# 循环携带反依赖的MPI自动并行化研究

赵 捷 赵荣彩 韩 林 许瑾晨
(解放军信息工程大学信息工程学院 郑州 450002)

**摘 要** 传统的面向MPI的自动并行化使用的依赖测试技术,只能确定代码中是否含有循环携带依赖,而不对循环携带依赖的类型进行判定。经研究发现,当循环携带的是反依赖时,代码仍然可以在一定条件下实现面向MPI的自动并行化。基于依赖测试方法和数据流信息,通过创建合理的依赖数据副本,提出了一种循环携带反依赖的MPI自动并行方法。实验结果表明,所提出的方法能够有效识别存在循环携带反依赖的并行循环,将其作为后端生成MPI代码的依据可有效提高MPI程序的效率。

**关键词** 自动并行化,依赖测试,MPI协议,反依赖,循环携带
**中图法分类号** TP301.6 **文献标识码** A

## Loop-carried Anti-dependence MPI Auto-parallelization Research

ZHAO Jie ZHAO Rong-cai HAN Lin XU JIN-chen
(Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

**Abstract** Traditional MPI auto-parallelization dependence testing methods can only detect whether there are loop-carried dependences, but not their types. It was proved that auto-parallelization with loop-carried anti-dependence can be achieved under certain conditions. By creating reasonable copies of dependence data, a MPI auto-parallelization method with loop-carried anti-dependence was proposed based on the dependence testing methods and the data-flow information. The experimental results show that the proposed method can effectively recognize the parallel loops with loop-carried anti-dependence. Using the results of the method to generate MPI codes can efficiently improve the efficiency of MPI programs.

**Keywords** Auto-parallelization, Dependence testing, MPI protocol, Anti-dependence, Loop-carried

## 1 引言

无论是面向共享存储结构的OpenMP(Open Multi-Processing)自动并行化,还是面向分布存储结构的MPI(Message Passing Interface)自动并行化,都以程序中的循环为研究基础,识别程序并行循环的实质就是测试循环体内的依赖关系。

依赖测试是用来判断循环嵌套中在两个相同数组的下标引用之间是否存在依赖的方法[1]。在实际应用程序中,数组的下标引用可以是任意表达式,也可以是数组,但是大多数情况下数组下标是当前循环索引的线性表达式,现有的依赖测试方法也只能处理线性数组下标的情况。针对程序中存在多层嵌套循环的特点,线性数组下标中可能存在多个循环索引,因此又可以将数组下标分为单数组下标和耦合数组下标。

单数组下标的依赖测试方法主要有GCD(Greatest Common Divisor)测试法[2]和Banerjee不等式测试法[3]。GCD测试法的思想是求解方程(访问相同数据空间的数组下标构成的线性方程)左边所有变量系数的最大公约数,如果不存在最大公约数则无依赖。Open64编译器中对简单数组下标的依赖测试使用的就是GCD测试法[4]。但是GCD测试法大多数情况下返回的最大公约数是1,这时GCD测试法只能认为测试结果可能有依赖。Banerjee不等式测试法是用迭代极限的方法消除GCD测试法无法解决的问题,测试结果也比较精确,但是对于循环边界与其它循环归纳变量相关的情况,Banerjee处理的效率不是十分理想,因此Banerjee不等式还没有扩展到复杂的迭代空间中。

耦合数组下标的依赖测试方法主要有Delta测试法[5]和Omega测试法[6]。Delta测试法的精确度取决于耦合数组下标的性质,而且测试的复杂度与下标数目线性相关,但Delta测试受限于约束传播。Omega测试法的原理是基于Fourier-Motzkin消去法,其已发展成为一种能消除部分伪依赖的工具。Omega测试法的缺点是时间复杂度较大。

随着多核处理器的发展,多核处理器的新特性给MPI应用带来了新的优化空间[7]。如何改进依赖测试方法,使其适应MPI并行程序已成为一项研究热点。上述依赖测试方法测试结果只说明被测数组下标有依赖,无法确定是何种依赖。并行化以程序中的循环为并行单位,而现有的并行循环识别方法是根据上述依赖测试方法测试循环中的携带依赖,所以不能确定是哪种依赖。对于循环携带反依赖,其产生依赖的两个语句之间是"先读后写"的关系。

