自动并行化中不规则循环的代码生成

丁 锐 赵荣彩 徐金龙 傅立国
(数字工程与先进计算国家重点实验室 郑州 450002)

摘 要 许多大规模计算程序包含了不规则循环,但在面向分布存储的自动并行化中,以往的研究难以在编译时为不规则循环生成并行代码。针对一类常见的不规则循环提出了一种代码生成方法,该方法能在编译时将串行代码转换成等价的并行计算和通信代码,通过计算分解和数组引用的访问表达式来求解不规则循环在各处理器的本地定义集,并通过部分冗余的通信来满足不规则数组引用的生产者—消费者关系。实验结果表明,该方法是有效的,并对测试用例取得了预期的加速比。

关键词 自动并行化,计算分解,不规则循环,部分冗余
中图法分类号 TP314 **文献标识码** A

Code Generation for Automatic Parallelization of Irregular Loops

DING Rui ZHAO Rong-cai XU Jin-long FU Li-guo
(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450002, China)

Abstract Many large-scale scientific applications contain irregular loops. But the prior work of automatic parallelization on distributed memory is hard to generate parallel code for irregular loops at compile-time. We proposed an approach for effective code generation of a common class of irregular loops, and it's able to transform the serial code of irregular loops to equal parallel computation and communication code at compile-time. The approach searches the local definition set of the loops on each processor by computation decomposition and access expression of array references, and satisfies the producer-consumer relation of irregular array references by partial communication redundancy. The experimental results show that our approach is valid and improves the speedup of test applications.

Keywords Automatic parallelization, Computation decomposition, Irregular loops, Partial redundancy

1 引言

在分布存储结构的并行系统中,各计算节点都拥有着独立的存储空间,节点间需要明确的消息传递完成数据的交换。这意味着自动并行化不仅要合理地划分程序中的计算和数据,还要适时地插入消息通信代码来保证数据在各个进程间的一致性。代码生成位于并行化编译器的后端,根据前端提供的划分和数据流等信息,自动将串行代码转换成等价的并行计算以及通信代码。

一些大规模的科学计算应用中包含了不规则循环。例如在涉及稀疏矩阵的程序中,数据数组的索引需要通过其它数组的值来完成,而这样的间接索引导致数据访问模式非常不规则。这种下标表达式依赖于变量或非仿射函数,在运行时才能确定数据访问模式的数组访问就是非规则数组引用,包含非规则数组引用的循环就是非规则循环。

由于不规则循环中数组的数据访问模式是不确定的,并行编译器就无法在数组的定义点和引用点利用仿射的数据分解来确定数组的分布方式,从而确定处理器间应该交换哪些数据。即并行编译器无法确定不规则数据访问的生产者-消费者关系。而传统的代码生成研究主要是基于多面体编译器的程序变换,在缺少仿射划分信息的情况下,无法处理这样的不规则循环[1-6]。还有些研究虽然能够在编译时生成通信集,但要求数组下标是具有单调性特点的非线性表达式,无法处理由间接索引引起的不规则问题[14,15]。另外一些面向不规则问题的并行化研究,或者采用了 Inspector/Executor 计算模型的运行时库,或者是采用了投机并行模型等方法,都无法在编译时生成最终的通信代码[7-13,17-20]。

对于一些常见的不规则循环,虽然其包含着在编译时无法确定的数据访问模式,但其循环体依然存在着边界仿射的可并行循环层。针对此类不规则循环,本文提出了一种面向分布存储自动并行化的代码生成方法,即利用部分冗余的通信来处理数组访问不确定的生产者-消费者关系,并在编译时将串行代码转换成有效的计算和通信代码。由于具有边界仿射的可并行循环层,我们可以利用仿射的计算分解来将不规则循环的计算分配到每个处理器,由此产生计算代码;并在需要通信时,根据分配到本地的迭代空间和不规则数组引用的

到稿日期:2013-02-13 返修日期:2013-06-04 本文受"核高基"重大专项子课题(2009AA01220,2009zx10036-001-001)资助。
丁 锐(1984—),男,博士生,主要研究方向为并行编译,E-mail:dr2012earth@gmail.com;赵荣彩(1957—),男,教授,博士生导师,主要研究方向为体系结构和先进编译技术;徐金龙(1985—),男,博士生,主要研究方向为并行编译;傅立国(1989—),男,硕士生,主要研究方向为并行编译。

下标表达式将各处理器定义的数据都更新到其它处理器，以保证每个处理器所引用的数据总是在本地内存中。

该方法主要基于下面3个理由：(1)随着高性能计算机中互联技术的迅猛发展，其通信能力相比过去逐渐增强；(2)虽然将数据从生产者冗余地发送到了消费者以外的处理器，但是能够忽略编译时较难分析的生产者-消费者关系；(3)由于通信规则相对简单，可以使用MPI消息传递库中的组通信，借用其优化策略来减少消息传递步。本文的主要贡献如下：

- 详细分析计算分解和数据分解的对应关系，并提出根据计算分解切割到处理器本地的迭代空间和数组访问表达式来寻找本地定义的数据集合；
- 提出了部分冗余和冗余并行两种方法，以便能在编译时为一类不规则循环生成有效的计算和通信代码。

本文第2节对自动并行中为不规则应用产生通信代码的问题进行描述；第3节介绍了两级映射模型，以及如何通过计算分解求解数组在本地的定义集；第4节介绍了本文的通信代码生成算法；第5节是实验与分析；第6节详细介绍了相关研究；最后是结束语。

