PageRank Computation

On Implementation Of An Efficient
Algorithm To Find PageRank

Neelesh Bhakt (2018201022) | Nishant Goyal (2018201038) | Souparna Das (2018201010)

[IPSC] [Spring 2019]

Introduction

PageRank (**PR**) is an algorithm used by Google Search to rank webpages in their search engine results. PageRank was named after Larry Page, one of the founders of Google. PageRank is a way of measuring the importance of website pages.

According to Google:

“PageRank works by counting the number and quality of links to a page to determine a rough estimate of how important the website is. The underlying assumption is that more important websites are likely to receive more links from other websites.”

Currently, PageRank is not the only algorithm used by Google to order search results, but it is the first algorithm that was used by the company, and it is the best known.

The PageRank computation proceeds in iterations and in each iteration the current PageRank of each node is updated by using the PageRank of its incoming neighbours. The computation stops when the PageRank values at all nodes change only by a small value across successive iterations. In each iteration, along each directed edge e = (u, v), a fraction of the PageRank value of u is transferred to the new PageRank value of v. This computation offers opportunities for optimization where certain edges of the graph can be identified as redundant for one or more iterations. These optimizations can be achieved by a deeper understanding of the structure of popular real-world graph classes such as web graphs, road networks, and citation networks. There have been a number of works on parallel algorithms to compute the PageRank of nodes in a graph. However, some of the techniques are directed only towards web graphs and may not translate to other real-world graphs. With PageRank being used for other purposes beyond its original intent, it is important to have an efficient algorithm for PageRank computation applicable for several classes of real-world graphs.

For our project we have worked on implementation of some of the important aspects of the Research Paper (from now on will be referred as “paper”) and analysing and comparing the results.

Motivation

As per the course we took “IPSC” (Introduction to Parallel Scientific Computing), our general way of optimizing a program has developed and it became a fun to think through ways of parallelizing programs and improve their performance significantly. The paper we have referred has applications not only for webpages, but also the algorithm can be applied on real graphs to find the rank of the nodes. If the graph size is big and the representation is often sparse, then the algorithm performs excellently which the real-world demands. One of the key motivations of our work is to understand the structural properties of real-world graphs and their impact on algorithms. Specific to the computation of PageRank, we seek properties that can allow us to reduce the number of computations that are invoked for each node of the graph. In this direction, we first note that real world graphs being sparse in nature tend to have several strongly connected components (SCCs). Real-world graphs have a large number of SCCs with a small size, and very few SCCs of a large size. This property indicates that a decomposition based on SCCs can be useful in parallel algorithms for PageRank computation.

Approaches Used for Computing PageRank

- ## Iterative Method (Baseline Algorithm):

Let $G = (V, E)$ be a directed graph with *n* = | *V* | vertices and *m* = | *E* | edges. Let *IN(u)* denote the set of nodes that have an incoming edge to u. Let $outdeg(u)$ denote the number of edges from u to other nodes. The PageRank of a node u, denoted $pr(u)$, is then given as follows:

$$pr(u) = \sum contribution(v \to u) + \frac{d}{n}$$

Here, d is called the damping constant and has its genesis in the manner in which PageRank is usually interpreted. The PageRank values represent a probability distribution where the PageRank of a node denotes the probability of a random walk to visit that node. The damping constant d can be interpreted as the probability that the random walk stays at the same node in the next step. The value of d is taken to be 0.15 usually. The quantity $contribution(v \to u)$ is defined as follows:

$$contribution(v \to u) = (1 - d)\frac{pr(v)}{outdeg(v)}$$

One can also combine the above two equations to arrive at the following simplified equation:

$$pr(u) = (1 - d) \sum_{v \in IN(u)} \frac{pr(v)}{outdeg(v)} + d/n$$

However, this formula is cyclic in nature as two nodes can make contributions to each other if they are a part of directed cycle. One way two resolve this dependency is to apply the formula from last equation iteratively over all the nodes until the PageRank converge. We say that the PageRank values of nodes have converged when there is very little change in their values across an iteration. This can be measured by a function such as the maximum difference of PageRank values or the total difference of PageRank values across an iteration.

