CSC4005

Distributed and Parallel Computing

Assignment #1

Hajun Lee

117010437

Introduction

In assignment1, we should build a parallel odd-even sorting program by using MPI. we need to use multiprocessor to sort a list of numbers as fast as possible and return sort list to user.

Method

Odd-Even Sort Algorithm:

The process of Odd-Even Sort algorithm can largely divide into 5 steps.

1. If the index of the number is odd
    - It should compare with the number preceding it in the even index.
2. If the former is greater than the latter
    - Reverse the order.
3. Select the number at the even index and select the number after that respectively.
4. If it is not in ascending order
    - Flip the two numbers again.
5. Do these steps until all the numbers are in the correct index.

Parallel Odd-Even Sort Algorithm:

1. At first, distribute m numbers to each of the n processes.
2. Process should follow the odd-even transposition sort.
3. In each process should compare numbers at odd index with latter number at even index and if before one is larger than the latter one, it should flip it.
4. If some numbers (even numbers) remain for compare at first, then compare with the boundary process. If latter one is smaller, flip it.
5. If number is even, should send to latter process and then compare with the boundary process.
6. Repeat these processes until nothing can be flip anymore.

Design

1. Sequential.cpp

Before main function:

1. Include Header files and defined namespace.
2. Modified constant variables.

Main function:

1. Print 'name', 'student ID' and 'AS1, odd-even transposition sort, sequential ver'
2. Init array data for sort random generated number using rand() first.
3. Before execute, start the timer
4. Execute
    A. Odd sort
    B. Even sort
    C. Stop when nothing can be change
5. Timer stop
6. Print time

Odd-even-sort.cpp

Before main function:

1. Include Header files and defined namespace.

Main function:

1. Modified constant variables.
2. Initialize MPI
3. Calculate temporary array size for size of thread which can be integer
    A. MAX_VALUE
4. (MASTER)Init array to sort
    A. Random number
5. Print sort array
6. (MASTER and SLAVE) Time start
7. Call doSortParallel()
    A. Compare odd, comp_odd()
    B. Compare even, comp_even()
    C. Compare boundary, comp_bound()
    D. Stop when nothing to flip(swap)
8. MPI_GATHER all the result to array
9. Time end
10. (MASTER) print total time
11. Print sorted array
12. MPI finish

Performance Analysis:

The MPI version doesn't have better performance than the sequential version when array has relatively small size. Even if, there has Although there are situations where parallel programs take much time to complete the sort, this could be because of communication between different processes.

MPI version's performance is good when array size is large relatively and MPI has the more array size increases, the clearer about the improvement.

Sequential.cpp Code:

Sequential part 1:

```
시작하기    odd-even-sort.cpp 3 ●    sequential.cpp ●
C: > Users > Andy > Desktop > cross-vm > sequential.cpp > ...
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <time.h>



using namespace std;

const int ARR_SIZE = 70000;                    //const array

int main(int argc, char* argv[]){



    cout<< "NAME: Hajun Lee\n";
    cout<< "STUDENT ID: 117010437\n";
    cout<< "Assignment 1, Odd-Even Transposition Sort, Sequential Version.\n";



    int arr[ARR_SIZE];
    time_t start_t, end_t;
    srand(10000);
                                               //random
    for (int i = 0; i < ARR_SIZE; i++){
        arr[i]=rand();
    }

    start_t = time(NULL);                      //start timer
```

Sequential part2:

이동(G) 실행(R) 터미널(T) 도움말(H) • sequential.cpp - Visual Studio Code

시작하기 | odd-even-sort.cpp 3 • | sequential.cpp •

C: > Users > Andy > Desktop > cross-vm > sequential.cpp > main(int, char * [])

```cpp
    for (int i = 0; i < ARR_SIZE; i++){
        arr[i]=rand();
    }

    start_t = time(NULL);                       //start timer


                                                //sort
    for (int i = 0; i < ARR_SIZE; i++){
        for (int j = ARR_SIZE - 1; j > i; j--){
            if (arr[j] <= arr[j - 1]){
                int k = arr[j - 1];
                arr[j - 1] = arr[j];
                arr[j] = k;
            }
        }
    }

    end_t = time(NULL);                         //end timer


    int total_t = end_t - start_t;              //find total time



    cout<< "Total time is : ";
    cout<<  total_t;
    cout<< "" << endl;
}
```

Odd-even-sort.cpp Code:

