第7卷第1期
1996年1月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 7, No. 1
Jan. 1996

一个新的多分支全局软件流水方法*

汤志忠　张赤红　陈　刚

（清华大学计算机系　北京　100084）

摘要　在指令级并行性很高的体系结构中，为了得到比较好的并行优化效果，通常需要设置多个分支控制机构，本文提出一个新的支持多个分支操作并行执行的全局软件流水方法——GPMB. 并用衡量全局软件流水方法性能的两个主要参数：时间开销和空间开销把我们的方法与其它几种全局软件流水方法进行了比较. 模拟实验结果表明：GPMB 方法的时间开销和空间开销都比较小，所需要的硬件支持也比较少.

关键词　多分支循环，软件流水，指令级并行，多路分支开关，多分支折叠，处理机.

软件流水是开发指令级并行性的一项重要技术.[1,2]采用全局代码压缩技术和全局软件流水技术，编译器可以安排多个操作并行执行. 但是，如果机器在一拍内只能执行一个两路的分支转移，那么所有的流程控制操作就必须串行执行. 据统计，在一般程序中分支操作占1/5左右，这对于超标量和超流水线体系结构，由于其并行度通常只有2～4，因此只设置1个分支控制机构就已经够用. 但是，对于 VLIW 体系结构，由于其并行执行的操作有几十个，甚至几百个[3]，这时分支操作就构成了开发程序并行性的瓶颈.

要充分开发程序的并行性就需要机器具备在一拍内完成多路分支转移的能力. 目前国际上有如下几种方法：

Cydra－5 机器[4]提供了一种叫条件执行(Predicated Execution)的机制，指令中每个操作将有条件地执行. 编译器首先消去程序中的所有分支，同时给每个操作加上一个执行条件的表达式(称为哨位)，然后就可以象基本块一样进行软件流水. 程序在实际执行时，同时计算各操作哨位的当前值，如果当前值为假，说明相应操作不符合执行条件，其结果将被取消.

IBM－VLIW 机器[5]支持判定树型指令，执行时只有指令树中满足判定条件的操作才是有效的，但不管一个操作的执行条件是否满足都要占用一个功能部件，所以它实际上也是一种条件执行机制. 另外，树状的指令格式还提供了多路分支跳转的能力. 这种机器是为一种称为渗透流水调度法 EPPS(enhanced pipelining－percolation scheduling)的软件流水算法设计的.

* 本课题得到国家863高技术发展计划和国家自然科学基金的资助.作者汤志忠，1946年生，教授，主要研究领域为并行计算机体系结构，并行算法及并行编译技术. 张赤红，1964年生，讲师，主要研究领域为 VLIW 体系结构及编译技术. 陈刚，1967年生，助教，主要研究领域为并行算法及并行编译技术.

本文通讯联系人：汤志忠，北京 100084，清华大学计算机系

本文1994－09－12收到，1994－12－02定稿

条件执行机制会造成功能部件的大量浪费.在功能部件有限的情况下(在一台具体机器中,功能部件的数量总是确定的),这类方法的时间开销就会增大.另外,由于要为每一个功能部件设置一个条件控制机构,因此其硬件实现的复杂度也比较大.

多分支全局软件流水的另一个问题是空间开销,从完善流水[6]和 GURPR *[7]两个算法的结果可以看出,把多条指令流合并成一条指令流会引起指令的组合爆炸,从而大大增加空间开销.

为此,根据分支操作占整个程序 1/5 的比例关系,我们提出了一种新的支持多分支循环并行执行的全局软件流水方法——GPMB(global pipelinig of multi-branch)[8],并用这种方法实现了一个 C 语言编译器,在这个编译器上进行了大量的模拟实验.实验结果表明,GPMB 多分支全局软件流水方法的时间开销和空间开销都比较小,而且所需要的硬件支持也比较简单.

1　GPMB 方法的基本思想

1.1　多路分支开关

在图 1 中,操作 1、2、…、n 将在同一拍中执行,但每次只有其中的一个是真正被执行的.如果每次只选出一个有效的操作来执行,所需功能部件的数目等于这 n 个操作所需功能部件的种类数;而如果采用条件执行的方法,所需功能部件的数目等于这 n 个操作所需功能部件的总数.显然,后者可能比前者要占用更多的功能部件.

