并行与分布式计算作业

第6次作业

姓名：郝裕玮

班级：计科1班

学号：18329015

**对于问题1：**

一、问题描述

找出给出的7个MPI程序中的bug，并将程序修改正确，比较修改前后的程序运行结果。 MPI程序见附件。

二、解决方案&运行结果

（1）对于mpi\_bug1.c：

错误原因：0号进程和1号进程互发消息时，Send和Recv函数的tag不匹配。

该程序错误部分如下：

① 29-31行：

  rc = MPI\_Send(&outmsg, 1, MPI\_CHAR, dest, tag, MPI\_COMM\_WORLD);

  printf("Sent to task %d...\n",dest);

  rc = MPI\_Recv(&inmsg, 1, MPI\_CHAR, source, tag, MPI\_COMM\_WORLD, &Stat);

应修改为：

  rc = MPI\_Send(&outmsg, 1, MPI\_CHAR, dest, 0, MPI\_COMM\_WORLD);//修改处1

  printf("Sent to task %d...\n",dest);

  rc = MPI\_Recv(&inmsg, 1, MPI\_CHAR, source, 1, MPI\_COMM\_WORLD, &Stat);//修改处2

② 39-41行：

  rc = MPI\_Recv(&inmsg, 1, MPI\_CHAR, source, tag, MPI\_COMM\_WORLD,

&Stat);

  printf("Received from task %d...\n",source);

  rc = MPI\_Send(&outmsg, 1, MPI\_CHAR, dest, tag, MPI\_COMM\_WORLD);

应修改为：

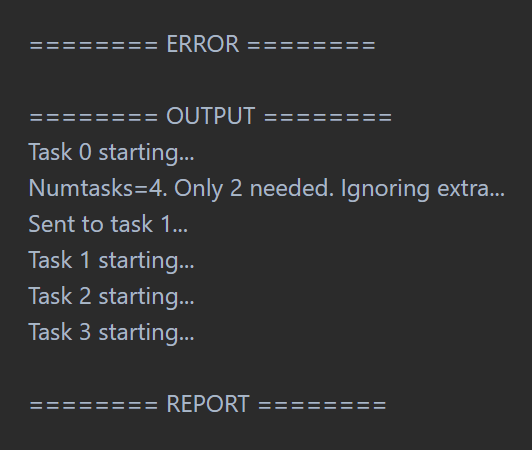
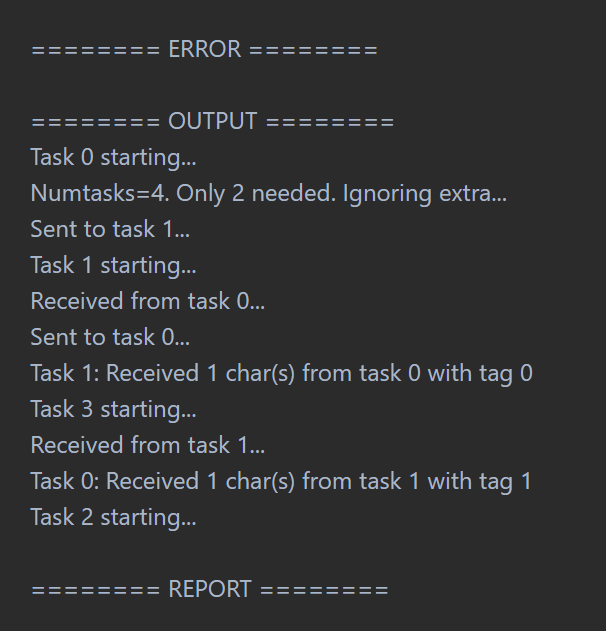
  rc = MPI\_Recv(&inmsg, 1, MPI\_CHAR, source, 0, MPI\_COMM\_WORLD,

