第34卷 第11期 计 算 机 工 程 2008年6月
Vol.34 No.11 Computer Engineering June 2008

·软件技术与数据库· 文章编号：1000—3428(2008)11—0077—03 文献标识码：A 中图分类号：TP311

分布内存系统中流水并行代码的自动生成

龚雪容[1]，陆林生[2]，赵荣彩[1]

(1. 解放军信息工程大学计算机科学与技术系，郑州 450002；2. 江南计算技术研究所，无锡 214083)

摘 要：并行循环分为DOALL和DOACROSS。DOACROSS循环携带数据依赖，在并行执行时需要通信支持，对于可以精确分析依赖关系的DOACROSS循环可通过流水并行方式提高性能。该文针对流水并行代码的自动生成进行讨论，包括数据依赖关系图和流水关系图的建立、流水并行判别准则和流水代码的自动生成等。实验证明流水并行后能获得较好的加速比。

关键词：流水并行；数据依赖关系图；流水关系图；流水通信

Automatic Generation of Pipeline Parallel Code in Distributed Memory System

GONG Xue-rong[1], LU Lin-sheng[2], ZHAO Rong-cai[1]

(1. Department of Computer Science and Technology, PLA Information Engineering University, Zhengzhou 450002;
2. Jiangnan Institute of Computing Technology, Wuxi 214083)

【**Abstract**】Parallel loops are divided into two kinds——DOALL and DOACROSS. Loops with data dependencies are often referred as DOACROSS loops. If the dependencies of DOACROSS loop can be precisely determined by compiler, pipeline parallel code for them can be created to improve the performance. This paper discusses the algorithms of creating the data dependence relation graph and pipeline relation graph, the discrimination rules of the pipeline parallel, and how to create the pipeline parallel code automatically. Experimental results show that the speedup ratio is satisfied with pipeline parallel.

【**Key words**】pipeline parallel; data dependence relation graph; pipeline relation graph, pipeline communication

1 概述

高性能计算是科学研究和开发的重要组成部分，在加速科学应用方面得到了广泛应用，然而在实际应用中，并行编程已被证明是非常困难的，因此，人们开始开发能自动生成并行程序的编译器，目前已有许多研究成果，如SUIF[1]等。

并行程序效率的高低与通信开销密切相关。提高程序并行度和降低通信开销是提高效率的主要手段，但简单挖掘程序的并行性并不总能获得高性能，其原因主要在于处理器间的通信开销相比计算和本地访存的开销要大得多。因此，降低通信开销就显得尤为重要，特别是分布存储系统的并行化编译器。在本文的系统中，目标并行程序采用了冗余并行执行模型，处理器间的通信由程序执行过程中的步通信和流水并行执行中的流水通信两部分组成，本文主要讨论流水并行代码的自动生成和流水通信。

2 基本概念和相关工作

2.1 基本概念

(1)迭代空间 $\boldsymbol{I}$：一个 n 层循环嵌套的迭代空间 $\boldsymbol{I}$ 定义为

$$\boldsymbol{I}=\left\{i=(i_1,i_2,\cdots,i_n)\in\Im \,\middle|\, \forall k=1,2,\cdots,n;\ \begin{matrix} i_k\ge l_k\,(\boldsymbol{v},i_1,i_2,\cdots,i_{k-1}) \\ i_k\le h_k\,(\boldsymbol{v},i_1,i_2,\cdots,i_{k-1}) \end{matrix}\right\}$$

其中，$\boldsymbol{I}$ 表示迭代空间向量；i 表示迭代空间的一次迭代；$\boldsymbol{v}$ 是符号常向量；l_k 和 h_k 是仿射函数。

(2)数据空间 $\boldsymbol{A}$：对一个 m 维的数组 $\boldsymbol{A}$，数据空间 $\boldsymbol{A}$ 定义为

$$\boldsymbol{A}=\left\{a=(a_1,a_2,\cdots,a_m)\in A\mid k=1,2,\cdots,m;0\le a_k\le u_k\right\}$$

其中，$\boldsymbol{A}$ 表示数据空间向量；a 表示数据空间的一个数据；u_k 表示数据空间第 k 维的上界。

(3)处理器空间 $\boldsymbol{P}$：对一个 q 维的处理器数组 $\boldsymbol{P}$，处理器空间 $\boldsymbol{P}$ 定义为

$$\boldsymbol{P}=\{p=(p_1,p_2,\cdots,p_q)\in\boldsymbol{P}\mid k=1,2,\cdots,q;0\le p_k\le u_k\}$$

