Parallel Pagerank Estimator
CPT_S 411: Introduction to Parallel Computing

Reet Barik (WSU ID: 11630142)

December 2020

1 Introduction

The journey that has made Google into the conglomerate that it is today, so much so that it's products have become an intrinsic part of our lifestyle started with their first significant product: the search. The underlying algorithm was effective and set Google apart from the competitors. The success of that piece of innovation can be grasped by the fact that the word 'Google' has since been used as a verb to mean 'search the internet'. Named after Larry Page, one of the co-founders of Google, the 'Pagrank' algorithm was used by Google's web search engine to rank web pages in the search engine results. The underlying philosophy was that websites which are important will be linked to a larger number of websites as compared to relatively unimportant ones. There is also a transitive nature to this philosophy which can be expressed as assigning proportionate importance to a website based on how many other import websites point to it.
One approximate solution of assigning a rank to websites as a reflection of their importance is to simply count the number of times they are visited. Thus, the basic Pagerank solution involves having a 'random surfer' start from a webpage and randomly click through links available on it and essentially perform a random walk through the graph described by the inter-linked webpages. Having performed enough of these random walks and keeping track of the number of times each node (webpage) is visited will help estimating the importance of those webpages.
The random walks described above are supposed to be mirroring the behavior of an internet user. Hence, they are parameterized by the length of the walk and damping factor (the probability with which an user might want to jump to a webpage that is not linked to by the current webpage they are on). It is easy to see that these walks are independent of each other and hence, one can leverage parallelization to execute walks with longer lengths so as to arrive at a more accurate solution.

2 Problem Statement

Given a directed graph $G(V,E)$ where $v \in V$ are nodes representing webpages and $e \in E$ are edges representing all (i,j) node pairs such that there is an edge from node i to j, conduct random walks of length k with a damping ratio of d and rank the nodes based on the number of times they were visited.

In terms of a blackbox: **Input**: Graph G, Walk-Length k, Damping ratio d. **Output**: List of all nodes sorted based on their pageranks.

3 Key Challenges in Parallelization

Some of the challenges that had to be overcome while coming up with a parallel version of the algorithm are as follows:

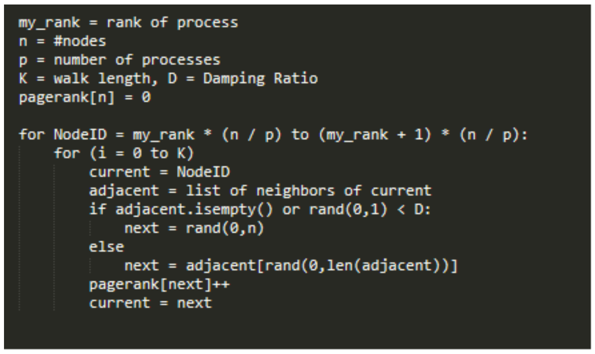
- Storage: The graph needs to be stored efficiently since the complexity associated to storing adjacency matrices of real-world graphs is too high. Also, the power law degree distribution in real-world graphs [2] implies that the corresponding adjacency matrices would be very sparse leading to memory wastage. Hence, there was a need to store the input graphs using more efficient data structures.
- Synchronization across walks: The output required at the end is essentially the total number of times each node is visited. Hence, there was a need to put a mechanism in place so that the variables responsible for keeping track of the number of visits to each node could be updated one thread at a time (atomic) to prevent the values getting corrupted by thread-unsafe handling.

4 Proposed Approach(es)

With the objective clear, there were two ways to parallelize:

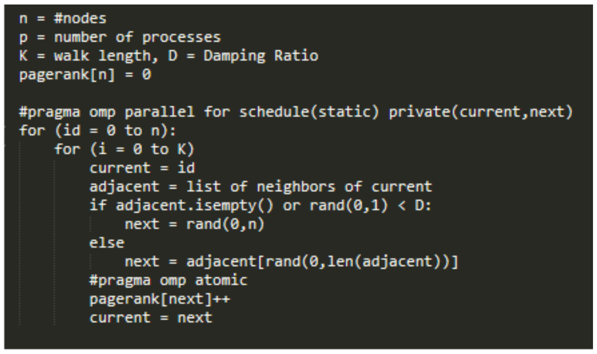
- Distributed memory version using MPI: In this setting, every process needs to perform a chunk of the total number of random walks. If the graph was partitioned and stored in a distributed fashion across processes, the communication cost might be too high (jump from one process to another if the next node is in a different process). Hence, it makes sense to store a copy of the whole graph on all the processes and accumulate the results in the end (this might be expensive in terms of storage).
- Multithreaded version using OpenMP: In this setting, there is no need to duplicate the graph since all the threads have access to the same memory. There is a need to store the graph efficiently. **Adjacency list** was the chosen data structure because it gives easy access to every node's neighbors. As in the other version, every thread performs a chunk of the total number of random walks. Hence, the block decomposition of tasks calls for **Static Scheduling**. The updates to the output array needs to be done in a thread safe fashion, hence those are **Atomic** updates.

As part of this project, only the multi-threaded version has been implemented. For the multi-threaded version:
1. **Space Complexity**: Space required to store the adjacency list, $O(m+n)$.
2. **Time Complexity**: Total n random walks of length k each spread across p threads, $O(kn/p)$.
The pseudocodes to the two approaches are shown below:



```
my_rank = rank of process
n = #nodes
p = number of processes
K = walk length, D = Damping Ratio
pagerank[n] = 0

for NodeID = my_rank * (n / p) to (my_rank + 1) * (n / p):
    for (i = 0 to K)
        current = NodeID
        adjacent = list of neighbors of current
        if adjacent.isempty() or rand(0,1) < D:
            next = rand(0,n)
        else
            next = adjacent[rand(0,len(adjacent))]
        pagerank[next]++
        current = next
```

Distributed memory



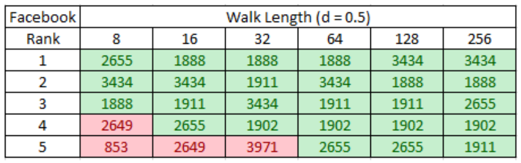
```
n = #nodes
p = number of processes
K = walk length, D = Damping Ratio
pagerank[n] = 0

#pragma omp parallel for schedule(static) private(current,next)
for (id = 0 to n):
    for (i = 0 to K)
        current = id
        adjacent = list of neighbors of current
        if adjacent.isempty() or rand(0,1) < D:
            next = rand(0,n)
        else
            next = adjacent[rand(0,len(adjacent))]
        #pragma omp atomic
        pagerank[next]++
        current = next
```

Multi-threaded

5 Experimental Results and Discussion

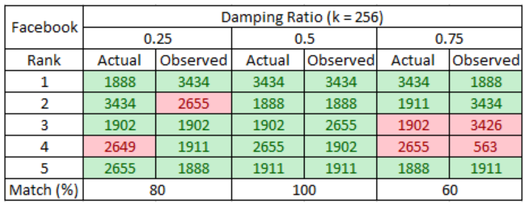
The values of walk-length (k) that were experimented with are: $8, 16, 32, 64, 128, 256$ (doubling at each step). The decision to cap the value of k at 256 was taken because, for that value of k, the runtime on the largest dataset (Google) was $\sim$4 minutes. The values considered were enough to give sufficient data points so as to infer the trend. Three types of experiments were carried out as part of this project. They are described as follows:

- **Stability of output**: The expectation was that due to the stochastic nature of the algorithm, there is no correct answer but there will be general consensus of the top 5 across results for different values of k, with the top 5 stabilizing as k is increased. For this, the values of k was varied keeping $d = 0.5$. The result for the Facebook dataset is shown in Figure 1.
  As can be observed, the contents of the set of nodes comprising the top 5 in terms of having the highest pageranks stabilizes as k increases (all green for $k = 64, 128, 256$) as compared to there being 1-2 stray entries (shown in red) while the value of k is low.



