分解式软件流水 DESP
——一种开发循环程序指令级并行性的新方法*

汤志忠 张赤红
（清华大学计算机科学与技术系，北京 100084）

王 剑
（法国国家研究院 INRIA）

摘要 本文在软件流水方面提出一种新观点，把软件流水看作是一种指令级变形，是把一维指令向量变换成二维指令矩阵. 这样，软件流水问题可以很自然地分解为两个子问题：一个是确定每个操作在指令矩阵中的行号，另一个是确定其在指令矩阵中的列号. 基于这种观点，我们开发出一种新的循环调度方法，叫做分解式软件流水——DESP.

关键词 循环调度，指令级并行性，软件流水，循环体间相关，问题分解.

在一般程序中，循环占了绝大部分的机器执行时间. 因此，在设计高性能计算机（如超标量、超流水线和超长指令字）的优化编译器时，对指令级并行性的开发是一个主要的挑战. 软件流水是一种有效开发循环程序指令级并行性的优化编译技术，其基本思想是：在资源限制、数据相关及周期性条件的保证下，一个循环可以变形，使循环的一次执行可以在前一次执行结束前启动，从而使多个循环体并行执行.

在过去 10 年中，提出了大量的软件流水方法[1-7]，这些方法在满足上述 3 个限制条件的同时，采用各种启发策略来重构循环体. 然而，由于资源限制和循环体间相关使软件流水问题非常复杂，因而现有的软件流水方法并不能在低的计算复杂度下得到满意的时间效益和空间效益.

最近，又有两种软件流水方法被提出[8,9]. 在文献[8]中，首先对给定的循环程序在资源无限的条件下进行预处理，并从预处理中得到一些信息，然后根据这些信息删除循环数据相关图（LDDG）中的一些边，使 LDDG 无环路，最后运用列表调度技术在资源限制的条件下构造新循环体. 另一种新方法在文献[9]中提出，它只能解决 LDDG 无环的循环调度问题. 该方法分为两步，首先，在资源限制条件下通过“忘记”一些相关边来构造新循环体，然后确定每个操作的循环体号，称做循环体索引，使所有相关边得到满足. 受以上两种方法的启发，我们提出了一种新的软件流水方法，叫做分解式软件流水——DESP，其基本思想是：首先把软件流水看作是一种指令级变形，是从一维指令向量变换成二维指令矩阵，然后把软件流水

* 本文 1994－02－03 收到，1994－04－11 定稿

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，并行算法及优化编译技术. 张赤红，1964 年生，讲师，主要研究领域为细粒度并行体系结构及优化编译器. 王剑，1966 年生，1993 年博士后毕业于法国国家研究院，主要研究领域为指令级并行算法及优化编译技术.

本文通讯联系人：汤志忠，北京 100084，清华大学计算机科学与技术系

分解成两个子问题，一个是确定每个操作在指令矩阵中的行号，另一个是确定其列号. 实际上，前一个问题是与循环无关的代码调度问题，可以用已经相当成熟的列表调度法有效解决，后一个问题与资源限制无关，可以很容易地用图论的经典算法来解决.

下面，首先提出分解式软件流水思想，然后在这种思想的基础上提出两种新的软件流水算法，并把我们的算法与现有的软件流水算法进行比较，最后是结论.

1　分解式软件流水思想

1.1　对软件流水的一种新观点

软件流水实际上是在资源限制、数据相关及周期性条件限制下的一种指令级变形. 我们用一个带双重权值的循环数据相关图(LDDG)来描述一个循环，LDDG$=(O,E,\lambda,\delta)$，其中O是循环体中操作的集合，E是相关边的集合，λ与δ是相关边的两个非负权值. 例如，$e=(OPi,OPj)\in E$，$(\lambda(e),\delta(e))$表示OPj只能在前第$\lambda(e)$个体的OPi启动$\delta(e)$个周期后才能启动. 这样，软件流水问题可以描述如下：

构造一个循环调度σ，它是一个从$O\times N$到N(非负整数集)的映象函数，如$\sigma(OP,i)$表示第i个循环体的OP操作的执行启动周期. 如果下述条件满足：

(1)资源限制：在每个周期中，同一个资源不能被占用超过一次.

