一种基于线性代数的计算和数据自动分解算法*)

韩 林 赵荣彩 董春丽 张 平
(解放军信息工程大学 郑州 450002)

摘 要 在针对分布内存体系结构的并行识别技术中,如何对计算和数据进行合理分解,以增加数据引用的本地化、减少处理器间的通信是提高并行程序性能的关键。本文通过对 Anderson-lam 分解算法完整性的补充,给出了一种可实现无通信的计算划分和数据分布算法,并阐述了对该算法在工程实践中的一些优化考虑。

关键词 并行编译,计算划分,数据分布

An Automatic Computation and Data Decomposition Algorithm Based on Linear Algebra

HAN Lin ZHAO Rong-Cai DONG Chun-Li ZHANG Ping
(The PLA Information and Engineering University, ZhengZhou 450002)

Abstract Increasing the locality of data references and minimizing communication of processors by computation and data decomposition are the key optimization for achieving high performance on large-scale parallel machines. A compiler algorithm that automatically finding computation and data decomposition with no communication based on Anderson-lam algorithm is described, and some improvement on engineering implement is also mentioned in this paper.

Keywords Parallel compilation , Data distribution, Computation partition

1 引言

在分布式存储环境下,各个计算节点拥有自己的存储器,节点间的通信通过消息机制来进行。由于计算节点访问本地存储器的速度远远快于通过网络访问异地存储器的速度,因此在自动并行化过程中如何对串行代码进行正确的分析,找出其中可并行的部分并且进行合理的计算和数据分解,从而达到在本地计算时只需访问本地存储器,即并行的分布和数据的本地化,已成为并行化编译的关键技术。

我们称映射计算到并行机的各个计算节点为"计算划分",将计算所引用的数据存放到各计算节点的本地存储器为"数据分布"。Anderson 和 Lam 提出了一种能够对计算和数据进行自动线性分解的算法。该算法适用于循环嵌套中对数组元素的访问能够被循环索引变量线性表示,即数组的下标是循环索引的仿射函数。算法通过对计算和数据并行化时约束条件的分析给出划分结果。基于对该算法的深入研究,我们发现该算法的约束条件不能够完全满足分解需求,在一些特定的情况下会产生大量的通信,因此对该算法进行了合理的补充。

本文讨论的分解算法适用于分布内存和共享内存体系结构,可应用于并行化编译器进行过程内计算和数据的静态自动分解,使得每一个循环嵌套和每一个数组有唯一的分解结果,结果以矩阵或者不等式的形式给出。在算法分析过程中,我们仅考虑将计算和数据分解到虚拟处理器上,即假定虚拟处理器个数和维数满足分解的要求,而不考虑虚拟处理器与物理处理器间的映射关系。

2 计算和数据分解概述

寻找单循环嵌套内的最大并行性和本地化技术已经比较成熟[1,2],许多研究者也致力于如何有效地将特定的单循环嵌套映射到并行机上[3~5]。我们称针对单循环嵌套的优化为局部分析,而如何进行全局的优化分析,即对程序中出现的多个循环嵌套进行最优的并行性和本地化分析则是一个需要解决的问题。

求解一个好的分解结果是困难的,Kennedy 和 Kremer 表明寻找最佳的数据分布模式的问题是 NP 完全问题[6]。这是因为,首先我们有许多种选择来进行计算和数据的分解,怎样选择最优是面临的问题;其次分解需要在整个程序内进行全局的考虑,并且数据和计算分解是相互关联的,在分解时要考虑到彼此的影响。我们假定程序中的每个循环嵌套已经做了各自的优化,并且应用幺模转换技术寻找到外层的最大并行性[2,7],经过上述处理后,每个循环嵌套规范为完全可置换的形式。算法针对程序中的多个循环嵌套进行全局并行分析,建立一组满足分解条件的约束等式,通过解方程来找到数据分布和计算划分结果。

定义循环迭代空间 L 为一个 l 维空间,用于表示深度为 l 的循环迭代,其空间中的任意一点表示循环嵌套的一次迭代,可用其索引向量 $\vec{i}=(i_1,i_2,\cdots,i_l)$ 来表示;数组下标空间 A 是一个 m 维的空间,用于表示 m 维数组,其空间中的任意一点表示该数组的一个元素,可用向量 $\vec{a}=(a_1,a_2,\cdots,a_m)$ 来表示;处理器空间 P 是一个 n 维的空间,用于表示 n 维的虚拟处理器,其空间中的任意一点表示一个处理器的位置,可用向量 $\vec{p}=(p_1,p_2,\cdots,p_n)$ 表示。

