Assignment 4

Stuart Hopkins, A02080107

February 2020

1 Bitonic Sorting

To create a bitonic sorting program I first initialize each process with a ”random” number. As currently written the array is not random but in reverse so that the sorting can be seen. After initialization, The starting array is printed to the console with the related rank. I then begin to sort the numbers.

Using the bitonic sorting algorithm, the numbers are sorted. To tell whether the numbers should be increasing or decreasing we look at the bits of the ”level” we are on. The increasing function can be seen as to how that decision is made using a mask. The numbers are then sorted according to the bitonic sorting algorithm. Lastly, the sorted array is printed to the console with the related rank of processes. The process then terminates. My code is as follows:

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

#define MCW MPI_COMM_WORLD

using namespace std;

void allPrint(int data) {
  int array[64];
  int rank, size;

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

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

```
bool increasing(int level){
  int rank, size;
  int mask = 2;  // Starts at 010 because first round already "done"
  mask <<= level;
  // cout << "\nMask " << mask << endl;

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

  // cout << "And " << (rank & mask) << endl;
  return !(rank & mask);
}


void cube(int f, bool inc, int *data){
  int rank, size;
  int dest;
  int mask=1;
  mask <<= f;

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

  dest = rank ^ mask;

  int myData = *data;

  // Should we send an array saying what processor it came from?
  MPI_Send(data, 1, MPI_INT, dest, 0, MCW);
  MPI_Recv(data, 1, MPI_INT, MPI_ANY_SOURCE, 0, MCW,MPI_STATUS_IGNORE);

  if(inc) {
    if(*data < myData && rank > dest)  // Change it back because it overrode it incorrectly
      *data = myData;
    else if(*data >= myData && rank < dest)  // Change it back because it overrode it incorrectly
      *data = myData;
  }
  else {
    if(*data > myData && rank > dest)  // Change it back because it overrode it incorrectly
      *data = myData;
    else if(*data <= myData && rank < dest)  // Change it back because it overrode it incorrectly
      *data = myData;
  }

  return;
}
```

```
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);

  data = (size - rank);

  allPrint(data);

  MPI_Barrier(MCW);
  if(!rank) cout << endl;
  for(int i = 0; i <=(size >> 2); i++) {
    bool inc = increasing(i);

    for(int j = i; j >= 0; j--) {
      MPI_Barrier(MCW);
      cube(j, inc, &data);
    }
  }
  allPrint(data);

  MPI_Finalize();
  return 0;
}
```

To verify it works here is the command line arguments and output It worked, here is the code:

```
$ mpic++ bitonicSort.cpp
$ mpirun -np 8 -oversubscribe a.out

0 1 2 3 4 5 6 7
8 7 6 5 4 3 2 1

0 1 2 3 4 5 6 7
1 2 3 4 5 6 7 8
```

You can see that the numbers are sorted correctly. This algorithm was checked with 2, 4, and 8 processors. It is a requirement to run this program with a power of two processors and an equal amount of data.