多重循环的软件流水:比较和提高

李文龙　汤志忠
(清华大学计算机科学与技术系　北京100084)

摘　要　循环并行化是并行编译的核心问题之一。许多科学计算程序的大部分执行时间花费在循环上,有效开发循环中的并行性将提高整个程序的执行效率。多重循环最为常见,因此并行化多重循环具有重要的理论和现实意义。现代处理器中硬件资源迅速增长,也使得在整个多维循环空间中开发并行性成为必要。目前大多数软件流水算法只对最内层循环,仅有少数的算法对多重循环进行软件流水,本文介绍几种多重循环的软件流水算法,比较它们之间的相似与不同之处,为编译器实现中算法的选择提供了指导。

关键词　软件流水,启动间距,并行性识别,多重循环

The Software Pipelining of Multiloops:Comparison and Improvement

LI Wen-Long　TANG Zhi-Zhong
(Department of Computer Science & Technique, Tsinghua University, Beijing 100084)

Abstract　Loops parallelization is one of key problems of parallelizing compile. A lots of Science Computation programs spend greater part of execution time on loops. Effectively developing parallelism in loops will improve execution speed of the entire program. This paper introduces some of software pipelining algorithms of multiloops, and compares similarities and differences between them.

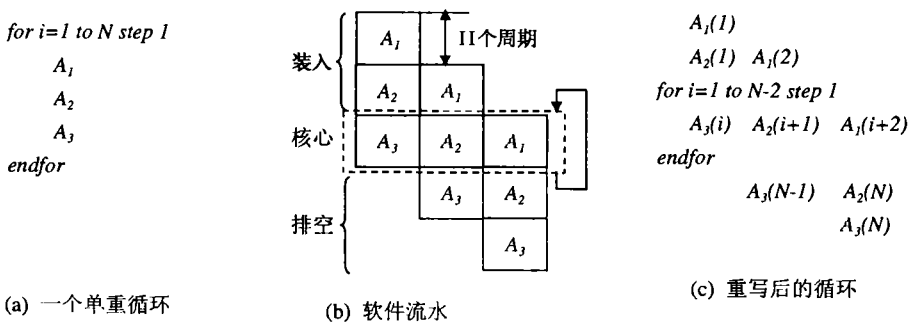
Keywords　Software pipelining, Initiation interval, Parallelism, Multiloop

1　介绍

软件流水[1]通过重叠不同循环体的执行来获得加速。在一个循环体(Iteration)未执行完毕之前,可以启动下一个循环体。二者的启动时刻之差称为启动间距(II: Initiation Interval)。所谓"调度"(Schedule),就是为每个循环体中的操作实例(Operation instance)赋予一个调度时刻(Schedule time)。

最优软件流水问题是如何在硬件资源限制和循环本身的相关限制下,以最短时间执行完循环,这是一个多机调度问题,一个NP难题。因此,不得不寻找启发式,希望以较小的时间复杂度,得到近优解。

最重要的启发式之一是模调度[2]:所有循环体按照一个固定的启动间距依次启动,每个循环体的调度都相同。调度结果由装入、核心和排空三部分组成。在装入部分,前 v 个循环体以 II 为间隔依次启动,直至出现一个重复模式,即核心。核心高度等于 II,它反复执行直到所有循环体都被启动。离开核心后进入排空部分,完成最后 v 个已启动但未执行完的循环体。以每 II 个周期为一个阶段($Stage$),一个循环体可以划分为 S 个阶段。若 Len 为循环体长度,则 $Len = S * II$。



(a) 一个单重循环　(b) 软件流水　(c) 重写后的循环

图1　经典的单重循环模调度软件流水

例如,对于图1(a)中的循环,经过软件流水后,所得调度如图1(b)所示。其中,每个循环体有3个阶段 A_1、A_2、A_3,循环体长度为 $3 * II$,虚框内是核心。可以等价地将此调度表示为一个并行化的循环(图1(c)),其中 $A_x(i)$ 表示第 i 个循环体中的 A_x 阶段。

