

一种基于代表元的划分算法

张为华 王 鹏 臧斌宇 朱传琪

(复旦大学并行处理研究所 上海 201203)

摘 要 划分是把程序中不同的计算和数据分配到并行处理系统的不同处理机来充分利用并行系统的计算资源、提高程序处理速度的一种优化技术. 划分的效果对程序在并行系统上的执行效率将产生至关重要的影响, 因此划分问题一直是并行领域研究的一个热点. 但是应用程序的一些特性, 如非紧密嵌套循环、一条语句对非只读数组的多次引用间存在重叠、不同语句对同一数组不同步长的引用, 给有效解决划分问题设置了极大的障碍. 已有的划分算法无法对具有这些特征的程序进行自动划分. 虽然在对具有这些特征的程序进行手工优化过程中, 存在一些直观上的划分策略, 但这些策略无法应用到编译器中来指导编译器完成对程序的自动划分. 文中根据这类程序的特点, 提出了一种基于代表元的划分算法. 该算法通过使用程序中对划分计算产生实际影响的数组引用作为代表元素构造各种划分的限制条件, 完成程序的划分. 同时通过寻找最大一致性数据划分方向有效减少了程序划分过程中的数据重组通信. 该算法已经在 AFT2004 中实现, 并对应用程序获得了很好的效果.

关键词 计算划分; 数据划分; 代表元; 非紧密嵌套循环; 数据划分一致性; 并行编译

中图法分类号 TP302

An Affine Partition Algorithm Based on Representative Element

ZHANG Wei-Hua WANG Peng ZANG Bin-Yu ZHU Chuan-Qi

(Parallel Processing Institute, Fudan University, Shanghai 201203)

Abstract Partition is an optimization technique that distributes computations and data onto the different processors of parallel systems to get the maximizing parallelism and minimizing communication. The effect of partition algorithm can directly affect the performance of parallel systems. But there are many obstacles to effective partition in practical programs, such as imperfect loop nests and different array access scope. Previous partition algorithms can only finish the partition of sequences of perfect loop nests or cannot solve data partition consistent problem for different array access scope. This paper presents an affine partition algorithm based on representative element. When constructing the constraint relation for partition, it only remains the array references, which have contributes to partition constraint relation indeed and remove the trivial partition conflicts through discarding redundant array references to same array. This paper also presents a consistent partition algorithm to solve the data partition consistent problem. The algorithms can not only solve more practical partition problems, but also effectively reduce data reorganization communication in data partition. This technique has been implemented in AFT2004 parallel compiling system and can get better results for some practical programs.

Keywords affine partition; data partition; representative element; imperfect loop; parallel compiling

收稿日期: 2006-09-10; 最终修改稿收到日期: 2007-12-21. 本课题得到国家博士点基金(20050246020)资助. 张为华, 男, 1974年生, 博士, 讲师, 主要研究方向为体系结构与编译优化. E-mail: zhangweihua@fudan.edu.cn. 王 鹏, 男, 1979年生, 博士研究生, 主要研究方向为体系结构与编译优化. 臧斌宇, 男, 1965年生, 教授, 博士生导师, 主要研究领域为体系结构与编译优化. 朱传琪, 男, 1943年生, 教授, 博士生导师, 主要研究领域为体系结构与编译优化.

1 引言

划分是把程序中不同的计算和数据分配到并行处理系统的不同处理机来充分利用并行系统的计算资源、提高程序处理速度的一种优化技术,划分的效果对程序在并行系统上的执行效率将产生至关重要的影响.作为目前各种大规模科学计算主要处理平台的分布式主存多处理机系统,一直把划分问题作为其领域要解决的关键问题.在分布式主存多处理机系统上,程序的划分由计算划分和数据划分两部分组成,计算划分是把不同的计算部分分配到不同处理机来充分利用并行系统计算资源,进行计算划分的循环称为计算的划分方向.数据划分是把不同计算所需的数据分配到不同处理机存储单元的过程,进行数据分割的数组下标维称为数组的数据划分方向.由于不是所有的数据都能与需要它的计算

分配到同一处理机的存储单元中,在划分过程中可能产生两种类型的通信:邻近处理机通信和数据重组通信^[1].在这两种类型的通信中,邻近处理机通信的开销要远远小于数据重组通信的开销,因此减小划分过程中数据重组通信是最小化通信的首要目标,而如何减少划分过程中的数据重组通信在本文称为数据划分一致性问题.

在实际的应用程序中,非紧密嵌套循环、不同语句对数组元素访问步长的不同以及一条语句对数组的多次引用存在重叠是非常普遍的现象.当这些情况出现在程序中时,划分尤其是数据的划分变得更加复杂.图 1 中的程序是这种情况的一个例子,在该程序中,SCC2 是一个非紧密嵌套循环,语句 S1 和 S2 对非只读数组 A, B 访问的步长不同,数组 B 在 S1 中的 6 个引用和数组 A 在 S2 中的 6 个引用存在数组元素访问的重叠,造成语句 S1 和 S2 不存在无通信的划分方向.

```
for i=1 to 100 do //SCC1
  for j=1 to 100 do
    for k=1 to 100 do
      A[i,j,k]=A[i,j-1,k]+B[2i,j,2k-1]+B[2i,j,3k]+B[2i,j,4k+1]+
        B[2i-1,j,2k-1]+B[2i-1,j,3k]+B[2i-1,j,4k+1]; S1
for i=1 to 100 do //SCC2
  for j=1 to 100 do
    C[i,j,1]=B[0,j,0]
    for k=1 to 100 do
      B[i,j,k]=C[i,j,1]*B[i-1,j,k]+A[i,2j-1,2k]+A[i,3j,2k]+A[i,4j+1,2k]+
        A[i,2j-1,3k]+A[i,3j,3k]+A[i,4j+1,3k]; S2
```

图 1 程序示例

虽然已有许多划分算法,但是这些已有的划分算法存在一些不足.一些划分算法只能解决紧密循环嵌套序列的划分问题^[1-10],一些划分算法只能发现无通信的划分方向^[11],一些算法无法解决分布主存多处理机系统数组元素访问步长不同时数据划分一致性问题^[12-14],而另一些算法则只能解决只读数组复制的问题^[15-16].因此已有的并行编译系统(如 Stanford 大学的 SUIF 系统、复旦大学的 AFT 系统)中采用的传统划分算法都无法自动完成对图 1 中这类程序的自动划分.虽然在针对这类程序的手工划分过程中存在一些经验性的划分策略,但这些策略无法应用到编译器中来指导编译器完成对这类程序的自动划分.因此目前对这类程序的划分主要是手工方式.然而,目前具有这种特征的应用程序众多^①,而对程序进行手工划分是一项十分繁琐的工作,不仅需要程序员对分布式主存系统的各种特性比较了解,也要程序员花费大量时间对应用程序进行分析,这种情况十分不利于分布式主存系统的应

用推广.在这种情况下,如何设计一种算法,增强编译器对应用程序自动划分的能力,指导编译器完成此类程序的自动划分就显得十分重要.

