Benjamin Wang (ID: 1179478)
Ojas Malwankar (ID: 1610494)
COEN 319 Parallel Computing
Dr. Anastasiu,
12/7/21

Modified PageRank Algorithm Parallelization Report

1. Introduction

PageRank is the famous equation on which Google, now known as Alphabet, is founded upon, developed by Google co-founders Larry Page and Sergey Brin. At the time, PageRank revolutionized the web search field by returning actually relevant results to user queries. The fundamental intuition of the algorithm itself is that “A page is important if many important pages exclusively link to it”, encapsulating the quantity, quality, and exclusivity of webpage links in one algorithm. In addition to ranking web pages' importance, PageRank can be applied to any directed graph to rank importance, for example you can find important persons on Twitter if you generate a graph from the Twitter follows.

Our objective is to implement the modified PageRank algorithm, and get the PageRank score for each page or node in the graph without running into dead-ends, so that we can parallelize the modified pageRank algorithm for this group project. The PageRank score found can be used to rank the pages/nodes if one so chooses, though for our case we simply want the PageRank algorithm to converge.

2. System Design & Implementation details

PageRank computes the importance of a node in a directed graph, where each node can have edges that point to another node, with the incoming-edges referred to as in-links and outgoing edges referred to as out-links. PageRank itself is defined as:

$$PR(i) \;=\; sum(PR(j) \,/\, N_j), \text{ for j pointing to i}$$

Where for each node *i*, only its in-links are considered. Since each node, represented above as *i* and *j*, has its own linear PageRank equation, this forms a linear system of equations of the size of the number of nodes. There is the constraint that since PageRank is normalized, the sum of all *PR(i)* must add up to 1:

$$sum(PR(i)) \;==\; 1.0,$$

for all nodes i in the graph. Since this is a linear system of equations, we can represent them in the form of:

$$MV$$

Where the matrix M contains the coefficients of the linear systems of PageRank equations, and the vector V represents the variables that need to be solved for, the eigenvector
Note that PageRank is an eigenvector equation with eigenvalue λ of 1:

$$\lambda V \;=\; MV$$

To solve for the eigenvector *V*, we typically uses the Power Iteration algorithm, also known as Power Method, which isn’t sensitive to initial guesses V_o :

$$V_{t+1} \;=\; MV_t$$

that can be reduced down to the form:

$$V_{t+1} \;=\; M^{t+1}V_0$$

Where we then get matrix-exponentiation, which was going to be the focus of the parallelization efforts. Upon further testing, however, we realized that this wouldn't work for graphs with "Dead-Ends", where the node points to itself and since the Power Iteration method is solved iteratively, the node will keep looping to itself, thus artificially increasing its probability score.

Thus, we switched to doing the modified PageRank algorithm, which solves the problem of "Dead Ends" by having a certain chance of teleporting away to a random node while still using the same Power Iteration method for solving it:

$$PR(i) = d * sum(PR(j)/N_j) \; for \; j \; -> \; i \; + \; (1 - d)/N$$

Where N_j is the number of out-links from node *j*, and N is the total number of webpages. This can be represented in matrix equation as:

$$V_t = dMV_{t-1} + (1 - d)/N * I,$$

for teleportation factor *d*, usually 0.8, and vector of all ones *I*. Convergence is defined as:

$$|| V_t - V_{t-1} || < \varepsilon,$$

For some small epsilon ε value, at most to the 6th digit (ie. 10e-6). In code the difference is taken as the L2-normalization of the delta vector, for a delta vector of the vector of this iteration minus the vector of the previous iteration.

One crucial difference between the modified PageRank algorithm and the naive PageRank algorithm is that since each iteration now needs to add a teleportation vector, which is a coefficient, we can no longer use the matrix-exponentiation.

Scaling-wise it is roughly linear in O(logn) time for n being the total number of webpages.

We elected to use C++ because we are both familiar with the programming language and the coursework has been taught in C++.

We used the Eigen matrices library for the convenience of having matrix operations that we don't have to implement ourselves if we just used the default arrays. Note that we did not use the Eigen library to directly compute the operations we want to parallelize, though we did test the algorithm first with the library in the original serial version of the code.

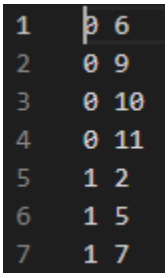
The base framework that we used was OpenMP, which has been taught in class and is the parallelization framework that we are most familiar with. Thus, out of the options taught in class, OpenMP, MPI, and OpenACC, we believed OpenMP would be the most forgiving to work with.

The second framework that we decided to use was C++ native threads. We chose this because we were already working with C++ for OpenMP and we wanted to see how its standard library compared to OpenMP. Most implementations of C++ native threads are built upon pthreads. The benefits to using C++ native threads are that they have a higher level of abstraction and the fact that native threads is a C++ class library.