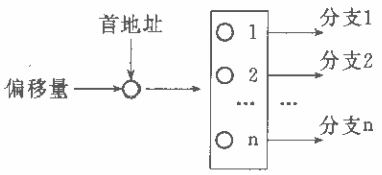


图1　多路分支开关

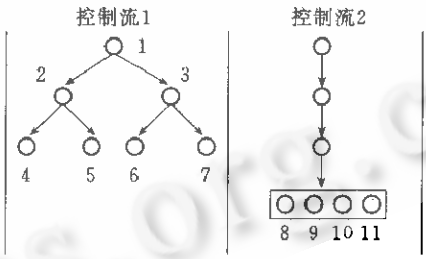


图2　在多控制流中使用多路分支开关

一种可在一拍中实现 n 选一的机构如图 2 所示,它把这 n 个操作编为一个操作组,放在指令存储器相连续的单元中.执行时,控制流根据首地址和偏移地址,就可以从 n 个操作中选出一个有效的操作来执行.这种机构好象一个多路开关,故我们称它为多路分支开关.

采用多路分支开关可以使分支控制机构相对比较简单,同时又可以省去条件执行机制所造成的功能部件的浪费.

1.2　控制流之间的协作

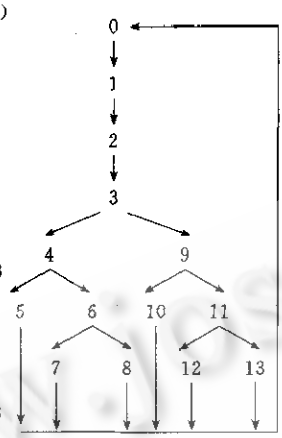
在图 2 中有两条控制流,其中,在控制流 1 中可以采用传统的两路分支转移;而在控制流 2 中,8、9、10、11 四个操作将在同一拍内执行,其执行条件分别和操作 4、5、6、7 一样.那么,为了在一拍中实现 4 路分支,可以采用多路分支开关.将操作 8、9、10、11 编成一个操作组,由控制流 2 的前趋操作给出本操作组的首地址,而由控制流 1 的 4、5、6 或 7 操作分别给出 8、9、10 或 11 操作的偏移量.程序执行时,控制流 2 从当前操作中得到下一个操作组的首地址,而控制流 1 从当前操作 4、5、6 或 7 中得到控制流 2 中的下一个操作的的偏移量,于是,控制流 2 根据本控制流给出的首地址和前一个控制流给出的偏址量计算出下一个要执

行操作 8、9、10 或 11 的地址. 这样,在控制流 1 的配合下,使用如图 1 所示的简单的多路分支开关,就能在控制流 2 中实现多路分支转移.

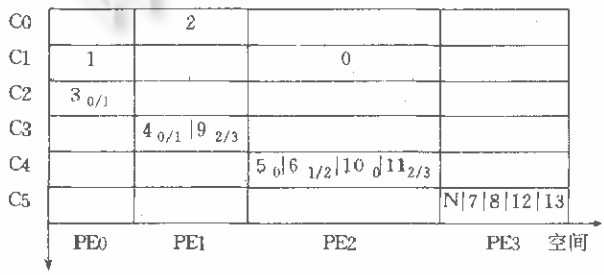
下面,通过一个具体例子说明一个多分支循环程序是如何编译和执行的.

```
for (x=0;x<imgx-1;x++)
  for (y=0;y<imgy-1;y+++)
0: {  r3=img[x-1][y];
1:    r1=img[x][y];
2:    r2=img[x+1][y];
3:    if (r1>r2)
4:    {  if(r3>r1)
5:       imgout[x][y]=r1;
6:       else if (r2>r3)
7:         imgout[x][y]=r2;
8:         else imgout[x][y]=r3
9:    }  else if (r1>r3)
10:      imgout[x][y]=r1;
11:      else if (r3>r2)
12:        imgout[x][y]=r2;
13:        else imgout[x][y]=r3
      }
```

(a) 串行程序



(b) 内层循环程序流程图



(c) 时空图

图3 多分支全局软件流水方法的一个例子(三值中值滤波程序)

1.3 一个例子

图 3 是一个中值滤波程序,为了简化起见,我们假设窗口的大小为 3x1,但一般情况下窗口的大小应为 3x3. 其中,图 3(a)是它的串行程序,图 3(b)是内层循环的程序流图,图 3(c)是一个循环体在时空图上的安放结果,时空图是一个两维的平面图,横轴表示处理机,给出操作的执行场地,纵轴表示时间,以节拍为单位.