2 问题描述

在并行编译研究中，“仿射”是一个经常见到的名词。如果关于一个或多个变量 $x_1, x_2, \cdots, x_n$ 的函数可以被表示为一个常数加上常数乘以这些变量的和，即 $c_0 + c_1x_1 + c_2x_2 + \cdots + c_nx_n$，其中 $c_0, c_1, c_2, \cdots, c_n$ 是常数，那么这个函数就是仿射。一个规则的循环要求循环边界、if条件和数组下标都是循环索引和符号常量的仿射函数。仿射函数通常也被称为线性函数[23]。

但一些常见的数据访问模式并不是仿射的，例如涉及稀疏矩阵的程序。稀疏矩阵的常用表示方法之一是只保存一个向量中的非零元素，并使用辅助的下标数组，即间接数组来记录某一行从哪里开始，以及哪些列包含非零元素。图1是共轭梯度方法的核心代码。在循环2中，向量 w 的值是矩阵 a 和向量 p 的乘积。其中 a 使用了稀疏矩阵的标准表示方法——压缩稀疏行。数组rowstr是大小为n+1的稀疏矩阵，*rowstr*[j]指向 a 存储在 j 行中非零元素的连续集合的开始位置。例如对循环2的任意 j，非零元素存储在 a[*rowstr*[j]],…,a[*rowstr*[$j+1$]−1]中。数组colidx的大小和 a 一致，存储着 a 中每个元素在矩阵中的列编号。

```
1. for (j=1;j≤n;j++){ //Loop1
2.   p[j]=…
3. }
4. for (j=1;j≤n;j++)//Loop 2
5.     for (k=rowstr[j];k<rowstr[j+1];k++){
6.       w[j]+=a[k]*p[colidx[k]];
7. }
8. //other computation;not shown
```

图1 共轭梯度的核心代码

图2列出了循环2中一些迭代中每个数组的样值和间接索引关系，设 p 数组的大小为1400。循环 k 的边界由rowstr中的值决定，p 访问 j 行哪个数据元素则由colidx中的值决定。这种影响着控制流和数据访问模式的数组就是间接数组(rowstr和colidx)。而其它数组则是数据数组(a, w 和 p)。假设在这里已经对循环完成了可并行性分析。其中循环1和循环2的 j 层都没有携带依赖，是具有仿射边界的可并行循环，属于本文算法处理的循环类。

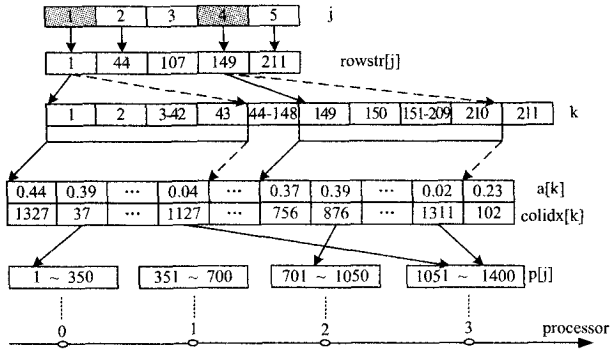

图2 共轭梯度中的间接索引关系

从图2可以看出，间接数组导致了 a 和 p 的数据访问模式是不规则的，无法在编译时用仿射函数表示其数据分布。假设对循环1划分后，数组 p 以块方式分布在4个处理器上。此时如果再划分循环2的 j 层迭代，p[colidx[k]]将以不规则的方式在各个处理器上进行读访问，即数据分布发生了剧烈变换。即使在编译时分析出colidx中存储的值，p 新的分布方式也无法表示为仿射的数据分解。而传统的通信代码生成需要编译前端给出的数据分解信息，才能生成远程数据交换的通信集，所以无法应用于这种不规则循环。另外，由于数组下标是间接数组，不是具有单调性的非线性函数，文献[14,15]中的通信集生成算法也无法适用。

3 数组的定义集

代码生成只有确定了数组在一个处理器定义了哪些数据，即数组引用的定义集，才能插入正确的通信。本节通过计算和数据分解的对应关系，介绍如何使用计算分解和数组下标来求解数组引用的定义集。

3.1 两级映射

本文的划分和代码生成算法采用了两级映射模型。其中第一级为虚拟映射，将计算和数据映射到一个规模不受限的虚拟处理器空间上。这里包含了3种类型的空间[23]：1)在并行化时，首先假设系统中有无限多个虚拟处理器，并表示成虚拟处理器空间。2)迭代空间是循环的迭代的集合，由循环的索引变量和边界给出。划分中计算分解通过对迭代空间的切割将计算映射到虚拟处理器空间。3)在科学计算程序中，数组是数据的主要载体。划分中数据分解切割的数据空间就是指数组元素的集合，由数组声明直接给出。图3解释了这3种空间之间的映射关系。

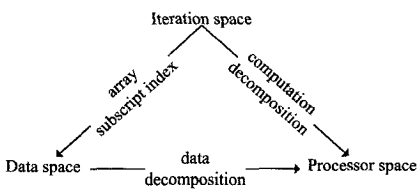

图3 两级映射

仿射分解是一种表示和求解划分的有效方法，使用仿射函数来表示计算和数据分解[24]。其中，数据分解刻画了 m 维数组空间到 n 维虚拟处理器空间的映射，表示为 $\vec{d}: A \to P$, $\vec{d}(\vec{a}) = D\vec{a} + \vec{\delta}$，其中 D 是一个 $n \times m$ 的线性转换矩阵，$\vec{\delta}$ 是偏移常向量，$\vec{a}$ 是数组索引向量。计算分解刻画了 l 维迭代空间到 n 维虚拟处理器空间的映射，表示为 $\vec{c}: I \to P$, $\vec{c}(\vec{i}) = C\vec{i} + \vec{\gamma}$，其中 C 是一个 $n \times l$ 的线性转换矩阵，$\vec{\gamma}$ 是偏移常向量，$\vec{i}$ 是迭代向量。