对于图 1 中这类程序的数据划分,同一语句对数组的多个引用存在重叠,不同语句对同一数组访问的步长不同,这种情况下程序不存在无通信的划分方向.由于数据重组通信的开销远远大于邻近处理机通信,因此在对这类程序进行划分时,虽然不能获得无通信的划分方向,但应保证同一数组多个引用尽可能沿相同的数据划分方向进行划分,以消除划分过程中的数据重组通信.也就是说,为了避免数据重组通信,同一语句中存在的多个对同一数组的引用应延相同的数据划分方向进行划分,因此在计算程序的划分时只需保留真正影响划分方向计算的数组引用做为代表元素,来计算数组的数据划分方向,这样可以消除同一语句对数组多个引用

① <http://www.mgnet.org>

存在重叠对数据划分方向的确定造成的冲突.而对于不同语句,为了消除划分过程中的数据重组织通信,只需保持不同语句中相同数组的代表元素确定的数据划分方向相同,而不用考虑不同步长对数据划分造成的影响,以语句为划分单位也解决了非紧密嵌套循环的划分问题.

而对于此类程序的计算划分,当无法获得无通信的划分方向时,如果语句中同一数组的两个引用对于程序中同一数组的所有其它引用的依赖方向向量集完全相同,则其中的一个引用对于程序的计算划分是冗余的,因此在构造计算划分限制条件时,只保留其中的一个作为代表元素来保持程序的依赖关系,计算程序的计算划分即可.

基于上面的分析,在对具有图 1 中这类特征的程序进行划分时,可以先选取用于确定数据划分方向的代表元素,再通过原程序的依赖关系,调整代表元素,以保证计算划分的正确性.本文在以上分析的基础上,针对分布式主存多处理机系统,提出了一种基于代表元的自动划分算法.该算法在对不存在无通信划分方向的程序进行划分时,通过使用程序中对划分计算产生实际影响的数组引用构造代表元程序,并把代表元程序数据划分和计算划分的解作为原始程序数据划分和计算划分的解.同时通过寻找最大一致性数据划分方向有效解决了数据划分一致性问题,减少程序划分过程中的数据重组织通信.该算法弥补了传统并行编译系统无法完成针对不存在无通信划分方向程序的自动化划分的缺陷,可以指导编译器完成更多程序的自动划分,并很好地解决了数据划分一致性问题.该算法已经在复旦大学并行处理研究所开发的 AFT2004 中实现,并对应用程序取得了较好的效果.

本文第 2 节介绍相关工作以及后面要使用的符号和定义;第 3 节给出代表元的概念、表示和如何构造代表元程序;第 4 节给出基于代表元的无通信划分算法以及为解决数据划分一致性问题的一致划分算法;第 5 节给出了实验结果;第 6 节对全文进行了总结.

2 背景知识

2.1 相关工作

作为并行领域的关键技术,针对划分技术的研究一直是并行领域研究的一个热点.许多研究机构都对划分问题进行了研究^[1-17].根据划分算法各自的特

点和适用范围,可以把已有的划分算法分成 3 类:

(1) 紧密嵌套循环级别的划分. Banerjee^[2] 和 Wolf 等^[3-4] 提出了 U 模转换模型,在这种模型下如何求解一个循环嵌套最粗粒度的并行性问题被转化为如何转换循环序列以获得最外层并行的问题. Anderson 等^[1,5-6] 提出了一种如何分解数据和计算以获得数据最小迁移的仿射分解算法. Bau 等^[7] 提出了一种如何对齐计算和数据以获得无通信的划分方法. Ramanujam^[8] 提出了一种基于超平面无通信的计算和数据划分方法,但是他们的算法只给出了单个嵌套循环的计算划分方法且只能对二维数组进行数据划分,另外,由于超平面最后所求得的计算和数据划分有限,因此并行度受到了限制. Huang^[9-10] 在文献[8]的基础上将超平面算法推广到任意维数组和多个嵌套循环,但他们的算法也无法摆脱超平面算法并行度有限的限制.虽然文献[1-10]这些算法出发点和解决的问题不尽相同,但是他们都用一个共同的缺陷,只能解决紧密嵌套循环的划分问题而无法解决非紧密嵌套循环序列的划分.

(2) 语句级的划分技术. Shih 等^[11] 提出了一种语句级的基于超平面的无通信计算和数据划分方法.他们的算法在进行划分计算时将每个语句当作一个调度单位,即每个语句都有一个独立的迭代空间,并分别考虑它们的划分.由于其方法是基于超平面的,所以也存在超平面计算划分并行度有限的问题. Lim^[12-14] 提出了一种最大化并行和最小化通信的仿射划分方法,因为他的算法只对计算进行了划分,而未考虑数据的划分,因此他的算法只适用于共享主存多处理机系统.同时 Lim 在文献[14]中试图解决数据划分一致性问题时,使用了邻近通常常数距离的限制条件,但是当不同语句中对同一数组的访问步长不同时,常数距离的限制条件很难使数据划分一致性问题得到解决.

(3) 其他相关划分技术. Beckmann^[17] 和 Kremer^[15] 分别提出了只读数组复制技术.只读数组复制技术通过发现只读数组的复制来满足并行处理中固有的读写操作来提高并行度.但只读数组复制技术存在一些不足,首先只读数组复制技术需要程序中有足够多的只读数组,其次,只读数组复制技术耗费了更多的本地内存,而这项冗余的耗费限制了应用程序的性能. Chen 和 Chang^[16] 针对传统维对准的不足,提出了一种扭曲对准的数组对准策略,从而使得与维对准相比较该方法能处理更复杂的情形.该方法是在迭代空间的划分已确定的基础上进行的,并且

也没有考虑数据复制与偏移常量对准问题,从而求得划分不是通信最小的划分.这些技术虽然有些情况下可以促进划分的效果,但是这些技术都不是独立的划分算法,只有在获得的划分空间已经确定的情况下,才能运用到各种应用程序.

2.2 基本概念

为了更好地描述算法,下面先给出一些后面用到的概念和形式化描述,主要包括仿射划分、程序和关系集合.

2.2.1 仿射划分

仿射分解最早由 Anderson 在文献[1]中提出,是一种非常有效的表示和求解计算划分和数据划分的方法.在仿射算法中,深度为 g 的循环嵌套的每个循环迭代 \vec{i} 到 n 维处理器具体处理机的映射可以表示为 $\vec{C}(\vec{i}) = \vec{C}\vec{i} + \vec{\gamma}$, 其中 \vec{C} 是 $n \times g$ 的线性转换矩阵, $\vec{\gamma}$ 是常数向量. m 维数组的每个下标 $\vec{\alpha}$ 到 n 维处理器具体处理机的映射,可以表示为 $\vec{D}(\vec{\alpha}) = \vec{D}\vec{\alpha} + \vec{\delta}$, 其中 \vec{D} 是 $n \times m$ 的线性转换矩阵, $\vec{\delta}$ 是常数向量.

仿射划分主要应用在循环边界和数组下标是循环变量或常数的仿射函数的程序中.在实际应用中绝大多数程序都满足这个条件,因此仿射划分算法具有十分广泛的应用范畴.本文的算法以仿射划分为基础,在后面讨论中,如不加特殊说明,程序中的数组下标都是满足仿射限制的.