*)国防重点科研项目资助、河南省杰出人才创新基金(0521000200)。韩 林 博士研究生,主要研究方向:计算机软件与并行编译;赵荣彩 教授,博士生导师,主要研究方向:体系结构、先进编译;董春丽 博士研究生,主要研究方向:计算机软件与并行编译;张 平 博士研究生,主要研究方向:计算机软件与并行编译。

定义数组仿射函数 $f(\vec{i})=F\vec{i}+\vec{\beta}$ 表示 l 维循环迭代空间到 m 维数组下标空间的映射，其中 F 是一个 $l*m$ 的线性转换矩阵，$\vec{\beta}$ 是常向量；定义数组分布仿射函数 $d(\vec{a})=D\vec{a}+\vec{\delta}$ 表示 m 维数组下标空间到 n 维处理器空间的映射，其中 D 是一个 $n*m$ 的线性转换矩阵，$\vec{\delta}$ 是常向量。定义计算划分仿射函数 $c(\vec{i})=C\vec{i}+\vec{\gamma}$ 表示 l 维循环迭代空间到 n 维处理器空间的映射，其中 C 是一个 $n*l$ 的线性转换矩阵，$\vec{\gamma}$ 是常向量。算法的目标就是为程序中的每一个循环嵌套和每一个数组寻找到唯一的 $c(\vec{i})$ 和 $d(\vec{a})$ 仿射函数，指示计算和数据到虚拟处理器如何分解。

3 线性分解算法

假定某个循环嵌套有 l 层，其计算划分矩阵为 C，那么该循环嵌套的并行度为 $rank(C)$，并且 $rank(C)=l-\dim(N(C))$，其中 $N(C)$ 为矩阵 C 的核空间。对于数组分解矩阵 D 和计算分解矩阵 C，其核空间在物理上的意义是指需要被分配到同一个处理器上的数据和计算。因此最大化并行就意味着寻找到一个 $rank(C)$ 尽可能大的线性分解结果，也即核空间 $N(C)$ 的维数最小。核空间的引入使得我们能够用一种简洁的方法表示那些需要被分配到同一个处理器上的数据和计算，当增加一个新的分解约束条件时，仅需要更新核空间，而不必重新计算分解等式。因此在最终的算法实现时，可以首先求解分解矩阵的核空间，然后再求出对应的分解矩阵。

数据和计算在分解时是相互影响的，我们使用干涉图 $G_s(V_c,V_d,E)$ 来描述程序中循环嵌套对数组的访问情况，其中 V_c 节点表示循环嵌套，V_d 节点表示数组，如果某个循环嵌套对数组进行了访问，则它们之间存在访问边 E。

考虑到计算和数据之间的紧密关系，到某个处理器上执行的计算，它所引用的数据也应该被分布到相应的本地存储器上，假设循环嵌套 j 的计算划分矩阵为 C_j，数组 X 的数据划分矩阵为 D_x，且在循环嵌套 j 中对数组 X 的第 k 次访问函数为 f^k_{xj}。对于所有的循环迭代 $\vec{i}$ 有公式

$$D_x(f^k_{xj}(\vec{i}))+\vec{\delta_x}=C_j(\vec{i})+\vec{\gamma_j}\text{[8]} \tag{1}$$

完全符合上述公式的分解为无通信分解。公式中的 $\vec{\delta}$ 向量和 $\vec{\gamma}$ 向量指出了分解时必要的偏移，在算法实现时我们先将其忽略，待计算划分矩阵 C_j 和数据分布矩阵 D_x 求解完成后再行推导。公式简化为

$$D_xF^k_{xj}(\vec{i})=C_j(\vec{i}) \tag{2}$$

线性分解算法通过研究上述等式成立时，分解矩阵 C_j 和 D_x 的核空间必须满足的条件来限定分解时计算和数据需要遵守的约束。通过对此算法的仔细研究和代码实现时的检验，我们发现该算法有时会产生大量的通信。下面我们把 Anderson 和 lam 算法中的核空间约束等式和自己发现的约束等式一起列出，基于该组约束等式，可以得到一组完全无通信的分解，以下约束等式基于公式(2)。

