CSE-613: Parallel Programming, Spring 2015
Homework #2

By Aashray Arora (SBU ID: 109940382)

(NET ID: aaarora)

&

Ajay Lakshminarayanarao (SBU ID: 109898256)

(NETID: alakshminara)

1

(a) Prove that whp in n, Par Randomized-CC Perform $O((m+n)\log n)$ work and has $\Theta \log^3 n$ span

n - number of vertices
m - number of edges in input graph.

The alogrithm is given as following

Step Par-Randomized-CC (V,E,L)

1. $|E| = 0$ then return L.
2. array $C[1:n]$, $M[1:n]$, $S[1:|E|]$
3. parallel for $v \leftarrow 1$ to n do $C[v] \leftarrow$ Random {Head, Tail}

In line 3 we are randomly assigning a higher significance to a vertex in the graph based on whether we get a Head or Tail. This brings a random factor to our algorithm. This step takes $O(n)$ time serially & $O(\log n)$ time in parallel.

4. parallel for each $(u,v) \in E$ do
5. if $C[u]$ = Tail and $C[v]$ = Head then $L[u] \leftarrow L[v]$

For each of the edges we are checking whether the involved edges are one is special & other is not. So the time taken is a factor of both m & n logarithmically.

6. parallel for $i \leftarrow 1$ to $|E|$ do

7. if $L[E[i].u] \neq L[E[i].v]$ then $S[i] \leftarrow 1$ else $S[i] \leftarrow 0$

Even in the above steps we are going through all edges to remove intra group edges. Apart from isolated vertices all vertices are checked. As this is done in parallel it takes logarithmically for the vertices in V and for all Edges.

8. $S \leftarrow$ Par - Prefix - Sum $(S, +)$.

Par prefix sum takes the following complexities as seen. Work

$$T_1(n) = \begin{cases} \Theta(1) & \text{if } n=1 \\ T_1(\frac{n}{2}) + \Theta(n) & \text{otherwise.} \end{cases}$$

$$= \Theta(n).$$

Span:

$$T_\infty(n) = \begin{cases} \Theta(1), & \text{if } n=1 \\ T_\infty(\frac{n}{2}) + \Theta(1) \end{cases}$$

$$= \Theta(\log n)$$

9. array $F[1 : S[|E|]]$

10. parallel for $i \leftarrow 1$ to $|E|$ do.

11. if $L[E[i].u] \neq L[E[i].v]$ then

$F[S[i]] \leftarrow (L[E[i].u], L[E[i].v])$

for all edges we are copying the inter group edges only to F.

12. $M \leftarrow$ Par-Randomized-CC (n, F, L)

We are recursively calling the function, but this time with our graph reduced with respect to edges. This contracted graph goes through the steps 1-10 till $|F| = 0$.

We have to find the level of contraction observed ~~and~~ or by factor of which the graph contracts.

→ Let us take a vertex v in the graph set V.

We observed in line 3, that each vertex is assigned a head or a tail randomly.

We know that an non-isolated vertex $v \in V$ has atleast one vertex it connects to.

In line 5 we observe that one of them (vertices in edge) should be a head and other a tail for the vertex to be removed.

The probability by which this happens is, Probability of vertex v_1 being head X Probability of vertex v_2 being tail in $(v_1, v_2) \in E$.

$$P = \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}.$$

Thus this case happens with a probability of 1/4 and graph contracts by this factor.

13. parallel for each $(u, v) \in E$ do
14. if $v = L[u]$ then $M[u] \leftarrow M[v]$
15. return M.

Now as this algorithm is recursively called we need to find the depth of recursion.

At every call the edge contraction observed is 1/4 as seen above.

Therefore number of levels is $\log_4 n$. ie $\log_{2^2} n = \frac{1}{2} \log$

$\therefore$ $\underline{D = O(\log n)}$. And this holds w.h.p as observed previously.

Now we need to find the Work and Span from Step 1-~~12~~ & 13-15.

We observe that line 4-5 and line 6-7 take logarithmic complexity of all edges and check for all vertices in them. Therefore the Span for lines ~~1-11~~ is given by $\Theta(\log^2 n)$ As it is max for vertices when it is a fully connected graph. In that case for every node is connected to all other nodes. Now for all depths we know that T_∞ for the entire algorithm can be given by

$$T_\infty(n,m) = \Theta(D \log^2 n)$$
$$= O(\log n) \times \Theta(\log^2 n)$$
$$= O(\log^3 n).$$

As D was calculated with respect to high probability and similar probability applies to the steps 1-11 & 13-15.

