MPI 上机实验

（a） （1.1）写个将 MPI 进程按其所在节点分组的程序；（1.2）在 1.1 的基础上，写个广播程序，主要思想是：按节点分组后，广播的 root 进程将消息“发送”给各组的“0 号”，再由这些“0”号进程在其小组内执行 MPI_Bcast。

```
MPI_Comm_split(MPI_COMM_WORLD, id_procs % 4, id_procs, &split_comm_world);
```

以节点模4值分组split_comm_world，再以新组的0号进程分组zero_comm，并把root进程加入这个分组zero_comm中。

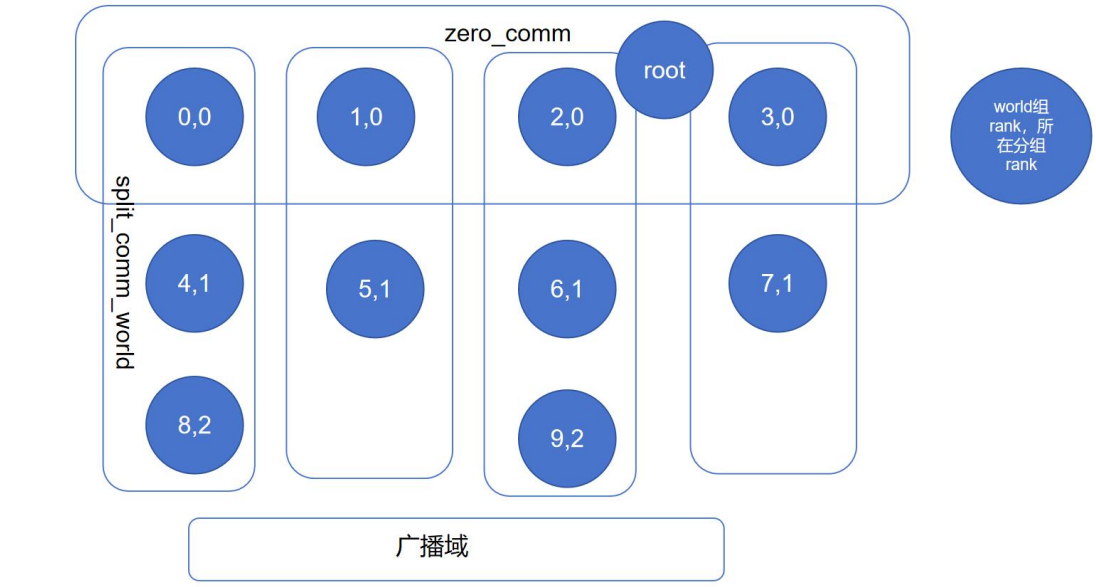
```
MPI_Comm_rank(split_comm_world, &rank);
MPI_Comm_split(MPI_COMM_WORLD, (rank == 0 || root == id_procs), id_procs,
&zero_comm);
```

root进程在zero_comm中广播发送到所有组的零号进程

```
MPI_Bcast(&buf, 16, MPI_CHAR, root, zero_comm);
```