Facebook	Walk Length (d = 0.5)					
Rank	8	16	32	64	128	256
1	2655	1888	1888	1888	3434	3434
2	3434	3434	1911	3434	1888	1888
3	1888	1911	3434	1911	1911	2655
4	2649	2655	1902	1902	1902	1902
5	853	2649	3971	2655	2655	1911

Figure 1



Facebook	Damping Ratio (k = 256)					
	0.25		0.5		0.75	
Rank	Actual	Observed	Actual	Observed	Actual	Observed
1	1888	3434	3434	3434	3434	1888
2	3434	2655	1888	1888	1911	3434
3	1902	1902	1902	2655	1902	3426
4	2649	1911	2655	1902	2655	563
5	2655	1888	1911	1911	1888	1911
Match (%)	80		100		60	

Figure 2

- **Correctness (quality of output)**: To judge the quality of output or the correctness of the implementation, the damping ratio (d) was varied and each output was compared against the output of Networkx's serial Pagerank implementation [1] based on [3] and [4]. The results for the Facebook dataset are shown in Figure 2.
  It is observed that for $d = 0.25, 0.5, 0.75$, the match between top 5 nodes according to this project's implementation and that of the Networkx library is 80%, 100%, and 60% respectively.

- **Parallel experiments**: To study how the implementation scales in the parallel setting, the following experiments were conducted:

– **Parallel Runtime**: The results are shown in Figure 3 below. A linear decrease in runtime with increasing number of threads is observed as per expectation. The curves are almost parallel to each other and equally spaced apart which means if number of threads are fixed, the runtime increases linearly w.r.t. ‘k’.

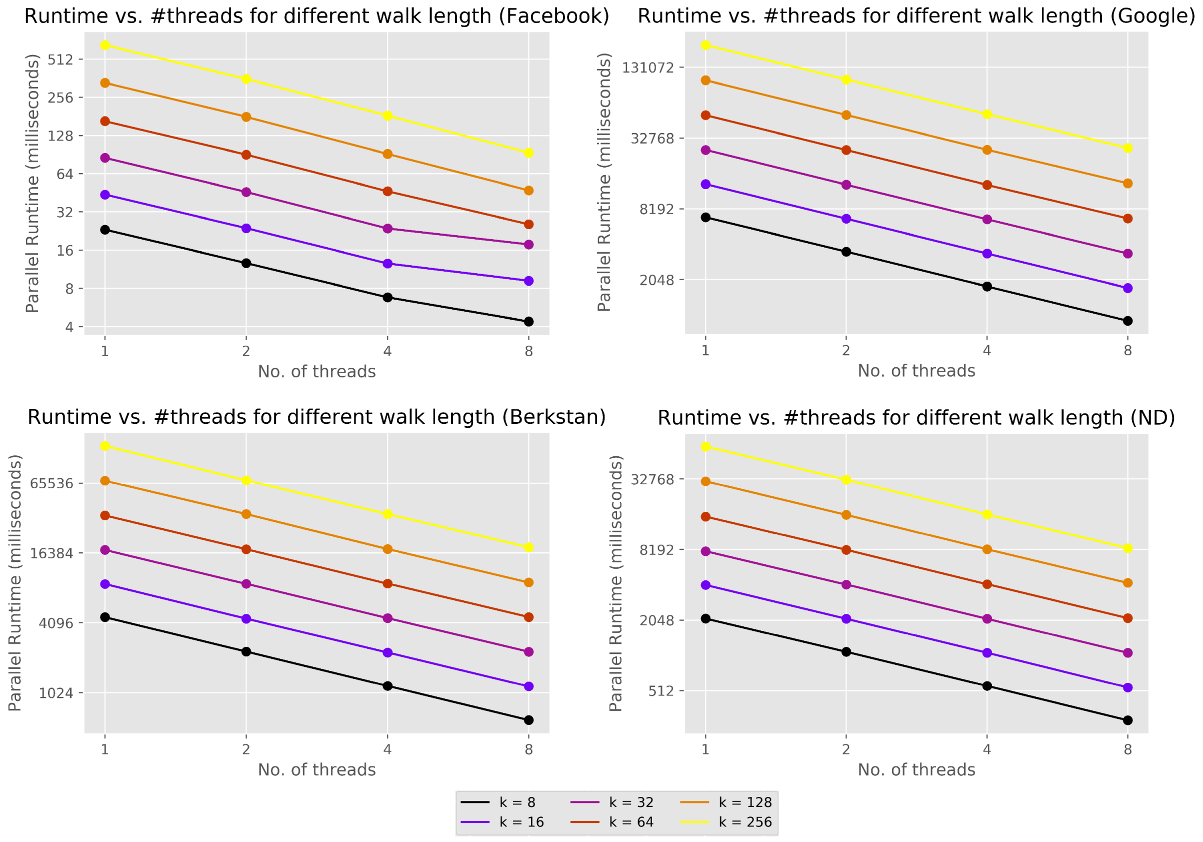
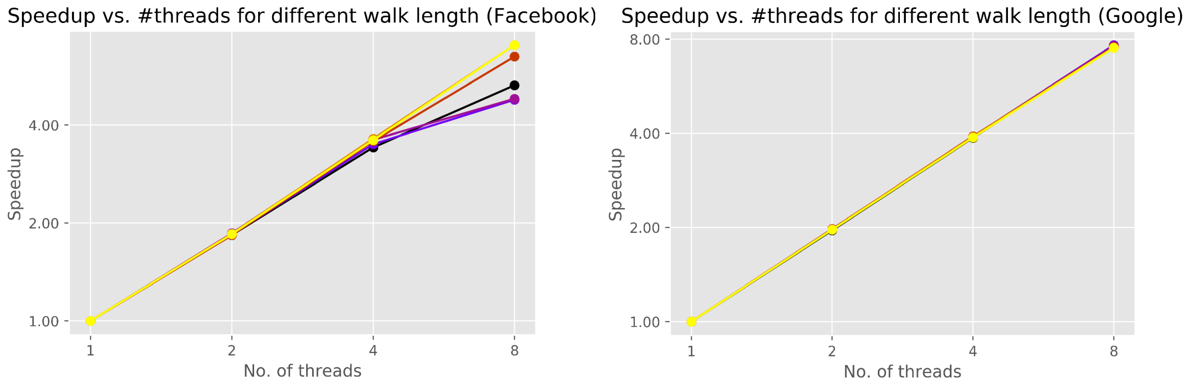


