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：

#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);

    }

  }

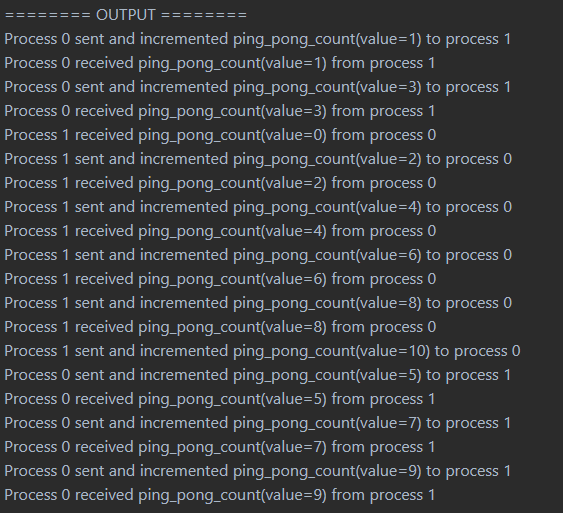
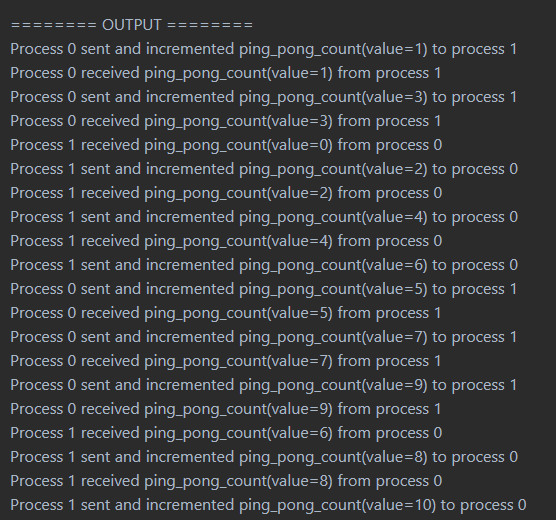
  MPI\_Finalize();

  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。

* 
* 
* 

1. 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：

#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\_IGNORE);

            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;

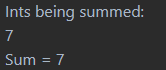
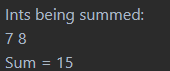
    }

    if(my\_rank==0) return my\_sum;

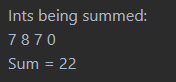
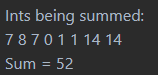
}  /\* Global\_sum \* /

运行结果：

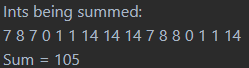
核数为1： 核数为2：

核数为4： 核数为8：

核数为16：



经检验，以上以树形结构通信计算数组和的结果正确

1. 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，）

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：

#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：



N=5：



N=10：



N=20：



N=40：



N=100：



与预期结果一致