&Stat);//修改处3

  printf("Received from task %d...\n",source);

  rc = MPI\_Send(&outmsg, 1, MPI\_CHAR, dest, 1, MPI\_COMM\_WORLD);//修改处4

超算习堂上修改前后运行结果如下所示（运行环境：4核）：

修改后0号进程和1号进程均收到了对方的消息并进行了回复。

（2）对于mpi\_bug2.c：

错误原因：发送方和接收方的数据类型不匹配（发送方是int，接收方是float）。

该程序错误部分如下：

① 16行：

float beta;

应修改为：

int beta;//修改处1

② 37-39行：

    MPI\_Irecv(&beta, 1, MPI\_FLOAT, 0, tag, MPI\_COMM\_WORLD, &reqs[i]);

    MPI\_Wait(&reqs[i], &stats[i]);

    printf("Task %d received = %f\n",rank,beta);

应修改为：

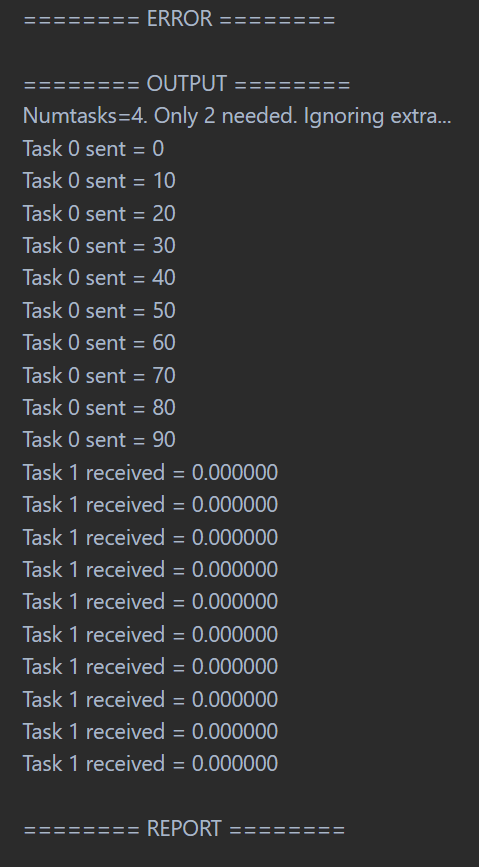
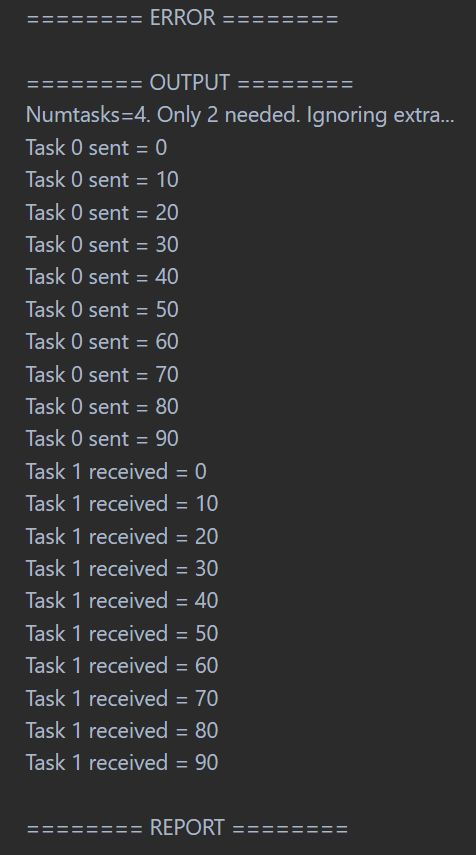
MPI\_Irecv(&beta, 1, MPI\_INT, 0, tag, MPI\_COMM\_WORLD, &reqs[i]);//修

改处2

    MPI\_Wait(&reqs[i], &stats[i]);

    printf("Task %d received = %d\n",rank,beta);//修改处3

超算习堂上修改前后运行结果如下所示（运行环境：4核）：

修改后1号进程收到了正确的数据。

（3）对于mpi\_bug3.c：

错误原因：MPI 环境未进行初始化和终止。需要添加 MPI\_Init和 MPI\_Finalize。

该程序错误部分如下：

① 24-25行：

/\*\*\*\*\* Initializations \*\*\*\*\*/

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numtasks);

