

并行与分布式计算 第6次作业

19335015 陈恩婷

第一题

mpi_bug1.c

这个程序主要是发送/接收信息的tag参数有问题，第一条信息统一tag为0，第二条统一tag为1即可。

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
    int numtasks, rank, dest, tag, source, rc, count;
    char inmsg, outmsg='x';
    MPI_Status Stat;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    printf("Task %d starting...\n",rank);

    if (rank == 0) {
        if (numtasks > 2)
            printf("Numtasks=%d. Only 2 needed. Ignoring extra...\n",numtasks);
        dest = rank + 1;
        source = dest;
        tag = rank;
        rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, 0, MPI_COMM_WORLD);
        printf("Sent to task %d...\n",dest);
        rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, 1, MPI_COMM_WORLD, &Stat);
        printf("Received from task %d...\n",source);
    }

    else if (rank == 1) {
        dest = rank - 1;
        source = dest;
        tag = rank;
        rc = MPI_Recv(&inmsg, 1, MPI_CHAR, source, 0, MPI_COMM_WORLD, &Stat);
        printf("Received from task %d...\n",source);
        rc = MPI_Send(&outmsg, 1, MPI_CHAR, dest, 1, MPI_COMM_WORLD);
        printf("Sent to task %d...\n",dest);
    }

    if (rank < 2) {
        rc = MPI_Get_count(&Stat, MPI_CHAR, &count);
        printf("Task %d: Received %d char(s) from task %d with tag %d \n",
            rank, count, Stat.MPI_SOURCE, Stat.MPI_TAG);
    }

    MPI_Finalize();
}
```

```
}
```

修改前的运行结果:

```
ocelot@ubuntu: ~/homework6/homework6_1
mpi_bug1.c mpi_bug2.c mpi_bug3.c mpi_bug4.c mpi_bug5.c mpi_bug6.c mpi_bug7.c
ocelot@ubuntu:~/homework6/homework6_1$ mpicc -g -Wall -o mpi_bug1 mpi_bug1.c
mpi_bug1.c: In function 'main':
mpi_bug1.c:14:40: warning: variable 'rc' set but not used [-Wunused-but-set-variable]
   14 | int numtasks, rank, dest, tag, source, rc, count;
      |                                         ^~
ocelot@ubuntu:~/homework6/homework6_1$ mpicc -g -Wall -o mpi_bug2 mpi_bug2.c
ocelot@ubuntu:~/homework6/homework6_1$ mpicc -g -Wall -o mpi_bug3 mpi_bug3.c
mpi_bug3.c: In function 'main':
mpi_bug3.c:28:4: warning: 'rc' may be used uninitialized in this function [-Wmaybe-uninitialized]
   28 |     MPI_Abort(MPI_COMM_WORLD, rc);
      |     ^
ocelot@ubuntu:~/homework6/homework6_1$ mpicc -g -Wall -o mpi_bug4 mpi_bug4.c
mpi_bug4.c: In function 'main':
mpi_bug4.c:30:4: warning: 'rc' may be used uninitialized in this function [-Wmaybe-uninitialized]
   30 |     MPI_Abort(MPI_COMM_WORLD, rc);
      |     ^
ocelot@ubuntu:~/homework6/homework6_1$ mpicc -g -Wall -o mpi_bug5 mpi_bug5.c
ocelot@ubuntu:~/homework6/homework6_1$ mpicc -g -Wall -o mpi_bug6 mpi_bug6.c
mpi_bug6.c: In function 'main':
mpi_bug6.c:33:5: warning: 'rc' may be used uninitialized in this function [-Wmaybe-uninitialized]
   33 |     MPI_Abort(COMM, rc);
      |     ^
ocelot@ubuntu:~/homework6/homework6_1$ mpicc -g -Wall -o mpi_bug7 mpi_bug7.c
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug1
Task 0 starting...
Task 1 starting...
Numtasks=4. Only 2 needed. Ignoring extra...
Sent to task 1...
Task 2 starting...
Task 3 starting...
```

修改后的运行结果:

```
ocelot@ubuntu: ~/homework6/homework6
Task 3 on ubuntu starting...
Task 1 on ubuntu starting...
Task 2 on ubuntu starting...
Task 0 on ubuntu starting...
Root: Number of MPI tasks is: 4
ocelot@ubuntu:~/homework6/homework6$ mpicc -g -Wall -o mpi_bug7 mpi_bug7.c
mpi_bug7.c: In function 'main':
mpi_bug7.c:14:44: warning: variable 'count' set but not used [-Wunused-but-set-variable]
   14 | int numtasks, taskid, len, buffer, root, count;
      |                                           ^~~~~
ocelot@ubuntu:~/homework6/homework6$ mpiexec -n 4 ./mpi_bug1
Task 1 starting...
Task 3 starting...
Task 0 starting...
Numtasks=4. Only 2 needed. Ignoring extra...
Sent to task 1...
Received from task 0...
Sent to task 0...
Task 1: Received 1 char(s) from task 0 with tag 0
Received from task 1...
Task 0: Received 1 char(s) from task 1 with tag 1
Task 2 starting...
ocelot@ubuntu:~/homework6/homework6$
```

mpi_bug2.c

这个程序主要是发送和接收信息的数据类型不匹配, 发送时使用MPI_INT, 接收却使用了MPI_FLOAT, 统一为MPI_INT即可。

```
#include "mpi.h"
#include <stdio.h>
```

```

#include <stdlib.h>

int main (int argc, char *argv[])
{
    int numtasks, rank, tag=1, alpha, i;
    int beta;
    MPI_Request reqs[10];
    MPI_Status stats[10];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0) {
        if (numtasks > 2)
            printf("Numtasks=%d. Only 2 needed. Ignoring extra...\n",numtasks);
        for (i=0; i<10; i++) {
            alpha = i*10;
            MPI_Isend(&alpha, 1, MPI_INT, 1, tag, MPI_COMM_WORLD, &reqs[i]);
            MPI_Wait(&reqs[i], &stats[i]);
            printf("Task %d sent = %d\n",rank,alpha);
        }
    }

    if (rank == 1) {
        for (i=0; i<10; i++) {
            MPI_Irecv(&beta, 1, MPI_INT, 0, tag, MPI_COMM_WORLD, &reqs[i]);
            MPI_Wait(&reqs[i], &stats[i]);
            printf("Task %d received = %d\n",rank,beta);
        }
    }