Parallel Baseline Algorithm:

Algorithm 1 Base Pagerank(G)

```
1: procedure MAIN(Graph G = (V, E), Array outdeg )
2:     pr=Compute((V, E), outdeg)
3:     return pr
4: end procedure
5:
6: procedure COMPUTE(Graph G, Array outdeg )
7:     error = ∞
8:     for all u ∈ V do
9:         prev(u) = d/n
10:    end for
11:    while error > threshold1 do
12:        for all u ∈ V in parallel do
13:            pr(u) = d/n
14:            for all v ∈ V such that (v, u) ∈ E do
15:                pr(u) = pr(u) + prev(v)/outdeg(v) * (1 − d)
16:            end for
17:        end for
18:        for all u ∈ V do
19:            error = max(error, abs(prev[u] − pr[u]))
20:            prev(u) = pr(u)
21:        end for
22:    end while
23:    return pr
24: end procedure
```

In Algorithm 1, initialize the error as ∞ and the initial PageRank values of all nodes to d/n . For reasons of efficiency of parallel computation of pagerank, in Algorithm 1, for each node u, the new PageRank value of *u* is computed as the sum of contributions from the incoming neighbours of u. The variable *Threshold1* is a constant that reflects the accuracy of the PageRank needed by an application. Lines 12–17 iterate over all the nodes in parallel. Each node updates its PageRank based on the contributions of its incoming edges. Lines 18–21 calculate the error function as the L1 norm of the change in the PageRank values of nodes across one iteration. The variable error is used to decide whether to proceed for the next iteration or declare convergence.

- Optimized Version of the baseline algorithm

In real life the graphs formed by the in-links and out-links of webpages are generally sparse. And number of webpages are typically huge in numbers. The paper increases efficiency in the above algorithm by using the strongly connected components in the graph. The procedure goes as follows:

- Find Strongly connected components:

After finding the strongly connected components of the graph, the graph looks like:

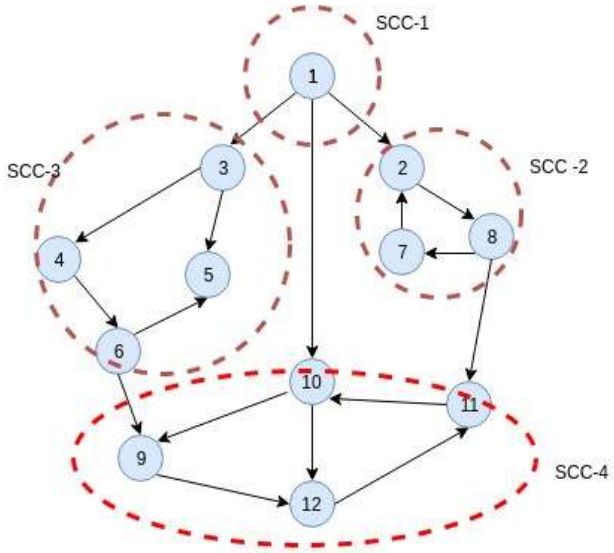


Figure 2:SCC Graph

Recall from iterative computation of PageRank values of nodes depend cyclically on one another. It is also to be observed that if the PageRank value of all incoming neighbours of a node v converge, then the PageRank value of v will also converge by the next iteration. This observation can be extended to a decomposition of a directed graph into its strongly connected components.

In Figure 2, The strongly connected components (SCCs) of the graph are denoted by SCC-1, SCC-2, SCC-3 and SCC-4. Each SCC is shown in a dotted red cluster.

