

奇偶排序实验报告

容逸朗 2020010869

源代码

- `odd_even_sort.cpp` 源代码

```
1  #include <algorithm>
2  #include <cassert>
3  #include <cstdio>
4  #include <cstdlib>
5  #include <mpi.h>
6
7  #include "worker.h"
8
9  // 合并两个有序序列 src1, src2, 取最大的 len1 个元素写回.
10 void get_large(float* src1, int len1, float* src2, int len2, float*
    dst) {
11     int pt1 = len1 - 1, pt2 = len2 - 1;
12     for (int i = len1 - 1; i >= 0; i--) {
13         if (pt2 < 0 || (pt1 >= 0 && src1[pt1] > src2[pt2])) {
14             dst[i] = src1[pt1--];
15         } else {
16             dst[i] = src2[pt2--];
17         }
18     }
19     // 把数据从缓冲区写回 data(src1) 中
20     memcpy(src1, dst, len1 * sizeof(float));
21 }
22
23 // 合并两个有序序列 src1, src2, 取最小的 len1 个元素写回.
24 void get_small(float* src1, int len1, float* src2, int len2, float*
    dst) {
25     int pt1 = 0, pt2 = 0;
26     for (int i = 0; i < len1; i++) {
27         if (pt2 >= len2 || (pt1 < len1 && src1[pt1] < src2[pt2])) {
28             dst[i] = src1[pt1++];
29         } else {
30             dst[i] = src2[pt2++];
31         }
32     }
33     memcpy(src1, dst, len1 * sizeof(float));
```

```

34 }
35
36 void Worker::sort() {
37     // 无元素的进程不参与排序
38     if (out_of_range) return;
39
40     // 对当前进程的私有元素排序
41     std::sort(data, data + block_len);
42
43     // 只有一个进程的情况下，排序结束
44     if (nprocs == 1) return;
45
46     // 计算相邻进程的数据量
47     size_t block_size = ceiling(n, nprocs);
48     int block_len_left = (int)block_size;
49     int block_len_right = (int)std::min(block_size, n - block_size *
(rank + 1));
50
51     // 接收数据和排序数据缓冲区
52     float *buffer = new float[std::max(block_len_left,
block_len_right)];
53     float *tmp = new float[block_len];
54
55     // 最坏情况下，只需 nprocs 轮即可完成排序
56     for (int i = 0; i < nprocs; i++) {
57
58         // 第一阶段：偶奇排序
59         MPI_Request req[2];
60
61         if (rank % 2) {
62             // 奇数进程
63             // 首先发送 1 个数据作比较
64             MPI_Irecv(buffer, 1, MPI_FLOAT, rank - 1, 0, MPI_COMM_WORLD,
&req[0]);
65             MPI_Isend(data, 1, MPI_FLOAT, rank - 1, 1, MPI_COMM_WORLD,
&req[1]);
66             MPI_Waitall(2, req, nullptr);
67
68             // 若数据不是有序的，则交换余下数据
69             if (data[0] < buffer[0]) {
70                 MPI_Irecv(buffer, block_len_left, MPI_FLOAT, rank - 1, 0,
MPI_COMM_WORLD, &req[0]);
71                 MPI_Isend(data, block_len, MPI_FLOAT, rank - 1, 1,
MPI_COMM_WORLD, &req[1]);
72                 MPI_Waitall(2, req, nullptr);

```

```

73         // 奇数进程保留大的
74         get_large(data, block_len, buffer, block_len_left, tmp);
75     }
76
77     } else if (rank + 1 < nprocs) {
78         // 偶数进程
79         MPI_Irecv(buffer, 1, MPI_FLOAT, rank + 1, 1, MPI_COMM_WORLD,
80 &req[0]);
81         MPI_Isend(data + block_len - 1, 1, MPI_FLOAT, rank + 1, 0,
82 MPI_COMM_WORLD, &req[1]);
83         MPI_Waitall(2, req, nullptr);
84
85         // 若数据不是有序的, 则交换余下数据
86         if (data[block_len - 1] > buffer[0]) {
87             MPI_Irecv(buffer, block_len_right, MPI_FLOAT, rank + 1, 1,
88 MPI_COMM_WORLD, &req[0]);
89             MPI_Isend(data, block_len, MPI_FLOAT, rank + 1, 0,
90 MPI_COMM_WORLD, &req[1]);
91             MPI_Waitall(2, req, nullptr);
92
93             // 偶数进程保留小的
94             get_small(data, block_len, buffer, block_len_right, tmp);
95         }
96     }
97
98     // 第二阶段: 奇偶排序
99     if (rank % 2) {
100         if (!last_rank) {
101             // 奇数进程, 同样先发送一个数据作比较
102             MPI_Irecv(buffer, 1, MPI_FLOAT, rank + 1, 1, MPI_COMM_WORLD,
103 &req[0]);
104             MPI_Isend(data + block_len - 1, 1, MPI_FLOAT, rank + 1, 0,
105 MPI_COMM_WORLD, &req[1]);
106             MPI_Waitall(2, req, nullptr);
107
108             // 若数据不是有序的, 则交换余下数据
109             if (data[block_len - 1] > buffer[0]) {
110                 MPI_Irecv(buffer + 1, block_len_right - 1, MPI_FLOAT, rank
111 + 1, 1, MPI_COMM_WORLD, &req[0]);
112                 MPI_Isend(data, block_len - 1, MPI_FLOAT, rank + 1, 0,
113 MPI_COMM_WORLD, &req[1]);
114                 MPI_Waitall(2, req, nullptr);
115
116                 // 奇数进程保留小的
117                 get_small(data, block_len, buffer, block_len_right, tmp);
118             }
119         }
120     }

```

```

110     }
111     } else if (rank > 0) {
112         // 偶数进程
113         MPI_Irecv(buffer + block_len_left - 1, 1, MPI_FLOAT, rank - 1,
114 0, MPI_COMM_WORLD, &req[0]);
115         MPI_Isend(data, 1, MPI_FLOAT, rank - 1, 1, MPI_COMM_WORLD,
116 &req[1]);
117         MPI_Waitall(2, req, nullptr);
118
119         // 若数据不是有序的, 则交换余下数据
120         if (data[0] < buffer[block_len_left - 1]) {
121             MPI_Irecv(buffer, block_len_left - 1, MPI_FLOAT, rank - 1, 0,
122 MPI_COMM_WORLD, &req[0]);
123             MPI_Isend(data + 1, block_len - 1, MPI_FLOAT, rank - 1, 1,
124 MPI_COMM_WORLD, &req[1]);
125             MPI_Waitall(2, req, nullptr);
126
127             // 偶数进程保留大的
128             get_large(data, block_len, buffer, block_len_left, tmp);
129         }
130     }
131 }

```

- 整体而言无额外优化。

性能

- 在元素数量 $n = 100000000$ 的情况下不同进程数运行 `sort` 函数的时间如下:

机器数 N	进程数 P	运行时间/ms	加速比
1	1	12496.814	1.000
1	2	6625.917	1.886
1	4	3534.059	3.536
1	8	1982.254	6.304
1	16	1220.961	10.235
2	16	836.900	14.932