    MPI_Finalize();
}

```

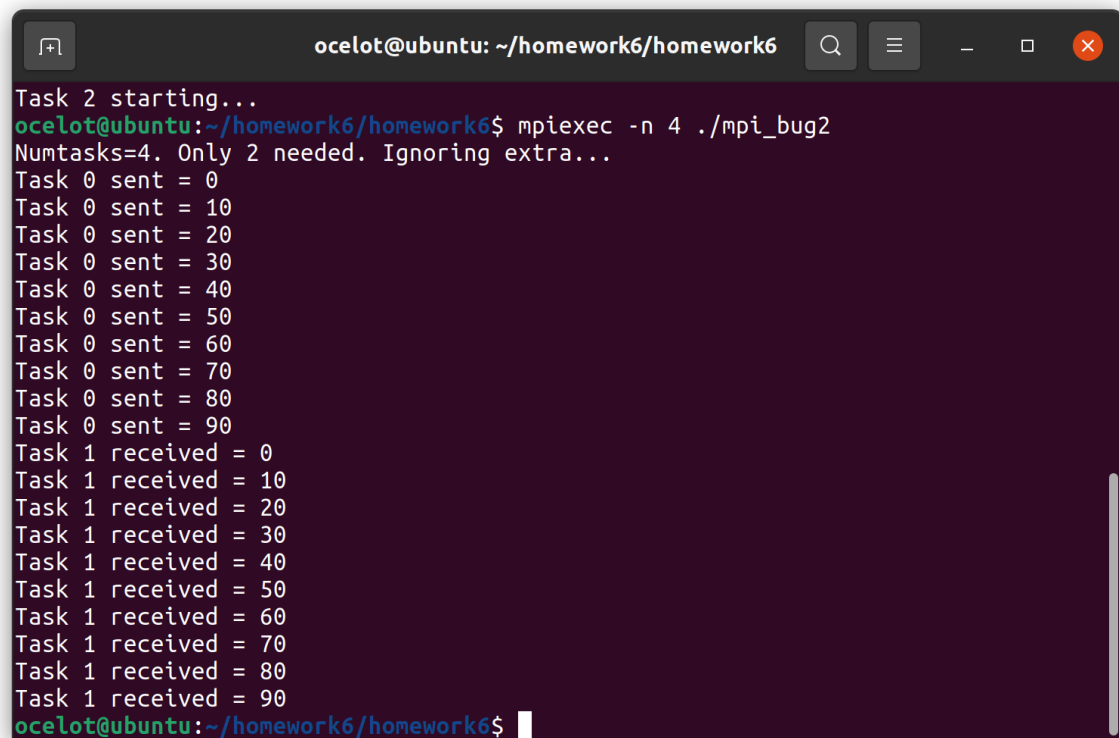
修改前的运行结果:

```

ocelot@ubuntu: ~/homework6/homework6_1
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug1
Task 0 starting...
Task 1 starting...
Numtasks=4. Only 2 needed. Ignoring extra...
Sent to task 1...
Task 2 starting...
Task 3 starting...
^C[mpiexec@ubuntu] Sending Ctrl-C to processes as requested
[mpiexec@ubuntu] Press Ctrl-C again to force abort
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug2
Numtasks=4. Only 2 needed. Ignoring extra...
Task 0 sent = 0
Task 0 sent = 10
Task 0 sent = 20
Task 0 sent = 30
Task 0 sent = 40
Task 0 sent = 50
Task 0 sent = 60
Task 0 sent = 70
Task 0 sent = 80
Task 0 sent = 90
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
ocelot@ubuntu:~/homework6/homework6_1$

```

修改后的运行结果：



```
Task 2 starting...
ocelot@ubuntu:~/homework6/homework6$ mpiexec -n 4 ./mpi_bug2
Numtasks=4. Only 2 needed. Ignoring extra...
Task 0 sent = 0
Task 0 sent = 10
Task 0 sent = 20
Task 0 sent = 30
Task 0 sent = 40
Task 0 sent = 50
Task 0 sent = 60
Task 0 sent = 70
Task 0 sent = 80
Task 0 sent = 90
Task 1 received = 0
Task 1 received = 10
Task 1 received = 20
Task 1 received = 30
Task 1 received = 40
Task 1 received = 50
Task 1 received = 60
Task 1 received = 70
Task 1 received = 80
Task 1 received = 90
ocelot@ubuntu:~/homework6/homework6$
```

mpi_bug3.c

这个程序主要是缺了MPI_INIT和MPI_FINALIZE，补上即可。

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define ARRAYSIZE 16000000
#define MASTER 0

float data[ARRAYSIZE];

int main(int argc, char *argv[])
{
    int numtasks, taskid, rc, dest, offset, i, j, tag1,
        tag2, source, chunksize;
    float mysum, sum;
    float update(int myoffset, int chunk, int myid);
    MPI_Status status;

    /***** Initializations *****/
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    if (numtasks % 4 != 0)
    {
        printf("Quitting. Number of MPI tasks must be divisible by 4.\n");
        MPI_Abort(MPI_COMM_WORLD, rc);
        exit(0);
    }
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    printf("MPI task %d has started...\n", taskid);
    chunksize = (ARRAYSIZE / numtasks);
```

```

tag2 = 1;
tag1 = 2;

/***** Master task only *****/
if (taskid == MASTER)
{

    /* Initialize the array */
    sum = 0;
    for (i = 0; i < ARRAYSIZE; i++)
    {
        data[i] = i * 1.0;
        sum = sum + data[i];
    }
    printf("Initialized array sum = %e\n", sum);

    /* Send each task its portion of the array - master keeps 1st part */
    offset = chunksize;
    for (dest = 1; dest < numtasks; dest++)
    {
        MPI_Send(&offset, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD);
        MPI_Send(&data[offset], chunksize, MPI_FLOAT, dest, tag2, MPI_COMM_WORLD);
        printf("Sent %d elements to task %d offset= %d\n", chunksize, dest,
offset);
        offset = offset + chunksize;
    }

    /* Master does its part of the work */
    offset = 0;
    mysum = update(offset, chunksize, taskid);

    /* Wait to receive results from each task */
    for (i = 1; i < numtasks; i++)
    {
        source = i;
        MPI_Recv(&offset, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status);
        MPI_Recv(&data[offset], chunksize, MPI_FLOAT, source, tag2,
MPI_COMM_WORLD, &status);
    }

    /* Get final sum and print sample results */
    MPI_Reduce(&mysum, &sum, 1, MPI_FLOAT, MPI_SUM, MASTER, MPI_COMM_WORLD);
    printf("Sample results: \n");
    offset = 0;
    for (i = 0; i < numtasks; i++)
    {
        for (j = 0; j < 5; j++)
            printf(" %e", data[offset + j]);
        printf("\n");
        offset = offset + chunksize;
    }
    printf("*** Final sum= %e ***\n", sum);

} /* end of master section */

/***** Non-master tasks only *****/

if (taskid > MASTER)

