实验四

PB17081504 廖洲洲

实验题目

利用MPI实现并行排序算法

实验环境

操作系统：Win 10

IDE：Microsoft Visual Studio 2015

编译器：cl.exe

硬件配置：CPU：Intel Core i7-8550U；CPU核心数:4；内存:8G

算法设计与分析

PSRS算法

Phase I: Initialization

Start up **p** processors, let the root processor ***0*** get data of size **n**.

Phase II: Scatter data, local sort and regular samples collected

Scatter the data values to the **p** processors. Each processors sorts its local data set, roughly of size **n/p**, using *quicksort*. Each processor chooses **p** sample points, in a very regular manner, from its locally sorted data.

Phase III: Gather and merge samples, choose and broadcast p-1 pivots

The root processor ***0*** gathers the **p** sets of **p** sample points. It is important to realize that each set of these **p** sample points is sorted. These **p** sets are sorted using *multimerge*. From these p^2 sorted points, **p−1** pivot values are regularly chosen and are *broadcast* to the other **p−1** processors.

Phase IV: Local data is partitioned

Each of the p processors *partitions* its local sorted data, roughly of size **n/p**, into p classes using the **p−1** pivot values.

Phase V: All i^{th} classes are gathered and merged

Processor ***i*** *gathers* the i^{th} class of data from every other processor. Each of these classes is sorted using *multimerge*.

Phase VI: Root processor collects all the data

The root processor *gathers* all the data and assembles the sorted list of **n** values.

快速排序的平均复杂度为O(nlogn),使用的归并排序花费的时间为O(n),因此单个进程的所需的时间复杂度为O(nlogn)。空间复杂度为O(n)

核心代码

```
    startTime = MPI_Wtime();
    //均匀划分，将待处理数据均分为size份，进程rank处理下标rank*partition~
(rank+1)*partiton-1段数组
    partition = N / size;
    int start = rank*partition;
    //局部排序，使用快速排序对本地数据进行排序
    quickSort(num + rank*partition, partition);
    //选取样本，从排序好的子序列中选取size个样本,并发送给0号进程，0号进程使用Gather收集数据
    double *cbuffer = (double *)malloc(sizeof(double)*size);
    int space = partition / size;
    for (i = 0; i < size; i++) {
        cbuffer[i] = (num + rank*partition)[i*space];
    }
    double *pivotbuffer = (double *)malloc(sizeof(double)*size*size);
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Gather(cbuffer, size, MPI_DOUBLE, pivotbuffer, size, MPI_DOUBLE, 0,
MPI_COMM_WORLD);
    //样本排序，对选取的size*size个样本进行多次归并排序
    if (rank == 0) {
        for (i = 1; i < size; i++) {
            merge(pivotbuffer, 0, i*size-1, (i + 1)*size-1);
        }
        //选择size-1个主元
        for (i = 1; i < size; i++) {
            cbuffer[i] = pivotbuffer[i*size - 1];
        }
    }
    //使用广播将主元播送给其他进程
    MPI_Bcast(cbuffer, size, MPI_DOUBLE, 0, MPI_COMM_WORLD);
    //主元划分，根据主元将本地数据划分为size段
    int *classStart = (int *)malloc(sizeof(int)*size);
    int *classLength = (int *)malloc(sizeof(int)*size);
    for (i = 0; i < size; i++) {
        classStart[i] = 0;
        classLength[i] = 0;
    }
    int index = 0;
    j = 1;
    for (i = 0; i < partition; i ++ ) {
        if (j<size&& num[start + i] > cbuffer[j]) {
            classStart[j] = i;
            j++;
            index++;
        }
        classLength[index]++;
    }
    //接收各段长度
    int *recvLength = (int *)malloc(sizeof(int)*size);
    int *recvStart = (int *)malloc(sizeof(int)*size);
    for (i = 0; i < size; i++) {
        MPI_Gather(classLength + i, 1, MPI_INT, recvLength, 1, MPI_INT, i,
MPI_COMM_WORLD);
    }
    recvStart[0] = 0;
```

```
    //计算接收的各段的起始地址
    for (i = 1; i < size; i++) {
        recvStart[i] = recvStart[i - 1] + recvLength[i-1];
    }
    //交换，各处理器将其有序段按段号交换到对应的处理器中
    double *recvBuffer = (double *)malloc(sizeof(double)*N);
    for (i = 0; i < size; i++) {
        MPI_Gatherv(num + start + classStart[i], classLength[i], MPI_DOUBLE,
recvBuffer , recvLength, recvStart, MPI_DOUBLE, i, MPI_COMM_WORLD);
    }
    MPI_Barrier(MPI_COMM_WORLD);
    //归并排序，各处理器将收到的元素进行归并排序
    for (i = 1; i < size; i++) {
        merge(recvBuffer, 0, recvStart[i] - 1, recvStart[i]+recvLength[i]-1);
    }
    //收集，0号进程收集各进程处理的数据
    int len = 0;
    for (i = 0; i < size; i++) {
        len += recvLength[i];
    }
    //将各个进程归并处理的数据长度发给0号进程
    MPI_Gather(&len, 1, MPI_INT, recvLength, 1, MPI_INT, 0, MPI_COMM_WORLD);
    recvStart[0] = 0;
    for (i = 1; i < size; i++) {
        recvStart[i] = recvStart[i - 1] + recvLength[i-1];
    }
    MPI_Gatherv(recvBuffer, len, MPI_DOUBLE, num, recvLength, recvStart,
MPI_DOUBLE, 0, MPI_COMM_WORLD);
    endTime = MPI_Wtime();
```