第二级映射为物理映射，完成虚拟处理器空间到物理处理器空间的映射。本文按照 Block 方式进行映射，映射函数可以表示为[24]：$M_s(\vec{p})=\vec{pid}=\lfloor(\vec{p}-\vec{lb_p})/\vec{bk}\rfloor$，其中$\vec{p}$表示需要映射的虚拟处理器，$\vec{bk}$表示按块映射时选取的块大小向量，由虚拟处理器个数与运行时物理处理器个数的比值上界确定。即$\vec{bk}=\lceil(\vec{ub_p}-\vec{lb_p}+1)/(\vec{ub_{pid}}-\vec{lb_{pid}}+1)\rceil$，其中$\vec{ub_p}$和$\vec{lb_p}$为虚拟处理器的上下界向量，$\vec{ub_{pid}}$和$\vec{lb_{pid}}$为物理处理器的上下界向量，$\vec{lb_{pid}}$一般为 0。由于虚拟处理器为整数空间，并且物理处理器空间为非负整数空间，因此可将 $M_s(\vec{p})$的等式关系等价转换为相应维上的不等关系(1)：

$$\vec{lb_p}+\vec{blksz}\times\vec{pid}\leqslant\vec{p}\leqslant\vec{lb_p}+(\vec{pid}+1)\times\vec{blksz}-1 \quad (1)$$

3.2 仿射计算分解

迭代空间和数据空间之间存在着这样的联系：一次迭代通过数组下标索引访问到相应的数组元素。如果一次迭代被分配到一个处理器，那在其执行时，迭代内访问的数据也必须要在这个处理器内。这说明，分别表示这两个空间划分方式的计算和数据分解，是可以相互转化的。即计算分解是对迭代空间的切割，而通过数组访问表达式，计算分解又完成了对数据空间的切割。而数据分解则必须与计算分解的切割保持一致，以保证一次迭代访问的数据就在本地。

如果一个嵌套循环中存在着间接数组等不规则的数据访问模式时，那么就无法使用仿射函数来表示数据分解。但根据计算和数据分解之间的对应关系，只要嵌套循环中存在满足仿射条件的可并行循环层，就可以从计算分解中找到传统代码生成算法需要的仿射特性。这样我们就可以通过计算分解和数组引用的下标索引来确定数组引用一个处理器中访问的数据元素。

例如在图 1 代码的不规则循环中，j 层循环是可并行的且具有仿射的循环边界，可以在编译时使用仿射函数来表示计算分解。通过计算分解将迭代空间映射到各个处理器后，也同时将 $p[colidx[k]]$访问的数据分布到各个处理器。我们在图 4 中描述了这样的映射关系。在循环 2 中，由于 $w[j]$的下标索引是仿射的，可以很容易通过数据分解判断出循环引用的数据集。这样的方法却不适用于不规则的数组引用 $p[colidx[k]]$。但依然可以根据计算分解和数据分解的对应关系，通过计算分解和数组下标找到 $p[colidx[k]]$引用的数组元素。

$$\begin{cases}\vec{lb_p}+\vec{bk}\times\vec{pid}\leqslant C\vec{i}+\vec{\gamma}\leqslant\vec{lb_p}+(\vec{pid}+1)\times\vec{bk}-1\\ \vec{lb}\leqslant\vec{i}\leqslant\vec{ub}\end{cases} \quad (2)$$

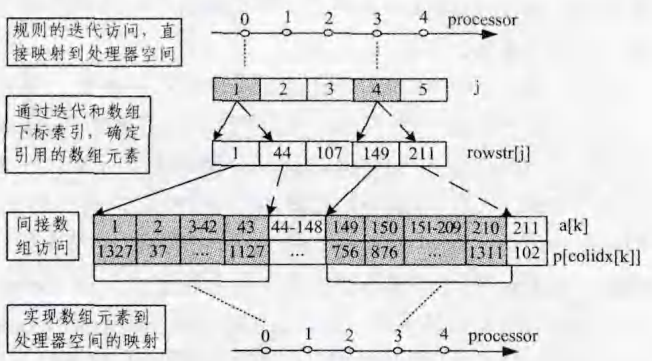

图 4 CG 中的空间映射

将不等式(1)中的虚拟处理器空间$\vec{p}$使用计算划分仿射函数$\vec{c}:I\to P, c(\vec{i})=C\vec{i}+\vec{\gamma}$替代，可以直接得出迭代空间向物理处理器的映射(2)，$\vec{lb}$、$\vec{ub}$分别表示循环索引$\vec{i}$在每一维上的上下界。此时可以对式(2)调用 FME 消元法[1,2](Fourier Motzkin Elimination)生成循环嵌套。然后，找到需要重分布的不规则数组引用，置于在生成的嵌套循环中。通过划分后的迭代空间和数组引用的下标索引，就访问到数组的定义集。图 5 中列出了循环 1 中 $p[j]$在当前处理器的定义集，pid 代表当前物理处理器的编号。

```
1. lb_j=max(pid * bk,0);
2. ub_j=min(pid * bk+bk-1,10);
3. for(j=lb_j;j<=ub_j;j++)
4.         p[j]=…;
```