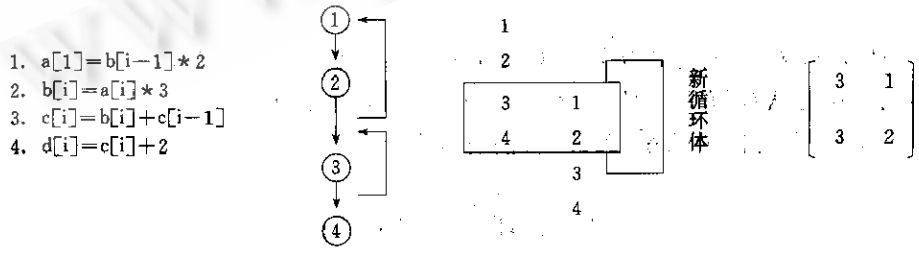
(2)数据相关：$\forall\ e=(OPi,OPj)\in E,\forall\ k\in N,\sigma(OPi,k)+\delta(e)\leqslant\sigma(OPj,k+\lambda(e))$.

(3)周期性限制：σ必须可以用循环的形式表达，即$\exists\ \alpha,\beta\in N,\forall\ OP\in O,\forall\ i\in N$且$i>0,\sigma(OP,i)=\sigma(OP,(i\bmod\beta))+\alpha*(\lceil i/\beta\rceil-1)$.

我们说σ是给定循环的一个有效调度，α/β被称作σ的平均启动周期，软件流水的目标就是寻找一个具有最小启动周期的有效调度.

若$\beta>1$，我们称σ是带循环展开的软件流水. 有时候循环展开能改进软件流水的时间效益，但同时使空间效益变坏. 实际上，我们可以先展开循环，然后按不带循环展开的方法进行软件流水[9]. 所以在本文中，我们着重讨论不带循环展开的情况.

下面，让我们来看一个软件流水的例子，如图1所示，假定机器有一个加法器和一个乘法器，并能并行工作. 其中，图1(a)是一个循环体的串行程序，图1(b)是循环数据相关图LDDG，图1(c)给出软件流水的结果，其中$\sigma(OPi,j)=i+2*(j-1)$.



(a) 一个循环体　(b) LDDG　(C) 软件流水结果　(d) 指令距阵

图1　一个软件流水的例子

从这个例子中我们看到，新循环体是通过对旧循环体作指令级循环变形得到的. 通常

在旧循环体中一条指令就是一个操作，而在新循环体中，每一周期内可能会执行多个旧循环体中的操作. 既然如此，我们可以认为软件流水是一种指令级变换，它将一维指令向量(指旧循环体，每个元素是一个操作)变换成二维指令矩阵(指新循环体，每个元素是一个操作或一组操作). 在指令矩阵中，行代表周期号，列代表循环体号. 上例中，我们把一维指令向量[1,2,3,4]变形为如图1(d)的二维指令矩阵.

定义 1. 行号与列号：设 σ 是一个有效调度，h 是经软件流水后新循环体的长度，在新循环体(指令矩阵)中，对每一个操作 OP，我们定义行号 rn，列号 cn，满足：

$$\sigma(OP,1)=rn(OP)+h*(cn(OP)-1),\quad \sigma(OP,i)=\sigma(OP,1)+h*(i-1).$$

这两个等式给出了循环周期、行号与列号之间的关系. 在上面的二维指令矩阵中，$rn(OP1)=1, rn(OP2)=2, rn(OP3)=1, rn(OP4)=2, cn(OP1)=1, cn=(OP2)=1, cn(OP3)=2, cn(OP4)=2$. 行号与列号是两个很重要的概念，如果两个操作具有相同的行号，那么它们将在同一时刻被启动，因此不能占用同一个资源. 如果两个操作具有相同的列号，那么它们属于同一个循环体，因此必须满足数据相关关系.

从以上讨论中，资源限制，数据相关和周期性限制可以用行号与列号描述如下：

(1)资源限制：若两个操作具有相同行号，则它们不能占用同一个资源.

