CSE 613: Parallel Programming

Homework 2

Aggarwal, Srikant(109890795)
Singh, Bharat (109324830)

Task 1:

a. Let X denote the poisson variable such that:

X = { 0 if the edge (u, v) is a {Head, Tail} pair
1 otherwise

Then, E[X] = P(X=1) = 3/4
Now, using Chernoff Bound 2, say $\delta = 1/4$ then

$Pr[X \geq (1+\delta)\mu] \leq e^{\wedge(-\mu\delta 2/3)}$

$\Rightarrow Pr[X \geq 5/4*3/4] \leq e^{\wedge(-3/4 * 1/4 * 1/4 * 1/3)}$

$\Rightarrow Pr[X \geq 15/16] \leq e^{\wedge(-1/64)}$

Therefore, w.h.p. the number of edges reduces by a constant fraction, 1/16 with a probability $1-e^{\wedge(-1/64)}$, at each iteration of the loop. Hence, w.h.p in log(n) levels the number of edges reduce to 1.

b. As the probability that a pair is assigned (Head, Tail) is 1/4 , so at each step of PAR-RANDOMIZED-CC, the edges decreases by a factor of ¼ and hence if the number of edges is E at any step the number of edges at next step would be ¾ E.

If total height is k then k = log(m) = log(n)

Calculating work:

Step 1-2 : O(1)
Step 3 : O(n)
Step 4-5 : O(m)
Step 6-7 : O(m)
Step 8 : O(n)
Step 9 : O(1)
Step 10-11 : O(m)
Step 12 : O(1)

Step 13-14 : O(m)
Step 15 : O(1)

So, total work = (O(n+m)*D) = O((n+m)log(n))

Calculating span:

Step 1-2 : O(1)
Step 3 : O(log(n))
Step 4-5 : O(log(m)) = O(log(n))
Step 6-7 : O(log(m)) = O(log(n))
Step 8 : $O(\log^2(n))$
Step 9 : O(1)
Step 10-11 : O(log(m)) = O(log(n))
Step 12 : O(1)
Step 13-14 : O(log(m)) = O(log(n))
Step 15 : O(1)

So, total span = $O((\log^2(n) + \log(n))*D) = O(\log^2(n)*D) = O(\log^3(n))$

Original:

Find-Roots (, ,)

```
parallel for  v ← 1 to n do
        S(v) ← P(v)
flag ← true
while flag = true do
        flag ← false
        parallel for v ← 1 to n do
                S(v) ← S(S(v))
                if S(v)  ≠ S(S(v)) then
                        flag ← true
```

Modified:

Find-Roots (, ,)

```
1.  array F[1:n]
2.  parallel for  v ← 1 to n do
3.          S(v) ← P(v)
4.          F(v) ← v
5.  flag ← true
6.  max ← n
7.  while flag = true do
8.          flag ← false
```

```
9.          array E[1:max]
10.         parallel for index ← 1 to max do
11.               v ← F[index]
12.               S(v) ← S(S(v))
13.               if S(v)  ≠ S(S(v)) then
14.                     flag ← true
15.                     E[index] ← 1
16.         E ← Par-Prefix-Sum(E, +)
17.         parallel for index ← 1 to max do
18.               prev ← 0;
19.               if (index ≠ 1)
20.                     prev ← E[index-1]
21.               if (E[index] - prev ≠ 0)
22.                     F[E[index]] ← F[index];
23.         max ← E[max]
```

Work for the modified Algorithm:
Step 1 : O(1)
Step 2-4 : O(n)
Step 5-6 : O(1)

Step 7 : O(1)
Step 8-9 : O(1)
Step 10-15 : O(max)
Step 16 : O(max)
Step 17-22 : O(max)
Step 23 : O(1)

Step 1-6 gets executed only once so work done in Step 1-6 = O(1) + O(n) + O(1) = O(n)

Now Step 7-23 gets executed till S changes for even one of the vertices. If the height of tree is h then the steps gets executed at O(log(h)) number of times.

Work done from Step 7-23 = O(1) + O(1) + O(max) + O(max) + O(max) + O(1) = O(max) for each iteration. Now consider the case of a complete perfectly balanced binary tree i.e. at each level the node splits into two child. Then in worst case, the $\log(h)^{th}$ iteration would merge nodes at leaf + some more nodes > n/2. So max will decrease from n to a number > n/2 = O(n).

Hence the loop in line 10 and 17 still gets executed O(n) times for each iteration. So, the asymptotic complexity doesn't change for the FIND-ROOTS algorithm.

c. Since, the asymptotic complexity of our algorithm doesn't change on doing the changes. Also, neither the work, nor the span changes after the modifications. So, the performance of PAR-DETERMINISTIC-CC remains same as before.

d. Parallel Randomized CC - We have added following improvements to the algorithm to make it faster:

- We have modified the hooking mechanism to add double hook (Based on Advanced Algorithm course Spring 2013).
- We reduce the number of vertices being passed to the next recursive level.

Parallel Deterministic CC - We have added following improvements to the algorithm to make it faster:

- We have optimized the hooking logic, here we are hooking from both direction. Its runtime performance is observed to be better than one directional hooking.
- We are using optimized Par-Find-Roots as mentioned in part b.

Running time comparison for Par-Randomized-CC and Par-Deterministic-CC, using all cores on given input set for each input file.

