如果有交集，说明其中 P1 某个输入访问的存储单元就是 P2 的输出写入的存储单元。此时如果 P1、P2 顺序颠倒执行的话，那么 P2 先将运算结果存入共同的存储单元中，而再运行 P1 时从此单元读入的数据已经改变，这样就会产生错误。因此，把这个条件作为判断是否能够并行执行的条件之一。

同理可分析条件 2 和 3。当这三个条件同时满足时，P1 和 P2 的存储单元没有任何交集，说明这两个程序段确实可以并行执行。

4.1.3 顺序结构程序并行化算法

先建立两个队列 A 和 B，用于存放两个 CPU 的执行语句。在理想情况下，程序具有良好的并行性，共有 n 功能程序块（P1，P2…Pn）。每个子程序块可以是语句集，函数的调用或简单的一条执行语句。不妨讨论简单的代数运算和赋值的情况，对于其他的形式可以同理推广。

将整个顺序执行程序并行化的算法如下：

（1） 初始化：将第一个子程序块送入队列 A 中，作为核 1 的执行单元；

（2） 运用 Bernstein 准则判断第二个子程序块与第一个子程序块的数据相关性，如果不相关，将其存入队列 B 中；如果相关，将其送入队列 A 中第一个子程序块之

后，此时队列 A 中的所有程序看成一个程序块，得到它的整体的输入集和输出集。直到找到第一个与当前状态下的队列 A 中的程序集不相关的一个子程序段。

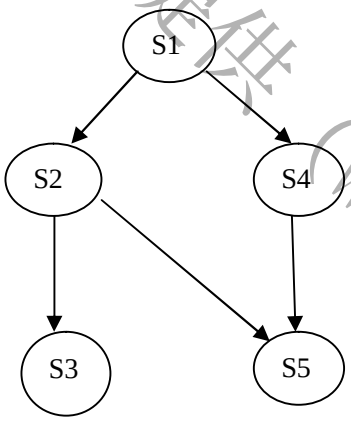
（3） 依次将接下去的各个子程序块与队列 A 与 B 中的程序集进行数据相关性比较，若仅跟 A 相关，则加入队列 A 中，若仅跟 B 相关，即加入 B 中，若与 A，B 均不相关，即加入 A 与 B 中已有程序的计算量较少的队列中，若与 A，B 均相关，则并行程序在此处分段，即两个处理器在此处要加入一个同步点，从下一段起重新分配。

上述算法仅对程序本身模块化、并发性较好的程序有较好的效果，且并行的粒度较大。 但是就一般情况而言，效果并不明显。

例如以下实例：

S1：A=1
S2：B=A+1
S3：C=B+1
S4：D=A+1
S5：E=D+B

显然其依赖关系图可表示为：



对于此例，若采用上述并行化算法，上述例子不能进行并行化，但如上图所示，语句 S2 与 S4 可以并发执行。可以考虑通过构造数据依赖关系的有向图，通过图的的层次关系将同一层次的语句（包括基本语句，复合语句及函数调用语句）并行化，但是由于并行的语句的计算复杂度往往不同，故这也很难均衡处理器负载，难以普遍适应。

4.2 分支结构并行化算法

4.2.1 控制相关性

对于单一的 if-else 结构（即 if-else 中不再嵌套 if-else），不能也没有必要对它进行并发性处理。但是对于嵌套的 if-else 结构，每一个分支下面又会有若干分支，这些分支有可能是并发的。若这些分支执行与否的判断条件不相关，并且各分支之间没有数据相关性，则这些分支是能够并发执行的。因此对于条件分支语句（if-else 语句），不仅要考虑各语句（语句集）之间的控制相关性，而且要考虑其数据相关性。

下面仅对嵌套的 if-else 结构进行讨论。在这种复杂的情况下，可以在编译原理中常用的控制流图（CFG）的基础上讨论。

在CFG中，一个结点表示一个基本块，而一条边表示两个基本块之间的控制流，若从结点 X 到出口结点的每一条路径都经过结点 Y，则称节点 Y 是结点 X 的后必经结点，记作 Y PDOM X，若 Y PDOM X 且 X≠Y，则称 Y 为 X 的严格后必经结点，记作 Y SPDOM X。