实验结果

运行时间

规模\进程数	1	2	4	8
10000	0.001100	0.001345	0.001975	0.004912
100000	0.013461	0.009364	0.007234	0.010221
1000000	0.215377	0.102547	0.067189	0.063981
10000000	19.275609	5.368064	1.799027	0.974133

加速比

规模\进程数	1	2	4	8
10000	1	0.817844	0.556962	0.223941
100000	1	1.437527	1.860796	1.316994
1000000	1	2.100276	3.20554	3.366265
10000000	1	3.590793	10.71446	19.78745

分析与总结

- PSRS算法使用了多次和多种通信，让我对MPI的各种通信方法有了更加深刻的理解。
- PSRS算法多个进程保留了所有数据并且还设置了接收buffer，故空间的使用会较串行算法更高些
- 当数据规模较小时，并行的加速效果并不明显，甚至进程间的大量通信会导致程序运行的时间更长
- 当数据规模增大，并行的加速效果越来越显著，在开启8个进程的情况下对1千万个数据进行排序，时间花费竟然小于1s，加速比达到了近20，效果惊人。

实验截图

数据规模N=64

- 进程数=1

```
E:\Visual Studio Projects\MPI_PSRS\x64\Debug>mpiexec -n 1 MPI_PSRS.exe
Before sort:
41.00  57.00  1.00  16.00  8.00  32.00  76.00  5.00  55.00  83.00  31.00  25.00  79.00  23.00  89.00  96.00  22.00  69.00  4.00  61.00  70.00  66.00  77.00
74.00  49.00  19.00  10.00  37.00  86.00  6.00  23.00  59.00  80.00  94.00  82.00  1.00  45.00  14.00  29.00  73.00  63.00  14.00  90.00  50.00  22.00  56.00
  41.00  67.00  71.00  73.00  99.00  66.00  60.00  53.00  39.00  56.00  2.00  62.00  86.00  52.00  39.00  72.00  87.00  65.00
After sort:
1.00  1.00  2.00  4.00  5.00  6.00  8.00  10.00  14.00  14.00  16.00  19.00  22.00  22.00  23.00  23.00  25.00  29.00  31.00  32.00  37.00  39.00  39.00  41.
00  41.00  45.00  49.00  50.00  52.00  53.00  55.00  56.00  56.00  57.00  59.00  60.00  61.00  62.00  63.00  65.00  66.00  66.00  67.00  69.00  70.00  71.00
 72.00  73.00  73.00  74.00  76.00  77.00  79.00  80.00  82.00  83.00  86.00  86.00  87.00  89.00  90.00  94.00  96.00  99.00

Time:0.000026
```