2.2.2 程序

定义 1. 一个串行程序可以表示为 $\mathcal{P} = \langle S, \delta, \mathcal{B}, \mathcal{N}, \mathcal{DVS}, \mathcal{F}, \mathcal{R}, \omega, \eta \rangle$, 其中:

S 是程序 \mathcal{P} 的语句集合,在程序 \mathcal{P} 中语句 s 先于语句 s' 执行可表示 $s <_p s'$;

δ_s 是语句 s 外层循环的层数;

$\mathcal{B}_s(\vec{i}) = \mathcal{B}_s(\vec{i}) + \mathcal{b}_s$ 是 \vec{i} 的仿射函数,如果 \vec{i} 是一个有效的循环迭代,当且仅当 $\mathcal{B}_s(\vec{i}) \geq 0$;

\mathcal{N}_{sz} 是语句 s 对数组 z 的引用次数;

\mathcal{DVS}_{zsm} 是语句 s 对数组 z 的第 m 个引用与程序中对该数组其他引用的依赖方向向量集;

$\mathcal{F}_{zsm}(\vec{i}) = \mathcal{F}_{zsm}(\vec{i}) + \mathcal{f}_{zsm}$ 是语句 s 对数组 z 的第 m 个引用的仿射函数,映射第 \vec{i} 个循环迭代到数组 z 具体的下标;

\mathcal{R}_{zsm} 为真,当且仅当语句 s 中对数组 z 的第 m 个引用为读操作;

\mathcal{W}_{zsm} 为真,当且仅当语句 s 中对数组 z 的第 m 个引用为写操作;

$\eta_{ss'}$ 是语句 s 和 s' 公共循环的层数.

定义 2. 程序 \mathcal{P} 中 g 层嵌套循环中语句 s 的计算划分可以表示为 $\vec{C}_s(\vec{i}) = \vec{C}_s\vec{i} + \vec{\gamma}_s, 0 \leq k \leq g-1$, $\vec{C}_{s-k}(\vec{i})$ 是 $\vec{C}_s(\vec{i})$ 第 k 维划分分量.

定义 3. 程序 \mathcal{P} 中 n 维数组在语句 z 中的数据划分可以表示为 $\vec{D}_{zs}(\vec{i}) = \vec{D}_{zs}\vec{i} + \vec{\delta}_{zs}$.

2.2.3 关系集

定义 4. 两个循环迭代在串行程序中的执行顺序可以表示为 $<$ 关系,即

$$\vec{i} <_{ss'} \vec{i}' \equiv \begin{cases} \exists m \leq \eta_{ss'}, m \geq 0, \vec{i}_{0:m} = \vec{i}'_{0:m} \wedge \vec{i}_{m+1} < \vec{i}'_{m+1}, \eta_{ss'} > 0 \\ s <_{ps'} s', \eta_{ss'} = 0 \end{cases}$$

定义 5. 程序中的数据依赖集 R 可以表示为

$$R = \{ \langle \mathcal{F}_{zs}, \mathcal{F}_{zs'r'} \rangle \mid (\mathcal{W}_{zs} \vee \mathcal{W}_{zs'r'}) \wedge (\exists \vec{i} \in \mathcal{Z}^{\delta_s}, \vec{i}' \in \mathcal{Z}^{\delta_{s'}} \mid (\vec{i} <_{ss'} \vec{i}') \wedge (\mathcal{B}_s(\vec{i}) \geq 0 \wedge \mathcal{B}_{s'}(\vec{i}') \geq 0) \wedge (\mathcal{F}_{zs}(\vec{i}) - \mathcal{F}_{zs'r'}(\vec{i}') = \vec{0}) \}$$

3 代表元

当程序循环序列中有非紧密嵌套循环、不同语句对数组元素访问步长不同、一条语句对数组的多次引用间存在重叠时,程序不存在无通信的划分方向.基于第 1 节的分析,对于这类程序,可以使用程序中数组引用的代表元素对程序进行划分,因此在对这类程序进行划分前,应该先确定程序中哪些数组引用作为代表元素,把原始程序转换为每条语句只包含代表元引用的程序,并把代表元程序计算划分和数据划分的解作为原程序对应计算划分和数据划分方向的解.在本节中,我们给出代表元的定义,代表元程序的定义以及如何选择代表元和构造代表元程序.

3.1 定义

定义 6. 代表元 (Representative Elements, RE) 是指程序中参与计算划分和数据划分计算的数组引用.非代表元 (Un-Representative Elements, URE) 是指程序中对计算划分和数据划分不产生实际影响,即不参与计算划分和数据划分计算的数组元素的引用.

定义 7. 代表元程序 (r -Program or r -P) 是指程序经过变换,每条语句只包含用来构造计算划分和数据划分限制条件的数组引用.代表元程序中 $\mathcal{F}, \mathcal{R}, \omega$ 几个特性用以下符号描述:

$\mathcal{F}_{zsr_r}(\vec{i}) = \mathcal{F}_{zsr_r}(\vec{i}) + \mathcal{f}_{zsr_r}$ 是在代表元程序中,语句 s 对数组 z 的第 r 次应用;

$\mathcal{W}_{zs r_r}$ 为真, 当且仅当语句 s 中对数组 z 的第 r 个代表元的引用为写操作;

$\mathcal{R}_{zs r_r}$ 为真, 当且仅当语句 s 中对数组 z 的第 r 个代表元的引用为读操作.

定义 8. 没经过代表元转换的原始程序在本文中也叫做完全代表元程序 (total r -P 简称为 tr -P), 即程序中对数组的所有引用都作为代表元的代表元程序.

定义 9. 代表元程序中的数据依赖关系集 R_r 可以表示为

$$\begin{aligned} R_r = \{ \langle \mathcal{F}_{zs r_r}, \mathcal{F}_{zs' r'_r} \rangle \mid (\mathcal{W}_{zs r_r} \vee \mathcal{W}_{zs' r'_r}) \wedge \\ (\exists \vec{i} \in \mathcal{Z}^{\delta_s}, \vec{i}' \in \mathcal{Z}^{\delta_{s'}} \mid (\vec{i} <_{ss'} \vec{i}') \wedge \\ (\mathcal{B}_s(\vec{i}) \geq \vec{0} \wedge \mathcal{B}_{s'}(\vec{i}') \geq \vec{0}) \wedge \\ (\mathcal{F}_{zs r_r}(\vec{i}) - \mathcal{F}_{zs' r'_r}(\vec{i}') = \vec{0}) \}. \end{aligned}$$

3.2 代表元程序

构造代表元程序的过程就是如何选取代表元的过程. 如果一条语句对数组的多次引用存在重叠, 造成该语句不存在无通信的划分方向, 该数组的多个引用应沿相同的数据划分方向以保证无数据重组织通信的存在. 在这种情况下, 我们可以使用其中的一个作为代表元来完成该数组数据划分方向的确, 而对于不同语句, 为了消除划分过程中的数据重组织通信, 只需保持不同语句中相同数组的代表元素确定的数据划分方向相同, 而不用考虑不同步长对数据划分造成的影响.