```

```

{

    /* Receive my portion of array from the master task */
    source = MASTER;
    MPI_Recv(&offset, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status);
    MPI_Recv(&data[offset], chunksize, MPI_FLOAT, source, tag2,
            MPI_COMM_WORLD, &status);

    mysum = update(offset, chunksize, taskid);

    /* Send my results back to the master task */
    dest = MASTER;
    MPI_Send(&offset, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD);
    MPI_Send(&data[offset], chunksize, MPI_FLOAT, MASTER, tag2, MPI_COMM_WORLD);

    MPI_Reduce(&mysum, &sum, 1, MPI_FLOAT, MPI_SUM, MASTER, MPI_COMM_WORLD);

} /* end of non-master */

MPI_Finalize();

} /* end of main */

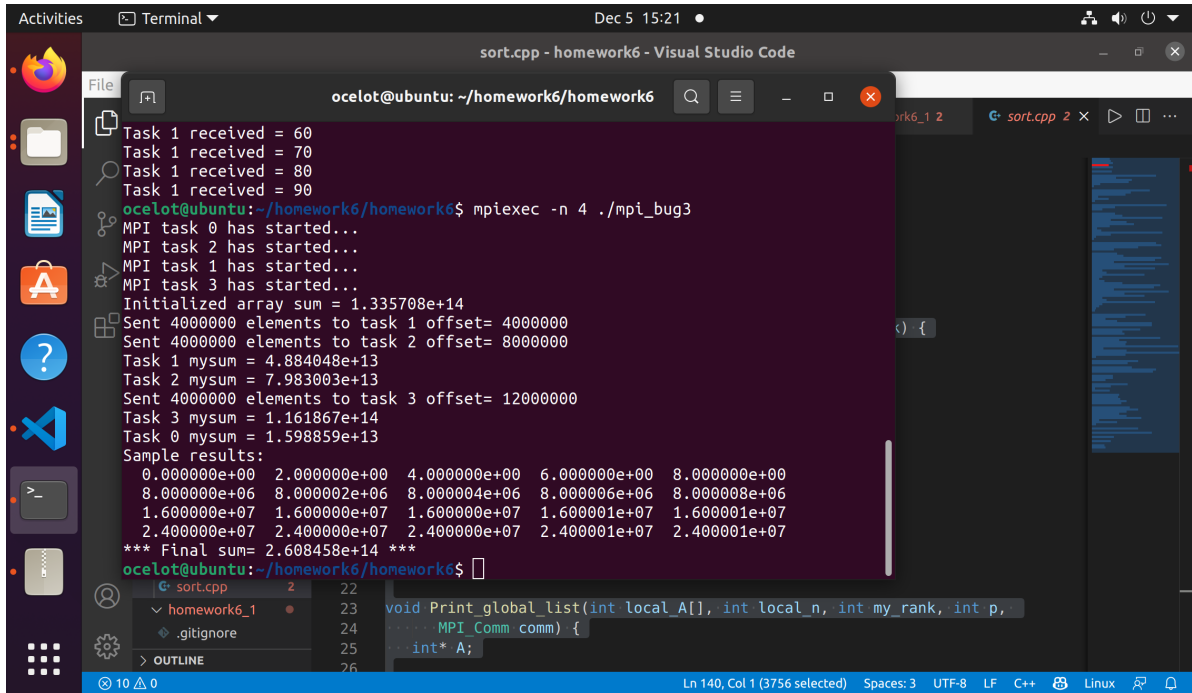
float update(int myoffset, int chunk, int myid)
{
    int i;
    float mysum;
    /* Perform addition to each of my array elements and keep my sum */
    mysum = 0;
    for (i = myoffset; i < myoffset + chunk; i++)
    {
        data[i] = data[i] + i * 1.0;
        mysum = mysum + data[i];
    }
    printf("Task %d mysum = %e\n", myid, mysum);
    return (mysum);
}

```

修改前的运行结果：

```
ocelot@ubuntu: ~/homework6/homework6_1
Task 2 starting...
Task 3 starting...
^C[mpiexec@ubuntu] Sending Ctrl-C to processes as requested
[mpiexec@ubuntu] Press Ctrl-C again to force abort
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug2
Numtasks=4. Only 2 needed. Ignoring extra...
Task 0 sent = 0
Task 0 sent = 10
Task 0 sent = 20
Task 0 sent = 30
Task 0 sent = 40
Task 0 sent = 50
Task 0 sent = 60
Task 0 sent = 70
Task 0 sent = 80
Task 0 sent = 90
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug3
Attempting to use an MPI routine before initializing MPICH
Attempting to use an MPI routine before initializing MPICH
Attempting to use an MPI routine before initializing MPICH
Attempting to use an MPI routine before initializing MPICH
ocelot@ubuntu:~/homework6/homework6_1$
```

