软件流水中的一种数据调度算法*

罗 军 汤志忠 张赤红

(清华大学计算机科学技术系 北京 100084)

E-mail:luo,tzz,zch@est4.cs.tsinghua.edu.cn

摘要 文章第1节对软件流水下多重循环中数据元素的调度进行了分析,着重讨论了用地址计数器完成简单地址运算的意义、ILSP(interlaced inner and outer loop software pipelining)算法的基本思想及其在此基础上分析了软件流水下多重循环中数据元素的调度特点;第2节进一步探讨了为完成调度而寻找地址控制信息序列的一般方法;第3、4节则分别讨论了用求得的地址控制信息序列控制地址计数器对数据元素的访问和将地址控制信息序列化简为精简地址控制信息序列的步骤;最后两节分别是实验结果和结论.

关键词 数据调度,数据空间,地址控制信息序列,地址计数器.

中图法分类号 TP311

指令级并行ILP(instruction level parallel)[1~6]是当前国内外学者十分关注的研究课题.它主要包括ILP的体系结构、ILP算法及其优化编译技术等.在指令级并行体系结构及相应优化编译器的研究中,数据调度是一个十分重要的问题.一个好的数据调度算法可使整个指令级并行体系结构得以高效地完成数据访问,并使优化编译系统在对应用程序进行编译时,减少许多有关地址运算调度方面的繁琐工作,从而可把精力更集中于操作与操作间的并行优化上.

ILSP(interlaced inner and outer loop software pipelining)[7]内外层交替执行的多重循环流水算法是一个细粒度的软件流水算法.[8~12]它较成功地解决了对多重循环的软件流水问题.因而本文主要在该算法的思想下,对多重循环下数据元素的调度算法进行讨论.

1 软件流水中数据元素的调度分析

1.1 为什么选择地址计数器完成简单地址计算

在科学计算(信号处理、图象处理等)的应用程序中,大部分数据元素的寻址方式都是有规律的面向数组的地址计算.对于这类寻址方式,通常采用地址计数器完成地址运算,以使系统对数组元素调度不单独占用CPU时间.这样不仅能够有效地简化编译程序、节约系统功能部件,而且可大大增加程序操作的并行度,明显地使程序的运行速度加快.

1.2 ILSP算法的基本思想

ILSP是一种新的软件流水算法,其核心思想是:通过同时展开多个外层循环体,对每个外层循环体的内层循环串行执行,并从最内层就开始软件流水,使内外层循环交替执行.图1(a)是一个典型的二重循环的例子,其中$OP_i(i=1,2,\ldots,5)$分别代表5个不同的操作.图1(b)是图1(a)的数据相关图.在不存在体间相关条件下,表1给出了该循环在ILSP算法下的执行情况.

在分析ILSP算法多重循环的执行过程时,可以将ILSP的基本特征描述如下:ILSP算法在对各层循环进行软件流水时,将每一层循环内嵌套的各层循环视为仅执行一次.每当一个由某内层循环体构成的新模式出现时,当前外层循环暂停执行,而转去对该内层循环进行软件流水,这时外层循环放弃对全部或部分资源的支配权,供该内层使用.内层循环在完成其流水时,将执行排空操作.在返回父循环后,恢复其父循环对资源的支配权.

ILSP算法有两个显著的优点:(1) 对于大多数多重循环程序,ILSP算法能够消去所有内层循环的体间数据相关.(2) 将多重循环视为一个循环程序整体进行软件流水,因此使各层循环程序间形成了一条高效的流水线.

* 本文研究得到国家自然科学基金资助.作者罗军,1971年生,硕士,主要研究领域为指令级并行体系结构及其优化编译器.汤志忠,1946年生,教授,主要研究领域为计算机并行处理.张赤红,1964年生,副教授,主要研究领域为指令级并行体系结构及其优化编译技术.

