文章编号:1001 - 9081(2006)10 - 2473 - 03

程序并行化中数据收集代码自动生成算法研究

龚雪容,生拥宏,沈亚楠
(信息工程大学 计算机科学与技术系, 河南 郑州 450002)
(gongxuerong@163.com)

摘 要:着重论述了串行程序并行化过程中的数据收集部分代码的自动生成。提出利用等价类的方法获取数据的最后写关系,并建立包括计算划分、循环迭代和数据最后写关系的不等式限制系统,最后利用 FME 消元法对不等式限制系统进行消元处理,最终实现数据收集代码的自动生成。

关键词:并行编译;等价类;数据收集;不等式系统
中图分类号: TP314 **文献标识码**:A

Study on automatic generation algorithm of the collection code in translating serial program into parallel program

GONG Xue-rong, SHENG Yong-hong, SHEN Ya-nan
(*Department of Computer Science and Technology, University of Information Engineering, Zhengzhou Henan* 450002, *China*)

Abstract: The parallelization of serial program is mainly made up of parallel identification, data and computation decomposition, dependence relation analysis and automatic code generation. Data gathering is a very important part of automatic code generation. This paper studied the automatic generation algorithm of data collection code, and brought forward how to get the last write relation of the data based on the equivalence class, then created an inequality system with computation decomposition, loop iteration and last write relation, and at last realized auto-generation of the data collection code by using FME elimination method.

Key words: parallel compilation ; equivalence class; data collection; inequality system

0 引言

高性能并行计算是现代科学研究、工程技术开发和大规模数据处理的关键技术。而并行编译系统是并行计算机系统软件中十分重要的一部分,提高并行编译技术对充分利用并行机资源和提高并行机效率起着十分重要的作用。

在分布式存储器机器中各处理器拥有自己的存储器,其编译过程可以分为两个阶段[1],一是确定如何进行计算划分和数据划分,使得计算并行化和计算中引用的数据尽量本地化以降低处理器间的通信开销;二是生成在各处理器上运行的并行程序,并行程序的作用在于正确地分配计算和进行处理器间的通信。数据收集属于后者,其作用在于对各处理器的计算结果进行收集以得到并行程序最终的执行结果。如何正确、高效地收集各处理器的执行结果是本文研究的重点。

1 问题提出

在并行程序的自动生成中无论是计算划分还是数据划分的技术都比较成熟,研究成果也比较多,而在如何做到比较精确地收集计算结果方面研究却比较少,这又是一个需要解决的问题,精确收集将会降低处理器间的通信开销并提高整个并行程序的性能。

通常,在生成数据收集循环代码时进行数据的全收集,利用初始数据分解在计算结束之后把所有发送出去的数据再全部收回,其过程是数据初始分布的逆。这样做的优点在于不会漏掉任何数据。这虽然保证了计算结果的正确性,但带来了冗余的通信开销。对例 1 而言,如果我们按照全收集的原则进行收集的话,将会对数组 a、b、c 都进行收集,而本例中数组 a、b 并没有被修改,仅仅是被引用,因此实际上是不需要收集的,这就极大地增加了通信开销。

例 1:

```
for(i=0;i<N;i++){
  for(j=0;j<N;j++){
    c[i][j]=0.0;
    for(k=0;k<N;k++){
      c[i][j]=c[i][j]+a[i][k]*b[k][j];
    }
  }
}
```

为了减少冗余通信、提高效率,理想情况下是只有被修改过的数据才被收集。如在例 1 中不再收集数组 a、b,甚至数组 c 中未被修改的元素也不收集,只收集数组 c 中那些被修改过的元素。

2 解决问题

为解决这个问题可以尝试利用计算划分来进行数据收集,而不是传统的利用数据划分。具体来说,首先建立一个以计算划分和循环索引边界限制为主的不等式系统,然后用

收稿日期:2006 - 04 - 04;**修订日期**:2006 - 06 - 08 **基金项目**:河南省杰出人才创新基金资助项目(0521000200)

作者简介:龚雪容(1975 -),女,四川井研人,助理工程师,博士研究生,主要研究方向:计算机软件、并行编译; 生拥宏(1977 -),男,江苏海安人,讲师,博士研究生,主要研究方向:面向对象、并行编译; 沈亚楠(1980 -),男,河南安阳人,硕士研究生,主要研究方向:计算机软件、并行编译.

