国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第18卷第3期 1996年9月 Vol. 18 No. 3

MPP Fortran 程序中串行循环的优化*

唐新春 郭克榕

(国防科学技术大学计算机系 长沙 410073)

摘 要 在大规模并行处理系统中，通讯开销是影响程序性能的一个重要因素。本文提出了一种MPP Fortran程序中串行循环的优化技术：在串行循环中加入同步控制、将串行循环转换成共享循环。该技术能减少通讯开销、提高程序的性能。

关键词 数据分布，通讯开销，依赖距离向量，同步控制

分类号 TP312

Optimizing Sequential Loops in MPP Fortran Programs

Tang Xinchun Guo Kerong

(Department of Computer Science, NUDT, Changsha, 410073)

Abstract Communication overhead is an important factor which affects program performance in massively parallel processing systems. This paper introduces an optimizing technique for sequential loops in MPP Fortran programs. This technique, through application of synchronization control to sequential loops and changing sequential loops into shared ones, will reduce communication overhead and improve program performance.

Key words deat distribution, communication overhead, data dependance distance vector, synchronization control

MPP Fortran[1]是面向大规模并行处理系统的一种并行程序设计语言。在MPP Fortran程序中，一般将共享循环中的赋值语句左部出现的数组作为共享数组，并分布在多个处理机上，以支持循环的并行执行。当该数组同时出现在一个由于存在跨迭代的数据依赖关系而不能并行化的循环中时，为了循环的正确执行，一般将循环放在串行区，由一个处理机顺序执行。然而这种控制方法对大规模并行处理系统而言存在严重的不足，数组的分布特性势必引起串行循环中大量的非局部数据访问，所产生的通讯开销将严重损

* 国防科技预研基金资助项目
1995年12月29日收稿

害程序并行执行的性能。

本文提出了一种串行循环的优化技术：首先计算循环中的依赖距离向量，在循环中加入同步控制，然后将串行循环转换为共享循环，由所有的处理机共享执行。该技术能减少通讯开销，提高程序的数据局部化性能。

1 循环优化

设有如图 1 所示串行循环。

```
CDIR $ MASTER
       DO  I=2, 32
         DO J=2, 64
           A(I, J)=A(I, J)+A(I-1, J)+A(I, J-1)
         ENDDO
       ENDDO
CDIR $ END MASTER
```

图 1

其中 CDIR $ MASTER 和 CDIR $ END MASTER 标识串行区的开始与结束，并行区只由一个处理机执行，其它处理机在串行区出口处等待。

假定 A 同时在某共享循环中出现，确定必须为共享数组，大小为 32×64，分布方式为(BLOCK(16)，BLOCK(32))，每维分配的处理机数为 2，则数组 A 的分布情况如图 2 所示。

A(1：16, 1：32) PE_0 (0,0)	A(1：16, 33：64) PE_2 (0,1)
A(17：32, 1：32) PE_1 (1,0)	A(17：32, 33：64) PE_3 (1,1)

图 2

其中 BLOCK(16)表示数组 A 的第一维下标在第一维处理机上按大小为 16 的块循环分布，而 BLOCK(32)表示数组 A 的第二维下标在第二维处理机上按大小为 32 的块循环分布。每维分配的处理机数组成一个 2×2 的二维空间，元组(0,0)对应 PE_0，…，(1,1)对应 PE_3。

从图 2 可以看出，由于数组元素分布在所有的处理机上，如果单独由处理机 PE_0 执行所有的迭代，则 PE_0 必须访问 PE_1、PE_2、PE_3 上几乎所有的数组元素，存在大量的非局部数据访问。若将迭代空间(I,J)分为 4 个部分：(2：16, 2：32)，(17：32, 2：32)，(2：16, 33：64)，(17：32, 33：64)，分别由 PE_0～PE_3 执行，这样每个 PE 读写 A(I, J)时都是局部的，而读取 A(I－1,J)和 A(I,J－1)时只在边界上有少量的通讯。

由于循环中存在跨迭代的数据依赖关系，不能直接将串行循环改为共享循环。必须加入同步控制，以保证数据依赖关系不被破坏。首先，确定哪些变量对引起流依赖，计算出依赖距离向量[2]。(对于输出依赖和反依赖，可以采用易名[3]，复制[4]等技术消除，本文不予考虑。) 在本例中，A(I,J)分别与 A(I－1,J)和 A(I,J－1)引起流依赖，相应的依赖距离向量为(1,0)和(0,1)确定依赖距离向量以后，在循环中加入事件同步，就可以将串行循环转换为共享循环，如图 3 所示

```
CDIR$ DOSHARED(I, J)ON A(I, J)
      DO I=2, 32
        DO J=2, 64
          CALL WAIT_EVENT(EV(I-1, J-0))
          CALL WAIT_EVENT(EV(I-0, J-1))
          A(I, J)=A(I, J)+A(I-1, J)+A(I, J-1)
          CALL SET_EVENT(EV(I, J))
        ENDDO
      ENDDO
```

