装订线 答题时不要超过此线

中 国 科 学 技 术 大 学

2010－2011 学年第一学期考试试卷

参考解答

考试科目：并行程序设计　　得分:_____

学生所在系:_____　姓名:_____　学号:_____

本试卷共五个大题！

一、（1）针对如下双重循环：

```
for I = 1 to  5 do
   for J = 1 to 5 do
S:    A(I, J) = A(I-1, J) + A(I, J-1)
   endfor
endfor
```

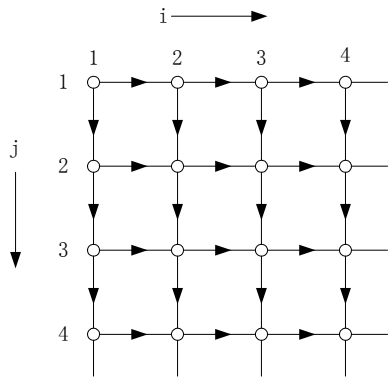
给出该双重循环的迭代依赖图；（10 分）

分析该双重循环向量化或并行化的情况（5 分）。

参考解答：

原双重循环中存在依赖方向向量为（1，0）和（0，1）的流依赖关系。因此，

迭代依赖图如下（部分）：



由上面的迭代依赖图可看出：无论是外层 i 循环，还是内层 j 循环均可携带跨迭代的依赖关系。故而双重循环的每一层均不能实施并行化。

由（0，1）的 SδS 流依赖关系知，内层循环 j 不能实施向量化！

（2） 若把（1）中的双重循环改为：

```
for I = 1 to  5 do
   for J = I+1 to I+5 do
```

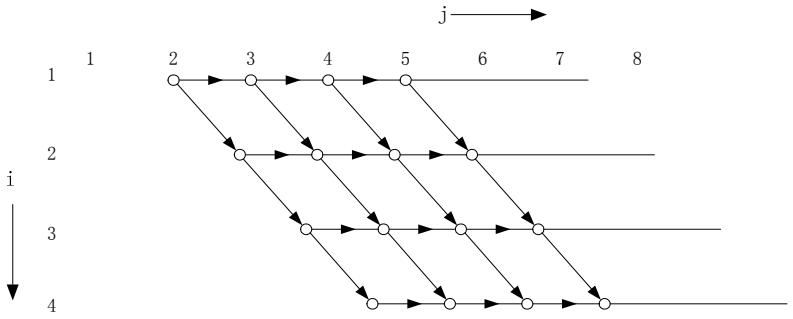
```
S:   A(I, J-I) = A(I-1, J-I) + A(I, J-I-1)
   endfor
endfor
```

试给出修改后的双重循环的迭代依赖图。（10 分）

分析该修改后的双重循环并行化的情况。如能实施并行，请给出相应的 OpenMP 实现。（5 分）。

参考解答：

修改后的循环，就是对原来循环做了拉伸系数为 1 的变换。变换后，迭代依赖图示意如下：（部分）



由上图可见，对相同 *j* 值，沿 *i* 方向的计算无相关关系，因此交换 *i*，*j* 循环先后次序，*i* 循环可并行执行。即，经过拉伸变换，修改后的双重循环中的依赖关系为：（1，1）和（0，1）。通过循环交换，可以得到（1，1）和（1，0）依赖关系，这样，交换到外层的循环 j 依然不可并行，但交换到内层的循环 i 则可以实施并行！

修改后得到并行程序如下：

```
for j=2 to 10 do
   for i=max(1, j - 5 ) to min(5, j - 1 ) par-do
      a[i, j - i ] = a[i-1, j - i ] + a[i, j - i - 1]
   endfor
endfor
```

其中上述程序的 par-do 部分，可以采用 OpenMP 的并行域编译制导来进行多线程并行化！

```
for( j=2;j<=10;j++)
{
  #pragma omp parallel for private(i)
  for (i=max(1, j - 5 );i<= min(5, j - 1 );i++)
     a[i][j - i ] = a[i-1][j - i ] + a[i][j - i - 1];
}
```