The work for the steps 1-11 involves going through all the edges and going through vertices in them. We will miss only if it is a isolated edge. Our w.h.p is based on non isolated edges. As we see both E & V the complexity is $\Theta(m+n)$

the work is given by $T_1(n,m) = \Theta(D(n+m))$
$$= O(\log n)\ \Theta(n+m)$$
$$= O(\log n\ (n+m)).$$

TASK 1 (b): Modify FIND-ROOTS:

Find roots can be modified so that a node is never considered again by the parallel for loop once its s value stops changing. We propose two solutions:

① Use a set, remove elements from the set when the S value changes.

```
Find-Roots (n, P, S)
  set D;
  parallel-for v ← 1 to n do
      s(v) ← P(v)
      D[v] ← v

  while (D is not empty)
      parallel for ; v ← to D.size
          S(d(v)) ← s(s(d(v)))
          if (s(d[v]) == s(s(d(v))) then D.remove(v);
```

→ We have a set D, (containing all nodes initially) only once the s value stops changing, we delete the value from the set, so that it is never considered again.

② Find-Roots (n, P, s)

parallel-for $v \leftarrow 1$ to n do

$s(v) \leftarrow P(v)$

while $s(v) \neq s(s(v))$ do

$s(v) \leftarrow s(s(v))$

- here in each iteration (all running parallel) we keep updating the ~~parent~~ s until it stops changing, hence it is never considered again.

Analysis: ~~for~~ algorithm ①

Even though we do not touch the nodes for which the s pointer is not changing

- In each iteration of the while loop (in ①) we will remove nodes that were at height 1 to its root, then in next iteration at height 2 to its root, and so on. So we will still take atleast h iterations to make the set empty.
- The distance bet v and s(v) doubles after each iteration, until $s(s(v))$ is the root containing v.
- Thus the no. of iterations is still $\log h$. (assuming the parallel for loop takes $\Theta(1)$ time).

Thus:

work: $T_1(n) = O(h \log h)$

in the worse case $h \approx n$.

$$\therefore T_1(n) = O(n \log h)$$

Span: $T_\infty(n) = \Theta(\log h)$

These algorithms do not improve the asymptopic performace of the original algorithm (FIND-ROOTS).

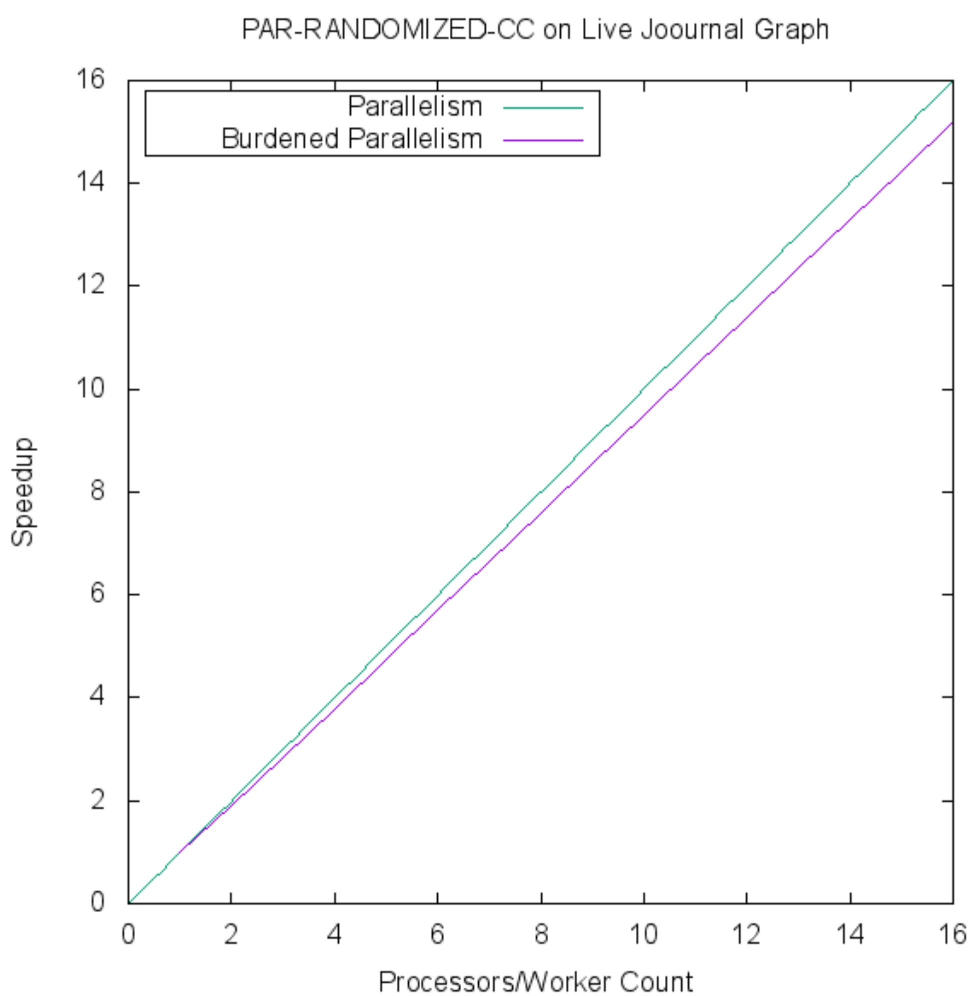
TASK 1 (c):