(2)数据相关：$\forall e=(OPi,OPj)\in E, rn(OPj)-rn(OPi)+h*(\lambda(e)+cn(OPj)-cn(OPi))\geqslant\delta(e)$.

(3)周期性限制：实际上，这种变换本身已经保证了周期性限制.

因此，一个有效调度 σ 可以用 (rn,cn,h) 来表示.

1.2 确定行号和确定列号之间的关系

在这一节中，我们进一步分析确定每个操作的行号与确定其列号之间的关系，并提出满足这些关系的两个充要条件.

首先我们分析行号确定后对确定列号产生的影响. 设 $e=(OPi, OPj)$ 是 LDDG 的一条相关边，h 是新循环体的长度. 为满足数据相关，OPi 与 OPj 的行号与列号必须满足：

$$rn(OPj)-rn(OPi)+h*(\lambda(e)+cn(OPj)-cn(OPi))\geqslant\delta(e),$$

显然，$rn(OPj)-rn(OPi)<h$，我们分两种情况讨论：

(1)若 $rn(OPj)-rn(OPi)\geqslant\delta(e)$，则 $cn(OPj)-cn(OPi)\geqslant-\lambda(e)$.

(2)若 $rn(OPj)-rn(OPi)<\delta(e)$，则 $cn(OPj)-cn(OPi)\geqslant-\lambda(e)+\lceil(\delta(e)+rn(OPi)-rn(OPj))/h\rceil$.

以上两种情况表明，行号的确定将对确定列号产生新的限制. 更准确地说，对两个操作 OPi 与 OPj，若有从 OPi 到 OPj 的相关，那么 $cn(OPj)$ 与 $cn(OPi)$ 之间的关系将依赖于 $rn(OPj)$ 与 $rn(OPi)$ 之间的关系. 为此，我们对 LDDG 的每条边引入一个新概念，称作循环距离 τ.

定义 2. 设 (O,E,λ,δ) 是一个给定循环的 LDDG，我们定义 $\text{LDDG}^{\tau}=(O,E,\tau)$，其中 τ 是每边的权值，以代替 (λ,δ). 在确定了每个操作的行号以后，我们可以定义每条边的 $e=(OPi,OPj)$ 的 τ 值：

(1)若 $rn(OPj)-rn(OPi)\geqslant\delta(e)$，则 $\tau(e)=-\lambda(e)$；

(2)若 $rn(OPj)-rn(OPi)<\delta(e)$，则 $\tau(e)=-\lambda(e)+\lceil(\delta(e)+rn(OPi)-rn(OPj))/h\rceil$.

定义 3. σ^r 是 $LDDG^r$ 的一个有效调度，每条边满足：

$$e=(OPi,OPj)\in LDDG^r,\quad \sigma^r(OPj)-\sigma^r(OPi))\geqslant\tau(e).$$

定理 1. 对一个给定的 $LDDG^r$，当且仅当其中的每一环路 C 满足：$\Sigma_{\forall e\in C}\tau(e)\leqslant 0$ 时，存在一个有效调度.

推论 1. 对不带环路的 $LDDG^r$，我们总可以找到一个有效调度.

基于同样的分析，我们用以下定理描述列号的确定对确定行号的影响.

定理 2. 设 (O,E,λ,δ) 是一个给定循环的 LDDG，假定已经确定了每个操作的列号，当且仅当 $cn(OPj)-cn(OPi)\geqslant-\lambda(e)$ 时，可以确定每个操作的行号满足：

$$\forall\ e=(OPi,OPj)\in E,\ rn(OPj)-rn(OPi)\geqslant\delta(e)-h*(\lambda(e)+cn(OPj)-cn(OPi)).$$

1.3　我们的新方法

DESP 方法的基本思想很简单，我们按以下两步对一个循环进行软件流水：一个是确定每个操作在新循环体中的行号，另一个是确定其列号. 对一个操作而言，如果它的行号和列号都确定了，那么它在新循环体中的位置也就确定了；如果所有操作的行号和列号都确定了，则新循环体也就形成了. 下面我们分两种情况来讨论：一种是先确定所有操作的行号，再确定其列号；另一种是先确定列号，再确定行号.