二、 针对数组 double A[100]，设计两种不同 MPI 实现，均可一次性发送数组 A 中所有奇数编号的元素。（15 分）

参考解答：

（1） MPI_Pack 函数调用方式：

```
double A[100];
MPI_Pack_size (50,MPI_DOUBLE,comm,&BufferSize);
TempBuffer = malloc(BufferSize);
j = sizeof(MPI_DOUBLE);
Position = 0;
for (i=0;i<50;i++)
    MPI_Pack(A+1+i*j,1,MPI_DOUBLE,TempBuffer,BufferSize,&Position,comm);
MPI_Send(TempBuffer,Position,MPI_PACKED,destination,tag,comm);
```

（2） MPI_Type_vector 构造新类型：

```
double A[100];
MPI_Datatype EvenElements;
...
MPI_Type_vector(50, 1, 2, MPI_DOUBLE, &EvenElements);
MPI_Type_commit(&EvenElements);
MPI_Send(A+1, 1, EvenElements, destination, ···);
```

三、向量化以下循环：（10 分）

```
for I = 2 to N do
    S1:   A(I) = B(I) + C(I+1)
    S2:   D(I) = A(I+1) + 1
    S3:   C(I) = D(I)
Endfor
```

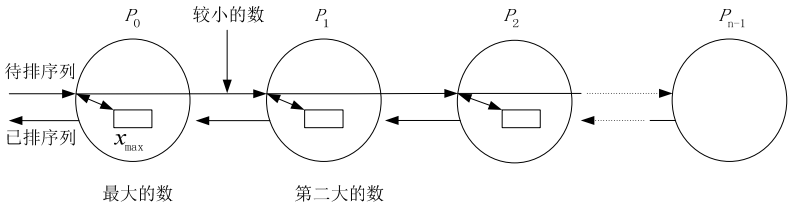
参考解答：

上述循环中依赖关系为： S2 δ^a S1 S1 δ^a S3 和 S2 δ^f S3，其中依赖关系：S2 δ^a S1 使得循环不能向量化。

考虑语句重排的循环变换。则循环中语句次序由原来的 S1－>S2->S3 变为：S2－>S1－>S3。则变换后，循环可向量化如下：

```
S2:   D(2:N) = A(3:N+1) + 1
S1:   A(2:N) = B(2:N) + C(3:N+1)
S3:   C(2:N) = D(2:N)
```

四、**流水线（Pipline）技术可用于实现插入排序**：让一组（n 个）进程中的第一个 p_0 每次接受要排序的 n 个数中的一个，保存当前接受到的最大的数字且把比这个数小的其它数传递给下一个进程。如果输入的数比当前保存的数大，就把当前的数传递给下一个进程，把输入的数保存为当前的最大数。每个后继进程都执行同样的算法，当所有的数都处理完之后，p_0 就保存有最大的数字，p_1 中保存有第二大的数字，p_2 中保存有第三大的数字，以此类推。显然，第 i 个进程要从上一个进程接受 n-i 个数，另外它要向下一进程传送 n-i-1 个数，而且当待排序的最后一个数通过它被传递时,它就可以返回结果给上一进程，即该进程不必等到所有数都被排序。最后被排好序的数字序列将由进程 p_0 处依次得到。下图给出了这种方法插入排序的流程：



请给出上述并行插入排序的**进程 P_i 的伪代码实现**。（20 分）

参考解答：

基于流水线技术的插入排序算法（进程 P_i 的伪代码）实现如下：

输入：n 个无序的数
输出：n 个有序的数
Begin
(1) 从进程 p_{i-1} 接受一个数 x 为进程 i 中的数
(2) **for** j=0 **to** n-i-2 **do**　　/* 向下传递数据 */
　(2.1)从进程 p_{i-1} 接受一个数 *number*
　(2.2)**if** (*number*>x) **then**
　　(ⅰ) 将 x 送到下一进程 p_{i+1}
　　(ⅱ) 接受 *number* 为进程 i 中的数
　　else
　　(ⅲ) 将 *number* 送到下一进程 p_{i+1}
　　end if
　end for
(3)将数 x 返回给上一进程 p_{i-1}
(4)**for** j=0 **to** n-i-2 **do**
　(4.1)从下一进程 p_{i+1} 接受数 *number*

(4.2)将 *number* 返回给上一进程 p_{i-1}

end for

End

注：对于第一个进程 P_0，则：
步骤（1）和（2.1）表示从标准输入（或文件、或其他存放 n 个待排序数据的场所中）读取一个数据；
步骤（3）和（4.2）则表示将一个数据写入标准输出（或文件，或存放 n 个最终排序结果的场所中）。