图 3

其中，CDIR $ DOSHARED (I,J) ON A(I,J)表示紧接其后的两层完全嵌套循环为多个处理机执行的共享循环，迭代空间按 A(I,J)划分，即迭代(i,j)由 A(i,j)所在的处理机执行。

过程 WAIT_EVENT 表示等待事件发生，而 SET_EVENT 表示宣告事件发生（将事件变量设置为 1)。EV 为共享数组，下标(I-1,J-0)和(I-0,J-1)由循环控制向量(I, J) 与依赖距离向量(1,0)和(0,1)相减而定。这样三条事件同步调用语句确保了 A(I, J)计算之前 A(I-1,J)和 A(I,J-1)已经计算完毕，即保证了数据依赖关系得到满足。

EV 的大小可以定义为(1：32，1：64)，分布方式与 A 相同。EV(I,J)与 A(I,J)是对准访问，全部为局部的，而 EV(I-1,J)和 EV(I,J-1)的访问只在边界上有少量通讯。

事件数组 EV 必须初始化。将其下标集分割为 3 个部分(1，1：64)，(2：32，1)和(2：32，2：64)，其中(2：32，2：64)与循环迭代空间相对应，所以 EV(2：32，2：64)应该初始化为“未发生”状态（设置成 0)。为了保证共享循环能够启动，EV 的其它部分即 EV(1，1：64)和 EV(2：32，1)应该初始化为“发生”状态（设置成 1)。初始化部分如图 4 所示。

```
CDIR$ DOSHARED(J) ON EV(1, J)
      DO J=1, 64
        CALL SET_EVENT(EV(1, J))
      ENDDO
CDIR$ DOSHARED(I)ON EV(I, 1)
      DO I=2, 32
        CALL SET_EVENT(EV(I, 1))
      ENDDO
CDIR$ DOSHARED(I, J)ON EV(I, J)
      DO I=2, 32
        DO J=2, 64
          CALL CLEAR_EVENT(EV(I, J))
        ENDDO
      ENDDO
```

图 4

其中 CLEAR－EVENT 表示清除事件（将事件变量清 0）。

一般情况下，如有如图 5 所示串行循环。

```
CDIR $ MASTER
      DO I1=l1, u1
        DO I2=l2, u2
          ……
            DO In=ln, un
              H (I1, I2, …, In)
            ENDDO
          ……
        ENDDO
      ENDDO
CDIR $ END MASTER
```

图 5

其中 l_j，$u_j(1\leqslant j\leqslant n)$为整常数，$H(I_1,I_2,\cdots,I_n)$为赋值语句，包含对共享数组的写操作。数组所有的下标表达式为 aI＋b 的形式，I 为循环控制变量，a，b 为整常数，I 最多出现在一个下标表达式中，所有的变量对之间只引起流依赖，设有 m 个依赖关系存在，相应的依赖距离向量分别为 $D_1=(d_{11},\cdots,d_{1n}),\cdots,D_m(d_{m1},\cdots,d_{mn})$，其中 $d_{ij}(1\leqslant i\leqslant m,\ 1\leqslant j\leqslant n)$也为整常数。通过加入事件同步，可将图 5 所示的串行循环转换为共享循环，如图 6 所示。

```
CDIR $ DOSHARED (I1, I2, …, In) ON array_ref
      DO I1=l1, u1
        DO I2=l2, u2
          ……
            DO In=ln, un
              CALL WAIT_EVENT(EV(I1－d11, …, In－d1n))
              ……
              CALL WAIT_EVENT(EV(I1－dm1, …, In－dmn))
              H (I1, I2, …, In)
              CALL SET_EVENT(EV(I1, …, In))
            ENDDO
          ……
        ENDDO
      ENDDO
CDIR $ END MASTER
```

