《并行计算》上机报告

姓名：	魏钊	学号：	PB18111699	日期：	2021.5.16
上机题目：	MPI 并行编程实验				

实验环境：
CPU：i7-8750H ； 内存: 16GB;操作系统：Ubuntu 20.04 ;软件平台：Visual Studio 2017;

一、算法设计与分析：

题目一:
用 MPI 编程实现 PI 的计算。
各进程独立计算，最后将结果归约到根进程。

题目二：
用 MPI 实现 PSRS 排序
begin
(1)均匀划分：将 n 个元素 A[1..n]均匀划分成 p 段，每个 pi 处理 A[(i-1)n/p+1..in/p]
(2)局部排序： pi 调用串行排序算法对 A[(i-1)n/p+1..in/p]排序
(3)选取样本： pi 从其有序子序列 A[(i-1)n/p+1..in/p]中选取 p 个样本元素
(4)样本排序：用一台处理器对 p
2 个样本元素进行串行排序
(5)选择主元：用一台处理器从排好序的样本序列中选取 p-1 个主元，并播送给其他 pi
(6)主元划分： pi 按主元将有序段 A[(i-1)n/p+1..in/p]划分成 p 段
(7)全局交换：各处理器将其有序段按段号交换到对应的处理器中
(8)归并排序：各处理器对接收到的元素进行归并排序
end.

二、核心代码：

题目一:

```
23      // Broadcast n to all other nodes
24          MPI_Bcast(&n, 1, MPI_LONG, 0, MPI_COMM_WORLD);
25          h = 1.0 / (double)n;
26          sum = 0.0;
27          for (i = my_rank + 1; i <= n; i += group_size)
28          {
29              x = h * (i - 0.5);
30              sum = sum + 4.0 / (1.0 + x * x);
31          }
32          mypi = h * sum;
33      //Global sum
34          MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

题目二：

```
88      MPI_Barrier(MPI_COMM_WORLD);
89      sort(a + id * seg, a + id * seg + seg);//局部排序
90      for (i = 0; i < ProcNum; i++)//正则采样
91      {
92          sample[id*NUM_PROCS + i] = a[id*seg + i * seg / NUM_PROCS];
93      }
94      MPI_Barrier(MPI_COMM_WORLD);
95      //debug
96      /* ... */
156     //收集采样
157     if(id!=0)
158         MPI_Send(&sample[id*NUM_PROCS], NUM_PROCS, MPI_INT, 0,0,MPI_COMM_WORLD);
159     if (id == 0)
160     {
161         for (i = 1; i < ProcNum; i++)
162             MPI_Recv(&sample[i*NUM_PROCS], NUM_PROCS, MPI_INT, i,0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
163         // ...
165         /* ... */
172         //采样排序
173         sort(sample, sample + NUM_PROCS * NUM_PROCS);
174         for (i = 0; i < NUM_PROCS - 1; i++)//选择主元
175         {
176             pivot[i] = sample[i*NUM_PROCS + NUM_PROCS];
177         }
178     }
179     MPI_Barrier(MPI_COMM_WORLD);
```

```
211     //广播主元
212     MPI_Bcast(pivot, NUM_PROCS - 1, MPI_INT, 0, MPI_COMM_WORLD);//广播主元
213     MPI_Barrier(MPI_COMM_WORLD);
214
215     //debug
216     /* ... */
228     //主元划分
229     j = 0;
230     k = 0;
231     for (i = 0; i < seg; i++)
232     {
233         if (a[id*seg + i] <= pivot[j] && j < NUM_PROCS - 1)
234         {
235             b[id][j][k] = a[id*seg + i];
236             c[id][j]++;
237             k++;
238         }
239         else
240         {
241             if (j < NUM_PROCS - 1)
242             {
243                 k = 0;
244                 j++;
245                 b[id][j][k] = a[id*seg + i];
246                 c[id][j]++;
247                 k++;
248             }
249             else
250             {
251                 b[id][j][k] = a[id*seg + i];
252                 c[id][j]++;
253                 k++;
254             }
255         }
256     }
```

