Parallel and Improved PageRank Algorithm for GPU-CPU Collaborative Environment

Akhil Maddineni
Department of Computer Science
Georgia State University
Atlanta, USA
amaddineni2@student.gsu.edu

Avinash Reddy Attluri
Department of Computer Science
Georgia State University
Atlanta, USA
aattluri1@student.gsu.edu

Praneeth Singirikonda
Department of Computer Science
Georgia State University
Atlanta, USA
psingirikonda1@student.gsu.edu

***Abstract*— Internet is a huge with billions of websites, to search in such a huge websites population around the world uses search engines to get relevant websites. So, the retrieval of correct and relevant websites plays a major role in deciding the accuracy of search engine. For this global head of search engines Google uses an algorithm called PageRank Algorithm. In Page Ranking Algorithm all the existing implementations are based on clusters and it will take huge amount of time to process the data base and get the results. So, we have to decrease the processing time by implementing parallel approaches. In cluster-based approaches the latency is in between the nodes. As it is a huge data set of web links distributed globally, the union of all these partial results is very big issue. So, by implementing parallel cluster computation the latency will overcome the performance. As we can host the complete list on a single data server, we can use PCI based communication as a solution with help of high parallel computation power of GPUs. So, we aim to provide parallel solution for the PageRank.**

Keywords- GPU, CUDA, PCI, SPMV (Sparse Matrix Vector), SDK.

I. INTRODUCTION

The internet at current trend is used most widely to search any kind of information. The search engines interestingly very good and important results which will completely align with the user requirement. This is only possible by search engines by defining a computed heuristic called PageRank. The rank here is just a value tagged to each page specifying its importance which is assigned by computing a PageRank algorithm. The algorithm iterates over all the all the web pages like a graph accordingly specified by the algorithm. The search engines give the values based on the rank assigned to each of the nodes, here a node defines a single webpage. The lower the PageRank value is means more popular page is. But to perform this algorithm on a large web base a multiple CPU-architecture is defined but communication overhead in the network is one of the major drawbacks in this kind of architecture. One more problem is low processing power of CPU that makes the process very slow. Hence designing a page ranking algorithm by using the power of a parallel GPU-CPU that can achieve higher accuracy and also performs at high throughput to calculate PageRank algorithm. PageRank calculation can be a non-trivial task. Many numbers of challenges will be faced while calculating the PageRank values of a web page. Having the input data very large is one of the major difficulties faced which will require a lot of computing power for the hardware. It is estimated that the number of web pages on World Wide Web is over 40 billion. Even if we consider a small fraction of that dataset (few billion or hundreds of million), it would still be a very difficult task to compute the PageRank value. Other problems faced would be the characteristic of the web which is a dynamic factor. So here the characteristic indicate that the webpage content keeps on changing which will indirectly lead to the change in the hyperlinks that are embedded in the page leading to the change of entire web structure. The second characteristic that needs to be considered is, everyday billions of webpages to the huge web graph every year. To consider these changes and keep the page rank values up to date the page rank algorithm has to be carried out in a short span of time .So considering all these aspects there is a need for the page rank algorithm to be deployed in a high performing architecture that might include specialized clusters or the high computing nodes and threads.

The design of the algorithm is the crucial to achieve maximum efficiency that has to be deployed in a high performing architecture. Hence a parallel algorithm can achieve great heights in performance by harnessing the many cores available in a GPU. By designing a PageRank algorithm which can be deployed in a GPU architecture and defining good heuristics that can be calculated in the less amount of time can attain a better throughput. This type of algorithm design is very much beneficial for search engines where a CPU architecture would perform much better in a case that requires fewer loads and CPU based applications. The rest of the paper is organized as follows. Section II gives the background and review of PageRank concept. Section III gives detailed description on previous work done on parallel approaches towards PageRanking algorithms. Section IV gives detailed description about our proposed model for improved parallel algorithm for PageRank computation followed by conclusion and references in section V and VI respectively.

