带条件分支的指令级循环优化新方法*

汤志忠 张赤红 王 剑

(清华大学计算机科学与技术系,北京 100084) (法国国家研究院 INRIA)

摘要 分解式软件流水 DESP 是我们最近提出来的一种对无分支循环进行有效调度的新方法,它通过把循环调度分解为两个子问题,把无分支调度问题转化为无环路图的调度,从而运用图论中一些经典的复杂度为多项式的方法来解决. 在本文中,我们把 DESP 方法扩展成可以优化带条件分支的循环,称为全局分解式软件流水方法——GDESP. 研究结果表明,GDESP 方法具有时间效益高和实用性好等优点,是一种有效实用的全局循环调度方法.

关键词 循环调度,指令级并行性,条件分支,全局软件流水.

在过去 10 年中,软件流水技术越来越受到重视. 它的基本思想是:一个循环可以在语义限制及资源限制的保证下重构,使得一次循环可以在前一次循环完成前启动. 语义限制即是操作先后次序的限制,资源限制与处理机的体系结构有关,它通过功能单元集及寄存器集的形式给出.

本文中,我们研究带有条件分支的软件流水,我们称做全局软件流水,以区别于不带分支的软件流水,我们称为局部软件流水. 目前对于全局软件流水已经有了一些方法,如 GURPR[1]和 GURPR*[2]全局软件流水算法,但是,由于体间相关的影响,这两种算法常常不能得到令人满意的空间与时间效益. 模限制算法[3]分别独立调度一个条件转移的两条分支,将这两条分支视作长度为最长情况的单一块;当最短分支执行最频繁时,它将得到一个时间效益相当坏的结果. 完善流水法[4]首先展开循环,然后调度,直至重复模式出现为止. 由于在资源限制下,保证收敛的条件并不清楚,这种方法可能会产生代码爆炸,使空间效益大大下降,在实际中很难应用. 基于相同思想的增强型完善流水法[5]运用软件窗口技术限制调度中的操作移动,这样可以得到很好的空间效益,但同时又降低了调度结果的时间效益.

最近,我们在软件流水方面提出了一种新观点,将软件流水问题很自然的分解成两个子问题,分别着眼于资源限制和语义限制,这一方法的关键在于通过删除一些相关边使数据相关图无环路,从而把局部软件流水问题转化为局部代码调度问题. 基于同样的想法,由于无体间相关回边的全局代码调度问题目前已经有了相当成熟的算法[6],因此,在本文中,通过

* 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

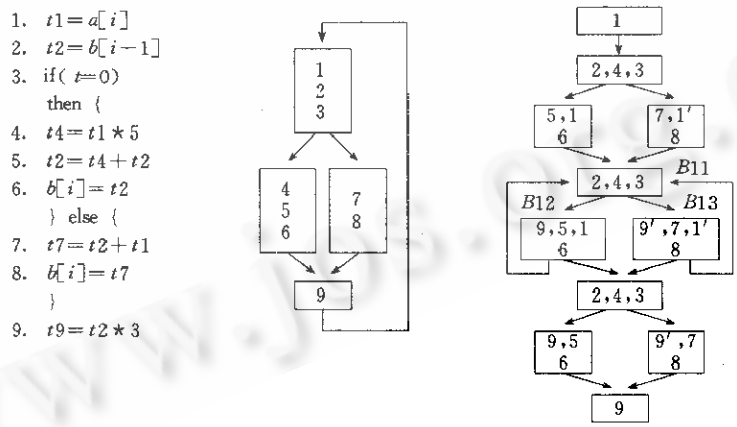
应用分解式软件流水思想,把全局软件流水问题转化为全局无环路的代码调度问题.

第1节讨论如何用局部分解式软件流水思想分解全局软件流水问题,第2节叙述全局分解式软件流水算法,第3节给出算法的详细描述,关于算法复杂性及与其它算法比较等问题在第4节讨论.