```
257     //收集计数
258     if (id != 0)
259     {
260         MPI_Send(&c[id], NUM_PROCS, MPI_INT, 0, 1, MPI_COMM_WORLD);
261     }
262     else
263     {
264         for (i = 1; i < ProcNum; i++)
265             MPI_Recv(&c[i], NUM_PROCS, MPI_INT, i, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
266     }
267     MPI_Barrier(MPI_COMM_WORLD);
268     if (id == 0)
269     {
270         /* ... */
279         for (i = 1; i < NUM_PROCS; i++)
280         {
281             MPI_Send(c, NUM_PROCS*NUM_PROCS, MPI_INT, i, 2, MPI_COMM_WORLD);//广播计数
282         }
283     }
284     else
285     {
286         MPI_Recv(c, NUM_PROCS*NUM_PROCS, MPI_INT, 0, 2, MPI_COMM_WORLD, MPI_STATUS_IGNORE);//各进程接受计数
287     }
288     MPI_Barrier(MPI_COMM_WORLD);
```

```
303     //将分组发送到根进程
304     if (id != 0)
305     {
306         MPI_Send(&b[id], NUM_PROCS * 100, MPI_INT, 0, 3, MPI_COMM_WORLD);
307     }
308     else
309     {
310         for (i = 1; i < NUM_PROCS; i++)
311         {
312             MPI_Recv(&b[i], NUM_PROCS * 100, MPI_INT, i, 3, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
313         }
314     }
315     MPI_Barrier(MPI_COMM_WORLD);
316     //debug
317     /* ... */
332     //将分组广播
333     if (id == 0)
334     {
335         for (i = 1; i < NUM_PROCS; i++)
336         {
337             MPI_Send(b, NUM_PROCS*NUM_PROCS * 100, MPI_INT, i, 4, MPI_COMM_WORLD);
338         }
339     }
340     else
341     {
342         MPI_Recv(b, NUM_PROCS*NUM_PROCS * 100, MPI_INT, 0, 4, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
343     }
344     MPI_Barrier(MPI_COMM_WORLD);
```

```
361     //计算各段长度
362     for (i = 0; i < NUM_PROCS; i++)
363         for (j = 0; j < NUM_PROCS; j++)
364         {
365             Len[i] = c[j][i] + Len[i];
366         }
367     //新划分间隔
368     L = 0;
369     for (i = 0; i < id; i++)
370         L = L + Len[i];
371     i = 0;
372     for (j = 0; j < NUM_PROCS; j++)
373     {
374         for (k = 0;; k++)
375         {
376             if (k < c[j][id])
377             {
378                 Final[L + i] = b[j][id][k];
379                 i++;
380             }
381             else
382                 break;
383         }
384     }
385     sort(Final + L, Final + L + Len[id]);//局部排序
386     MPI_Barrier(MPI_COMM_WORLD);
```

```
387     if (id != 0)
388     {
389         MPI_Send(&Final[L], Len[id], MPI_INT, 0, 5, MPI_COMM_WORLD);
390     }
391     else
392     {
393         for (i = 1; i < NUM_PROCS; i++)
394         {
395             L = L + Len[i - 1];
396             MPI_Recv(&Final[L], Len[i], MPI_INT, i, 5, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
397         }
398         cout << "排序结果：";
399         for (i = 0; i < len; i++)
400             cout << Final[i] << " ";
401             cout << "\n";
402     }
403     MPI_Barrier(MPI_COMM_WORLD);
```

三、结果与分析：

题目一：

```
vmware@ubuntu:~$ mpirun -np 3 ./pi
pi is approximately : 3.1415926744231273
```

题目二：

```
vmware@ubuntu:~$ mpirun -np 3 ./psrs
请输入数组大小: 32
随机数为: 51 25 38 50 17 35 88 27 58 88 13 29 48 60 26 47 8 26 80 69 34 9 50 57 23 99 98 74 0 49
 44 3
排序结果: 0 3 8 9 13 17 23 25 26 26 27 29 34 35 38 44 47 48 49 50 50 51 57 58 60 69 74 80 88 88
98 99
答案验证: 0 3 8 9 13 17 23 25 26 26 27 29 34 35 38 44 47 48 49 50 50 51 57 58 60 69 74 80 88 88
98 99
```

四、备注（* 可选）：

有可能影响结论的因素：

默认为 3 进程并行。

总结：

MPI 通过进程级并行提高计算效率，通过进程间通信交换数据。

附录（源代码）

算法源代码（C/C++/JAVA 描述）

PI 的代码由 PPT 提供这里不再描述。

