CS 5500
Spring 2021
Homework 4
By Griffin Hackley
A02224681

There is a commented out line that reads ”uncomment the below line to show each cube swap operation”. This line can be uncommented in order to show the results after each individual cube swap operation

The command to compile the code is:
mpic++ hw4.cpp

The command to run the code is:
mpirun -np (num) -oversubscribe a.out

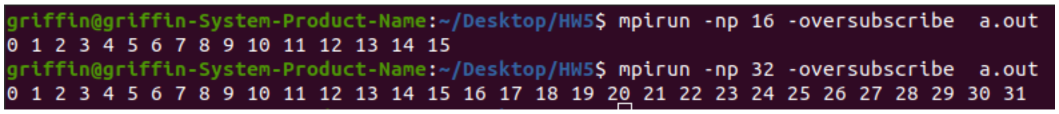
where (num) is the number of processes (which must be a power of 2)

The program begins by creating an array that is the size of the number of processors. The array is then shuffled (Note: this is not actually a random shuffle. It will be the same each time)

This array is then passed into the bitonic sort function where each process grabs its respective value from the array. Next the program will reverse the second half of each individual list in order to make it a bitonic list and the data is sent back to each process.

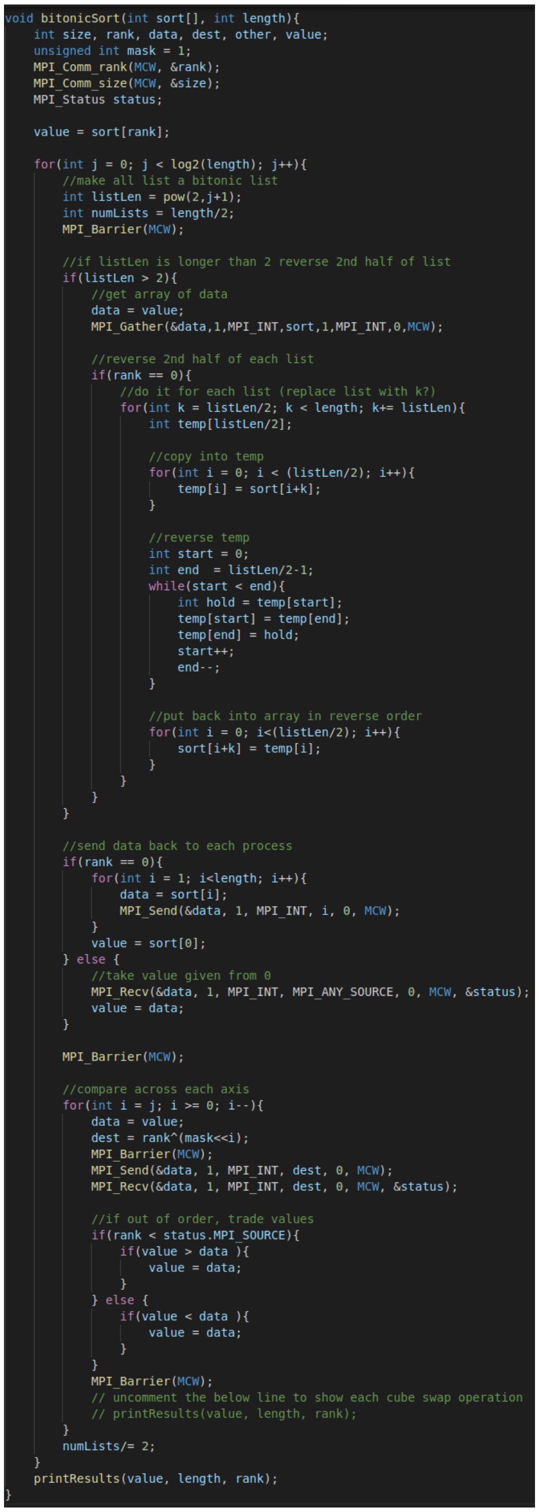
Then the program starts to do a bitonic sort by comparing across each axis as necessary. The process then repeats but with half the amount of lists (since they are merged together) and one higher cube swap operation. After the final sort and merge, the program then prints out the results