1　全局软件流水问题的分解

在文献[7]中,我们把局部软件流水看作是从一维指令向量到二维指令矩阵的指令级变换,从而很自然地把局部软件流水问题分解为两个子问题:确定每个操作在指令矩阵中的行号和列号.这种思想经过扩展,可直接用于全局软件流水问题.在局部软件流水中,原始循环体由一个基本块组成,流水后的新循环体也由一个基本块组成,而在全局软件流水中,原始循环体和流水后的新循环体各由一组有偏序关系的基本块组成.因而,我们可以将全局软件流水问题看作是从一组带偏序关系的指令向量到一组带偏序关系的指令矩阵的变换.实际上,偏序关系可以通过循环体的控制流图表示出来.

图1是一个全局软件流水的例子,其中图1(a)是一个循环体的串行代码,为了更加直观,我们用程序流图[2]表达原始循环体,如图1(b)所示;图1(c)是全局软件流水结果,其中新循环体由$B11$、$B12$和$B13$三个基本块组成.在全局软件流水时,操作的移动可能要跨越分支会聚点;因此,对一个操作来说,它在新循环体中可能有多个相同的拷贝.另外,新循环体中每个操作有不同的循环体号;比如,1,$1'$是第i个循环体的操作,2,3,4,5,6,7,8是第$i-1$个循环体的操作,9,$9'$是第$i-2$个循环体的操作.



(a) 一个循环体　　(b) 原始循环体流图　　(c) 全局软件流水结果

图1　一个全局软件流水的例子

在文献[7]中,我们定义了局部软件流水的行号和列号,对于全局软件流水,我们用下面的方法定义扩展行号和扩展列号.

定义 1. 对于全局软件流水,我们定义每个操作的扩展行号与扩展列号:

(1)设OP是一个操作,我们定义它的扩展行号为:$ern(OP)=(B(OP),rn(OP))$,其中$B(OP)$是OP所在的基本块号,$rn(OP)$是OP在$B(OP)$中的行号;

(2)设 OPi,OPj 是两个操作拷贝,OPi 属于第 i 个循环体,OPj 属于第 j 个循环体,则它们的扩展列号满足:$ecn(OPj)-ecn(OPi)=i-j$,且 $\mathrm{MIN}_{for\ all\ OP}(ecn(OP))=1$.

举例说明,在图 1(c)的新循环体中,$ern(2)=ern(3)=ern(4)=(B11,1)$,$ern(1)=ern(5)=ern(9)=(B12,1)$,$ern(6)=(B12,2)$,$ern(7)=ern(1')=ern(9')=(B13,1)$,$ern(8)=(B13,2)$,$ecn(1)=ecn(1')=1$,$ecn(2)=ecn(3)=ecn(4)=ecn(5)=ecn(6)=ecn(7)=ecn(8)=2$,$ecn(9)=ecn(9')=3$.

有了扩展行号和扩展列号两个概念,我们可以把全局软件流水分解为两个子问题,一个是确定每个操作的扩展行号,包括确定新循环体中各基本块的构成及它们之间的偏序关系,这一步可以用一种与循环无关的全局代码调度算法来解决,如路径调度法[5]或渗透调度法[6];另一个是确定每个操作的扩展列号.下一节,我们将用这种思想开发一种新的全局软件流水算法,称作全局分解式软件流水算法——GDESP.

2 GDESP:全局分解式软件流水算法

流图是表示控制相关最普通的方法,每个结点代表一个基本块,两个基本块之间的有向边表示它们之间的控制相关[5].一个基本块就是一个指令序列,除了第一条指令外,不会再有转移指令跳入该序列,除了最后一条指令外,也不会再有其它跳出这个序列的转移指令.为了表示控制相关和数据相关,我们首先定义几个概念.

定义 2. 设 $FG=(BB,CD)$是一个循环的流图,BB 是结点的集合,CD 是边的集合;若 $e=(Bexit,Bentrance)$表示一条循环回边,则 $Bentrance$ 是循环入口基本块,$Bexit$ 是循环出口基本块,执行路径就是一条从 $Bentrance$ 到 $Bexit$ 的简单路径.