各零号进程分别在自己的组内广播

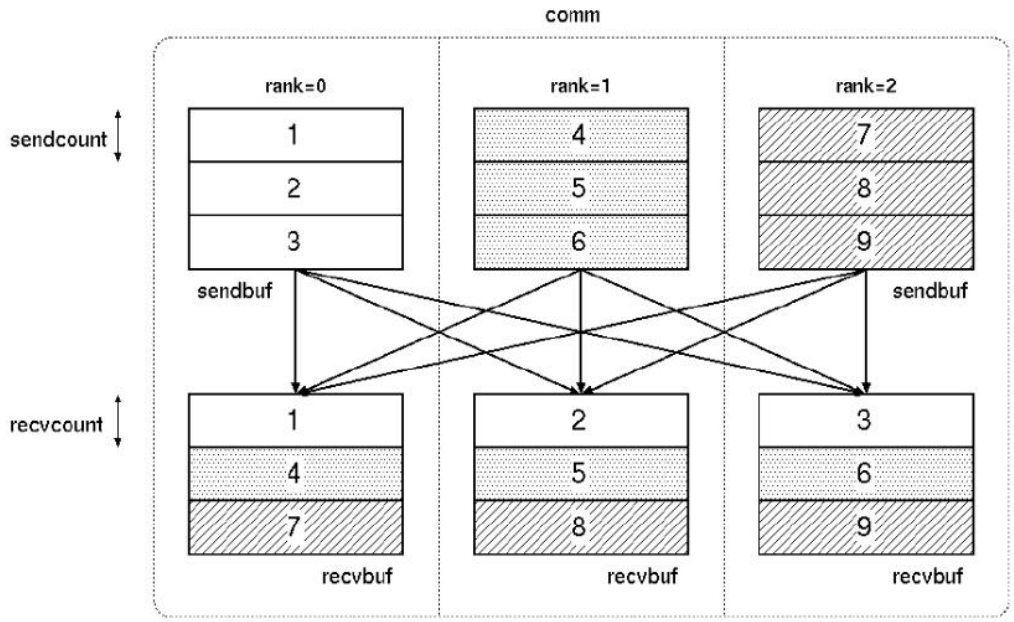
```
MPI_Bcast(&buf, 16, MPI_CHAR, 0, split_comm_world);
```




```
[pp24@node1 repo]$ mpirun -np 10 laba
MPI Comm rank 0, node id 0
MPI Comm rank 1, node id 0
MPI Comm rank 2, node id 0
MPI Comm rank 3, node id 0
MPI Comm rank 4, node id 1
MPI Comm rank 5, node id 1
MPI Comm rank 6, node id 1
MPI Comm rank 7, node id 1
MPI Comm rank 8, node id 2
MPI Comm rank 9, node id 2
MPI Comm rank 0, node id 0, buf: wake
MPI Comm rank 1, node id 0, buf: wake
MPI Comm rank 2, node id 0, buf: wake
MPI Comm rank 3, node id 0, buf: wake
MPI Comm rank 4, node id 1, buf: wake
MPI Comm rank 5, node id 1, buf: wake
MPI Comm rank 6, node id 1, buf: wake
MPI Comm rank 7, node id 1, buf: wake
MPI Comm rank 8, node id 2, buf: wake
MPI Comm rank 9, node id 2, buf: wake
[pp24@node1 repo]$
```

（b） 使用 MPI_Send 和 MPI_Recv 来模拟 MPI_Alltoall。将你的实验与相关 MPI 通信函数做评测和对比。

流程图：



自定义Alltoall实现：

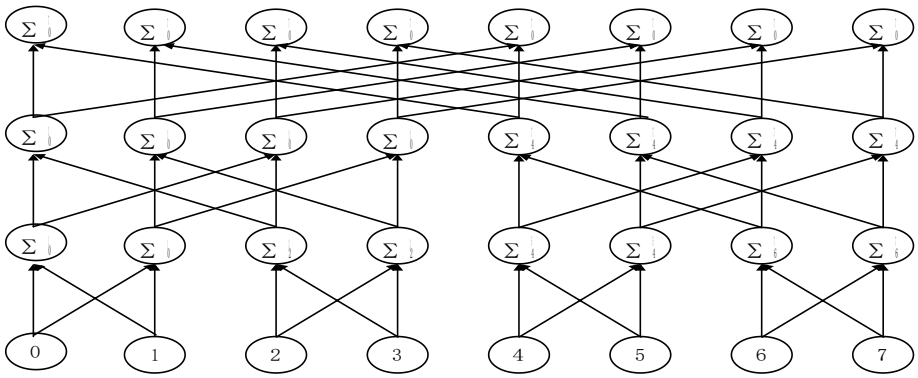
```
int My_Alltoall(void *sendbuf, int sendcount, MPI_Datatype sendtype,
                void *recvbuf, int recvcount, MPI_Datatype recvtype,
                MPI_Comm comm) {
    int world_size, world_rank;
    MPI_Comm_size(MPI_COMM_WORLD, &world_size);
    MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

    int sendtype_size, recvtype_size;
    MPI_Type_size(sendtype, &sendtype_size);
    MPI_Type_size(recvtype, &recvtype_size);

    for (int i = 0; i < world_size; ++i) {
        if(i != world_rank){
            MPI_Send(sendbuf + i * sendtype_size * sendcount, sendcount, sendtype,
                    i, 0, comm);
            MPI_Recv(recvbuf + i * recvtype_size * recvcount, recvcount, recvtype,
                    i, 0, comm, MPI_STATUS_IGNORE);
        }
        else{
            memcpy(recvbuf + i * recvtype_size * recvcount, sendbuf + i * sendtype_size * sendcount, sendtype_size * sendcount);
        }
    }


    return 0;
}
```

