6550 Parallel Programming HW3

Riley Densley

September 2019

1 Integer Sort

The main array to be sorted is created by process 0 as the Master. For simplicity, the size of the array is a multiple of the remaining processes. The Slave processes probe the Master process to get the size of array they will be receiving. They then receive them and sort them with bubble sort. Then the Slave processes send the arrays back to Master. The Master probes to get the size and sender. It then receives the sorted array and merges it with its own.

2 Code

```
#include <iostream>
#include <mpi.h>
#include <random>
#include <unistd.h>
#define MCW MPI_COMM_WORLD

using namespace std;

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
    int i, j;
    for (i = 0; i < n - 1; i++)

        // Last i elements are already in place
        for (j = 0; j < n - i - 1; j++)
```

```
            if (arr[j] > arr[j + 1])
                swap(&arr[j], &arr[j + 1]);
}

void merge(int temp[], int sizeM, int slave[], int sizeS, int masterSorted[])
{
    int mStep = 0;
    int sStep = 0;

    for (int i = 0; i < sizeM + sizeS; i++)
    {
        if (sStep < sizeS && mStep < sizeM)
        {
            if (temp[mStep] < slave[sStep])
            {
                masterSorted[i] = temp[mStep];
                mStep++;
            }
            else
            {
                masterSorted[i] = slave[sStep];
                sStep++;
            }
        }
        else if (mStep < sizeM)
        {
            masterSorted[i] = temp[mStep];
            mStep++;
        }
        else
        {
            masterSorted[i] = slave[sStep];
            sStep++;
        }
    }
}

void print(int nums[], int size)
{
    for (int i = 0; i < size; i++)
    {
        cout << nums[i] << " ";
    }
    cout << endl;
}
```

```
int main(int argc, char **argv)
{
    int rank, size;
    int data;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);
    MPI_Status mystatus;
    srand(time(NULL));

    //initialize Array of random numbers
    int subSize = rand() % 10 + 5;
    int arraySize = (subSize * (size - 1));

    //Process 0 starts the game
    //Master
    if (rank == 0)
    {
        int toSortMaster[arraySize];
        for (int i = 0; i < arraySize; i++)
        {
            toSortMaster[i] = rand() % 50;
        }
        cout << "To sort Master array \n";
        print(toSortMaster, arraySize);

        for (int i = 1; i < size; i++)
        {
            //initialize array for sending
            int sendingArray[subSize];
            cout << "Sent section to array " << i << endl;
            for (int j = 0; j < subSize; j++)
            {
                sendingArray[j] = toSortMaster[((i - 1) * subSize) + j];
            }
            MPI_Send(sendingArray, subSize, MPI_INT, i, 0, MCW);
        }

        //receive arrays and order them
        int sortedSize = 0;
        int *masterSorted;
        for (int i = 1; i < size; i++)
        {
            MPI_Probe(MPI_ANY_SOURCE, MPI_ANY_TAG, MCW, &mystatus);
            int quantity;
            MPI_Get_count(&mystatus, MPI_INT, &quantity);
            int sortedSlave[quantity];
            MPI_Recv(&sortedSlave, quantity, MPI_INT, mystatus.MPI_SOURCE, 0, MCW, MPI_STATU

            int *temp = masterSorted;
            masterSorted = new int[sortedSize + quantity];
            merge(temp, sortedSize, sortedSlave, quantity, masterSorted);
            sortedSize += quantity;
        }
        cout << "Master sorted array\n";
        print(masterSorted, sortedSize);
    }
    else
    {
        //Slave
        //Sorting Processes
        MPI_Probe(0, MPI_ANY_TAG, MCW, &mystatus);
        int quantity;
        MPI_Get_count(&mystatus, MPI_INT, &quantity);
        int toSortSlave[quantity];
        MPI_Recv(&toSortSlave, quantity, MPI_INT, 0, 0, MCW, MPI_STATUS_IGNORE);
        //Sort this Array
        //sleep(rank);
        cout << rank << " sorting job is  ";
        print(toSortSlave, quantity);
        bubbleSort(toSortSlave, quantity);

        cout << rank << " sorted is  ";
        print(toSortSlave, quantity);

        MPI_Send(&toSortSlave, quantity, MPI_INT, 0, 0, MCW);
    }

    MPI_Finalize();

    return 0;
}
```

3 Output

3.1 Commands

```
mpic++ assign3.cpp
mpirun -np 4 ./a.out
```

3.2 Output

```
To sort Master array
43 26 15 18 12 45 16 41 35 15 41 18 12 29 35 32 34 37 2 37 11 8 24 20 19 11 11
Sent section to array 1
Sent section to array 2
Sent section to array 3
Master sorted array
2 8 11 11 11 12 12 15 15 16 18 18 19 20 24 26 29 32 34 35 35 37 37 41 41 43 45
1 sorting job is  43 26 15 18 12 45 16 41 35
1 sorted is  12 15 16 18 26 35 41 43 45
2 sorting job is  15 41 18 12 29 35 32 34 37
2 sorted is  12 15 18 29 32 34 35 37 41
3 sorting job is  2 37 11 8 24 20 19 11 11
3 sorted is  2 8 11 11 11 19 20 24 37
```