定义 3. 分支数据相关是从条件转移指令到其它指令的相关;设 Bi 与 Bj 是条件分支指令 C 的两个直接后继基本块,Pi 与 Pj 是从 Bi 到 $Bexit$ 和从 Bj 到 $Bexit$ 的两条简单路径.如果变量 d 在 Pi 顶端活跃,并且在 Bj 有操作 OP 定义 d,则有从 C 到 OP 的分支数据相关.

定义 4. 对于体内、体间和分支三种数据相关,每条相关边 $e=(OPi,OPj)$有两个非负权值 $\lambda(e)$和 $\delta(e)$,$(\lambda(e),\delta(e))$表示 OPj 必须在前第 $\lambda(e)$个体的 OPi 执行 $\delta(e)$个周期后才能开始执行.

定义 5. 设 $FG=(BB,CD)$是一个循环的流图,$P=\{P1,P2,\cdots,Pn\}$是 FG 所有执行路径的集合;可以把每个 Pi 当作一个基本块构造其 $LDDG$,设 $LDDG(Pi)=(O(Pi),E(Pi))$,则全局循环数据相关图 $GLDDG$ 定义为:$(\bigcup_{\forall Pi\in P}O(Pi),\bigcup_{\forall Pi\in P}E(Pi))$.

全局分解式软件流水 GDESP 的基本思想可分为构造新循环体、确定扩展列号和引入重复体的概念构造新循环体三步,下面,对其中的关键问题作较详细的讨论.

2.1 构造新循环体

构造新循环体实际上是要确定新循环体中各基本块的构成、它们之间的偏序关系及每个操作的扩展行号.其中的关键是要尽可能多地删除 $GLDDG$ 中的数据相关边,并保证修改后的 $GLDDG$ 无环路.由于所有环路都包含在强连通块中,因此必须首先处理强连通块.

定义 6. 对于一个给定的 $GLDDG$,我们定义 $GLDDGscc$ 为 $LDDG$ 的一个子图,它包含

且仅包含 $LDDG$ 中的所有强连通块.

算法 1. 从 $GLDDG$ 中寻找 $GLDDGscc$ 的一个简单算法,如图 2(a)和图 2(b)所示.

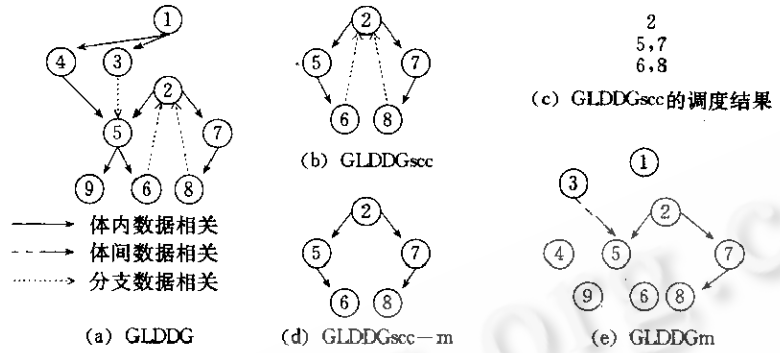


图2 从GLDDG变换到GLDDGm的一个例子

1. 令 $GLDDG'scc=GLDDG$;

2. 计算 $GLDDG$ 中所有操作的入度与出度;

3. 若有出度或入度为零的操作,从 $GLDDG'scc$ 中移走该点及与该点相联的边,转 2;

4. 寻找 $GLDDG'scc$ 中的强连通块,结果为 $GLDDGscc$.

算法 2. 视 $GLDDGscc$ 为无分支的 $LDDG$,用局部软件流水算法生成结果,如图 2(c).

