Homework #2

(Due: April 16)

Task 1. [40 Points] Parallel BFS with Cilk's Work Stealing.

(a) [**20 Points**] In homework 1 we analyzed and implemented parallel BFS algorithms based on randomized distributed work-stealing, and we explicitly implemented the work-stealing mechanism. Explain how you will reimplement the algorithm from part $1(f)$ of homework 1 (without the modication for avoiding duplicate exploration) so that it uses Cilk's work-stealing scheduler for load balancing instead of your own work-stealing code.

(b) [**20 Points**] Update the table you created for part $1(j)$ of homework 1 with the running times of your new implementation.

Task 2. [160 Points] Parallel Connected Components.

(a) [**10 Points**] Consider the function RANDOM-HOOK(V, L, N) in Figure 1, and assume that V has no zero degree vertices. Suppose we start with an empty set Q, and add edges to it as follows. We traverse the vertices in V in some order, and for each $v \in V$ encountered in that order, we add $(v, N[v])$ to Q provided the graph induced by $Q \cup \{(v, N[v])\}$ does not contain a cycle. Prove that $|Q| \geq \frac{|V|}{2}$.

(b) [**10 Points**] Consider the set Q constructed in part $2(a)$. We say that an edge $(u, v) \in Q$ is a *hook* provided $C[u] \neq C[v]$ right after step 3 of RANDOM-HOOK. Prove that for all $Q' \subset Q$, if one knows the *hook* status of all edges in Q' that still does not reveal anything about the hook status of any edge in $Q \setminus Q'$.

(c) [**10 Points**] Explain why RANDOM-HOOK (in Figure 1) is superior to the random hooking technique we saw in the class (used in PAR-RANDOMIZED-CC).

(d) [**10 Points**] Use your results from parts $2(a)$–$2(c)$ to show that in any invocation of RANDOM-HOOK(V, L, N), with probability at least $1 - \frac{1}{e^{|V|/32}}$ at least $\frac{|V|}{16}$ vertices of V will change their L values.

(e) [**10 Points**] Use your result from part $2(d)$ to prove high probability bounds for the work and span of PAR-RANDOMIZED-CC-2.

(f) [**15 Points**] Consider PAR-RANDOMIZED-CC-3. The algorithm repeatedly chooses a random sample (of geometrically decreasing sizes) from the edges of the input graph, and uses the sample to identify vertices (or supervertices) that have *potentially high degree* (PhD). Each vertex of the graph starts with a PhD status, but loses the status as soon as a chosen sample

of edges fails to include an edge connecting that vertex with another vertex with a PhD status. The RANDOM-HOOK function is called only on the vertices that retain the PhD status. After a sufficient number of such sampling and hooking rounds, the number of edges among the supervertices reduce to a sufficiently small number. At that point connected components among the supervertices are found by calling PAR-RANDOMIZED-CC-2. Now suppose at recursion depth d of PAR-RANDOMIZED-CC-3, n_d denotes the number of vertices/supervertices in V with a PhD status. Then n_0 is the number of vertices in the original input graph. Let $n = n_0$. Prove that for each $d \in [0, d_{max}]$, $n_d \leq \alpha^{2d} \cdot n$ w.h.p. in n, where $\alpha = \sqrt{\frac{15}{16}}$ and $d_{max} = \left\lceil \frac{1}{4} \log_{\frac{1}{\alpha}} n \right\rceil$.

(g) [**15 Points**] We call an edge (u, v) *heavy* provided $PhD[u] = PhD[v] =$ TRUE, otherwise we call it *light*. At recursion depth d of PAR-RANDOMIZED-CC-3, let r_d be the expected number of heavy edges that become light. Prove that for each $d \in [0, d_{max}]$, $r_d \leq \alpha^d \cdot n$.

(h) [**10 Points**] Prove that in the call to PAR-RANDOMIZED-CC-2 in line 21 of PAR-RANDOMIZED-CC-3, the expected number of edges in E' is $\mathcal{O}(n)$.

(i) [**10 Points**] Compute the expected work and span of PAR-RANDOMIZED-CC-3.

(j) [**60 Points**] Implement PAR-RANDOMIZED-CC (the one we saw in the class), PAR-RANDOMIZED-CC-2 (from Figure 1) and PAR-RANDOMIZED-CC-3 (from Figure 2), and optimize them. Create a table that compares the running times of these three implementations using all cores. For each input file (in Appendix 1) create a separate row in the table showing the running times of all three algorithms.

(k) [**20 Points**] Generate a `Cilkview` strong scalability plot (see slides 15–18 of lecture 5) for each of the two parallel implementations from part (j) using the Live Journal graph (see Appendix 1) as input.

PAR-RANDOMIZED-CC-2($V,\ E,\ L$)

(Input is an unweighted undirected graph with vertex set V and edge set E. For each $v \in V$, $L[v]$ is set to v before the invocation of this function. When this function terminates, for each $v \in V$, $L[v]$ contains the unique id of the connected component containing v. We call an edge (u, v) *live* provided $L[u] \neq L[v]$.)