应修改为：

/\*\*\*\*\* Initializations \*\*\*\*\*/

MPI\_Init(&argc, &argv);//修改处1

MPI\_Comm\_size(MPI\_COMM\_WORLD, &numtasks);

② 106-108行：

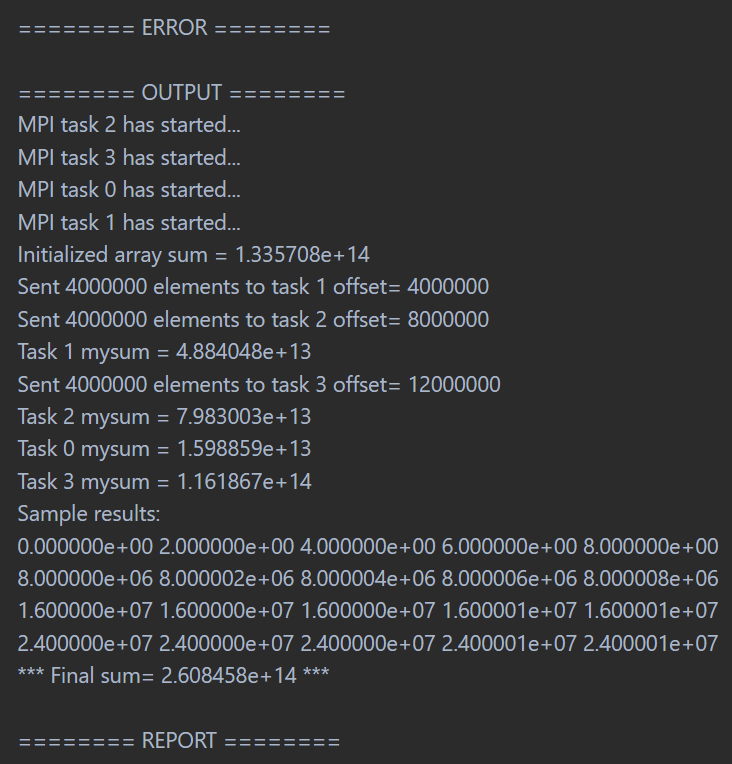
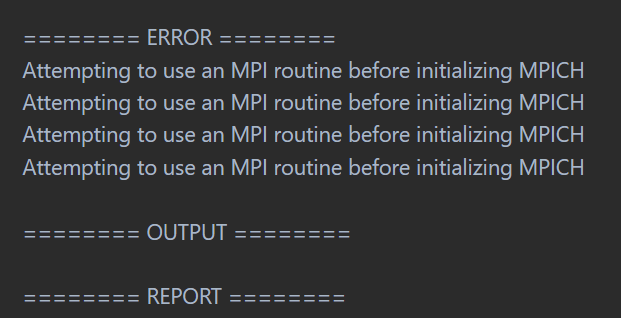
}   /\* end of main \*/