其中，$\boldsymbol{P}$ 表示处理器空间向量；p 表示处理器空间的一个处理器；u_k 表示处理器空间第 k 维的上界。

2.2 冗余并行执行模型

冗余并行执行模型[2]是一种程序执行模型，如图1所示。

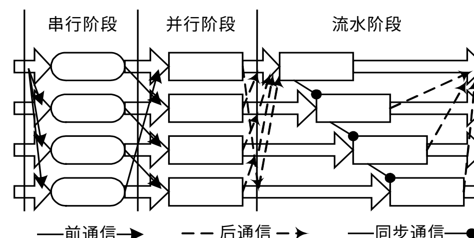

图1 冗余并行执行模型框架

在该模型中程序的执行阶段分为串行、并行和流水3种。串行阶段程序在所有的处理器上冗余执行；并行阶段程序在

基金项目：国家部委科研基金资助重点项目

作者简介：龚雪容(1975－)，女，助理工程师、博士研究生，主研方向：并行识别；陆林生、赵荣彩，教授、博士生导师

收稿日期：2007-07-22 **E-mail**：gongxuerong@163.com

多个处理器上并行执行；流水执行阶段任何一个时刻都只有一个处理器在执行，多个处理器以流水的方式执行一个循环的迭代。在该模型中，串行和并行执行阶段内部不需要通信，而流水执行阶段内部存在流水通信；不同的执行阶段之间存在通信，每个阶段开始前的通信为前通信，其作用是从数据的拥有者处获得数据，每个阶段结束后的通信为后通信，其作用是把在本处理机上修改过的数据发送给数据的拥有者，以保证各处理器数据的一致性。

2.3 计算和数据划分

计算划分$\boldsymbol{C}$[3]是迭代和处理器对的集合，将迭代映射到处理器，用函数 $p=\boldsymbol{C}\times i+\gamma$ 表示，其中，$\boldsymbol{C}$是 $q\times n$ 的矩阵；p 表示处理器空间的一个处理器；γ 是常向量。

数据划分$\boldsymbol{D}$[3]是数据元素和处理器对的集合，将数据元素映射到处理器，用函数 $p=\boldsymbol{D}\times a+\delta$ 表示，其中，$\boldsymbol{D}$是 $q\times m$ 的矩阵；p 表示处理器空间的一个处理器；δ 是常向量。

3 流水并行代码自动生成

通常，可并行的循环分为DOALL和DOACROSS[4]。前者没有数据依赖，不需要通信支持；后者存在数据依赖，需要通信支持。可精确确定依赖关系的DOACROSS循环又称为规则DOACROSS循环，这类循环可以使用流水方式并行执行。在流水执行时每个处理器计算一个流水块，把计算结果传给下一处理器，后一处理器必须等待前一处理器的执行结果，所有的处理器按照流水线方式执行所有计算。

例 1

```
#define N 128
for(i=4; i<=N-1; i++)
    for(j=3; j<=N-1; j++)
        {
        x[i][j]=x[i][j]+x[i][j-1]+b[i][j];
        }
```

例 2

```
for(i=4; i<=N-1; i++)
    for(j=3; j<=N-1; j++)
        {
        x[i][j]=x[i][j]+x[i-1][j]+b[i][j];
        }
```

在例 1 和例 2 中，如果计算划分是 $C=[1\ \ 0]+0$，数据划分为 $D=[1\ \ 0]+0$，则例 1 可以无通信的并行执行，是 DOALL 循环；例 2 则因为存在循环携带的依赖关系($x[i][j]$依赖于$x[i-1][j]$)需要通信支持，在该例中依赖关系可精确确定，是一个规则 DOACROSS 循环。对例 2 中的循环有 3 种处理方法：(1)串行执行；(2)数据重分布；(3)流水执行。本文主要讨论第(3)种处理方法。

3.1 分块策略

在流水并行中流水块大小的选择是决定流水性能高低的关键，分块方法可分为静态和动态。若流水块大小在整个流水过程中保持不变则这种分块方式为静态的；反之则为动态的。这 2 种方法各有优缺点，静态分块实现简单但很难得到最好的流水性能，特别是在处理器的性能差异较大的时候；动态分块尽管可以得到较好的流水性能但实现相比静态分块要难得多。从串行程序并行化的角度来讲，采用静态的分块策略易于实现而且也能取得一定的加速比，因此，本文采用静态分块策略。