当 Y SPDOM X∧（∀W|W SPDOM X）[W PDOM X]时称 Y 为 X 的紧密后必经节点，计作 Y IPDOM X。可以把 PDOM 看作 CFG 节点集 上的一种偏序关系，可用 PDOM 树表示以这种关系。

故基于以上的讨论，可以给出控制相关性的定义。

CFG 中的两结点 X 和 Y，如果满足以下两个条件，则称节点 Y 控制依赖于 X，记为 X△Y：

(1) Y 是 X 的某些后继结点的后必经结点；

(2) Y 不是 X 的所有后继结点的后必经结点。

通俗的说，就是

(1) 从 X 的某一条出边到出口的路径总要通过 Y；

(2) 从 X 的另一些出边到出口的路径则可以比经过 Y。

另外，引入定义：

CFG 中的结点 X 和 Y，如果满足下述条件，则称 Y 以标记 l 控制依赖于结点 X：

(1) Y 是 X 的某个后继点的后必经结点

(2) Y 不是 X 的紧密后必经结点；

X 到 Y 的边为 $X \to_l Y$

4.2.2 控制相关性测试算法

给定一个 CFG 和对应的 PDOM 树，对于每一条 CFG 中的边 $X \to_l Y$，本算法求出以标记 l 控制依赖于 X 的所有结点。

(1) if Y=IPDOM（X），则 Y 不控制依赖于 X，end；

(2) else Z=Y；

(3) if Z≠Y；then

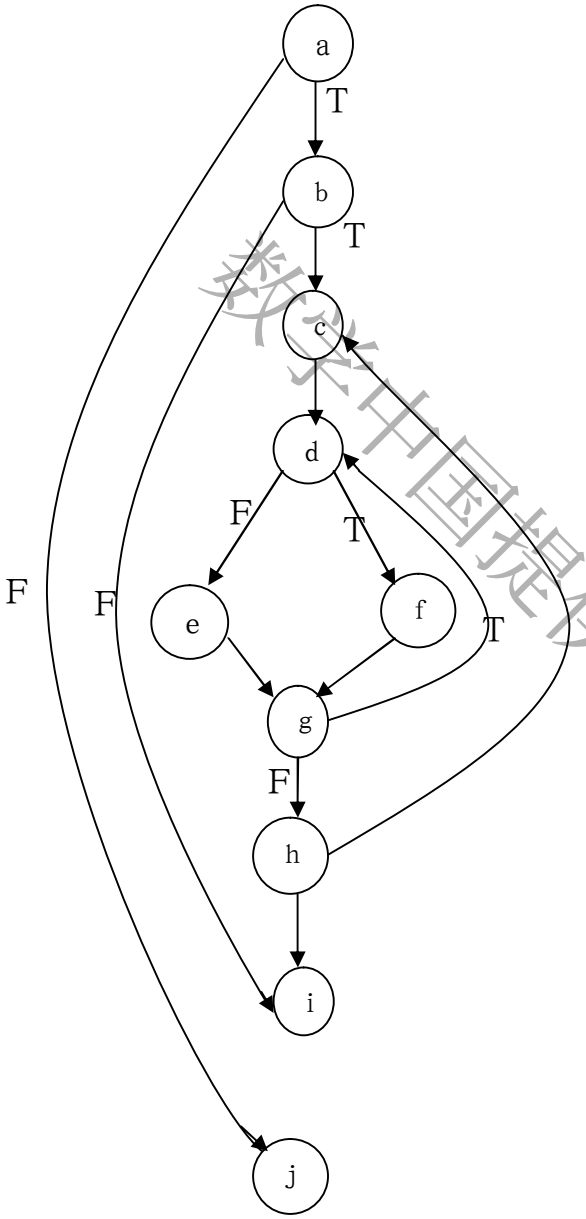
Z 以标记 l 控制依赖于 X；

Z=IPDOM（Z）