执行10000次16个进程相互通讯比较性能测试结果：

```
[pp24@node1 repo]$ mpirun -np 16 labb
My_Alltoall耗时: 0.256907
MPI_Alltoall耗时: 0.165811
```

（c） N 个处理器求 N 个数的全和，要求每个处理器均保持全和。

（1）蝶式全和的示意图如下：由于使用了重复计算，共需 logN 步。

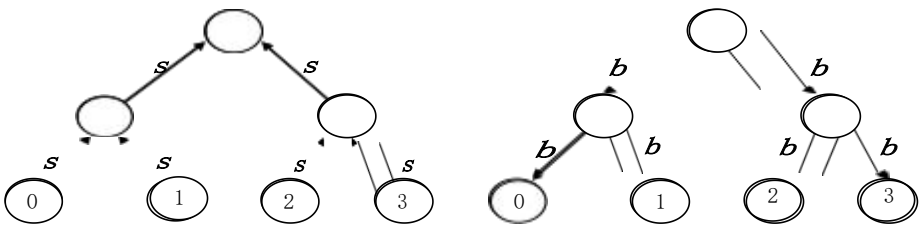


给出蝶式全和计算的 MPI 程序实现（设 N 为 2 的幂次方）。
算法描述如题图。

```
对于 step = 0 到 log2(p) - 1:
    1. 计算通信目标进程:
       partner = rank XOR (1 << step)

    2. 与目标进程交换数据:
       发送 global_sum 到 partner
       接收 partner 的 global_sum

    3. 合并数据:
       global_sum = global_sum + partner 的 global_sum
```

```
Rank 1, Global sum: 36
Rank 2, Global sum: 36
Rank 3, Global sum: 36
Rank 4, Global sum: 36
Rank 5, Global sum: 36
Rank 6, Global sum: 36
Rank 7, Global sum: 36
Rank 0, Global sum: 36
碟式全和耗时: 0.000164
```

（2）二叉树方式求全和示意图如下：需要 2logN 步。



给出二叉树方式全和计算的 MPI 程序实现。

```
对于 step = 1 到 log2(p):
    1. 计算父进程:
       parent = rank - (1 << (step - 1)), 如果 rank 可被 2^step 整除

    2. 如果 rank 是接收进程:
       接收子进程的数据, 累加到 global_sum
    3. 如果 rank 是发送进程:
       发送 global_sum 到 parent
对于 step = log2(p) 到 1:
    1. 计算子进程:
       child = rank + (1 << (step - 1)), 如果 rank + 2^(step-1) < p

    2. 如果 rank 是发送进程:
       发送 global_sum 到子进程
    3. 如果 rank 是接收进程:
       接收 global_sum
```

```
Rank 0, Global sum: 36
二叉树全和耗时: 0.000089
Rank 1, Global sum: 36
Rank 2, Global sum: 36
Rank 3, Global sum: 36
Rank 4, Global sum: 36
Rank 5, Global sum: 36
Rank 6, Global sum: 36
Rank 7, Global sum: 36
```