应修改为：

MPI\_Finalize();//修改处2

}   /\* end of main \*/

超算习堂上修改前后运行结果如下所示（运行环境：4核）：



修改后程序正常运行。

（4）对于mpi\_bug4.c：

错误原因：也应该在 Master 任务中也应当调用MPI\_Reduce，否则 Master 任务中的总和不会被累积。

该程序错误部分如下：

① 69-71行：

    }

  /\* Get final sum and print sample results \*/

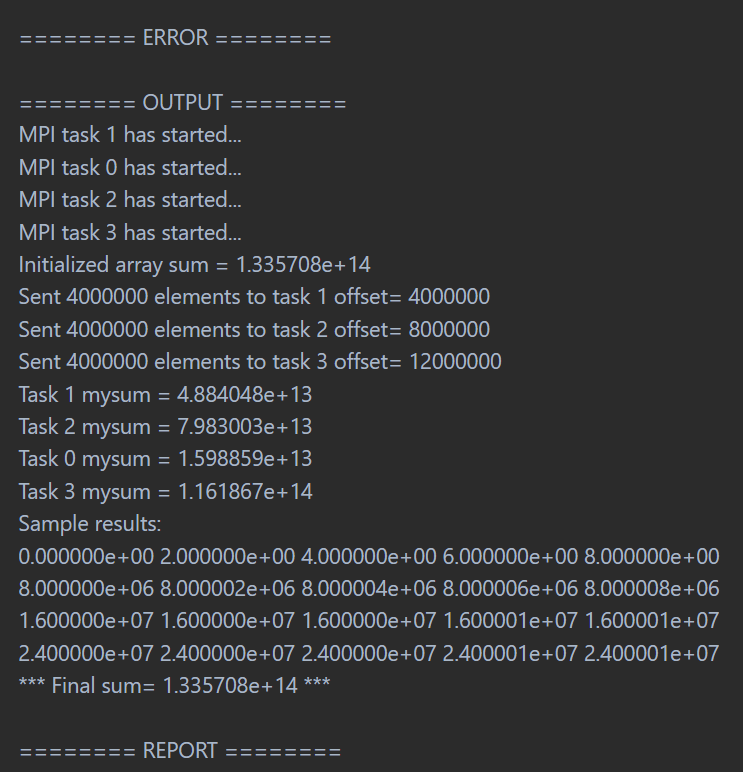
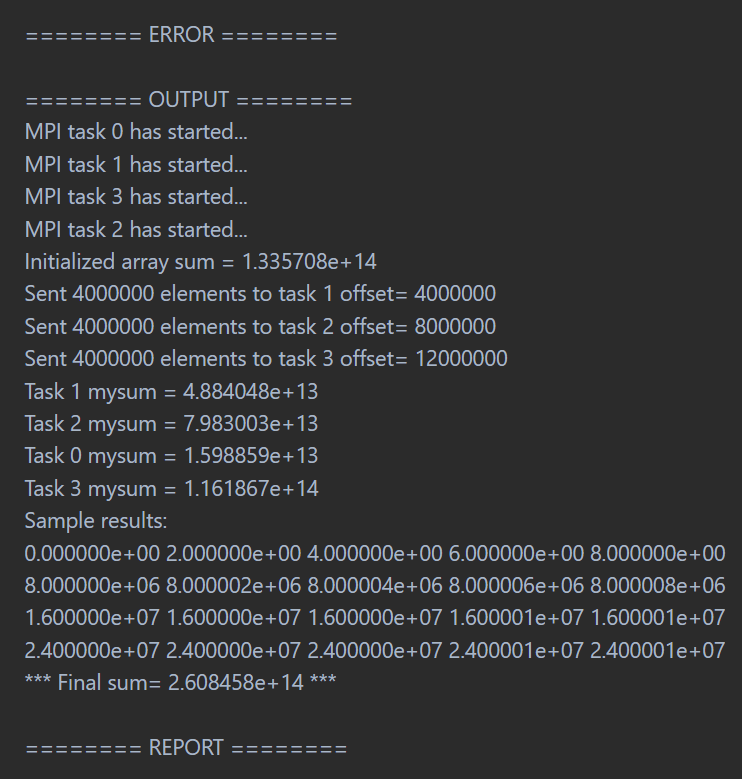
应修改为：

    }

  MPI\_Reduce(&mysum, &sum, 1, MPI\_FLOAT, MPI\_SUM, MASTER, MPI\_COMM\_WORLD);//修改处1

  /\* Get final sum and print sample results \*/

超算习堂上修改前后运行结果如下所示（运行环境：4核）：

修改前Final sum=Initialized array sum。

修改后Final sum得到了正确的数据。

（5）对于mpi\_bug5.c：

错误原因：接收方的缓冲区是有限的。所以该程序可能会因为接收方的缓冲区耗尽导致程序崩溃。所以需要添加MPI\_Barrier，设置路障以保证一个通信子中所有进程的同步。

且当进程有多个（n > 2）时，我们需要使用冗余的while循环来触发其他进程中的Barrier以使得进程同步。

该程序错误部分如下：

① 50-52行：

      }

    }

  }

应修改为：

      }

    MPI\_Barrier(MPI\_COMM\_WORLD);//修改处1

    }

  }

② 56-64行：

if (rank == 1) {

  while (1) {

    MPI\_Recv(data, MSGSIZE, MPI\_BYTE, source, tag, MPI\_COMM\_WORLD, &status);