为了保证代表元程序能获得正确的计算划分方向, 必须保证代表元程序中保留原程序的所有相关性信息, 因此对于为数据划分选取的代表元素需做出调整. 如一条语句中两个数组的引用的 DVS 相等, 则其中一个对数组的引用对确定计算划分方向是冗余的, 因此在为计算划分的计算调整代表元程序的过程中, 我们只需在为数据划分确定的代表元程序中增加那些对原程序相关信息产生实际影响的数组引用. 基于这两点原则, 下面给出代表元程序的构造过程:

首先, 根据数据划分的关系初始化程序中每条语句 s 的代表元集合 RES_s (Representative Element Set). 该语句对于数组的写引用属于 RES_s , 对于同一数组的多个读引用, 先随机选择一个读引用作为代表元加入 RES_s 中. 此时, 未加入 RES_s 的数组元素的引用属于非代表元集合 $URES_s$ (Un-Representative Element Set).

其次, 为了保证代表元程序所计算的计算划分的正确性, 根据原始程序的依赖关系, 对每条语句的 RES_s 进行调整, 以使代表元程序的依赖关系与原始

程序保持一致. 对于属于 $URES_s$ 的非代表元素, 测试它与 RES_s 中的同一数组的每一个读引用的 DVS 之间的关系, 如果不存在相同的 DVS , 则把该引用加入到 RES_s 中, 否则丢弃该读引用. 重复该过程, 直至 $URES_s$ 为空. 构造代表元程序的算法, 如图 2 所示.

```

Algorithm constructing- $r$ -P( $P$ )
//  $P$  is the program that would be optimized
for each  $s \in S$  in  $P$  {
     $RES_s = \text{NULL}$ 
     $URES_s = \text{all array references in instruction } s$ 
}
for each  $s \in S$  in  $P$  do {
    for each  $0 \leq e \leq N_{sz}$  do {
        if  $\mathcal{W}_{zse} = 1$  then
             $RES_s = RES_s + F_{zse}$ ;
        else
            if  $(\exists F_{zse} \in RES_s) \& (\exists DVS_{zsk} = DVS_{zse})$  then
                continue;
            else {
                 $URES_s = URES_s - F_{zse}$ ;
                 $RES_s = RES_s + F_{zse}$ ;
            }
        } //end foreach
    } //end foreach
}

```

图 2 构造 r -P 算法

4 基于代表元程序的仿射划分算法

4.1 r -P 的仿射划分

对于代表元程序的一个具体循环嵌套, 当循环中一条语句的具体迭代与它访问的数组元素被划分到同一处理机时, 没有通信发生. 当没有通信发生时, 两条访问同一数组元素的不同语句的循环迭代应该被划分到同一处理机上, 基于该条件我们可以使用式(1)来形式化表示语句间的计算划分 (Computation affine Partition Constraint for r -P, CPC); 当没有通信发生时, 数组的每个元素应该与访问它的语句的对应迭代划分到同一处理机上, 基于此条件我们可以使用式(2)来形式化表示数据划分 (Data affine Partition Constraint for r -P, DPC); 式(1)和式(2)合称为代表元程序的划分限制条件 (Computation and Data affine Partition Constraint for r -P, CDPC). 对于完全代表元程序 tr -P 程序, 对应的式(1)简称为 Total-CPC, 即 TCPC, 同理另外两个限制条件简称为 TDPC 和 TCDPC.

$$\forall \langle \mathcal{F}_{zs r_r}, \mathcal{F}_{zs' r'_r} \rangle \in R_r, \vec{i} \in \mathcal{Z}^{\delta_s}, \vec{i}' \in \mathcal{Z}^{\delta_{s'}}$$

$$\text{s. t. } \mathcal{B}_s(\vec{i}) \geq \vec{0} \wedge \mathcal{B}_{s'}(\vec{i}') \geq \vec{0} \wedge$$

$$\mathcal{F}_{zs r_r}(\vec{i}) - \mathcal{F}_{zs' r'_r}(\vec{i}') = 0, \vec{\mathcal{C}}_s(\vec{i}) - \vec{\mathcal{C}}_{s'}(\vec{i}') = \vec{0} \quad (1)$$

$$\vec{\mathcal{d}}_{zs}(\mathcal{F}_{zs r_r}(\vec{i})) = \vec{\mathcal{C}}_s(\vec{i}) \quad (2)$$

对于式(1),我们可是使用式(3)来进行简化. 其中 $\mathcal{X}(\vec{i}) = \mathbf{X}(\vec{i}) + x$, \mathbf{X} 是 $1 \times g$ 的一维向量, x 为常数.

$$\forall \langle \mathcal{F}_{zs r_r}, \mathcal{F}_{zs' r'_r} \rangle \in R_r, \vec{i} \in Z^{\delta_s}, \vec{i}' \in Z^{\delta_{s'}}$$

$$\text{s. t. } \mathcal{B}_s(\vec{i}) \geq \vec{0} \wedge \mathcal{B}_{s'}(\vec{i}') \geq \vec{0} \wedge$$

$$\mathcal{F}_{zs r_r}(\vec{i}) - \mathcal{F}_{zs' r'_r}(\vec{i}') = 0, \mathcal{X}(\vec{i}) - \mathcal{X}'(\vec{i}') = 0 \quad (3)$$

定理 1. 对于式(1)中语句 s 的每一维计算划分向量 \mathbf{C}_s , 应与式(3)中的 \mathbf{X} 具有相同的解空间, 或式(3)的解空间是式(1)各维的解空间.

证明. 假设 g 是 \mathbf{C}_s 的维数, g' 是 $\mathbf{C}_{s'}$ 的维数 $\forall k, 0 \leq k \leq g-1, \forall k', 0 \leq k' \leq g'-1, \vec{\mathcal{C}}_{s-k}(\vec{i}), \vec{\mathcal{C}}_{s'-k'}(\vec{i}')$ 应该满足式(2)

$$\forall \langle \mathcal{F}_{zs r_r}, \mathcal{F}_{zs' r'_r} \rangle \in R_r, \vec{i} \in Z^{\delta_s}, \vec{i}' \in Z^{\delta_{s'}}$$

$$\text{s. t. } \mathcal{B}_s(\vec{i}) \geq \vec{0} \wedge \mathcal{B}_{s'}(\vec{i}') \geq \vec{0} \wedge$$

$$\mathcal{F}_{zs r_r}(\vec{i}) - \mathcal{F}_{zs' r'_r}(\vec{i}') = 0, \vec{\mathcal{C}}_{s-k}(\vec{i}) - \vec{\mathcal{C}}_{s'-k'}(\vec{i}') = 0 \quad (4)$$

因此, $\vec{\mathcal{C}}_{s-k}(\vec{i}), \vec{\mathcal{C}}_{s'-k'}(\vec{i}')$ 应该是式(3)的解, 所以

$$\forall \langle \mathcal{F}_{zs r_r}, \mathcal{F}_{zs' r'_r} \rangle \in R_r, \vec{i} \in Z^{\delta_s}, \vec{i}' \in Z^{\delta_{s'}}$$

$$\text{s. t. } \mathcal{B}_s(\vec{i}) \geq \vec{0} \wedge \mathcal{B}_{s'}(\vec{i}') \geq \vec{0} \wedge$$

$$\mathcal{F}_{zs r_r}(\vec{i}) - \mathcal{F}_{zs' r'_r}(\vec{i}') = 0, \mathcal{X}(\vec{i}) - \mathcal{X}'(\vec{i}') = 0$$

证毕.