1. 计算循环初始启动间隔 $h0$,其中 C 代表一个简单环路.

$$h0=\mathrm{MAX}_{\forall C\in GLDDG}\lceil\delta(C)/\lambda(C)\rceil,\delta(C)=\Sigma_{\forall e\in C}\delta(e)\text{ 且 }\lambda(e)=\Sigma_{\forall e\in C}\lambda(e)$$

2. 对 $GLDDGscc$ 中每条边增加一个权值 d,$d(e)=\delta(e)-\lambda(e)*h0$;

3. 添加一源点 S,对 $GLDDGscc$ 中每一操作,联接一条从 S 到该点的权值为 0 的新边;

4. 设 σ 为局部软件流水的调度结果,令 $\sigma(S)=1$;

5. 用计算最长路径算法,寻找 S 到各操作 OPi 的最长距离,用 $ld(S,OPi)$ 表示,规定 $\sigma(OPi)=ld(S,OPi)$;

6. 运用行号和列号的概念,σ 可以表示为 $(rn,cn,h0)$.

算法 3. 用算法 1 和算法 2 修改 $GLDDGscc$,使之无环路,如图 2(d)所示.

1. 用算法 1 寻找 $GLDDGscc$,若 $GLDDGscc$ 为空,则返回;

2. 用算法 2 生成 σ;

3. 对 $GLDDGscc$ 每条边 $e=(OPi,OPj)$,若 $rn(OPi)+\delta(e)>rn(OPj)$,则从 $GLDDGscc$ 中移走 e,我们把修改过的 $GLDDGscc$ 记为 $GLDDGscc-m$.

定义 7. 设 $GLDDG=(O,E)$,$GLDDGscc-m=(Oscc-m,Escc-m)$,定义修改过的 $GLDDG$ 为 $GLDDGm=(O,Escc-m\cup Ecc)$,$Ecc$ 为分支数据相关边,如图 2(e)所示.

定理 1. 若 $GLDDG$ 中每个环路 C,有 $\delta(C)=\Sigma_{\forall e\in C}\delta(e)>0$,则 $GLDDGm$ 是无环的.

2.2 确定扩展列号

为确定新循环体的扩展列号,我们首先构造新循环体的扩展数据相关图($EGLDDG$),它在原 $GLDDG$ 中增加了因全局代码调度产生的所有操作拷贝;然后计算循环距离 r,并构造 $EGLDDG^r$;最后根据 $EGLDDG^r$,用最长路径算法很容易确定扩展列号.

定义 8. 设 $GLDDG=(O,E,\lambda,\delta)$ 是给定循环的 $GLDDG$，$LBnew$ 是新循环体；我们定义 $LBnew$ 的扩展 $GLDDG$，记作 $EGLDDG(LBnew)$：

(1)每个结点代表 $LBnew$ 的一个操作；

(2)对于同一条执行路径上的两个操作 OPi' 与 OPj'，如果其原始操作为 OPi 和 OPj，若 $e=(OPi,OPj)$ 是 $GLDDG$ 的一条边，则在 $EGLDDG(LBnew)$ 中有边 $e'=(OPi',OPj')$，且令 $\lambda(e')=\lambda(e),\delta(e')=\delta(e)$.

定义 9. 设 $LBnew$ 是新循环体，$EGLDDG(LBnew)=(E0,EE,\lambda,\delta)$ 是其扩展的 $GLDDG$，我们定义 $EGLDDG^{\tau}(LBnew)=(EO,EE\cup Ec,\tau)$，其中：

(1)$Ec=\{e|e=(OP,OP'),OP,OP'\in EO$，且 OP,OP' 是同一操作的两个拷贝$\}$；

(2)$\forall\ e\in Ec,\tau(e)=0$；

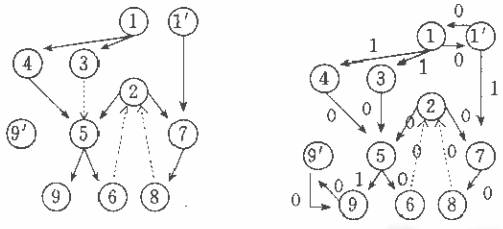
(3)$\forall\ e=(OPi,OPj)\in EE$；

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

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

其中，$dern(OPj,OPi)$ 表示 OPj,OPi 扩展行号的差，其绝对值实际是在新循环体中从 OPi 到 OPj 的路径长度；$dern(OPj,OPi)$ 可以很容易地由 $ern(OPi)$ 和 $ern(OPj)$ 计算得到，Lsp 表示在新循环体中，从循环入口到循环出口最短路径的长度.