1. ***if*** $|E| = 0$ ***then return*** {*no edge to contract*}
2. ***parallel for*** each $(u, v) \in E$ ***do*** $N[u] \leftarrow v,\ \ N[v] \leftarrow u$ {*try to associate the edge* (u, v) *with* u *and* v}
3. RANDOM-HOOK($V,\ L,\ N$) {*hook among vertices in* V *based on the edges chosen in the previous step*}
4. $V' \leftarrow \{\ v \mid v \in V\ \wedge\ v = L[v]\ \}$ {V' *contains only the roots after hooking*}
5. $E' \leftarrow \{\ (L[u], L[v]) \mid (u, v) \in E\ \wedge\ L[u] \neq L[v]\ \}$ {E' *contains only edges among roots, and no duplicate edges and self loops*}
6. PAR-RANDOMIZED-CC-2($V',\ E',\ L$) {*recurse on the smaller instance*}
7. ***parallel for*** each $v \in V$ ***do*** $L[v] \leftarrow L[L[v]]$ {*map the solution back to the current instance*}

RANDOM-HOOK($V,\ L,\ N$)

(Input is an unweighted undirected graph with vertex set V. For each $v \in V$, $L[v]$ is set to v before the invocation of this function. For each $u \in V$, $N[u]$ is set to a v such that (u, v) is an edge in the graph. This function randomly hooks vertices in V to their neighbors in such a way that after the function terminates these vertices form a set of disjoint stars. For each $v \in V$, $L[v]$ is set to u (possibly $u = v$) provided u is the center of the star containing v.)

1. ***parallel for*** each $u \in V$ ***do*** {*for each vertex in* V}
2. $\quad C_u \leftarrow$ RANDOM { HEAD, TAIL } {*toss a coin*}
3. $\quad H_u \leftarrow$ FALSE {*record that this vertex has not yet been hooked*}
4. ***parallel for*** each $u \in V$ ***do*** {*for each vertex* u *in* V}
5. $\quad v \leftarrow N[u]$ {*will try to hook* u *with* $v = N[u]$}
6. $\quad$ ***if*** $C_u =$ TAIL ***and*** $C_v =$ HEAD ***then*** {*if* u *tossed* TAIL *and* v *tossed* HEAD}
7. $\quad\quad L[u] \leftarrow v$ {*make* u *point to* v}
8. $\quad\quad H_u \leftarrow$ TRUE, $H_v \leftarrow$ TRUE {*record that both* u *and* v *are hooked*}
9. ***parallel for*** each $u \in V$ ***do*** {*manipulate the coin tosses to hook more in a second try*}
10. $\quad$ ***if*** $H_u =$ TRUE ***then*** $C_u \leftarrow$ HEAD {*if* u *is already hooked, will try to hook unhooked vertices pointing to* u}
11. $\quad$ ***else if*** $C_u =$ TAIL ***then*** $C_u \leftarrow$ HEAD ***else*** $C_u \leftarrow$ TAIL {*if* u *is not hooked, flip* C_u}
12. ***parallel for*** each $u \in V$ ***do*** {*try to hook again*}
13. $\quad v \leftarrow N[u]$ {*will try to hook* u *with* $v = N[u]$}
14. $\quad$ ***if*** $C_u =$ TAIL ***and*** $C_v =$ HEAD ***then*** {*if* u *has* TAIL *and* v *has* HEAD}
15. $\quad\quad L[u] \leftarrow L[v]$ {*make* u *point to whatever* v *is pointing to*}

Figure 1: Parallel connected components (CC) on a graph.

PAR-RANDOMIZED-CC-3($V,\ E,\ L,\ PhD,\ N,\ U,\ d$)

(Input is an unweighted undirected graph with vertex set V and edge set E. The recursion depth of the function is given by d which is set to 0 when the function is invoked for the first time. Let n be the number of vertices in the graph when $d = 0$, and $m = |E|$. Each $v \in V$ is an integer in $[1, n]$. Pointers L (label), PhD (potentially high degree), N (neighbor) and U (updated) point to arrays $L[1:n]$, $PhD[1:n]$, $N[1:n]$ and $U[1:n]$, respectively. For each $v \in [1, n]$, $L[v]$ is set to v, and $PhD[v]$ is set to TRUE before the initial invocation of this function. When this function terminates, for each $v \in V$, $L[v]$ contains the unique id of the connected component containing v. We assume that $\alpha = \sqrt{\frac{15}{16}}$ and $d_{max} = \left\lceil \frac{1}{4} \log_{\frac{1}{\alpha}} n \right\rceil$. We call an edge (u, v) *live* provided $L[u] \neq L[v]$. Edge (u, v) is *heavy* provided $PhD[u] = PhD[v]$, otherwise it is *light*.)