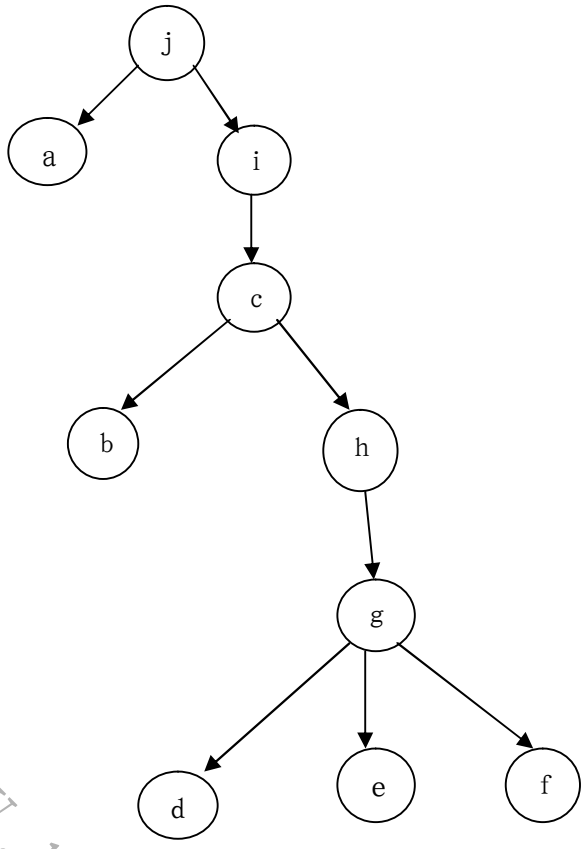
goto （3）

(4) else end;

例如,有如下所示 CFG，及相应的 PDOM 树，应用上述算法，可求得其带标记的控制依赖关系图（CDG）：



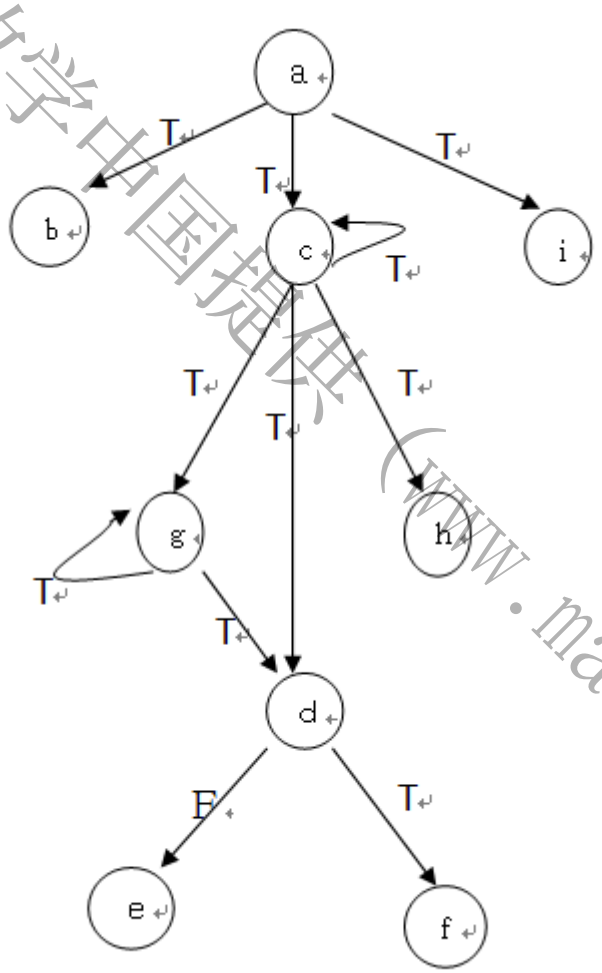
C F G



P D O M树

利用上述算法，得出其控制相关性图如下：

引起控制关系的 CFG 边	控制关系
$a \rightarrow_T b$	$a \triangle^T b$,$a \triangle^T c$,$a \triangle^T i$
$c \rightarrow_T d$	$c \triangle^T d$,$c \triangle^T g$, $c \triangle^T h$, $c \triangle^T c$
$g \rightarrow_T d$	$g \triangle^T d$, $g \triangle^T g$
$d \rightarrow_F e$	$d \triangle^T e$
$d \rightarrow_F f$	$d \triangle^T f$



CDG

4.2.3 分支结构并行化算

到此，我们已经解决了判断分支结构中控制相关以及数据相关的问题。在上面的 CDG 图中，我们可以将它看成一个层次关系图。a 是第一层节点。在第二层与第一层的关系是：b、c 以及 i 均控制依赖于 a，“T”表示只有 a 为真时它们才执行。而 b、c 以及 i