II. BACKGROUND

A. What is PageRank and how is it computed?

In order for search engines to retrieve most relevant and important webpages as per the search we need to rank the pages according to their importance, so this ranking of webpages by google is called as Page Rank Algorithm. In Page Rank algorithm the page rank is calculated based up on the importance of incoming links to that page. The intuition here is, if a page ‘p’ has incoming links from

important webpages then it means page p is also an important page so its rank should be high. Therefore, the page rank of page p is an addition of part of page ranks of incoming links (Figure 1.) Similarly, page rank of page p is also divided among the pages it is pointing to. So, the formula of page rank is

$$r(P_i) = \sum_{P_j \in B_{P_i}} \frac{r(P_j)}{|P_j|}$$

So here page rank of Pi is the sum of a division of PageRanks of incoming pages Pj (Pj/|Pj|). Here the |Pj| is the number of pages Pj is pointing to. As in the starting PageRank of Page Pj will be unknown, so in starting PageRank will start by giving uniform page rank for each page i.e., 1/n where n is the number of pages, then page rank for each page will be calculated iteratively by using below formula. Here rk +1(Pi) is the PageRank of page Pi at iteration k+1 [1]. Then,

$$r_{k+1}(P_i) = \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|P_j|}$$

This iteration will be continued till the page ranks scores converges to a stable value. But if we use the above initial values, few pages will get more PageRank if they are pointed to them self, and there is no outgoing link from that page.This results in the PageRank will sink to zero eventually and for some page it will become 1 in the iterative process. To fix the problem, they have identified nodes with zero outdegree and introduced damping factor[2] which will influence the random walker. The modified formula with damping factor is as below:

$$r_k + 1(P_i) = \frac{1-\alpha}{N} + \alpha[\frac{\pi^{k(T)}a}{N} + \sum_{P_j \in B_{P_i}} \frac{r_k(P_j)}{|P_j|}]$$

α - Damping factor.
$\pi^{k(T)}$ - Row vector containing PageRank scores of the web pages at the k^{th} iteration.
α - Binary vector ('1' → if page is a dangling node '0' → Otherwise.)

B. Webgraph Data

The algorithm required to compute PageRank requires information about webpages. This information has been retrieved from Amazon Web Graph, a .txt file containing nodes that represent webpages and directed edge (in-links and out-links) between them. Search engines collect data from the web by using a Web crawler. Since the web graph is very huge, there is a need to store data the data in an efficient way. Hence, usage of a special data structure is necessary. This data structure is like Binary Link Structure file [1] which contains the following fields:

Source ID	Out Degree	Outlink IDs
id 1	4	id 87, id 123, id 51, id 45
id 2	2	id 7, id 15
id 3	3	id 23, id 81, id 245
...	...	...

- Source ID - 4-byte serial number (of Integer type) which is the index of the corresponding row tuple.
- Out Degree - 4-byte whole number which represents the count of out-links of that webpage.
- Series of 4-byte integers that represents a list of Destination IDs.

III. PREVIOUS WORK DONE ON PARALLEL PAGERANK COMPUTATION

Referring through many papers there were quite many parallel implementations for the PageRank algorithm but all of them were following a CPU based infrastructure but not the GPU bases. Some of the implementations were PC Cluster, P2P architectures. All these methods are more of a hardware configuration change so as to improve the network architecture and its efficiency. One of the major problems that are faced during a network-based architecture is the communications overhead between the nodes. The existing GPU based architecture tries to aim of implementing the sparse matrix vector problem very efficiently.

One of the other approaches which was implementing the page rank on CUDA platform was aiming at improving the node representation and its architecture. In this approach linear vector formation is used to store the page rank values after computation, while the nodes representing each web page is stored in a different data structure called binary link data structure. But although this approach has produced enough efficient, it has not been tested on different datasets considering different metrics which does not prove any reliability. In the aforementioned paper, each of the cuda threads is used to calculate the page ranks of each outgoing link of a single node in a sequential order which takes a complexity of O(n). This approach faces a drawback of not using the whole of CUDA block containing 1024 threads efficiently. The large dataset is divided in to chunks of 1024 tuples from the binary link structure which are computed simultaneously.