到稿日期:2011-07-31 返修日期:2011-10-28 本文受"核高基"重大专项(2009ZX01036-001-001-2)资助。
赵 捷(1987-),男,硕士生,主要研究方向为先进编译技术,E-mail:zjbc2005@163.com;赵荣彩(1957-),男,博士,教授,博士生导师,主要研究方向为高性能计算与先进编译技术;韩 林(1978-),男,博士,副教授,主要研究方向为先进编译技术;许瑾晨(1978-),男,硕士生,主要研究方向为高性能计算。

面向 OpenMP 自动并行化基于多线程方式实现并行的，线程之间通过共享的数据存储空间维持数据的一致性。而面向 MPI 的自动并行化是基于多进程方式实现并行的，数据的一致性通过进程之间的通信来完成。循环携带反依赖无法实现 OpenMP 自动并行化，因为存在读写共享的数据存储空间而导致无法保证数据一致性的问题。但是循环携带反依赖可以通过创建数据副本的方式实现 MPI 自动并行化，通过将产生依赖的数据副本分发给每个进程。由于反依赖是先读数据后写数据，因此对于相同地址单元的数据，处理器上只需具备原始数据。

本文根据 MPI 程序将数据分发到各个处理器的特点，通过创建适当的数据副本，将这些数据副本分发给需要的处理器来实现循环携带反依赖的 MPI 自动并行化。

## 2 循环携带反依赖的测试

现存的依赖测试方法能够测试循环中的携带依赖，因此可以先借助现有的依赖测试方法确定是否含有携带依赖，在此基础上再根据数据流分析判断产生依赖的源点和汇点的数据访存类型和数组下标之间的关系来确定循环携带反依赖。

### 2.1 测试循环携带依赖

为方便测试循环携带依赖，文献[8]引入了距离向量(Distance Vectors)和方向向量(Direction Vectors)。

**定义 1** 假设从 $n$ 层循环嵌套的迭代 $i$ 中语句 $S_1$ 到迭代 $j$ 中语句 $S_2$ 有依赖，则依赖距离向量 $d(i,j)$ 定义为长度为 $n$ 的向量，使得 $d(i,j)_k=j_k-i_k$。

**定义 2** 假设从 $n$ 层循环嵌套的迭代 $i$ 中语句 $S_1$ 到迭代 $j$ 中语句 $S_2$ 有依赖，那么依赖方向向量 $D(i,j)$ 定义为长度为 $n$ 的向量，使得

$$D(i,j)_k=\begin{cases}"<", & d(i,j)_k>0\\ "=", & d(i,j)_k=0\\ ">", & d(i,j)_k<0\end{cases}$$

如果一个方向向量的所有分量都为“=”，那么意味着这个嵌套循环中无循环携带依赖。如果方向向量的分量中有非“=”，那么这个循环中有循环携带依赖，而且方向向量的最左非“=”分量必定是“<”。最左非“=”分量是“>”，意味着依赖的汇点出现在源点之前，这与事实不符。因此，要实现以循环为并行单元的自动并行化，只需考虑含非“=”分量的方向向量。

考虑形如图 1 的嵌套循环代码片段。

```
    for(i1=L1;i1<=U1;i1++){
  for (i2=L2;i2<=U2;i2++){
      …
        for(in=Ln;in<=Un;in++){
S1      A(f(i1,i2,…,in))=…
S2        …=A(g(i1,i2,…,in))
      }
    }
  }
```

图 1 嵌套循环中的依赖语句

由于依赖是访问相同的数据引起的，因此考察 $S_2$ 是否依赖于 $S_1$，等价于求解方程组

$$f(x_1,x_2,\cdots,x_n)=g(y_1,y_2,\cdots,y_n) \quad (1)$$

循环索引的上下界构成了式(1)解空间 $R$，而方向向量的每个分量 $D_i(1\leqslant i\leqslant n)$ 为方程组的变量对 $(x_i,y_i)$ 限定了关系。因此循环的依赖测试就是在空间 $R=\{L_i\leqslant x_i,y_i\leqslant U_i\mid 1\leqslant i\leqslant n\}$ 上满足方向向量限定的关系 $x_iD_iy_i$，$1\leqslant i\leqslant n$ 的条件下，求解式(1)。其中，$L_i$ 和 $U_i$ 分别为第 $i$ 层循环的下界和上界。