在图 3 的例子中，图 3(a)是图 2(a)中的 $GLDDG$ 所对应的 $EGLDDG(LBnew)$. 设每个操作的延迟均为一个周期，则有 $Lsp=3,dern(1,3)=-1,dern(1',3)=-1,dern(2,7)=1,dern(3,5)=1$ 等；由此，我们得到图 3(b)中所示的 $EGLDDG^{\tau}(LBnew)$.



(a) EGLDDG(LBnew) (b) EGLDDG$^{\tau}$(LBnew)

图3 EGLDDG和EGLDDG$^{\tau}$

算法 4. 计算扩展列号

1. 在 $EGLDDG^{\tau}(LBnew)$ 中引入一个源结点，记作 S，从 S 到 $EGLDDG^{\tau}(LBnew)$ 的每个结点，引一条 τ 值为 0 的边；

2. 令 $ecn(S)=1$；

3. 运用最长路径算法，计算 S 到每个操作拷贝 OP' 的最长距离，记作 $ld(S,OP')$，并令 $ecn(OP')=ld(S,OP')$.

以上算法只有在 $EGLDDG^{\tau}(LBnew)$ 中的每个环路 C，$\Sigma_{\forall\ e\in C}\tau(e)\leqslant 0$ 时，才能应用.

定理 5. 对于任何环路 $C\in EGLDDG^{\tau}(LBnew)$，$\Sigma_{\forall\ e\in C}\tau(e)\leqslant 0$.

2.3 构造新的循环

我们先引入一个新概念，叫重复体(iteration－body).

定义 10. 设 $ECN=\{1,2,\cdots,K\}$，其中 $K=\mathrm{MAX}_{OPi\in LBnew}(ecn(OPi))$，我们称 $LBnew$ 为 ECN 重复体，记作 $ECN-ib$；设 $ECNs$ 是 ECN 的一个子集，如果从 $LBnew$ 中移走所有扩展行号为 $ECNs$ 的操作，剩下的结果叫 $ECN-ECNs$ 重复体，如图 4 所示.

现在，我们运用重复体的概念，按下列方法构造新的循环.

1. 设 $LBnew$ 是新循环体，$K=\mathrm{MAX}_{OPi\in LBnew}(ecn(OPi))$；

2. 寻找$\{1\}-ib$；

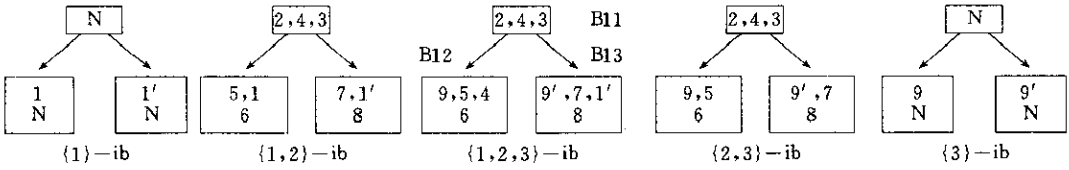


图4 重复体(用ib表示)

3. 设已经构造好$\{1,2,\cdots,K'\}-ib$,$k'<k$,我们寻找$\{1,2,\cdots,k'+1\}-ib$,并且从$\{1,2,\cdots K'\}-ib$的每一个出口基本块到$\{1,2,\cdots,K'+1\}-ib$的每一个入口基本块建立一条边,这一过程一直继续到建好$\{1,2,\cdots,k\}-ib$为止,$\{1,2,\cdots,K\}-ib$就是新循环体;