在图 3(c)中,PEi 中一个操作的下标指出 PEi+1 中下一个节拍将要执行的操作的偏移量. 特别应当注意:有一些分支操作,如操作 4 有两个下标 0 和 1,这表示当条件为真时,传送给右邻 PE 的偏移量为 0,当条件为假时,则为 1. 按照图 3(c),一个循环体的执行过程是这样的:

周期 0:操作 2 在 PE1 上执行;

周期 1:操作 1 在 PE0 上执行,操作 0 在 PE2 上执行;

周期 2:操作 3 在 PE0 上执行,如果结果为真,则送往 PE1 的偏移量为 0,否则为 1;

周期 3:PE1 根据从 PE0 来的偏移量为 0 还是为 1 选择执行操作 4 或 9;如果操作 4 执行,那么传送给 PE2 的偏移量为 0 或 1,否则将由操作 9 给出偏移量 2 或 3;

周期 4:PE2 根据得到的偏移量选择执行操作 5、6、10 或 11;

周期 5:PE2 与 PE3 之间的合作和 PE0 与 PE1 或 PE1 与 PE2 之间的合作类似.

从这个例子中可以看出,当循环体内的数据相关和控制相关很严重时,单纯把循环体分配到多个 PE 上并不能得到很好的并行优化效果. 为了进一步开发循环体体间的并行性,对已经安放在时空图上的循环体还必须再进行一次软件流水,分别折叠 4 个 PE 上的程序流图,最终得到一个新循环体. 由于图 3 中的例子没有体间相关,因此可以每个节拍启动一个循环体.

如果把一个循环体分配到多个 PE 上的过程称为第一次软件流水,那么,在各个 PE 上分别进行流水折叠就是第二次软件流水.[9]通过两次软件流水可以充分开发程序中的指令级并行性,并且能充分利用硬件资源.

下面,介绍如何在 GPMB 优化编译器中实现图 3 中的变换过程.

2 循环优化方法

2.1 GPMB 方法的主要步骤

(1)循环体安放:首先生成循环体的数据和控制相关图(D&CDG),然后根据 D&CDG 调度循环体中的操作,确定一个循环体中每个操作的执行场地和启动时刻.这是第一次软件流水.

(2)寄存器分配:根据 D&CDG 和循环体安放的结果,为变量分配寄存器,并根据需要在循环体中加入寄存器间的数据传送操作.

(3)重建单个循环体在各个 PE 上的程序流图:把已经分配在各 PE 上的操作组织成程序流图,这是进行循环体折叠和生成目标代码的需要.

(4)分支折叠:为了进一步开发并行性,按照计算好的循环体启动间隔分别在各个 PE 上进行软件流水或分支折叠,并生成目标代码.

下面简要介绍其中的要点.

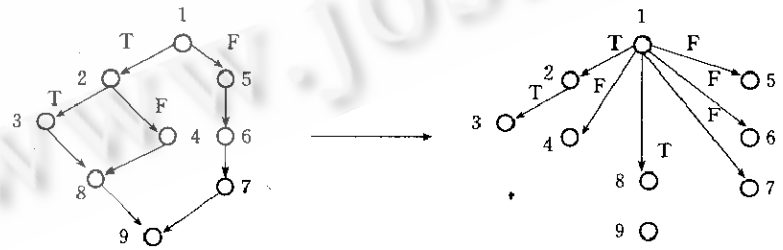
2.2 循环体安放

一个指令级并行的体系结构在一个节拍内有多个功能部件在同时工作,也就是说它是在一个二维的时空平面上执行程序的,纵轴代表时间,以节拍为单位,横轴代表处理机 PE,每个 PE 中又有多个功能部件.因此,循环体安放的目标是把一个一维的操作向量(一个循环体)映射到一个二维的时空平面上,即不仅要确定每个操作的启动时刻,还要确定其执行的场地.