本文通讯联系人:汤志忠,北京 100084,清华大学计算机科学技术系

本文1997-03-13收到原稿,1997-06-12收到修改稿

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

(a)　　(b)

图1　两层循环的典型图例

表1　ILSP算法的基本原理

周期	操作	说明
0	$op_1(1,-)$	外层装入
1	$op_1(2,-)$　$op_2(1,1)$	内、外层同时装入
2	$op_1(3,-)$　$op_2(2,1)$　$op_3(1,1)$	
3	$op_1(4,-)$　$op_2(3,1)$　$op_3(2,1)$　$op_4(1,1)$	内层充满,切换
4	$op_2(1,2)$　$op_3(3,1)$　$op_4(2,1)$	内层流水,外层暂停
5	$op_2(2,2)$　$op_3(1,2)$　$op_4(3,1)$	
⋮	⋯	
x	$op_2(3,m)$　$op_3(2,m)$　$op_4(1,m)$	
$x+1$	$op_1(5,-)$　$op_2(4,1)$　$op_3(3,m)$　$op_4(2,m)$　$op_5(1,-)$	第1次内层排空,同时外层装入、流水,同时进行第2次内层装入
$x-2$	$op_1(6,-)$　$op_2(5,1)$　$op_3(4,1)$　$op_4(3,m)$　$op_5(2,-)$	
$x+3$	$op_1(7,-)$　$op_2(6,1)$　$op_3(5,1)$　$op_4(4,1)$　$op_5(3,-)$	
$x+4$	$op_2(4,2)$　$op_3(6,1)$　$op_4(5,1)$	再次内层流水,外层暂停
⋮	⋯	
y	$op_2(n,m)$　$op_3(n-1,m)$　$op_4(n-2,m)$	
$y+1$	$op_3(n,m)$　$op_4(n-1,m)$　$op_5(n-2,-)$	最后,整个循环排空
$y+2$	$op_4(n,m)$　$op_5(n-1,-)$	
$y+3$	$op_5(n,-)$	

注:$op_k(i,j)$表示操作k处于第i个外层循环体与第j个内层循环体之中.

1.3　软件流水中数据元素调度特点的分析

首先,设数据元素按照其在循环程序执行时,第1次被读取的时刻的先后次序,采用顺序结构存储于内存.以矩阵乘法为例分析多重循环下数据元素调度.实现矩阵乘法的多重循环程序如图2所示.

```
for(i=0;i<n;i++)
  for(j=0;j<m;j++)
    for(k=0;k<t;k++)
      C[i][j]=C[i][j]+A[i][k]*B[k][j];
```

图2　矩阵乘法应用实例

其中$C[i][j]$的初值设为零.按照矩阵A,B,C在应用实例执行过程中的读取操作,将矩阵A,C的元素按行主序分配内存,矩阵B的元素按列主序分配内存,则矩阵A,B,C的元素在内存中的位置关系分别为:$a_{0,0}$ $a_{0,1}$…$a_{0,17}$ $a_{1,0}$…$a_{1,17}$,…,$a_{17,0}$…$a_{17,17}$,$b_{0,0}$ $b_{1,0}$…$b_{17,0}$ $b_{0,1}$…$b_{17,1}$,…,$b_{0,17}$…$b_{17,17}$,$c_{0,0}$ $c_{0,1}$…$c_{0,17}$ $c_{1,0}$…$c_{1,17}$,…,$c_{17,0}$…$c_{17,17}$.

在上述内存分配的基础上,矩阵A,B,C的调度是不同的.矩阵A在每一行元素被连续访问18次后,下一行元素方能被访问;而矩阵B在每一列元素均被访问一遍后,地址计数器的值要重新设置为元素$b_{0,0}$在内存中的位置,并重新开始计数;矩阵C则是在每个元素连续访问18次后,才开始访问下一个元素.由此可见,对于同一个多重循环中的不同数组,其地址访问的控制信息是不同的.

在深入分析之前,本文对若干符号约定如下:符号$n_Array(X)$表示一个$n(n\geqslant 1)$维数组,符号$SUB(n_Array(X))$表示数组X在各维上的下标变量的集合.类似地,符号$k_Loop(A)$表示一个$k(k\geqslant 1)$重循环A,并将k重循环的

层号规定为从最外层循环到最内层循环，各层循环的层号依次为 1,2,...,k. 其中 $k\_Loop(A)$ 的第 $i(1\leqslant i\leqslant k)$ 层循环记作 $k\_Loop^i(A)$. 符号 $var(k\_Loop^i(A))$, $len(k\_Loop^i(A))$, $i(i=1,2,\ldots,k)$ 层循环记作 $rec(k\_Loop^i(A))$, $init(k\_Loop^i(A))$ 分别表示 $k\_Loop^i(A)$ 的循环控制变量、循环次数、循环控制变量的修正值和循环变量的初值. $k\_Loop(A)$ 的各层循环控制变量的集合用 $VAR(k\_Loop(A))$ 表示. $op(k\_Loop(A), n\_Array(X))$ 表示 n 维数组 X 在 k 重循环 A 中对存储器的操作. 本文进一步给出下列定义.