C: > Users > Andy > Desktop > cross-vm > csc4005-assignment-1 > src > odd-even-sort.cpp > comp_Even(int *, int)

```cpp
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <iostream>
#include <mpi.h>
#include <stdint.h>
#include <algorithm>
#include <vector>


using namespace std;


int comp_Odd(int *arr, int size){

    int sort_Odd = 1;                                        // odd sort

    for (int j = 0; j < size - 1; j += 2){
        if (arr[j] > arr[j + 1]) {
            int flip = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = flip;
            sort_Odd = 0;
        }
    }

    return sort_Odd;
}

int comp_Even(int *arr, int size){

    int sort_Even = 1;                                       // even sort

    for (int j = 1; j < size - 1; j += 2){
        if (arr[j] > arr[j + 1]) {
            int flip = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = flip;
            sort_Even = 0;
        }
    }

    return sort_Even;
}

```

```c
int comp_Bound(int rank, int *arr, int size, int comp_Size){
    int num;
    int code = 1;

    if (rank == 0){
        MPI_Recv(&num, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        if (num < arr[size - 1]) {
            swap(num, arr[size - 1]);
            code = 0;
        }

        MPI_Send(&num, 1, MPI_INT, rank + 1, 1, MPI_COMM_WORLD);
    }

    else if (rank == comp_Size - 1){
        MPI_Send(&arr[0], 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD);
        MPI_Recv(&arr[0], 1, MPI_INT, rank - 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }

    else{

        MPI_Send(&arr[0], 1, MPI_INT, rank - 1, 0, MPI_COMM_WORLD);
        MPI_Recv(&num, 1, MPI_INT, rank + 1, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        if (num < arr[size - 1]){
            swap(num, arr[size - 1]);
            code = 0;
        }
        MPI_Send(&num, 1, MPI_INT, rank + 1, 1, MPI_COMM_WORLD);
        MPI_Recv(&arr[0], 1, MPI_INT, rank - 1, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    }

    return code;
}

```

```c
void doSortParallel(int *arr, int size){

    int sort_Odd = 0;
    int sort_Even = 0;
    int rank = -1;
    int comp_Size = 0;
    int comp_bound_info = 0;



    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &comp_Size);


    while (true){

        sort_Odd = comp_Odd(arr, size);

        if (size % 2 != 0 && comp_Size != 1){
            comp_bound_info = comp_Bound(rank, arr, size, comp_Size);
        }

        sort_Even = comp_Even(arr, size);

        if (size % 2 == 0 && comp_Size != 1){
            comp_bound_info = comp_Bound(rank, arr, size, comp_Size);
        }

        int info = sort_Odd + sort_Even + comp_bound_info;
        int result;

        MPI_Allreduce((void *) & info, (void *) & result, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);

        if (result == 3){
            break;
        }
    }

    return;
}
```

```
int main(int argc, char *argv[]){


    int comp_Size, comp_Rank, partner;
    double T_process, T_sort;


    // def const
    const int ARRAY_SIZE = 20;
    const int MAX_VALUE = 100;
    int temp_size = ARRAY_SIZE;



    // init mpi
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &comp_Size);
    MPI_Comm_rank(MPI_COMM_WORLD, &comp_Rank);

    if (ARRAY_SIZE % comp_Size != 0){
        temp_size += (comp_Size - (ARRAY_SIZE % comp_Size));
    }

    const int ARRAY_TEMP_SIZE = const_cast<const int&>(temp_size);

    int size = ARRAY_TEMP_SIZE / comp_Size;
    int *sourceArray;
```

```
    // init array
    if (comp_Rank == 0){


        srand(time(NULL));
        sourceArray = new int[ARRAY_TEMP_SIZE];

        for (int i = 0; i < ARRAY_SIZE; i += 1){
            sourceArray[i] = rand() % MAX_VALUE;
        }

        if (ARRAY_TEMP_SIZE != ARRAY_SIZE){
            for (int i = ARRAY_SIZE; i < ARRAY_TEMP_SIZE; i += 1){
                sourceArray[i] = MAX_VALUE;
            }
        }

        for (int i = 0; i < ARRAY_SIZE; i++){
            cout << sourceArray[i] << " ";
        }

        cout << endl;

    }


    //start
    T_process = MPI_Wtime();

    int *data = new int[size];

    MPI_Scatter(sourceArray, size, MPI_INT, data, size, MPI_INT, 0, MPI_COMM_WORLD);
    doSortParallel(data, size);
    MPI_Gather(data, size, MPI_INT, sourceArray, size, MPI_INT, 0, MPI_COMM_WORLD);

    T_process = MPI_Wtime() - T_process;

    MPI_Reduce((void *) & T_process, (void *)&T_sort, 1, MPI_DOUBLE, MPI_MAX, 0, MPI_COMM_WORLD)

    //end
```

```
    if (comp_Rank == 0){

        cout<< "NAME: Hajun Lee\n";
        cout<< "STUDENT ID: 117010437\n";
        cout<< "Assignment 1, Odd-Even Transposition Sort, MPI Version.\n";

        printf("Total time: %f s\n", T_sort * 1000);

        for (int i = 0; i < ARRAY_SIZE; i++){
            cout << sourceArray[i] << " ";
        }

        cout << "" << endl;

    }

    delete[] data;

    if (comp_Rank == 0){
        delete[] sourceArray;
    }

    MPI_Finalize();

    return 0;

}
```