1. ***if*** $d \leq d_{max}$ ***then*** {*need to recurse more to sufficiently reduce #vertices with PhD status*}
2. $m_d \leftarrow \lceil m \cdot \alpha^d \rceil$ {*size of edge sample which geometrically decreases with d*}
3. $\widehat{E} \leftarrow$ a sample of size m_d chosen uniformly at random from E {*do not always touch all edges in E*}
4. ***parallel for*** each $v \in V$ ***do*** $U[v] \leftarrow$ FALSE {*flag U[v] keeps track if an edge in $\widehat{E}$ hits v*}
5. ***parallel for*** each $(u, v) \in \widehat{E}$ ***do*** {*check each edge in the sample*}
6. $u' \leftarrow L[u],\ v' \leftarrow L[v]$ {*find the root of the tree containing each endpoint*}
7. ***if*** $u' \in V$ ***and*** $v' \in V$ ***and*** $u' \neq v'$ ***and*** $PhD[u'] = PhD[v'] =$ TRUE ***then*** {*if (u', v') is live and heavy*}
8. $N[u'] \leftarrow v',\ N[v'] \leftarrow u'$ {*try to associate the edge with u' and v'*}
9. $U[u'] \leftarrow$ TRUE, $U[v'] \leftarrow$ TRUE {*$\widehat{E}$ hits u' and v'*}
10. ***parallel for*** each $v \in V$ ***do*** {*check each vertex v in V*}
11. ***if*** $U[v] =$ FALSE ***then*** $PhD[v] \leftarrow$ FALSE {*if $\widehat{E}$ does not hit v then v loses its PhD status*}
12. $\widehat{V} \leftarrow \{\ v \mid v \in V\ \wedge\ U[v] =$ TRUE $\}$ {*$\widehat{V}$ contains the vertices from V which still have PhD status*}
13. RANDOM-HOOK($\widehat{V},\ L,\ N$) {*hook among vertices in $\widehat{V}$ (see Figure 1)*}
14. $V' \leftarrow \{\ v \mid v \in V\ \wedge\ v = L[v]\ \}$ {*V' contains only the roots after hooking*}
15. PAR-RANDOMIZED-CC-3($V',\ E,\ L,\ PhD,\ N,\ U,\ d+1$) {*recurse on the smaller instance*}
16. ***parallel for*** each $v \in V$ ***do*** $L[v] \leftarrow L[L[v]]$ {*map the solution back to the current instance*}
17. ***endif***
18. ***if*** $d = 0$ ***then*** {*done compressing*}
19. $V' \leftarrow \{\ v \mid v \in V\ \wedge\ v = L[v]\ \}$ {*collect only the root vertices*}
20. $E' \leftarrow \{\ (L[u], L[v]) \mid (u, v) \in E\ \wedge\ L[u] \neq L[v]\ \}$ {*E' contains only edges among roots, and no duplicate edges and self loops*}
21. PAR-RANDOMIZED-CC-2($V',\ E',\ L$) {*use the algorithm from Figure 1 to solve the problem once the number of edges reduces to a sufficiently small number*}
22. ***parallel for*** each $v \in V$ ***do*** $L[v] \leftarrow L[L[v]]$ {*map the solution back to the current instance*}
23. ***endif***

Figure 2: Parallel connected components (CC) based on edge sampling.

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 ($1 \leq u, v \leq n$) denoting an undirected edge between vertices u and v. The edges will be sorted in nondecreasing order of the first vertex.
- **Output Format:** The first line of the output will contain an integer r giving the number of connected components in the input graph. This will be followed by the size of each of the r connected components, each on a separate line, ordered from the largest to the smallest. The size of a connected component is an integer giving the number of vertices in that component.

- **Sample Input/Output:** `/work/01905/rezaul/CSE638/HW2/samples` on Lonestar.
- **Test Input/Output (see Table 1):** `/work/01905/rezaul/CSE638/HW2/turn-in` on Lonestar.

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-friendster	Friendster online social network	65M	1.8B
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 `/work/01905/rezaul/CSE638/HW2/turn-in/` on Lonestar. If the name of the input file is `xxxxx-in.txt` in task 2(j), please name the output files as `xxxxx-2j-CC-1-out.txt`, `xxxxx-2j-CC-2-out.txt` and `xxxxx-2j-CC-3-out.txt` for Par-Randomized-CC, Par-Randomized-CC-2 and Par-Randomized-CC-3, respectively.
- Output files generated by Cilkview.

APPENDIX 3: Things to Remember

- **Please never run anything that takes more than a minute or uses multiple cores on TACC login nodes.** TACC policy strictly prohibits such usage. They reserve the right to suspend your account if you do so. 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. Do the same thing when you run Cilkview.
- Please make sure that speedup values for trial runs (or measured speedups) are included in the Cilkview plots you generate.