式(1), (3)给出了代表元程序无通信计算划分的限定条件, 式(2)给出了对于代表元程序无通信数据划分的限定条件. 采用与文献[1]中类似的方法^①, 如果指令的两个循环迭代的实例 $\vec{i}_1 \in Z^{\delta_s}, \vec{i}_2 \in Z^{\delta_s}$ 被映射到同一处理机上, 则他们访问的数组 z 的相同元素也将被映射到同一处理机上. 对于计算划分, 由于指令 s 两个迭代被映射到同一处理机上, 可以得到 $\vec{\mathcal{C}}_s(\vec{i}_1) = \vec{\mathcal{C}}_s(\vec{i}_2)$ 即 $\mathbf{C}_s(\vec{i}_1 - \vec{i}_2) = \vec{0}$, 不妨设 $\vec{i} = \vec{i}_1 - \vec{i}_2$, 则 $\vec{i} \in \ker \mathbf{C}_s$. 对于数据划分, 可以得到 $\vec{d}_{zs}(\mathcal{F}_{zs r_r}(\vec{i}_1)) = \vec{d}_{zs}(\mathcal{F}_{zs r_r}(\vec{i}_2))$ 和 $\vec{d}_{zs}(\mathcal{F}_{zs r_r}(\vec{i}_1 - \vec{i}_2)) = \vec{0}$. 根据式(2), $\mathbf{D}_{zs} \mathbf{F}_{zs r_r} \vec{i} = \mathbf{C}_s \vec{i}$, 因此 $\mathbf{F}_{zs r_r} \vec{i} \in \ker \mathbf{D}_{zs}$. 根据上述分析可以得到如下关系:

$$\ker \mathbf{D}_{zs} \supseteq \text{span}\{\vec{s} \mid \vec{s} = \mathbf{F}_{zs r_r} \vec{i}, \vec{i} \in \ker \mathbf{C}_s\} \quad (5)$$

同理, 对于 $\vec{i} \in \ker \mathbf{C}_s$, 当 $\mathbf{F}_{zs r_r} \vec{i} \in \ker \mathbf{D}_{zs}$, 不妨设 $M_{zs r_r} = \text{range}(\mathbf{F}_{zs r_r})$, 则有如下关系:

$$\ker \mathbf{C}_s \supseteq \text{span}\{\vec{i} \mid \mathbf{F}_{zs r_r} \vec{i} \in (\ker \mathbf{D}_{zs} \cap M_{zs r_r})\} \quad (6)$$

式(3)可以通过使用文献[19]中介绍的方法化简为一系列线性等式. Lim 在文献[12]中给出了式(3)的详细解法. 因此我们可以使用文献[12]中的方法得到每条语句的计算划分的解空间, 在得到各语句计算划分的解空间后, 根据关系(5)则可得各

语句访问数组的数据划分的解空间. 图 3 给出该算法的具体过程.

```

Algorithm calcul_computa_partition(P) {
  //P is the program that would be partitioned
  S-CDPCp = true
  for each s ∈ S in P do {
    simplify equation using Affine Form of the Farkas Lemma[19]
    compute Cs through algorithm in [12]
    compute kerCs
    if Cs = NULL then S-CDPCp = false
  }
}

Algorithm calcul_data_partition(P) {
  for each s ∈ S in P do {
    kerDzs = NULL
  }
  for each s ∈ S in P do {
    for each 0 ≤ r ≤ Nzs r do {
      kerDzs = kerDzs + span{ $\vec{s} \mid \vec{s} = \mathbf{F}_{zs r_r} \vec{i}, \vec{i} \in \ker \mathbf{C}_s$ }
      if Dzs = NULL then S-CDPCp = false
    }
  }
}

```

图 3 CDPC 算法

4.2 一致划分算法

对于程序的数据相关图, 因为其中强联通图 (Strongly Connected Component, SCC) 的各个节点至少有一层公用循环^[12]. 当程序只包含一个 SCC 时, 如果 CDPC 算法对于 $tr-P$ 无解, 即对于原始程序不存在无通信的划分方向, 则可以通过对代表元程序使用 CDPC 算法对 $r-P$ 程序求解, 来决定程序的数据划分和计算划分. 如果 CDPC 算法求得的划分解空间为空, 则先使用文献[12]中介绍的时间划分对 SCC 进行划分, 之后再对划分过的 SCC 使用 CDPC 求解.

如果程序中含有多个 SCC, 则避免 SCC 间数据重组织通信对划分后程序的性能将产生至关重要的影响. 当在一个程序中数组的访问步长都相同时, 可以使用文献[14]中计算划分的常数距离方法来解决, 但是一方面在很多程序中, 对于同一数组的访问步长不同, 此时无法找到常数距离, 另一方面, 对于分布主存多处理机, 减少数据重组织通信最重要的还是保证同一数组在不同 SCC 中的数据划分沿相同的数据划分方向. 当程序中的多个 SCC 间不存在数据重组织通信时, 同一数组在不同 SCC 中的数据划分方向相同, 即同一数组在不同 SCC 各语句中的

① 文献[1]中虽然给出了计算分解与数据分解的关系, 但这些关系不是针对语句级划分的, 且推导前, 先舍弃了划分关系中代表偏移的常数项, 因此本文在此处重新给出推导过程. 同时由于一致性数据划分问题, 实际上还是涉及过程间的数组访问, 因此本文使用文献[18]中的过程间分析算法进行过程间分析.

划分方向是对齐的,它们的数据划分应使用相同的分解矩阵 D_z . 因此它们应该满足如下限制式(Data Consistent Constraint,DCC):

$$\begin{aligned} & \forall s, s' \text{ in different SCCs} \\ & \forall \langle \mathcal{F}_{zs_{r_r}}, \mathcal{F}_{zs'_{r'_r}} \rangle \in R_r, \vec{i} \in Z^{\delta_s}, \vec{i}' \in Z^{\delta_{s'}} \\ & \text{s. t. } \mathcal{B}_s(\vec{i}) \geq \vec{0} \wedge \mathcal{B}_{s'}(\vec{i}') \geq \vec{0} \wedge \\ & \quad \mathcal{F}_{zs_{r_r}}(\vec{i}) - \mathcal{F}_{zs'_{r'_r}}(\vec{i}') = \vec{0}, \\ & \quad \mathbf{D}_z(\mathcal{F}_{zs_{r_r}}(\vec{i})) - \mathbf{D}_z(\mathcal{F}_{zs'_{r'_r}}(\vec{i}')) = \vec{0} \quad (7) \end{aligned}$$

