ISSN:0975-9646

Parallel and Improved PageRank Algorithm for GPU-CPU Collaborative Environment

Prasann Choudhari , Eikshith Baikampadi, Paresh Patil , Sanket Gadekar

Department of Computer Engineering,
Pimpri Chinchwad College of Engineering, Pune, India

Abstract—**The internet is a huge collection of websites in the order of 10^8 bytes. Around 90% of the world's population uses search engines for getting relevant information. According to Wikipedia, more than 200 million Indians use the Internet every day. Thus the correct data retrieval least time domain is the most important task. Hence need of efficient and parallel PageRanking algorithm. All the existing implementations are cluster based and to process huge lists of data take awful lot of time. The difficulty in cluster based approach is latency among different nodes participating in the computation. Since internet has large distributions of weblinks, collaboration of partial results after processing is a major issue. Thus latency factor overcomes the performance achievement of parallel cluster computation. As complete list can be hosted on one data server, PCI based communication mechanism can be used as a solution in addition of high parallel computation power with GPUs. So our approach aims at providing a parallel solution to it.**

Keywords- GPU - Graphics Processing Unit, CUDA - Compute Unified Device Architecture, PCI - Peripheral Component Interconnect, SPMV Sparse Matrix Vector, SDK – Software Development Kit.

I. Introduction

We use the Internet to search for information on a daily basis. We use search engines to find out the useful information from the vast Internet. This is possible because the search engines are using a heuristic called PageRank [1]. It is nothing but a value to a web page that is assigned by a PageRanking algorithm. This algorithm scans all possible webpages and then calculates the rank accordingly given by a formula. Search engines show results according these ranks, which stand for the popularity of the page. The lower is the rank, more popular the page is. Traditional approaches use multi-CPU architecture and this is not a very good choice due to the communication overhead and the low processing power of CPU compared to GPU. Hence, designing a PageRanking algorithm efficiently modified for parallel GPU-CPU environment that achieves higher accuracy and consumes lesser time to evaluate the PageRank of a given webgraph. PageRank calculation is a non-trivial task. There are many challenges we encounter when calculating PageRank value of web pages. The first difficulty is that the input data is extremely huge; therefore, it requires a lot of computing effort. It is estimated that the number of web pages on World Wide Web is over 40 billion. Even when the size of data we have is just a fraction of that number (few billion or hundreds of million), it is still not easy to compute efficiently. The second problem comes from a characteristic of the Web: it is dynamic. This characteristic is reflected in two aspects. First, the content of web pages may be changed along the time. This leads to the change of hyperlinks in the pages and therefore the change of the Web's structure. Moreover, the size of the World Wide Web, which is determined by the number of web page, increased rapidly with billions of web pages being created every year. To make PageRank values always up-to-date despite these changes, PageRank calculation should to be carried out in as short a time period as possible. This comes to the need of deploying PageRank to run on high performance computing infrastructures such as specialized hardware or clusters including computing nodes.

Designing a parallel algorithm for PageRank evaluation can be very crucial when it comes to designing hardware architecture to give maximum performance in low cost. Hence a parallel algorithm can achieve great heights in performance by harnessing the many cores available in a GPU. By operating in multiple GPU architecture we can process much larger chunks of webgraph and as a result attain a better throughput. This is beneficial for search engines and the CPU architecture have fewer loads and can do better in other 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?

Search engines run a special program in order to assess the PageRanks of websites. The program assigns a special score to the websites which is an indication towards the importance of the page. The overview of the idea is: A webpage is marked as important if it is pointed towards by other important webpages. To understand this thesis, we assume that each hyperlink is a recommendation; thus, a webpage which has more in-links or recommendations must be an important webpage. If the recommendation is from another important webpage, then it is considered to be more valuable than recommendations from less important webpages. Therefore, an important page is the page which (1) has many in-links, (2) has in-links from other important pages or (3) both. From that assumption, the original formula of PageRank [2] is defined as:

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

in which, r(P_i) is the PageRank of page P_i , B(Pi) is the set of pages pointing to P_i and $|P_j|$ is the number of outlines. Since the PageRank of source pages Pj are still unknown, the above formula is converted to an iterative formula where the web page's PageRank is calculated from other PageRank of the previous iteration. Let r_k +1(P_i) denote PageRank of page P_i at iteration k+1 [1]. Then,