软件流水算法已被证明是一种有效的用于开发循环指令级并行性的优化技术,但是绝大多数软件流水算法只针对最内层循环。多重循环的软件流水算法比较少,本文接下来的几部分将详细讨论各种不同的多重循环软件流水算法,比较它们的相似和不同之处。

2　嵌套循环的最优软件流水算法

本方法[3]假定处理机提供充足的资源,利用线性规划试图为多重循环的每层找到一个合适的II。本方法只适合于完

美嵌套循环，形式如图2所示：

```
for I1=l1 to u1
  ...
  for In=ln to un
    S1(T)
      ...
    Sr(T)
  endfor
  ...
endfor
```

图2　一个简单的完美嵌套循环

算法的基本思想就是利用线性规划，在满足相关限制的条件下，给出每个操作在不同循环索引 $I(i_1,i_2,\cdots,i_n)$ 中调度时刻的表达式。本算法要求各层循环的执行次数在编译时确定并且相等，即针对计数循环且循环体空间是矩形的。

算法的缺点是用线性规划求解找到最优的调度，计算复杂度高，而且所适用的多重循环限制条件较多，要求各层的循环次数在编译时已知并且相同，目标机器资源充足，因为复杂度及资源的限制，本算法无法在实际的编译器中实现。其优点就是不同的循环层启动间距是不一样的，而且启动间距可以是分数。

3　实时信号处理机中的多重循环软件流水

本方法[4]只适用于完美嵌套循环，形式如图2所示。对于那些非完美多重循环，可以通过易名(renaming)和循环分布(loop distribution)转为完美循环。

此方法的基本思想是先对内层循环做软件流水，形成装入、核心和排空阶段，然后展开外层循环体，此时不同外层循环体的内层循环软件流水都有相应的装入和排空阶段，如果一个外层循环体的内层循环软件流水的排空阶段和下一个外层循环体的内层软件流水的装入阶段没有相关，则利用楔型对接(dovetailing)将装入和排空合并，形成一个重复的执行模式，然后将转换后的循环重写。经过这样的处理，可以减小内层循环软件流水装入和排空部分的开销

本算法主要应用在 DSP 的编译器中，用于处理那些信号程序。

4　展开压缩

类似于前一种方法，外层展开，但是内层并不展开，内层通过插入合适的 shift/rotate 语句，使得不同外层循环体的内层循环上的操作通过一个轮转的方式来并行。

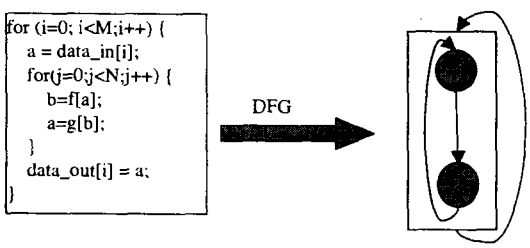


图3　简单嵌套循环及数据流图

从图3到图4的 DFG 图可以看出，经过展开压缩，不同外层循环体的内层循环上的操作可以并行执行。在此方法[5]中，外层循环展开因子(DS)的选择非常关键。DS 决定了外层可以并行的循环体数和内层软件流水的阶段数。从性能角度讲，如果外层没有相关或者相关可以通过变换去掉，那么外层循环的执行次数由原来的 M 减为 M/DS，内层循环的次数由 N 增大到 N×DS－(DS－1)，整个循环执行所需的次数差不多仍等于原来的 M×N。

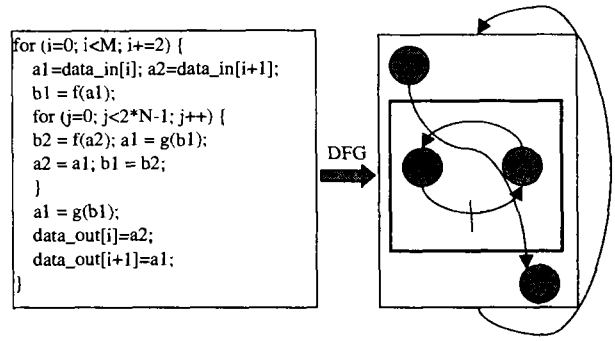


图4　应用展开压缩方法后的嵌套循环及数据流图

5　嵌套循环软件流水

在一个多重循环中，如果只对内层循环软件流水，那么每当执行一个外层循环时，相应的内层循环都要做装入和排空，特别是对外层循环次数很大，内层循环次数很小的多重循环而言，开销更大，看下面的例子：

```
REAL * 4 A(10,100),B(10,100),C(10,100)
DO J=1,100
   DO I=1,3
      A(I,J)=B(I,J)/C(I,J)
   ENDDO
ENDDO
```