根据式(7),在求解多个 SCC 中同一数组的一致性划分方向时,可以应用 CDPC 算法先求解各 SCC 中每一条访问数组 z 的语句 s 的计算划分,之后根据这些语句的计算划分方向,计算数组 z 满足

所有访问它的语句的公共划分方向. 根据式(5)我们可以得到如下关系:

$$\forall s, \text{ if } \mathcal{N}_{zs} > 0 \text{ then}$$

$$\ker D_z \supseteq \text{span}\{\vec{s} \mid \vec{s} = \mathbf{F}_{zs_{r_r}} \vec{i}, \vec{i} \in \ker C_s\} \quad (8)$$

如果使用关系(8)求得数组 z 的公共划分方向的解空间不为空,则基于式(6),使用新求得的数据划分解重新计算各语句的计算划分. 具体描述如下:

$$\forall z, \text{ if } ((\mathcal{N}_{zs} > 0) \wedge (\ker D_z \neq \text{NULL})) \text{ then}$$

$$\ker C_s \supseteq \text{span}\{\vec{i} \mid \mathbf{F}_{zs_{r_r}} \vec{i} \in (\ker D_z \cap M_{zs_{r_r}})\} \quad (9)$$

因为寻找程序的最优划分方向是 NP-难问题^[1], 下面我们给出了基于关系(8)和(9)启发式的一致性划分算法,具体算法如图 4 所示.

```

Algorithm test_consistent(P) {
  for each array z accessed in P do {
    kerDz = NULL
    for each SCCi in P do
      for each instruction s ∈ SCCi do
        for each 0 ≤ r ≤ Rsz do
          kerDzs = kerDzs + span{  $\vec{s} \mid \vec{s} = \mathbf{F}_{zs_{r_r}} \vec{i}, \vec{i} \in \ker C_s$  }
        compute Dz
        if Dz = NULL then
          return S-DPCp = 0
      }
    for each SCCi in P do
      for each instruction s ∈ SCCi do {
        kerCs = NULL
        for each array z accessed in s do
          kerCs ⊇ span{  $\vec{i} \mid \mathbf{F}_{zs_{r_r}} \vec{i} \in (\ker D_{zs} \cap M_{zs_{r_r}})$  }
        compute Cs
        if Cs = NULL then
          return S-DPCp = NULL
      }
  }
}

void time_partition(program)
{
  partition a loop with time partition in [13];
  return is the program sequence of SCCs after time partition
}

Algorithm consis_partition(P) {
  //P is the program that would be partitioned
  //TN: the SCC account in topological order of a program dependence graph
  //SCCi: the i-th SCC element in topological order of a program dependence graph
  //S-DPCp: is 1 when there is consistent data partition for P of program P
  //S-TDPCp: is 1 when there is consistent data partition for program P
  MLPSS = NULL;
  for each 0 ≤ i ≤ TN do {
    T-MLPSS = MLPSS + SCCi
    test_consistent(T-MLPSS)
    if (S-DPCT-MLPSS = NULL) {
      if (MLPSS != NULL)
        if (S-TDPCMLPSS != NULL)
          partition MLPSS with T-CDPC
        else
          partition MLPSS with CDPC
      MLPSS = NULL;
      if (S-CDPCSCCi = NULL) {
        PSCC = time_partition(SCCi)
        consis_partition(PSCC)
      }
      else
        MLPSS = SCCi
    }
  }
  else
    MLPSS = T-MLPSS
  } //end if
} //end for
} //end algorithm

```

图 4 一致性划分算法

基于代表元的一致性划分算法主要基于两个基本原则：首先，最小化数据重组通信是本算法的主要目标。如果两个 SCC 各自有无通信划分方向，但是当把两个 SCC 一起考虑时，他们之间不存在无通信的划分方向，但有无数据重组通信的数据划分方向。对于这种情况，在一致性划分算法中优先选择后者作为数据划分方向，因为后者可以完全消除各 SCC 数据划分后的数据重组通信，而选择各自无通信的划分方向则不能保证可以消除不同 SCC 间的数据重组通信。其次，只有当完全代表元程序（原始程序）不存在无通信一致性划分方向时，才使用代表元程序计算一致性划分方向，因为基于代表元程序计算的一致性划分方向，只能保证无数据重组通信，而不能保证求得的划分方向是无通信的。

在一致性划分算法中，我们通过对数据相关图各 SCC 的拓扑序构造最大一致划分 SCC 集合 (MLPSS) 来解决数据划分一致性问题。该算法主要分为两个步骤：

1. MPLSS 为空， SCC_i 指向拓扑序中的当前 SCC (初始时当前节点是拓扑序的第一个节点)；

2. MLPSS 与 SCC_i 组成 T -MLPSS (临时 MLPSS)，使用 *test_consistent* 测试 T -MLPSS 中各数组是否存在一致划分方向。如果存在一致性划分方向，则 SCC_i 加入到 MLPSS 中， SCC_i 指向拓扑序中下一个 SCC，重复步 2。如果 T -MLPSS 不存在一致性划分方向，则使用 MLPSS 的一致性划分方向对 MLPSS 的各 SCC 进行划分，之后进入步 1。

在基于代表元的一致性划分算法中，函数 *test_consistent* 主要测试 T -MLPSS 中各 SCC 是否存在一致的数据划分方向，分为两步：首先，使用 CDPC 算法计算各 SCC 中语句的计算划分解空间。之后基于关系 (8) 计算各 SCC 公共访问数组 z 的数据划分方向的零空间 $kerD_z$ 。其次，如果多 SCC 公共访问的数组存在一致的数据划分方向，再根据关系 (9) 使用多 SCC 中各公共访问数组的 $kerD_z$ 对各 SCC 中引用该数组语句的计算划分方向进行重新计算，重复这两个步骤，直至 T -MLPSS 中所有数组的 $kerD_z$ 和所有语句的 $kerC_s$ 不再发生变化。如果重新计算获得的 T -MLPSS 所有语句计算划分的解空间和所有公共访问数组的数据划分的解空间都不为空，则 T -MLPSS 返回值为真；否则为假。

4.3 算法示例

为了说明基于代表元的一致性划分算法，我们使用图 1 中给出的程序作为例子来说明该算法

的工作过程。该程序有两个 SCC，且对于原始程序，两个 SCC 都不存在无通信的划分方向，因此我们首先构造该程序的 r -P。构造后的 r -P 如图 5 所示。

```

for i=1 to 100 do
  for j=1 to 100 do
    for k=1 to 100 do
      A[i,j,k]=A[i,j-1,k]+B[2i,j,2k-1];
    for i=1 to 100 do
      for j=1 to 100 do
        C[i,j,1]=B[0,j,0]
        for k=1 to 100 do
          B[i,j,k]=C[i,j,1] * B[i-1,j,k]+A[i,2j-1,2k];
        
```