在 GPMB 多分支全局软件流水方法中,循环体安放采用模调度法的思想[10],但对原来的模调度法作了许多重大的改进.

(1)控制相关的表示

在安放一个循环体时,不仅要考虑体内和体间的数据相关,还必须考虑其控制相关.因此,在原来数据相关图(DDG)中增加了表示控制相关的边,将它扩充为数据和控制相关图(D&CDG),在图 4 中举例说明了控制相关的表示方法.



注:由于op1控制op2,op2控制op3和op4,所以不需要从op1到op3、op4的边

图4 数据和控制相关图D&CDG中的控制相关表示

(2)资源占用的描述

有分支时的资源占用和无分支时不太一样,尽管仍然存在资源占用的模限制,但是在有分支时,属于不同执行路径上的两个操作,当且仅当它们在时空图上的启动时刻相同时,才可以占用同一个功能部件.

(3)数据和控制信号的传送

在不同的体系结构上安放带有分支的循环体时,对数据和控制信号的传送可以有不同的处理方法.如果有一个共享寄存器堆,则不必考虑数据的传送,如果是分布寄存器,则循环体调度时就必须考虑数据传送的时间.图3的例子是一个典型的分布寄存器体系结构(每个PE都有自己独立的寄存器堆,相邻PE同一编号的寄存器之间有直接数据通路).在周期0,PE1的r2的寄存器被装入数据,由于需要一个周期把数据从PE1的r2寄存器中传到PE0的r2寄存器中,因此PE0上的操作3最早要在周期2才可以引用r2寄存器中的数据.

另一种延迟是传递控制信息的时间.在图3(c)中,控制信息从PE0传到PE1,再传送到PE2,最后到PE3,这样就规定了分支中操作的执行顺序.

2.3 寄存器分配

在这一步中,不仅要给循环体中的操作指定读出和写入的物理寄存器,还要在循环体中插入寄存器间的数据传送操作.

带有分支的软件流水的寄存器分配比传统的寄存器分配更加复杂,这是因为:

(1)变量对每个寄存器的占用时间不能超过循环体的启动间隔.对于生存期大于循环体启动间隔的变量,需要分配多个寄存器,并在循环体中插入寄存器之间的数据传送操作.

(2)一个实际机器中的寄存器数目总是有限的.所以,一个实用的寄存器分配算法必须考虑当找不到可用的寄存器时如何处理.

在寄存器不够用时,我们综合采用了以下手段:(1)重新调整操作的启动时间;(2)在循环中加入spilling code;(3)增大循环体启动间隔.

有关寄存器分配方法的细节,将另文介绍.

2.4 在每个PE上重建程序流图

经过上述几步之后,循环体中每个操作有了确定的功能部件、寄存器和启动时间.但是,要生成目标代码,还需要把每个PE上的操作序列组织成程序流图的形式.和分支消去的过程相反,重建程序流图的过程是把一个没有分支的操作序列变成一个有多条支路的指令流,图5是这种变换过程的一个例子.