Below are the sample results

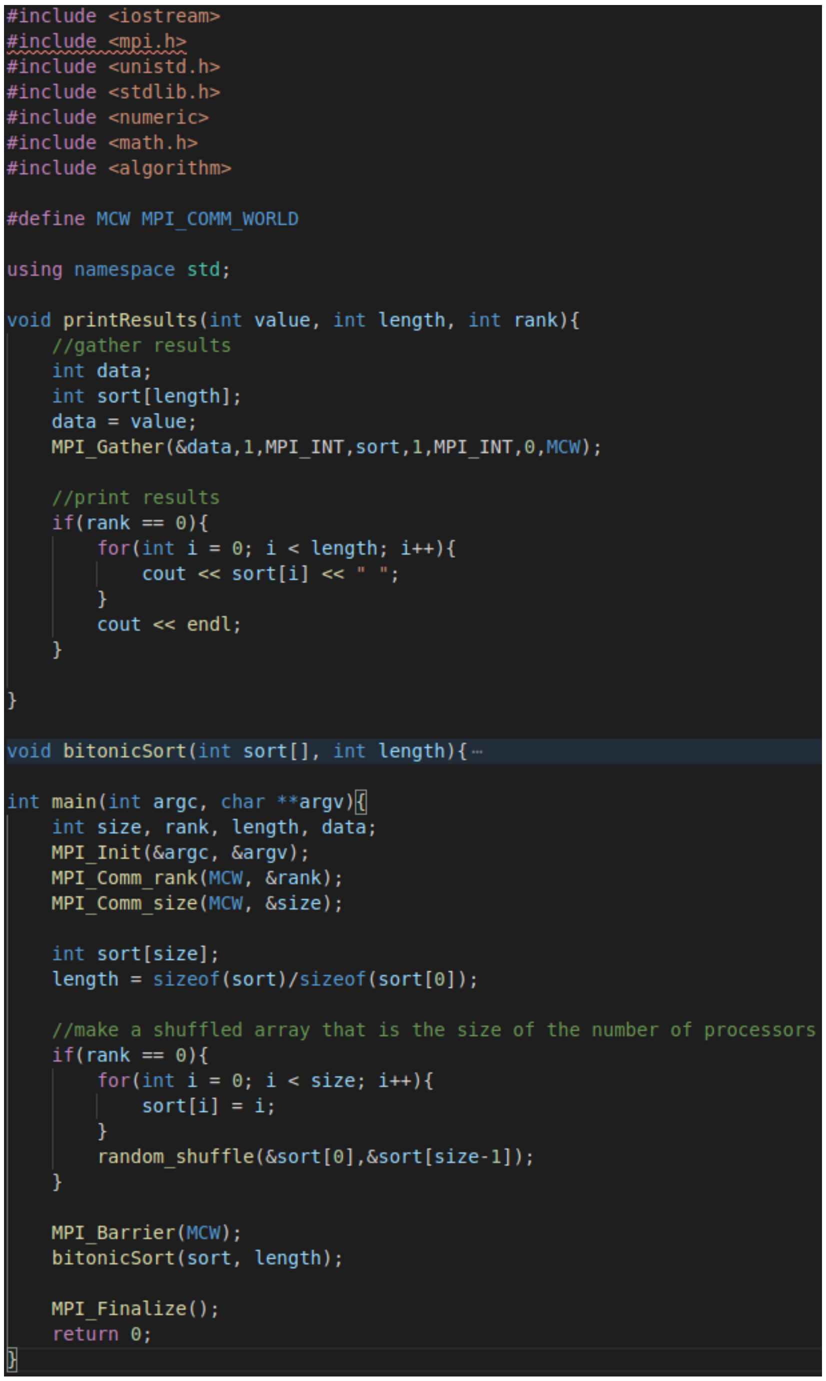


```
griffin@griffin-System-Product-Name:~/Desktop/HW5$ mpirun -np 16 -oversubscribe  a.out
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
griffin@griffin-System-Product-Name:~/Desktop/HW5$ mpirun -np 32 -oversubscribe  a.out
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
```