同步约束：该等式描述了循环嵌套中需要被分配到相同处理器的迭代。考虑在一个循环嵌套中存在同步约束，当外层循环并行时，内层需要同步的循环需要被分配到相同的处理器上。

假设循环嵌套 j 的深度为 l，其中外层循环 $1\cdots s$ 可并行，内层循环 $q=(s+1)\cdots l$ 存在同步，那么循环迭代 $\vec{i}$ 和 $\vec{i}+\vec{e_q}$ 需要被分配到同一个处理器上，此处 $\vec{e_q}$ 是 l 维空间中第 q 个元素向量。公式描述为 $C_j(\vec{i}+\vec{e_q})=C_j(\vec{i})\Rightarrow C_j((\vec{i}-\vec{e_q})-\vec{i})=\vec{0}\Rightarrow C_j(\vec{e_q})=\vec{0}$，即

$$\vec{e_q}\in N(C_j) \tag{3}$$

迭代约束：该等式保证当一个循环嵌套的多次迭代对某个数组的同一元素进行访问时这些迭代被分配到同一个处理器上。

假设循环嵌套 j 的两次迭代 $\vec{i_1}$ 和 $\vec{i_2}$ 访问了数组 X 的某个元素，由数组仿射函数推出。当 $F^k_{xj}(\vec{i_1})=F^k_{xj}(\vec{i_2})$ 时，迭代 $\vec{i_1}$ 和 $\vec{i_2}$ 访问数组 X 的同一个元素，即 $F^k_{xj}(\vec{i_1}-\vec{i_2})=\vec{0}$。令 $\vec{t}=\vec{i_1}-\vec{i_2}$，则有 $\vec{t}\in N(F^k_{xj})$。由式(2)得知，如果 $\vec{t}\in N(D_xF^k_{xj})$，那么 $\vec{t}\in N(C_j)$。又因为 $N(D_xF^k_{xj})\geqslant N(F^k_{xj})$，因此由 $\vec{t}\in N(F^k_{xj})$，可以推导出 $\vec{t}\in N(C_j)$，即

$$\forall\vec{t}\in N(F^k_{xj}),\vec{t}\in N(C_j) \tag{4}$$

单环访问约束：该等式描述了当一个循环嵌套对某个数组进行多次访问时，对该数组分布的约束。Anderson 和 lam 忽略了该约束等式。经过理论验证和实验我们发现，如果没有此约束条件，某些例程在代码自动生成时有通信产生。

假设 F^1_{xj} 和 F^2_{xj} 表示循环嵌套 j 对数组 X 的两次访问，那么在引用图 G_s 中循环嵌套 j 和数组 X 之间存在一个简单的环。由式(2)可推导出等式 $D_xF^1_{xj}(\vec{i})=C_j(\vec{i})$ 和等式 $D_xF^2_{xj}(\vec{i})=C_j(\vec{i})$ 都成立，推导得 $D_x(F^1_{xj}(\vec{i})-F^2_{xj}(\vec{i}))=\vec{0}$ 成立，即

$$range(F^1_{xj}(\vec{i})-F^2_{xj}(\vec{i}))\in N(D_x) \tag{5}$$

多环访问约束：该等式保证某个数组被多个循环嵌套引用时，对该数组分布的约束。多环约束的作用是保证对每个数组仅存在单一的分解矩阵。多环访问的判定条件是，当干涉图 C_s 中存在由数组 X 出发到数组 Y 的多条不同路径，则存在多环访问。在工程实现时，还应该考虑由于访问传播引起的多环约束。

