Assignment 4 Report

Cory Christensen

February 7, 2020

1 Implementation

For my implementation I kicked the program off with process zero and sent one random number to each process. There is then two loops, one nested in the other. The first loop goes through sqrt(p) iterations where p is the number of processes. The second loop starts at the current outer iteration and decrements to zero. The second iterator is associated with the first cube parameter. Inside of this set of loops each process sends its value to it's cube compliment and then receives from it's cube compliment. Each process then decides if it will keep the value it has been sent based on four conditions.

1. If the number received is less than the current number and the process number is smaller than it's compliment and is in the ascending list.
2. If the number received is greater than the current number and the process number is larger than it's compliment and is in the ascending list.
3. If the number received is less than the current number and the process number is larger than it's compliment and is in the descending list.
4. If the number received is less than the current number and the process number is larger than it's compliment and is in the descending list.

Each iteration prints all of the numbers from each process until the last iteration where everything has been sorted.

2 Code

```
#include <iostream>
#include <mpi.h>
#include <unistd.h>
#include <stdlib.h>
#include <math.h>

//#include "/usr/local/include/mpi.h"
```

```
#define MCW MPI_COMM_WORLD

using namespace std;

int cube(int f, int process){
    int dest;
    int mask=1;
    mask <<= f;
    dest = process ^ mask;
    return dest;
}

void allPrint(int data){
    int rank, size;

    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);

    const int printSize = size;
    int print[printSize];
    MPI_Gather(&data,1,MPI_INT,print,1,MPI_INT,0,MCW);
    if(!rank) {
        for (int i = 0; i < size; i++) {
            cout << print[i] << " ";
        }
        cout << endl;
    }
    return;
}

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

    int rank,recvRank, size;
    int data, myNum, prevNum, process;
    int mask;
    bool descending, larger;
    MPI_Status mystatus;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);
    MPI_Request numRequest;

    //Send out random numbers
    if(!rank){
        for(int i = 0; i < size; i++){
            data = rand()%100;
```

```
            MPI_Send(&data, 1, MPI_INT, i, 0, MCW);

        }
    }

    //recieve numbers
    MPI_Recv(&data,1,MPI_INT,MPI_ANY_SOURCE,0,MCW,&mystatus);
    myNum = data;
    allPrint(myNum);

    //loop through bitonic sort
    for(int i= 0; i < sqrt(size); i++){
        mask = 1;
        mask <<= i + 1;
        descending = rank & mask;
        for(int j = i; j >= 0; j--){
            mask = 1;
            mask <<= j;
            larger = rank & mask;
            process = cube(j,rank);


            data = myNum;
            MPI_Send(&data,1,MPI_INT,process,0,MCW);
            MPI_Recv(&data,1,MPI_INT,process,0,MCW,&mystatus);

            //swap if any of the four cases are true
            if((data < myNum && !larger && !descending) || (data > myNum && la
                myNum = data;
            }

            MPI_Barrier(MCW);
            allPrint(myNum);
        }
    }
    MPI_Finalize();
    return 0;
}
```

3 Run Commands

```
mpic++ Assn4.cpp
mpirun -np  -oversubscribe a.out
```

4 Output

```
83 86 77 15 93 35 86 92 49 21 62 27 90 59 63 26
83 86 77 15 35 93 92 86 21 49 62 27 59 90 63 26
77 15 83 86 92 93 35 86 21 27 62 49 63 90 59 26
15 77 83 86 93 92 86 35 21 27 49 62 90 63 59 26
15 77 83 35 93 92 86 86 90 63 59 62 21 27 49 26
15 35 83 77 86 86 93 92 90 63 59 62 49 27 21 26
15 35 77 83 86 86 92 93 90 63 62 59 49 27 26 21
15 35 62 59 49 27 26 21 90 63 77 83 86 86 92 93
15 27 26 21 49 35 62 59 86 63 77 83 90 86 92 93
15 21 26 27 49 35 62 59 77 63 86 83 90 86 92 93
15 21 26 27 35 49 59 62 63 77 83 86 86 90 92 93
```