The PageRank calculation terminates when a convergence condition is satisfied. The algorithm is performed iteratively and after every iteration the distance between the two vectors πk and πk+1 goes below a particular threshold value the convergence condition is satisfied meaning the page rank vectors are having a stable value by following the eigen value principle. The distance between the two PageRank vectors is calculated using Chebyshev formula, which states that distance between two vectors is the maximum of the difference between corresponding individual elements of the two vectors.

$$|\pi^k - \pi^{k+1}| = max_i(|\pi_i^k - \pi_i^{k+1}|)$$

Finding the maximum between the individual differences has a complexity of O(n), if a sequential algorithm is used (i.e. loop through all nodes of n - node array). The method discussed in [1], exploits the GPU to quickly find the maximum value of in the array by using the scan algorithm that is presented in [11].

The thought behind the algorithm is to parallelly build a balanced binary tree using bottom- up approach. Any 2 nodes at same level are compared against each other, the greater value will be the new root of the tree and leaf nodes are discarded from the tree. This process is repeated until only a single node in left in the tree which, contains the maximum value of the elements in the array.

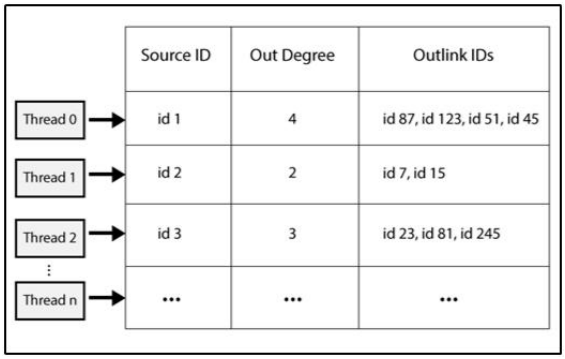

Figure 2: Parallelism using Threads

IV. OUR PROPOSED MODEL FOR IMPROVED PARALLELISM

There are not many changes we cannot do the current existing PageRank formula, we found two points which can be optimized by implementing in parallel fashion, they are – convergence of results of each page rank calculation and memory utilization of CUDA blocks and threads.

A. Conditional Model for Convergence

To achieve the first key condition instead of using Chebyshev distance formula as to calculate this we need to construct balanced binary tree, we propose an efficient way by using an over-flow flag as this operation is performed in just O(1) on a GPU. So based on this operation we can determine if the distance between individual elements of two vectors(πk and $\pi k+1$) reaches the threshold or not which means if the flag is set to true, we exceed the value of threshold and if it is not set to true we can say that the distance has not exceed the threshold value.

$$\text{difference} := \left|\pi_i^k - \pi_i^{k+1}\right|$$
$$\text{if (difference} \geq \epsilon \text{) then}$$
$$\text{flag} := \text{true}$$

So, by checking the flags we can say that if the distance does not exceed the threshold and the PageRank values are stable.

B. Improved Block-Thread Utilization

In this method, we have maximized the number of pages whose pageranks can be calculated simultaneously. In the latest GPUs (GTX 10 series) manufactured by Nvidia, there are maximum of 1024 threads per block and 65535 blocks.

We have assigned one complete block to a tuple in the Binary Link Structure and Threads present in that block will be used to calculate the pageranks of all the out-links of that source page simultaneously. This improves parallelism which results in an overall performance boost. Hence a pageranks of at least 65535 pages can be computed at any given time (assuming all tuples have at least one out-links). It is a huge boost when compared to the previous implementation discussed, which merely provides 1024 parallel computations.