（d） 《并行算法实践》单元 V 习题 v-3。给出 FOX 矩阵相乘并行算法的 MPI 实现。

算法描述：

对于 $k = 0$ 到 $\sqrt{p} - 1$，执行以下步骤：

1. 广播 A 的子块

   在每行中广播 $A_{i,(j+k) \mod \sqrt{p}}$，并将其存储为 A_{current}。

2. 执行局部计算

   每个处理器计算：

$$C_{ij} = C_{ij} + A_{\text{current}} \times B_{ij}$$

3. 循环移动 B 的子块

   每列中的处理器将 B_{ij} 向下发送到下一个处理器：

$$P(i,j) \to P((i+1) \mod \sqrt{p}, j)$$

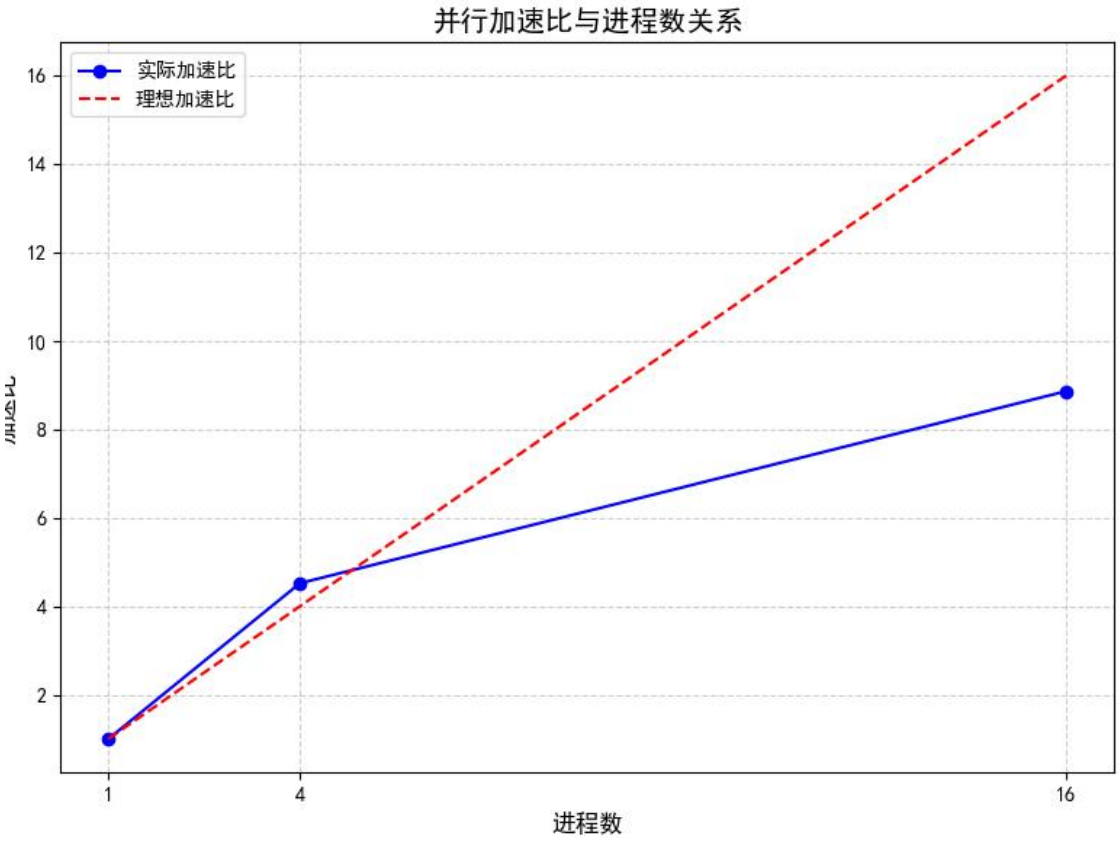
加速图表如下，矩阵大小为2048x2048：

4核：

```
[pp24@node1 repo]$ mpirun -np 4 labd
fox并行乘法耗时: 100.088921

串行乘法耗时: 452.144938
并行4核加速比为: 4.517432
```