修改后的运行结果：



```
Activities Terminal Dec 5 15:21
sort.cpp - homework6 - Visual Studio Code
ocelot@ubuntu: ~/homework6/homework6
Task 1 received = 60
Task 1 received = 70
Task 1 received = 80
Task 1 received = 90
ocelot@ubuntu:~/homework6/homework6$ mpiexec -n 4 ./mpi_bug3
MPI task 0 has started...
MPI task 2 has started...
MPI task 1 has started...
MPI task 3 has started...
Initialized array sum = 1.335708e+14
Sent 4000000 elements to task 1 offset= 4000000
Sent 4000000 elements to task 2 offset= 8000000
Task 1 mysum = 4.884048e+13
Task 2 mysum = 7.983003e+13
Sent 4000000 elements to task 3 offset= 12000000
Task 3 mysum = 1.161867e+14
Task 0 mysum = 1.598859e+13
Sample results:
0.000000e+00 2.000000e+00 4.000000e+00 6.000000e+00 8.000000e+00
8.000000e+06 8.000002e+06 8.000004e+06 8.000006e+06 8.000008e+06
1.600000e+07 1.600000e+07 1.600000e+07 1.600001e+07 1.600001e+07
2.400000e+07 2.400000e+07 2.400000e+07 2.400001e+07 2.400001e+07
*** Final sum= 2.608458e+14 ***
ocelot@ubuntu:~/homework6/homework6$
```

mpi_bug4.c

这个程序主要是缺了MPI_REDUCE，补上即可。

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#define ARRAYSIZE 16000000
#define MASTER 0

float data[ARRAYSIZE];

int main (int argc, char *argv[])
{
    int numtasks, taskid, rc, dest, offset, i, j, tag1,
```

```

    tag2, source, chunksize;
float mysum, sum;
float update(int myoffset, int chunk, int myid);
MPI_Status status;

/***** Initializations *****/
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
if (numtasks % 4 != 0) {
    printf("Quitting. Number of MPI tasks must be divisible by 4.\n");
    MPI_Abort(MPI_COMM_WORLD, rc);
    exit(0);
}
MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
printf("MPI task %d has started...\n", taskid);
chunksize = (ARRAYSIZE / numtasks);
tag2 = 1;
tag1 = 2;

/***** Master task only *****/
if (taskid == MASTER){

    /* Initialize the array */
    sum = 0;
    for(i=0; i<ARRAYSIZE; i++) {
        data[i] = i * 1.0;
        sum = sum + data[i];
    }
    printf("Initialized array sum = %e\n", sum);

    /* Send each task its portion of the array - master keeps 1st part */
    offset = chunksize;
    for (dest=1; dest<numtasks; dest++) {
        MPI_Send(&offset, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD);
        MPI_Send(&data[offset], chunksize, MPI_FLOAT, dest, tag2, MPI_COMM_WORLD);
        printf("Sent %d elements to task %d offset= %d\n", chunksize, dest, offset);
        offset = offset + chunksize;
    }

    /* Master does its part of the work */
    offset = 0;
    mysum = update(offset, chunksize, taskid);

    /* Wait to receive results from each task */
    for (i=1; i<numtasks; i++) {
        source = i;
        MPI_Recv(&offset, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status);
        MPI_Recv(&data[offset], chunksize, MPI_FLOAT, source, tag2,
            MPI_COMM_WORLD, &status);
    }

    /* Get final sum and print sample results */
    MPI_Reduce(&mysum, &sum, 1, MPI_FLOAT, MPI_SUM, MASTER, MPI_COMM_WORLD);
    printf("Sample results: \n");
    offset = 0;
    for (i=0; i<numtasks; i++) {
        for (j=0; j<5; j++)
            printf(" %e", data[offset+j]);

```



```

    printf("\n");
    offset = offset + chunksize;
}
printf("*** Final sum= %e ***\n",sum);

} /* end of master section */

/***** Non-master tasks only *****/

if (taskid > MASTER) {

    /* Receive my portion of array from the master task */
    source = MASTER;
    MPI_Recv(&offset, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status);
    MPI_Recv(&data[offset], chunksize, MPI_FLOAT, source, tag2,
        MPI_COMM_WORLD, &status);

    mysum = update(offset, chunksize, taskid);

    /* Send my results back to the master task */
    dest = MASTER;
    MPI_Send(&offset, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD);
    MPI_Send(&data[offset], chunksize, MPI_FLOAT, MASTER, tag2, MPI_COMM_WORLD);

    MPI_Reduce(&mysum, &sum, 1, MPI_FLOAT, MPI_SUM, MASTER, MPI_COMM_WORLD);

} /* end of non-master */

MPI_Finalize();

} /* end of main */

float update(int myoffset, int chunk, int myid) {
    int i;
    float mysum;
    /* Perform addition to each of my array elements and keep my sum */
    mysum = 0;
    for(i=myoffset; i < myoffset + chunk; i++) {
        data[i] = data[i] + i * 1.0;
        mysum = mysum + data[i];
    }
    printf("Task %d mysum = %e\n",myid,mysum);
    return(mysum);
}

```

修改前的运行结果：

```
oceleot@ubuntu: ~/homework6/homework6_1
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
Task 1 received = 0.000000
oceleot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug3
Attempting to use an MPI routine before initializing MPICH
Attempting to use an MPI routine before initializing MPICH
Attempting to use an MPI routine before initializing MPICH
Attempting to use an MPI routine before initializing MPICH
oceleot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug4
MPI task 2 has started...
MPI task 3 has started...
MPI task 0 has started...
MPI task 1 has started...
Initialized array sum = 1.335708e+14
Sent 4000000 elements to task 1 offset= 4000000
Task 1 mysum = 4.884048e+13
Sent 4000000 elements to task 2 offset= 8000000
Task 2 mysum = 7.983003e+13
Sent 4000000 elements to task 3 offset= 12000000
Task 0 mysum = 1.598859e+13
Task 3 mysum = 1.161867e+14
Sample results:
0.000000e+00 2.000000e+00 4.000000e+00 6.000000e+00 8.000000e+00
8.000000e+06 8.000002e+06 8.000004e+06 8.000006e+06 8.000008e+06
1.600000e+07 1.600000e+07 1.600000e+07 1.600001e+07 1.600001e+07
2.400000e+07 2.400000e+07 2.400000e+07 2.400001e+07 2.400001e+07
*** Final sum= 1.335708e+14 ***
oceleot@ubuntu:~/homework6/homework6_1$
```

修改后的运行结果：