- Topological Ordering after finding SCCs:

After Ordering the SCCs according to Topological order what we get is as follows:

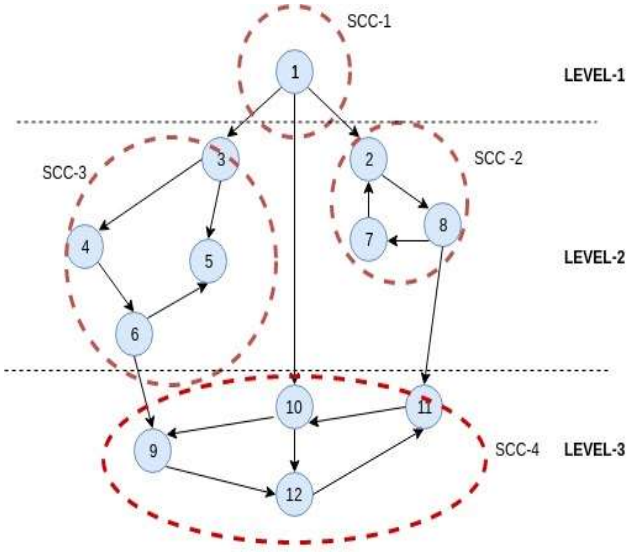


Figure 3:Level wise Representation

In Figure 3, After doing topological ordering we get the level wise representation of the graph. In the level wise representation, in each level, more than one SCC can be present. The more SCCs present in a level, the better performance is expected as all the SCCs in a level can be parallelized. In the diagram, SCC-1 contains node 1 in level-1, SCC-2 contains nodes 2, 7 and 8, SCC-3 contains nodes 3, 4, 5 and 7 in level-2 and SCC-4 contains nodes 9, 10, 11 and 12 in level-3.

Parallelization of the Algorithm:

Now having done SCCs and Topological Sorting, we can parallelize the procedure as follows:

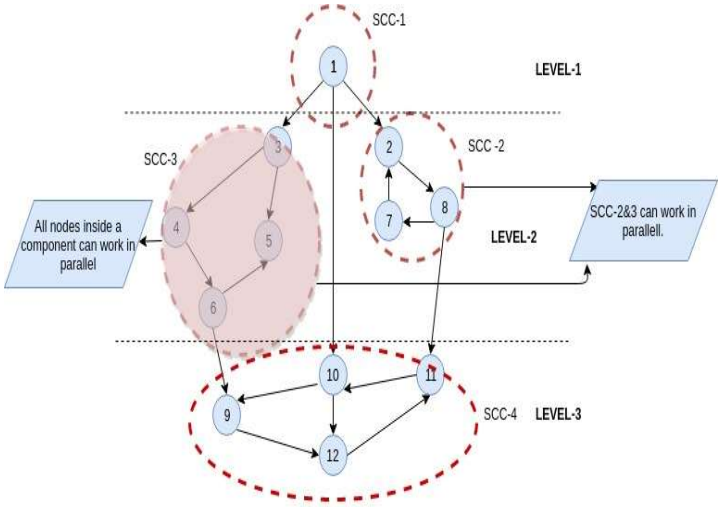


Figure-4: Parallelism in the optimized code

In Figure 4, in each of level-1, 2 and 3 there can be one or more SCCs. As there are no dependencies between two or more SCCs within a level, the SCCs can work in parallel. At first we compute level-1, then level-2 and so on. Within a SCC, PageRank of all the nodes can be parallelized using the iterative parallel method which have been discussed earlier.

So, the above procedure is the optimized version of the normal iterative parallel method. This method boosts the performance in real world because in real world, the graphs are often sparse and number of nodes (webpages) are usually huge in numbers. Next we will show the results we got before and after applying the optimized version on same graph on our implementation.

Analysing the performance

The dataset used for analysing the performance is provided in the references section below.