16核：

```
[pp24@node1 repo]$ mpirun -np 16 labd
fox并行乘法耗时: 51.166527
串行乘法耗时: 495.610342
并行16核加速比为: 9.686222
```



（e） 参数服务器系统的 MPI 模拟。

设系统中总计有 N 个进程，其中 P 个进程作为参数*服务器进程*，而 Q 个进程作为*工作进程*（N= P + Q， 且 0 < P << Q）。工作进程和服务器进程的互动过程如下：

1. 第 i 个工作进程首先产生一个随机数，发送给第 i%P 个参数服务器进程。然后等待并接收它对应的参数服务器进程发送更新后的数值，之后，再产生随机数，再发送……。
2. 每个参数服务器进程等待并接收来自它对应的所有工作进程的数据，在此之后，经通信，使所有的参数服务器获得所有工作进程发送数据的平均值。
3. 每个参数服务器发送该平均值给它对应的所有工作进程，然后再等待 ……。

试给出上述互动过程的 MPI 程序实现。

（f） 矩阵 A 和B 均为 N*N 的双精度数矩阵，有 P 个处理器。针对以下程序片段，分别采用<u>按行块连续划分</u>以及<u>棋盘式划分</u>方式，给出相应的 MPI 并行实现。

```
for(i=1; i<N-1; i++)
    for( j=1;j<N-1; j++)
        B[i][j] = (A[i-1][j] + A[i][j+1] + A[i+1][j] + A[i][j-1]) / 4.0
```

有两种实现思路，一种是将A作为共享内存，这样直接划分好每个进程计算的B区块最后收集即可完成计算，另一种就是A，B都需要进行划分，这样在计算的时候仍然需要通信，这里选择后者实现。因为前者的实现太过于简单。

按行块连续划分：

1. **计算每个进程负责的行范围：**
2. **分配局部矩阵 `local_A` 和 `local_B`**
3. **主进程将 A 按行块划分，并分发给各个进程。**
4. **每个进程执行以下操作：**

- 如果有上邻居或下邻居，交换邻近行。
- 对于行范围 `[start_row:end_row]` 中的每一行 `i`，以及列范围 `[1:N-1]` 中的每一列 `j`，计算：

```
B[i][j] = (A[i-1][j] + A[i][j+1] + A[i+1][j] + A[i][j-1]) / 4.0
```