```
oceleot@ubuntu: ~/homework6/homework6
8.000000e+06 8.000002e+06 8.000004e+06 8.000006e+06 8.000008e+06
1.600000e+07 1.600000e+07 1.600000e+07 1.600001e+07 1.600001e+07
2.400000e+07 2.400000e+07 2.400000e+07 2.400001e+07 2.400001e+07
*** Final sum= 2.608458e+14 ***
oceleot@ubuntu:~/homework6/homework6$ mpiexec -n 4 ./mpi_bug4
MPI task 0 has started...
MPI task 1 has started...
MPI task 3 has started...
MPI task 2 has started...
Initialized array sum = 1.335708e+14
Sent 4000000 elements to task 1 offset= 4000000
Sent 4000000 elements to task 2 offset= 8000000
Task 1 mysum = 4.884048e+13
Task 2 mysum = 7.983003e+13
Sent 4000000 elements to task 3 offset= 12000000
Task 0 mysum = 1.598859e+13
Task 3 mysum = 1.161867e+14
Sample results:
0.000000e+00 2.000000e+00 4.000000e+00 6.000000e+00 8.000000e+00
8.000000e+06 8.000002e+06 8.000004e+06 8.000006e+06 8.000008e+06
1.600000e+07 1.600000e+07 1.600000e+07 1.600001e+07 1.600001e+07
2.400000e+07 2.400000e+07 2.400000e+07 2.400001e+07 2.400001e+07
*** Final sum= 2.608458e+14 ***
oceleot@ubuntu:~/homework6/homework6$
```

mpi_bug5.c

这个程序主要是发送没有阻塞直到确认接收完毕，导致最开始几次发送间隔非常短。我查了一下网上的解决方案，有[这样一个回答](#)：

Ways to improve this code so that the receiver doesn't end up with an unlimited number of [unexpected messages](#) include:

- Synchronization - you mentioned MPI_Barrier, but even using MPI_Ssend instead of MPI_Send would work.
- Explicit buffering - the use of MPI_Bsend or Brecv to ensure adequate buffering exists.

- Posted receives - the receiving process posts IRecv before starting work to ensure that the messages are received into the buffers meant to hold the data, rather than system buffers.

In this pedagogical case, since the number of messages is unlimited, only the first (synchronization) would reliably work.

这里提到了三种方法，但是作者说后面两种不适合这种无限多信息的场景，因为bsend/brecv和isend/irecv都是需要调用 [MPI_Buffer_attach](#) 提前分配buffer的空间，这种无限多信息的情况下难以保证分配的buffer空间够用。所以选用第一种方法，将send修改为ssend：

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

#define MSGSIZE 2000

int main (int argc, char *argv[])
{
    int      numtasks, rank, i, tag=111, dest=1, source=0, count=0;
    char     data[MSGSIZE];
    double   start, end, result;
    MPI_Status status;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    if (rank == 0) {
        printf ("mpi_bug5 has started...\n");
        if (numtasks > 2)
            printf("INFO: Number of tasks= %d. Only using 2 tasks.\n", numtasks);
    }

    /***** Send task *****/
    if (rank == 0) {

        /* Initialize send data */
        for(i=0; i<MSGSIZE; i++)
            data[i] = 'x';

        start = MPI_Wtime();
        while (1) {
            MPI_Ssend(data, MSGSIZE, MPI_BYTE, dest, tag, MPI_COMM_WORLD);
            count++;
            if (count % 10 == 0) {
                end = MPI_Wtime();
                printf("Count= %d Time= %f sec.\n", count, end-start);
                start = MPI_Wtime();
            }
        }
    }

    /***** Receive task *****/

    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_Finalize();
}

```

修改前的运行结果:

```

ocelot@ubuntu: ~/homework6/homework6_1
Sent 4000000 elements to task 2 offset= 8000000
Task 2 mysum = 7.983003e+13
Sent 4000000 elements to task 3 offset= 12000000
Task 0 mysum = 1.598859e+13
Task 3 mysum = 1.161867e+14
Sample results:
0.000000e+00 2.000000e+00 4.000000e+00 6.000000e+00 8.000000e+00
8.000000e+06 8.000002e+06 8.000004e+06 8.000006e+06 8.000008e+06
1.600000e+07 1.600000e+07 1.600000e+07 1.600001e+07 1.600001e+07
2.400000e+07 2.400000e+07 2.400000e+07 2.400001e+07 2.400001e+07
*** Final sum= 1.335708e+14 ***
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug5
mpi_bug5 has started...
INFO: Number of tasks= 4. Only using 2 tasks.
Count= 10 Time= 0.000011 sec.
Count= 20 Time= 0.000003 sec.
Count= 30 Time= 0.000003 sec.
Count= 40 Time= 0.000004 sec.
Count= 50 Time= 0.000004 sec.
Count= 60 Time= 0.000004 sec.
Count= 70 Time= 0.110662 sec.
Count= 80 Time= 0.240462 sec.
Count= 90 Time= 0.268707 sec.
Count= 100 Time= 0.241388 sec.
Count= 110 Time= 0.313324 sec.
Count= 120 Time= 0.219512 sec.
Count= 130 Time= 0.214861 sec.
Count= 140 Time= 0.218968 sec.
Count= 150 Time= 0.220095 sec.
^C[mpiexec@ubuntu] Sending Ctrl-C to processes as requested
[mpiexec@ubuntu] Press Ctrl-C again to force abort
ocelot@ubuntu:~/homework6/homework6_1$

```

修改后的运行结果:

```

ocelot@ubuntu: ~/homework6/homework6
1.600000e+07 1.600000e+07 1.600000e+07 1.600001e+07 1.600001e+07
2.400000e+07 2.400000e+07 2.400000e+07 2.400001e+07 2.400001e+07
*** Final sum= 2.608458e+14 ***
ocelot@ubuntu:~/homework6/homework6$ mpiexec -n 4 ./mpi_bug5
mpi_bug5 has started...
INFO: Number of tasks= 4. Only using 2 tasks.
Count= 10 Time= 0.222952 sec.
Count= 20 Time= 0.243782 sec.
Count= 30 Time= 0.230220 sec.
Count= 40 Time= 0.236475 sec.
Count= 50 Time= 0.255943 sec.
Count= 60 Time= 0.231773 sec.
Count= 70 Time= 0.239645 sec.
Count= 80 Time= 0.235680 sec.
Count= 90 Time= 0.239881 sec.
Count= 100 Time= 0.236299 sec.
Count= 110 Time= 0.242690 sec.
Count= 120 Time= 0.244879 sec.
Count= 130 Time= 0.232409 sec.
Count= 140 Time= 0.243081 sec.
Count= 150 Time= 0.231713 sec.
^C[mpiexec@ubuntu] Sending Ctrl-C to processes as requested
[mpiexec@ubuntu] Press Ctrl-C again to force abort
ocelot@ubuntu:~/homework6/homework6$