FME 消元法对不等式进行处理,从而建立数据收集部分的接收和发送数据循环。对于此例来讲,这样的处理尽管可以不再收集数组 a 和 b,但是对数组 c 仍然进行的是全收集,同样带来了冗余通信增加了通信开销,因为对数组 c 中的每个被修改的元素都会重复收集 N 次,也就没有真正实现数据的精确收集。如何做到真正的精确收集,确保只收集被修改的数据且每个数据只收集一次,这就需要能精确判断数据最后被修改的地方,也就是数据的最后一次写操作发生的位置。

在本文中用统一的线性不等式的框架来表示和处理多维整数空间,包括计算划分、数据划分、精确的依赖关系分析、迭代空间和数据空间,并在此基础上利用投影和多面体扫描的方法[6]生成计算代码、通信代码、数据分布代码和数据收集代码。

1) 计算划分

在分布式存储器机器上,循环级并行的并行性是通过循环嵌套迭代空间划分的方法来实现的。把循环迭代分布到各个进程,使每个进程执行一部分计算,加速程序执行的速度。

定义 1 迭代空间 I

对于一个深度为 m 的循环嵌套,其循环边界是循环索引的线性函数,定义一个迭代空间 I 来表示,该空间是一个 m 维的多面体。循环嵌套的每次迭代对应多面体中的一个整数点,用索引向量表示为 $\vec{i} = (i_1, i_2, \cdots, i_m)$,每个迭代就是一次计算操作。

定义 2 虚拟处理器空间 P

一个 n 维的处理器数组定义了 n 维的处理器空间 P,该空间是一个 n 维的矩形。

计算划分即是对迭代空间 I 的划分,把迭代分布到各个进程进行计算操作,其定义如下:

定义 3 计算划分 C[3]

计算划分 $C = \{(\vec{i}, \vec{p}) \in I \times P \mid (\vec{b} + B\vec{u})\vec{p} \leqslant \boldsymbol{U}(\vec{i} - \vec{t}) < (\vec{b} + B\vec{u})(\vec{p} + 1)\}$,是满足特定关系的迭代和处理器对 $(\vec{i}, \vec{p})$ 的集合,其中 $\boldsymbol{U}$ 是一个扩展的幺模矩阵,$\vec{t}, \vec{b}$ 是整数向量,$\boldsymbol{B}$ 是整数矩阵,$\vec{u}$ 是符号向量如 $(\vec{b} + B\vec{u}) > 0$,处理器 $\vec{p}$ 执行迭代 $\vec{i}$ 当且仅当 $(\vec{i}, \vec{p}) \in C$。

在计算划分关系 C 中,$\boldsymbol{U}$ 指示计算划分是迭代空间的哪些维以及划分方式是正分还是斜分;$(\vec{b} + B\vec{u})$ 则指示了分块的大小;$\vec{t}$ 则指出了偏移的大小。计算划分在我们的系统中由上一遍自动生成。

2) 数据的最后一次写

为了建立数据收集不等式系统,我们需要找到对某个数据的最后一次写迭代 $\vec{i}$:

定义 4 数据的最后一次写

对一个循环嵌套,迭代 $\vec{i}$ 满足该循环的边界限制 I,其中 $\vec{i} = (i_1, i_2, \cdots, i_n)$,$lb_j \leqslant i_j \leqslant ub_j, 1 \leqslant j \leqslant n$,且 $!\exists \vec{i}'$,使得 $\vec{i}' \in I, F(\vec{i}) = F(\vec{i}'), \vec{i} < \vec{i}'$,其中 F 是数组访问函数,即在迭代 $\vec{i}$ 之后不存在另一个迭代 $\vec{i}'$ 与 $\vec{i}$ 访问同一个数据元素。

为获取最后写关系我们建立如下数学模型:数组访问函数 F 的等价类。利用数组访问函数 F 是等价关系的特点,通过划分等价类来寻找最后写关系。

对等价关系和等价类[5]有如下定理。

定理 1[5] 在一个集合上的每个等价关系生成这个集合的唯一的一个划分,使这个划分的块对应于 R - 等价类

对于访问函数 F 有如下定理:

定理 2 数组访问函数 F,是一个等价关系,定义域为 I。

证明如下:

a) 自反性证明

$\vec{i} \in I$,显然有 $F(\vec{i}) = F(\vec{i})$

b) 对称性证明

$\vec{i}, \vec{j} \in I$,若 $F(\vec{i}) = F(\vec{j})$,则 $F(\vec{j}) = F(\vec{i})$

c) 传递性证明

$\vec{i}, \vec{j}, \vec{k} \in I$,若 $F(\vec{i}) = F(\vec{j}), F(\vec{j}) = F(\vec{k})$,则 $F(\vec{i}) = F(\vec{k})$