方法 1：先行后列(FRLC)

在这个方法中，首先为所有操作确定行号，过程如下：

对于每一个操作，寻找一个行号，使得新循环体的长度 h 最小. 且满足如下两个限制：

限制 1：对 $LDDG^r$ 中的每一环路 C，都有：$\Sigma_{\forall e\in C}\tau(e)\leqslant 0$.

限制 2：若两个操作有相同的行号，则它们不能占用同一个资源(或同一个资源段).

根据上述推论 1，若给定的 LDDG 是无环路的，我们可以直接按照限制 2 来调度操作. 然而，若给定的 LDDG 是有环路的，我们必须同时满足限制 1 和限制 2. 另外，由于所有环路都可能包含在 LDDG 的强连通块中，为了减少算法的复杂度，在确定行号之前，必须首先处理强连通块，处理过程如下：

1. 寻找 LDDG 中的所有强连通块；

2. 从强连通块中删除某些边，使得：

(1)修改后的强连通块没有环路，

(2)在资源无限的条件下，用列表调度法得到强连通块中每个操作的行号，调度结果必须满足限制 1 且高度 h 最小；

(3)删除不包含在强连通块中的所有边；

(4)用列表调度法，在修改过的 LDDG 及资源限制下得到每一个操作的行号.

详细算法在下一节中描述.

方法 2：先列后行(FCLR)

在这种方法中，首先确定每一个操作的列号. 我们可以很容易地确定新循环体长度的下限 $h_{lb}=MAX(h_0,h_{cs})$，其中 h_{cs} 是关键资源的使用次数，h_0 定义为：

$$h_0=MAX_{\forall C\in LDDG}\lceil\delta(C)/\lambda(C)\rceil,\ \delta(C)=\Sigma_{\forall e\in C}\delta(e)\ 且\ \lambda(C)=\Sigma_{\forall e\in C}\lambda(e).$$

在确定列号之后，希望给定的 LDDG 能转化成不带环路的图(用 $LDDG^{cr}$ 表示)，这实际上是要把那些满足 $cn(OPj)-cn(OPi)+\lambda(e)>0$ 的边删除掉. 很容易看到，若 LDDG 无环

路，我们可以确定操作的列号，并使所有的边满足 $cn(OPj)-cn(OPi)+\lambda(e)>0$；若 LDDG 有环路，我们不能确定列号，使所有的边都满足 $cn(OPj)-cn(OPi)+\lambda(e)>0$.

因此，这个问题可以这样来描述，确定每个操作的列号，使得：

1. $LDDG^{cr}$无环，

2. 对每条边 $e=(OPi,OPj)$，$cn(OPj)-cn(OPi)\geqslant-\lambda(e)$，

3. $LDDG^{cr}$高度最小.

为了解决上述问题，我们提出以下步骤：

1. 确定 LDDG 中所有强连通块 SCC；

2. 对所有 SCC 生成软件流水结果 $\sigma0=(rn0,cn0,h0)$；其中 $h0$ 是循环启动间隔，对 SCC 中的每条边 $e=(OPi,OPj)$，若 $rn0(OPj)-rn0(OPi)\geqslant\delta(e)$，则 $cn(OPj)-cn(OPi)+\lambda(e)=0$；若 $rn0(OPj)-rn0(OPi)<\delta(e)$，则 $cn(OPj)-cn(OPi)+\lambda(e)=\lceil(\delta(e)+rn0(OPi)-rn0(OPj))/h0\rceil$；

3. 对于任何不包含在 SCC 中的边 $e=(OPi,OPj)$，$cn(OPj)-cn(OPi)+\lambda(e)=1$；

4. 构造 $LDDG^{cn}=(O,E,cn)$，其中每条边 $e=(OPi,OPj)$ 的权值为 $cn(e)$，$cn(e)=cn(OPj)-cn(OPi)$；

5. 对 $LDDG^{cn}$中每条环路 C，$\Sigma_{\forall e\in C}\tau(e)\leqslant0$；所以可以确定列号.

详细算法描述在下一节中给出.