定义 1. 若在 $k\_Loop^j(A)(1\leqslant j\leqslant k)$ 中有 $op(k\_Loop(A), n\_Array(X))$，则称 $k\_Loop(A)$ 在第 j 层上对 $n\_Array(X)$ 包含，记作 $con(k\_Loop(A), j, n\_Array(X))$. 当 $j=k$ 时，称 $k\_Loop(A)$ 全包含 $n\_Array(X)$，记作 $fullcon(k\_Loop(A), n\_Array(X))$.

定义 2. 若有 $k\_Loop(A)$, $n\_Array(X)$，且有 $con(k\_Loop(A), j, n\_Array(X))$，其中 $1\leqslant j\leqslant k$，若对某个 $i_h(i_h\in SUB(n\_Array(X), 1\leqslant h<n))$，存在 $f(1\leqslant f\leqslant j)$，使等式 $i_h=var(k\_Loop^f(A))$ 成立，则称 $n\_Array(X)$ 与 $k\_Loop^f(A)$ 相关，且称 $var(k\_Loop^f(A))$ 为关于 $n\_Array(X)$ 的相关控制变量，$k\_Loop^f(A)$ 为关于 $n\_Array(X)$ 的相关循环层，相关循环层的层号 f 记作 $layer(k\_Loop(A), i_h)$.

尽管不同的数组在多重循环执行时所需要的调度方法是不同的，但是它们有一个共同的特点，即任何一个数据元素在多重循环执行过程中的任一时刻都必然处在某个或某些数据空间内. 所谓数据空间，本文是如下定义的.

定义 3. 设有一个 k 重循环和一个 n 维数组，则数据空间是满足下列条件之一的数据元素集合：

(1) 全体数组元素；或者是

(2) 当多重循环中的外层(一层或几层)循环的控制变量取某一定值时所访问到的数组元素.

假设现有一个 k 重循环 A 和一个 n 维数组 X，并且将循环与数组示意地表示为图 3(a)的形式(其中 $I_j\in SUB(n\_Array(X), T_k\notin SUB(n\_Array(X)))$. 数组 X 在 $k\_Loop(A)$ 串行执行时的地址变化示意地表示为图 3(b)的形式. 则在图 3(b)中，本文列出了两个数据空间，我们认为这两个数据空间元素的存取过程是相同的. 由图中可以看出数据空间 1 与数据空间 2 是不同的，主要表现在程序访问数据空间有先有后以及数据空间之间的元素不同. 前者反映了时间上的差异，后者表明了空间上的差异. 从图 3(b)可明显地看出造成这种差异的原因在于变化 1→2 的发生. 即变化 1→2 是从数据空间 1 切换到数据空间 2 的充分必要条件. 对照图 3(a)可知，变化 1→2 的发生等价于第 l 层循环的控制变量 I_j 增加 step1. 进一步分析 I_j 增加 step1 的原因在于，第 $l+1$ 层循环执行完毕并返回第 l 层循环. 即当 t_k 小于 Upper2 与 step2 的差值时，t_k 每增加 step2，第 $l+1$ 层循环便执行 1 次，此时，程序所访问的数据在数据空间 1 中. 当 t_k 大于或等于 Upper2 时，则程序控制必然从第 $l+1$ 层循环返回到第 l 层循环，则 I_j 增值 step1，且使程序开始访问数据空间 2 中的数据. 由此，可以得出两点结论：(1) 数据空间 1 到数据空间 2 的切换是由于第 $l+1$ 层及其嵌套的内层循环已执行完毕；(2) 数据空间 1 与数据空间 2 的存取过程之所以相同，在于它们同属第 l 层循环.