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

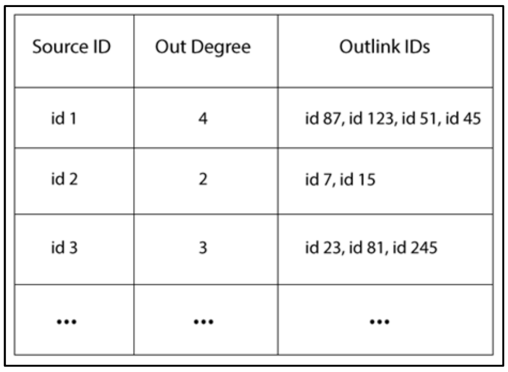
The initial PageRank values of all pages are assigned to 1/n where n is the number of pages. The iterative process is repeated until PageRank scores converge to the final stable values. However, there is a problem with this definition of PageRank. In practice, with the above initial values, some pages will accumulate more and more PageRank after each iteration and refuse to share to other pages. This problem is possible because the web graph is not strongly connected and there are many pages without any out-links in the graph. This results in that there will be PageRank value sinking to zero at the end of iterative process; hence, it becomes hard to rank web pages using PageRank when these values are mostly 0. To overcome this challenge, Brin and Page introduce some adjustments: firstly, they replaced nodes with out-degree zero (called dangling nodes) by nodes linking to all other nodes; and secondly, they added a damping factor [2] which influencing the random walks of the random surfer process of the graph. Equation (2) can be written as

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

Where α is the damping factor that descripts the probability of a user follows hyperlinks in the web graph. In the above equation. $\pi^{k(T)}$ which is row vector containing PageRank scores of N web pages at the kth iteration. α is a binary vector and its value is set to 1 if page is a dangling node else it is set to 0.

B. Webgraph Representation

The algorithm to calculate PageRank needs information related to webpages. This information is supplied to it by providing a Web Graph. Web Graph is a graph containing nodes representing webpages and directed edges to represent corresponding in-links and out-links. Search engines collect data from WWW by a program called Web crawler (or Web spider) which works like an automatic browser: it arrives at a web page, assesses hyperlinks in that page and follows those links to jump to other pages. Web crawlers create a copy of all visited page for later processing by a search engine such as indexing, displaying search results, etc. The Web graph is also constructed from this data. This graph is very huge, hence we need to store the data in an efficient way. Hence, usage of a special data structure is necessary. This data structure is similar to 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
...	...	...

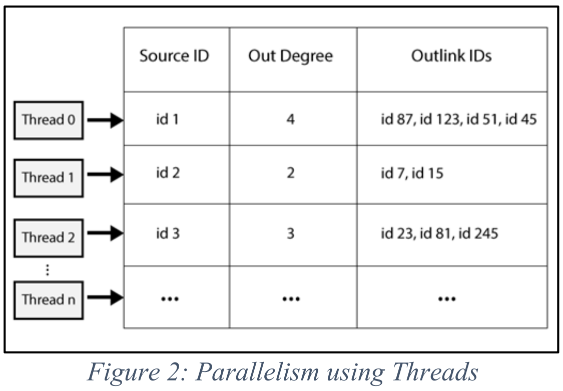
Figure 1: Binary Link Structure

- Source ID - It is a 4-byte integer serial number or the index of the corresponding row tuple.
- Out Degree - It is the 4-byte integer number of out-links of the webpage.
- Sequence of 4-byte integers representing a list of Destination IDs.

III. Previous Work Done On Parallel PageRank Computation

There have been a few implementations of PageRank on CPU based infrastructures, including PC cluster [2,3,4] or P2P architectures [5]. These methods need many processors connected together via network; therefore, a common problem issued is that the communication overhead over the computation. Existing implementation of PageRanking algorithm using GPU consists of architectures mostly focused on implementing the SPMV problem efficiently [6, 7, 8].

Another approach [1], comprises of an overview of PageRanking algorithm implemented on CUDA platform which addresses the drawbacks of huge number of nodes and their representation. They have used linear vectors to store the computed PageRank values and the nodes are stored in a special data structure called as Binary Link Structure [1]. This approach has been tested on not too-large data sets. In the aforementioned paper [1], they have used CUDA threads to achieve parallelism where each CUDA thread takes one page-id and perform PageRank computation for out-links pages pointed by that source page sequentially. This implementation harnesses only a single CUDA block which provides 1024 threads. The large dataset is divided into chunks of 1024 tuples from the Binary Link Structure [1] which are acted upon simultaneously.