2 算法描述

定义 4. 对于一个给定的 LDDG，我们定义它的一个子图 $LDDG_{scc}$，$LDDG_{scc}$中包含 LDDG 的所有强连通块.

算法 1：从 LDDG 中寻找 $LDDG_{scc}$

1. 令 $LDDG'_{scc}=LDDG$；

2. 计算 LDDG 所有结点的入度和出度；

3. 若存在某结点的出度或入度为零，则从 $LDDG'_{scc}$中删除该结点及与之相连接的所有边，然后转 2，否则转 4；

4. 寻找 $LDDG'_{scc}$中的所有强连通块，定为 $LDDG_{scc}$.

算法 2：生成软件流水结果

1. 计算 $LDDG_{scc}$的初始启动间隔 $h0$；

2. 对 $LDDG_{scc}$中的每一条边，增加一个权值 d，$d(e)=\delta(e)-\lambda(e)*h0$；

3. 增加一个源结点 S，对 $LDDG_{scc}$中每个点，增加一条从 S 到该点权值为零的边，该图记作 $LDDG'_{scc}$；

4. 用最长路径算法，可以得到 $LDDG_{scc}$的一个有效调度，记作：

$$\sigma^d=(rn^d,cn^d,h0).$$

算法 3：用算法 1 和算法 2 修改 $LDDG_{scc}$

1. 用算法 1 寻找 $LDDG_{scc}$，若 $LDDG_{scc}$=空集，则返回；

2. 用算法 2 生成 σ^d；

3. 对 $LDDG_{scc}$ 中每条边 $e=(OPi,OPj)$，若 $rn_d(OPi)+\delta(e)>rn^d(OPj)$，则将 e 从 $LDDG_{scc}$ 中移走，并将该图记作：$LDDG_{scc-m}$.

2.1 FRLC 算法描述

1. 用算法 1 寻找 LDDG 中的 $LDDG_{scc}$；

2. 用算法 2 和算法 3 修改 $LDDG_{scc}$，得到 $LDDG_{scc-m}$；

3. 删除不包含在 $LDDG_{scc}$ 中的所有边，我们把这些边记作 Er，记 $LDDG_{scc}$ 的边的集合为 E_{scc}，$LDDG_{scc-m}$ 的边的集合为 E_{scc-m}，因而 $LDDG_m=(O,E-Er-E_{scc}+E_{scc-m},\delta)$；

4. 在 $LDDG_m$ 及资源限制下，用列表调度技术生成一个有效调度 σ；

5. 根据 σ，确定循环体的行号和高度 h，$rn(OPi)=\sigma(OPi)$；

6. 构造 $LDDG^r$；

7. 生成 $LDDG^r$ 的一个有效调度 σ^r；

8. 根据 σ^r 确定列号，$cn(OPi)=\sigma^r(OPi)$；

9. 利用行号和列号构造新循环体；

10. 形成装入和排空部分.

FRLC 算法的正确性可以由定理 1 及下面的定理 3、4 和 5 保证.

定理 3. 对于 LDDG 中的每一环路 C，$\Sigma_{\forall e\in C}d(e)\leqslant 0$.

定理 4. 若 LDDG 中的任一环路 C，$\delta(C)>0$，则 $LDDG_{scc-m}$ 无环路.

定理 5. 对于 $LDDG^r$ 中的任一环路 C，$\Sigma_{\forall e\in C}\tau(e)\leqslant 0$.

图 2 给出了一个完整的例子来说明 FRLC 算法的工作过程，这个例子的原始程序和它的 LDDG 见图 3. 我们假设该机器有一个加法器、一个乘法器和一个访问存储器部件，这些部件能并行工作，它们的工作时间都为一个周期.