The Power Iteration algorithm works by multiplying an initial guess vector with the matrix. The resultant vector is then multiplied with the same matrix. The previous step gets repeated until each point in the vector converges. For this reason, parallelizing the loop was not an option as the current iteration requires the resultant vector from the previous iteration. Given how large the graphs to PageRank can get, we decided that parallelizing the matrix vector multiplication would be crucial. We decided to use the tiling method to parallelize matrix vector multiplication because it is very likely that the graphs that PageRank works with do not fit into cache or even RAM. Furthermore, for the OpenMP version, we decided to parallelize the L2 norm calculation.

3. Experiments / Proof of concept evaluation

The Dataset is sourced from Open Source PageRank Implementation's testing suite under its /test/ directory by Panos Louridas, Georgios Gousios, and Yanran Li. The dataset includes famous graphs from mathematics such as the Folkman or Meredith graphs. The graphs for testing are stored in foo.txt files as a pair of "from" and "to" node-indices that form the directed edges of the graph or "links", with the indices being 0-based:



Size-wise, there are two sets of graphs in the "barabasi" and "erdos" set that contain 10k to 100k node graphs in increments of 10k nodes. While the graphs are mostly sparse, the largest of datasets have up to 50k links. The testing dataset does come with the corresponding PageRank scores that can be used for validation, though it should be noted that if one were to examine the C++ code under "tables.cpp", it would be clear that their implementation of PageRank is using a far more complicated version of the PageRank algorithm that is far beyond the scope of this group, and thus their PageRank scores are different from our simplified modified PageRank algorithm's scores.

We decided to run each version of the algorithm using 1,2,4,8,12,14, and 16 threads with a quick Bash script to simplify the code execution without having to use sBatch, which neither of us are familiar with. We measure the time it took for each version to run the algorithm only. We did not include the time for preprocessing, setup, or validation. Each version of the algorithm ran the dodecahedral, heawood, icosahedral, robertson, and smallest cyclic group graphs.

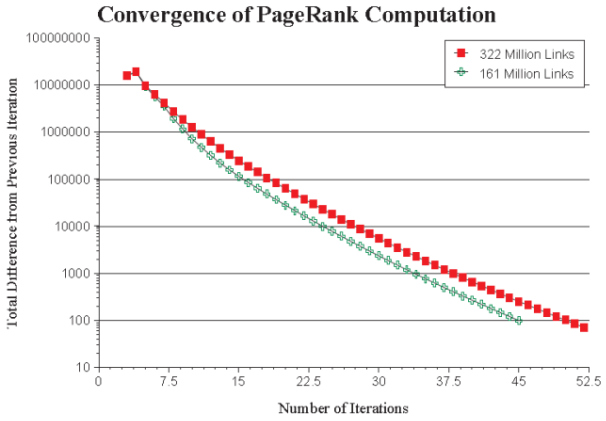
Since we know that PageRank's Power Iteration method should converges without too many iterations, given that with 161 million links it only take 47 iterations but when the number of links is doubled to 322 million links it only needs 7 more iterations to converge, we setted the maximum-iteration limit of 100 Power Iterations.