图 5 $p[j]$的定义集

综上所述，如果由数组引用导致了不规则数据访问，但其所在的不规则循环具有边界仿射的可并行循环层，就可以使用计算分解和数组引用的下标索引来求解定义集。

4 通信代码生成算法

数组在定义点生成数据，并在引用点进行消费。如果数组在定义点和引用点的数据分解是已知的，编译器就可以通过分解矩阵和偏移之间的差别，分析出一个数组元素在处理器间确切的定义-引用轨迹，生成准确的通信代码。但不规则的数组引用无法使用仿射函数来表示数据分解。特别是一些数组引用的下标索引是间接数组，很难在编译时确定间接数组包含的值。这导致无法在编译时为不规则数组引用分析出数据的“消费者”和“生产者”。

为了弱化定义-引用问题，本文利用仿射的计算分解和数组引用求出定义集，并使用部分冗余和冗余并行两种方法来生成通信代码。下面分别对这两种方法进行介绍。

4.1 部分冗余法

在并行程序中，当数组的分布方式在定义点和引用点不一致时，就需要启动通信来维持数据的一致性。一旦数据分解在维间发生了改变，就会引起代价较大的数据重组通信。不规则问题的通信就属于数据重组通信。此时每个处理器都要和其它所有的处理器交换数据。以三维数组 $a[6][6][6]$为例，我们在图 6 中分别列出划分第一维和第二维的数据分布情况。图中灰色平面代表了数据分解对数据空间的切割，若一个数据在两个数据空间的颜色不一致，则其将在对应颜色的处理器间进行迁移。可以看出在 a 发生数据重分布后，每个处理器都要和其它所有的处理器交换数据。如果数组以不规则的方式分布，那么通信将更加复杂。

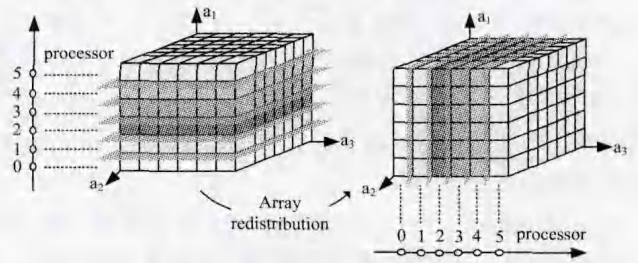

图 6 数组重分布

在数组不规则分布的情况下，即使能够分析出数组元素

在处理器间确切的定义-引用信息，编译器也需要产生复杂的代码来对缓冲区、偏移、数据量等必要的通信参数进行设置，才能精确地控制处理间的数据交换。这样的代码生成算法实现难度较大。另外由于数组定义的数据可能会被多次引用，而这些数组引用又可能具有不同分布方式，因此还需要启动多次精确的通信代码来完成不同的数组重分布。这无疑会增加通信的代价。

在处理不规则循环时，本文并没有产生精细的通信代码，而是通过计算划分和数组引用的下标引用，针对每个处理器寻找数组在本地的定义集合，并全部更新到其它每个处理器上，如图 7 所示。这样代码生成算法的复杂度较低，容易实现。通过这样的通信方式，能够保证无论对数据进行定义或数组引用采用何种分布方式，数据访问总是能够在本地得到满足。同时由于通信目的就是将本地定义的数据传输到其它处理器，通信规则相对简单，可以使用 MPI 消息传递库中高效的集合通信，如全交换 MPI_Alltoallv()。

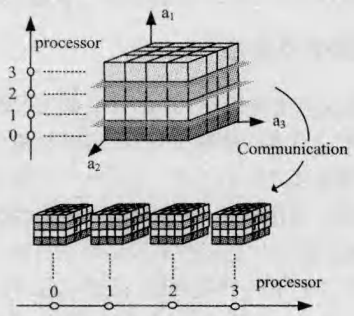

图 7　部分冗余通信

随着高性能计算机中互联技术的迅猛发展，其通信能力不断增强，可以适当地增加一些通信冗余来使每个处理器都拥有最新的数据，从而能够并行主导程序性能的核心不规则循环，并维护数据的一致性。文中将这样的代码生成方法称为部分冗余法。

首先将计算分解等信息代入式(2)，并对其调用 FME 消元法生成循环嵌套，替换目标循环的循环层。然后，找到需要重分布的不规则数组引用，生成通信中必要的缓冲区数据填充语句。复制已生成的嵌套循环层后，将填充语句置于其中。接下来添加消息传递的通信原语。最后生成在通信后从缓冲区取出数据的语句和循环。图 8 就是针对图 1 代码中 $p[colidx[k]]$生成的并行代码。其中第 1－10 行是缓冲区设置，根据局部迭代空间的边界来设置集合通信的通信参数。第 12 和 13 行是打包代码，将写引用 $p[j]$在当前处理器的定义集填充至发送缓冲区。第 15 行是集合通信的通信原语，根据通信参数的设置来完成数据交换。第 17－18 行是解包代码，此时接收缓冲区已经拥有了每个处理器最新定义的数据集，需要将其一一回写至数组 p 中。完成通信后，每个处理器都拥有 p 数组最新更新的所有数组元素，$p[colidx[k]]$读访问的数据都在本地。