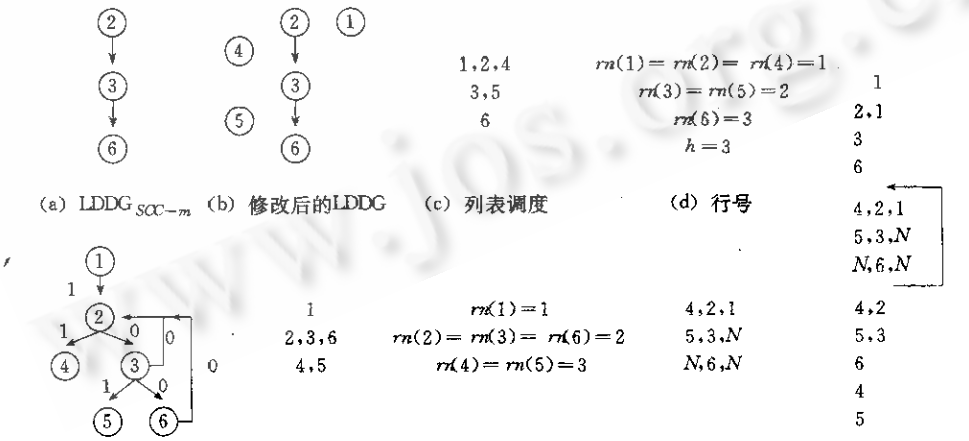


图2 用FRLC算法优化图3的循环程序

2.2 FCLR 算法描述

1. 用算法 1 寻找 LDDG 中的 $LDDG_{scc}$；

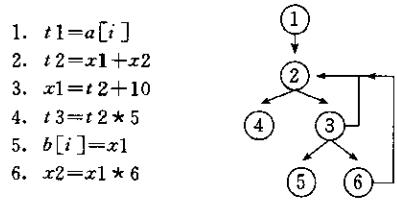


图3 一个循环的例子

2. 在资源无限的条件下，运用算法 2 对 $LDDG_{scc}$ 生成一个优化的软件流水结果：

$$\sigma 0=(rn0,cn0,h0);$$

3. 对任意边 $e=(OPi,OPj)\in E$，若 $rn0(OPj)-rn0(OPi)\geqslant\delta(e)$，则 $cn(OPj)-cn(OPi)=-\lambda(e)$；若 $rn0(OPj)-rn0(OPi)<\delta(e)$，则 $cn(OPj)-cn(OPi)=-\lambda(e)+\lceil\delta(e)+rn0(OPi)-rn0(OPj))/h0\rceil$；对于每一边 $e=(OPi,OPj)\in E-E_{SCC}$，$cn(OPj)-cn(OPi)=-\lambda(e)+1$；

4. 构造 $LDDG^{cn}$；

5. 用最长路径算法，生成 $LDDG^{cn}$ 的一个有效调度 σ^{cn}；

6. 对于 LDDG 中的每一操作 OP；$cn(OP)=\sigma^{cn}(OP)$；

7. 对于每一条边 $e=(OPi,OPj)\in E_{SCC}$，若 $cn(OPj)-cn(OPi)>-\lambda(e)$，将从 E_{SCC} 中删除，结果记作 E'_{SCC}；

8. 删除所有边 $e\in E-E_{SCC}$，我们得到 $LDDG^{cr}=(O,E'_{SCC},\delta_{new})$；

9. 在 $LDDG^{cr}$ 及资源限制条件下，应用列表调度技术生成 $LDDG^{cr}$ 的一个有效调度 σ^{cr}；

10. 对于 LDDG 中的每一操作 OP，$rn(OP)=\sigma^{cr}(OP)$；

11. 利用行号和列号构造新循环体；

12. 形成装入和排空部分.

FCLR 算法的正确性可以由定理 2 及下面的定理 6 和定理 7 保证.

定理 6. 对于 $LDDG^{cr}$ 中的任一环路 C，$\Sigma_{\forall e\in C}cn(e)\leqslant 0$.

定理 7. 如果任一环路 $C\in LDDG$，有 $\delta(C)>0$，则 $LDDG^{cr}$ 无环路.

FCLR 算法的详细例子见图 4.

3 讨论与比较

不难得出，DESP 算法的计算复杂度最坏为 $O(n^3)$，其中 n 是循环体中操作的个数.

3.1 与限制性软件流水算法比较

通常，把软件流水算法分为通用性和限制性两类. 通用性算法，例如完善流水法[2]，允许每个体有不同的调度结果和启动间隔. 限制性算法，如 URPR[4-6]，要求每个体有相同的调度结果和启动间隔.