Figure 3

– **Speedup**: The results are shown in Figure 4 below. The speedup achieved is linear as can be observed. The Speedup curve for Facebook (smallest) dataset starts to plateau (for smaller values of *k* and higher number of threads) echoing the sentiment that smaller problems show smaller benefits from parallelization.



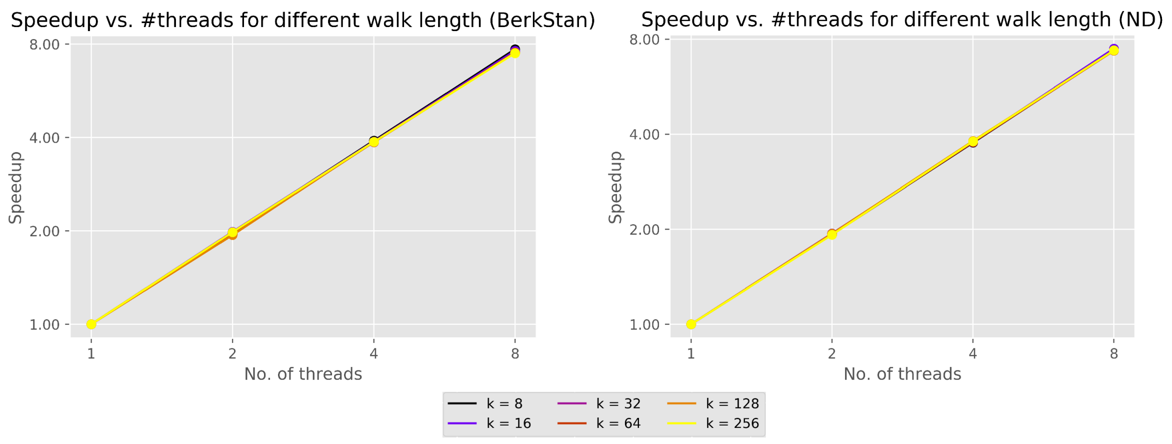


Figure 4

- **Efficiency**: The results are shown in Figure 5 below. As expected, there is a drop in efficiency for all datasets. The drop is largest for Facebook (to $\sim 60\%$) further highlighting how parallelizing gives maximum benefits when the problem to be solved is large.