Power Method

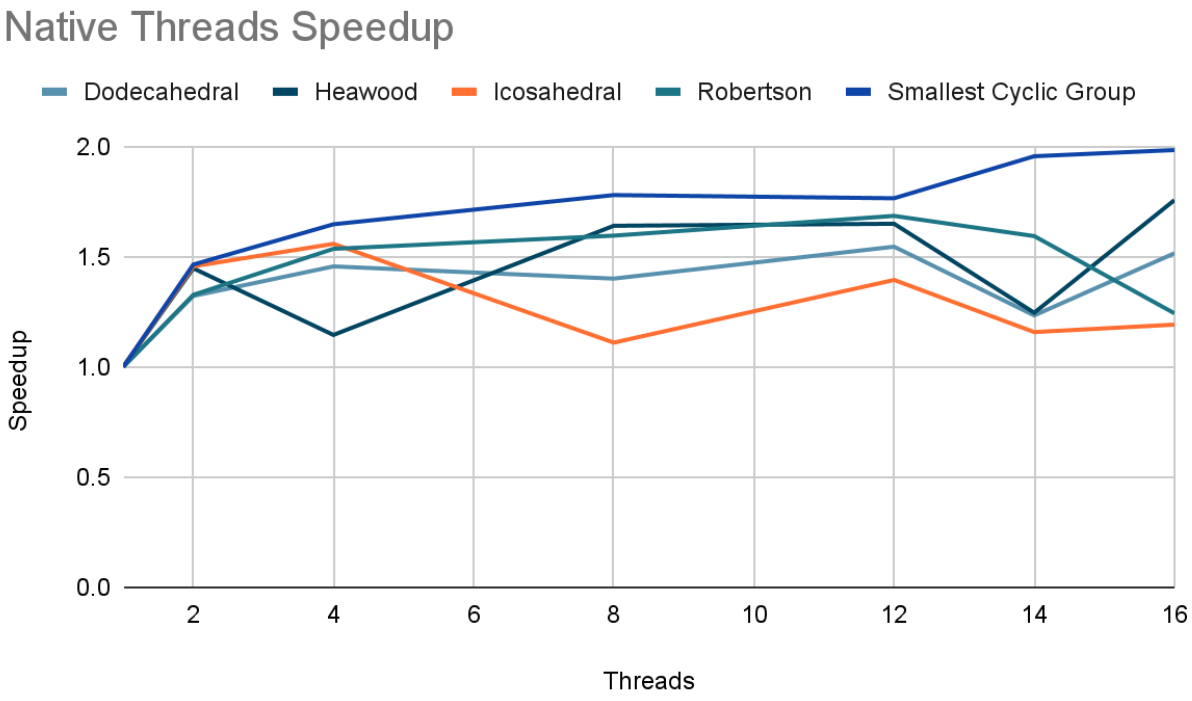
Convergence Property

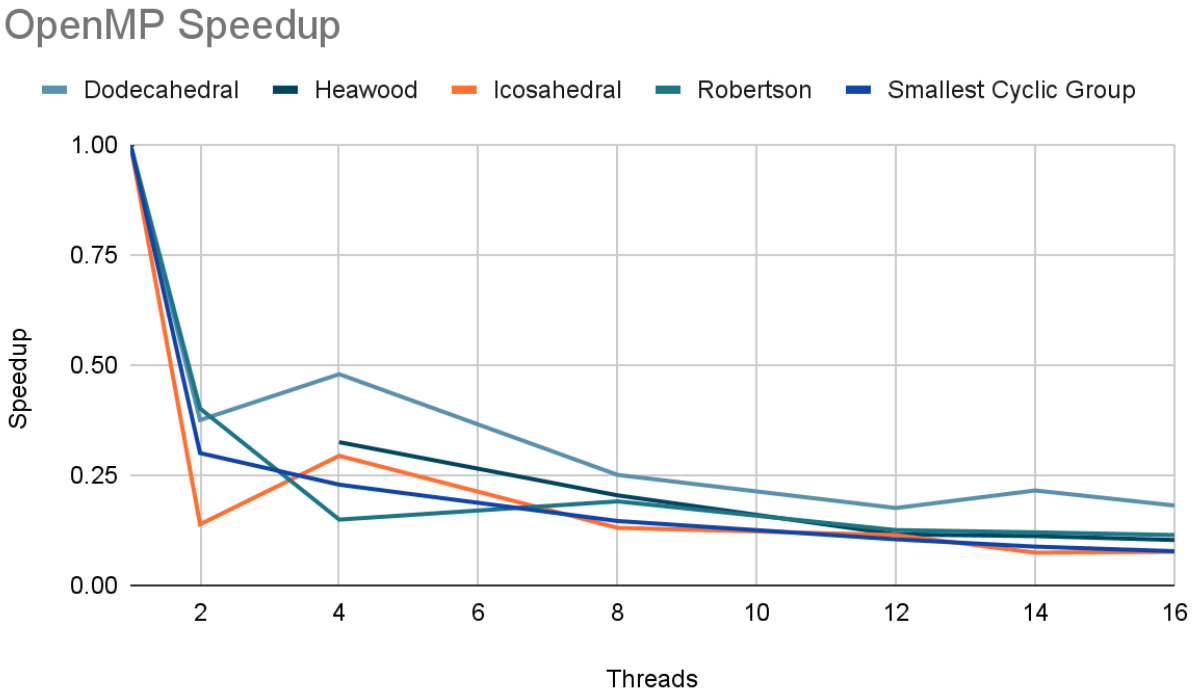
- PR (322 Million Links): 52 iterations
- PR (161 Million Links): 45 iterations
- Scaling factor is roughly linear in *logn* → O(log n)

n = Total # of webpages



For the other primary perimeters of the modified PageRank algorithm, we used the teleporting factor *(1 - d)* to 0.2, *d = 0.8*, since it is empirically usually around *d = 0.8 - 0.85*, and for the epsilon ε prevision we used 10^{-6} because according to Dr. Yi Fang, who is the author of the lecture slides referenced so far, that's enough precision without doing too much needless computation to get even more precision.