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
...	...	...

Figure 2: Parallelism using Threads

The PageRank calculation was done iteratively in a loop which halted when the convergence condition was satisfied i.e. when the distance between two vectors π^k and π^{k+1} was less than a particular threshold value [1]. The distance between two vectors is calculated by Chebyshev formula [10], which states that distance between two vectors is the maximum of the difference between corresponding individual elements of the two vectors.

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

The work of finding the maximum of the individual differences has complexity O(n) if using a sequential algorithm that loop all elements of n-elements array, where n is the number of nodes. The implementation in [1], exploits GPU to find quickly the maximum value of an array by applying scan algorithm that presented in [11]. The idea of the algorithm is to build in parallel a balanced binary tree using bottom-up approach. The two nodes at the same level are compared with each other, the greater value will be used to make new root of the tree. The process repeated until only a single node, this node contains the maximum value of the element in the array.

IV. Our Proposed Model For Improved Parallelism

After detailed study of the PageRanking algorithms, we have concluded that the PageRank formula, cannot be altered anymore. Hence to achieve an improved parallel algorithm, there are two key points which have scope for optimization - Convergence condition and the efficient memory coalescing and utilization of CUDA blocks and threads.

A. Conditional Model for Convergence

For the implementation of convergence condition, instead of using Chebyshev Distance formula to calculate the maximum, which involves construction of a balanced binary tree, we propose a simpler and efficient way by using an overflow flag. The method includes finding the difference between individual elements of the two vectors π^k and π^{k+1}. This operation is performed in just in O(1) on a GPU. We maintain an overflow flag which is modified as following.

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

Now based on the status of the flag, we can determine if the distance between two vectors exceeds the threshold, completely eliminating the need to find out the maximum difference. If *flag* is set to *true*, then we can say that the distance exceeds threshold. If *flag* is not set to *true* at all, then we can say that the distance hasn't exceeded the threshold and the PageRank values are now stable.

B. Improved Block-Thread Utilization

In this approach, we have attempted to maximize the number of pages whose pageranks are calculated simultaneously. In CUDA, there are 1024 threads per block and there are a maximum of 65535 such blocks.

We assign a block to each tuple in the Binary Link Structure and Threads in that block will then calculate pageranks of all the out-links IDs of that source ID simultaneously. This gives us a higher level of parallelism and better performance. Hence a minimum of 65535 pageranks are computed at any given instant (assuming all tuples have at least one out-links). The previous implementation provided a mere 1024 parallel computations.

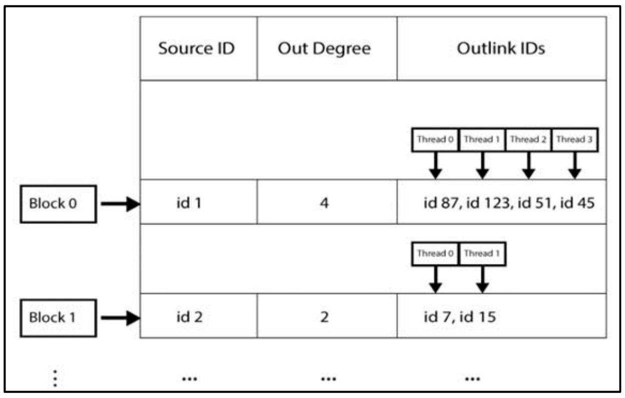


Figure 3: Improved Block-Thread utilization

V. Conclusion

In this paper, we have proposed an optimal parallel PageRanking algorithm, by efficiently utilizing the architectural benefits that CUDA provides. Clearly, our proposed approach increases the performance in terms of time taken to compute PageRank of a given dataset.

References

[1] Nhat Tan Duong, Anh Tu Nguyen, Quang Anh Pham Nguyen, Huu-Duc Nguyen, 2012, *Parallel PageRank Computation using GPUs*, in SoICT 2012, Ha Long, Vietnam - ACM - 978-1-4503-1232-5.

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

[3] 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.

[4] A. Rungsawang and B. Manaskasemsak. 2003. *PageRank computation using PC cluster*. In Proceedings of the 10th European PVM/MPI User's Group Meeting.

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

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

[7] 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.

[8] 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.

[9] 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.

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