    /\* Do some work  - at least more than the send task \*/

    result = 0.0;

    for (i=0; i < 1000000; i++)

      result = result + (double)random();

    }

  }

应修改为：

if (rank == 1) {

  while (1) {

    MPI\_Recv(data, MSGSIZE, MPI\_BYTE, source, tag, MPI\_COMM\_WORLD, &status);

    /\* Do some work  - at least more than the send task \*/

    result = 0.0;

    for (i=0; i < 1000000; i++)

      result = result + (double)random();

    MPI\_Barrier(MPI\_COMM\_WORLD);//修改处2

    }

  }

//修改处3

if(rank > 1){

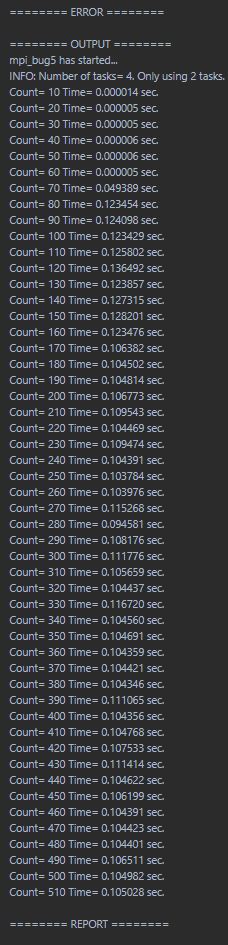
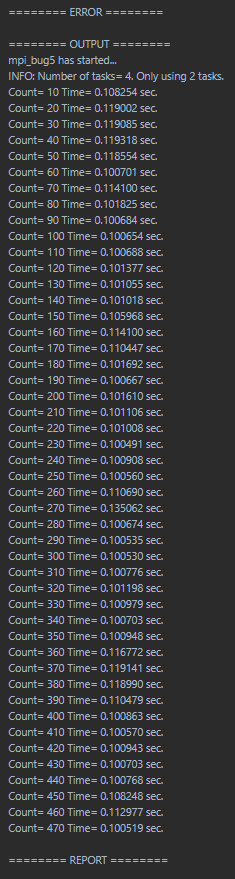
  while(1){

    MPI\_Barrier(MPI\_COMM\_WORLD);

    }

  }

超算习堂上修改前后运行结果如下所示（运行环境：4核）（见下页）：

题目中Hint提示说：

\* Hint: If possible, try to run the program on two different machines,

\* which are connected through a network. You should see uneven timings;

\* try to understand/explain them.

但我未能理解对于该程序如何实现多机之间的运行（程序主体仅仅是Send和Recv函数），所以无法通过运行结果来体现程序运行差异，但修改后的代码逻辑肯定是正确的。

（6）对于mpi\_bug6.c：

错误原因：1.进程1的偏移量应该从0开始，若它从REPS开始索引访问，它将超出reqs数组的边界。

2.进程2是非阻塞的发送接收，所以它不应该调用MPI\_Waitall。

该程序错误部分如下：

① 57行：

    offset = REPS;

应修改为：

    offset = 0;//修改处1

② 75-100行：

if (rank > 1) {

  nreqs = REPS;

/\* Task 2 does the blocking send operation \*/

  if (rank == 2) {

    dest = 3;

    for (i=0; i<REPS; i++) {

      MPI\_Send(&rank, 1, MPI\_INT, dest, tag1, COMM);

      if ((i+1)%DISP == 0)

        printf("Task %d has done %d sends\n", rank, i+1);

      }

    }

/\* Task 3 does the non-blocking receive operation \*/

  if (rank == 3) {

    src = 2;

    offset = 0;

    for (i=0; i<REPS; i++) {

      MPI\_Irecv(&buf, 1, MPI\_INT, src, tag1, COMM, &reqs[offset]);

      offset += 1;

      if ((i+1)%DISP == 0)

        printf("Task %d has done %d irecvs\n", rank, i+1);

