第 7 卷第 7 期
1996 年 7 月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 7, No. 7
Jul. 1996

多重循环的软件流水技术*

汤志忠 王 雷 钱 江

(清华大学计算机系 北京 100084)

摘要 为了解决多重循环的指令级并行编译问题,本文提出了反刍方法,以一种新的思维方式处理多重循环,将其视为一个程序流整体,有效地开发了多重循环的并行度.另外,本文还给出了实现反刍方法的基本步骤以及相应的硬件支持.最后,通过一些初步实验的结果验证了本算法的有效性,并讨论了其时间和空间效益,分析了其主要特点.

关键词 软件流水,多重循环,反刍方法,循环调度,指令级并行性.

软件流水是一项通过重叠不同次循环的循环体来开发循环程序指令级并行性的重要技术.[1~3]目前,国际上已提出了多种软件流水调度算法和启发方法[4~7],但它们都是针对最内层循环的.然而,在很多科学计算应用领域中,具有二层、三层甚至更多层嵌套的多重循环程序占据了大部分的处理机时间,而且它们的外层循环往往远远长于内层循环,且循环次数也较多.因此,软件流水技术必须能处理多重循环,才能在信号处理、图象处理等广泛科学计算领域中更好地开发指令级并行性.

本文针对这个问题,提出了反刍方法来实现多重循环的软件流水.该方法将多重循环视为一个循环程序流整体来进行软件流水,能取得较优的时间效益,并具有实用性,其算法复杂度与传统软件流水算法相当.第 1、2 节将分别介绍其基本思想和实现方法,第 3 节介绍反刍方法的硬件支持,第 4 节通过实验结果来分析反刍方法的性能、效益及其主要特点.

1 反刍方法的基本思想

让我们先来比较一下单重循环与多重循环的执行特点.

图 1 给出 2 种典型程序结构的模型.程序模型(a)是 1 个典型双重循环,(b)是 1 个典型的单重循环.其中,假设 $opi(i=1,2,\ldots,5)$在 2 个模型中代表同样的操作,且 2 种模型中的数据相关关系完全一样,如图 2 所示.

假设资源能满足需求,我们可以得到模型(b)的软件流水模型(图 3),其体间启动间距 Ⅱ 为 1.在图 3 中,从周期 4~n−1 重复的执行模式被称为流水模式.在流水阶段之前和之

---

* 作者汤志忠,1946 年生,教授,主要研究领域为并行计算机算法,体系结构及编译技术.王雷,女,1971 年生,硕士,主要研究领域为并行计算机算法,体系结构及编译技术.钱江,1970 年生,硕士,主要研究领域为指令级并行编译技术,虚拟现实.

本文通讯联系人:汤志忠,北京 100084,清华大学计算机系

本文 1995−06−09 收到修改稿

后的那 2 段分别称为装入阶段和排空阶段.

```
For ( i=1; i++; i<=n ) {
    op1 ;
    for ( j=1; j++; j<=m ){
        op2 ;
        op3 ;
        op4 ;
    }
    op5 ;
}
```

(a)

```
For ( i=1; i++; i<=n ) {
    op1 ;
    op2 ;
    op3 ;
    op4 ;
    op5 ;
}
```

(b)

图1 程序模型

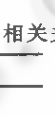

图2 数据相关关系图

周期	操作				
0	$op1(1)$				
1	$op1(2)$	$op2(1)$			
2	$op1(3)$	$op2(2)$	$op3(1)$		
3	$op1(4)$	$op2(3)$	$op3(2)$	$op4(1)$	
4	$op1(5)$	$op2(4)$	$op3(3)$	$op4(2)$	$op5(1)$
...	...	...	...	...	...
$n-1$	$op1(n)$	$op2(n-1)$	$op3(n-2)$	$op4(n-3)$	$op5(n-4)$
n		$op2(n)$	$op3(n-1)$	$op4(n-2)$	$op5(n-3)$
$n+1$			$op3(n)$	$op4(n-1)$	$op5(n-2)$
$n+2$				$op4(n)$	$op5(n-1)$
$n+3$					$op5(n)$

注：$OP_k(p)$ $(k=1,2,\ldots 5)$代表第 p 次 i 循环中的 OP_k.