图 5 图 1 程序的代表元程序

对于该段程序的代表元程序，当 MLPSS 只包含 SCC1 时，使用 CDPC 算法求得计算划分的划分方向的解空间是 (1, 0, 0) 和 (0, 0, 1)，而数组 A, B 数据划分方向的解空间是 (1, 0, 0) 和 (0, 0, 1)。当 MLPSS 包含 SCC1 和 SCC2 时，此时程序不存在无通信的划分方向，使用 *test_consistent* 计算的数组 A, B 数据划分的划分方向的解空间是 (0, 0, 1)，语句 S1 和 S2 计算划分方向的解空间分别是 (0, 0, 1)。

5 实验结果

为了说明算法的效果，我们在复旦大学并行处理研究所开发的 AFT2004 中实现了该算法，并使用 Mgrid 程序作为测试程序对算法进行了测试。

Mgrid (多重网格 multigrid) 是十分典型的测试程序，SPEC2000, NASA 等测试包中都包含该程序。Mgrid 程序与图 1 中程序的特性十分相似，在 Mgrid 程序中存在许多非紧密嵌套循环，不同语句对数组的访问步长不同，且许多语句都有对非只读数组的多个引用，而多个引用之间存在的数组访问重叠使这些语句不存在无通信的划分方向。因此，以前的各种划分算法很难解决该类程序的划分问题，虽然针对 Mgrid 程序的手工优化中存在一些经验性的划分策略，但这些策略无法应用到编译器中来完成程序的自动划分。我们选择 mgrid 作为测试程序，主要基于以下两方面考虑：

(1) 从理论上来说，多重网格算法是求解大规模科学与工程计算问题最有效的方法之一。该算法由前苏联计算数学家 Fedorenko 在 1961 年基于差分法提出的，20 世纪 70 年代中期开始受到普遍的重视，被认为是一种行之有效的数值分析方法。20 世

纪 80 年代以后,世界各国的计算数学家在这方面投入了大量的人力、物力,使得这一方法在理论研究中取得了很多重要成果,并在实际应用中也显示了强大的生命力。

(2) 从具体应用来说, <http://www.mgnet.org/> 列出了许多以基于多重网格的应用. 虽然这些应用不同,但是这些应用的核心(即整个程序中最耗费计算时间的部分)都是多重网格算法,因此我们在实际程序的测试效果中只是使用 mgrid 程序作为测试程序,而没有进一步测试这些程序. 因为做为这些应用的核心算法, mgrid 的划分效果一方面可以说明问题,另一方面,选择过多的同类程序,也容易造成不必要的混淆。

测试环境是复旦大学高性能计算中心的 HP 64 节点 ProLiant DL360 G3 Beowulf 机群系统. 每个节点是 1 个 3.06GHz 的 Intel 置强处理器,每个处理器有 512KB 的片上 Cache,每个节点本地 2GB 主存. 互联网络是 Myrinet PCI NIC, Myrinet 8 Port Fibre Switch 和 HP Procurve Switch. 操作系统是 Red Hat Linux 8.0,并行编译环境是实现了该算法 AFT2004. 为了说明自动算法的划分效果,我们主要与两组手动划分的数据进行了比较. 一组手动数据是我们不做进一步通信优化手动划分的算法,实验数据显示两者针对不同节点数的实验数据效果基本一致. 为了进一步说明自动划分算法的效果,我们也与文献[20]中对通信性能进行过优化的手工划分版本进行了性能比较. 文献[20]在研究机群通信性能优化过程中,使用 Mgrid 作为测试程序,程序经过手工划分后,对程序的通信部分进行了特定的优化. 为了说明使用代表元算法的自动划分效果,我们使用我们自动划分算法的实验结果与文献[20]中的经过通信优化的手工划分程序的实验结果进行了比较. 在 Mgrid 程序中, LMI 是程序计算中数组的每一维大小的参数. 我们分别测试了 $LMI=64$ 和 $LMI=128$ 两组数据. 实验结果如图 6 和图 7 所示(其中, rp 为代表元自动划分的测试结果, nps 为文献[20]程序的测试结果)。

从图 6 和图 7 可以看出,基于代表元的一致性划分算法相对于串行程序的加速比随着处理器数目的增加而增加,其原因是随着处理器的增加,程序可以获得的并行性越来越大. 从图中的性能数据趋势可以看出,自动划分算法的执行效率,随着机器数目的增多,与经过优化的手工算法的增长趋势一致,因此说明自动划分算法可以有效解决多重网格类程序的自动划分问题。

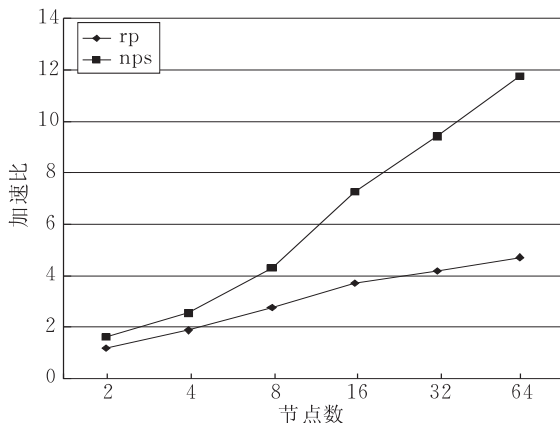


图 6 一致性划分算法的效率($LMI=64$)

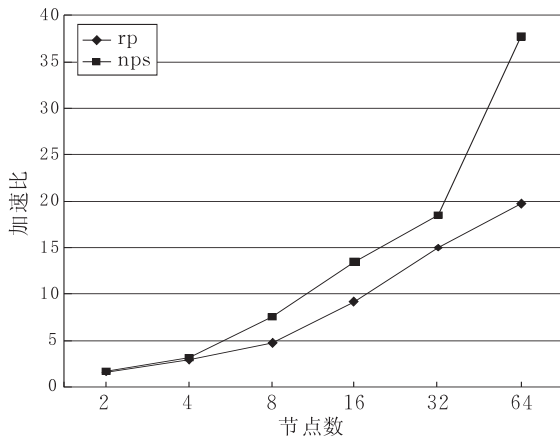


图 7 一致性划分算法的效率($LMI=128$)

从图 6 和图 7 中也可以看出,自动划分算法与手工划分算法之间存在一定的性能差距. 而性能差距在处理器节点较少的时候很低,而随着处理器数目的增加,性能差距越来越大. 造成代表元划分算法与文献[20]中手工划分程序性能差距的主要原因是,文献[20]中在对 Mgrid 进行划分时,专门根据程序的特点,进行了通信部分的优化. 而在 Mgrid 程序中,不存在无通信的划分方向,在采用代表元程序一致性划分算法进行划分后,当节点较少时,划分后存在的边界通信由于分块(blocking)而被大大降低,在处理器较少时通信所占的比重还比较小,因此,在处理器较少的时候,自动划分和文献[20]中手工划分对应用程序的加速比是十分接近的. 随着处理机的增加,通信开销所占的比例越来越大,而文献[20]中手工修改程序版本对划分后的程序进行了专门通信的优化,因此,对于应用程序可以获得更高的加速比。

6 结 论

本文针对应用程序中存在的非紧密嵌套循环,

不同语句中对数组的访问跨度不同,且不存在无通信的划分方向的程序,提出了一种基于代表元的一致性划分算法.该算法给出了基于代表元程序无通信划分时的计算划分和数据划分的数学描述和求解,并给出了有效消除程序中数据重组通信的一致性划分算法.该算法在 AFT2004 并行编译系统中实现,并对应用程序获得了很好的效果.