      }

    }

  }

应修改为：

if (rank > 1) {

/\* Task 2 does the blocking send operation \*/

  if (rank == 2) {

    nreqs = 0;//修改处2

    dest = 3;

    for (i=0; i<REPS; i++) {

      MPI\_Send(&rank, 1, MPI\_INT, dest, tag1, COMM);

      if ((i+1)%DISP == 0)

        printf("Task %d has done %d sends\n", rank, i+1);

      }

    }

/\* Task 3 does the non-blocking receive operation \*/

  if (rank == 3) {

    nreqs = REPS;//修改处3

    src = 2;

    offset = 0;

    for (i=0; i<REPS; i++) {

      MPI\_Irecv(&buf, 1, MPI\_INT, src, tag1, COMM, &reqs[offset]);

      offset += 1;

      if ((i+1)%DISP == 0)

        printf("Task %d has done %d irecvs\n", rank, i+1);

      }

    }

  }

③ 102-103行：

/\* Wait for all non-blocking operations to complete and record time \*/

MPI\_Waitall(nreqs, reqs, stats);

应修改为：

/\* Wait for all non-blocking operations to complete and record time \*/

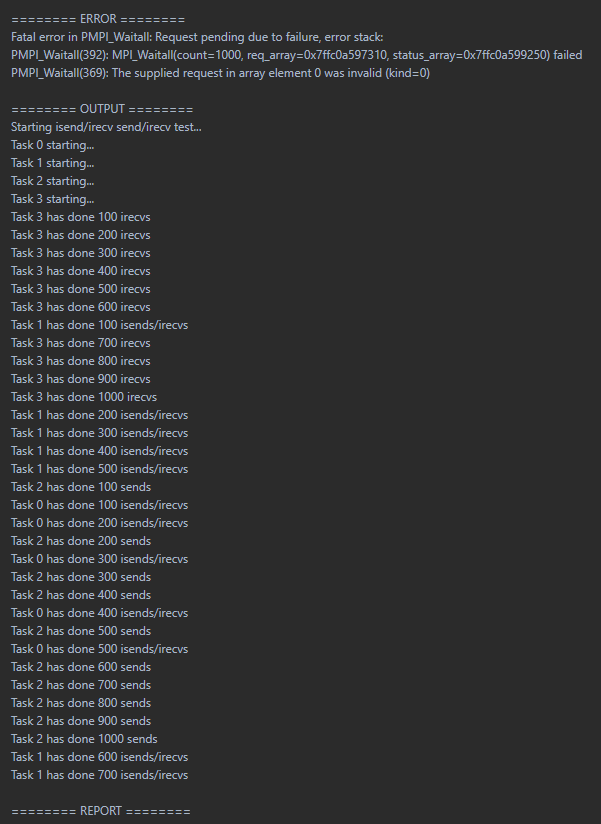
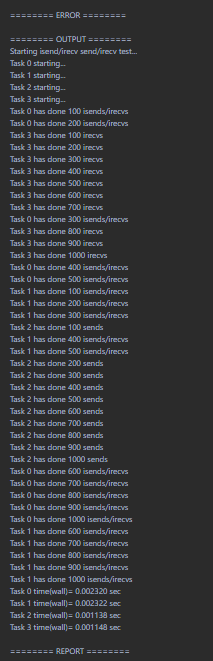
//修改处4

  if(rank != 2){

    MPI\_Waitall(nreqs, reqs, stats);