图 3 对程序模型 1(b)进行软件流水

比较图 1 中的程序模型(a)和(b)，我们可以发现其主要区别就在于每次循环中操作 $OP2$,$OP3$ 和 $OP4$ 的执行次数. 如果模型(a)中 $m=1$，它们就完全一样(为了简化问题，我们忽略了模型(a)中用于控制 j 循环的操作). 再仔细观察图 3，我们可以发现，从周期 3 开始，每隔 3 个周期，就出现了一个由 $OP2(3k)$，$OP3(3k-1)$，$OP4(3k-2)$ $(1\leqslant k\leqslant m)$构成的执行模式. 该模式中的 3 个操作分别属于 3 次不同的 i 循环. 如果每当该模式出现后，我们就将程序的执行暂停，而重复执行该模式 $3(m-1)$次，再配以相应的控制机制，实际上就实现了程序模型(a)的软件流水，如图 4 所示(为简化讨论，我们假设 n 可被 3 整除. 当 n 不能被 3 整除时，我们可以用空操作代替实际不存在的循环体).

分析反马方法下多重循环运行的过程，我们可以将反马方法的基本特征描述如下：反马方法是 1 种适用于多重循环的软件流水方法，对其各层循环进行软件流水时，将该层循环内嵌套的各内层循环视为只执行 1 次. 每当 1 个由某内层循环体构成的新模式出现时(我们称该内层循环被激活了)，当前外层循环的执行暂停，转去对该内层循环软件流水，这时外层循环放弃对部分或全部资源的支配权，供该内层循环使用. 内层循环在完成其流水阶段，将进入排空阶段时，就返回父循环(即进入该内层循环前正在执行的那个外层循环)，并恢复其父循环对所有资源的支配权. 如果有 2 个以上的内层循环同时被激活，嵌套层次最深的子循环优先执行，若这几个循环的嵌套深度相同，只要数据相关关系和控制关系允许，它们可按任何顺序执行.

周期	操作						备注
0	$op1(1,-)$						外层装入
1	$op1(2,-)$	$op2(1,1)$					内层也装入
2	$op1(3,-)$	$op2(2,1)$	$op3(1,1)$				
3	$op1(4,-)$	$op2(3,1)$	$op3(2,1)$		$op4(1,1)$		内层充满，切换
4		$op2(1,2)$	$op3(3,1)$		$op4(2,1)$		内层流水，外层暂停
5		$op2(2,2)$	$op3(1,2)$	$op4(3,1)$			
…		…					
x		$op2(3,m)$	$op3(2,m)$		$op4(1,m)$		
$x+1$	$op1(5,-)$	$op2(4,1)$	$op3(3,m)$		$op4(2,m)$	$op5(1,-)$	第1次内层排空，同时外层装入、流水，
$x+2$	$op1(6,-)$	$op2(5,1)$	$op3(4,1)$		$op4(3,m)$	$op5(2,-)$	
$x+3$	$op1(7,-)$	$op2(6,1)$	$op3(5,1)$		$op4(4,1)$	$op5(3,-)$	同时也是第2次内层装入
$x+4$		$op2(4,2)$	$op3(6,1)$		$op4(5,1)$		再次内层流水，外层暂停.
…	…						
y		$op2(n,m)$	$op3(n-1,m)$		$op4(n-2,m)$		
$y+1$			$op3(n,m)$		$op4(n-1,m)$	$op5(n-2,-)$	最后，整个循环排空
$y+2$					$op4(n,m)$	$op5(n-1,-)$	
$y+3$						$op5(n,-)$	

注：$OP_k(p,q)$表示第 p 次 i 循环中第 q 次 j 循环的 OP_k 操作.

图 4 反当算法下的软件流水

2 反当方法的实现

在第1部分中，我们将1个单重循环的软件流水过程视为装入、流水、排空3个阶段间的切换过程. 类似地，我们也可以将1个多重循环在反当方法下的软件流水过程视为在若干个阶段间切换的过程. 一般说来，在反当方法下，多重循环的软件流水过程可包含多个流水阶段，最外层循环及每个子循环与各流水阶段一一对应；整个软件流水过程只有1个装入阶段和1个排空阶段，它们是对应于最外层循环的，在实际运行过程中，这些阶段可能由于某子循环的激活、切换而被分割成若干片段. 以图4为例，最外层的装入阶段包括周期0至3，排空阶段包括周期 $y+1$ 至 $y+3$. 周期4至 x 为内层循环的第1个流水阶段，周期 $x+1$ 至 $x+3$ 为外层循环的第1个流水阶段. 各阶段间的切换条件是由循环的结构决定的：某内层循环 L 第1次被激活的时刻可从最外层循环的装入阶段推算出来，以后就每隔 $T1$ 周期激活1次，$T1=LoopLength(L)$，其中，不包括因其子循环被激活、切换而导致 L 暂停所产生的延迟. 每当 L 被激活时，它将执行 $T2$ 周期后才返回其父循环(不包括因其子循环被激活、切换导致的暂停延迟)，$T2=LoopLength(L)*(Number\ Of\ Iteration(L)-1)$.

