2023 年春季超级计算原理与实践 MPI 编程作业

1. 请使用 MPI 函数库实现简单的乒乓游戏。要求使用两个进程模拟两位球手，并使用 MPI_Send 和 MPI_Recv 方法来“推挡”信息。该信息在代码中用变量 ping_pong_count 表示，两个进程会轮流成为发送者和接受者，发送者每次发送信息之后 ping_pong_count 都会递增 1。部分代码已经给出，请同学们补充完整并打印出每个回合下发送者和接收者的进程标号以及对应的 ping_pong_count 值。示例：

   Process 0 sent and incremented ping_pong_count(value=1)to process1

   Process 0 received ping_pong_count(value=2)from process 1

   C 源码：

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

int main(int argc, char** argv) {
  const int PING_PONG_LIMIT = 10;

  // Initialize the MPI environment
  MPI_Init(NULL, NULL);
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size);

  // We are assuming 2 processes for this task
  if (world_size != 2) {
    fprintf(stderr, "World size must be two for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  int ping_pong_count = 0;
  //PLEASE ADD THE RIGHT CODES
  int another=(world_rank+1)%2;
  int times=PING_PONG_LIMIT;
  while(times--){ //两个进程轮流交换信息
    if(world_rank==ping_pong_count%2){ //看谁发送 从初始0开始 双数发送，单数接收
        ping_pong_count++;
        printf("Process %d sent and incremented ping_pong_count(value=%d) to process %d\n",world_rank,ping_pong_count,another);
        MPI_Send(&ping_pong_count,1,MPI_INT,another,0,MPI_COMM_WORLD);
    }
```

```
    else{
        printf("Process %d received ping_pong_count(value=%d) from
process %d\n",world_rank,ping_pong_count,another);
        MPI_Recv(&ping_pong_count,1,MPI_INT,another,0,MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    }
  }
  MPI_Finalize();
  return 0;
}
```

运行结果：

总共运行三次，每次输出结果都不同，但含义是相同的，这是由并行执行导致的输出结果不确定。

-
```
======== OUTPUT ========
Process 0 sent and incremented ping_pong_count(value=1) to process 1
Process 0 received ping_pong_count(value=1) from process 1
Process 0 sent and incremented ping_pong_count(value=3) to process 1
Process 0 received ping_pong_count(value=3) from process 1
Process 1 received ping_pong_count(value=0) from process 0
Process 1 sent and incremented ping_pong_count(value=2) to process 0
Process 1 received ping_pong_count(value=2) from process 0
Process 1 sent and incremented ping_pong_count(value=4) to process 0
Process 1 received ping_pong_count(value=4) from process 0
Process 1 sent and incremented ping_pong_count(value=6) to process 0
Process 1 received ping_pong_count(value=6) from process 0
Process 1 sent and incremented ping_pong_count(value=8) to process 0
Process 1 received ping_pong_count(value=8) from process 0
Process 1 sent and incremented ping_pong_count(value=10) to process 0
Process 0 sent and incremented ping_pong_count(value=5) to process 1
Process 0 received ping_pong_count(value=5) from process 1
Process 0 sent and incremented ping_pong_count(value=7) to process 1
Process 0 received ping_pong_count(value=7) from process 1
Process 0 sent and incremented ping_pong_count(value=9) to process 1
Process 0 received ping_pong_count(value=9) from process 1
```

```
======== OUTPUT ========
Process 0 sent and incremented ping_pong_count(value=1) to process 1
Process 0 received ping_pong_count(value=1) from process 1
Process 1 received ping_pong_count(value=0) from process 0
Process 1 sent and incremented ping_pong_count(value=2) to process 0
Process 1 received ping_pong_count(value=2) from process 0
Process 1 sent and incremented ping_pong_count(value=4) to process 0
Process 1 received ping_pong_count(value=4) from process 0
Process 1 sent and incremented ping_pong_count(value=6) to process 0
Process 1 received ping_pong_count(value=6) from process 0
Process 1 sent and incremented ping_pong_count(value=8) to process 0
Process 0 sent and incremented ping_pong_count(value=3) to process 1
Process 0 received ping_pong_count(value=3) from process 1
Process 0 sent and incremented ping_pong_count(value=5) to process 1
Process 0 received ping_pong_count(value=5) from process 1
Process 0 sent and incremented ping_pong_count(value=7) to process 1
Process 0 received ping_pong_count(value=7) from process 1
Process 0 sent and incremented ping_pong_count(value=9) to process 1
Process 0 received ping_pong_count(value=9) from process 1
Process 1 received ping_pong_count(value=8) from process 0
Process 1 sent and incremented ping_pong_count(value=10) to process 0
```