假设在干涉图 $G_s(V_c,V_d,E)$ 中存在节点 $V_x,V_y\in(V_d\cup V_c)$，不失讨论的一般性，假设 $V_x,V_y\in V_d$，如果有路径 $(V_x,\cdots,V_y,\cdots,V_x)$，则至少存在两个循环嵌套分别访问了数组 X 和数组 Y。假设这两个循环嵌套分别为 L_m 和 L_n，那么满足式(2)，等式 $D_xF^k_{xm}(\vec{i_m})=D_yF^k_{ym}(\vec{i_m})=C_m(\vec{i_m})$ 和 $D_xF^k_{xm}(\vec{i_n})=D_yF^k_{yn}(\vec{i_n})=C_n(\vec{i_n})$ 成立。考虑对数组 X 的约束得 $D_y=D_xF^k_{xm}(\vec{i_m})F^k_{ym}{}^{-1}(\vec{i_m})$ 和 $D_y=D_xF^k_{xn}(\vec{i_n})F^k_{yn}{}^{-1}(\vec{i_n})$ 成立，消元推出 $D_x(F^k_{xm}(\vec{i_m})F^k_{ym}{}^{-1}(\vec{i_m})-F^k_{xn}(\vec{i_n})F^k_{yn}{}^{-1}(\vec{i_n}))=\vec{0}$，即

$$range(F^k_{xm}(\vec{i_m})F^k_{ym}{}^{-1}(\vec{i_m})-F^k_{xn}(\vec{i_n})F^k_{yn}{}^{-1}(\vec{i_n}))\in N(D_x) \tag{6}$$

计算约束传播：前面 4 个约束等式阐述了并行分解时，一个数组中的哪些元素应该被映射到相同处理器，以及循环嵌套中哪些迭代应该被映射到相同处理器。下面两个约束等式将要阐述计算和数据分解矩阵核空间如何相互影响和传播。

计算约束传播等式保证了每个循环嵌套中映射到相同处理器上的迭代，所引用的数组元素也被映射到相应的处理器上。假设循环嵌套 L_j 中的迭代 $\vec{i_1}$ 和 $\vec{i_2}$ 被映射到相同的处理器，那么它们所引用的数组 X 中的元素也应该被映射到该处理器。由计算划分仿射函数，当等式 $C_j(\vec{i_1})=C_j(\vec{i_2})$ 满足时，迭代 $\vec{i_1}$ 和 $\vec{i_2}$ 被映射到同一个处理器，即等式 $C_j(\vec{i_1}-\vec{i_2})=\vec{0}$ 成立。设向量 $\vec{t}=\vec{i_1}-\vec{i_2}$，推导得 $\vec{t}\in N(C_j)$。由式(2)可知等式 $D_xF^k_{xj}\vec{t}=C_j\vec{t}=\vec{0}$ 成立，可知 $F^k_{xj}\vec{t}\in N(D_x)$，即：

$$span\{\vec{s}\mid\vec{s}=F^k_{xj}\vec{t},\vec{t}\in N(C_j)\}\subseteq N(D_x) \tag{7}$$

数据约束传播：该等式保证引用本地数组元素的循环迭代也被映射到本地的处理器。假设循环嵌套 L_j 中的循环迭代 $\vec{i_1}$ 和 $\vec{i_2}$ 引用了数组 X 中相同的元素，那么这两个迭代也应该被映射到相同的处理器。依然令向量 $\vec{t}=\vec{i_1}-\vec{i_2}$，由式(2)

知 $D_x F_{xj}^k \vec{t} = C_j \vec{t} = \vec{0}$ 成立，那么当 $F_{xj}^k \vec{t} \in N(D_x)$ 时，有 $\vec{t} \in N(C_j)$，即

$$span\{\vec{t} \mid F_{xj}^k \vec{t} \in (N(D_x) \cap range(F_{xj}^k))\} \subseteq N(C_j) \quad (8)$$

利用约束等式(3)，(4)，(5)，(6)初始化计算与数据分解矩阵核空间，利用约束传播等式(7)，(8)完成约束等式在计算与数据之间的传播，可以求得一组计算与数据分解矩阵核空间的解。当分解矩阵核空间矩阵满秩时表示无并行分解，否则存在无通信的并行分解。

定理 1 求解分解矩阵核空间的算法能够终止。

证明：在干涉图 G_s 中，对于任意 $v_x \in V_d$ 和 $V_j \in V_c$，向量空间 $N(D_x)$ 和 $N(C_j)$ 在算法执行过程中单调增。最坏情况下向量空间扩展为全空间，算法结束。

定理 2 基于等式(2)和上述约束等式求出的计算和数据分解矩阵核空间是最小的。

证明：算法仅使用约束等式(3)，(4)，(5)，(6)作为核空间初始条件，使用约束传播等式(7)，(8)保持计算和数据约束的一致性，最终找到一组稳定的分解矩阵核空间。因为求解计算和数据分解核空间算法中各约束等式是必要的，因此求解的核空间的约束条件最少，根据此条件得到的核空间的解也最小，此时计算和数据并行性最大。