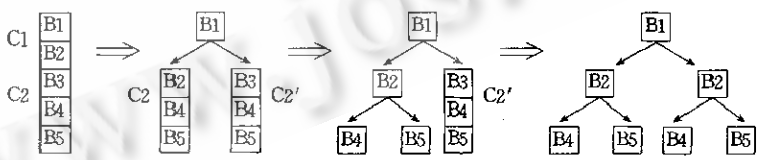


图5 重建程序流图

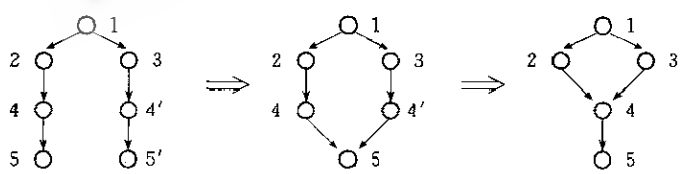


图6 全并程序流图中的冗余操作

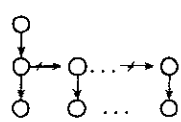


图7 程序流图中的操作组

重建程序流图的过程是这样的:因为一个操作序列的开头通常不是分支中的操作,所以首先从操作序列的头开始搜索,当遇到分支操作时,以下的操作序列就分成了两条执行路

径；然后再分别从两条路径的头开始，继续上述过程，直至循环体结束. 如图 5 所示，最终结果是一个树状的程序流图. 但是，一般来说，在程序流图中可能有重复的操作，对于有相同内容、相同启动时间和相同后继的操作结点可以进行合并，合并的过程应当从树叶开始. 合并程序流图中冗余操作的过程如图 6 所示.

在重建程序流图过程中，可能有如下两种分支操作：(1)由同一个 PE 内的分支操作产生的分支；(2)由多路分支开关产生的分支. 对于第 2 种分支，可以在一个节拍内产生多路分支. 为了表示这种分支操作，需要引入一种新的边把同一操作组中的操作串接起来，如图 7 所示.

2.5　分支折叠

当一个多分支循环体被安放到时空图上后，为了进一步开发其并行性，可以在各个 PE 上按照统一的循环体启动间隔分别作软件流水. 软件流水实际上就是程序流图的折叠，折叠的过程简单地讲可以分为两步：

(1)生成折叠后所有可能的操作组合作为新程序流图中的结点；

(2)根据原程序流图，计算新程序流图上的边.

图 8 是一般分支程序折叠的例子. 图 9 是有多路分支开关时，两个操作组的分支折叠.

应当指出，分支的折叠会因起操作复制. 但在我们的 GPMB 方法中，分支折叠是局部的，它只局限在一个 PE 上进行，与全局范围内的分支折叠相比，操作的复制量要少得多，因此 GPMB 方法的空间开销是很小的.

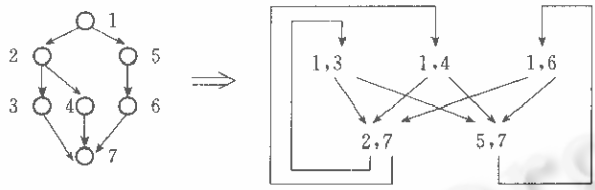


图8 分支折叠例1

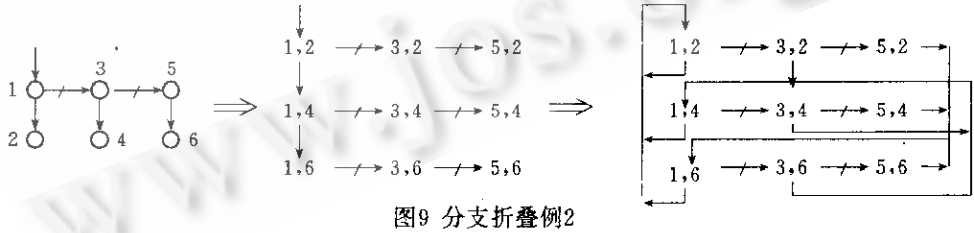


图9 分支折叠例2

3　实验与比较

目前，国际上采用的全局软件流水方法主要有 3 种. 下面，通过 6 个典型例子，把我们提出的 GPMB 多分支全局软件流水方法与这 3 种全局软件流水方法进行比较，这些方法是：(1)GPSB——单分支全局软件流水方法；(2)MSPE——条件执行机制下的模调度法；(3)EPPS——增强型流水渗透调度法.

一种软件流水方法的性能可以用两个主要参数来衡量：时间开销和空间开销. 一个循环

程序经过软件流水后生成了另一个与它语义完全等价的新循环程序,时间开销是指执行这个新循环程序所需要的时间,通常用节拍数来表示,它主要与在资源限制下循环体的平均启动间隔有关.由于实际上不可能为几种完全不同的软件流水方法规定一个统一的资源数目和资源种类,因此,我们用另外两个参数来间接表示时间开销,一个是在资源无限情况下的循环体启动间隔,另一个是在这种启动间隔下所需要的最大功能部件的数量.空间开销是指这个新循环程序的代码总量,通常用操作个数来表示,它主要与在生成新循环程序过程中操作复制的多少有关.在表1～6中我们用原循环体的操作数目加上在循环体优化过程中复制的操作数来表示.