定义 1 计算块是指分解方向的迭代。计算块的总数 PV 是分解方向的迭代数即虚拟处理器数，PV 的值是从计算和数据划分得到的。

定义 2 物理块由几个同时在同一处理器执行的计算块组成，记为 blk。

3.2 数据依赖关系图

根据依赖关系分析提供的依赖关系信息可以创建各计算块之间的数据依赖关系图(DDRG)。

DDRG 是一个加权有向图 $G_{\text{iter}}=(V,E,\omega,\tau)$，其中顶点 $V=\{v_1,v_2,\cdots,v_n\}$ 与计算块对应且顶点个数与计算块个数相同，为 n；边 $E=\{e_1,e_2,\cdots,e_m\}\subseteq V\times V$，表示了计算块之间的依赖关系和通信关系，如果 $E=\varphi$ 说明所有计算块之间都不需要通信；ω 表示节点权重 $\omega:V\rightarrow R$，τ 表示边权重 $\tau:E\rightarrow R$。该加权有向图也可以用一个邻接矩阵 $\boldsymbol{A}_{iter}$ 表示，其中，$\boldsymbol{A}_{iter}=(a_{ij})_{n\times n},1\le i,j\le n$，$a_{ij}$=1 (即存在一条从 v_j 到 v_i 的边)表示节点 v_i 需要 v_j 的数据；$a_{ij}=0$ 表示节点 v_i 不需要 v_j 的数。

创建一个循环嵌套的 DDRG 图的算法如下：

```
输入：计算和数据划分信息；依赖关系信息；迭代空间
输出：DDRG
步骤：
利用计算分解和迭代空间确定 DDRG 中的节点，并用 0 标记
利用数据分解和迭代空间确定 DDRG 中每个节点的数据
While(DDRG 中存在标记为 0 的节点){
  利用依赖信息确定该节点（vi）与其他所有标记为 0 的节点（设为vj）之间的通信关系：
   If(两个节点之间存在依赖关系){
     If（vi需要vj）增加一条从vj到vi的有向边
     If（vj需要vi）增加一条从vi到vj的有向边
   }
   标记该节点为 1
}
return
```

如果DDRG中没有回路，则该循环可以以流水方式执行，而如果在 DDRG 中存在回路，则说明 2 个计算块相互需要对方的数据，此时不能采用流水方式执行。如果该回路可以通过算法消除则一样可以采用流水并行执行方式，在实际应用中可以通过对计算块的合理调度以及修改相应的计算和数据划分方式来达到消除 DDRG 中回路的目的。计算块调度算法主要目的是消除 DDRG 中的回路，主要思想是通过修改计算和数据分解方式来消除 DDRG 中的回路，具体算法如下，算法输出是布尔变量 del，若 del 为真则表示可以消除回路，反之则不能消除。

```
输入：DDRG
输出：del
步骤：
If(DDRG 中有回路){
If（在所有方向均存在回路）del=false
Else if（存在没有回路的方向）{
  修改计算和数据分解信息：只分解迭代空间中没有回路的维
  del=true
  }
}
Return del
```

图 2(a)是利用未修改的计算和数据划分信息以及依赖关系信息建立的 DDRG 图，在 i 和 j 方向存在回路，但是 k 方向没有，此时就可以通过修改计算和数据划分信息来消除这

些回路，修改后的计算和数据划分只划分 k 维而不是 i, j 和 k 三维，在修改后的计算和数据划分信息基础上建立的 DDRG 图见图 2(b)。

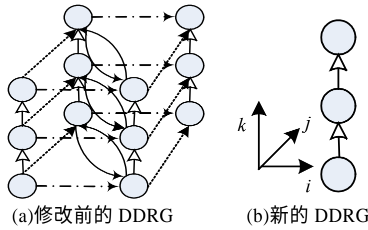

图 2　修改前后的 DDRG 图

3.3　流水关系图

在 DDRG 基础上可以创建处理器间的流水关系图(PRG)。PRG 是一个加权有向图 $G=(U,F,\rho,\sigma)$，表示处理器间的流水关系，其中，$U=\{u_1,u_2,\cdots,u_p\}$ 表示 p 个进程；进程间的通信关系用 $F=\{f_1,f_2,\cdots,f_k\}\subseteq U\times U$ 表示；$\rho:U\to R$ 表示节点权重；$\sigma:F\to R$ 表示边的权重。创建 PRG 的算法如下：

```
输入：P 个进程集合 U；DDRG；流水块大小 blk
输出：PRG
步骤：
标记 DDRG 中的节点为未分配，用 0 标记；
为 PRG 建立 p 个节点；
按照流水方向和 blk 确定各进程每次执行的计算块；
Count=0；
j=0；
While(DDRG 中存在节点标记为 0 的节点){
  vi-1＝vi；
  沿流水方向取节点vi；
  if(若vi标记为 0){
   将vi分配给进程uj；
   标记vi为 1，表示vi已分配；
   Count++;
   if(count >blk) {
     j++；
     if(vi-1与vi间存在边) 建立一条从uj-1到uj的边；
       j=j mod p；
   }
  }
}
Return；
```