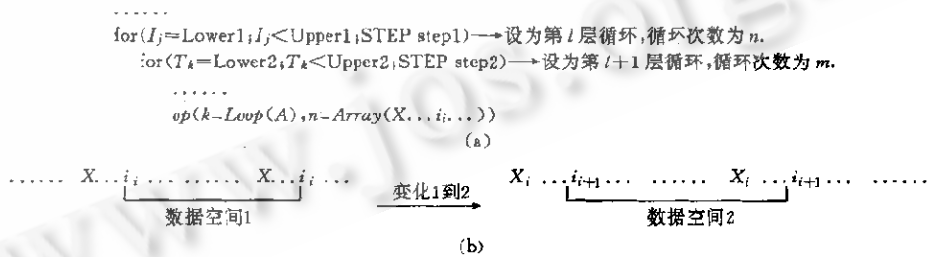

图3　数据元素在软件流水中的调度

从以上分析可以看出，对某一数组而言，包含它的各层循环在程序执行过程中所起的作用是有差异的：一类由数组的相关循环层组成，它们唯一地决定了当前数据元素的存取在哪一个数据空间运行；另一类由多重循环中其余的子循环组成，它们的功能是控制数据元素的存取何时从一个数据空间切换到另一个数据空间. 研究数组在多重循环下的调度，在一定意义上就是研究这两类子循环的不同作用及其之间的关系. 然而对于一个未知应用程序，我们无法预测包含一个数组的多重循环的结构. 即使我们知道包含某个数组的多重循环由 n 个与数组相关的循环和 m 个非相关循环组成. 那么，满足此条件的多重循环的结构也有 $\frac{(n+m)!}{n!}$ 种可能根据前面的假定，我们认为在不同的多重循环结构中数组的相关循环层之间的相对位置是固定的). 当 $n=2, m=3$ 时，则有 60 种可能. 显然，对每一种可能的多重循环

类型分别寻找出控制方法是不现实的.但是在分析数组元素在多重循环中存取特点的过程中,我们可以在语义等价的前提下,对多重循环进行变换,使得不同结构的多重循环体现出一种控制上的共性.

设现有一个 k 重循环 $k\_Loop(A)$,等价变换的目标是将相异循环程序归并为同一种模式,并记为 $k'\_Loop(A)$.在该模式中,包围数组的最内层与最外层循环均为数组的非相关循环层,在其内部数组的相关循环层与非相关循环层交错排列.具体地说,$k'_Loop(A)$具有如下形式:

```
for(t0=. . . . . .)
  for(I0=. . . . . .)
    for(t1=. . . . . .)
      for(I1=. . . . . .)
        for(t2=. . . . . .)
                ⋮
          for(In-1=. . . . . .)
            for(tn=. . . . . .)
              op(k_Loop(A),n_Array(X));
```

其中 $I_j \in SUB(n\_Array(X))$, $j=0,\ldots,n-1$; $t_h \notin SUB(n\_Array(X))$, $h=0,\ldots,n$.

将 $k_Loop(A)$归纳为一般模式 $k'_Loop(A)$的步骤如下:

Step1. 若 $var(k_Loop^1(A)) \in SUB(n_Array(X))$转 Step2;否则转 Step3;

Step2. 若 $layer(k\_Loop(A),I_{j+1}) - layer(k\_Loop(A),I_j) > 1$,

则 $len(layer(k'\_Loop(A),t_j) = \prod_{f=layer(k\_Loop(A),I_j)+1}^{layer(k\_Loop(A),I_{j+1})-1} len(k\_Loop^f(A))$,

否则,$len(layer(k'\_Loop(A),t_j)=1$,其中 $j=1,\ldots,n-1$;

$len(layer(k'\_Loop(A),t_0))=1$;

转 Step4;

Step3. 若 $layer(k\_Loop(A),I_{j+1}) - layer(k\_Loop(A),I_j) > 1$,

则 $len(layer(k'\_Loop(A'),t_j)) = \prod_{f=layer(k\_Loop(A),I_j)+1}^{layer(k\_Loop(A),I_{j+1})-1} len(k\_Loop^f(A))$,

否则,$len(layer(k'\_Loop(A),t_j))=1$,其中 $j=0,1,\ldots,n-1$;

转 Step4;

Step4. 若 $k - layer(k\_Loop(A),I_{n-1}) \geq 1$,

则 $len(k'\_Loop^k(A)) = \prod_{f=layer(k\_Loop(A),I_{n-1})+1}^{k} len(k\_Loop^f(A))$

否则,$len(k'\_Loop^k(A))=1$;

转 Step5;

Step5. 结束.