图5　简单的二重循环

假定内层循环软件流水需要12个阶段，II 为5，假定外层需要5个时钟准备这个软件流水(数组地址重置，旋转条件寄存器重置，EC/LC 的重置等等)，执行这个循环需要的时钟为：

内层循环的装入和核心阶段 cycles 为：

100 * (3 * 5) = 1500

内层循环的排空阶段 cycles 为：

100 * (11 * 5) = 5500

外层循环准备内层软件流水的 cycles 为：

100 * (5) = 500

总共所需7500个 cycles

可以看出，程序的大部分执行时间都花在了内层循环软件流水的排空阶段，所以降低这部分的开销对提高整个多重循环的性能来说是非常重要的。

嵌套循环软件流水算法[6]提出在满足程序语义的情况下，将一个外层循环体所对应的内层循环软件流水的排空阶段与后面的一个或者多个外层循环体的内层循环软件流水的装入阶段(排空阶段)折叠，使得整个嵌套循环的内层循环软件流水装入和排空阶段只执行一次，这样就节省了内层循环装入、排空阶段的开销，提高了整个循环的性能。在这种方法中，内层在做软件流水的同时，外层也做流水，同时不同外层的内层循环相互重叠。上面的例子经过外层流水之后，整个程序执行所需的时钟周期减至2155cycles(100 * 3 * 5 + 11 * 5 + 100 * 6)。

此算法已经在基于 IA-64的优化编译器中实现，测试了14个 SPECfp2000和12个 SPECint2000的 benchmarks。测试表明，并不是所有的循环都有很大的性能提高，因为很多循环内层的循环次数很大，在这种情况下内层循环软件流水的时间

大部分在核心阶段，排空阶段的开销相对来说很小，此算法带来的性能提高不是很明显，但是应用此算法性能仍有一定的提高。

6 交替移位旋转算法(ISR)

本算法[7]以移位和旋转，调度与执行不完全一致为特征。通过识别并且并行化具有最大并行性的那一层循环，而对它内部的循环串行执行，避免了经典作法(无论并行性如何，总是并行化最内层)的缺陷。对于所识别的具有最大并行性的这个多重循环，通过"按单重循环调度，按多重循环执行"，并采用启发式的模调度，避免了经典的线性规划方法调度复杂性高的缺点，将调度代价降低到了线性复杂度。与传统方法不同，ISR 调度和执行不完全一致：调度满足资源限制，但并不直接满足相关限制，而是通过一些调度限制，与执行相联系：只要调度遵守了这些限制，执行一定遵守相关限制，从而具有正确语义；经典的模调度方法被 ISR 所包含，成为一个特例。

本算法支持循环重写。按单重循环调度之后，通过机械而简单的循环重写，将调度结果变换为具有正确语义的并行化多重循环，从而最终实现了"按单重循环调度，按多重循环执行"的目的。循环重写过程只涉及两种基本动作：移位和旋转。新循环指标的上下界和步长是规整的，且无需求解复杂的不等式方程组。图6描述了 ISR 的处理过程：

