6550 Parallel Programming HW4

Riley Densley

September 2019

1 Bitonic Sorting

The array to be sorted is made using the random-shuffle function which seeds the random so all processes have the same array. Then all processes get their number by getting the rank position in the array. All processes then begin the bitonic sorting by sending their number to the process that they may be switching with. This is saved in a temp variable. The process then sees if it should be assending or decending. Then, debening on its rank and the rank that was sent to it, it decides if it should keep the new number or keep its old number. This process continues following the bitonic sorting methond of making many small bitonic lists and adding to those lists until the entire array is sorted.

2 Code

```
#include <iostream>
#include <mpi.h>
#include <unistd.h>
#include <stdio.h>
#include <algorithm>
#include <cmath>
//#include "/usr/local/include/mpi.h"
#define MCW MPI_COMM_WORLD

using namespace std;

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);
MPI_Status mystatus;

int temp;
bool biggerRank;
int sendTo;

int initialSet[size];
for (int i = 0; i < size; i++)
{
  initialSet[i] = size - 1 - i;
}

random_shuffle(&initialSet[0], &initialSet[size - 1]);
if (rank == 0)
{
  for (int i = 0; i < size; i++)
  {
    cout << initialSet[i] << "  ";
  }
  cout << endl;
}

data = initialSet[rank];

for (int j = 0; j < log2(size); j++)
{
  for (int i = j; i >= 0; i--)
  {
    bool ascending = true;
    if (rank & (1 << (j + 1)))
    {
      ascending = false;
    }
    biggerRank = (rank & (1 << i));
    sendTo = (rank xor (1 << i));
    MPI_Send(&data, 1, MPI_INT, sendTo, 0, MCW);
    MPI_Recv(&temp, 1, MPI_INT, sendTo, 0, MCW, MPI_STATUS_IGNORE);

    if (ascending)
    {
      data = (biggerRank == data > temp) ? data : temp;
    }
    else
    {
      data = (biggerRank == data < temp) ? data : temp;
      }
    }
  }
  MPI_Barrier;

  //Send all values back to 0 for cout
  if (rank == 0)
  {
    initialSet[0] = data;
    for (int i = 1; i < size; i++)
    {
      MPI_Recv(&temp, 1, MPI_INT, i, 0, MCW, MPI_STATUS_IGNORE);
      initialSet[i] = temp;
    }
    for (int i = 0; i < size; i++)
    {
      cout << initialSet[i] << "  ";
    }
    cout << endl;
  }
  else
  {
    MPI_Send(&data, 1, MPI_INT, 0, 0, MCW);
  }


  MPI_Finalize();

  return 0;
}
```

3 Output

3.1 Commands

```
mpic++ assign4.cpp
mpirun -np 16 ./a.out
```

3.2 Output

```
11  5  4  3  1  10  2  14  9  6  12  8   7   13  15  0
0   1  2  3  4  5   6  7   8  9  10  11  12  13  14  15
```