对于一个 n 维数组而言,对包围它的多重循环进行归纳,所求得的一般模式必然由 $2n+1$ 个子循环构成.由于控制数组的不同的数据存取空间共需要 n 个操作数,如果把程序对数组进行访问的前后分别看作处于不同的数据空间,则控制数组内部不同数据空间及数组内部数据空间与外部数据空间的转换需要 $n+1$ 个数.此外,我们还需要一个控制信息,该信息用来记录当某个数据元素第 1 次被访问后到下一个元素被访问之前,该数据元素被连续访问的次数.因此,本文有如下结论与定义.

结论. 对于一个 k 重循环 $k_Loop(A)$和一个 n 维数组 $n\_Array(X)$,且 $fullcon(k_Loop(A),n_Array(X))$成立,则最多需要 $2n+2$ 个有序数即可完成对数据元素的调度.

定义 4. 对于一个 k 重循环 $k_Loop(A)$和一个 n 维数组 $n\_Array(X)$,序列 $C_0,C_1,\ldots,C_{2n+1}$表示 $n_Array(X)$的元素在 $k_Loop(A)$中的地址变化控制信息,其中 C_0 称为元素活动位;当 $i=2t+1(1\leqslant t\leqslant n)$时,$C_i$ 称为地址变化位;当 $i=2t(1\leqslant t\leqslant n)$时,$C_i$ 称为模式重复位.该序列简记为 $CONTROL(k\_Loop(A),n\_Array(X))$.

2　地址控制信息序列的获取

从上节的讨论得知,对一个多重循环中的 n 维数组可以用 $2n+2$ 个有序数进行控制.本节针对上面所得到的 $k'_Loop(A)$讨论该地址控制信息序列序列 $C_0,C_1,\ldots,C_{2n+1}$的确定.

由于 ILSP 算法中同时展开了多个外层循环体,因此在下面的讨论中,用符号 II 表示一次同时展开的外层体.确定序列 $C_0,C_1,\ldots,C_{2n+1}$的步骤如下:

Step1. 若 $var(k_Loop^1(A)) \in SUB(n_Array(X))$转 Step3;否则,转 Step2;

Step2. $C_0 \leftarrow \Pi * len(k'\_loop^k(A))$;

$C_r \leftarrow len(k'\_loop^{k-f}(A)), f=1,\ldots,n-1,2n+1$;

$C_{2n} \leftarrow len(k'\_loop^1(A)) * \left\lfloor \frac{len(layer(k'\_loop(A), I_0))}{\Pi} \right\rfloor$;

转 Step4;

Step3. $C_0 \leftarrow 1$;

$C_1 \leftarrow \Pi$;

$C_r \leftarrow len(k'\_loop^{k-f+2}(A)), f=2,\ldots,2n$;

$C_{2n+1} \leftarrow \left\lfloor \frac{len(layer(k'\_loop(A), I_0))}{\Pi} \right\rfloor$;

Step4. 结束.

3 地址控制信息序列的应用

本文用地址控制信息序列完成计数器对数组元素寻址的有效控制.其控制过程如下:将 $2n+2$ 个寄存器(记为 $R_0R_1\ldots R_{2n+1}$)的初值分别设置为地址控制信息序列中的 $2n+2$ 个数.在计数器计数过程中,程序每访问一次数组,寄存器 R_0 值减 1,若 R_0 值为零,则计数器值自动增 1,同时寄存器 R_1 值减 1,R_0 恢复为初值;当 $R_d(d=1,\ldots,2n+1)$减 1 时,若 R_d 值不为零且 d 为不等于 $2n+1$ 的奇数,则计数器值减去 Πc_g(其中 g 为大于等于 1 且小于等于 $d-1$ 的奇数),同时寄存器 R_v(其中 v 大于等于 1 且小于等于 $d-1$)恢复为初值,若 R_d 值为零,则 R_d 恢复为初值,且使 R_{d+1}减 1;若 R_d 减 1 后为零,且 d 等于 $2n+1$,则此时对数据元素的调度结束.