  }

超算习堂上运行结果如下所示（运行环境：4核）（见下页）：

修改前程序报错。

修改程序正常运行且得到预期结果。

（7）对于mpi\_bug7.c：

错误原因：缓冲区条目数不等于进程的rank值，应将Bcast中的第2个参数的值从count改为numtasks。

该程序错误部分如下：

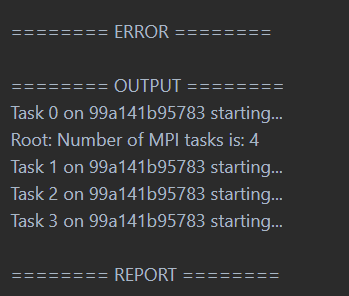
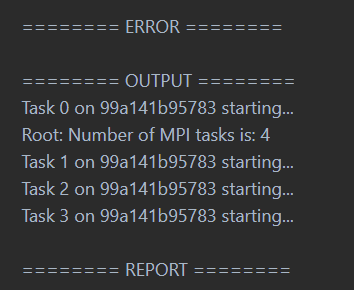
① 29行：

MPI\_Bcast(&buffer, count, MPI\_INT, root, MPI\_COMM\_WORLD);

应修改为：

MPI\_Bcast(&buffer, numtasks, MPI\_INT, root, MPI\_COMM\_WORLD);//修改处1

超算习堂上运行结果如下所示（运行环境：4核）：

运行结果无差别，但仍应该对初始代码进行修改。

四、遇到的问题及解决方法

该题未遇到难以解决的问题，实验过程较为顺利。

**对于问题2：**

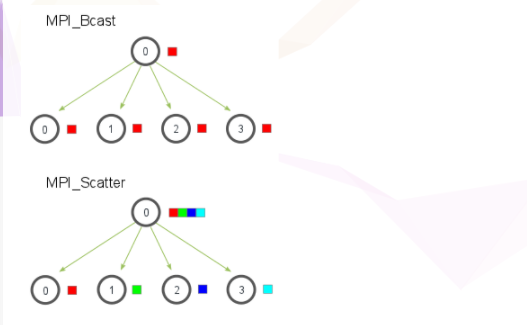
一、问题描述

利用MPI程序实现n个整形数的排序过程，排序算法不限， MPI的进程不限，可以使用单机运行多个MPI进程。提示：在排序的过程中需要MPI集合通信如MPI\_Allgather等。

二、解决方案

（1）MPI\_Scatter

MPI\_Scatter与MPI\_Bcast非常相似，都是一对多的通信方式，不同的是后者的0号进程将相同的信息发送给所有的进程，而前者则是将一段array 的不同部分发送给所有的进程，其区别可以用下图概括：



0号进程分发数据的时候是根据进程的编号进行的，array中的第一个元素发送给0号进程，第二个元素则发送给1号进程，以此类推。

MPI\_Scatter(

    void\* send\_data,//存储在0号进程的数据，array

    int send\_count,//具体需要给每个进程发送的数据的个数

    //如果send\_count为1，那么每个进程接收1个数据；如果为2，那么每个进程接收2个数据

    MPI\_Datatype send\_datatype,//发送数据的类型

    void\* recv\_data,//接收缓存，缓存 recv\_count个数据

    int recv\_count,

    MPI\_Datatype recv\_datatype,

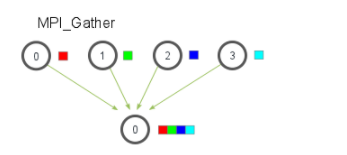
    int root,//root进程的编号

    MPI\_Comm communicator)

通常send\_count等于array的元素个数除以进程个数。

（2）MPI\_Gather

MPI\_Gather和MPI\_scatter刚好相反，他的作用是从所有的进程中将每个进程的数据集中到根进程中，同样根据进程的编号对array元素排序，如图所示：



MPI\_Gather(

    void\* send\_data,

    int send\_count,

    MPI\_Datatype send\_datatype,

    void\* recv\_data,

    int recv\_count,//注意该参数表示的是从单个进程接收的数据个数，不是总数

    MPI\_Datatype recv\_datatype,

    int root,

    MPI\_Comm communicator)

代码如下所示（具体思路和详细分析均已包含在代码注释中）：

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <mpi.h>

//Merge函数和Merge\_Sort函数用于归并排序

/\*

  函数功能：合并

  函数参数说明：

  arr: 目标数组

  start: 待合并段的开始下标

  mid: 待合并段的中间下标

  end: 待合并段的结束下标

\*/

void Merge(int\* arr, int start, int mid, int end)