在使用部分冗余法生成通信代码时，算法根据计算分解切割到本地的迭代空间，将写引用访问的数据依次填充到连续的缓冲区中，从而能够将处理器在本地定义的数据都更新到其它处理器，维护数据的一致性。

```
1  /* Set sendbuf and recvbuf */
2  rdipls[0]=1; j = k = 0;
3  for (i = 0; i < comm_size; i++)
4      recvconuts[i]=n-1+1;
5  for (i = 1; i < comm_size; i++)
6      rdipls[i] = rdipls[i-1] + recvconuts[i-1];
7  for (i = 0; i < comm_size; i++){
8      sendconuts[i]=recvconuts [pid];
9      sdipls[i] = rdipls[pid];
10 }
11 /* Fill sendbuf */
12 for (i = max(bk*pid, 1); i ≤min(pid*bk+bk -1, n); i++)
13         sendbuf[j++] = p[i];
14 /* MPI Collective Primitive */
15 MPI_Alltoallv(sendbuf, recvbuf, sdipls, rdipls);
16 /* Get data from recvbuf */
17 for (i = 1; i ≤n; i++)
18         p[i] = recvbuf[k++];
```

图 8　不规则循环的并行代码

4.2　冗余并行法

本文讨论的代码生成算法假设并行化产生的目标并行代码是 SPMD(single program multiple data)程序。其中并行代码的执行可以分为多个阶段，每个阶段的执行模式是两种类型之一：冗余并行执行、完全并行执行。不能并行化的串行代码被复制，在所有的处理机上冗余并行执行；并行循环被划分到多个处理机以完全并行方式执行。

如果一个处理器引用的数据就在本地，那么就不需要通信。由于串行代码的冗余并行执行，串行阶段被修改的数据在每个处理器上都是有效的，因此串行阶段无须进行后通信。对于由数组读引用引起的不规则访问，可以通过定义点的冗余并行来满足数据访问需求。这样即使划分后数组以不规则的方式分布在各个处理器，也不需要启动远程数据访问。图 9 说明了冗余并行和完全并行两种执行方式在遇到数组重分布时的不同之处。

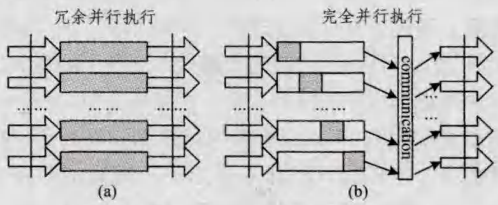

图 9　两种执行模式的数组重分布

由于必须牺牲定义点循环的并行，让其以冗余并行执行的方式来挖掘引用点循环的并行性能，我们称之为冗余并行法。只有在定义点循环没有位于程序的核心代码段，并行确实没有收益，才会设置其为串行。冗余并行也可以看作部分冗余的极端情况。

图 1 就是 NPB(NAS Parallel Benchmark)基准测试集中 CG 的核心代码，其中 $a[rowstr[j]]$使用了压缩稀疏行，需要间接数组 $rowstr[j]$中的值来访问 a 存储的非零元素。在划分循环索引 j 之后，计算分解对循环的迭代空间进行了切割，而通过数组访问表达式 $a[rowstr[j]]$，计算分解又完成了对数据空间的切割，将数组 a 分布到了各个处理器。而 $a[rowstr[j]]$并不满足仿射条件，所以无法用仿射分解表示其数据分布，因此以往的研究会放弃 $a[rowstr[j]]$的并行。由于 a 的定义点循环未处于共轭梯度法内部，并行收益不高，循环 2

又只对 a 进行了引用，我们就可以设置定义点循环为冗余并行，来满足 a 的数据访问模式。

综合 4.1 节和 4.2 节，本文的代码生成算法如图 10 所示。

Algorithm 1: Code_of_Irregular_Loop($\vec{c}, \overline{lb}, \overline{ub}, a, L$)

InPut : $\vec{c}$: Computation decomposition
a: Irregular array; L: Irregular loop
$\overline{lb}, \overline{ub}$: upper and lower bounds of loop
OutPut : Communication and computation code

```
1  Begin
2    substitute c, lb and ub into equation (2);
3    partition L, by using FME to (2);
4    if a defined outside critical code then
5      set loop of a's definition to serialize;
6    else
7      generate code :
8      Statement S1 to fill send-buff with data of a's reference;
9      Loop nest l1 to enclose S1, by using FME to (2);
10     MPI Primitive of Collective communication;
11     Statement S2 to copy data from buff to a's reference ;
12     Loop nest l1 to enclose S2, by lb and ub;
```

图 10　不规则循环的代码生成算法

5　实验与分析

为了验证本文的通信代码生成算法对并行程序的优化效果，对 NPB 基准测试集中的两个用例进行了测试。NPB 是 NASA (National Aeronautics and Space Administration)提供的基准测试集，目前广泛应用于并行计算机性能的评价，包含有串行版本、MPI 并行版本以及 OpenMP 并行版本等等，实验选取了 CG(共轭梯度)和 IS(整数排序)两个包含不规则数组引用和循环问题的用例进行测试。

基于 Open64 编译器和文献[26]中的划分算法，实现了本文的通信代码生成算法。为了对比算法优化前后的效果，对 NASA 提供的串行版本进行了自动并行化，分别自动生成了两种带有 MPI 函数的并行程序。首先在传统代码生成算法的基础上生成优化前无法处理不规则问题的并行程序，并在实验结果中标记为 Affine；然后使用部分冗余和冗余并行两种方法为不规则循环生成并行程序，在实验结果中标记为 Optimized；为了进一步说明并行性能，我们对 NASA 提供的 MPI 的并行版本进行了测试，在实验结果中标记为 Manual。