4. 从$\{1,2,\cdots,K\}-ib$的每一个出口基本块引入一条回边到它的入口基本块;

5. 设已经构造好$\{K',K'+1,\cdots,K\}-ib$,$K'<K$,我们寻找$\{K'+1,\cdots,K\}-ib$,并且从$\{k',k'+1,\cdots,K\}-ib$的每一个出口基本块到$\{k'+1,\cdots,k\}-ib$的每一个入口基本块建立一条边,这一过程一直继续到完成$\{K\}-ib$为止;

6. 在流水线限制下,删除空的环路,合并冗余操作.

定理 3. 在用*GDESP*方法生成的新的循环中,对任意$e=(OPi,OPj)\in EGLDDG$,如果(OPi,k)在第ti周期执行,$(OPj,k+\lambda(e))$在第tj周期执行,则$tj-ti\geqslant\delta(e)$.

3 算法描述

下面我们给出*GDESP*全局软件流水算法的详细描述,图5是一个完整的例子.

1. 构造给定循环的*GLDDG*;

2. 删除*GLDDG*中的一些边,使*GLDDG*无环路:

(1)用算法1从*GLDDG*中寻找*GLDDGscc*,

(2)在资源无限条件下,对*GLDDGscc*用算法2生成局部软件流水结果,

(3)用算法3修改*GLDDGscc*,

(4)由定义7得到*GLDDGm*;

3. 在*GLDDGm*、控制流和资源限制下,运用与循环无关的全局代码调度技术,例如路径调度法,对原始循环体进行调度,生成新循环体,记作*LBnew*;

4. 对*LBnew*中的所有操作计算扩展行号;

5. 计算扩展列号:

(1)由定义8得到$EGLDDG(LBnew)$;

(2)由定义9得到$EGLDDG^r(LBnew)$;

(3)用算法4对*LBnew*中的所有操作计算扩展列号;

6. 构造新的循环

(1)$K=\mathrm{MAX}_{OPi\in LBnew}(ecn(OPi))$;

(2)构造装入部分:

①寻找重复体$\{1\}-ib$,并令$k'=1$;

②*while* $(K<k')$ *do* {寻找$\{1,2,\cdots,K'+1\}-ib$,并且从$\{1,2,\cdots,K'\}-ib$的每一个出口基本块到$\{1,2,\cdots,K'+1\}$的每一个入口基本块建立一条边,$k'=k'+1$;}