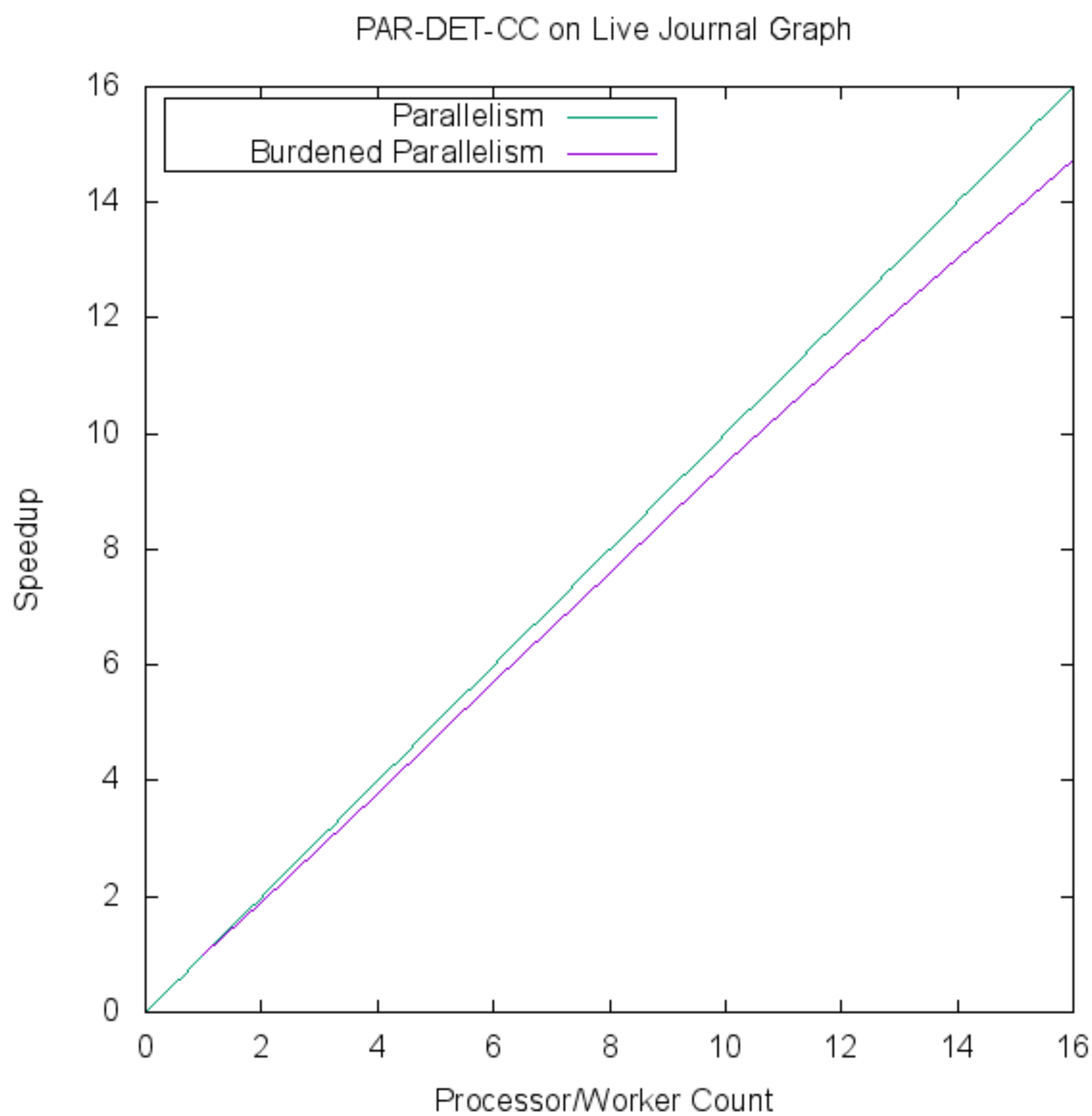
The modified find-roots algorithm still has the same asymptopic complexity. Although it is a better PAR-DETERMINISTIC-algorithm to use. It does not change CC's running time (asymptopically).

TASK 1: D

	par-randomized-cc (time in seconds)	par-deterministic-cc (time in seconds)
as-skitter	10.91	0.52
ca-AstroPh	0.16	0.17
com-amazon	1.97	0.07
com-dblp	1.855	0.084
com-friendster	Segfault/too large/Error	Segfault/too large/Error
com-lj	30.95	1.7
com-orkut	20.92	1.084
roadNet-CA	13.56	0.27
roadNet-PA	6.63	0.14
roadNet-TX	7.72	0.19

TASK 1:E




PAR-DET-CC on Live Journal Graph
Parallelism
Burdened Parallelism
Speedup
Processor/Worker Count
0
2
4
6
8
10
12
14
16