首先，在超微服务器上对两个程序的 W 规模进行了加速比测试，以验证算法的适用性和正确性。超微服务器配置了 4 颗 6 核 CPU，型号是 Intel Xeon X5670，主频为 2.93GHz，内存为 36GB。然后，为了验证算法在较大规模时的并行性能，我们在神威蓝光(Sunway Blue-Light)集群下对 CG 和 IS 进行了 A 和 B 规模的加速比测试。神威蓝光是由我国自主研发的高性能计算机系统，CPU 为 16 核 64 位处理器的 SW1600，单核最高频率为 1.1GHz。测试使用了 4 个 CPU，共 64 个核，实验结果如图 11 所示。

CG 用于求解大型稀疏对称正定矩阵的最小特征值的近似值，最外层循环使用反幂法，次外层循环则使用了共轭梯度法，但这两层循环均不能并行。图 1 的不规则循环是 CG 的核心循环，占据了共轭梯度法 93%的执行时间[25]。并行程序 Optimized 是应用本文算法后生成的，能够划分该循环并生成正确的通信代码，取得预期的加速比。IS 通过木桶排序法对小型整数进行排序，不包含浮点数运算。在核心函数 Rank() 中，占据计算主导地位的嵌套循环包含了使用间接数组的数组写引用。将其划分并生成通信代码后，Optimized 程序也取得较好的加速比，并在 B 规模接近手工并行的效果。

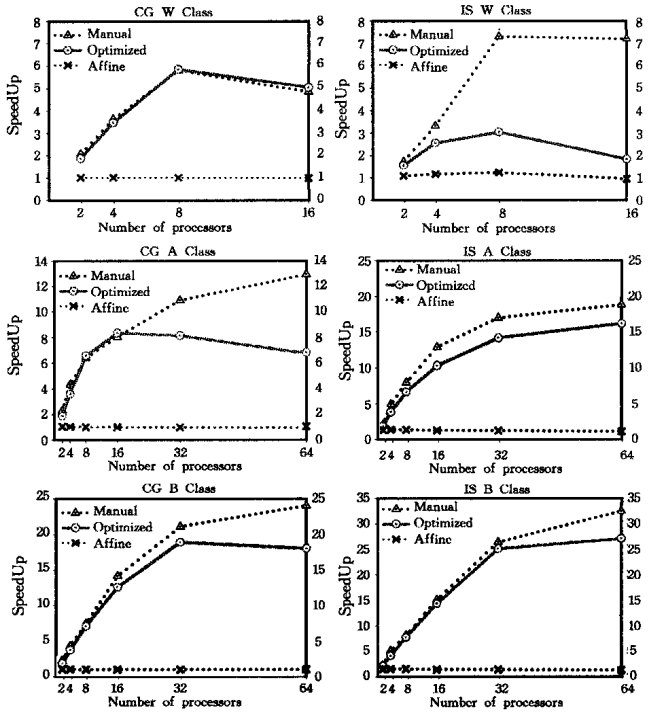

图 11　实验结果

NASA 由于还没有通信冗余，因此相比应用部分冗余法和冗余并行法产生通信代码的 Optimized 程序，加速比更加理想。而基于多面体的传统代码生成算法不能处理规则循环，所以 Affine 程序没有并行 CG 和 IS 中的核心循环，加速比始终在 1 左右徘徊，这也和文献[25]中的论断是一致的。

在 CG 和 IS 的 W、A 和 B 3 个规模的测试中，Optimized 版本的并行程序都出现了性能拐点，并且随着规模的增大，拐点出现的时间越来越晚。这是由于随着启动进程总数的增加，分配到每个进程的计算量不断减少，虽然计算时间一直在下降，但越到最后变化就越细微；而通信的总量却随着处理器个数不断地攀升，通信和计算的能力越来越不匹配，最终导致程序的并行性能开始下降。

从实验结果可以看出，传统算法能够为规则程序生成高效的通信代码，但却不能处理不规则问题。本文算法则能够对传统算法进行扩展，利用仿射计算分解以及冗余并行模型，在编译时生成有效的通信代码。测试结果说明了本文算法的有效性。

6　相关研究

针对自动并行化中的代码生成问题，有许多研究成果。文献[1]最早给出了利用一系列 FME 投影产生循环嵌套的算法。文献[2]则将计算和数据分解、数据流等信息表示为由线性不等式系统构成的多面体，并将多面体投影到低维空间来处理代码生成和通信优化等问题。文献[3,4]基于文献[2]的理论建立了一个开源的并行化编译器，用于产生分布存储结构的消息传递代码，并对该理论在打包和解包时的错误进

行了改进以及引入了物理处理器映射来减少通信量。文献[5,6]首次提出了一种端到端的自动分布存储并行化和代码生成编译器，通过多面体编译器框架来生成 MPI 并行程序。这些研究都是基于多面体模型，只能处理满足仿射条件的规则循环。

一些针对不规则问题的研究主要是通过采用 Inspector/Executor (I/E) 非规则计算模型的运行时库来解决。程序在运行时，通过 Inspector 阶段分析数组引用并生成通信调度，在 Executor 阶段则使用通信调度进行通信并完成计算。关于这方面的研究，Saltz 等人开发了 CHAOS/PARTI 运行时库，提供有效的运行时原语在处理器间分布计算和数据，减轻手工开发消息传递代码的压力[7,8]。文献[9]中提出了一种在编译预处理不规则循环时，根据最小通信计算原则来划分循环迭代的策略，它能够减少通信开销。文献[10,11]针对不规则循环中数组访问表达式是由索引、参数和间接数组中存储的值所组成的一类情况，提出了一种自动并行化和分布存储代码生成框架，其混合静态和运行时分析，产生有效的 I/E 代码。