图 6

其中，ON array-ref 为循环迭代划分机制，可以选择访问频度最高的共享数组访问模式作为 array-ref，编译器据此划分循环迭代空间。EV 为共享数组。设

$$min_j=\min(l_j,l_j-d_{1j},\cdots,l_j-d_{mj})$$

$$max_j=\max(u_j,u_j-d_{1j},\cdots,u_j-d_{mj})\ (1\leqslant j\leqslant n)$$

则 EV 在循环中被访问的下标集为$(min_1:max_1,\cdots,min_n:max_n)$。

EV 的分布方式和维下界根据数组访问模式 array-ref 确定。设 array-ref 为 $X(a_1I_1+b_1,\cdots,a_nI_n+b_n)$，数组 X 的分布方式为(BLOCK($M_1$),…,BLOCK($M_n$))，每维下界分别为 S_1，…，S_n，给每维分配的处理机数分别为 NP_1，…，NP_n，形成 $NP_1*\cdots*NP_n$ 的 n 维处理机空间，则 $X(a_1I_1+b_1,\cdots,a_nI_n+b_n)$ 所在的处理机元组为 $\left(MOD\left(\left[\frac{a_1I_1+b_1-s_1}{M_1}\right],NP_1\right),\cdots,MOD\left(\left[\frac{a_nI_n+b_n-s_n}{M_n}\right],NP_n\right)\right)$，设 EV 的分布方式为(BLOCK($N_1$),…,BLOCK($N_n$))，每维的下界分别为 t_1，…，t_n，则 $EV(I_1,\cdots,I_n$ 所在的处理机元组为 $\left(MOD\left(\left[\frac{I_1-t_1}{N_1}\right],NP_1\right),\cdots,MOD\left(\left[\frac{I_n-t_n}{N_n}\right],NP_n\right)\right)$，为了使 $X(a_1I_1+b_1,\cdots,a_nI_n+b_n)$ 与 $EV(I_1,\cdots,I_n)$ 在同一处理机上，令 $\frac{a_jI_j+b_j-s_j}{M_j}=\frac{I_j-t_j}{N_j}+k*NP_j$ ($1\leqslant j\leqslant n$，k 为任意整常数)

即 $\left(\frac{a_j}{M_j}-\frac{1}{N_j}\right)I_j=\frac{s_j-b_j}{M_j}-\frac{t_j}{N_j}+k*NP_j$

其中，只有 I_j 为变量，为了使$\forall I_j(I_j\in[l_j,u_j])$等式保持成立，有

$$\begin{cases}\frac{a_j}{M_j}-\frac{1}{N_j}=0\\ \frac{s_j-b_j}{M_j}-\frac{t_j}{N_j}+k*NP_j=0\end{cases}\quad\text{即}\quad\begin{cases}N_j=\frac{M_j}{a_j}\\ t_j=\frac{s_j-b_j}{a_j}+k*NP_j*N_j\end{cases}$$

对 N_j 和 t_j 取整，得到 EV 数组的分布方式为 $\left(BLOCK\left(\left[\frac{M_1}{a_1}\right]\right),\cdots,BLOCK\left(\left[\frac{M_n}{a_n}\right]\right)\right)$，以及 $t_j=\left[\frac{s_j-b_j}{a_j}\right]+k*NP_j*N_j$。为了保证 EV 数组在循环中被访问的每一维下限 min_j 不小于维下界 t_j，需要选择合适的 k，令 $min_j\geqslant t_j$，即 $\left[\frac{s_j-b_j}{a_j}\right]+k*NP_j*N_j\leqslant min_j$。得到 $k\leqslant\frac{min_j-\left[\frac{s_j-b_j}{a_j}\right]}{NP_j*N_j}$ 取 $k=\left[\frac{min_j-\left[\frac{s_j-b_j}{a_j}\right]}{NP_j*N_j}\right]$ 上式中，a_j、b_j、s_j 为已知数，min_j 和 N_j 在前面已经求出，NP_j 可根据用户申请的处理机数和数组 X 的分布方式确定[1]，因此 k 和 t_j 的值都可确定下来，EV 数组的维说明符分别为：$t_1:max_1$，…，$t_n:max_n$。

为了对事件数组 EV 进行初始化，将 EV 数组在循环中被访问的下标集分割成 2n+1 个部分：

$(min_1:l_1-1,\ min_2:max_2,\ \cdots,\ min_n:max_n)$

$(l_1:max_1,\ min_2:l_2-1,\ \cdots,\ min_n:max_n)$

…

$(l_1:max_1,\ l_2:max_2,\ \cdots,\ min_n:l_n-1)$

$(u_1+1:max_1,\ l_2:max_2,\ \cdots,\ l_n:max_n)$

$(l_1:u_1,\ u_2+1:max_2,\ \cdots,\ l_n:max_n)$

…

$(l_1:u_1,\ l_2:u_2,\ \cdots,\ u_n+1:max_n)$

$(l_1:u_1,\ l_2:u_2,\ \cdots,\ l_n:u_n)$

图 7

其中最后一个部分为 $EV(I_1,\cdots,I_n)$对应循环迭代空间的下标集，因此该部分初始化为“未发生”状态，其余部分初始化为“发生”状态。

2 性能分析

下面对图 1 所示的串行循环及其优化以后的循环（图 3 和图 4）进行性能分析。在分析中主要考虑数据访存时间和优化后增加的同步开销。处理机的计算时间在优化前后相同，不予考虑。