### 2.2 提取循环携带反依赖

确定了循环中含有循环携带依赖之后，下一步工作就是要判断依赖类型是否为反依赖。根据反依赖的定义，产生依赖的两个语句只要满足第一个语句先读某一单元的数据，第二个语句再向这个单元写入数据，那么这两个语句之间就是反依赖的关系。

从程序数据流信息的角度出发，输出依赖不影响反依赖的识别，因为输出依赖是两次写数据操作而引起的依赖，而真依赖和反依赖都是一次写操作和一次读操作引起的依赖。区别读写操作的先后顺序，含有循环携带依赖的数据，其操作的先后顺序取决于循环迭代，循环迭代映射到数组数据上的方式是通过数组下标实现，因此要判断真依赖还是反依赖，可以在确定数据操作类型的基础上，通过考察数组下标来判断。算法如下。

输入：程序依赖图；

输出：循环携带反依赖的并行性；

过程：

1)遍历依赖图中的边 E，获取依赖边的个数 num，设 i=0；

2)如果 i=num，那么进入 7)；

3)获取 $E_i$ 的源点 $S_i$ 和汇点 $T_i$，如果 $S_i$ 和 $T_i$ 的数据操作类型都是 STORE，那么 i=i+1，进入 2)；

4)分别分析 $S_i$ 和 $T_i$ 中间代码，并查找符号表，获得 $S_i$ 和 $T_i$ 对应数据的数组下标 $INDEX_s$ 和 $INDEX_t$；

5)令 $SUB=INDEX_t-INDEX_s$，由于分析的是线性数组下标，因此 SUB 必定是循环索引的线性表达式，不妨设 $SUB=a_1x_1+a_2x_2+\cdots+a_nx_n+c$，其中 n 是循环个数，x 是索引。如果 $\forall a_i(1\leqslant i\leqslant n, a_i\geqslant 0)$ 不成立那么 i=i+1，进入 2)；

6)如果 SUB<0，那么该依赖为真依赖，i=i+1，进入 2)；SUB>0 则该依赖为伪依赖，i=i+1，进入 2)；

7)如果循环中只有循环携带反依赖，则标记该循环为可并行，否则为不可并行，算法结束。

算法过程考察 $\forall a_i(1\leqslant i\leqslant n, a_i\geqslant 0)$ 的情况，如果存在某个 $a_i(1\leqslant i\leqslant n)$ 为负，那么 *SUB* 的值在循环体中就有可能发生改变，这样既有真依赖又有反依赖，不适合 *MPI* 自动并行化。

其中，6)不考虑 $SUB=0$ 的情况，因为如果 $SUB=0$，则说明该依赖为循环无关依赖，不影响 *MPI* 自动并行化。图 2 是提取循环携带反依赖的流程图。

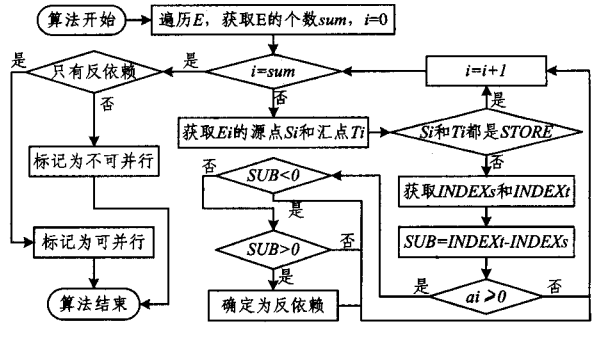


图 2 提取循环携带反依赖的流程图

从算法的过程中不难发现，影响算法时间复杂度的关键步骤在于搜索循环依赖图的边。因此算法的时间复杂度是线性的，值为 $O(n)$，$n$ 为被识别循环依赖图中的依赖边个数。

## 3 MPI 自动并行化的实现

循环携带反依赖的 MPI 自动并行化的实现，实质上就是对多项式 $a_1x_1+a_2x_2+\cdots+a_nx_n+c$ 的考察。为方便说明，先分析单数组下标的情况，即考察多项式 $ax+c$。其中，$x$ 是循环索引。