Strout 等人提出了一种扩展的多面体框架 SPF(Sparse Polyhedral Framework)，它将间接内存引用、运行时生成的数据以及迭代重排序表示为解析函数符号(uninterpreted function symbol)，即一种输出值是未知的函数。该框架产生的也是 I/E 代码[12,13]。Basumallik 等人在研究 OpenMP 到 MPI 自动程序变换时，插入 Inspector 代码来动态分析不规则数组的实际数据访问模式，并通过计算和通信隐藏来减少开销[17]。以上这些研究的特点是，需要在运行时的 Inspector 阶段划分迭代空间、生成本地内存访问序列，并最终生成通信集。Inspector 阶段的开销最终都会转嫁到程序的并行性能上。

还有些研究能够不借助 I/E 模型，在编译时就完成不规则循环的划分和代码生成。例如文献[14]使用符号的分析技术，将由循环边界、数组下标表达式、条件语句以及计算和数据分布等信息组成符号化等式和不等式组来表示一个限制(restriction)，并通过寻找一个满足循环中所有数组限制的解来生成最小消息的通信集。文献[15]提出了 AIS(Algebraic Inequations Solver)算法，即用代数不等式来表示循环边界、数组引用和分布约束条件，通过解代数等式组能在编译时得到非规则通信集的代数解。这些研究为了能够在编译时生成通信集，要求数组下标是具有单调性特点的非线性表达式，无法处理由间接数组引起的不规则问题。

文献[16]研究了 OpenMP 到 GPGPU 程序的自动转换和优化，针对不规则应用提出了一种循环折叠(loop-collapsing)技术。它们能够将图 1 循环 2 的两层循环折叠成一个单独循环，并使用混合编译和运用时分析来证明数组 a 和 $coldix$ 是连续的内存访问。文献[18]在多核处理器上根据对不规则程序加速比的模拟以及动态反馈的信息，自动选择合适的循环进行并行，并能减少由循环携带依赖引起的线程间通信开销。针对 Clusters 的自动并行，Kim 等人使用投机并行来处理那些包含指针、间接数组以及过程间依赖等问题的循环[19]，类似的研究还有文献[20]等。文献[21]针对不规则的单一索引访问和简单的间接数组访问(间接数组的下标由所处循环的索引组成)提出了一种编译时依赖分析方法。文献[22]则根据二次规划提出了一种非线性数组下标的精确依赖测试方法。这些研究都没有解决分布存储的代码生成问题。

结束语 并行编译器很难通过静态分析来寻找不规则循环中数组访问的定义-引用关系。以往的研究或者通过运行时分析以及动态反馈来处理这样的循环，或者直接放弃并行。随着互联技术的发展，高性能计算系统中结点间的通信速度越来越快。本文通过部分冗余和冗余并行两种方法，保证每个处理器定义的数据都能传输到其它处理器的本地内存中，从而满足了不规则循环中只能在运行时确定的数据访问模式，并以此在编译时回避了数组的定义-引用问题。在接下来的工作中，我们将对计算和通信隐藏技术进行研究，以此来提高通信的效率。

参 考 文 献

[1] Ancourt C, Irigoin F. Scanning polyhedra with do loops[C]// Proceedings of the third ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP'91), 1991. NY, USA: ACM New York, 1991: 39-50

[2] Amarasinghe S P, Lam M S. Communication optimization and code generation for distributed memory machines[C]// Proceedings of The ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation (PLDI'93), 1993. NY, USA: ACM New York, 1993: 126-138

[3] Ferner C S. The Paraguin Compiler-Message-Passing Code Generation Using SUIF[C]// Proceedings IEEE SoutheastCon, 2002. Washington DC: IEEE Computer Society, 2002: 1-6

[4] Ferner C S. Revisiting Communication Code Generation Algorithms for Message-passing Systems[J]. International Journal of Parallel, Emergent and Distributed Systems, 2006, 21(5): 323-344

[5] Bondhugula U, Hartono A, Ramanujam J, et al. A practical automatic polyhedral parallelizer and locality optimizer[C]// Proceedings of The ACM SIGPLAN 2008 Conference on Programming Language Design and Implementation (PLDI'08), 2008. NY, USA: ACM New York, 2008: 101-113

[6] Bondhugula U. Automatic distributed-memory parallelization and code generation using the polyhedral framework[R]. IISc-CSA-TR-2011-3. Bangalore: Indian Institute of Science, 2011

[7] Ponnusamy R, Saltz J, Choudhary A. Runtime-compilation techniques for data partitioning and communication schedule reuse [C]// Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, 1993. NY, USA: ACM New York, 1993: 361-370

[8] Mukherjee S S, Sharma S D, Hill M D, et al. Efficient support for irregular applications on distributed-memory machines[C]// Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming 1995. NY, USA: ACM New York, 1995: 68-79

[9] Guo Min-yi, Li Li, Chang Weng-long. Efficient loop partitioning for parallel codes of irregular scientific computations[J]. IEICE Transactions on Information and Systems, 2003, 86(9): 1825-1834

(下转第 44 页)

[2] Pei J, Han J, Mortazavi-Asl J, et al. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth[C]// Proceedings of the 7th International Conference on Data Engineering. IEEE Computer Society Washington, DC, USA, 2001: 215-224