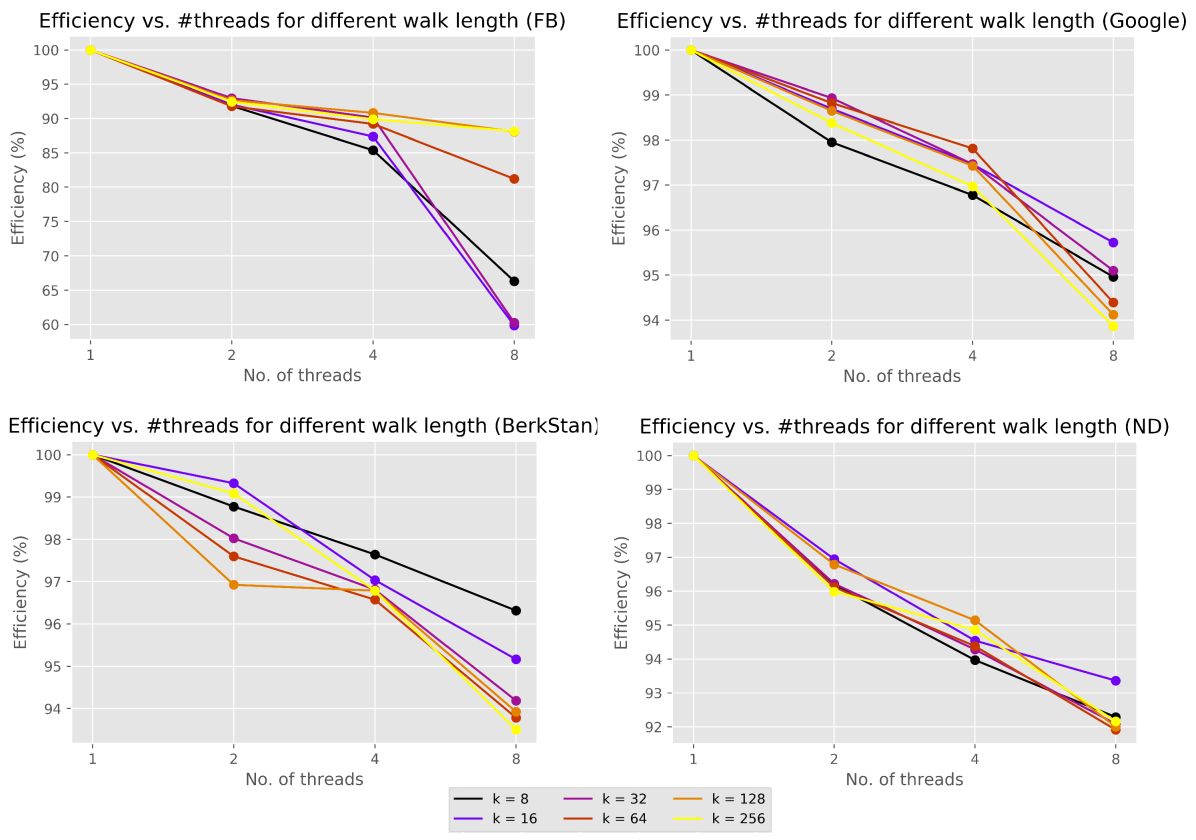


Figure 5

Overall, it can be concluded that the implementation is stable and correct (as correct as it can be, given the project specifications). It also shows the expected scaling for inputs of varying sizes. [Note: The raw figures for runtime, speedup, and efficiency were not included in report due to lack of space. They have been included as a separate worksheet in the submitted folder.]

References

[1] Networkx link-analysis pagerank. https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.link_analysis.pagerank_alg.pagerank.html.

[2] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[3] Amy N Langville and Carl D Meyer. A survey of eigenvector methods for web information retrieval. *SIAM review*, 47(1):135–161, 2005.

[4] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.