因此,访问函数满足自反性、对称性和传递性,是一个等价关系。

证明结束。

既然数组访问函数 F 是集合 I 上的等价关系,因此可以根据等价关系 F 将集合 I 划分为不同的等价类,从而找出最后写关系。

由定理 2 可以得到查找最后写关系的算法,具体算法步骤如下:

输入:循环迭代空间 I、数组访问函数 F

输出:最后写关系(收集矩阵 gather)

算法步骤:

① 确定写操作所在的位置,在第几层循环嵌套的循环体中,从而确定 gather 矩阵的列数:层数 + 1;

② 根据数组写指令确定访问函数 F;

③ 根据循环边界确定数组访问函数的定义域;

④ 扫描整个定义域,利用数组访问函数把定义域划分为不同的等价类。具体实现方法是:建立一个链表,链表的节点为包含数组访问函数 F 的值(val)和当前迭代的结构,每当得到一个 F 值就需要扫描链表,如果链表中已经存在一个相同 val 的节点则把该节点的迭代值更新,如此在链表中的每个节点均表示最后一次对数据的写操作;

⑤ 把链表中各节点的迭代取出分析其各项之间的线性关系,该关系即是最后写关系;

⑥ 把得到的线性关系用不等式形式写入 gather 矩阵中。

3) 数据收集算法

在数据收集中首先需要得到数据的最后一次写关系,然后在根据计算划分 C 和循环迭代建立数据收集不等式系统,最后遍历该不等式系统用 FME 消元法生成数据收集通信代码。

建立的数据收集不等式系统应包含如下信息:

a) 迭代 $\vec{i}$ 满足循环边界限制 $\vec{i} \in I$,其中 $\vec{i} = (i_1, i_2, \cdots, i_n)$,且 $lb_j \leqslant i_j \leqslant ub_j, 1 \leqslant j \leqslant n, i_j$ 是循环索引;

b) 迭代 $\vec{i}$ 经计算划分之后分配到处理器 $\vec{p}_w$,即 $\vec{p}_w = \phi(\vec{i}) = C_s(i) + c_s$,其中 c_s 是一个 $1 \times n$ 的符号常量矩阵,C_s 是标量符号常量,表示了迭代 $\vec{i}$ 到一个进程的映射;

c) $\vec{p}_w \neq \vec{0}$ 被收集的对象是非0号进程上被修改的数据;

d) $!\exists \vec{i}'$,使得 $\vec{i}' \in I, F(\vec{i}) = F(\vec{i}'), \vec{i} < \vec{i}'$,其中 F 是数组访问函数,即在 $\vec{i}$ 之后不存在一个迭代 $\vec{i}'$ 与 $\vec{i}$ 访问同一数据元素,迭代 $\vec{i}$ 是对该数据的最后一次写迭代。

由最后一次写关系生成数据收集代码的算法描述如下:

输入:循环边界 I、计算划分 C、收集矩阵 gather

输出:含 MPI 调用的数据收集代码

算法步骤:

① 建立包含 gather 矩阵、计算划分 C 和循环迭代空间 I 以及限制条件 $\vec{p}_w \neq 0$ 的不等式系统 G,即:

$$G = \begin{cases} I: \text{循环边界限制} \\ \vec{p}_w = (\vec{i}) \text{ 计算划分} \\ \vec{p}_w \neq 0 \text{ 进程限制,进行收集的是主线程} \\ gather(\vec{i}) \text{ 最后写关系} \end{cases}$$

② 用 FME 消元方法遍历不等式系统 G,生成数据的循环发送代码,并在发送代码外插入 if(mypid! =0)以确保主进程以外的其他进程发送数据;

③ 生成主线程的数据循环接收代码,并在接收代码外插入 if(mypid = =0)以确保只有主进程接收数据;

④ 对其他需要收集的数组重复操作①~③。

3　实例分析

对前面提到的例 1 来讲,以收集数据 $c[i][j]$ 为例,该写操作在第三层循环内,故 *gather* 矩阵应该有 4 列,分别是常数列,循环索引 i、j、k,数组访问函数 $F = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix}$,设 $N = 4$,由循环边界可知 F 的定义域为 $\begin{cases} 0 \leqslant i \leqslant 3 \\ 0 \leqslant j \leqslant 3 \\ 0 \leqslant k \leqslant 3 \end{cases}$,扫描整个定义域,根据访问函数 F 把定义域划分为不同的等价类建立一个链表如图 1 所示,从各等价类中取出代表元(最后一个节点),分析各分项之间的关系可以知道当 $k = 3$ 时是对同一数据的最后一次写操作,因此 *gather* 矩阵的值应该是 $k = 3$,用不等式表示为 $\begin{cases} k - 3 \geqslant 0 \\ 3 - k \geqslant 0 \end{cases}$,表示为矩阵