定理 3 算法的解是计算和数据无通信分解时矩阵的核空间。

证明：算法将循环嵌套中存在的同步、迭代和它们引用的数组元素、数组元素和引用它们的迭代都放置到同一个处理器中，仅对完全可并行的计算和数据作了分解处理，因此最后的分解无数据依赖问题，即是一种无通信的分解。

由上述算法可以确定数组分布和计算划分矩阵的核空间，在干涉图 G_s 的一组相互关联的计算节点和数据节点中，选定其中一个分解矩阵作为基准，通过计算与数组关联等式(2)，可以推导出其它的计算划分矩阵和数组分布矩阵，该过程称之为方向对准操作。

如果给出的分解矩阵中所有的 C 和 D 都为零矩阵，说明利用该算法分析的串行代码无法并行，需要在一个处理器上串行执行，否则存在无通信的分解。对于某个循环嵌套 L_j，其并行度为 $rank(C_j)$，假设该循环嵌套有 l 层，那么并行度也可以表示为 $l - \dim(N(C_j))$。同理，对数组也有此分析结果。因此分解矩阵核空间的维数越大则并行性越小，分解矩阵的秩越大并行性越大。

最终的线性分解结果由线性分解矩阵和偏移向量组成，将计算分解矩阵 C 和数据分布矩阵 D 代入式(1)，使用贪心算法递归推导出最终分解需要的向量偏移。循环嵌套 L_j 的偏移向量为

$$\vec{\gamma}_j = D_x \vec{\beta}_{xj}^k + \vec{\delta}_x \quad (9)$$

同理，对于其它访问该循环嵌套的数组 Y 的分解偏移向量为

$$\vec{\delta}_y = \vec{\gamma}_j - D_y \vec{\beta}_{yj}^k \quad (10)$$

上述等式中，数据分布矩阵 D_x 和 D_y 已知，向量 $\vec{\beta}$ 是我们读取的数组仿射函数中的偏移向量，因此可以利用贪心算法求得一组稳定的偏移向量的解。

4 一个线性分解的例子

假设代码段中存在如下所示的两个循环嵌套 L_1 和 L_2，每个循环嵌套中有两层循环，共引用了 X,Y,Z3 个两维数组。经单循环嵌套局部分析可得，循环嵌套 L_1 中两层循环可做 doall 并行，循环嵌套 L_2 中的第一层循环做 doacross 流水并行，第二层做 doall 并行。

```
(1)for i : 0 to N do
    for j : 0 to N do
      Y[i,j]+=X[i,j];
(2)for i : 0 to N do
    for j : 0 to N do
      Z[i,j]=Z[j,i]+Y[j,i];
```

下面使用算法对该段代码进行分析。我们将讨论使用所有约束等式分析的结果，以及不使用单环约束等式(5)分析的结果，以阐述单环约束等式在无通信分解时的必要性。

首先讨论使用所有约束等式的分析。利用前 4 个约束等式对分解矩阵的核空间初始化。由于该段代码仅在循环嵌套第一层做流水并行，因此同步约束等式(3)对 L_2 进行约束，将第一层循环置于同一个处理器中，即 $span\{(1,0)\} \in N(C_2)$；考察代码中所有的数组访问线性转换矩阵 F 满秩，那么 F 矩阵的核空间为空，因此迭代约束等式(4)对分解矩阵无约束；该段代码中第二个循环嵌套对 Z 数组多次引用，即存在单环，使用单环约束等式(5)有 $span\{(1,1)\} \in N(D_z)$ 成立；代码对应的干涉图 G_s 中不存在多环，因此多环约束等式(6)对核空间没有增加约束。利用计算和数据传播约束等式(7)，(8)进行约束传播，最终的计算和数据分解矩阵都为空。说明该段代码无法找到一组无通信的划分。