虽然本文提出的一致性划分算法可以有效解决多重网格类程序的自动划分问题,但是与经过专门通信优化的手工版本也存在一定的性能差距,因此在进一步的工作中,我们将深入分析多重网格类程序的通信优化,以便对一致性自动划分算法进行进一步优化,提高一致性划分算法的效率.

参 考 文 献

- [1] Anderson J M, Lam M S. Global optimizations for parallelism and locality on scalable parallel machines//Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation. Albuquerque, New Mexico, USA, 1993: 112-125
- [2] Banerjee U. Unimodular transformations of double loops//Proceedings of the 3rd Workshop on Languages and Compilers for Parallel Computing. Irvine, California, USA, 1990: 192-219
- [3] Wolf M E, Lam M S. A loop transformation theory and an algorithm to maximize parallelism. IEEE Transactions on Parallel and Distributed Systems, 1991, 2(4): 452-470
- [4] Wolf M E. Improving locality and parallelism in nested loops [Ph. D. dissertation]. Stanford University, CSL_TR-92-538, 1992
- [5] Anderson J M, Amarasinghe S P, Lam M S. Data and computation transformations for multiprocessors//Proceedings of the 5th ACM/SIGPLAN Symposium on Principles and Practice of Parallel Programming. Santa Barbara, California, USA, 1995: 166-178
- [6] Anderson J. Automatic computation and data decomposition for multiprocessors[Ph. D. dissertation]. Stanford University, Standford, CA, 1997
- [7] Bau D, Kidukula I et al. Solving alignment using elementary linear algebra//Proceedings of the 7th Workshop on Languages and Compilers for Parallel Computing. Ithaca, NY, USA, 1994: 46-60
- [8] Ramanujam J, Sadayappan P. Compile-time techniques for data distribution in distributed memory machines. IEEE Transactions on Parallel and Distributed Systems, 1991, 2(4): 472-482
- [9] Huang C H, Sadayappan P. Communication-free hyperplane partitioning of nested loops. Journal of Parallel and Distributed Computing, 1993, 19(2): 90-102
- [10] Shih K-P, Sheu J P, Huang C H. Statement level communication free partitioning techniques for parallelizing compilers//Proceedings of the 9th Workshop on Languages and Compilers for Parallel Computing. Minneapolis, Minnesota, USA, 1997: 389-403
- [11] Shih K P, Sheu J P, Huang C H. Statement-level communication-free partitioning technique for parallelizing compilers. The Journal of Supercomputing, 2000, 15(3): 243-269
- [12] Lim A W, Lam M S. Maximizing parallelism and minimizing synchronization with affine transforms//Proceedings of the Conference Record of the 24th ACM SIGPLAN/SIGACT Symposium on Principles of Programming Languages. Paris, France, 1997: 201-214
- [13] Lim A W, Lam M S. Maximizing parallelism and minimizing synchronization with affine partitions. Parallel Computing, 1998, 24(3-4): 445-475
- [14] Lim A W, Cheong G I, Lam M S. An affine partitioning algorithm to maximize parallelism and minimize communication//Proceedings of the 13th ACM-SIGARCH International Conference on Supercomputing. Rhodes, Greece, 1999: 228-237
- [15] Kremer Ulrich. Automatic data layout with read-only replication and memory constraints//Proceedings of the 10th International Workshop on Languages and Compilers for Parallel Computing. Chapel Hill, NC, USA, 1998: 419-422
- [16] Chen T S, Chang C Y. Skewed data partition and alignment techniques for compiling programs on distributed memory multicomputers. The Journal of Supercomputing, 2002, 21(2): 191-211
- [17] Beckmann Olav, Jpaul H. A linear algebra formulation for optimizing replication in data parallel programs//Proceedings of the 12th International Workshop on Languages and Compilers for Parallel Computing. Youktown Heights, NY, USA, 2000: 100-116
- [18] Zang Bin-Yu. Constructing a parallel compiler system; AFT [Ph. D. dissertation]. Fudan University, Shanghai, 1999(in Chinese)
(臧斌宇. 并行化编译系统 AFT 的构造[博士学位论文]. 复旦大学, 上海, 1999)
- [19] Feautrier P. Some efficient solutions to the affine scheduling problem, Part I: One-dimensional time. International Journal of Parallel Programming, 1992, 21(5): 313-348
- [20] Tang Yuan. Research on the performance evaluation and communication optimization of large scale cluster system [Ph. D. dissertation]. Institute of Software, Chinese Academy of Sciences, Beijing, 2004(in Chinese)
(唐渊. 大规模集群系统的性能评价与通信优化研究[博士学位论文]. 中国科学院软件研究所, 北京, 2004)



ZHANG Wei-Hua, born in 1974, Ph. D., lecturer. His main research interests include parallel computing and compiling optimization.

WANG Peng, born in 1979, Ph. D. candidate. His main research interests include parallel compiling and optimization.

ZANG Bin-Yu, born in 1965, professor, Ph. D. supervisor. His current research interests include parallel compiler and computer architecture.

ZHU Chuan-Qi, born in 1943, professor, Ph. D. supervisor. His main research interests include parallel processing and compiling.

Background

This work is supported by Specialized Research Fund for the Doctoral Program of Chinese Higher Education under grant No. 20050246020.

Partition is an optimization technique that distributes computations and data onto different processors of parallel systems to get the maximal usage of parallel systems' computation resources to speedup the programs' processing time. The effect of partition algorithm directly affects the performance of parallel systems. The partition problems have been always the key problem to high performance computing on distributed memory multi-computers which is the mainstream platform of large-scale scientific calculation. The major goal of partition is to get the maximizing parallelism and minimizing communication.

As a research hotspot, there are a lot of researches focusing on partition problems. Some of them deal with perfect loop nest partition, in which it treats all the statements within a loop nest as a single unit. In such a condition, they can only solve the partition for the program where there are only perfect loop nests. Some of them are partition algorithms for statements, in which each statement is the basic partition unit. However, they can not solve the data partition consistent problem when array access stride in different statements are unequal. Some other research is presented to solve read-

only data replication problems, alignment problems and so on.

At present, MGID type algorithms are widely used to solve different problems in different areas. There are imperfect loop nests and different array access strides in such programs. When these patterns appear in programs, partition especially data partition becomes extremely complex. Such characteristics put great obstacles to effective partition since multiple data access strides would lead to partition conflicts based on those prior partition algorithms.

This paper presents an affine partition algorithm based on representative elements. When constructing the constraint relation for partition, it only remains the array references, which have contributed to partition constraint relation indeed and remove the trivial partition conflicts through discarding redundant array references to the same array. This paper also presents a consistent partition algorithm to solve the data partition consistent problem. The new algorithms can not only solve more practical partition problems, but also effectively reduce data reorganization communication in data partition. This technique has been implemented in AFT2004 parallel compiling system and can get better results for some practical programs.