### 3.1 循环索引无关依赖

当 $a=0$ 时，循环携带反依赖的依赖距离即为 $c$（因为是单数组下标，所以多项式的依赖距离向量只有一列）。MPI 自动并行化根据数据划分将数据分发到各个处理器上，假设产生循环携带反依赖的数组数据个数为 $N$，处理器的个数为 $proc$，那么每个处理器上的数据个数为 $data=N/proc$（如果不能整除则将多余数据分发到最后一个处理器上）。分析依赖距离与处理器上迭代次数的关系（即 $data$ 与 $c$ 之间的关系）来实现 MPI 的自动并行化。

#### 3.1.1 依赖距离大于等于处理器迭代次数

当依赖距离大于等于处理器迭代次数（即 $c\geqslant data$）时，$n$ 号处理器要读入的数据没有划分到该处理器上，此时循环携带反依赖可以视为不存在。因此需要将其它处理器上的数据通信给 $n$ 号进程。但由于通信的开销可能导致并行效率的降低，这种情况下可以将产生依赖（指循环携带反依赖）的所有数据副本全分发给所有处理器，即每个处理器上都有产生依赖数据的全部副本，来维持数据的一致性。图 3 是 $c\geqslant data$ 时数据副本分发给处理器的示意图。

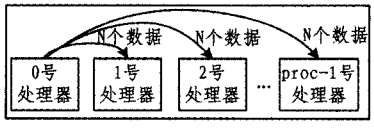


图 3 $c\geqslant data$ 时数据副本的分发示意图

#### 3.1.2 依赖距离小于处理器迭代次数

当依赖距离小于处理器迭代次数（即 $c<data$）时，$n$ 号处理器要读入的数据在之后的循环迭代中会写入新的值。但是在 $n$ 号处理器上的循环迭代是串行执行的，因此只需考虑 $n$ 号处理器上最后 $c$ 次迭代要读入的数据。最后 $c$ 次迭代要读入的数据在 $n+1$（或 $n-1$）号处理器上。如果按 $c<data$ 的情况处理，*MPI* 自动并行化也可以实现，但是如果参与并行的数据太多，这种方式会导致过多的通信开销，在 $c\ll data$ 时这种开销显得更明显。因此只需将 $n+1$（或 $n-1$）号处理器上的前（或后）$c$ 次迭代数据副本也划分给 $n$ 号处理器即可。图 4 是 $0<c<data$ 时数据副本分发给处理器的示意图。

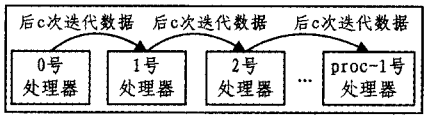


图 4 $0<c<data$ 时数据副本的分发示意图

为简化实现过程，可以设置一个阈值，这个阈值对是否进行并行化没有结果，只会决定程序是采取哪一种（图 3 还是图 4）并行方式。如假设设置的阈值为 $data/2$（表示当 $0<c<data/2$ 时按照本节提出的方式，当 $data/2\leqslant c<data$ 时按照上节提出的方式），分别有两个程序 PROGRAM1 和 PROGRAM2，其中 PROGRAM1 的数据依赖距离是 $data/3$，PROGRAM2 的数据依赖距离是 $2data/3$，那么对 PROGRAM1 程序实施循环携带反依赖并行化的结果按照本节提出的方法进行数据副本分发，而对 PROGRAM2 程序则采用上节提出的方法进行分发。

### 3.2 循环索引相关依赖

当 $a\neq 0$ 时，循环携带反依赖的依赖距离为 $ax+c$。$x$ 是循环索引，假设循环索引的上下界分别为 $lb$ 和 $ub$，那么依赖距离的取值范围就是 $[a\times lb+c, a\times ub+c]$。与 3.1 节中不同的是要分析一维区间上的点 $data$ 与线段（$[a\times lb+c, a\times ub+c]$）的位置。

如果按照点与线段之间的相互位置选择 MPI 自动并行化实现的方式，那么分析的过程比较复杂，即使分析出结果，其并行效率也可能不会十分理想。因此，采用携带反依赖数据全分布的方式处理即可。