Input graph	Par-Randomized-CC (msecs)	Par-Deterministic-CC (msec)
ca-AstroPh-in.txt	170569	35553
com-amazon-in.txt	330817	77785
com-dblp-in.txt	518914	111768
roadNet-PA-in.txt	568989	394124
roadNet-TX-in.txt	690228	325515
roadNet-CA-in.txt	1002943	379849
as-skitter-in.txt	4023802	800234
com-lj-in.txt	26005711	2320132
com-orkut-in.txt	99708485	3059457
com-friendster-in.txt	Timeout	Timeout

Note: com-friendster job was failing with bad_alloc(), system had around 10G of free memory, but our input file itself is 30G. Par-Deterministic-CC performs better than Par-Randomized-CC.

e. Strong scalability plot using cilkview for parallel Deterministic and Random connected

component implementations from part (d) using the Live Journal graph as input. Cilkview job with multiple trials were failing, so we have ran it with default cilkview configuration, i.e. single trial, 16 workers.

Command used:

/home1/03442/tg827409/cilkutil/bin/cilkview Par_Deterministic_CC.out < /work/01905/rezaul/CSE613/HW2/turn-in/com-lj-in.txt >> scale-com-lj-det-CC-out.txt

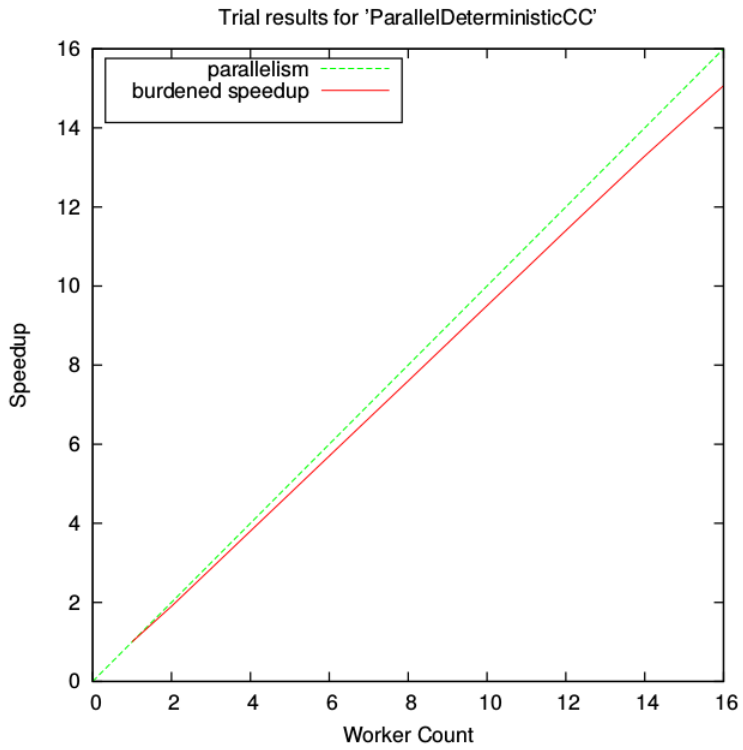

Figure 1: Scalability plot for parallel deterministic CC

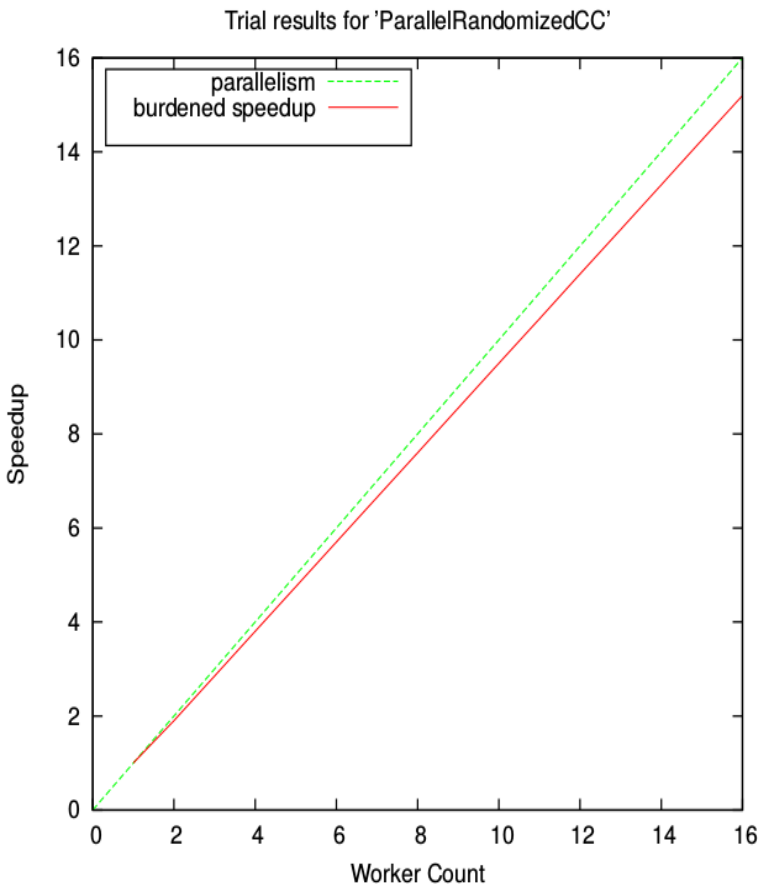

Figure 2: Scalability plot for parallel randomized CC