```
#include <stdio.h>
#include <omp.h>
#include <stdlib.h>
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <iostream>
#include <mpi.h>
#include<limits.h>
using namespace std;
#define NUM_PROCS 3

int INF = INT_MAX;//无穷大
void PSRS(int *a, int len,int len_t);

int main(int argc, char* argv[])
{
    int localPID;
    int Num;//进程数
    int n, i, j, k;//数组大小
    int n_t;//处理不是进程倍数
    MPI_Init(&argc, &argv);//初始化
    MPI_Comm_rank(MPI_COMM_WORLD, &localPID);//本地进程号
    MPI_Comm_size(MPI_COMM_WORLD, &Num);//总进程数
    if (localPID == 0)
```

```
{
    cout << "请输入数组大小：";
    cin >> n;
    if (n%NUM_PROCS != 0)
        n_t = NUM_PROCS - n % NUM_PROCS;
    else
        n_t = 0;

}

MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&n_t, 1, MPI_INT, 0, MPI_COMM_WORLD);
srand((int)time(0));//随机数种子
int *a = new int[n+n_t];

for (i = 0; i < n; i++)
{
    a[i] = rand() % 100;
}
for (i = n; i < n + n_t; i++)
{
    a[i] = INF;
}
if (localPID == 0)
{
    cout << "随机数为：";
    for (i = 0; i < n; i++)
    {
        cout << a[i] << " ";
    }
    cout << "\n";
}
MPI_Barrier(MPI_COMM_WORLD);
PSRS(a, n, n_t);
sort(a, a + n);
if (localPID == 0)
{
    cout << "答案验证：";
    for (i = 0; i < n; i++)
        cout << a[i] << " ";
    cout << "\n";
}
MPI_Finalize();
return 0;
```

```
}

void PSRS(int *a, int len,int len_t)
{
    int id, ProcNum, i, j, k, L;
    int seg;//分段
    int *sample = new int[NUM_PROCS*NUM_PROCS];//采样
    int *pivot = new int[NUM_PROCS - 1];//主元
    int b[NUM_PROCS][NUM_PROCS][100] = { 0 };//主元划分
    int c[NUM_PROCS][NUM_PROCS] = { 0 };//统计数目
    int Len[NUM_PROCS] = { 0 };//划分后各段长度
    int *Final = new int[len];//排序结果
    MPI_Comm_size(MPI_COMM_WORLD, &ProcNum);
    MPI_Comm_rank(MPI_COMM_WORLD, &id);
    MPI_Status status[ProcNum];
    MPI_Request request[ProcNum];
    seg = (len + len_t) / ProcNum;

    MPI_Barrier(MPI_COMM_WORLD);
    sort(a + id * seg, a + id * seg + seg);//局部排序
    for (i = 0; i < ProcNum; i++)//正则采样
    {
        sample[id*NUM_PROCS + i] = a[id*seg + i * seg / NUM_PROCS];
    }
    MPI_Barrier(MPI_COMM_WORLD);
    //debug
    /*
    if (id == 0)
    {
        cout << "id=" << id << "\n";
        for (i = 0; i < len; i++)
        {
            cout << a[i] << " ";
        }
        cout << "\n";
        for (i = 0; i < ProcNum; i++)
        {
            cout << sample[id*NUM_PROCS + i] << " ";
        }
        cout << "\n";
    }
    MPI_Barrier(MPI_COMM_WORLD);
    if (id == 1)
    {
```

