1、乒乓游戏

代码

```
/*
 * Purpose:  Using MPI to imitate a table tennis match.
 *
 * Compile:  mpicc -g -Wall -o ex1_ping_pong ex1_ping_pong.c
 * Run:      mpiexec ./ex1_ping_pong
 *
 * Input:    None
 * Output:   None
 *
 * Note:    On each turn, please print out the rank for each process and the value of
"ping_pong_count".
 */

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

int main(int argc, char **argv)
{
  const int PING_PONG_LIMIT = 10; // 限定交互次数

  // Initialize the MPI environment
  MPI_Init(NULL, NULL); // 初始化 MPI
  // Find out rank, size
  int world_rank;
  MPI_Comm_rank(MPI_COMM_WORLD, &world_rank); // 获取当前进程的编号
  int world_size;
  MPI_Comm_size(MPI_COMM_WORLD, &world_size); // 获取进程总数

  // We are assuming 2 processes for this task
  if (world_size != 2) // 限定只有两个进程
  {
    fprintf(stderr, "World size must be two for %s\n", argv[0]);
    MPI_Abort(MPI_COMM_WORLD, 1);
  }

  int ping_pong_count = 0;
  /*****************************************************************/

  int partner_rank = (world_rank + 1) % 2;
  while (ping_pong_count < PING_PONG_LIMIT)
```

```
  {
    if (world_rank == ping_pong_count % 2) // 判断当前进程是否需要发送
    {

      ping_pong_count++;
      MPI_Send(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD);
// 发送交互次数给对方进程
      printf("Process %d sent and incremented ping_pong_count(value=%d) to process
%d\n", world_rank, ping_pong_count, partner_rank);
    }
    else
    {
      MPI_Recv(&ping_pong_count, 1, MPI_INT, partner_rank, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);    // 接收对方进程发送的交互次数
      printf("Process %d received ping_pong_count(value=%d) from process %d\n",
world_rank, ping_pong_count, partner_rank); // 输出接收信息
    }
  }

  /*********************************************************************/
  MPI_Finalize();
  return 0;
}
```

运行

```
(base) lin@LindeMacBook-Pro Code_超算 % mpicc -o ex1_ping_pong ex1_ping_pong.c
(base) lin@LindeMacBook-Pro Code_超算 % mpirun -np 2 ./ex1_ping_pong
Process 1 received ping_pong_count(value=1) from process 0
Process 0 sent and incremented ping_pong_count(value=1) to process 1
Process 0 received ping_pong_count(value=2) from process 1
Process 0 sent and incremented ping_pong_count(value=3) to process 1
Process 0 received ping_pong_count(value=4) from process 1
Process 0 sent and incremented ping_pong_count(value=5) to process 1
Process 1 sent and incremented ping_pong_count(value=2) to process 0
Process 1 received ping_pong_count(value=3) from process 0
Process 1 sent and incremented ping_pong_count(value=4) to process 0
Process 1 received ping_pong_count(value=5) from process 0
Process 1 sent and incremented ping_pong_count(value=6) to process 0
Process 0 received ping_pong_count(value=6) from process 1
Process 0 sent and incremented ping_pong_count(value=7) to process 1
Process 0 received ping_pong_count(value=8) from process 1
Process 1 received ping_pong_count(value=7) 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=9) to process 1
Process 0 received ping_pong_count(value=10) from process 1
Process 1 received ping_pong_count(value=9) from process 0
Process 1 sent and incremented ping_pong_count(value=10) to process 0
(base) lin@LindeMacBook-Pro Code_超算 %
```

2、数组求和

代码

```
/*
 * Purpose:  Use tree-structured communication to find the global sum
 *           of a random collection of ints
 *
 * Compile:  mpicc -g -Wall -o ex2_tree_sum ex2_tree_sum.c
 * Run:      mpiexec -n <comm_sz> ./ex2_tree_sum
 *
 * Input:    None
 * Output:   Random values generated by processes, and their global sum.
 *
 * Note:     This version assumes comm_sz is a power of 2.
 */

#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 */

/*---------------------------------------------------------------------
 * Function:   Global_sum
 * Purpose:    Implement a global sum using tree-structured communication
 * Notes:
 * 1.  comm_sz must be a power of 2
 * 2.  The return value is only valid on process 0
 */
int Global_sum(
  int my_int /* in */,  // 随机数
  int my_rank /* in */, // 自身任务ID
  int comm_sz /* in */, // MIP任务总数
  MPI_Comm comm /* in */) // MIP通信器
{

  /*********************************************************************/

  int num = comm_sz;
  int half;
  int recieve_data;
  int remainder;
  int my_sum=my_int;

  while (num > 1)
  {
    half = num / 2;
    remainder = num % 2;

    if (my_rank < half)
    {
      MPI_Recv(&recieve_data, 1, MPI_INT, my_rank + half + remainder, 0, comm,
MPI_STATUS_IGNORE);
      my_sum+=recieve_data;
    }
```

```
    if (my_rank > half - 1 + remainder)
    {
      MPI_Send(&my_int, 1, MPI_INT, my_rank - half - remainder, 0, comm);
    }
  }

  /*****************************************************************/

  return my_sum;
} /* Global_sum */
```

运行

```
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 1 ./ex2_tree_sum
Ints being summed:
   3
Sum = 3
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 2 ./ex2_tree_sum
Ints being summed:
   3 10
Sum = 13
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 3 ./ex2_tree_sum
Ints being summed:
   3 10 6
Sum = 19
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 4 ./ex2_tree_sum
Ints being summed:
   3 10 6 1
Sum = 20
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 5 ./ex2_tree_sum
Ints being summed:
   3 10 6 1 15
Sum = 35
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 6 ./ex2_tree_sum
Ints being summed:
   3 10 6 1 15 1
Sum = 36
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 7 ./ex2_tree_sum
Ints being summed:
   3 10 6 1 15 1 17
Sum = 53
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 8 ./ex2_tree_sum
Ints being summed:
   3 10 6 1 15 1 17 16
Sum = 69
(base) lin@LindeMacBook-Pro Code_超算 %
```

3、圆周率π的并行计算

代码

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

#define N 9999999

int main(int argc, char *argv[])
{
    int comm_sz, my_rank;
    double my_sum = 0.0, pi = 0.0;
    int i;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &comm_sz);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_rank);

    for (i = my_rank; i < N; i += comm_sz)
    {
        my_sum += sqrt(1 - ((double)i / N) * ((double)i / N)) / N;
    }
    MPI_Reduce(&my_sum, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (my_rank == 0)
    {
        pi *= 4.0;
        printf("调用%d个进程求得近似的pi值为:%.15f\n",comm_sz, pi);
    }

    MPI_Finalize();
    return 0;
}
```

运行

```
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 1 ./ex3_pi
调用1个进程求得近似的pi值为:3.141594652413821
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 2 ./ex3_pi
调用2个进程求得近似的pi值为:3.141594652413728
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 3 ./ex3_pi
调用3个进程求得近似的pi值为:3.141594652413801
(base) lin@LindeMacBook-Pro Code_超算 % mpiexec -n 4 ./ex3_pi
调用4个进程求得近似的pi值为:3.141594652413807
```