4. Discussion & Conclusion

There were several turning points in this group project. Our original proposal was to parallelize a Neural Network classifier used to estimate a person's age from a single image. This was decided back when we still have our third member in the group, Ryan. Due to personal emergencies, however, Ryan had to drop the quarter, and unfortunately Ryan was also the one most familiar with Neural Networks out of the three of us. After attempting and failing to comprehend how to write a neural network from scratch and figure out a parallelization framework for it, we pivoted by ditching the neural network idea altogether after brief discussions with other classmates let us realize we chose a harder problem to solve than necessary. Thus we decided to work on something we understand better, which ultimately became the PageRank algorithm, which was covered in another course this quarter, and we thought we could parallelize the naive PageRank algorithm. Specifically we were going to parallelize the Matrix-Exponentiation portion of the naive PageRank algorithm, which would also let us re-use the Matrix-Matrix Multiplication project code we did earlier in the quarter. Unfortunately testing the first iteration of the naive PageRank algorithm made us realize that we need to switch to the modified version of the algorithm with teleporting or else we would be running into Dead-Ends frequently. Thus we did, but at this point we both had finals for other classes so our available time to work on this project was relatively short. Thus we decided to go with the conventional, empirically frequently used parameters for the algorithm parameters like the maximum-iterations allowed, epsilon ε precision, and teleportation factor $d = 0.8$. We also limited our testing of different numbers of threads to just 1 to 16 threads since it was difficult to allocate more threads on WAVE HPC when everyone's scrambling to finish their group projects.

Difficulties we encountered includes not fully comprehending how to build a neural network from the ground up, knowledgeable group member leaving, not realizing that we didn't

necessarily have to choose the hardest problem to tackle, trying to use the naive PageRank algorithm but running into Dead-Ends that forced us into using the modified PageRank algorithm, and finally correctly reading in the graphs from the testing directory. Ojas has the difficulty of using a parallelization framework that was not taught in class and had to be learned to implement his portion of the project's parallelization.

Fortunately, reading in the graph and constructing the linear system of PageRank equations as a Matrix-Vector worked after some debugging into how to resize the matrix when expansion is required. Most test datasets that ran did converge without going over the maximum-iteration limit, and we managed to get one of the larger datasets, "erdos-10000.txt"'s graph running on WAVE and converging once. Another milestone reached is that the Native Threads version actually achieved speed-up, however modest it may be. The OpenMP version also had shorter execution times in general when compared to the Native Threads version for the same number of threads.

Unfortunately, some of the larger datasets in the barabasi and erdos sets were not able to be run on WAVE for some reason unknown to us, without being killed automatically after a brief computation. It also only ran successfully for the OpenMP version, while the Native Threads version was killed before convergence. Another thing that didn't work well as originally intended was that while the testing directory provides the expected PageRank scores for each node, due to the simplified nature of our modified PageRank algorithm the values we got were inevitably different from PageRank scores reached by the authors of the repository we pulled the testing directory from. Another thing that didn't work well was the modified PageRank algorithm itself, which while avoiding the Dead-End problem of naive Pagerank algorithms, was not outputting the correct eigenvector as expected when checked for correctness. We suspect there's some edge cases we didn't cover or flaws in the simplified version of the modified PageRank algorithm we used that we didn't realize and comprehend in time to fix. The OpenMP version also failed to achieve speedup from the parallel tiling operation.

In conclusion, we found this project very challenging. At the same time, we feel that we were able to learn a lot through the struggles we encountered. This includes information such as better hardware utilization through tiling, code optimization through compilers, and use of debuggers to cut down on debugging time. Moving forward, we can apply this knowledge not only to parallel computing problems but software engineering in general.

5. **Project Plan / Task Distribution**
    a. Benjamin Wang
        i. Researched PageRank algorithm, Power Iteration algorithm, and Eigen library for convenient matrix operations.
        ii. Implemented the original serial version of the Naive PageRank algorithm and Power Iteration method of solving the linear system of PageRank equations
        iii. Implemented the testing functions that read in the graphs from the testing directory and convert into linear system of PageRank equations as a matrix
        iv. Implemented the OpenMP version of the modified PageRank algorithm
        v. Co-wrote this report and presentation
    b. Ojas Malwankar

    i. Found the testing directory of the directed graphs to test our PageRank implementation on.
    ii. Implemented the Native Threads version of the modified PageRank algorithm
    iii. Co-wrote this report and presentation
c. Our task distribution went without much modification once we decided to pivot to the PageRank algorithm for parallelization instead of the original proposal of neural networks for age estimation, so we both more or less ended up doing the tasks we assigned ourselves.

Work Cited

Fang, Yi “COEN 272 Lecture 8 PageRank” Lecture at Department of Computer Science and Engineering, Santa Clara University, CA, November 15, 2021. Accessed December 6, 2021.

Panos Louridas, Georgios Gousios, and Yanran Li. “Open-Source PageRank Implementation”, (2014), GitHub Repository, https://github.com/louridas/pagerank