在图 1 中，所有的迭代由 PE_0 执行，A(I,J)被访问（包括读和写）的下标集为(2：32，2：64)，其中局部下标集为(2：16，2：32)，局部访问次数为 465，非局部访问次数为 1488。同理，A(I−1,J)和 A(I,J−1)的局部访问次数分别为 496，480，非局部访问次数分别为 1457，1473。所以串行循环中总的局部访问次数为 1906，总的非局部访问次数为 5906。

串行循环经过优化以后（见图 3），迭代空间按 A(I,J)划分，即每个 PE 访问 A(I,J)时都是局部的。A(I−1,J)的非局部访问为：PE_1 访问 A(16，2：32)，PE_3 访问 A(16，33：64)，A(I,J−1)的非局部访问为：PE_2 访问 A(2：16，32)，PE_3 访问 A(17：32，32)，所以对数组 A 的非局部访问次数总计为 94，而局部访问次数总计为 7718。

在优化后的循环中，由于引入了同步机制，所以增加了同步开销。实际上，同步调用过程的处理就是对事件变量进行设置、清除或判断，经过过程嵌入后，相当于一次访存操作。在图 3 中，对事件数组 EV 的局部访问次数为 5765，非局部访问次数为 94。在图 4 中，所有访问都是局部的，访存次数为 2048，由于该部分的循环无跨迭代的依赖关系，4 个处理机可同时执行各自的迭代，所以该部分的实际开销为 512 次局部访存操作。EV 数组引入的同步开销总计为 6277 次局部访存，94 次非局部访存。

假定一次非局部数据访问的时间为 T_r，而一次局部数据访问的时间为 T_l，则优化前串行循环总的访存时间为 $5906T_r+1906T_l$，优化后 A 数组的访存时间为 $94T_r+7718T_l$，引入的同步开销为 $94T_r+6277T_l$，总计为 $188T_r+13995T_l$。

对于大规模并行处理系统而言，局部数据访问和非局部数据访问所需的时间相差很大，一般在一个数量级以上。假定 $T_l=T_r/10$，则优化前的总开销为$6096.6T_r$，优化后的总开销为$1587.5T_r$，约为优化前的 26%。

3 小 结

本文提出的串行循环优化技术对于提高 MPP Fortran 程序在大规模并行处理机上的运行性能具有显著的效果。在图 6 中，设 $d_{max_j}=\max(|d_{1j}|,\cdots,|d_{mj}|)(1\leqslant j\leqslant n)$，对于每个处理机上的分布块(大小为 $M_1*\cdots*M_n$)而言，非局部访问所涉及的元素最多为 $M_1*\cdots*M_n-(M_1-2d_{max_1})*,\cdots,*(M_n-2d_{max_n})$，当对$\forall j$ 有 $d_{max_j}\ll M_j$ 时，非局部访问涉及的元素个数远远小于分布块大小，此时优化效果最为显著。

当依赖距离向量与依赖变量对为一对多的关系时，同步调用语句不随依赖变量对增加，同步开销所占比例相对减小，优化效果相应提高。

对于共享数组某一维不分布的情况，如果该维对应的那层循环在最外层或最内层

(否则可尝试循环交换)，可以将该层循环排除在共享循环之外，计算依赖距离向量时也可不考虑该层循环，EV 数组也相应减少一维，从而进一步减少通讯开销和同步开销。

当串行循环中含有多个共享数组时，如果共享数组之间都是对准的（维数、每维大小和分布方式完全相同)，则串行循环的优化效果和优化过程与含单个共享数组的情况完全相同。否则，可尝试循环分布等技术，将串行循环裂解为多个循环，使其中某些循环或所有循环只含一个共享数组或只含对准的共享数组，再对它们进行优化。

总之，在 MPP Fortran 程序中，通过加入同步控制，将串行循环转换为共享循环是一项十分有效的优化技术，对提高并行程序在大规模并行系统上的运行性能具有重要的意义。

参 考 文 献

1 Douglas M. Pase, Tom Mcdonald, Andrew Meltzer. MPP Fortran Programming Model. Cray Research, Inc. Eagan, Minnesota, 1993, 2

2 Utpal Banerjee. Dependence Analysis for Supercomputing. Kluwer Academic Publishers, 1988

3 Ron Cytron, Jeanne Ferante. What's in a Name?or The Value of Renaming for Parallelism Detection and Storage Allocation. In Proc. 1987 International Conf. on Parallel Processing, August 1987: 19～27

4 Wolfe M J. Optimizing Supercompilers for Supercomputers, Ph. D Thesis, University of Illinois at Urbana-Champaign, DCS Report No. UIUCCDCS-R-82-1105, 1982

(责任编辑　张　静)