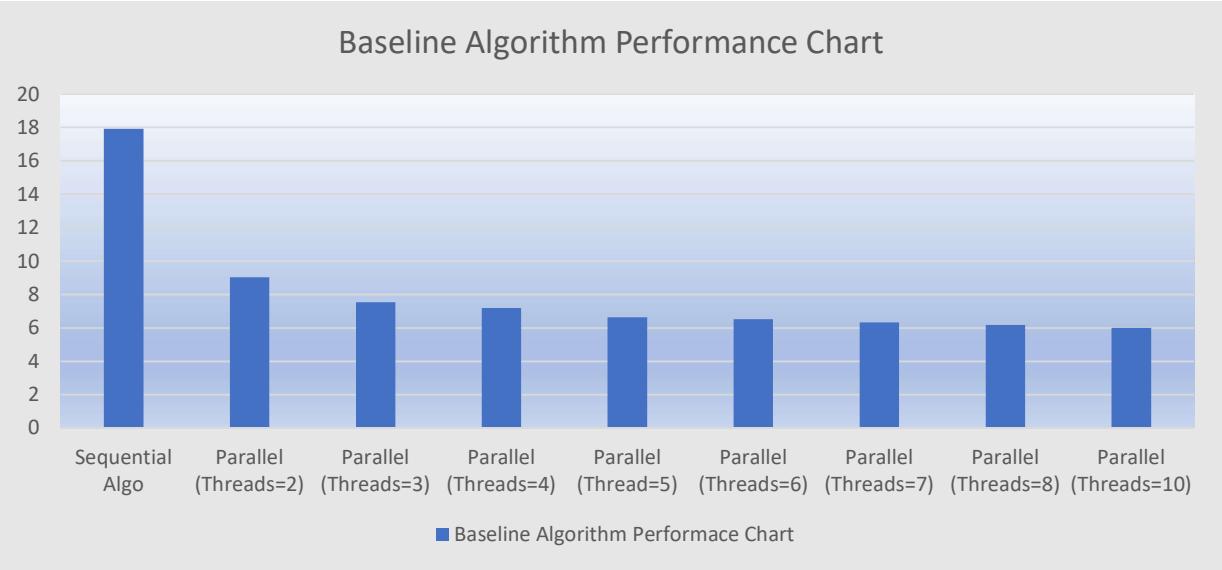
Performance of the Baseline model:

The below table shows the performance of the baseline model on a dataset contains 281903 nodes.

Baseline Sequential model is the simplest model with no parallelization done. Baseline parallel are parallel implementation of the baseline sequential model with different number of threads.

Algorithm	Threads	Time (secs)
Baseline Sequential	___	17.9233
Baseline Parallel	2	9.02756
Baseline Parallel	3	7.53381
Baseline Parallel	4	7.17177
Baseline Parallel	5	6.64329
Baseline Parallel	6	6.5151
Baseline Parallel	7	6.37422
Baseline Parallel	8	6.1716
Baseline Parallel	10	6.00551

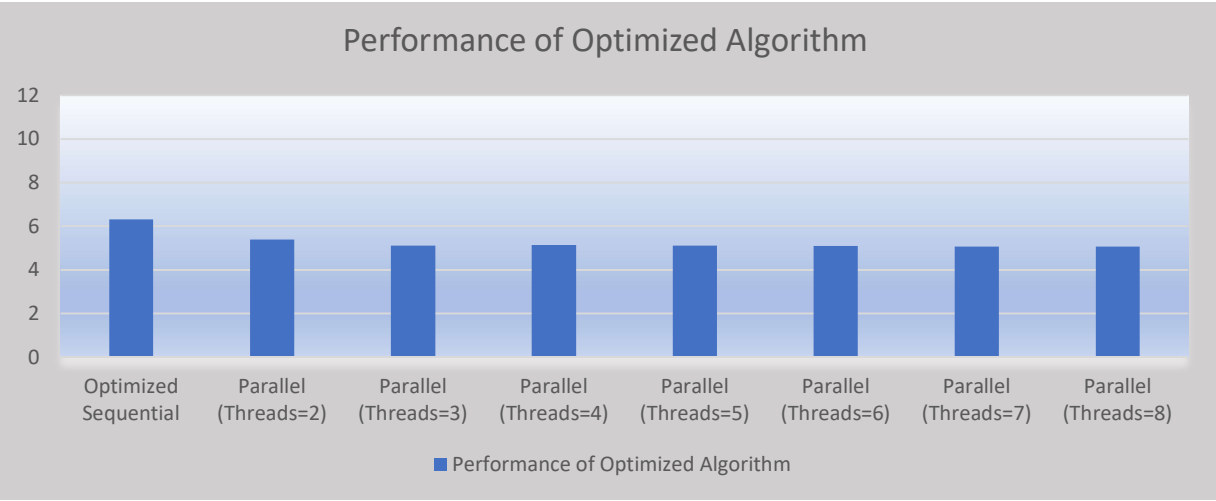
Here is the plot for the performance of the baseline models. The y-axis refers to time in seconds and the x-axis refers to the algorithm that has been used.



