MPI实现even_odd_sort

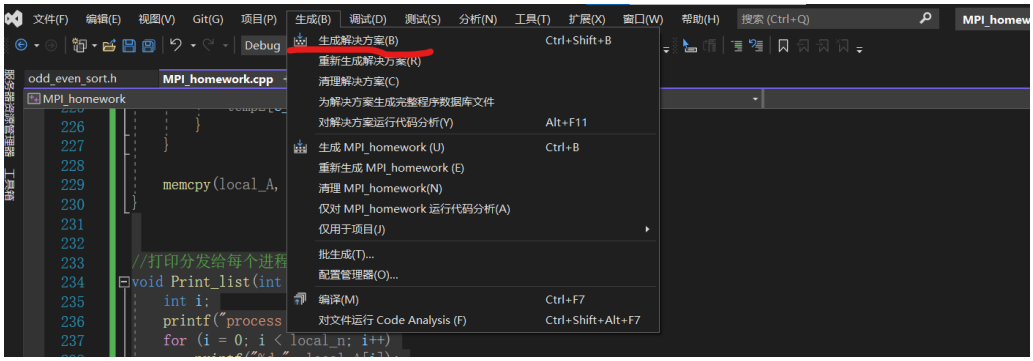
使用MSMPI实现并行化的奇偶排序，可以根据命令行参数的不同选择是生成随机数还是用户输入数字进行排序。

环境

- Visual Studio Community 2022　17.1.3
- Microsoft MPI v10.1.2

用法

- 编译
  - 在配制好MSMPI的VS中点击“生成解决方案”进行编译：

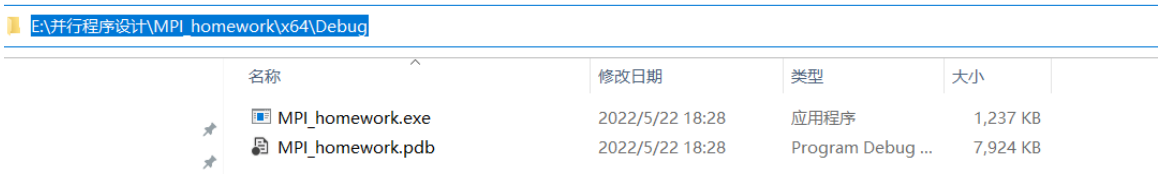


- 运行

  在含有MPI_homework.exe的目录下打开终端输入以下格式的命令：

```
mpiexec -n <进程个数> <可执行文件文件名> <g/i> <数组元素个数>

其中g代表随机生成数组元素并排序输出
i代表由用户在终端输入
```



运行示例

1. **4**个进程，生成**16**个随机数：

```
E:\并行程序设计\MPI_homework\x64\Debug>mpiexec -n 4 MPI_homework g 16

排序前的数组元组:
process 0's data:  52 71 99 40
process 1's data:  63 96 63 73
process 2's data:  73 69 26 54
process 3's data:  84 42 90 87

********************************************************************************
排序后的数组元素:
26 40 42 52 54 63 63 69 71 73 73 84 87 90 96 99
```

2. **4**个进程，用户在命令行输入**16**个数字：

```
E:\并行程序设计\MPI_homework\x64\Debug>mpiexec -n 4 MPI_homework i 16
33 456 32  546 674 23  53 1 234 6 77 23 53 77 55 32

********************************************************************************
排序后的数组元素:
1 6 23 23 32 32 33 53 53 55 77 77 234 456 546 674

E:\并行程序设计\MPI_homework\x64\Debug>
```

代码说明

1. merge_low():该函数用来合并两个进程的数据，并取较小的一半数据:

```
void Merge_low(int  my_keys[],int recv_keys[],int
temp_keys[],int local_n) {
    int m_i, r_i, t_i;
    m_i = r_i = t_i = 0;
    while (t_i < local_n) {
        if (my_keys[m_i] <= recv_keys[r_i]) {
            temp_keys[t_i++] = my_keys[m_i++];
        }
        else {
            temp_keys[t_i++] = recv_keys[r_i++];
        }
    }

    memcpy(my_keys, temp_keys, local_n * sizeof(int));
```

2.merge_high():该函数用来合并两个进程的数据，并取较大的一半数据

```
void Merge_high(int local_A[], int temp1[], int temp2[],int
local_n) {
```

```
    int a_i, b_i, c_i;
    a_i = local_n - 1;
    b_i = local_n - 1;
    c_i = local_n - 1;
    while (c_i >= 0) {
        if (local_A[a_i] >= temp1[b_i]) {
            temp2[c_i--] = local_A[a_i--];
        }
        else {
            temp2[c_i--] = temp1[b_i--];
        }
    }

    memcpy(local_A, temp2, local_n * sizeof(int));
}
```

3.Sort():排序函数，使用内置的快速排序函数对local list进行排序，再用奇偶排序对global list进行排序：

```
//排序函数，对local list进行排序，使用奇偶排序对global list进行排序
void Sort(int local_A[], int local_n, int my_rank,int p,
MPI_Comm comm) {
    int phase;
    int* temp1, * temp2;
    int even_partner;
    int odd_partner;
    temp1 = (int*)malloc(local_n * sizeof(int));
    temp2 = (int*)malloc(local_n * sizeof(int));
    //获取某个阶段，某个进程的通信伙伴
    if (my_rank % 2 != 0) {
        //奇通信阶段，奇数为通信双方的较小进程
        even_partner = my_rank - 1;
        odd_partner = my_rank + 1;
        if (odd_partner == p)
            odd_partner = MPI_PROC_NULL;
    }
    else {
        //偶通信阶段，偶数为通信双方的较小进程
        even_partner = my_rank + 1;
        if (even_partner == p)
            even_partner = MPI_PROC_NULL;
        odd_partner = my_rank - 1;
    }
```

```
    //用内置的qsort函数对local list进行快速排序
    qsort(local_A, local_n, sizeof(int), cmp);
    //定理:如果p个进程运行奇偶排序算法，那么p个阶段后，输入列表就有序
    for (phase = 0; phase < p; phase++)
        Odd_even_iteration(local_A, temp1, temp2, local_n,
phase,even_partner, odd_partner, my_rank, p, comm);
    //释放内存
    free(temp1);
    free(temp2);
}
```

3.Odd_even_iteration():一次奇偶转换排序的迭代函数

```
void Odd_even_iteration(int local_A[], int temp1[], int
temp2[],int local_n, int phase, int even_partner, int
odd_partner,int my_rank, int p, MPI_Comm comm) {
    MPI_Status status;
    if (phase % 2 == 0) {
        //even phase
        if (even_partner >= 0) {
            //为了保证MPI程序的安全性，采用MPI自己提供的调度通信的办
法MPI_Sendrecv
            //与对方进程交换数据
            MPI_Sendrecv(local_A, local_n, MPI_INT,
even_partner, 0,temp1, local_n, MPI_INT, even_partner, 0,
comm,&status);
            if (my_rank % 2 != 0)
                Merge_high(local_A, temp1, temp2, local_n);
            else
                Merge_low(local_A, temp1, temp2, local_n);
        }
    }
    else {
        //odd phase
        if (odd_partner >= 0) {
            MPI_Sendrecv(local_A, local_n, MPI_INT,
odd_partner, 0,temp1, local_n, MPI_INT, odd_partner, 0,
comm, &status);
            if (my_rank % 2 != 0)
                Merge_low(local_A, temp1, temp2, local_n);
            else
                Merge_high(local_A, temp1, temp2, local_n);
```

```
            }
        }
    }
```