How to run a code (sequential.cpp):

To run the sequential version,

1. Move sequential.cpp file to virtual machine desktop.
    - cp /mnt/host/sequential.cpp /home/csc4005/Desktop/.
2. Compile it with below.
    - sudo g++ sequential.cpp -o ./output.out
3. Run it with below.
    - ./output.out

How to run a code (odd-even-sort.cpp):

To run the MPI version,

1. Move odd-even-sort.cpp file to server (use sftp)
2. Compile it with below.
    - mpic++ odd-even-sort.cpp -o ./output.out
3. Run it with below.
    - mpirun -n 8 ./output.out

Output (sequential.cpp)

```
[csc4005@localhost Desktop]$ cp /mnt/host/sequential.cpp /home/csc4005/Desktop/
[csc4005@localhost Desktop]$ sudo g++ sequential.cpp -o ./output.out
[sudo] password for csc4005:
Sorry, try again.
[sudo] password for csc4005:
[csc4005@localhost Desktop]$ ./output.out
NAME: Hujun Lee
STUDENT ID: 117010437
Assignment 1, Odd-Even Transposition Sort, Sequential Version.
Total time is : 1
```

```
[csc4005@localhost Desktop]$ cp /mnt/host/sequential.cpp /home/csc4005/Desktop/
[csc4005@localhost Desktop]$ sudo g++ sequential.cpp -o ./output.out
[csc4005@localhost Desktop]$ ./output.out
NAME: Hujun Lee
STUDENT ID: 117010437
Assignment 1, Odd-Even Transposition Sort, Sequential Version.
Total time is : 1
[csc4005@localhost Desktop]$ cp /mnt/host/sequential.cpp /home/csc4005/Desktop/
[csc4005@localhost Desktop]$ sudo g++ sequential.cpp -o ./output.out
[csc4005@localhost Desktop]$ ./output.out
NAME: Hujun Lee
STUDENT ID: 117010437
Assignment 1, Odd-Even Transposition Sort, Sequential Version.
Total time is : 3
[csc4005@localhost Desktop]$ cp /mnt/host/sequential.cpp /home/csc4005/Desktop/
[csc4005@localhost Desktop]$ sudo g++ sequential.cpp -o ./output.out
[csc4005@localhost Desktop]$ ./output.out
NAME: Hujun Lee
STUDENT ID: 117010437
Assignment 1, Odd-Even Transposition Sort, Sequential Version.
Total time is : 6
[csc4005@localhost Desktop]$ cp /mnt/host/sequential.cpp /home/csc4005/Desktop/
[csc4005@localhost Desktop]$ cp /mnt/host/sequential.cpp /home/csc4005/Desktop/
[csc4005@localhost Desktop]$ sudo g++ sequential.cpp -o ./output.out
[csc4005@localhost Desktop]$ ./output.out
NAME: Hujun Lee
STUDENT ID: 117010437
Assignment 1, Odd-Even Transposition Sort, Sequential Version.
Total time is : 9
[csc4005@localhost Desktop]$ cp /mnt/host/sequential.cpp /home/csc4005/Desktop/
[csc4005@localhost Desktop]$ sudo g++ sequential.cpp -o ./output.out
[csc4005@localhost Desktop]$ ./output.out
NAME: Hajun Lee
STUDENT ID: 117010437
Assignment 1, Odd-Even Transposition Sort, Sequential Version.
Total time is : 13
```

```
[csc4005@localhost Desktop]$ cp /mnt/host/sequential.cpp /home/csc4005/Desktop/
[csc4005@localhost Desktop]$ sudo g++ sequential.cpp -o ./output.out
[csc4005@localhost Desktop]$ ./output.out
NAME: Hajun Lee
STUDENT ID: 117010437
Assignment 1, Odd-Even Transposition Sort, Sequential Version.
Total time is : 19
[csc4005@localhost Desktop]$
```

From beginning: 1,2,3,4,5,6,7

Output (odd-even-sort.cpp)

```
bash-4.2$ mpirun -n 8 output.out
92 12 31 55 97 14 44 22 23 19 52 47 90 29 23 74 92 78 63 62
NAME: Hajun Lee
STUDENT ID: 117010437
Assignment 1, Odd-Even Transposition Sort, MPI Version.
Total time: 0.432886 s
12 14 19 22 23 23 29 31 44 47 52 55 62 63 74 78 90 92 92 97
bash-4.2$
```