Homework #2

(Due: April 15)

Task 1. [100 Points] Parallel Sorting.

(*a*) [**25 + 25 Points**] Implement the following two parallel sorting algorithms:

- randomized quicksort (slides 2–18, lecture 8), and
- radix sort with ranking by counting sort (slides 13–17, lecture 10).

(*b*) [**10 Points**] In order to avoid the overhead of recursion in your randomized quicksort implementation do not recurse down to inputs of size 1. Instead, you should stop recursing at some suitable input size m (≥ 1) and switch to a serial iterative sorting algorithm, e.g., insertion sort, that is known to perform well on small inputs. Let's assume for simplicity that both n and m are powers of 2, and $m \geq 32$. First, find the largest value n' of n such that sorting n' integers with $m = 32$ using your parallel quicksort implementation takes less than 15 seconds on average (average of 3 runs). Now keeping n fixed at n' find the value m' of m that gives you the smallest running time for your quicksort implementation. Produce a table or a graph showing how the running time varies as you change m.

(*c*) [**20 Points**] Find the largest integer r such that both your sorting implementations take less than 2 minutes to sort $n = 2^r$ integers when only one processing core is used. Use the optimized m value from part (*a*) for your quicksort implementation. Now produce a plot showing the running times of both your implementations for $n = 2^r$ when you vary the number of processing cores p from 1 to the maximum number of available cores.

(*d*) [**20 Points**] Produce a plot showing the running times of both your implementations using all processing cores when you vary n from m to 2^r. Use only powers of 2 for n.

Task 2. [100 Points] Parallel MSF.

(*a*) [**25 + 25 Points**] Implement the two randomized parallel MSF algorithms where priority concurrent write is simulated using:

- radix sort with ranking by counting sort (slides 13–21, lecture 10), and
- binary search (slides 22–38, lecture 10).

In both cases use your quicksort implementation from task 1 for the initial sorting of the edges by weight.

(*b*) [**25 Points**] Create a table that compares the running times of the two MST implementations using all cores. For each input file (in Appendix 1) create a separate row in the table showing the running times of both algorithms.

(*c*) [**25 Points**] Generate a strong scalability plot (see slide 19 of lecture 2) for each of the two implementations using the Live Journal graph (see Appendix 1) as input.

APPENDIX 1: Input/Output Format for Task 2

Your code must read from standard input and write to standard output.

- **Input Format:** The first line of the input will contain two integers giving the number of vertices (n) and the number of edges (m), respectively. Each of the next m lines will contain two integers u and v ($1 \leq u, v \leq n$) and a nonnegative floating point number w denoting an undirected edge of weight w between vertices u and v.
- **Output Format:** The first line of the output will contain an integer m' giving the number of edges in the MSF and a floating point number c giving the total cost of the MSF. Each of the next m' lines will contain an MSF edge specified by its two end points (i.e., two integers) and the weight (i.e., a floating point number).
- **Sample Input/Output:** `/work2/01905/rezaul/CSE613/HW2/samples` on Stampede2.
- **Test Input/Output (Table 1):** `/work2/01905/rezaul/CSE613/HW2/turn-in` on Stampede2.

Graph	Description	n	m
as-skitter	Internet topology graph, from traceroutes run daily in 2005	1.7M	11M
ca-AstroPh	Collaboration network of Arxiv Astro Physics	18.7K	396K
com-amazon	Amazon product network	334K	925K
com-dblp	DBLP collaboration network	317K	1M
com-lj	LiveJournal online social network	4M	34M
com-orkut	Orkut online social network	3M	117M
roadNet-CA	Road network of California	2M	2.7M
roadNet-PA	Road network of Pennsylvania	1M	1.5M
roadNet-TX	Road network of Texas	1.4M	1.9M

Table 1: Input graphs with #vertices (n) and #edges (m).

APPENDIX 2: What to Turn in

One compressed archive file (e.g., zip, tar.gz) containing the following items.

- Source code, makefiles and job scripts.
- A PDF document containing all answers and plots.
- Output generated for the input files in `/work2/01905/rezaul/CSE613/HW2/turn-in/` on Stampede2. If the name of the input file is `xxxxx-in.txt`, please name the output files as `xxxxx-MSF-sort-out.txt` and `xxxxx-MSF-search-out.txt` for MSF algorithms where priority concurrent write is simulated using radix sort with ranking by counting sort and binary search, respectively.

APPENDIX 3: Things to Remember

- **Please never run anything that takes more than a minute or uses multiple cores on login nodes.** Doing so may result in account suspension. All runs must be submitted as jobs to compute nodes (even when you use Cilkview or PAPI).
- Please store all data in your work folder ($WORK), and not in your home folder ($HOME).
- When measuring running times please exclude the time needed for reading the input and writing the output. Measure only the time needed by the algorithm.