限制性软件流水算法的主要思想是：在考虑资源、数据相关和周期性限制的同时，运用各种启发式调度策略构造新循环体. 然而，体间相关和资源限制使限制性软件流水算法非常复杂. 因而，现有的限制性算法并不能在低的计算复杂度下得到满意的时间效益.

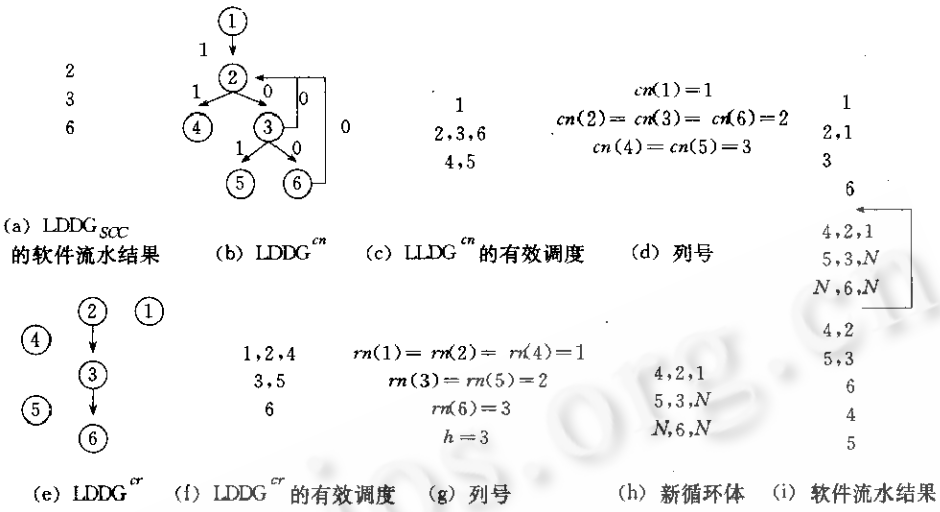


图4　用FCLR算法优化图3的循环程序

我们的DESP方法将软件流水分成两步，确定每个操作在指令矩阵中的行号和列号. 时间效益主要与行号确定有关；当我们确定行号时只受资源和修改后的LDDG的限制，而在修改后的LDDG中，绝大多数的边都被删除，所以DESP方法可以得到比现有限制性软件流水算法更好的时间效益.

3.2　与通用性软件流水方法比较

一般来说，通用性软件流水算法比限制性软件流水算法有更好的时间效益，但空间效益较差，而且算法的复杂度也更高. 通用性软件流水算法首先允许循环展开无限次，然后用一种调度策略来压缩无限展开的循环，希望找到一种“模式”，即新循环体. 这种模式可能很长，因而通用性软件流水算法的空间效益比较差；而且，寻找模式有时侯非常困难，因而计算复杂度很高.

DESP方法通过循环展开，可以得到与通用性算法相同的时间效益，在运用DESP方法之前，我们可以先确定循环展开的次数，展开次数可以由下面的公式计算：

$$K_{unroll}=\mathrm{MAX}_{\forall C\in LDDG}(\lceil \mathrm{MAX}(\delta(C)/\lambda(C))\rceil,0)$$

尽管循环展开可能增加新循环体长度，但带循环展开的DESP方法还是有比通用性算法好得多的空间效益.

3.3　与文献[9]中的算法比较

最近，Eisenbeis和Windheiser提出一种优化无环LDDG的软件流水算法，其内在思想与我们的FRLC算法非常相似，两种算法的不同处在于，我们的FRLC算法可以优化任意循环，而他们的算法只能优化LDDG无环的循环. 在资源限制的条件下，对于无环的LDDG，两种算法都能达到最优的结果.

3.4　与文献[8]中的算法比较

最近，Gasperoni和Schwiegelshohn提出一种新的循环调度算法，其内在思想与我们的FCLR算法很相似，然而从思想发展出来的算法不同. 在文献[8]中，循环首先被Bellman-Ford单一资源最长路径算法及一个二叉树搜索算法预处理，生成无资源限制的新循环体；

然后，若某一相关边在新循环体中保持，它将被留在修改后的 LDDG 中，修改后的 LDDG 无环；最后，利用修改后的 LDDG 在资源限制条件下列表调度得到新循环体.