[3] 陆介平,刘月波,倪巍伟,等. 基于投影数据库的序列模式挖掘增量式更新算法[J]. 东南大学学报,2007,36(3):457-462

[4] 张坤,朱杨勇. 无重复投影数据库扫描的序列模式挖掘算法[J]. 计算机研究与发展,2007,44(1):126-132

[5] 张利军,李战怀. 基于位置信息的序列模式挖掘算法[J]. 计算机应用研究,2009,26(2):529-531

[6] 汪林林,范军. 基于 PrefixSpan 的序列模式挖掘改进算法[J]. 计算机工程,2009,35(23):56-58,61

[7] Saputra D, Rambli D R A, Foong O M. Sequential Pattern Mining using PrefixSpan with Pseudoprojection and Separator Database[C] // 2008 International Symposium on Information Technology. Kuala Lumpur, Malaysia, Aug. 2008: 1-7

[8] 公伟,刘培玉,贾娴. 基于改进 PrefixSpan 的序列模式挖掘算法[J]. 计算机应用,2011,31(9):2045-2047

[9] 鲍钰,黄国兴,张召. 基于 Web 日志挖掘的网站结构优化方法[J]. 计算机工程,2003,29(12):82-84

[10] 胡建武,何贞铭,张贻权. Web 日志挖掘及其实现[J]. 计算机工程与应用,2004,40(14):156-158

[11] Yang Z L, Wang Y T, Kitsuregawa M M. An Effective system for mining web log[C] // Proceedings of the 8th Asia-Pacific Web conference on Frontiers of WWW Research and Development (APWeb'06). Harbin, China, January 2006: 40-52

[12] Mabroukeh N R, Ezeife C I. A Taxonomy of Sequential Pattern Mining Algorithms[J]. Journal of ACM Computing Surveys, 2010, 43(1): 1-41

(上接第 14 页)

[10] Ravishankar M, Eisenlohr J, Pouchet LN, et al. Code generation for parallel execution of a class of irregular loops on distributed memory systems[C]// The International Conference for High Performance Computing, Networking, Storage, and Analysis (SC12), 2012. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012

[11] Ravishankar M, Eisenlohr J, Pouchet LN, et al. Code generation for parallel execution of a class of irregular loops on distributed memory systems[R]. OSU-CISRC-5/12-TR10. The Ohio State University, 2012

[12] Strout M M, George G, Olschanowsky C. Set and relation manipulation for the sparse polyhedral framework[C]// Proceedings of the 25th International Workshop on Languages and Compilers for Parallel Computing (LCPC2012), 2012. Springerburg LNCS series, 2013

[13] LaMielle A, Strout M. Enabling code generation with sparse polyhedral framework[R]. CS-10-102Colorado State University, 2010

[14] Guo Min-yi, Pan Yi, Liu Zhen. Symbolic Communication Set Generation for Irregular Parallel Applications[J]. The Journal of Supercomputing, 2003, 25(3): 199-214

[15] 胡长军,李静,王珏,等. 一类非规则并行应用问题的通信集生成算法[J]. 计算机学报,2008,31(1):120-126

[16] Lee S, Min S J, Eigenmann R. OpenMP to GPGPU: a compiler framework for automatic translation and optimization[C]//Proceedings of the 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP'99), 2009. NY, USA: ACM New York, 2009: 101-110

[17] Basumallik A, Eigenmann R. Optimizing irregular shared-memory applications for distributed-memory systems[C] // Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPOPP'06), 2006. NY, USA: ACM New York, 2006: 119-128

[18] Campanoni S, Jones T, Holloway G, et al. HELIX: Automatic Parallelization of Irregular Programs for Chip Multiprocessing [C]//Proceedings of the 10th International Symposium on Code Generation and Optimization (CGO'12), 2012. NY, USA: ACM New York, 2012: 84-93

[19] Kim H, Johnson N P, Lee J W, et al. Automatic Speculative DOALL for Clusters[C] // Proceedings of the 10th International Symposium on Code Generation and Optimization (CGO '12), 2012. NY, USA: ACM New York, 2012: 94-103

[20] Zhuang X, Eichenberger A E, Luo Y, et al. Exploiting parallelism with dependence-aware scheduling[C]// Proceedings of the 2009 International Conference on Parallel Architectures and Compilation Techniques (PACT' 09), 2009. Washington DC, USA: IEEE Computer Society, 2009: 193-202

[21] Lin Y, Padua D. Compiler analysis of irregular memory accesses [C]//Proceedings of The ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation 2000. NY, USA: ACM New York, 2000: 157-168

[22] Zhao J, Zhao R C, Han L. A Nonlinear Array Subscripts Dependence Test[C]//Proceedings of the 14th IEEE International Conference on High Performance Computing and Communications. Liverpool, England, 2012: 764-771

[23] Aho A V, Lam M S, Sethi R, et al. Compilers principles, techniques, and tools (Second edition)[M]. Boston: Addison-Wesley, 2007

[24] Amarasinghe S P, Lam M S. Communication optimization and code generation for distributed memory machine: [C]//Proceedings of The ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation (PLDI'93), 1993. NY, USA: ACM New York, 1993: 126-138

[25] Mitchell N, Carter L, Ferrante J. Localizing Non-affine Array References[C]//Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques (PACT' 99), 1999. DC, USA: IEEE Computer Society Washington, 1999: 192-202

[26] 丁锐,赵荣彩,韩林. 基于主导值的计算和数据划分算法[J]. 计算机科学,2012,39 (3):290-294