在以上讨论中，$LoopLength(L)$ 应理解为循环 L 在调度后执行1次所需时间(视其各内层循环只执行1次).

要实现反当方法，我们可用一段指令来对应一个阶段，并给最外层循环及每个子循环分配一个专用计数器来计算该循环激活的时刻和循环的次数. 各阶段间的切换可用JUMP或CALL等指令来实现，也可采用硬件机制. 因此，只要对嵌套的各循环分别进行软件流水(可直接采用传统的软件流水方法)，再用适当的机制把各段指令联系起来，就可实现反当方法.

至于先调度最内层循环，还是先调度最外层循环，属于技术问题，不在本文讨论范围之内. 在我们的实践中，为了更好地提高内层循环的时间效率(因为内层循环的总次数比外层循环的多得多)，我们首先调度最内层循环，并采用模调度法调度每个循环. 在模调度法下，一个长为 H 的循环体被划分为段，其中 Π 为该循环的体间启动间距. 这就要求在调度外层循环时，属于其内层循环的指令的分段形式要保持一致，否则就会因各层循环启动的体数不同而造成混乱.

虽然反馬方法的实现过程包含若干步骤，但其算法复杂度是由在调度各子循环时所采用的算法决定的. 设被调度循环包含 N 个操作，若采用模调度法，则其算法复杂度可为 $O(N^2)$.[8]

3　反馬方法的硬件支持

为了实现反馬方法，我们设计了反馬机体系结构来作为编译器设计的基础.

反馬机是根据 URPR－1 机[9]改造的，如图 5 所示，它主要由 8 个相同的处理器(PE)和流水寄存器堆、存储器及有关控制部件组成. 每个 PE 包含 1 个浮点加法器，1 个浮点乘法器，1 个 Load/Store 部件及本地 PC 控制逻辑和指令存储单元. 流水寄存器堆是该体系结构的核心，它采用分布式结构，由分属于各 PE 的局部流水寄存器堆构成，每个局部流水寄存器堆具有内部垂直方向流水的功能. 反馬机的一个重要特色在于虽然每个 PE 只能向其局部流水寄存器堆写入数据，但可以从任一寄存器堆中读出数据.

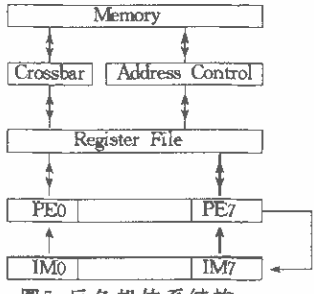

图5　反馬机体系结构

4　初步实验结果及分析

为了检验反馬方法的性能，我们根据 Perfect Benchmark[10]的 ADM 选编了 4 个多重循环来作初步实验. 由于我们目前的实验主要是手工模拟的，因此我们只是随机地选了几个较简单的例子. 由于缺乏其他可供比较的方法，我们选择了目前多数编译器所采用的简单处理法作对照，其实质就是只对最内层循环做软件流水，而其外各层循环都是串行执行的.

例子	S	包含的循环个数	K_0	II_0	K_{s-1}	II_{s-1}	II'_{s-1}	N_0	TR	TS	加速比 ($Speedup$)
1	3	4	39	4	8	1	1	20	8550	15600	1.82
2	2	2	13	1	10	1	2	50	1012	2550	2.52
3	3	9	52	1	40	1	3	40	100866	332800	3.30
4	3	3	29	2	24	2	2	40	16027	35320	2.20

注：S：该多重循环的层数；K_0：最外层循环执行一次的时间(假设其内各层循环只执行一次)；II_0：在反馬方法下，最外层循环的体间启动间距；K_{s-1}：各最内层循环执行一次的时间之和；II_{s-1}：在反馬方法下，最内层循环的平均体间启动间距；II_{s-1}：在简单处理法下，最内层循环的平均体间启动间距；N_0：最外层循环的循环次数；TR，TS：分别为反馬方法和简单处理法的总执行时间；$Speedup=TS/TR$：为反馬方法相对于简单处理法的加速比.