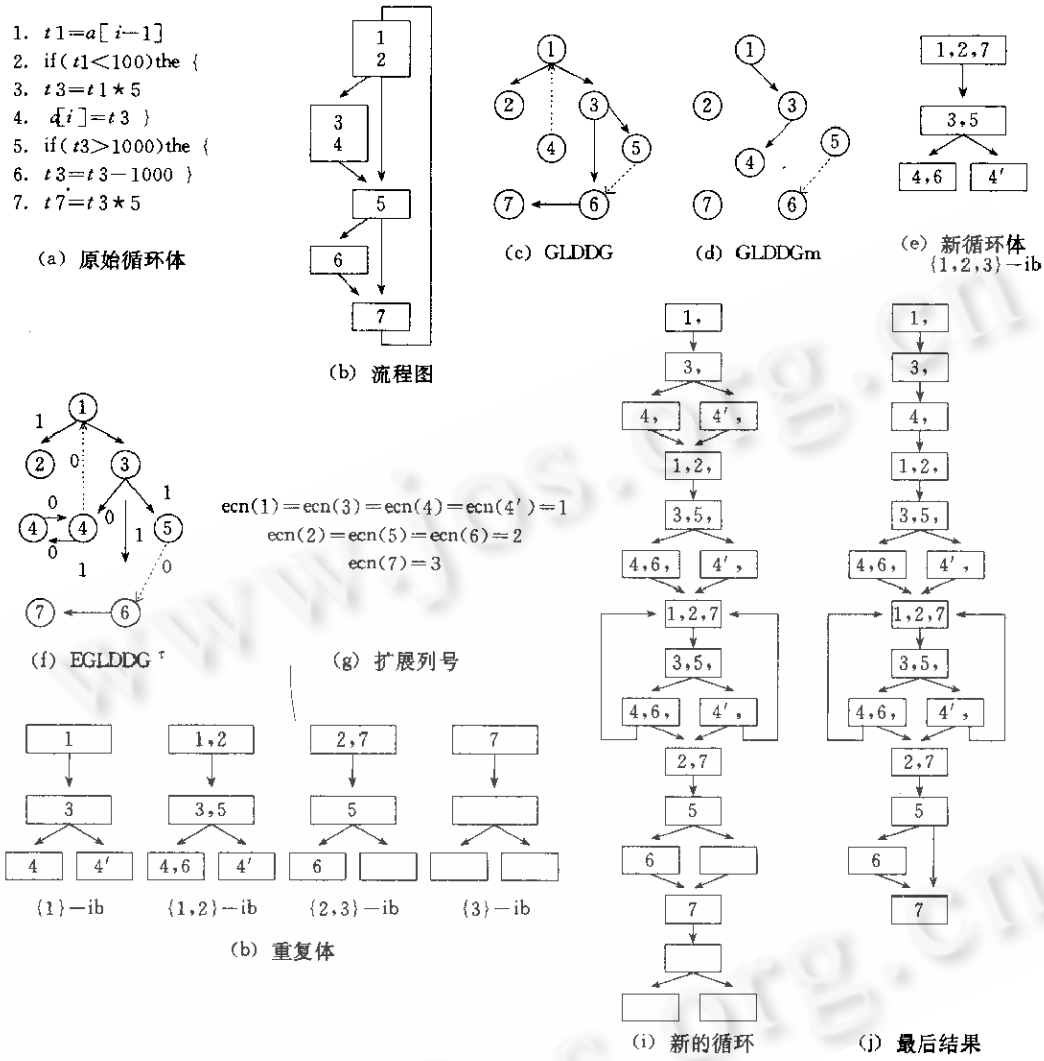


图5 GDEPS算法的一个例子

(3)从$\{1,2,\cdots,K\}-ib$的每一个出口基本块引一条回边到它的入口基本块;

(4)构造排空部分:

①令$k'=1$;

②*while* $(K'<k)$ *do* {寻找$\{K'+1,\cdots,K\}-ib$,并且从$\{k',k'+1,\cdots,K\}-ib$的每一个出口基本块到$\{k'+1,\cdots,k)-ib$的每一个入口基本块建立一条边,$k'=k'+1$;}

7. 在流水线限制下,删除空的环路,合并冗余操作.

4 讨论与比较

我们首先分析GDESP算法的计算复杂度,然后与几种全局软件流水方法作比较.

GDESP算法的计算复杂度:设n是给定循环的操作个数,在GDESP算法中,第1步是构造GLDDG,复杂度为$O(n^2)$;第2步寻找强边通块,并用到最长路径算法,复杂度为

$O(n^3)$;第 3 步要用到一种与循环无关的全局代码调度技术,若用路径调度法,复杂度为 $O(n^2)$;第 4 步的复杂度为 $O(n)$;第 5 步中,用到最长路径算法,复杂度为 $O(n^3)$;第 6 步中,我们首先寻找$(2k-1)$重复体,花费时间为 $O(k*n^2)$,由于 K 在给定的体系结构中是常数,因而第 6 步的复杂度为 $O(n^2)$. 因此,整个 GDESP 算法,在最坏情况下的计算复杂度为 $O(n^3)$.