下面我们不使用单环约束等式(5)参与对分解矩阵核空间的初始化，由上文的分析可以看出，仅有同步约束等式(3)做了约束。利用计算和数据传播约束等式(7)，(8)为核空间进行传播，并选择数组 X 作为对准操作的基本方向，最终的分解结果为 $C_1[1\ 0]$，$C_2[0\ -1]$，$D_x[1\ 0]$，$D_y[1\ 0]$，$D_z[0\ 1]$。该划分在一维处理器空间上对循环嵌套 L_1、数组 X 和 Y 按行划分，对循环嵌套 L_2 和数组 Z 按列划分。考察循环嵌套 L_2 中的数组 Z，该数据分解后依然存在依赖关系，并且读数组元素不在本地处理器，因此存在通信。

5 算法在工程实现中的改进

基于该算法的优化编译器能够对串行代码进行自动分解。如果存在并行，说明并行部分计算和数据严格的对准，是一种无通信的分解结果，编译器以矩阵或者线性不等式的形式给出分解结果，否则编译器报告所分析的串行代码不能够并行。在工程实现时我们发现，按照该算法实现的优化编译器在并行识别时会损失许多形式的并行性，因此考虑放松约束等式，尽可能利用程序中的并行性。

流水并行是一种常见的并行方式，在流水并行中允许循环嵌套内存在同步信息。如果循环嵌套内数据依赖关系的距离向量是有限的，我们放松同步约束等式(3)，允许存在流水并行的迭代被分配到不同的处理器上，但依然限制依赖距离向量为无限，或者单循环嵌套内不能进行完全可置换变化的子循环进行并行。此时的分解结果无数据重组，但存在少量因数据依赖产生的通信。

复制只读数组以提高并行机的执行是一种通用的技术。假设只读数组 X 的某个元素被不同的迭代同时访问，如果对该数组元素进行复制，那么就不必映射这些迭代到同一个处理器上。因此对只读数组可以放松迭代约束等式(4)，在数据分布时对只读数组在各处理器或者处理器某维上广播，此优

(下转第 293 页)

点锁序和事务 T 锁序。若所有的事务均按上述规则加锁，则这样的两段加锁机制是无死锁的。

证明：设系统在 t_0 时刻，该结点的结点锁顺序为：$L(f_{k1})<L(F_{k2})<\cdots L(F_{kk})$，事务 T_i 申请对文件 F_i 加锁，若有 $L(f_{ki})<L(F_j)$，则不会形成相互循环的等待状态，这由"<"的定义是显然的；若某次申请不满足 $L(f_{ki})<L(F_j)$，则说明本次申请的加锁和已有的加锁产生了冲突，从而形成了死锁的必要条件；但由于对冲突加锁的事务做了回滚操作，同时从结点锁序和事务 T_j 锁序中删除了相应的加锁，从而打破了 T_i 等待的条件，而事务 T_j 的再次加锁只能满足 $L(f_{ki})<L(F_j)$，这也就形成了一种按动态序加锁的方法，故不会形成死锁(证毕)。

2 无死锁的 2P-L 实现

2.1 结点加锁管理器的数据结构

(1)描述该结点锁序的双向队列 PQF：队列的每个结点为一结构类型，成员有文件名、写(Lw)或读锁 (Lr) 标记、实行加锁的事务名以及前向和后向指针，其头结点的两个指针分别指向队列的第一个元素和最后一个元素；

(2)描述每个事务锁序的队列 T_iQF：结点的结构除没有事务名外，其余相同。

结点加锁管理器采用分布式管理。

2.2 算法描述

当事务 T_i 要求对该结点的某个文件 F 加锁时

(1)取出 PQF 的最后一个元素的文件名 F_i、以及锁类型描述(Lw 或 Lr)；

(2)扫描队列 PQF，若 F 位于 F_i 之前且申请的加锁不属于共享锁，则转(5)否则转(3)；

(3)若 PQF 队列中没有 F，则将对 F 的加锁插入到 PQF 和 T_iQF 的队尾；

(4)若本次对 F 的加锁是共享锁(本次申请读文件 F，而 PQF 中对文件 F 的加锁也为读锁)，则将本次加锁插入到 T_iQF 的队尾，PQF 队列不变；

(5)把所有的对文件 F 加锁的子事务回滚(这些事务名从 PQF 可以查出)，并从 PQF 和相应的 T_iQF 队列中删除该事务对文件 F 的相应加锁(原有的其它锁序不变)；