与GPSB方法相比:只要循环体内多于一个分支操作,GPMB方法的时间开销均小于GPSB方法.即使用功能部件的时空积(循环体启动间隔与所用功能部件数的乘积)来比较,GPMB方法也优于GPSB方法.另外,因为GPMB方法的分支折叠被限制在一个PE内,而GPSB方法则必须在全局范围内进行分支折叠,因此GPMB方法的空间开销也小于GPSB方法.但是,由于GPMB方法支持多个分支操作并行执行,因此它需要的硬件支持比GPSB方法多.

与MSPE方法相比:在资源无限情况下,两种方法的循环体启动间隔是差不多的,但是,由于GPMB方法允许不同支路上的操作可以占用同一个功能部件,而MSPE方法则不管分支上的操作是否实际执行,每个操作都必须占用一个功能部件.因此,GPMB方法所需的功能部件数量通常要比MSPE方法少,如表2～6所示.那么,从另一个角度讲,因为在一台实际机器中,功能部件的数量总是有限的,因而GPMB方法的时间开销通常要比MSPE方法小.如表6所示,如果某机器有4个功能部件,则MSPE方法要用4个节拍做完一个循环体,而GPMB方法只要用1个节拍就能做完一个循环体,这样,GPMB方法的速度就是MSPE方法的4倍.另外,因为MSPE方法需要对每一个操作进行分支控制,所以控制硬件的复杂度比GPMB方法要高.但是,由于GPMB方法在各个PE上可能要进行分支折叠,需要复制少量操作,而MSPE方法则一定不复制操作,因此GPMB方法的空间开销可能要比MSPE方法稍差,如表5所示.

与EPPS方法相比:如果循环体启动间隔是唯一的,在资源无限条件下两种方法的空间开销是一样的.但是,对于表3、4中的两个例子,由于在一条分支上有体间数据相关,而在另一条分支上则没有,这时,因为GPMB方法是按照一个固定的启动间隔安放循环体的,而EPPS方法则允许不同执行路径可以有不同的启动间隔,因此GPMB方法的平均时间开销要比EPPS方法大.但是EPPS方法为了实现不同执行路径可以有不同的启动间隔需要使用较多的功能部件和相当复杂的分支控制机构,如果给定的功能部件数量较少,GPMB方法的时间开销通常比EPPS方法小.另外,由于操作在树型指令上的复制,为了达到同样的并行度,从表1、3、4中看出,MPPS方法的操作复制量都比GMBP方法大,因此,在大多数情况下GPMB方法的空间开销比MPPS方法小.

总之,因为GPMB方法可以有多个分支操作并行执行,其时间开销和空间开销通常都比GPSB方法小,又由于GPMB方法允许不同支路上的操作共用同一个功能部件,其功能部件利用率要比MSPE方法及EPPS方法高,而且在GPMB方法中分支的折叠被限制在一个PE内,代码的复制量也很少.另外,GPMB方法所需要的硬件支持虽然比GPSB方法多,

但要比另外两种方法少.然而,应当指出的是:要和准确地比较各种软件流水方法的性能是很困难的,因为,首先不可能从理论上证明,在任何情况下,一种方法的各项指标均优于另一种方法,其次很难有一个统一的公平的条件,对各种方法的实验结果做出统计.所以,以上的实验与比较有一定的局限性,我们只想说明 GPMB 方法对支持多分支循环程序的并行执行是一个很好的方法.