Full code is on the following pages



```c
void bitonicSort(int sort[], int length){
    int size, rank, data, dest, other, value;
    unsigned int mask = 1;
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);
    MPI_Status status;

    value = sort[rank];

    for(int j = 0; j < log2(length); j++){
        //make all list a bitonic list
        int listLen = pow(2,j+1);
        int numLists = length/2;
        MPI_Barrier(MCW);

        //if listLen is longer than 2 reverse 2nd half of list
        if(listLen > 2){
            //get array of data
            data = value;
            MPI_Gather(&data,1,MPI_INT,sort,1,MPI_INT,0,MCW);

            //reverse 2nd half of each list
            if(rank == 0){
                //do it for each list (replace list with k?)
                for(int k = listLen/2; k < length; k+= listLen){
                    int temp[listLen/2];

                    //copy into temp
                    for(int i = 0; i < (listLen/2); i++){
                        temp[i] = sort[i+k];
                    }

                    //reverse temp
                    int start = 0;
                    int end  = listLen/2-1;
                    while(start < end){
                        int hold = temp[start];
                        temp[start] = temp[end];
                        temp[end] = hold;
                        start++;
                        end--;
                    }

                    //put back into array in reverse order
                    for(int i = 0; i<(listLen/2); i++){
                        sort[i+k] = temp[i];
                    }
                }
            }
        }

        //send data back to each process
        if(rank == 0){
            for(int i = 1; i<length; i++){
                data = sort[i];
                MPI_Send(&data, 1, MPI_INT, i, 0, MCW);
            }
            value = sort[0];
        } else {
            //take value given from 0
            MPI_Recv(&data, 1, MPI_INT, MPI_ANY_SOURCE, 0, MCW, &status);
            value = data;
        }

        MPI_Barrier(MCW);

        //compare across each axis
        for(int i = j; i >= 0; i--){
            data = value;
            dest = rank^(mask<<i);
            MPI_Barrier(MCW);
            MPI_Send(&data, 1, MPI_INT, dest, 0, MCW);
            MPI_Recv(&data, 1, MPI_INT, dest, 0, MCW, &status);

            //if out of order, trade values
            if(rank < status.MPI_SOURCE){
                if(value > data ){
                    value = data;
                }
            } else {
                if(value < data ){
                    value = data;
                }
            }
            MPI_Barrier(MCW);
            // uncomment the below line to show each cube swap operation
            // printResults(value, length, rank);
        }
        numLists/= 2;
    }
    printResults(value, length, rank);
}
```



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

#define MCW MPI_COMM_WORLD

using namespace std;

void printResults(int value, int length, int rank){
    //gather results
    int data;
    int sort[length];
    data = value;
    MPI_Gather(&data,1,MPI_INT,sort,1,MPI_INT,0,MCW);

    //print results
    if(rank == 0){
        for(int i = 0; i < length; i++){
            cout << sort[i] << " ";
        }
        cout << endl;
    }

}

void bitonicSort(int sort[], int length){⋯

int main(int argc, char **argv){
    int size, rank, length, data;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);

    int sort[size];
    length = sizeof(sort)/sizeof(sort[0]);

    //make a shuffled array that is the size of the number of processors
    if(rank == 0){
        for(int i = 0; i < size; i++){
            sort[i] = i;
        }
        random_shuffle(&sort[0],&sort[size-1]);
    }

    MPI_Barrier(MCW);
    bitonicSort(sort, length);

    MPI_Finalize();
    return 0;
}
```