6. **收集每个进程计算的结果到主进程。**
7. **主进程输出结果矩阵 B。**
8. **结束 MPI**，终止 MPI 执行。

实验结果：

```
按行块划分耗时: 0.019592
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

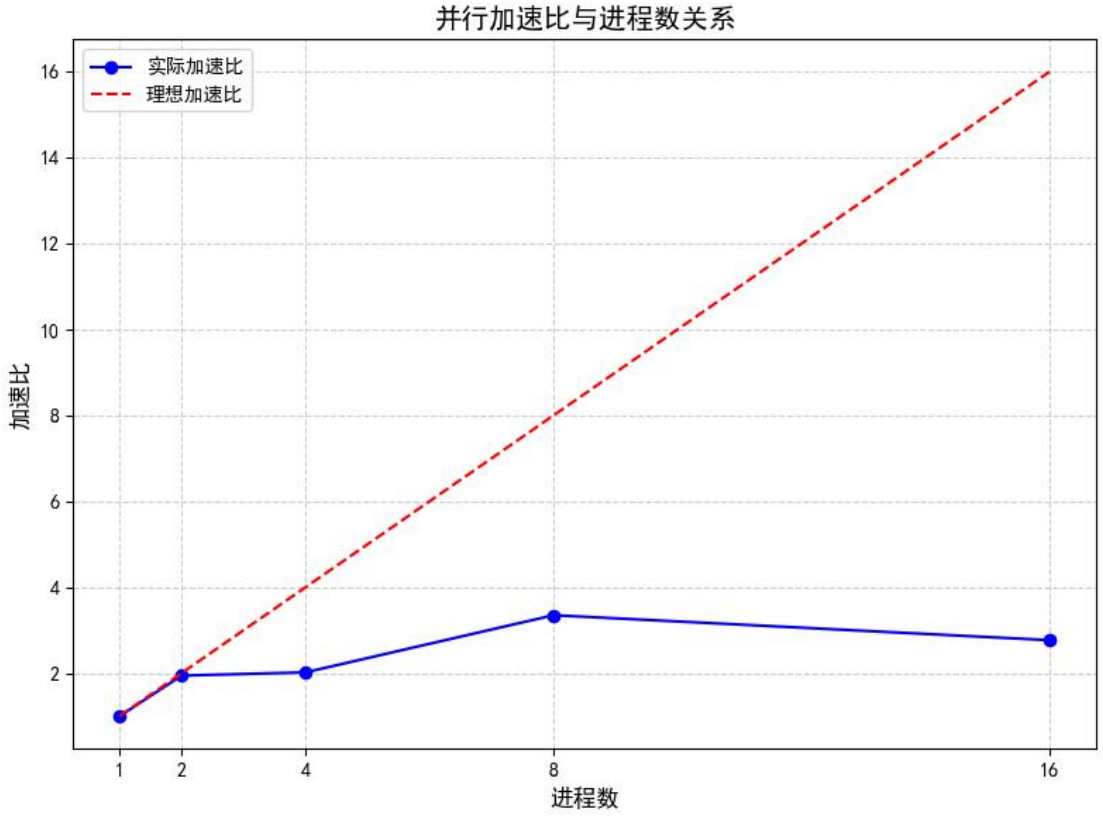
性能测评：（2048x2048）

```
[pp24@node1 repo]$ mpirun -np 1 labf1
Matrix update completed.
按行块划分1核耗时: 0.462738
```

```
[pp24@node1 repo]$ mpirun -np 2 labf1
Matrix update completed.
按行块划分2核耗时: 0.238548
[pp24@node1 repo]$ mpirun -np 4 labf1
Matrix update completed.
按行块划分4核耗时: 0.229008
[pp24@node1 repo]$ mpirun -np 8 labf1
Matrix update completed.
按行块划分8核耗时: 0.138298
```

```
[pp24@node1 repo]$ mpirun -np 16 labf1
Matrix update completed.
按行块划分16核耗时: 0.167009
```

加速比：



棋盘式划分：

1. **计算进程网格尺寸** `q x q`，其中 `q = sqrt(size)`，每个进程对应一个二维坐标 `(proc_row, proc_col)`。
2. **确定每个子块的尺寸：**
3. **分配局部矩阵** `local_A` **和** `local_B`，包括行和列。
4. **主进程将矩阵 A 按棋盘方式划分，并分发给各个进程**（使用 Scatter）。
5. **每个进程执行以下操作：**

- 与上、下、左、右邻居交换行/列数据（使用 Send/Recv）。
- 对于局部块范围中的每个元素 `(i, j)`，计算：

```
B[i][j] = (A[i-1][j] + A[i][j+1] + A[i+1][j] + A[i][j-1]) / 4.0
```

6. **收集所有进程的局部计算结果到主进程**（使用 Gather）。
7. **主进程输出结果矩阵 B。**
8. **结束 MPI，**终止 MPI 执行。

```
[pp24@node1 repo]$ mpirun -np 4 labf2
Matrix update completed.
棋盘划分4核耗时: 0.255350
[pp24@node1 repo]$ mpirun -np 16 labf2
Matrix update completed.
棋盘划分16核耗时: 0.131030
```

加速比：