表 1　最大值滤波(窗口 3x3)

	GPSB	MSPE	EPPS	GPMB
启动间隔	8	1	1	1
部件数量	4	25	51	25
操作个数	25+18	25+0	25+26	25+0

表 2　二值图象收缩(窗口 3x3)

	GPSB	MSPE	EPPS	GPMB
启动间隔	3	1	1	1
部件数量	5	14	14	12
操作个数	14+14	14+0	14+0	14+4

表 3　求最大值和最小值

	GPSB	MSPE	EPPS	GPMB
启动间隔	3	2	1−2	2
部件数量	3	6	24	4
操作个数	11+6	11+0	11+28	11+0

表 4　大小写字母转换

	GPSB	MSPE	EPPS	GPMB
启动间隔	2	2	1−2	2
部件数量	5	6	14	5
操作个数	11+9	11+0	11+5	11+0

表 5　三值中值滤波(窗口 1x3)

	GPSB	MSPE	EPPS	GPMB
启动间隔	3	1	1	1
部件数量	4	14	14	7
操作个数	14+12	14+0	14+0	14+6

表 6　折半查找

	GPSB	MSPE	EPPS	GPMB
启动间隔	3	1	1	1
部件数量	2	15	15	4
操作个数	15+7	15+0	15+0	15+0

4　结　论

本文中,我们提出了一种新的支持多分支循环程序并行执行的全局软件流水方法,其基本思想是:把位于不同分支上的对应操作分成操作组,在流水安放时仅为每个操作组分配一套功能部件.在程序运行时,根据各控制流的当前状态、分支条件及前一个控制流送来的偏移量从操作组中选取满足分支条件的操作来实际执行.

GPMB 方法与目前流行的几种全局软件流水方法相比,其时间开销和空间开销都比较小,所需要的硬件支持也不多,而且适用于共享寄存器、分布寄存器、同步 MIMD 及 XIMD 等各种指令级并行的体系结构.

到目前为止,我们已经用 GPMB 多分支全局软件流水方法实现了一个 C 语言编译器,并且已经通过了几十个典型例子的编译.另外,与该方法配合的 VLIW 体系结构设计和模拟工作也已经完成.下一步,我们将进一步完善有关的设计,并准备实现一个可实用的优化编译器.

参考文献

1　Fisher J A, Rau B R. Instruct−level parallel processing. Science, Sept. 1991,**253**:1233～1241.

2　Jones R B, Allan V H. Software pipelining: an evaluation of enhanced pipeline. Proc. of MICRO−24, Nov. 1991. 82～92.

3　Wall D W. Limits of instruction−level parallism. Proc. of 4th Int'l Conf. ASPLOS, April 1991.

4 Dehnert C, Hus P T Y, Bratt J P. Overlapped loop support in cydra—5. Proc. of the Third International Conference on Architecture of Support for Programming Language and Operating System, 1989. 26～38.
5 Ebcioglu K. Some design ideas for a VLIW architectures for sequential—natured software. Proc. of the IFIP WG 10.3 Working Conf. on Parallel Processing, 1988. 1～21.
6 Allen R, Kennedy K. Conversion of control dependence to data dependence. Proc. of the Tenth Annual ACM Symposium on Principles of Programming Languages, Jan. 1983.
7 Su B, Tang Z, Wang J, Zhao W. A software pipelining based VLIW architecture and optimization compiler. Proc. of MICRO—23.
8 Tang Z, Zhang C, Zheng G, Su B. GPMB: a new software pipelining branch intensive loops. Proc. of MICRO—26, 1993.
9 Su B, Tang Z, Wang J *et al.* URPR—1: a single—chip VLIW architecture. EuroMicro Journal, 1993.
10 Labousse J, Slavenburg G A. CREATE—LIFE: a modular design approach for High performance ASIC's. Proc. of COMPLON 1990.

A NEW SOFTWARE PIPELINING BRANCH—INTENSIVE LOOPS

Tang Zhizhong Zhang Chihong Chen Gang

(*Department of Computer Science Tsinghua University Beijing* 100084)

Abstract To achieve higher instruction—level parallelism, the constraint imposed by a single control flow must be relaxed. Control operations should execute in parallel just like data operations. In this paper, we present a new software pipelining method called GPMB (Global Pipelining with Multiple Branches) which is based on architectures supporting multi—way branching and multiple control flows. Preliminary experimental results show that, GPMB performs as well as modulo scheduling, and for branch—intensive loops, GPMB performs much better than software pipelining assuming the constraint of one two—way branching per cycle.

Key words Branch—intensive loops, software pipelining, instruction—level parallelism, multi—branch switch, branch overlapping, proc