五、 串行的矩阵-向量乘法如下所示。现在考虑并行计算并假设：

（1）有 p 个处理器（p 为平方数）且划分为 $\sqrt{p}\times\sqrt{p}$ 的二维处理器网格；

（2）n 阶方阵 A 分成 p 个方块 $A_{ij}(0\le i,j\le\sqrt{p}-1)$，每块大小为 $\lceil n/\sqrt{p}\rceil\times\lceil n/\sqrt{p}\rceil$，并将它们分配给 $\sqrt{p}\times\sqrt{p}$ 个处理器 $(P_{00},P_{01},...,P_{\sqrt{p}-1\sqrt{p}-1})$。

（3）列向量 B 被划分 $\lceil n/\sqrt{p}\rceil$ 大小的 $\lceil\sqrt{p}\rceil$ 个子向量并分布在最后一列处理器中。

请给出上述假设下的矩阵-向量乘法的 MPI 实现。（不用写数据分发过程，但需给出最终结果向量的收集过程。）（25 分）

参考解答：

并行实现的基本思想：

（1） 首先，最后一列处理器将所持有的列向量片段发送给它们所在处理器行的主对角线上的处理器；
（2） 其次，主对角线上的处理器将收到的列向量片段广播到它所在的处理器列上的所有处理器中；
（3） 所有处理器将本地 A 分块子矩阵与所收到的列向量片段进行矩阵-向量乘法；
（4） 对步骤（3）中得到的矩阵-向量乘积，按照处理器行进行归约累和，结果存放在各个处理器行的首列处理器上；
（5） 对步骤（4）中所得结果，进行收集，最终结果收集在编号（0，0）的进程中【即，主进程】。
（6） 并行算法结束。

主要的 MPI 实现如下：

```
MPI_Init( &argc, &argv );
MPI_Comm_size( MPI_COMM_WORLD, &nproc );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );
p = (int) sqrt( nproc );
dim = N / p;
```

```
// 以下将进程 rank 映射为二维拓扑坐标
 row = rank / p;  col = rank % p;

  （0）
// 然后形成处理器行-列通信子域
// 首先，形成处理器行通信子域
 color = row; key = col;
MPI_Comm_split(MPI_COMM_WORLD,Color,Key,&RowComm);
// 再形成处理器列通信子域
 color = col; key = row;
MPI_Comm_split(MPI_COMM_WORLD,Color,Key,&ColComm);
```

（1）首先，最后一列处理器将所持有的列向量片段发送给它们所在处理器行的主对角线上的处理器；

```
if( col == p - 1 ){ // 进程位于最后一列上；
        MPI_Send(sub_B, dim, MPI_DOUBLE, row*P+row, 999,
                 MPI_Comm comm）;
    }
else if( row == col ){
        MPI_Recv(sub_B, dim, MPI_DOUBLE, row*p+p-1, 999,
                 MPI_COMM_WORLD, &status);
     }
```

（2）其次，主对角线上的处理器将收到的列向量片段广播到它所在的处理器列上的所有处理器中；

```
// 在处理器列 col 中，主对角线上的处理器（进程）坐标为（col,col）。
MPI_Bcast( sub_B,dim, MPI_DOUBLE, col, ColComm );
```

（3）所有处理器将本地 A 分块子矩阵与所收到的列向量片段进行矩阵-向量乘法；

```
for(i=0;i<dim;i++){
   c[i] = 0.0;
   for(j=0;j<dim;j++)
     c[i] = c[i] + sub_A[i][j] * sub_B[j]
}
```

（4）对步骤（3）中得到的矩阵-向量乘积，按照处理器行进行归约累和，结果存放在各个处理器行的首列处理器上；

```
// 在各个 RowComm 通信子域中，首列处理器的编号即为 0；
MPI_Reduce(result_c, c, dim, MPI_DOUBLE, MPI_SUM, 0, RowComm);
```

（5）对步骤（4）中所得结果，进行收集，最终结果收集在编号（0，0）的进程中【即，主进程】。

```
if( col == 0 ) //在首处理器列（所在通信子域）进行最终结果的收集！
MPI_Gather(result_c, dim, MPI_DOUBLE, final_c, dim, MPI_DOUBLE,
```

```
            0, ColComm );
```

（6）并行计算结束！