图 6 初步实验结果

图 6 显示了实验的有关数据. 从我们的初步实验可看出，与简单处理法相比，反馬方法

的加速比是令人满意的,这主要是由于在反刍方法下,内层循环的装入和排空阶段与外层循环的流水阶段重叠了.该加速比与很多因素有关,比如外层循环的次数、各层循环的长度及体间启动间距等循环结构上的特征.我们通过实验发现,在影响加速比的诸多因素中,最内层循环的体间启动间距的作用尤为突出,这主要是因为最内层循环的循环次数是各层循环中最多的,而在其循环次数固定的前提下,其体间启动间距是影响该循环总运行时间的主要因素.本实验中,例3表现出相对较高的加速比,这主要是因为其II_{s-1}与II'_{s-1}有明显差距.由于在反刍方法下,内层循环是串行的,因此其体间启动间距只受资源限制,与体间数据或控制相关关系无关,所以其II_{s-1}总是小于或等于II'_{s-1}的,而且在大多数情况下都可化为1.当内层循环的体间数据或控制相关较为严重时,传统的软件流水方法的应用就很受限制,而这时恰恰是反刍方法大显身手的时候.只要优化内层循环时不进行跨越式安放,即让属于不同次外层循环的内层循环体进行重叠,都不可能获得这种好处.而且,从反刍方法的执行方式我们可以发现,当循环嵌套结构越复杂,循环次数越多时,其时间效益比之于其他的方法就越优越,而在空间开销上的增长却很有限,而且不会给实际编译带来压力.

虽然本文没有给出有关寄存器需求的数据,我们承认一个多重循环对寄存器的需求量在反刍方法下是明显大于在简单处理法下的.这是由于反刍方法是直接对整个多重循环进行软件流水,其中涉及的数据与控制相关要比只对最内层循环进行软件流水时大得多,而且涉及的变量数也多得多.考虑多重循环的这些特点,我们认为这种现象是正常的,并不是反刍方法的缺点,另一方面,采用合理的寄存器分配算法,可以大大减少反刍方法下多重循环对寄存器的需求量.

5 结 论

本文的核心是为解决多重循环软件流水而提出的反刍方法,其关键在于从最外层循环着眼进行软件流水.循环层次切换的思想是反刍方法的灵魂,是将各层次的循环融合在程序整体中的粘合剂,又是多重循环能够连贯、高效运行的润滑剂.

从用手工模拟编译、运行的结果看,反刍方法使具有任意嵌套结构的多重循环的软件流水得以实现,并具有较优的时间与空间效益,算法复杂度及实现难度都不大,是对多重循环软件流水课题的有益尝试.反刍方法的价值就在于它有一套较完整的思想,能使多重循环高效率"转"起来.

参考文献

1 Rau B R, Glaeser C D. Some scheduling techniques and an easily schedulable horizontal architecture for high performance scientific computing. MICRO—14, Oct. 1981. 183~198.

2 Aiken A, Nicolau A. A realistic resource—constrained software pipelining algorithm. In: A. Nicolau, D. Gelernter, T. Gross *et al* editors, Advances in Languages and Compilers for Parallel Processing, Res. Monographs in Parallel and Distrib, Computing, Chapter 14, 1991. 274~290.

3 Lam M. Software pipelining: an effective scheduling technique for VLIW machines. Proc. of the SIGPLAN'88 Conf. on Programming Language Design and Implementation, Jun. 1988. 318~328.

4 Gasperoni F, Schwiegelshohn U. Efficient algorithms for cyclic scheduling. Res. Rep. RC 17068, IBM T. J. Watson Res. Center, Yorktown Heights, NY, 1991.

5 Huff R A. Lifetime—sensitive modulo scheduling. In: Proc. of the SIGPLAN'93 Conf. on Programming Language Design and Implementation, Jun. 1993. 258～267.

6 Rau B R, Fisher J A. Instruction—level parallel processing: history, overview and perspective. In: J. of Supercomputing, 7, May 1993. 9～50.

7 Wang J, Eisenbeis C, Jourdan M *et al*. DE—composed software pipelining: a new approach to exploit instruction—level parallelism for loop programs. Res. Rep. RR—1838, INRIA—Rocquencourt, France, Jan. 1993.

8 Rau B R. Iterative modulo scheduling: an algorithm for software pipelining loops. In MICRO—27. , Nov. 30～Dec. 2, 1994. 63～74.

9 Su B, Wang J, Tang Z *et al*. A software pipelining based VLIW architecture and optimization compiler. MICRO—23, 1990. 17～27.

10 Berry M *et. al*. The perfect club benchmarks: effective performance evaluation of supercomputers. Tech. Rep. CSRD Rpt. No. 827, Center for Supercomputing Research and Development, University of Illinois, 1989.

SOFTWARE PIPELINING ON PROGRAM WITH COMPLICATED LOOPS

Tang Zhizhong Wang Lei Qian Jiang

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

Abstract This paper discusses about the problem of software pipelining on complicated loops. It first introduced a software pipelining method called ruminate method, which can optimize program with complicated loops. Then it outlined the procedure to realize ruminate method and described the hardware support. The performance of ruminate method is analyzed at the end of this paper by the aid of the preliminary experimental results.

Key words Software pipelining, complicated loops, ruminate method, loop scheduling, instruction—level parallelism.