Bitonic Sort

Aaron Morgenegg

09/28/18

OpenMPI Bitonic Sort Program

This program uses a bitonic sorting algorithm to sort a bitonic list using muliple processes. First, process 0 will generate a bitonic list, print it, and scatter the values of that list to each process. Then the algorithm loops from 0 to log2(worldSize), and calls the cube(i) function. Each process will compare the value it holds with the value that the next process holds(based on the cube function), and will swap if they are out of order. At the end, the data is sent back to process 0 and the sorted list is printed.

```
/*
Example of Compiling and running code + example output

mpic++ main.cpp
mpiexec -np -oversubscribe 8 a.out

Unsorted List:
0:4 1:5 2:6 3:7 4:3 5:2 6:1 7:0
Sorted List:
0:0 1:1 2:2 3:3 4:4 5:5 6:6 7:7
*/

#include <iostream>
#include <time.h>
#include <stdlib.h>
#include <mpi.h>
#include <unistd.h>
#include <algorithm>
```

```
#include <math.h>

int * getBitonicList(int size){ // Create a bitonic list of given size
// CAUTION : designed to work with size=power of 2
int * list = new int[size];
int value = size/2;
for(int i = 0; i < size; i++){
list[i] = value;
if(i == size/2-1){
value = size/2-1;
} else if(i >= size/2){
value--;
} else{
value++;
}
}
return list;
}

int * printList(int * list, int size){
for(int i=0;i<size;++i){
std::cout<<i<<":";
std::cout<<list[i]<<" ";
}
std::cout<<std::endl;
}

void print1per(int data, int rank, int size){ // Print 1 per function from class
int *dArray = new int [size];
MPI_Gather(&data,1,MPI_INT,dArray,1,MPI_INT,0,MPI_COMM_WORLD);
if(rank==0){
std::cout << "Sorted List: " << std::endl;
printList(dArray, size);
}
return;
}

int cube(int i, int sendData){ // Cube function from class
int rank;
int size;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
MPI_Comm_size(MPI_COMM_WORLD, &size);
int dest;
int recvData;
int mask=1;
mask = mask << i;
dest = rank ^ mask;
MPI_Send(&sendData,1,MPI_INT,dest,0,MPI_COMM_WORLD);
MPI_Recv(&recvData,1,MPI_INT,dest,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
if(rank < dest){
if(sendData<recvData) return sendData;
return recvData;
}
if(sendData<recvData) return recvData;
return sendData;
}

void bitonicSort(int world_size, int world_rank, int list_size){
int * list = new int[list_size];
if(world_rank==0) {
list = getBitonicList(list_size);
std::cout << "Unsorted List: " << std::endl;
printList(list, list_size);
for(int i = 0; i < world_size; i++){
MPI_Send(&list[i],1,MPI_INT,i,0,MPI_COMM_WORLD);
}
}
int recv_data;
MPI_Recv(&recv_data,1,MPI_INT,0,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
for(int i = 0; i < log2(world_size); i++){
recv_data = cube(i, recv_data);
}
MPI_Send(&recv_data,1,MPI_INT,0,0,MPI_COMM_WORLD);
print1per(recv_data, world_rank, list_size);
}
```

```
int main(int argc, char** argv) {
MPI_Init(&argc, &argv);
srand(time(NULL));
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

bitonicSort(world_size, world_rank, world_size);

MPI_Finalize();
return 0;
}
```