Report

`sort` 函数

```
void Worker::sort() {
    // you can use variables in class Worker: n, nprocs, rank, block_len, data

    size_t max_size = ceiling(n, nprocs); // max size of each block
    int T = 0;
    float* tmp = new float[max_size]; // data from neighbour
    float* new_data = new float[block_len]; // merge data and tmp

    std::sort(data, data + block_len); // sort in each block

    MPI_Request request[2];
    MPI_Status status;
    int neigh_len;
    while (T < nprocs)
    {
        int neigh = ((T % 2) == (rank % 2)) ? (rank + 1) : (rank - 1);
        if (neigh < 0 || neigh >= nprocs)
        {
            T++;
            continue;
        }
        int ps = ((T % 2) == (rank % 2)) ? 1 : -1;

        MPI_Irecv(tmp, max_size, MPI_FLOAT, neigh, T, MPI_COMM_WORLD, &request[
        MPI_Isend(data, block_len, MPI_FLOAT, neigh, T, MPI_COMM_WORLD, &reques
        MPI_Wait(&request[1], &status);
        MPI_Get_count(&status, MPI_FLOAT, &neigh_len); // get length of neighbo

        if (ps == 1) // merge two sorted sequence
        {
            int pt1 = 0, pt2 = 0;
            int now = 0;
            while (now < (int)block_len)
            {
                if (pt2 == neigh_len || data[pt1] < tmp[pt2])
                {
                    new_data[now] = data[pt1];
                    pt1++;
                }
```

```
                else
                {
                    new_data[now] = tmp[pt2];
                    pt2++;
                }
                now++;
            }
        }
        else
        {
            int pt1 = block_len - 1, pt2 = neigh_len - 1;
            int now = block_len - 1;
            while (now >= 0)
            {
                if (pt2 < 0 || data[pt1] > tmp[pt2])
                {
                    new_data[now] = data[pt1];
                    pt1--;
                }
                else
                {
                    new_data[now] = tmp[pt2];
                    pt2--;
                }
                now--;
            }
        }
        T++;
        MPI_Wait(&request[0], nullptr);
        std::swap(data, new_data);
    }
    delete[] tmp;
    delete[] new_data;
}
```

性能优化

1. 该实现将计算过程于发送数据时间进行重叠。
2. 使用手工绑核，由于使用 `lscpu` 查看NUMA拓扑后可看到

```
NUMA:
NUMA node(s):          2
NUMA node0 CPU(s):     0,2,4,6,8,10,12,14,16,18,20,22,24,26
NUMA node1 CPU(s):     1,3,5,7,9,11,13,15,17,19,21,23,25,27
```

由于是相邻进程之间通信，所以将前 `$NCPUS / 2` 个进程依次绑定到 `0,2,4,...` 。将后 `$NCPUS / 2` 个进程绑定到 `1,3,5,...` 的core上。
不手动绑核时使用2机56进程的运行时间为 `821.283000 ms` ，绑核后为 `664.187000 ms` ，有较为明显的提升。

测试结果

$N \times P$	运行时间	加速比
1×1	`12283.614000 ms`	$S_1 = 1$
1×2	`6469.283000 ms`	$S_2 = 1.899$
1×4	`3456.476000 ms`	$S_4 = 3.554$
1×8	`1929.296000 ms`	$S_8 = 6.367$
1×16	`1215.141000 ms`	$S_{16} = 10.109$
2×16	`818.412000 ms`	$S_{32} = 15.009$