### 3.3 耦合数组下标

当多项式涉及到耦合数组下标，即多项式并非简单的 $ax+c$ 而是 $a_1x_1+a_2x_2+\cdots+a_nx_n+c$ 时，MPI 自动并行化实现的实质是一个 $n$ 维数据空间上的一个点（0,…,0,data,0,…,0）与 $n$ 维多面体的位置关系。同 3.2 节中的原因，采用携带反依赖数据全分布的方式处理即可。

这种 MPI 自动并行化的实现方式能够解决绝大部分实际应用问题。在实际应用程序中，大多数数组的依赖只与最内层循环的循环索引有关，而且多数是发生数据偏移的情况，即依赖与循环索引无关。耦合数组下标的情况比较少见。

## 4 实例分析与测试

### 4.1 Jacobi 迭代改写例子的分析

Jacobi 迭代是一种比较常见的迭代方式，主要思想就是新的一次迭代计算值是上次获得的相邻数值点的平均值。Jacobi 迭代的核心代码，如图 5 所示。

```
  for (j=1; j<=N; j++)
    for (i=1; i<=N; i++){
S1    B[i, j] =(A[i-1, j]+ A[i+1, j]+ A[i, j-1]+ A[i, j+
              1])/4;
    }
```

图 5 Jacobi 核心代码

为便于分析循环携带反依赖，对 Jacobi 迭代的核心代码段简化并进行改写，如图 6 所示。

```
  for (j=1; j<=N; j++)
    for (i=1; i<=N; i++){
S1    B[i, j] =(A[i-1, j]+ A[i+1, j]+ A[i, j-1]+ A[i, j+
              1])/4;
S2    A[i, j] = 0.0;
  }
```

图 6 改写后的 Jacobi 核心代码

上面改写的方式虽然没有实际应用的意义，但是存在循环携带反依赖。该实例中存在 4 个循环携带反依赖，得到的多项式分别为

$$\begin{cases}0*i-1\\0*i+1\\0*j-1\\0*j+1\end{cases}$$

将 $i$ 和 $j$ 写入了多项式的原因是为了区别由哪个循环索引导致的依赖。由于依赖距离为 1，只需将 $n+1$（或 $n-1$）号处理器上前一次迭代的数据副本划分给 $n$ 号处理器即可。但

是这个例子中 $i$ 和 $j$ 两层循环都具有循环携带依赖，产生依赖的数据较多，不如采用循环携带反依赖数据全分布的方式。如果 $S_1$ 语句是 $B[i,j]=(A[i-1,j]+A[i+1,j])/4$ 的形式，那么按照 $j$ 层循环进行数据划分，采用前一种方式就可以获得更好的效果。

### 4.2 IS 程序的测试

将本文提出的方法作为后端生成 MPI 代码的依据，可有效提高 MPI 程序的效率。测试采用 NPB(NAS Parallel Benchmarks)基准测试集进行测试，以 NPB3.2.1 中的 IS(Integer Sort，整数排序)程序为例，该程序是通过木桶排序法对小型整数进行排序，不包含浮点数运算。对 IS 程序中可能产生循环携带反依赖的代码进行了分析，其分析方法是先确定循环迭代内含有携带依赖，从中提取反依赖，再通过比较依赖距离和处理器迭代的大小来选择数据副本的分发方式。表 1 列出了 IS 程序中可能产生循环携带反依赖的代码段及相应的分析结果。

表 1 IS 程序中循环携带反依赖分析

| 代码在程序中行号 | 代码片段 | 是否循环携带反依赖 | 数据分发方式 |
|---|---|---|---|
| 395 | key_array[i−1] > key_array[i] | 否 | 无 |
| 452 | bucket_ptrs[i]=bucket_ptrs[i−1]+bucket_size[i−1] | 是 | 发给相邻处理器 |
| 493 | key_buff_ptr[i+1] += key_buff_ptr[i] | 是 | 发给相邻处理器 |

将上述分析结果应用于后端生成 MPI 代码，测试生成的 MPI 代码的效率。测试平台建立在目标机 SunWay 集群系统上，该集群由 20 个节点组成。每个节点配置 4 个主频为2.8 GHZ 的 Xeon(TM)处理器。节点间通过百兆交换机相连。操作系统为 Red Hat Linux 7.2 2.96-118.7.2 smp，MPI 编译器为 MPICH 1.2.7。分别测试未实现循环携带反依赖的 MPI 自动并行化和已实现循环携带反依赖的 MPI 自动并行化的加速比，测试结果如图 7 所示。

