Integer Sort

Aaron Morgenegg

09/21/18

OpenMPI Integer Sort Program

This program uses a master/slave configuration to sort a list of integers in a parallel fashion. First, the master process generates a list of random integers, and scatters a sub list to each process. Then, each process(including master) will sort the sub list they have receieved. After sorting, the sub lists will be sent back to the master process, where they will be merged together to create the final, sorted list.

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

mpic++ main.cpp
mpiexec -np 2 a.out

Sorting the following list:
70,987,282,584,768,487,922,729,465,271,40,428,19,296,942,463,357,841,
337,96,312,722,575,31,501,251,840,141,500,668,79,570,655,361,506,776,
200,428,857,665,52,898,445,71,194,740,886,903,933,575,999,597,297,927,
628,799,530,821,292,30,489,371,600,496,84,106,272,285,886,130,302,938,
380,100,9,574,840,895,829,773,823,180,722,120,107,351,271,637,172,563,
19,13,286,619,509,723,77,134,360,964,
Sorted List:
9,13,19,19,30,31,40,52,70,71,77,79,84,96,100,106,107,120,130,134,141,
172,180,194,200,209,251,271,271,272,282,285,286,292,296,297,302,312,
337,351,357,360,361,371,380,428,428,445,463,465,487,489,496,500,501,
506,509,530,563,570,574,575,575,584,597,600,619,628,637,655,665,668,
722,722,723,729,740,768,773,776,799,821,823,829,840,840,841,857,886,
886,895,898,903,922,927,933,938,942,964,987,

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

const int MAX_INT_SIZE = 1000;
const int LIST_SIZE = 100;

int * GetUnsortedList(int n){
int * list = new int[n];
for(int i = 0; i < n; i++){
list[i] = rand() % MAX_INT_SIZE;
}
return list;
}

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

int * SortList(int * unsorted_list, int list_length){
std::sort(unsorted_list, unsorted_list+list_length);
return unsorted_list;
}

int * MergeLists(int ** split_lists, int world_size){
int * merged_list = new int[LIST_SIZE];
for(int i = 0; i < LIST_SIZE; i++){
int min_index = 0;
for(int j = 1; j < world_size; j++){
```

```
if(split_lists[j][0] < split_lists[min_index][0]){
min_index = j;
}
}
merged_list[i] = split_lists[min_index][0];
split_lists[min_index]+=1;
}
return merged_list;
}

int * CreateAndPrintList(int world_size){
int * unsorted_list = GetUnsortedList(LIST_SIZE);
std::cout << "Sorting the following list: " << std::endl;
PrintList(unsorted_list, LIST_SIZE);
return unsorted_list;
}

int main(int argc, char** argv) {
// initialize MPI
MPI_Init(&argc, &argv);

// Initialize rng
srand(time(NULL));

// stores number of processes in world_size
int world_size;
MPI_Comm_size(MPI_COMM_WORLD, &world_size);

// Get the rank of this process
int world_rank;
MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);

int sub_length = LIST_SIZE/(world_size);

int * unsorted_list;
if(world_rank == 0){
unsorted_list = CreateAndPrintList(world_size);
}
```

```
int * sub_list = new int[sub_length];
MPI_Scatter(unsorted_list, sub_length, MPI_INT, sub_list, sub_length, MPI_INT, 0,

SortList(sub_list, sub_length);
MPI_Send(sub_list, sub_length, MPI_INT, 0, 0, MPI_COMM_WORLD);

if(world_rank == 0){
int ** sub_lists = new int*[world_size];
for(int i = 0; i < world_size; i++){
sub_lists[i] = new int[sub_length];
MPI_Recv(sub_lists[i], sub_length, MPI_INT, i, 0, MPI_COMM_WORLD, MPI_STATUS_IGNOR
}
int * sorted_list = MergeLists(sub_lists, world_size);
std::cout << "Sorted List: " << std::endl;
PrintList(sorted_list, LIST_SIZE);
}

// Finalize the MPI environment.
MPI_Finalize();
return 0;
}
```