(6)所有事务的提交，都应从 T_iQF 和 PQF 中删去相应的加锁。算法的正确性是显然的。

结束语 本文采用类似按序分配资源以预防死锁的方法，给出了两段加锁无死锁的充分条件，同时给出了加锁管理器的实现方法，由于结点和事务的锁序是按事务访问文件的顺序安排，且由于事务的回滚，原有的锁序也会发生改变，而不是事先规定的锁序，所以，它的效率和按序分配资源有本质的区别，该算法的实现也仅是几个队列的线性查找、删除、插入，算法复杂度为 $O(n)$，是完全可以接受的。算法的缺陷在于：(1)如何最大限度的减少事务回滚的次数；(2)若在同一结点或不同结点存在一个或多个副本时，如何优化该算法，还需进一步研究。

参 考 文 献

1 Al-Jumah N B, Hassanein H S, El-Sharkawi M. Implementation and modeling of two-phase concurrency control——a performance study [J]. Information and Software Technology, 2000,42:257～273

2 Kao A C B, Lam Kam-yiu. Comparing Two-Locking and Optimistic concurrency control Protocols in Multprocessor Real-Time Database. IWPDRTS(International Workshop on Parallel and Distributed Real-Time Systems)IEEE, 1997. 141～146

3 Tanenbaum A S. Modem Operating Systems(现代操作系统). 陈向群，等译. 北京机械工业出版社, 1999

(上接第 280 页)

化不影响分解的正确性。

如果在一个循环嵌套内存在对某个数组的多次访问，单环约束等式(5)往往会导致无法并行。而此类串行代码大量存在，为了使得算法更具有一般性可放松该约束等式。由于在计算划分和数据分布的过程中保留了依赖关系，因此对分解之后的代码仍可以运用依赖分析技术进一步的处理。这一步可以在代码自动产生时使用数据调整技术完成，通过在 SPMD 代码中增加显示的通信语句得到解决。

小结 本文主要阐述了在分布式存储环境中的一种基于线性代数的计算和数据自动分解算法，基于该算法实现的并行化编译器能够自动对串行代码进行无通信的分解。通过对计算划分和数组分布并行条件的深入分析，我们给出了完整的无通信分解约束等式，并对算法作了形式化的描述。在工程实现的基础上，为了增加算法的实用性，我们逐步放松了某些约束等式，并且对相应出现的分解结果进行了分析和实现，使得该算法的处理能力进一步的增强。

参 考 文 献

1 Knobe K, Lukas J D, Steele G L. Data optimization: Allocation of arrays to reduce communication on SIMD machies. Journal of Parallel and Distributed Computing, 1990,8:102～118

2 Lam M S, Wolf M E. Compilation techniques to achieve parallelism and locality. In: Proceedings of the DARPA Software Technology Conference, April 1992. 150～158

3 Huang C H, Sadayappan P. Communication-free hyperplane positioning of nested loops. In: Banerjee U, Gelernter D, Nicolau A, et al, eds. Languages and Compilers for Parallel Computing. Berlin, Germany, Springer-Verlag, 1992. 186～200

4 Kulkarni D, Kumar K G, Basu A, et al. Loop partitioning for distributed memory multiprocessors as unimodular transformations. In: Proceedings of the 1991 ACM International Conference on Supercomputing, June 1991. 206～215

5 Kumar K G, Kulkarni D, Basu A. Deriving good transformations for mapping nested loops on hierarchical parallel machines in polynomial time. In: Proceedings of the 1992 ACM International Conference on Supercomputing, July 1992. 82～91

6 Kennedy K, Kremer U. Automatic Data Layout for High Performance Fortra [A]. Proc Supercomputer [C]. San Diego, Calif, 1995

7 Wolf M E. Improving Locality and Parallelism in Nested Loops: [PhD Thesis]. Stanford University, August 1992

8 Anderson J M, Lam M S. Global optimizations for parallelism and locality on scalable parallel machines. In: Proceedings of the SIGPLAN '93 Conference on Programming Language Design and Implementation, Albuquerque, NM, June 1993. 112～135

9 Anderson J M. automatic computation and data decomposition for multiprocessors: [PhD Thesis]. Stanford University, March 1997

10 Guo Minyi. Efficient Techniques for Data Distribution and Redistribution in Parallelizing Compilers: [PhD Thesis]. Tsukuba University, 1998