```
        cout << "id=" << id << "\n";
        for (i = 0; i < len; i++)
        {
            cout << a[i] << " ";
        }
        cout << "\n";
        for (i = 0; i < ProcNum; i++)
        {
            cout << sample[id*NUM_PROCS + i] << " ";
        }
        cout << "\n";
    }
    MPI_Barrier(MPI_COMM_WORLD);
    if (id == 2)
    {
        cout << "id=" << id << "\n";
        for (i = 0; i < len; i++)
        {
            cout << a[i] << " ";
        }
        cout << "\n";
        for (i = 0; i < ProcNum; i++)
        {
            cout << sample[id*NUM_PROCS + i] << " ";
        }
        cout << "\n";
    }
    MPI_Barrier(MPI_COMM_WORLD);


    if (id == 0)
    {
        cout << "id=" << id << "\n";
        cout << "sample=" ;
        for (i = 0; i < ProcNum*ProcNum ; i++)
        {
            cout << sample[i] << " ";
        }
        cout << "\n";
    }
    MPI_Barrier(MPI_COMM_WORLD);
    */
    //收集采样
    if(id!=0)
```

```
        MPI_Send(&sample[id*NUM_PROCS], NUM_PROCS, MPI_INT,
0,0,MPI_COMM_WORLD);
    if (id == 0)
    {
        for (i = 1; i < ProcNum; i++)
            MPI_Recv(&sample[i*NUM_PROCS], NUM_PROCS, MPI_INT, i,0,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        //cout << "sample2=";
        //debug
        /*
        for (i = 0; i < ProcNum*ProcNum; i++)
        {
            cout << sample[i] << " ";
        }
        cout << "\n";
        */
        //采样排序
        sort(sample, sample + NUM_PROCS * NUM_PROCS);
        for (i = 0; i < NUM_PROCS - 1; i++)//选择主元
        {
            pivot[i] = sample[i*NUM_PROCS + NUM_PROCS];
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    //debug
    /*
    if (id == 0)
    {
        cout << "new sample=";
        for (i = 0; i < ProcNum*ProcNum; i++)
        {
            cout << sample[i] << " ";
        }
        cout << "\n";
        cout << "主元=";
        for (i = 0; i < NUM_PROCS - 1; i++)//选择主元
        {
            cout << pivot[i] << " ";
        }
        cout << "\n";
    }
    */
    /*
    MPI_Barrier(MPI_COMM_WORLD);
```

```
    if (id == 1)
    {
        cout << "id_1" << "\n";
        for (i = 0; i < NUM_PROCS - 1; i++)
        {
            cout << pivot[i] << " ";
        }
        cout << "\n";
    }
    MPI_Barrier(MPI_COMM_WORLD);
    */
    //广播主元
    MPI_Bcast(pivot, NUM_PROCS - 1, MPI_INT, 0, MPI_COMM_WORLD);//广播主
元
    MPI_Barrier(MPI_COMM_WORLD);

    //debug
    /*
    if (id == 1)
    {
        cout << "id_1_new" << "\n";
        for (i = 0; i < NUM_PROCS - 1; i++)
        {
            cout << pivot[i] << " ";
        }
        cout << "\n";
    }
    MPI_Barrier(MPI_COMM_WORLD);
    */
    //主元划分
    j = 0;
    k = 0;
    for (i = 0; i < seg; i++)
    {
        if (a[id*seg + i] <= pivot[j] && j < NUM_PROCS - 1)
        {
            b[id][j][k] = a[id*seg + i];
            c[id][j]++;
            k++;
        }
        else
        {
            if (j < NUM_PROCS - 1)
            {
```

```
                    k = 0;
                    j++;
                    b[id][j][k] = a[id*seg + i];
                    c[id][j]++;
                    k++;
                }
                else
                {
                    b[id][j][k] = a[id*seg + i];
                    c[id][j]++;
                    k++;
                }
            }
        }
        //收集计数
        if (id != 0)
        {
            MPI_Send(&c[id], NUM_PROCS, MPI_INT, 0, 1, MPI_COMM_WORLD);
        }
        else
        {
            for (i = 1; i < ProcNum; i++)
                MPI_Recv(&c[i], NUM_PROCS, MPI_INT, i, 1, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        }
        MPI_Barrier(MPI_COMM_WORLD);
        if (id == 0)
        {
            /*
            cout << "count:\n";
            for (i = 0; i < NUM_PROCS; i++)
            {
                for (j = 0; j < NUM_PROCS; j++)
                    cout << c[i][j] << " ";
                cout << "\n";
            }
            */
            for (i = 1; i < NUM_PROCS; i++)
            {
                MPI_Send(c, NUM_PROCS*NUM_PROCS, MPI_INT, i, 2,
MPI_COMM_WORLD);//广播计数
            }
        }
        else
```