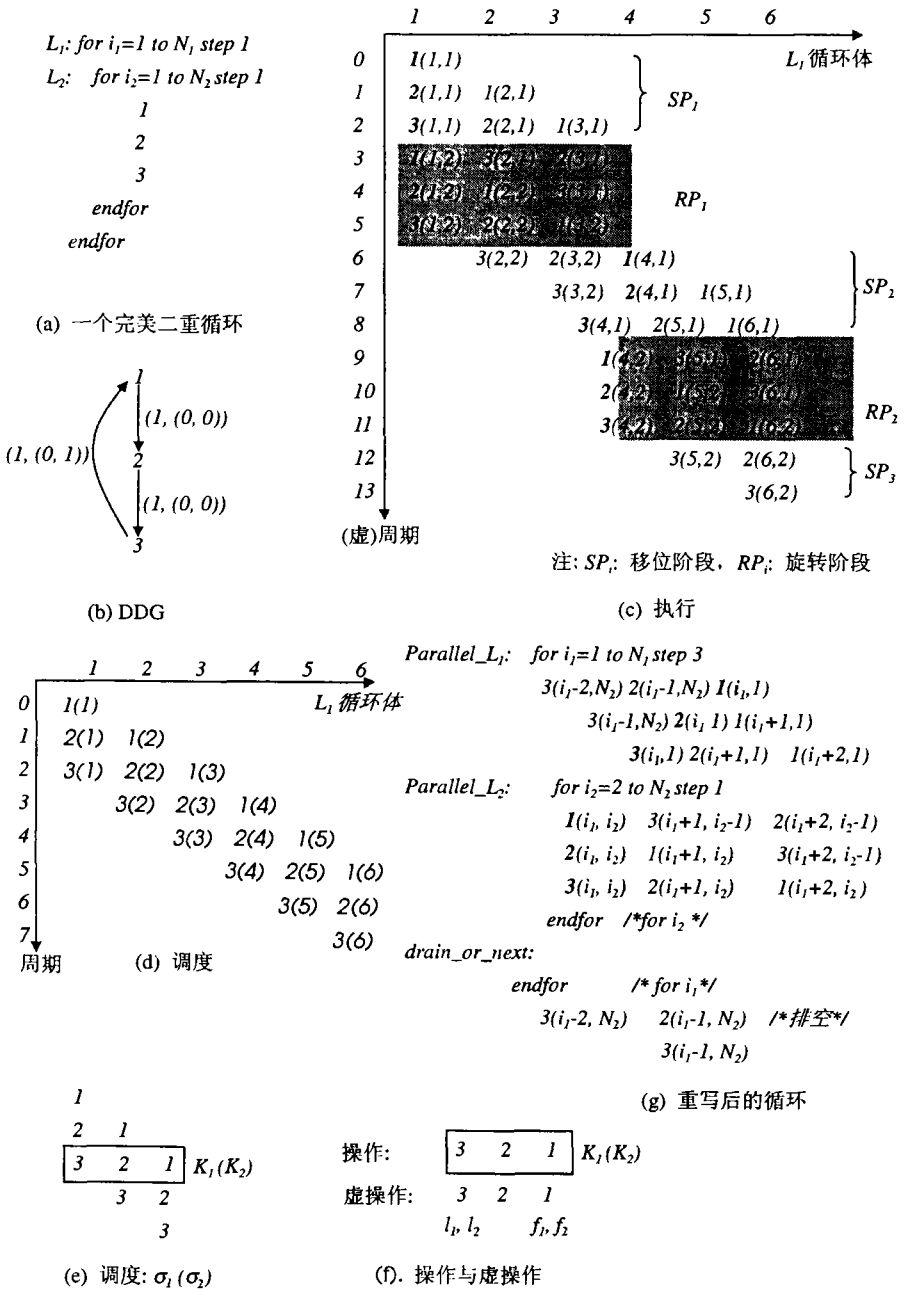


图6 ISR 用于完美循环的例子

ISR 算法提出了一种新的解决多重循环软件流水的方法，它从具有最大并行性的那一层开始软件流水，同时调度和执行不一致，这与以往的算法有很大的差别。缺点是调度比较复杂。

7 内外层交替执行的多重循环软件流水(ILSP)

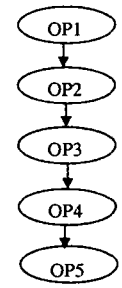
此方法[8]适用的多重循环比较广，既可以是计数循环，也可以是非计数循环，对完美和非完美循环都支持。在这个算

法中，内层循环的装入和排空阶段跟外层的部分重叠，节省了这部分的开销。对于同一个外层循环，它的内层循环实际上串行执行的，执行时刻的先后次序与串行意义下的执行顺序是完全一样的，因此在硬件资源充分的情况下，每个时钟周期能够启动一个内层循环体。用下面的例子来解释 ILSP 的基本原理。