Performance of optimized algorithm:

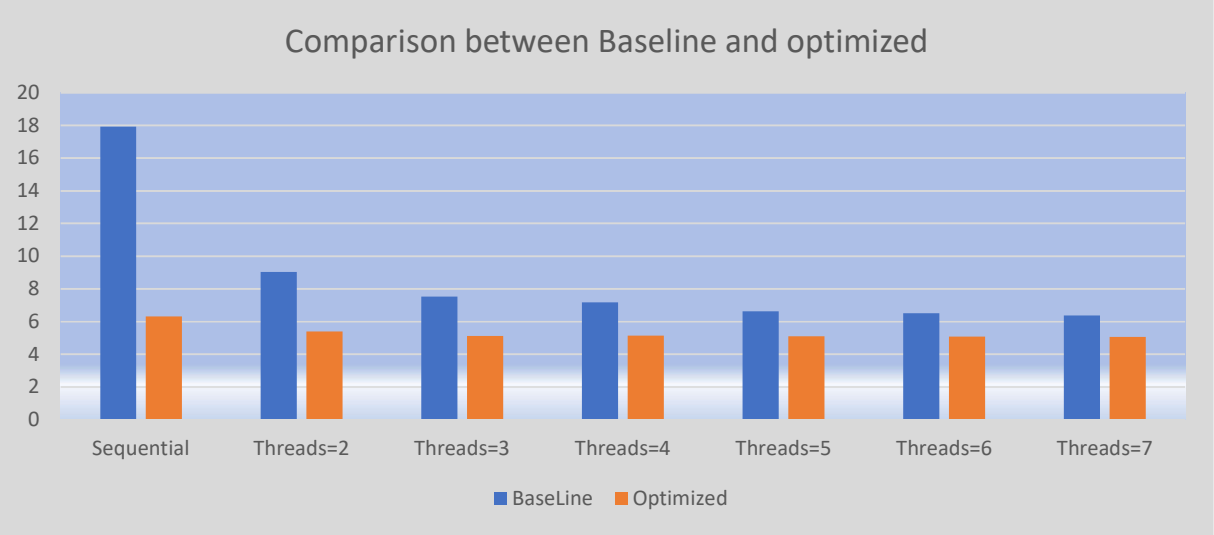
Algorithm	Threads	Time(secs)
Optimized Sequential	___	6.32042
Optimized Parallel	2	5.39614
Optimized Parallel	3	5.12399
Optimized Parallel	4	5.14017
Optimized Parallel	5	5.10191
Optimized Parallel	6	5.08657
Optimized Parallel	7	5.06873
Optimized Parallel	8	5.06655

The plot for the performance of optimized algorithm:



Comparison between Baseline and optimized Algorithm

The following chart represents the comparison between optimized and the baseline algorithm:



Conclusion

From the Baseline Sequential model to the Optimized parallel model, the performance of computing the PageRank of the webpages turned out to be far more beneficial in the case of the optimized model.

The baseline sequential model took 17.9233 seconds to compute PageRank of 281903 number of pages. Then we parallelized the sequential model and with high number of threads got a better performance.

The optimized model, without even parallelized, performed better than the baseline parallel model with lower number of threads. As we parallelized the PageRank computation of optimized algorithm by parallelization in each of the levels and further parallelizing each of the strongly connected components. With only 2 threads the optimized algorithm performed better than the baseline parallel algorithm.

References

1. http://cstar.iiit.ac.in/~kkishore/pagerank.pdf (Research Paper)
2. https://en.wikipedia.org/wiki/PageRank
3. https://en.wikipedia.org/wiki/Strongly_connected_component
4. https://en.wikipedia.org/wiki/Topological_sorting
5. https://snap.stanford.edu/data/web-Stanford.html (Dataset)