gather　1　*C*　*i*　*j*　*k*

−3　0x0　0x0　1

3　0x0　0x0　−1

,其中"*gather*"是关键字,指明后面的矩阵是数据收集矩阵;"1"是数组引用编号,指明是对哪个数组引用的操作;"*C*"表示矩阵的该列为常量;"*i*、*j*、*k*"是循环索引;余下部分是数据收集矩阵。得到了收集矩阵 *gather* 之后建立生成收集代码的不等式系统,该不等式包含的条件有 $\begin{cases} 0 \leqslant i \leqslant 3, 0 \leqslant j \leqslant 3 \\ 0 \leqslant k \leqslant 3, p_w = i \\ p_w \neq 0, k = 3 \end{cases}$,对此不等式运用 FME 消元法即可建立数据收集通信的发送和接收循环代码。

对例 1 来讲采用这种精确的收集方法大大降低了通信,降低到原来通信量的 $1/N$,当然对于别的程序来讲效果是不一样的,只要循环中存在多次对要收集的对象的写操作,采用本方法都会减少通信,减少通信的多少与程序中收集对象被修改的次数有关,循环中同一数据被修改的次数越多采用本文所讲方法降低通信的效果就越明显。

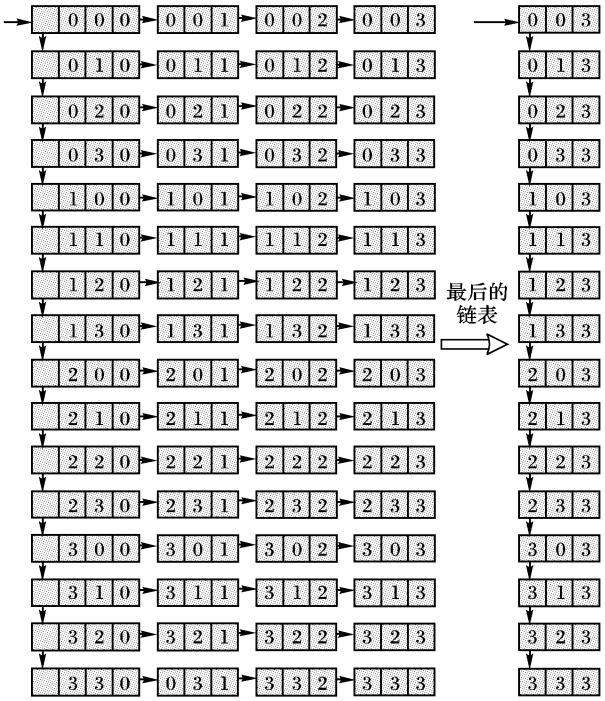

图 1　例 1 的等价类链表表示

4　结语

本文对精确的数据收集算法进行了初步的探讨,提出运用划分等价类的方法来查找最后写关系从而实现精确的数据收集,该方法的优点很明显,可明显的减少通信,但是该方法需要遍历整个迭代空间以获取最后写关系,导致寻找最后写关系的效率不高,后续工作将会围绕如何更好更快捷的得到最后写关系展开。

参考文献:

[1] AMARASINGHE SP, LAM MS. Communication optimization and Code Generation for distributed memory machines[A]. In the Proceedings of The ACM SIGPLAN'93 Conference on Programming Language Design and Implementation (PLDI)[C]. Albuquerque, New Mexico, 1993. 126-138.

[2] FERNER CS. The Paraguin compiler Message-passing code generation using SUIF[A]. in the Proceedings of the IEEE SoutheastCon 2002[C]. Columbia, SC, 2002. 1-6.

[3] ANDERSON JM, LAM MS. Global Optimizations for Parallelism and Locality on Scalable Parallel Machines[A]. In Proceedings of the SIGPLAN'93 Conference on Program Language Design and Implementation[C]. June 1993.

[4] 沈志宇, 胡子昂, 廖湘科, 等. 并行编译方法[M]. 北京: 国防工业出版社, 2000.

[5] TRFMBLAY JP, MANOHAR R. 离散数学结构及其在计算机科学中的应用[M]. 罗远诠, 李盘林, 池忠先, 等译. 上海: 上海科学技术出版社, 1982.

[6] ANCOURT C, IRIGOIN F. Scanning Polyhedral with DO Loops [A]. In Third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming[C]. PPOPP, 1991. 39-50.