时间效益:由于体间相关的影响,GURPR[1]与 GURPR *[2]两个全局软件流水算法的时间效益较差,另外,它们用插入指令的方法解决资源冲突,也带来了时间效益的恶化. 利用层次消减法的 Lam 算法也可以处理带分支的循环,然而,它将较短分支延长,与最长路径相匹配,因而时间效益较差. GDESP 算法可以有效地解决 GURPR、GURPR * 及 lam 算法所面临的问题,它通过运用一种与循环无关的全局代码调度技术生成新循环体,因此 GDESP 算法对大部分循环程序能得到满意的时间效益. 另外,对小循环来说,循环展开可以极大提高 GDESP 算法的时间效益.

代码爆炸问题:为了尽可能开发带分支循环的指令级并行性,现有的全局软件流水算法[8]首先假定循环被展开无限次,然后用全局循环无关代码调度技术压缩无限展开的循环. 在代码移动越过基本块时,为保证语义正确性而生成的代码的潜在大小是原代码的指数级. 增强型流水法[8]是一种改进的流水调度算法,它运用软件窗口控制代码爆炸,但限制了并行度的开发. 在 GDESP 算法中,我们只用与循环无关的全局代码调度技术压缩原循环体,因而代码爆炸可以有效得到控制.

兼容性:我们用一个新的术语"兼容性"来评价一种新的代码优化技术,若该方法可以很容易的在现有的编译器中实现,我们说该方法兼容性好,反之,兼容性不好. GDESP 算法实际上将循环调度问题分解为与循环无关的代码调度问题及一些简单问题,这些问题与资源限制无关,且很容易用图论的经典算法解决,对于与循环无关的代码调度问题,从 1979 年 J. Fisher 提出路径调度算法之后,已经有很多算法提出,并且在现有的实际编译器中被广泛地应用. 因而,基于与循环无关代码调度方法的 GDESP 算法,可以很容易在实际编译器中实现.

5 结　论

本文中,我们提出了一种带分支循环的指令级并行优化新方法,叫 GDESP 全局分解式软件流水算法,其基本思想是:把复杂的全局循环调度问题转化为与循环无关的全局代码调度问题和一些与资源限制无关的图论问题;前者可以用成熟的全局代码调度方法,如路径调度法或渗透调度法等有效地解决,后者可以用一些经典的图论算法解决.

因为 GDESP 算法中,GLDDG 首先被删除绝大部分数据相关边,然后按照资源限制、控制流及修改的 GLDDG 生成新循环体,因而 GDESP 算法有很好的时间效益,能有效控制一般全局软件流水面临的代码爆炸问题,且兼容性很好,可以很容易地用到实际编译器中.

目前,我们正在用 GDESP 算法实现一个 C 语言编译器,并且通过大量的例子对 GDESP算法作全面的性能评价,我们相信 GDESP 算法是一种很有潜力的全局软件流水方法,它必将得到广泛的应用.

参考文献

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

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

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

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

5 Fisher J A. Trace scheduling: a technique for global microcode compaction. IEEE Trans. on Computers, 1981, **C—30(7)**: 487—490.

6 Nicolau A. Percolation scheduling: a parallel compilation technique. Technical Report TR—85—678, Cornell University, 1985.

7 汤志忠，张赤红，王剑. 分解式软件流水 DESP——一种开发循环指令级并行性的新方法. 软件学报，1995，6(增刊)：138—147.

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

A NEW APPROACH OF INSTRUCTION—LEVEL OPTIMIZATION OF LOOPS WITH CONDITIONAL JUMPS

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 Decomposed software pipelining is a new approach for effective branch—free loop scheduling. By decoupling loop scheduling into two subproblems, branch—free loop scheduling problem is transformed into acyclic code scheduling problem, which can be resolved by the classical polynomial algorithm of graph theory. In this paper, this method is extended to the case of loop with conditional jumps, called global decomposed software pipelining. This results in an effective and practical approach for general loop scheduling, combining time efficiency of software pipelining with practicability of usual global loop—free code scheduling algorithms.

Key words Loop scheduling, instruction—level parallelism, conditional jumps, global software pipelining.