各自之间又没有连线，表明它们彼此不控制依赖。因此，我们可以根据 CDG 图，判断出不相互控制依赖的基本块。这些基本块可能可以并行化，但是还需判断它们是否数据相关。如果两个基本块既不控制相关性，而又不数据相关，那么它们就可以进行并行化操作。

总结以上分析，可以得出分支结构并行化算法：

（1） 将程序分为不同程序块，每个程序块作为一个基本块。
根据控制流图(CFG)，及相应的 PDOM 树，应用 4.2.2 的控制相关性检测算法，求得其带标记的控制依赖关系图（CDG）。

（2） 在 CDG 中分别将同一层中没有相互连线的基本块列出，如上图中第二层 b、c、i 第三层 g、h，第五层 e、f。再根据 4.1.2 Bernstein 准则判断每一层中的各个基本块之间是否数据相关。如果数据不相关，则可以进行并性化处理，将它们按 4.1.3 思想，在两个队列中并行化。如果数据相关，则将进行检测的这两个基本块按照 CFG 的方向叠加插入队列之中，在 CFG 较上层的先存入队列，而在下层的存入它之后。

（3） CDG 每一层的控制不相关的基本块并行化处理完毕之后，转入下一层以同样方式进行处理，一直进行到最后一层结束。

4.3 循环结构并行化算法

4.3.1 预备知识

Z^m 表示 m 维整数空间，Z^m 的元素 $i=(i_1,i_2,...,i_m)$ 是维数为 m 的整向量。

定义符号函数 $sign(x)$：

$$sign(x)=\begin{cases}1 & x>0\\ 0 & x=0\\ -1 & x<0\end{cases}$$

向量 $i=(i_1,i_2,...,i_m)$ 的符号 $sign(i)$ 是由它的元素的符号组成的向量，即 $sign(i)=(sign(i_1),sign(i_2),...,sign(i_m))$。

若向量 i 的每个元素都是 1，0 或－1，则称它为方向向量。显然，对任意的向量 i，$sign(i)$ 就是它的方向向量。

设 $i=(i_1,i_2,...,i_m)$，$j=(j_1,j_2,...,j_m)$，向量 $i<j$ 当且仅当存在整数 l $(1\le l\le m)$，使得

$$i_1=j_1,i_2=j_2,...,i_{l-1}=j_{l-1},i_l<j_l$$

等价地有：$i<j$ 当且仅当 $j-i$ 的方向向量 $s=(s_1,s_2,...,s_m)$ 满足：

$$s_1=s_2=...=s_{l-1}=0,s_l=1,s_{l+1}=...=s_m=*$$

这里*表示 1，0 或－1。

4.3.2 循环结构的并行性检测

在给出循环结构并发性检测的条件之前，先看一个例子：

```
FOR i=2 TO N DO
[S]  A(i)=B(i+2)+1     /*方便描述起见，用[S]表示该语句的序号为 S*/
[T]  B(i)=A(i-1)-1     /*方便描述起见，用[T]表示该语句的序号为 T */
END
```

显然：$A(1)\hat{I}O(S),A(1)\hat{I}I(T)$。S(1)写入 A(1)后，T(2)读 A(1)，S 和 T 不能同时对存

储单元 A(1)进行操作，因此 T 的执行依赖于 S。循环中所有关于数组 A 的下标可以表示成：

$$\{(S(i),T(j)): j = i+1, 1 \le i \le N-1\}$$

4.3.2.1 实例

考虑三层循环：

```
FOR  i1 =0 TO 100 DO
     FOR  i2 =0 TO 100 DO
          FOR  i3 =0 TO 100 DO
[S]            A[f1(i), f2(i), f3(i)] = A[g1(i), g2(i), g3(i)]     /*A 是三维数组*/
     END
END
```

其中 $i = (i_1, i_2, i_3)$，f_i，g_i $(1 \le i \le 3)$均是线性整数函数。当 i_1，i_2，i_3 取具体值时的 S，称为语句 S 的一个实例，记为 S(i)。

4.3.2.2 语句的依赖性

嵌套循环 **L** 中的语句 T 和 S，若存在 S 的一个实例 S(**i**),T 的一个实例 T(**j**),以及 S 的一个变量 x，T 的一个变量 y，且满足以下条件时，有语句 T 依赖于语句 S：

