CS 5500 HW5 : Bitwise Sorting

Marshal Taylor

February 23, 2022

Abstract

A concise report describing my implementation if Bitwise Sorting

1 Introduction

Each time the program is run, every processor will get a random number. We will then sort each of the values given to each of the processors in ascending order.

2 Commands

1) Compile: VS code Build

2) `mpiexec -np {number of processsors} .\HM5BitwiseSorting.exe`

3) Due to the hypercube, the number of processors needs to be a power of 2

3 Implementation

3.1 Bitonic Sort

By increasing the dimensionality of the hypercube, we pair each processor up with a correct successive processor. Then, we swap the values of processors based on the minimum and maximum values out of each pair of numbers.

Since Bitwise sorting requires ascending and descending "lists" of values, we "flip" what the maximum and minimum values of the pair is based on the processors position in the hypercube. If we are a processor that needs to flip during a swap, and we are usually supposed to take the minimum value, we will instead take the maximum value.

Once we have a single list of ascending and a single list of descending values, we need to decrease the dimensionality of the cube while swapping each pair of values into the minimum and maximum spot, no flipping required. If we want an ascending list, then the lower rank takes the lower value. If we want a descending list, then the higher rank takes the lower value.

```
void bitonicSort(int rank, int data, int size) {
    int max_d = findSquare(size);
    int d = 0;
    int dest = rank;
    int recvData;
    bool flip = false;
    //int array[8];
    //int recvArray[8];

    unsigned int mask = 1;

    //cout << "Init | Rank: " << rank << " | Data: " << data << endl;
```

```
    while (d <= max_d) {

        dest = rank ^ (mask << d);
        //cout << "power "<< pow(2, (d+1)) << endl;
        flip = getFlip(rank, pow(2, (d+1)));

        cout << "Rank " << rank << " | Dest " << dest << " | Data: " << data << " | flip
        MPI_Send(&data, 1, MPI_INT, dest, 0, MCW);
        MPI_Recv(&recvData, 1, MPI_INT, dest, 0, MCW, MPI_STATUS_IGNORE);
        if (flip) {
            if (rank > dest) {
                data = min(data, recvData);
            }
            else {
                data = max(data, recvData);
            }
        }
        else {
            if (rank < dest) {
                data = min(data, recvData);
            }
            else {
                data = max(data, recvData);
            }
        }
        d++;
    }
    d--;
    while (d >= 0) {

        dest = rank ^ (mask << d);
        //cout << "power "<< pow(2, (d+1)) << endl;
        cout << "Rank " << rank << " | Dest " << dest << " | Data: " << data <<
endl;
        MPI_Send(&data, 1, MPI_INT, dest, 0, MCW);
        MPI_Recv(&recvData, 1, MPI_INT, dest, 0, MCW, MPI_STATUS_IGNORE);
        if (rank < dest) {
            data = min(data, recvData);
        }
        else {
            data = max(data, recvData);
        }
        d--;
    }
    cout << "Final | Rank: " << rank << " | Data: " << data << endl;

    return;
}
```

4 Expected Output

Number of processors lines of printed output. Each printed command line should show the starting value of the a given processor, and the ending value of each processor.

With the final output, you can piece together the correct sorted list, since each successive processors rank will have a larger/smaller number than the previous processors rank.