4 地址控制信息序列的化简

通常应用程序中的多重循环结构要比第 2 节中得到的一般模式 $k'_Loop(A)$简单得多.例如,对于矩阵乘法在应用实例中只有 3 个循环,但是归纳为一般模式后,就成为一个五重循环.这是由于在归纳过程中,加入了运行次数为 1 的循环——相当于将原来的 1 个数据空间划分成两个,从而增加了控制上的复杂度.因此,对 $CONTROL(k_Loop(A), n_Array(X))$进行化简,从而将上面所增加的控制信息从 $CONTROL(k_Loop(A), n_Array(X))$中删除.

化简的思路是将上述划分成的数据空间重新合并,并合并相应的地址控制信息序列 $CONTROL(k_Loop(A), n_Array(X))$中相应的控制数.化简的步骤如下:

Step1. $i \leftarrow 2n+1, count \leftarrow 2n+2$,若 C_i 等于 1,则 $i \leftarrow i-2, count \leftarrow i-1$;若 $2n+1$ 大于 3,则转 Step2;否则,转 Step3;

Step2. 若 $count$ 小于等于 3,则转 Step3;否则,若 C_i 等于 1,则 $C_{i-1} \leftarrow C_{i-1} * C_{i-1}, C_j \leftarrow C_{j+2}$,其中 $j=i,\ldots,count$,且 $i \leftarrow i-2, count \leftarrow count-2$,转 Step2;

Step3. 若 C_i 等于 1,则 $C_j \leftarrow C_{j+1}, j=i,\ldots,count; count \leftarrow count-1$;转 Step4;

Step4. 结束.

最后所得到的序列 $C_0C_1\ldots C_{count-1}$即是所求的精简地址控制序列信息 $RCONTROL(k\_Loop(A), n\_Array(X))$,序列长度为 $count$ 的当前值.

定义 5. 称从 $CONTROL(k_Loop(A), n_Array(X))$化简后所得到的序列为精简地址控制信息序列,表示为 $RC_0RC_1\ldots RC_{count}$,并记作 $RCONTROL(k\_Loop(A), n\_Array(X))$.

5 实验结果

以图 2 中的矩阵乘法为例,表 2 和表 3 分别给出了矩阵 A,B,C 的元素在三重循环中的地址控制信息序列和精简地址控制信息序列.

表 2 地址控制信息序列

	C_0	C_1	C_2	C_3	C_4	C_5
A_{18*18}	1	9	1	18	18	2
B_{18*18}	9	18	1	18	2	1
C_{18*18}	1	9	18	18	1	2

表 3 精简地址控制信息序列

	RC_0	RC_1	RC_2	RC_3
A_{18*18}	1	162	18	2
A_{18*18}	9	324	2	−1
C_{18*18}	1	9	18	36

本文对 6 个较常见的应用程序采用本文提出的算法进行了数据调度,并在实际的指令级体系结构的仿真系统上进行了测试.测试的结果如表 4 所示.

表4　实验比较结果

程序名称	循环层数	采用算法1的实验结果			未采用算法1的实验结果			加速比 (T_2/T_1)
		最内层循环环操作个数	内层循环启动间距	执行时间 T_1(拍)	最内层循环环操作个数	内层循环启动间距	执行时间 T_2(拍)	
矩阵加法	2	4	1	150	10	1	150	1
矩阵乘法	3	6	1	5 841	12	2	11 682	2
卷积乘	3	6	1	333	15	2	666	2
求最大最小值	2	7	1	90	13	2	180	2
快速排序	3	5	1	301	13	2	602	2
求水仙花	3	7	1	1 031	15	2	2 062	2

从表中可以看到,通过采用数据调度算法可以获得平均加速比1.83.通常,一个多重循环的执行时间主要取决于最内层循环的执行时间.由于最内层循环的执行时间受硬件资源的限制,因此,如果能够减少最内层循环对硬件的需求,就有可能缩小循环的启动间距,并因此缩短最内层循环的执行时间.由于算法1的目标是将地址计算交给硬件完成,我们可以省去大量的与地址运算有关的操作.在硬件资源受限的情况下,运算量的减少不仅有助于缩小软件流水的起动间距,而且可以将节约出来的功能部件执行其它操作,从而增加程序执行的并行度和加快程序的执行时间.