```
    {
        MPI_Recv(c, NUM_PROCS*NUM_PROCS, MPI_INT, 0, 2, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);//各进程接受计数
    }
    MPI_Barrier(MPI_COMM_WORLD);
    //debug
    /*
    if (id == 1)
    {
        cout << "id_1_count:\n";
        for (i = 0; i < NUM_PROCS; i++)
        {
            for (j = 0; j < NUM_PROCS; j++)
                cout << c[i][j] << " ";
            cout << "\n";
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    */
    //将分组发送到根进程
    if (id != 0)
    {
        MPI_Send(&b[id], NUM_PROCS * 100, MPI_INT, 0, 3, MPI_COMM_WORLD);
    }
    else
    {
        for (i = 1; i < NUM_PROCS; i++)
        {
            MPI_Recv(&b[i], NUM_PROCS * 100, MPI_INT, i, 3,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        }
    }
    MPI_Barrier(MPI_COMM_WORLD);
    //debug
    /*
    if(id==0)
        for (i = 0; i < NUM_PROCS; i++)
        {
            cout << "进程" << i << "\n";
            for (j = 0; j < NUM_PROCS; j++)
            {
                for (k = 0; k < c[i][j]; k++)
                    cout << b[i][j][k] << " ";
                cout << "\n";
```

```
            }
            cout << "\n";
        }
    MPI_Barrier(MPI_COMM_WORLD);
    */
    //将分组广播
    if (id == 0)
    {
        for (i = 1; i < NUM_PROCS; i++)
        {
            MPI_Send(b, NUM_PROCS*NUM_PROCS * 100, MPI_INT, i, 4,
MPI_COMM_WORLD);
        }
    }
    else
    {
        MPI_Recv(b, NUM_PROCS*NUM_PROCS * 100, MPI_INT, 0, 4,
MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }
    MPI_Barrier(MPI_COMM_WORLD);
    //debug
    /*
    if (id == 1)
        for (i = 0; i < NUM_PROCS; i++)
        {
            cout << "进程" << i << "\n";
            for (j = 0; j < NUM_PROCS; j++)
            {
                for (k = 0; k < c[i][j]; k++)
                    cout << b[i][j][k] << " ";
                cout << "\n";
            }
            cout << "\n";
        }
    MPI_Barrier(MPI_COMM_WORLD);
    */
    //计算各段长度
    for (i = 0; i < NUM_PROCS; i++)
        for (j = 0; j < NUM_PROCS; j++)
        {
            Len[i] = c[j][i] + Len[i];
        }
    //新划分间隔
    L = 0;
```

```
    for (i = 0; i < id; i++)
        L = L + Len[i];
    i = 0;
    for (j = 0; j < NUM_PROCS; j++)
    {
        for (k = 0;; k++)
        {
            if (k < c[j][id])
            {
                Final[L + i] = b[j][id][k];
                i++;
            }
            else
                break;
        }
    }
    sort(Final + L, Final + L + Len[id]);//局部排序
    MPI_Barrier(MPI_COMM_WORLD);
    if (id != 0)
    {
        MPI_Send(&Final[L], Len[id], MPI_INT, 0, 5, MPI_COMM_WORLD);
    }
    else
    {
        for (i = 1; i < NUM_PROCS; i++)
        {
            L = L + Len[i - 1];
            MPI_Recv(&Final[L], Len[i], MPI_INT, i, 5, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
        }
        cout << "排序结果：";
        for (i = 0; i < len; i++)
            cout << Final[i] << " ";
            cout << "\n";
    }
    MPI_Barrier(MPI_COMM_WORLD);
}
```