如图7所示，如果把内层循环的3个操作看成是外层循环的，于是，这个程序就变成了一个单重循环，对这个单重循环进行软件流水时，可以发现，从周期3开始，每隔3个周期，就出现了一个 op2(3K)、op3(3K－1)、op4(3K－2)构成的执行模式，该执行模式的3个操作分别属于3个不同的外层循环体，每当该模式出现，就将程序暂停执行，重复执行该模式3(m－1)次，再配以相应的控制机制，就得到了2重循环的软件流水，表1是实际执行的过程。

```
for (i = 0; i <= n; i++) {
    op1;
    for (j = 0; j < m; j++) {
        op2;
        op3;
        op4;
    }
    op5;
}
```

(a) 2 重循环程序



(b) 相关性图

图7 简单的二重循环及相关性图

表1 多重循环软件流水的过程

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,-)	第一次内层排空，
x+2	op1(6,-)	op2(5,1)	op3(4,1)	op4(3,m)	op5(2,-)	同时外层装入、流水
x+3	op1(7,-)	op2(6,1)	op3(5,1)	op4(4,1)	op5(3,-)	第二次内层装入
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,-)	

本算法能够达到较高的 ILP，但是硬件控制机制非常复杂，难以在实际的机器中实现。

结论 第一种方法利用线性规划求出各层的启动间距，为每个操作找到一个合适的调度时刻，因为复杂度高，限制条件多而无法在实际的编译器中实现，但是其找到一个最优的调度，可以作为其它软件流水算法性能的参照，具有一定的理论参考价值。第二和第四种方法主要是通过外层的流水来避免内层循环软件流水过程中装入和排空的开销，其性能提高受循环的结构限制。第二和第三种方法都对外层展开，会带来代码的膨胀，其中第三种方法性能提高受展开因子以及循环自身的相关所限制。当内层循环装入和排空开销非常小时，第二和第四种方法所带来的性能提高很小。第五种方法通过找到最大并行性的循环进行软件流水，其需要借助硬件控制机制实现它的执行机制，纠正调度中的问题。第六种方法从没有体间相关的最外层开始软件流水，使得最内层循环可以串行执行，最大限度地开发了最内层循环的并行度，但是这种方法需要大量的硬件支持，实现复杂。从并行性开发方面，第五和第六种方法能实现的并行度很高。从实现方面来讲，第二、三、四种方法比较容易实现。第五种和第六种方法的思想对于我们提出新的软件流水算法都有很好的指导意义。在算法选择方面，如果只想减轻内层软件流水装入和排空的开销，可以考虑第二和第四种方法，另外第五种方法如果有一定的硬件支持，同时适当地做一些调整，实现起来还是相对第六种容易。

目前软件流水算法都没有考虑数据 Cache 的影响，其中很多经典的软件流水算法都假定数据 Cache 的命中率为100%，另外单重循环的软件流水算法都假定流水的部分也就是频繁执行的那部分指令始终在指令 Cache 中，这些假定在实际的处理机和编译器中是不可能完全达到的。在多重循环的软件流水中，ILSP 和 ISR 很好地利用了指令 Cache，但是都没有考虑数据 Cache 的影响。同时考虑数据和指令 Cache 的影响，是我们进一步研究多重循环软件流水算法的方向。

参考文献

1 Lam M S. Software Pipelining: An Effective Scheduling Technique for VLIW Machines. In: Proc. of the ACM Sigplan 1988 Conf. on Programming Language Design and Implem-entation. 1988

2 Rau B R. Iterative Modulo Scheduling. International Journal of Parallel Processing, 1996

3 Ramanujam J. Software pipelining of nested loops

4 Wang Jian, Su Bogong. Software pipelining of nested loops for real-time DSP Applications

5 Darin P, Randolph H, Saman A. Efficient pipelining of nested loops: unroll-and-jam

6 Kalyan M, Gautam D. Software pipelining of nested loops

7 容洪波. 多重循环的软件流水算法:[博士生论文]

8 Tang Zhizhong, Wang Lei, Zhang Chihong. Runinate Method-Software pipelining on nested loops, ICPA'95, Oct. 1995