3.4　流水代码自动生成

创建PRG后即可依据流水并行规则[5]确定是否需要生成流水并行代码。在一些特定情况下流水是不能实现的，如某些复杂物理问题，对于这类问题很难自动分析其依赖关系以及进行数据和计算划分。目前在自动生成并行程序时对这类问题往往是串行执行的。自动生成流水并行代码算法如下：

```
输入：循环迭代空间；计算和数据分解；依赖关系
输出：流水并行代码
步骤：
创建 DDRG；
If (若 DDRG 中存在回路){
  消除 DDRG 中的回路；
  If(回路不能消除 ) return;
  }
创建 PRG；
If(在 PRG 中有回路) return；
If(若在进程 P 的左边存在进程 p-1) 生成 p 进程的前通信代码；
生成 p 进程的计算代码；
If(若在进程 P 的右边存在进程 p+1) 生成 p 进程的后通信代码；
return；
```

4　性能测试和分析

例 2 的迭代空间为 $4\le i\le 127, 3\le j\le 127$，计算和数据划分为 $C=[1\ \ 0]$，$D=[1\ \ 0]$，读写操作 $x[i][j]$ 和 $x[i-1][j]$ 间的依赖关系为 $iw=ir-1, jw=jr$，依赖级为 1。设有 4 个物理处理器，记为 $p0\sim p3$。

4.1　静态性能分析

假设计算一个数据的时间为 T_e，启动一次通信的时间为 T_{st}，传输一个数据的时间为 T_{tr}，则非流水并行执行时间为 T_1 =15 375 T_e +3 T_{st} +375 T_{tr}。流水并行执行时间为 T_2 = 4 000 T_e + 84 blk × T_e +3 T_{st} +3 blk × T_{tr}。假设 blk =5，则 T_2 = 4 420 T_e + 3 T_{st} + 15× T_{tr}。

4.2　实验结果

对本文的算法进行试验和静态分析，图 3 是以例 2 中的循环为例在较大规模时的测试结果，规模为 4 096×4 096，测试环境是 8 节点的 SunWay 集群，每个节点有 2 个 2.8 GHz Xeon CPU, 4 GB 存储器，16 MB 二级 Cache 和 128 KB 一级 Cache。各节点通过 100 Mb/s 交换机连接，OS 是 Red Hat Linux 7.2 2.96-118.7.2 smp，MPI 编译器是 MPICH 1.2.6。图 3 显示的是使用流水并行后的加速比，ns 表示在一个节点启动几个进程，np 表示启动的总进程数。由图 3 可知，加速比令人满意，这也说明流水是有效果的。

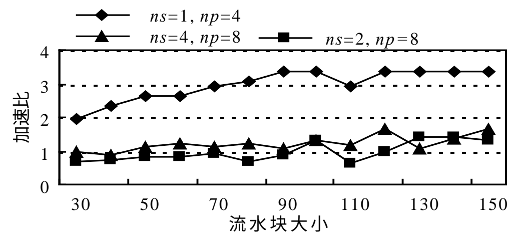

图 3　采用流水并行的加速比

5　结束语

本文针对流水并行代码的自动生成进行了讨论，包括 DDRG 和 PRG 图的建立、流水并行判别准则和流水代码的自动生成等。实验证明流水并行后能获得较好的加速比。在提高流水并行性能方面还可以采用动态的分块策略进一步提高流水性能。

参考文献

[1] Stanford SUIF Compiler Group. SUIF: A Parallelizing Optimizing Research Compiler[R]. Computer Systems Lab, Stanford University, Tech. Rep.: CSL-TR-94-620, 1994-05.

[2] 钟洪涛. 基于区域图数据流分析的通信优化算法[J]. 软件学报 2003, 14(2): 175-182.

[3] 董春丽. 并行编译中一种线性数据和计算划分算法[J]. 计算机工程, 2006, 32(24): 26-28.

[4] Hurson A, Lim J T. Parallelization of DOALL and DOACROSS Loops——A Survey. [S. l.]: Academic Press, 1997.

[5] 陆林生. 并行程序概念设计方法的研究[J]. 计算机学报, 2003, 26(9): 1086-1093.