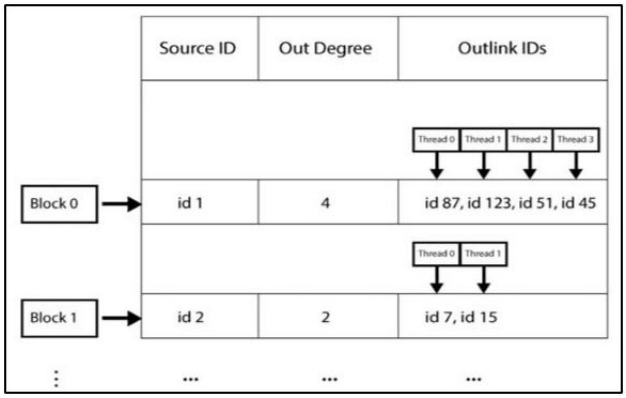

Figure 3: Improved Block-Thread utilization

V. IMPLEMENTATION AND RESULTS

For this project we have considered 2 heuristics to improve the parallel performance of the pagerank algorithm. Hence, we have evaluated our work on the two metrics which are Accuracy and Computation time. The accuracy is calculated using the mean error formula (difference between the pagerank value we got and the pagerank computed through the traditional sequential approach).

The GPU used for this project is Nvidia GTX 1050 with 2GB video RAM. This GPU contains 65535 blocks with each block containing 1024 threads. The data set contains 262111 nodes and 1234877 directed edges. The computation time for sequential and parallel algorithms is calculated using a time library in C++. From the results it is evident that our parallel algorithm has a significant speedup when compared to the existing parallel implementation and huge improvement in running time when compared to sequential algorithm. The accuracy achieved using the 2 heuristics is 82.4%.

PageRank Algorithm	Running time (in milli secs)
Sequential	62351
Existing Parallel implementation [1]	1578
Our Algorithm	992

VI. CONCLUSION

By utilizing the architectural benefits of CUDA, we improved the calculation time to compute PageRank for a given dataset. So, we proposed a parallel algorithm which improves the Page Rank Calculation.

REFERENCES

[1] Nhat Tan Duong, Anh Tu Nguyen, Quang Anh Pham Nguyen, Huu-Duc Nguyen, 2012, *Parallel PageRank*

[2] Rungsawang and B. Manaskasemsak. 2004. An Efficient Partition-Based Parallel PageRank Algorithm. In Proceedings of the 11th International Conference Parallel and Distributed Computing.

[3] Nathan Bell and Michael Garland. 2008. *Efficient Sparse Matrix-Vector Multiplication on CUDA*. NVIDIA Technical Report.

[4] Xintian Yang, Srinivasan Parthasarathy, P. Sadayappan. 2011. Fast Sparse Matrix-Vector Multiplication on GPUs: Implications for Graph Mining. Proceedings of the LDB Endowment, Vol. 4, No. 4. Seattle, Washington.

[5] Praveen K., Vamshi Krishna K., Anil Sri Harsha B., S. Balasubramanian, P.K. Baruah. 2011. Cost Efficient PageRank Computation using GPU. IEEE International Conference on High Performance Computing (HiPC), Student Research Symposium.

[6] S. Brin and L.Page. 1998. *The anatomy of a large-scale hypertextual web search engine*. In Proceedings of the 7th WWW Conference.

[7] A. Rungsawang and B. Manaskasemsak. 2004. *Parallel PageRank Computation on a Gigabit PC Cluster*. In Proceedings of the 18th International Conference on Advance Information Networking and Application.

[8] Tianji WU, Bo WANG, Yi SHAN, Feng YAN, Yu WANG and Ningyi XU. 2010. Efficient PageRank and SpMV Computation on AMD GPUs. 39th International Conference on Parallel Processing, DOI 10.1109, p.81-89.

[9] Chebyshev distance http://en.wikipedia.org/wiki/Chebyshev-distance
M. Harris. 2007. Parallel Prefix Sum (Scan) with CUDA. NVIDIA Corporation.