```

mpi_bug6.c

这个程序主要是有两个问题，一个是编号为1的进程offset设置得不对，导致最后调用waitall时因为request不是从reqs处开始存放而出错，另外一个为rank为2的进程其实没有非阻塞式的请求，所以waitall时nreqs应该是零，修改好即可。

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

#define COMM MPI_COMM_WORLD
#define REPS 1000
#define DISP 100

int main (int argc, char *argv[])
{
    int numtasks, rank, buf, tag1=1, i, rc, dest, src, offset, nreqs;
    double T1, T2;
    MPI_Request reqs[REPS*2];
    MPI_Status stats[REPS*2];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(COMM, &numtasks);
    MPI_Comm_rank(COMM, &rank);

    /* Require 4 tasks */
    if (rank == 0 ) {
        if (numtasks != 4) {
            printf("ERROR: Number of tasks must be 4. Quitting.\n");
            MPI_Abort(COMM, rc);
        }
        printf("Starting isend/irecv send/irecv test...\n");
    }

    /* Use barriers for clean output */
    MPI_Barrier(COMM);
    printf("Task %d starting...\n", rank);
    MPI_Barrier(COMM);

    T1 = MPI_Wtime();      /* start the clock */

    /* Tasks 0 and 1 do the isend/irecv test.
     * Determine who to send/receive with. nreqs specifies how many non-blocking
     * operation request handles to capture. offset is where the task should
     * store each request as it is captured in the reqs() array.          */
    if (rank < 2) {
        nreqs = REPS*2;
        if (rank == 0) {
            src = 1;
            offset = 0;
        }
        if (rank == 1) {
            src = 0;
            offset = 0;
        }
        dest = src;
    }
```

```

/* Do the non-blocking send and receive operations */
for (i=0; i<REPS; i++) {
    MPI_Isend(&rank, 1, MPI_INT, dest, tag1, COMM, &reqs[offset]);
    MPI_Irecv(&buf, 1, MPI_INT, src, tag1, COMM, &reqs[offset+1]);
    offset += 2;
    if ((i+1)%DISP == 0)
        printf("Task %d has done %d isends/irecvs\n", rank, i+1);
}

/* Tasks 2 and 3 do the send/irecv test.
Determine who to send/receive with. nreqs specifies how many non-blocking
operation request handles to capture. offset is where the task should
store each request as it is captured in the reqs() array. */
if (rank > 1) {
    nreqs = REPS;

/* Task 2 does the blocking send operation */
    if (rank == 2) {
        dest = 3;
        nreqs=0;
        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);
        }
    }

}

/* wait for all non-blocking operations to complete and record time */
MPI_Waitall(nreqs, reqs, stats);
T2 = MPI_Wtime();      /* end time */
MPI_Barrier(COMM);

printf("Task %d time(wall)= %lf sec\n", rank, T2-T1);

MPI_Finalize();
}

```

修改前的运行结果：

```
Count= 110 Time= 0.313324 sec.
Count= 120 Time= 0.219512 sec.
Count= 130 Time= 0.214861 sec.
Count= 140 Time= 0.218968 sec.
Count= 150 Time= 0.220095 sec.
^C[mpiexec@ubuntu] Sending Ctrl-C to processes as requested
[mpiexec@ubuntu] Press Ctrl-C again to force abort
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug6
Starting isend/irecv send/irecv test...
Task 0 starting...
Task 1 starting...
Task 2 starting...
Task 3 starting...
Task 0 has done 100 isends/irecvs
Task 0 has done 200 isends/irecvs
Task 0 has done 300 isends/irecvs
Task 0 has done 400 isends/irecvs
Task 0 has done 500 isends/irecvs
Task 1 has done 100 isends/irecvs
Task 1 has done 200 isends/irecvs
Task 1 has done 300 isends/irecvs
Task 1 has done 400 isends/irecvs
Task 1 has done 500 isends/irecvs
Task 1 has done 600 isends/irecvs
Task 1 has done 700 isends/irecvs
Task 1 has done 800 isends/irecvs
Task 1 has done 900 isends/irecvs
Task 1 has done 1000 isends/irecvs
Fatal error in PMPI_Waitall: Request pending due to failure, error stack:
PMPI_Waitall(356): MPI_Waitall(count=2000, req_array=0x7fff074001a0, status_array=0x7fff074020e0) failed
PMPI_Waitall(332): The supplied request in array element 0 was invalid (kind=0)
ocelot@ubuntu:~/homework6/homework6_1$
```

修改后的运行结果：

```
[mpiexec@ubuntu] Press Ctrl-C again to force abort
ocelot@ubuntu:~/homework6/homework6$ mpiexec -n 4 ./mpi_bug6
Starting isend/irecv send/irecv test...
Task 0 starting...
Task 1 starting...
Task 2 starting...
Task 3 starting...
Task 0 has done 100 isends/irecvs
Task 0 has done 200 isends/irecvs
Task 0 has done 300 isends/irecvs
Task 0 has done 400 isends/irecvs
Task 0 has done 500 isends/irecvs
Task 0 has done 600 isends/irecvs
Task 0 has done 700 isends/irecvs
Task 0 has done 800 isends/irecvs
Task 0 has done 900 isends/irecvs
Task 0 has done 1000 isends/irecvs
Task 1 has done 100 isends/irecvs
Task 1 has done 200 isends/irecvs
Task 1 has done 300 isends/irecvs
Task 1 has done 400 isends/irecvs
Task 1 has done 500 isends/irecvs
Task 3 has done 100 irecvs
Task 3 has done 200 irecvs
Task 3 has done 300 irecvs
Task 3 has done 400 irecvs
Task 3 has done 500 irecvs
Task 3 has done 600 irecvs
Task 3 has done 700 irecvs
Task 3 has done 800 irecvs
Task 3 has done 900 irecvs
Task 3 has done 1000 irecvs
```