```
======== OUTPUT ========
Process 0 sent and incremented ping_pong_count(value=1) to process 1
Process 0 received ping_pong_count(value=1) from process 1
Process 0 sent and incremented ping_pong_count(value=3) to process 1
Process 0 received ping_pong_count(value=3) from process 1
Process 1 received ping_pong_count(value=0) from process 0
Process 1 sent and incremented ping_pong_count(value=2) to process 0
Process 1 received ping_pong_count(value=2) from process 0
Process 1 sent and incremented ping_pong_count(value=4) to process 0
Process 1 received ping_pong_count(value=4) from process 0
Process 1 sent and incremented ping_pong_count(value=6) to process 0
Process 0 sent and incremented ping_pong_count(value=5) to process 1
Process 0 received ping_pong_count(value=5) from process 1
Process 0 sent and incremented ping_pong_count(value=7) to process 1
Process 0 received ping_pong_count(value=7) from process 1
Process 0 sent and incremented ping_pong_count(value=9) to process 1
Process 0 received ping_pong_count(value=9) from process 1
Process 1 received ping_pong_count(value=6) from process 0
Process 1 sent and incremented ping_pong_count(value=8) to process 0
Process 1 received ping_pong_count(value=8) from process 0
Process 1 sent and incremented ping_pong_count(value=10) to process 0
```

2. 请使用 MPI 函数库实现数组求和的实例。要求进程间通过树结构通信的方式输出求和结果。部分代码已经给出，请同学们补充完整，需要补充的部分参见注释 PLEASE ADD THE RIGHT CODES。本例程中可以首先考虑简单情况，comm_sz 是 2 的幂；进一步可考虑 comm_sz 是任意正整数。

C 源码：

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

int Global_sum(int my_int, int my_rank, int comm_sz, MPI_Comm comm);

const int MAX_CONTRIB = 20;

int main(void) {
    int i, sum, my_int;
    int my_rank, comm_sz;
    MPI_Comm comm;
    int* all_ints = NULL;

    MPI_Init(NULL, NULL);
    comm = MPI_COMM_WORLD;
    MPI_Comm_size(comm, &comm_sz);
    MPI_Comm_rank(comm, &my_rank);

    srandom(my_rank + 1);
    my_int = random() % MAX_CONTRIB;

    sum = Global_sum(my_int, my_rank, comm_sz, comm);

    if ( my_rank == 0) {
        all_ints = malloc(comm_sz*sizeof(int));
        MPI_Gather(&my_int, 1, MPI_INT, all_ints, 1, MPI_INT, 0, comm);
        printf("Ints being summed:\n   ");
        for (i = 0; i < comm_sz; i++)
            printf("%d ", all_ints[i]);
        printf("\n");
        printf("Sum = %d\n",sum);
        free(all_ints);
    }
    else {
        MPI_Gather(&my_int, 1, MPI_INT, all_ints, 1, MPI_INT, 0, comm);
    }

    MPI_Finalize();
```

```
    return 0;
}  /* main */

int Global_sum(
    int my_int    /* in */, //当前进程的数值
    int my_rank   /* in */,
    int comm_sz   /* in */,
    MPI_Comm comm /* in */) {

   //PLEASE ADD THE RIGHT CODES
   int remain=comm_sz,my_sum=my_int,temp;
   int half,rm;
   while (remain!=1){
        half=remain/2;
        rm=remain%2;
        if(my_rank<half){
            MPI_Recv(&temp,1,MPI_INT,my_rank+half+rm,0,comm,MPI_STATUS_I
GNORE);
            my_sum+=temp;
        //前半部分接受后半部分送来的值
        }
        else{
            MPI_Send(&my_sum,1,MPI_INT,my_rank-half-rm,0,comm);
        //return;
        //后半部分发送值
        }
        remain=half+rm;
    }
    if(my_rank==0) return my_sum;
}  /* Global_sum * /
```

运行结果:

核数为 1:

```
Ints being summed:
7
Sum = 7
```

核数为 2:

```
Ints being summed:
7 8
Sum = 15
```

核数为 4:

```
Ints being summed:
7 8 7 0
Sum = 22
```

核数为 8:

```
Ints being summed:
7 8 7 0 1 1 14 14
Sum = 52
```

核数为 16:

```
Ints being summed:
7 8 7 0 1 1 14 14 14 7 8 8 0 1 1 14
Sum = 105
```

经检验，以上以树形结构通信计算数组和的结果正确

3. 请使用 MPI 函数库实现圆周率π的并行计算。（提示：可考虑π的数值计算公式，转化为无穷级数或者定积分的计算。譬如，$\frac{\pi}{4}=\int_0^1 \frac{1}{1+x^2}dx$）

根据题目提示，我在 n 个核中计算 4/1+x²的值并在最后除 n 取平均。当 n 越大，计算结果就越接近Π 代码如下

C 源代码:

```c
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc,char *argv[]){
  int my_rank,num_procs;
  int i,n=0;
  double sum,width,local,mypi,pi;
  int proc_len;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  MPI_Init(&argc,&argv);
  MPI_Comm_size(MPI_COMM_WORLD,&num_procs);
  MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
  MPI_Get_processor_name(processor_name,&proc_len);
  printf("Processor %d of %d on %s\n",my_rank,num_procs,processor_name);
  if(my_rank==0){
    printf("pls give n=");
    scanf("%d",&n);}
  MPI_Bcast(&n,1,MPI_INT,0,MPI_COMM_WORLD);
  width=1.0/n;
  sum=0.0;
  for(i=my_rank;i<n;i+=num_procs){
    local=width*((double)i+0.5);
    sum+=4.0/(1.0+local*local);
  }
  mypi=width*sum;
  MPI_Reduce(&mypi,&pi,1,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
  if(my_rank==0){
    printf("PI is %.20f\n",pi);
    fflush(stdout);
  }
  MPI_Finalize();
  return 0;}
```

N=1:

```
Processor 0 of 1 on 99a141b95783
pls give n=PI is 3.20000000000000017764
```

N=5:

```
Processor 0 of 1 on 99a141b95783
pls give n=PI is 3.14492586400332774232
```

N=10:

```
Processor 0 of 1 on 99a141b95783
pls give n=PI is 3.14242598500109870940
```

N=20:

```
Processor 0 of 1 on 99a141b95783
pls give n=PI is 3.14180098689309428295
```

N=40:

```
Processor 0 of 1 on 99a141b95783
pls give n=PI is 3.14164473692265744376
```

N=100:

```
Processor 0 of 1 on 99a141b95783
pls give n=PI is 3.14160098692312539370
```

与预期结果一致