在我们的 FCLR 算法中，首先处理强连通块并确定列号；然后，根据列号移走强连通块中的一些边和不在强连通块中的所有边，使 LDDG 无环；最后，运用列表调度法生成结果. 因为我们主要处理强连通块，因此 FCLR 算法的复杂度比他们的算法低；同样，因为 FCLR 算法比他们的算法能从 LDDG 中移走更多的边，因此 FCLR 算法常常能得到更优的结果.

3.5 DESP 算法的扩展

在我们的下一篇论文“带条件分支的指令级循环优化新方法”中，我们把本文中的 FRLC 算法扩充为可以处理条件分支的全局软件流水算法，其主要思想是：首先引入全局 LDDG(GLDDG)的概念；然后用 DESP 的思想修改 GLDDG，通过删除一些边使 GLDDG 无环；最后，运用全局代码调度技术(如路径调度法或渗透调度法)生成新循环体；同时，引入了重复体(Iteration—body)的概念，使新的循环体比较容易地构造出来.

4 结 论

在本文中，我们在软件流水方面提出了一种新观点，并基于这种观点开发出一种新的软件流水算法——DESP. 该算法将软件流水问题分解成两个子问题，一个是可以用列表调度法有效解决的，与循环无关的代码调度问题；另一个是可以用经典的图论算法很容易解决的与资源无关的问题. 手工模拟实验结果表明，DESP 算法与目前流行的软件流水算法相比，具有算法复杂性低，时间效益和空间效益好等优点.

目前，我们正在用 DESP 算法实现一个优化编译器，并通过实验对 DESP 算法作出更加全面的性能评价. 另外，基于本文中提出的对于软件流水的新观点，我们还提出了一个带条件分支的指令级循环优化新算法(GDESP)和新的寄存器分配技术.

参考文献

1 Touzeau R F. A Fortran compiler for the FPS—164 scientific computer. Proc. ACM SIGPLAN Symposium on Compiler Construction, 1984.

2 Aiken A, Nicolau A. Perfect pipelining: a new loop parallelization technique. European Symposium on Programming, 1988. 221—235.

3 Lam M S. Software pipelining: an effective scheduling technique for VLIW Machine. Proc. SIGPLAN'88 Conference on PLDI, 1988.

4 Su B, Ding S, Xia J. URPR——an extension of URCR for software pipelining. Proc. of MICRO—19, 1986. 104—108.

5 Su B, Ding S, Wang J *et al*. GURPR——a method for global software pipelining. Proc. of MICRO—20, 1987.

6 Su B, Wang J. GURPR*: a new global software pipelining algorithm. Proc. of MICRO—24, 1991.

7 Nakatani T, Ebcioglu K. Using a lookahead window in compaction—based parallelizing compiler. Proc. of MICRO—23, 1990.

8 Gasperoni F, Schwiegelshohn U. Scheduling loops on parallel processors: a simple algorithm with close to optimum performance, INRIA, 1992.

9 Eisenbeis C, Windheiser D. A new class of algorithms for software pipelining with resource constraints. Rapport de Recherche INRIA, 1992.

DECOMPOSED SOFTWARE PIPELINING: A NEW APPROACH TO EXPLOIT INSTRUCTION LEVEL PARALLELISM FOR LOOP PROGRAMS

Tang Zhizhong　Zhang Chihong

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

Wang Jian

(*INRIA, Domaine de Voluceau B. P.* 105—7815, *Lechesnay Cedex, France*)

Abstract　This paper presents a new view on software pipelining, in which the authors consider software pipelining as an instruction level transformation from a vector of one—dimension to a matrix of two—dimensions. Thus, the software pipelining problem can be naturally decomposed into two subproblems, one is to determine the row—numbers of operations in the matrix and another is to determine the column—numbers. Using this view—point as a basis, the authors develop a new loop scheduling approach, called decomposed software pipelining.

Key words　Loop scheduling, instruction level parallelism, software pipelining, loop—carried dependence, problem decomposition.