```
ocelot@ubuntu: ~/homework6/homework6
Task 1 has done 400 isends/irecvs
Task 1 has done 500 isends/irecvs
Task 3 has done 100 irecvs
Task 3 has done 200 irecvs
Task 3 has done 300 irecvs
Task 3 has done 400 irecvs
Task 3 has done 500 irecvs
Task 3 has done 600 irecvs
Task 3 has done 700 irecvs
Task 3 has done 800 irecvs
Task 3 has done 900 irecvs
Task 3 has done 1000 irecvs
Task 2 has done 100 sends
Task 1 has done 600 isends/irecvs
Task 1 has done 700 isends/irecvs
Task 1 has done 800 isends/irecvs
Task 1 has done 900 isends/irecvs
Task 1 has done 1000 isends/irecvs
Task 2 has done 200 sends
Task 2 has done 300 sends
Task 2 has done 400 sends
Task 2 has done 500 sends
Task 2 has done 600 sends
Task 2 has done 700 sends
Task 2 has done 800 sends
Task 2 has done 900 sends
Task 2 has done 1000 sends
Task 0 time(wall)= 0.264372 sec
Task 2 time(wall)= 0.207818 sec
Task 1 time(wall)= 0.248458 sec
Task 3 time(wall)= 0.209513 sec
ocelot@ubuntu: ~/homework6/homework6$
```

mpi_bug7.c

这个程序主要是调用mpi_bcast时count参数在不同进程中大小不一样导致的问题，统一成1即可。

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>

int main (int argc, char *argv[])
{
    int    numtasks, taskid, len, buffer, root, count;
    char   hostname[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
    MPI_Comm_rank(MPI_COMM_WORLD, &taskid);
    MPI_Get_processor_name(hostname, &len);

    printf ("Task %d on %s starting...\n", taskid, hostname);
    buffer = 23;
    root = 0;
    count = taskid;
    if (taskid == root)
        printf("Root: Number of MPI tasks is: %d\n", numtasks);

    MPI_Bcast(&buffer, 1, MPI_INT, root, MPI_COMM_WORLD);

    MPI_Finalize();
}
```

修改前的运行结果：


```
ocelot@ubuntu: ~/homework6/homework6_1
[mpiexec@ubuntu] Press Ctrl-C again to force abort
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug6
Starting isend/irecv send/irecv test...
Task 0 starting...
Task 1 starting...
Task 2 starting...
Task 3 starting...
Task 0 has done 100 isends/irecvs
Task 0 has done 200 isends/irecvs
Task 0 has done 300 isends/irecvs
Task 0 has done 400 isends/irecvs
Task 0 has done 500 isends/irecvs
Task 1 has done 100 isends/irecvs
Task 1 has done 200 isends/irecvs
Task 1 has done 300 isends/irecvs
Task 1 has done 400 isends/irecvs
Task 1 has done 500 isends/irecvs
Task 1 has done 600 isends/irecvs
Task 1 has done 700 isends/irecvs
Task 1 has done 800 isends/irecvs
Task 1 has done 900 isends/irecvs
Task 1 has done 1000 isends/irecvs
Fatal error in PMPI_Waitall: Request pending due to failure, error stack:
PMPI_Waitall(356): MPI_Waitall(count=2000, req_array=0x7fff074001a0, status_array=0x7fff074020e0) failed
PMPI_Waitall(332): The supplied request in array element 0 was invalid (kind=0)
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug7
Task 0 on ubuntu starting...
Task 2 on ubuntu starting...
Root: Number of MPI tasks is: 4
Task 3 on ubuntu starting...
Task 1 on ubuntu starting...
```

修改后的运行结果：

```
ocelot@ubuntu: ~/homework6/homework6
Task 3 has done 500 irecvs
Task 3 has done 600 irecvs
Task 3 has done 700 irecvs
Task 3 has done 800 irecvs
Task 3 has done 900 irecvs
Task 3 has done 1000 irecvs
Task 2 has done 100 sends
Task 1 has done 600 isends/irecvs
Task 1 has done 700 isends/irecvs
Task 1 has done 800 isends/irecvs
Task 1 has done 900 isends/irecvs
Task 1 has done 1000 isends/irecvs
Task 2 has done 200 sends
Task 2 has done 300 sends
Task 2 has done 400 sends
Task 2 has done 500 sends
Task 2 has done 600 sends
Task 2 has done 700 sends
Task 2 has done 800 sends
Task 2 has done 900 sends
Task 2 has done 1000 sends
Task 0 time(wall)= 0.264372 sec
Task 2 time(wall)= 0.207818 sec
Task 1 time(wall)= 0.248458 sec
Task 3 time(wall)= 0.209513 sec
ocelot@ubuntu:~/homework6/homework6$ mpiexec -n 4 ./mpi_bug7
Task 0 on ubuntu starting...
Root: Number of MPI tasks is: 4
Task 3 on ubuntu starting...
Task 2 on ubuntu starting...
Task 1 on ubuntu starting...
ocelot@ubuntu:~/homework6/homework6$
```

第二题

这一题的实现我主要使用的是奇偶排序的算法，主要流程如下：

```

Sort local keys;
for (phase = 0; phase < comm sz; phase++) {
    partner = Compute partner(phase, my rank);
    if (I'm not idle) {
        Send my keys to partner;
        Receive keys from partner;
        if (my rank < partner)
            Keep smaller keys;
        else
            Keep larger keys;
    }
}
}