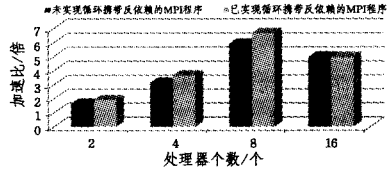


图 7 循环携带反依赖的 MPI 程序加速比

从图 7 可以看出，已实现循环携带反依赖的 MPI 程序有效提升了 MPI 程序的加速比，从而验证了循环携带反依赖 MPI 自动并行化研究的必要性。

**结束语** 本文基于循环携带依赖的测试和反依赖的提取，主要阐述了通过创建有效的数据副本实现循环携带反依赖的 MPI 自动并行化。文中分别对单数组下标情况和耦合数组下标情况进行了讨论，通过对 Jacobi 迭代程序的改写和理论分析，证明了循环携带反依赖的 MPI 自动并行化的可行性；通过对 NPB3.2.1 中 IS 程序的测试，证明了将这种方法作为后端生成 MPI 代码的依据，可有效提高 MPI 程序的效率。下一步工作是降低循环携带反依赖的 MPI 通信开销和全局一致的数据划分，以提高并行代码的收益。

## 参 考 文 献


[1] Allen J R, Kennedy K. 现代体系结构的优化编译器[M]. 张兆庆，乔如良，冯小兵，等，译. 北京：机械工业出版社，2004：23-90

[2] Bulic P, Gustin V. On Dependence Analysis for SIMD Enhanced Processors[J]. Lecture Notes in Computer Science, 2005, 3402(1): 527-540

[3] Bulic P, Gustin V. D-Test: An Extension to Banerjee Test for a Fast Dependence Analysis in a Multimedia Vectorizing Compiler [C]// Proceedings of IPDPS, 2004. Washington DC: IEEE Computer Society, 2004

[4] Venkatasubramanyam R D. Array Access Analysis in Open64 [D]. Houston: University of Houston, 2004

[5] Pouzols F M, Lendasse A, Barros A B. Autoregressive time series prediction by means of fuzzy inference systems using nonparametric residual variance estimation[J]. Fuzzy Sets and Systems, 2010, 161(4): 471-497

[6] Zhou Jing, Zeng Guo-sun. A general data dependence analysis for parallelizing compilers[J]. The Journal of Supercomputing, 2008, 45(2): 236-252

[7] 王洁，曾宇，张建林. 多核机群下基于神经网络的 MPI 运行时参数优化[J]. 计算机科学，2010，37(6)：229-232

[8] Drakakis K, Gow R, Rickard S. Distance Vectors in Costas Arrays[C]// Proceedings of the 42nd Annual Conference on Information Sciences and Systems. Washington DC: IEEE Computer Society Press, 2009


(上接第 282 页)


[5] 陈书智，王未央. 基于 Harris 角点检测的改进算法[J]. 现代计算机，2010，12(6)：44-46，57

[6] 林鹏岳，李玲玲，李翠华. 一种改进的快速 SUSAN 角点检测算法[J]. 计算机与现代化，2010，15(2)：66-68，72

[7] 张海燕，李元媛，储晨昀. 基于图像分块的多尺度 Harris 角点检测方法[J]. 计算机应用，2011(2)：70-71

[8] Tissainayagam P, Suter D. Asses sin g the performance of corner detectors for point feature tracking applications [J]. Image and Vision Computing, 2004, 22(8): 663-679

[9] 刘贵喜，刘冬梅，刘凤鹏，等. 一种稳健的特征点配准算法[J]. 光学学报，2008，28(3)：454-461

[10] Di Lui-gi, Mattoccia S, Mola M. An efficient a lg orithm for exhaustive template matching based on normalized cross correlation[C]// Proceedings of the 12th International Conference on Image Analysis and Processing. Los Atamitos CA, USA, 2003: 322-327

[11] 罗佳宇，田会永，王燕，等. 基于角点特征的自动图像配准[J]. 软件，2011，11(02)：67-70

[12] 陈显毅. 图像配准技术及 MATLAB 编程实现[M]. 北京：电子工业出版社，2009