HW 4
Bitonic Sort

Philip Nelson

2018 September 28

Introduction

The purpose of this assignment is to write an MPI program that performs a bionic integer sort using integers. The sort only works when there is a power of 2 number of processors, it will exit otherwise.

Code

```
#include "random.hpp"
#include <cmath>
#include <iomanip>
#include <iostream>
#include <mpi.h>

#define MCW MPI_COMM_WORLD

void print1per(int data, std::string title)
{
  int rank;
  int size;

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

  int* dArray = new int[size];
  MPI_Gather(&data, 1, MPI_INT, dArray, 1, MPI_INT, 0, MCW);

  if (rank == 0)
  {
    std::cout << title << '\n';
    for (int i = 0; i < size; ++i)
    {
      std::cout << std::setw(5) << i << std::setw(5) << dArray[i] << "\n";
    }
    std::cout << std::endl;
  }
}

int cube(int c, int sendData, int rank)
{
  int recvData;
  auto dest = rank ^ (1 << c);

  MPI_Send(&sendData, 1, MPI_INT, dest, 0, MCW);
  MPI_Recv(&recvData, 1, MPI_INT, dest, 0, MCW, MPI_STATUS_IGNORE);

  return recvData;
}

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

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

  if (0 != (size & (size - 1)))
  {
    if (rank == 0)
    {
      std::cerr << "There must be a power of 2 number of threads\n";
    }

    MPI_Finalize();
    exit(EXIT_FAILURE);
  }

  int data = random_int(0, 100);
  print1per(data, "unsorted");
  int steps = log2(size);
  for (int i = 0; i < steps; ++i)
  {
    for (int j = i; j >= 0; --j)
    {
      auto recv = cube(j, data, rank);
      auto dest = rank ^ (1 << j);
      if (rank % (int)pow(2, i + 2) < pow(2, i + 1))
      {
        // ascending
        if (rank < dest)
          data = std::min(recv, data);
        else
          data = std::max(recv, data);
      }
      else
      {
        // descending
        if (rank < dest)
          data = std::max(recv, data);
        else
          data = std::min(recv, data);
      }
    }
  }
  print1per(data, "sorted");

  MPI_Finalize();

  return EXIT_SUCCESS;
}
```

Output

```
# mpic++ -O3 bitonic.cpp -o release.out

# mpiexec -n 8 --oversubscribe release.out

unsorted
    0   38
    1   72
    2   52
    3   22
    4   19
    5   83
    6   82
    7   90

sorted
    0   19
    1   22
    2   38
    3   52
    4   72
    5   82
    6   83
    7   90

# mpiexec -n 16 --oversubscribe release.out

unsorted
    0   39
    1   30
    2   52
    3   37
    4   88
    5   76
    6   21
    7   30
    8   62
    9    4
   10    3
   11   76
   12   39
   13   68
   14   81
   15   50
```

```
sorted
    0    3
    1    4
    2   21
    3   30
    4   30
    5   37
    6   39
    7   39
    8   50
    9   52
   10   62
   11   68
   12   76
   13   76
   14   81
   15   88
```