CS 5500 HW 4

Zac Johnson

February 2020

1 Implementation

1. Create unsorted array
2. Calculate sub-array size and send sub-arrays to processes
3. Each process sorts their sub-array via bitonic sort. First ranks sort in ascending order and the last ranks sort in descending order
4. Gather sub-arrays at ROOT
5. Perform Bitonic merge at ROOT

2 Compile and Run Commands

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

3 Code

```
#include <iostream>
#include <cstdlib>
#include <ctime>
//#include <mpi.h>
#include "/usr/local/include/mpi.h"

#define MCW MPI_COMM_WORLD
#define ROOT 0

using namespace std;

// NOTE: bitonic algorithm from geeksforgeeks.org

/*The parameter dir indicates the sorting direction, ASCENDING
   or DESCENDING; if (a[i] > a[j]) agrees with the direction,
   then a[i] and a[j] are interchanged.*/
void compAndSwap(int a[], int i, int j, int dir)
{
    if (dir==(a[i]>a[j]))
        swap(a[i],a[j]);
}

/*It recursively sorts a bitonic sequence in ascending order,
  if dir = 1, and in descending order otherwise (means dir=0).
  The sequence to be sorted starts at index position low,
  the parameter cnt is the number of elements to be sorted.*/
void bitonicMerge(int a[], int low, int cnt, int dir)
{
    if (cnt>1)
    {
        int k = cnt/2;
        for (int i=low; i<low+k; i++)
```

```
            compAndSwap(a, i, i+k, dir);
        bitonicMerge(a, low, k, dir);
        bitonicMerge(a, low+k, k, dir);
    }
}

/* This function first produces a bitonic sequence by recursively
    sorting its two halves in opposite sorting orders, and then
    calls bitonicMerge to make them in the same order */
void bitonicSort(int a[],int low, int cnt, int dir)
{
    if (cnt>1)
    {
        int k = cnt/2;

        // sort in ascending order since dir here is 1
        bitonicSort(a, low, k, 1);

        // sort in descending order since dir here is 0
        bitonicSort(a, low+k, k, 0);

        if (cnt == 16) {
            cout << "Bitonic: ";
            for (int i = 0; i < cnt; i++) {
                cout << a[i] << " ";
            }
            cout << endl;
        }

        // Will merge wole sequence in ascending order
        // since dir=1.
        bitonicMerge(a,low, cnt, dir);
    }
}

int main(int argc, char** argv) {
    int rank;
    int size;

    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MCW, &rank);
    MPI_Comm_size(MCW, &size);

    // Create unsorted array
    int n = 16;
    int *unsortedArray = new int[n];

    srand(time(NULL));

    for (int i = 0; i < n; i++) {
        unsortedArray[i] = rand() % 100;
    }

    if (rank == ROOT) {
        cout << "Unsorted: ";
        for (int i = 0; i < n; i++) {
            cout << unsortedArray[i] << " ";
        }
        cout << endl;
    }

    // Calculate subarray size
    int subSize = n / size;
    int *subArray = new int[subSize];

    // Send sub-arrays to processes
    MPI_Scatter(unsortedArray, subSize, MPI_INT, subArray, subSize, MPI_INT, ROOT, MCW);

```

```
    if (rank < size/2) {
        bitonicSort(subArray, 0, subSize, 1);
        cout << "Rank " << rank << " sorted ascending: ";

        for (int i = 0; i < subSize; i++) {
            cout << subArray[i] << " ";
        }
        cout << endl;
    } else {
        bitonicSort(subArray, 0, subSize, 0);
        cout << "Rank " << rank << " sorted descending: ";

        for (int i = 0; i < subSize; i++) {
            cout << subArray[i] << " ";
        }
        cout << endl;
    }

    // Gather sorted subarrays
    int *sorted = NULL;

    if(rank == 0) {
        sorted = new int[n];
    }

    MPI_Gather(subArray, subSize, MPI_INT, sorted, subSize, MPI_INT, ROOT, MCW);

    // Merge sorted sub-arrays at ROOT
    if (rank == ROOT) {

        int *final_array = new int[n];
        bitonicSort(sorted,0,n,1);

        cout << "Sorted: ";
        for (int i = 0; i < n; i++) {
            cout << sorted[i] << " ";
        }
        cout << endl;

        // Clean memory
        delete[] sorted;
        delete[] final_array;
    }

    // Clean up memory
    delete[] unsortedArray;
    delete[] subArray;

    MPI_Barrier(MCW);
    MPI_Finalize();

}
```

4 Desired Output

```
Unsorted: 42 57 17 8 31 9 11 89 22 50 97 48 97 91 39 15
Rank 0 sorted ascending: 42 57
Rank 7 sorted descending: 39 15
Rank 2 sorted ascending: 9 31
Rank 1 sorted ascending: 8 17
Rank 5 sorted descending: 97 48
Rank 4 sorted descending: 50 22
Rank 6 sorted descending: 97 91
Rank 3 sorted ascending: 11 89
Bitonic: 8 9 11 17 31 42 57 89 97 97 91 50 48 39 22 15
Sorted: 8 9 11 15 17 22 31 39 42 48 50 57 89 91 97 97
```