我们另外还对38个带有多重循环结构的科学计算和图象处理程序分别用本文提出的数据调度算法进行了测试,并分别得出了相应的地址控制信息序列和精简地址控制信息序列.统计结果显示,本文提出的数据调度算法全都能够应用于这些应用程序,这表明本文提出的算法具有较大的适用性.

6　结　论

综上所述,本文在对软件流水下数据元素在多重循环中存取特点的深入分析的基础上,给出了获取数据元素控制信息的一套算法.虽然不同的软件流水算法对数据元素的内存分配有较为独特的要求,而在此基础上对数据元素的寻址控制相互差别不大.因此,本文的数据调度算法虽然是根据ILSP算法的特点提出的,但其思想却具有较大的通用性.目前,根据本文提出的数据调度算法已经应用到实际的指令级并行体系结构及相应的优化编译器中,该体系结构模拟系统运行结果已经证明,本文提出的算法在实际中是适用的和有效的.

参考文献

1　Rau B R, Fisher J A. Instruction-level parallel processing: history, overview, and perspective. The Journal of Supercomputing, 1993,1:9~50

2　Colwell R P, Nix P R, Donnell J J *et al*. A VLIW architecture for a trace scheduling compiler. IEEE Transactions on Computers, Aug. 1988,C-37(8):967~979

3　Fisher J A. Very long instruction word architecture and the ELI-512. In: Proceedings of the 10th Annual International Symposium on Computer Architecture. Stockholm, June 1983. 140~150

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

5　Jouppi N P, Wall D. Available instruction level parallelism for superscalar and superpipelined machine. In: Proceedings of the 3rd International Conference on Architectural Support for Programming Languages and Operating Systems. Boston, Apr. 1989. 272~282

6　Acosta R D, Kjelstrup J, Torng H C. An instruction issuing approach to enhancing performance in multiple function unit processors. IEEE Transactions on Computers, Sep. 1986,C-35(9):815~828

7　Wang Lei, Tang Zhi-zhong, Zhang Chi-hong. Ruminate method——a novel framework for software pipelining on nested loops. In: Proceedings of the MPCS'96 Conference on Massively Parallel Computing Systems. May 1996. 475~482

8　Wolf M, Lam M. A loop transformation theory and algorithm to maximize parallelism. IEEE Transactions on Parallel and Distributed Systems, Oct. 1991,2(4):452~471

9　Rau B R. Iterative modulo scheduling: an Algorithm for software pipelining Loops. In: MICRO-27. Nov. 30~Dec. 2, 1994. 63~74

10　Lam M. Software pipelining: an effective scheduling technique for VLIW machines. In: Proceedings of the SIGPLAN'88 Conference on Programming Language Design and Implementation. Jun. 1988. 318~328

11 Su B, Wang J, Tang Z, Zhao W. A software pipelining based VLIW architecture and optimization compiler. In: MICRO-23. 1990. 17~27

12 Charlesworth A E. An approach to scientific array processing: the architectural design of the AP-120B/FPS-164 family. Computer, 1981,14(9):18~27

An Algorithm on Data Schedule in Software Pipeline

LUO Jun TANG Zhi-zhong ZHANG Chi-hong

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

Abstract In the first part of this paper, an analysis of the characteristics of data schedule under nested loops is presented. Three aspects which include the significance of the fulfillment of data accessing by the address counter, the principles of ILSP (interlaced inner and outer loop software pipelining) algorithm and under the idea of ILSP algorithm are presented. And a discussion of the data scheduling characteristics is given. In part 2, the key steps of how to derive the data addressing control sequence from the applications is listed. In part 3 and part 4, the authors respectively discuss how to apply the sequence of data accessing control to control the address counter and how to simplify the data accessing control sequence to the reduced data accessing control sequence. The part 5 is experiment results and part 6 is conclusion.

Key words Data schedule, data space, data accessing control sequence, address counter.