```

编写好的代码如下：

```

#include <stdio>
#include <cstring>
#include <algorithm>
#include <mpi.h>
using namespace std;

#define GLOBAL_N 20

void Generate_list(int local_A[], int local_n, int my_rank) {
    srand(my_rank+1);
    for (int i = 0; i < local_n; i++){
        local_A[i] = random() % 200;
    }
}

void Print_list(int local_A[], int local_n, int rank) {
    for (int i = 0; i < local_n; i++){
        printf("%d ", local_A[i]);
    }
    printf("\n");
}

void Print_global_list(int local_A[], int local_n, int my_rank, int p,
    MPI_Comm comm) {
    int* A;

    if (my_rank == 0) {
        A = new int[p*local_n];
        MPI_Gather(local_A, local_n, MPI_INT, A, local_n, MPI_INT, 0,
            comm);
        printf("\nGlobal list:\n");
        Print_list(A, p*local_n, my_rank);
        delete[] A;
    }
    else {
        MPI_Gather(local_A, local_n, MPI_INT, A, local_n, MPI_INT, 0,
            comm);
    }
}

void Odd_even_iter(int local_A[], int temp_B[], int temp_C[],

```

```

    int local_n, int phase, int even_partner, int odd_partner,
    int my_rank, int p, MPI_Comm comm) {
MPI_Status status;

if (phase % 2 == 0) {
    if (even_partner >= 0) {
        MPI_Sendrecv(local_A, local_n, MPI_INT, even_partner, 0,
            temp_B, local_n, MPI_INT, even_partner, 0, comm,
            &status);
        merge(local_A, local_A+local_n, temp_B, temp_B+local_n, temp_C);
        if (my_rank % 2 != 0){
            copy(temp_C+local_n, temp_C+local_n*2, local_A);
        }
        else{
            copy(temp_C, temp_C+local_n, local_A);
        }
    }
} else {
    if (odd_partner >= 0) {
        MPI_Sendrecv(local_A, local_n, MPI_INT, odd_partner, 0,
            temp_B, local_n, MPI_INT, odd_partner, 0, comm,
            &status);
        merge(local_A, local_A+local_n, temp_B, temp_B+local_n, temp_C);
        if (my_rank % 2 != 0){
            copy(temp_C, temp_C+local_n, local_A);
        }
        else{
            copy(temp_C+local_n, temp_C+2*local_n, local_A);
        }
    }
}
}

void Sort(int local_A[], int local_n, int my_rank,
    int p, MPI_Comm comm) {
    int phase;
    int *temp_B, *temp_C;
    int even_partner; /* phase is even or left-looking */
    int odd_partner; /* phase is odd or right-looking */

    /* Temporary storage used in merge-split */
    temp_B = new int[local_n];
    temp_C = new int[local_n*2];

    /* Find partners: negative rank => do nothing during phase */
    if (my_rank % 2 != 0) {
        even_partner = my_rank - 1;
        odd_partner = my_rank + 1;
        if (odd_partner == p) odd_partner = MPI_PROC_NULL; // Idle during odd
phase
    } else {
        even_partner = my_rank + 1;
        if (even_partner == p) even_partner = MPI_PROC_NULL; // Idle during even
phase
        odd_partner = my_rank-1;
    }

    /* Sort local list using built-in sort */

```

```

    sort(local_A, local_A+local_n);

    for (phase = 0; phase < p; phase++)
        odd_even_iter(local_A, temp_B, temp_C, local_n, phase,
            even_partner, odd_partner, my_rank, p, comm);

    delete [] temp_B;
    delete [] temp_C;
} /* Sort */

int main(int argc, char* argv[]) {
    int my_rank, p;

    MPI_Comm comm;

    MPI_Init(&argc, &argv);
    comm = MPI_COMM_WORLD;
    MPI_Comm_size(comm, &p);
    MPI_Comm_rank(comm, &my_rank);
    int local_n = GLOBAL_N/p;

    int * local_A = new int[local_n];
    Generate_list(local_A, local_n, my_rank);
    for ( int i = 0; i < p; i++ ){
        if ( i == my_rank ){
            printf("%d: ", my_rank);
            Print_list(local_A, local_n, my_rank);
        }
        MPI_Barrier(comm);
    }

    Sort(local_A, local_n, my_rank, p, comm);

    Print_global_list(local_A, local_n, my_rank, p, comm);

    delete [] local_A;

    MPI_Finalize();

    return 0;
}

```

由于使用的是C++编程，很多步骤可以调用库实现，如 `std::merge()`，`std::sort()` 等等。

运行结果如下：

```
ocelot@ubuntu: ~/homework6/homework6
Task 1 has done 400 isends/irecvs
Task 1 has done 500 isends/irecvs
Task 1 has done 600 isends/irecvs
Task 1 has done 700 isends/irecvs
Task 1 has done 800 isends/irecvs
Task 1 has done 900 isends/irecvs
Task 1 has done 1000 isends/irecvs
Fatal error in PMPI_Waitall: Request pending due to failure, error stack:
PMPI_Waitall(356): MPI_Waitall(count=2000, req_array=0x7fff074001a0, status_array=0x7fff074020e0) failed
PMPI_Waitall(332): The supplied request in array element 0 was invalid (kind=0)
ocelot@ubuntu:~/homework6/homework6_1$ mpiexec -n 4 ./mpi_bug7
Task 0 on ubuntu starting...
Task 2 on ubuntu starting...
Root: Number of MPI tasks is: 4
Task 3 on ubuntu starting...
Task 1 on ubuntu starting...
^C[mpiexec@ubuntu] Sending Ctrl-C to processes as requested
[mpiexec@ubuntu] Press Ctrl-C again to force abort
ocelot@ubuntu:~/homework6/homework6_1$ cd ..
ocelot@ubuntu:~/homework6$ cd homework6
ocelot@ubuntu:~/homework6$ ls
6.c      mpi_bug1.c  mpi_bug2.c  mpi_bug3.c  mpi_bug4.c  mpi_bug5.c  mpi_bug6.c  mpi_bug7.c  sort.cpp
mpi_bug1  mpi_bug2    mpi_bug3    mpi_bug4    mpi_bug5    mpi_bug6    mpi_bug7    sort
ocelot@ubuntu:~/homework6/homework6$ mpiexec -n 4 ./sort
0: 183 86 177 115 193
1: 90 119 188 175 61
2: 146 185 168 40 25
3: 101 83 74 126 163

Global list:
25 40 61 74 83 86 90 101 115 119 126 146 163 168 175 177 183 185 188 193
ocelot@ubuntu:~/homework6/homework6$
```

可见成功实现了排序算法。