{

    int len\_l, len\_r; //左右待合并区间的长度

    len\_l = mid - start + 1;

    len\_r = end - mid;

    int l[len\_l], r[len\_r]; //两个临时数组，分别保存待合并的两个区间

    memcpy(l, arr + start, sizeof(int) \* len\_l);

    memcpy(r, arr + mid + 1, sizeof(int) \* len\_r);

    int i = 0, j = 0, k = start;

    while(i < len\_l && j < len\_r)

    {

        arr[k++] = l[i] < r[j] ? l[i++] : r[j++];

    }

    while(i < len\_l)

    {

        arr[k++] = l[i++];

    }

}

/\*

  函数功能：归并排序

  函数参数说明：

  arr: 待排序的数组

  start: 待排序数组的开始下标

  end: 待排序数组的结束下标

\*/

void Merge\_Sort(int\* arr, int start, int end)

{

    if(start < end)

    {

        int mid = (start + end) / 2;

        //归

        Merge\_Sort(arr, start, mid);

        Merge\_Sort(arr, mid + 1, end);

        //并

        Merge(arr, start, mid, end);

    }

}

int main(int argc,char \*argv[])

{

    int comm\_sz;

    int my\_rank;

    int length = 20; //数组大小，可修改

    int size; //每个进程所需数组的大小

    int\* sub; //子数组，用于各进程内部的归并排序

    int\* result = NULL; //结果数组，存储最终结果

    int i;

    int arr[length]; //待排序的数组

    //MPI初始化

    MPI\_Init(NULL, NULL);

    MPI\_Comm\_size(MPI\_COMM\_WORLD, &comm\_sz);

    MPI\_Comm\_rank(MPI\_COMM\_WORLD, &my\_rank);

    //在分发(Scatter)数据前，若进程号为0(根进程)，则对数组进行随机数初始化并打印初始数组信息

    if(my\_rank == 0){

        printf("初始数组为:\n");

        for(i = 0;i <= length-1; i++){

            arr[i] = rand()%50; //每个元素的值控制在0-49

            printf("%d ",arr[i]);

        }

        printf("\n");

    }

    size = length / comm\_sz; //计算每个进程所需数组的大小

    sub = (int\*)malloc(size \* sizeof(int)); //为进程所用数组开辟空间

    MPI\_Scatter(arr, size, MPI\_INT, sub, size, MPI\_INT, 0, MPI\_COMM\_WORLD); //将数组元素分发给每个进程

    Merge\_Sort(sub, 0, size-1); //将每个进程分到的子数组进行归并排序

    //在分发(Scatter)数据后，若进程号为0(根进程)，则开辟结果所需的数组空间（与输入数组一样大）

    if(my\_rank == 0) {

        result = (int \*)malloc(length \* sizeof(int));

    }

    MPI\_Gather(sub, size, MPI\_INT, result, size, MPI\_INT, 0, MPI\_COMM\_WORLD); //将各进程排序的数组收集到0号进程(根进程)

    //在收集(Gather)各进程数据到根进程后，若进程号为0(根进程)，将收集的结果进行最后一次归并排序

    if (my\_rank == 0) {

        Merge\_Sort(result, 0, length-1);

        //打印排序后的结果

        printf("\n排序后的结果为:\n");

        for (i = 0;i <= length-1; i++) {

            printf("%d ", result[i]);

        }

        printf("\n");

        free(result);//释放内存

    }

    free(sub);//释放内存

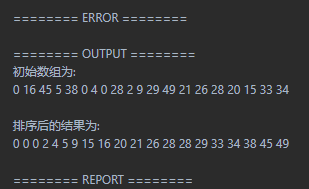
    MPI\_Finalize();//终止MPI模拟环境

    return 0;

}

三、实验结果

超算习堂上运行结果如下所示（运行环境：4核）：



四、遇到的问题及解决方法

本次实验主要困难是对MPI\_Scatter和MPI\_Gather的应用不够熟练（之前没有专门练习过）。在查阅网上资料和相关例程后完成了对n个整数的MPI并行归并排序。