- 进程数=2

```
E:\Visual Studio Projects\MPI_PSRS\x64\Debug>mpiexec -n 2 MPI_PSRS.exe
Before sort:
41.00  11.00  54.00  5.00  32.00  7.00  94.00  99.00  14.00  52.00  21.00  28.00  33.00  84.00  46.00  13.00  4.00  62.00  33.00  12.00  71.00  53.00  31.00
 67.00  74.00  95.00  6.00  93.00  88.00  36.00  34.00  43.00  58.00  99.00  26.00  8.00  90.00  75.00  51.00  7.00  5.00  3.00  8.00  24.00  45.00  38.00  4
7.00  25.00  22.00  57.00  57.00  42.00  90.00  4.00  34.00  41.00  49.00  42.00  55.00  54.00  51.00  93.00  54.00  17.00
After sort:
3.00  4.00  4.00  5.00  5.00  6.00  7.00  7.00  8.00  8.00  11.00  12.00  13.00  14.00  17.00  21.00  22.00  24.00  25.00  26.00  28.00  31.00  32.00  33.00
 33.00  34.00  34.00  36.00  38.00  41.00  41.00  42.00  42.00  43.00  45.00  46.00  47.00  49.00  51.00  51.00  52.00  53.00  54.00  54.00  54.00  55.00  57
.00  57.00  58.00  62.00  67.00  71.00  74.00  75.00  84.00  88.00  90.00  90.00  93.00  93.00  94.00  95.00  99.00  99.00

Time:0.000390
```

- 进程数=4

```
E:\Visual Studio Projects\MPI_PSRS\x64\Debug>mpiexec -n 4 MPI_PSRS.exe
Before sort:
41.00  74.00  91.00  93.00  83.00  3.00  70.00  28.00  17.00  2.00  17.00  29.00  74.00  88.00  88.00  36.00  59.00  12.00  77.00  86.00  50.00  31.00  98.00
  38.00  30.00  65.00  26.00  96.00  32.00  34.00  7.00  59.00  98.00  27.00  59.00  75.00  40.00  9.00  62.00  27.00  84.00  16.00  84.00  28.00  5.00  50.0
0  18.00  54.00  27.00  8.00  95.00  99.00  88.00  74.00  28.00  70.00  61.00  5.00  70.00  3.00  12.00  22.00  38.00  35.00
After sort:
2.00  3.00  3.00  5.00  5.00  7.00  8.00  9.00  12.00  12.00  16.00  17.00  17.00  18.00  22.00  26.00  27.00  27.00  27.00  28.00  28.00  28.00  29.00  30.0
0  31.00  32.00  34.00  35.00  36.00  38.00  38.00  40.00  41.00  50.00  50.00  54.00  59.00  59.00  59.00  61.00  62.00  65.00  70.00  70.00  70.00  74.00
74.00  74.00  75.00  77.00  83.00  84.00  84.00  86.00  88.00  88.00  88.00  91.00  93.00  95.00  96.00  98.00  98.00  99.00

Time:0.001585
```

- 进程数=8

```
E:\Visual Studio Projects\MPI_PSRS\x64\Debug>mpiexec -n 8 MPI_PSRS.exe
Before sort:
41.00  97.00  26.00  88.00  81.00  31.00  51.00  58.00  75.00  45.00  9.00  31.00  57.00  48.00  75.00  41.00  14.00  81.00  84.00  50.00  12.00  29.00  68.0
0  45.00  77.00  6.00  61.00  51.00  11.00  16.00  81.00  84.00  6.00  18.00  87.00  98.00  59.00  2.00  28.00  51.00  36.00  98.00  63.00  49.00  98.00  27.
00  88.00  97.00  26.00  17.00  34.00  97.00  70.00  91.00  76.00  40.00  52.00  53.00  31.00  31.00  44.00  78.00  64.00  78.00
After sort:
2.00  6.00  6.00  9.00  11.00  12.00  14.00  16.00  17.00  18.00  26.00  26.00  27.00  28.00  29.00  31.00  31.00  31.00  31.00  51.00  34.00  36.00  40.00
41.00  41.00  44.00  45.00  51.00  61.00  45.00  48.00  49.00  50.00  51.00  52.00  59.00  70.00  77.00  53.00  57.00  58.00  63.00  68.00  76.00  81.00  87.
00  64.00  75.00  75.00  78.00  78.00  81.00  81.00  84.00  88.00  91.00  98.00  84.00  88.00  97.00  97.00  97.00  98.00  98.00

Time:0.003139
```