（1） x 和 y 至少有一个为所在语句的输出变量；

（2） x 在 S(i)中与 y 在 T(j)中为同一存储单元 M；

（3） 程序串行执行时，S(i)先于 T(j)而执行；

（4） 程序串行执行时，从 S(i)先执行结束到 T(j)开始执行前，没有对 M 的写操作

两个依赖的语句只能顺序执行，否则这种依赖性将使某个存储单元在同一时刻同时被读、写。

4.3.2.3 依赖方向向量

设语句 S 和 T 是 m 层循环嵌套中的语句，且 S(i)和 T(j)是 S 和 T 的实例(i，j 是 m 个循环变量组成的 m 维整数向量)，若 T(j)的执行依赖于 S(i)（即 T(j)只能再 S(i)之后执行），则令 $d = j - i$，$s = sign(d)$，称向量 s 是依赖方向向量。

为了更清楚这种定义方式，下面举一个例子，考虑下述循环：

```
FOR i=0 TO 5 DO
     FOR j=0 TO 4 DO
[S]        A(i+1,j+1)=A(i,j)+B(i,j)
     END
END
```

这是一个 2 层循环，它的实例可以表示成 $S(i_1, j_1)$，$S(i_2, j_2)$，则有 $i_2 = i_1 + 1$，$j_2 = j_1 + 1$，则它的依赖方向向量是（1，1）。

单层循环可以并行化的条件是：循环体中只有依赖方向为 0 的向量。

证明：假若依赖方向向量 $s \neq 0$（在一层循环中 s 是标量），则循环体中必有形如 $A(i+k)=A(i)+d$（其中 k 是整数，d 任意）的语句，那么数组 A 的第 $i+k$ 个单元的计算要依赖于第 i 个单元的结果，显然这个循环不能并发执行。

循环嵌套 $L=(L_1,L_2,...,L_m)$ 中的第 l 层循环 L_l 可以并行化的条件是：不存在 $s_1=s_2=...=s_{l-1}=0$，$s_l=1$，$s_{l+1},...,s_m$ 任意的方向向量【1】。

具体操作时可以先求取循环体中每条语句的依赖方向向量，再根据上面的判据即可判定第 l 层循环 L_l 是否可以并行化。

考虑如下循环：

```
L1：FOR i=2 TO N DO
   L2：FOR j=2 TO M DO
    [S]        A(i , j)=A(i , j)+1
    [T]        A(i , j+1)=2*A(i , j)
        END
   END
```

显然它的依赖方向向量分别为（0，0）和（0，1）。

L_1 并发的条件：不存在（1，*）的依赖方向向量。

L_2 并发的条件：不存在（0，1）的依赖方向向量。

因此，L_1 可以并发，L_2 不能。

4.3.3 循环的并行化算法

故对循环进行并行化的算法可简述为：

首先对各层循环进行并行性分析,通过计算各个依赖性方向向量 $(s_1,s_2,s_3.....s_n)$，然后通过上述循环是否可并发的判据，可以得出可以并发执行的循环 $(L_{p_1},L_{p_2}.....L_{pm})$，在为处理机分配时，应从最外层循环开始搜索，首先应当按照最外层可并发执行的循环（L_{p_1}）分配，当循环 L_{p_1} 共进行偶数次时，可均匀分配给两个处理器，此时两个处理器的负载大致相当。若循环 L_{p_1} 共进行奇数（2k+1）次，首先给每个处理器个分配 k 次循环，对于最后一次 L_{p_1} 循环，应按照次外层可并发进行的循环对两个处理器进行分配，分配方式与最外层一致；若各层循环都不满足可并发进行的条件，还可以考虑对循环提中的顺序语句采用上面提到的对顺序语句进行并行化的方法进行并行化处理，使两个处理器的负载尽量相等，提高并行执行的效率。

对于一个一般的串行程序，本模型假设程序中只有简单的代数运算，赋值运算，条件分支语句（if-else 语句），循环语句（while 语句），故可根据模型中的分类叙述对器进行并行化处理。

4.3.4 循环中计算量的估算

考虑如下 m 重循环：

L_1：FOR i_1=1 TO N1 DO

L_2：FOR i_2=1 TO N2 DO

……

L_m：FOR i_m=1 TO Nm DO

{循环体}

END

……

END

END

设循环中的计算量为Q，则总的计算量大致是$L = Q(\prod_{i=1}^{m} N_i)$。

4.4 并行任务的分配

设$L_1, L_2, ..., L_n$是n个可以并发执行的程序段的计算量。为了将它们合理地分配到两个处理器上并使二者的计算时间相当，这就要让两个处理器的计算量接近$\frac{1}{2}\sum_{i=1}^{n} L_i$。为此设一组 0——1 变量$x_i$ $(1 \le i \le n)$

$$x_i = \begin{cases} 0 & L_i \text{ 不在A队列中} \\ 1 & L_i \text{ 在A 队列中} \end{cases}$$

