Parallel Programing: HW4

Andres Imperial

February 8, 2020

Implementation

I implemented my code by first having srand() init my random number generator. Then I had process 0 start by creating an array of random integers size ¡n¿ where n is the number of processors used to start the program. Process 0 then sends the data to corresponding threads. The threads then find their partners through bit shift there values using the log(n) nature of bitonic sorting. After the threads compare multiple times with their partners the 0 thread then collects everyones result and prints them out.

Code:

Listing 1: main.cpp

```
#include <cmath>
#include <cstdio>
#include <mpi.h>
#include <time.h>
#include <vector>

using namespace std;
#define MCW MPI_COMM_WORLD

pair<bool, int> compareLow(int myIdx, int j, int myVal) {
  int myPartner = myIdx ^ (1 << j);
  MPI_Send(&myVal, 1, MPI_INT, myPartner, 0, MCW);
  int compVal;
  MPI_Recv(&compVal, 1, MPI_INT, myPartner, 0, MCW, MPI_STATUS_IGNORE);
  return make_pair(compVal < myVal, compVal);
}

pair<bool, int> compareHigh(int myIdx, int j, int myVal) {
  auto result = compareLow(myIdx, j, myVal);
  return make_pair(!result.first, result.second);
}
```

```
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);
  auto dimensions = log2(size);
  int myVal;
  std::vector<int> totalArray = {};

  if (rank == 0) {
    // create random array sized < 100
    srand(time(0));
    // Populate array with random int < 100
    for (int i = 0; i < size; i++) {
      totalArray.push_back(rand() % 100);
    }

    myVal = totalArray[0];
    for (int i = 1; i < size; i++) {
      MPI_Send(&totalArray[i], 1, MPI_INT, i, 0, MCW);
    }
  } else {
    MPI_Recv(&myVal, 1, MPI_INT, 0, 0, MCW, MPI_STATUS_IGNORE);
  }

  {
    for (int i = 0; i < dimensions; i++) {
      for (int j = i; j >= 0; j--) {
        if (((rank >> (i + 1)) % 2 == 0 && (rank >> j) % 2 == 0) ||
            ((rank >> (i + 1)) % 2 != 0 && (rank >> j) % 2 != 0)) {
          auto result = compareLow(rank, j, myVal);
          if (result.first) {
            myVal = result.second;
          }
        } else {
          auto result = compareHigh(rank, j, myVal);
          if (result.first) {
            myVal = result.second;
          }
        }
      }
    }
  }
```

```
    if (rank == 0) {
        totalArray[0] = myVal;
        for (int i = 1; i < size; i++) {
            int temp;
            MPI_Recv(&temp, 1, MPI_INT, i, 0, MCW, MPI_STATUS_IGNORE);
            totalArray[i] = temp;
        }
    } else {
        MPI_Send(&myVal, 1, MPI_INT, 0, 0, MCW);
    }

    if (rank == 0) {
        printf("totalArray:");
        for (int i = 0; i < size; i++) {
            printf("%i,", totalArray[i]);
        }
        printf("\n");
    }

    MPI_Finalize();

    return 0;
}
```

How to run:

```
mpic++ main.cpp -o main
mpirun -np <n> -oversubscribe ./main
```

Output

```
(if 32 was used for np)
totalArray:4,8,10,13,14,15,26,26,28,35,38,42,45,55,55,58,59,60,61,65,68,69,69
```

Outcome subject to change because of randomly generated numbers, but should always be in order.