使下式取得极值的一种分配方法就是使两个处理器的计算时间相当的一种任务调度：

$$\min\left|\sum_{i=1}^{n} x_i L_i - \frac{1}{2}\sum_{i=1}^{n} L_i\right|$$

这样让$x_i = 1$的对应程序段在处理器 A 中执行，让$x_i = 0$的对应程序段在处理器 B 中执行。

五、仿真测试

为了对模型进行检验，我们分别对顺序、分枝、循环三种结构进行检验。

5.1 顺序结构：

S1：A＝1;
S2：D=1;
S3：B=A+1;
S4：C=B+1;
S5：E=2;
S6：D++;
S7：E=E-1
S8：F=(A+B+C+D+E)/5;

这样依据我们提出的顺序结构的并发性分解算法，即得：

QueueA	QueueB
S1	S2
S3	S5
S4	S6
S7	

QueueA 和 QueueB 是可以并发执行的，而 S8 必须在 A，B 处理器执行完毕之后才能执行。调用 clock( )系统函数（可以精确到 1ms），得到原来的串行程序运行时间（ms 级，必须将该程序执行 1000000 次才能看到结果）和分别执行 QueueA 和 QueueB 的时间，那么 max(t(QueueA),t(QueueB))近似为双核处理器并行处理所用的时间。可以发现，二者运行时间相当。这说明对于简单的顺序结构，并行化并不能明显缩短串行程序的运行时间。

5.2 分支结构:

分支结构的并行化的测试较为复杂，限于能力和时间有限，在此略去。但可以得到一个定性的认识：并行化的效果取决于具体的程序，对于分枝定界法等易并行任务有较好的效果；

5.3 循环结构

考虑如下二重循环：

```
T1=CLOCK( )
FOR i=1 TO 500 DO
    FOR j=2 TO 500 DO
        A[i][j]=A[i][j-1]*2
    END
END
T2=CLOCK( )
OUTPUT (T2-T1)
```

根据模型的判据得知，此程序的第一层循环可以并发，由于当 i 不同时，内循环的计算量大致相等，因此可以任意将 500 个内循环均分到两个处理器上。经计算，直接计算耗时 23.656000ms，并行执行耗时 11.616720ms。并发执行节约近一半时间。这说明串行程序并行化对循环较为有效，可以大大缩短程序的执行时间。

六、模型的进一步讨论

在解决运算数据的相关性时，还可以在不改变程序逻辑准确性的前提下先对程序进行语句重排，语句分裂，循环分裂或循环合并，还可以加入局部变量进行无关化处理，甚至可以在任务分配时加入动态调配成分，以提高并行化效果，尽量均衡负载。但是在串行算法并行化的问题上还可以进行深入探索，得到更为普遍适用的算法解决好这一问题。

七、模型优缺点

优点：

（1） 针对不同程序结构并行化进行建模，模块化思想强，理论性好，可操作性强。

（2） 建模过程，对准备知识阐述充分，定理的证明与分析紧靠问题实质。

（3） 模型具体，可操作性强。

缺点：

（1） 考虑程序趋于特殊性以及理想化，不适用于普遍串行算法。

（2） 只建立了描述性的模型，不利于定量分析且模型也不太严谨。

八、参考文献

【1】 沈志宇 《并行编译方法》 国防工业出版社 2000 年

【2】 陈国良 《并行计算》 高等教育出版社 1999 年

【3】 胡玥 高庆狮 高小宇 《串行算法并行化基础》 科学出版社 2008 年

【4】 David E.Culler Jaswinder Pal Singh Anoop Gupta
《Parallel Computer Architecture》China Machine Press 2003 年

【5】 白中英 杨旭东 《并行机体系结构》科学出版社 2006 年

【6】 徐士良 《计算机软件技术基础》 清华大学出版社 2006 年

【7】 张德富 《并行处理技术》 南京大学出版社 1992 年