Cost Efficient PageRank Computation using GPU

Praveen K.*, Vamshi Krishna K.†, Anil Sri Harsha B.‡, S. Balasubramanian, P.K. Baruah
Sri Satya Sai Institute of Higher Learning, Prasanthi Nilayam, India
{praveen.kkp, k.vamshi.krish, anilharsha.b}@gmail.com,
{sbalasubramanian, pkbaruah}@sssihl.edu.in

***Abstract*—The PageRank algorithm for determining the "importance" of Web pages forms the core component of Google's search technology. As the Web graph is very large, containing over a billion nodes, PageRank is generally computed offline, during the preprocessing of the Web crawl, before any queries have been issued. Viewed mathematically, PageRank is nothing but the principal Eigen vector of a sparse matrix which can be computed using any of the standard iterative methods. In this particular work, we attempt to parallelize the famous iterative method, the Power method and its variant obtained through Aitken extrapolation for computing PageRank using CUDA. From our findings, we conclude that our parallel implementation of the PageRank algorithm is highly cost effective not only in terms of the time taken for convergence, but also in terms of the number of iterations for higher values of the damping factor.**

***Index Terms*—PageRank, Principal Eigenvector, Eigenvalue, Extrapolation, Markov Matrix, CUDA, CUBLAS, CUSPARSE.**

I. Introduction

The evolution of GPU technology has outperformed the traditional Parallel Programing paradigms, introducing a new phase in the field of scientific computing. GPU based solutions have outperformed the corresponding sequential implementations by leaps and bounds. Particularly, for scientific computing applications, they provide a very valuable resource to parallelize and achieve optimum performance.

Determining a page's relevance to query terms is a complex problem for a search engine and this decides how good a search engine is. Google solves this complex problem through its PageRank [1][2] algorithm which quantitatively rates the importance of each page on the web, allowing it to rank the pages and thereby present to the user, the more important (and typically most relevant and helpful) pages first.

*Student Author
†Student Author
‡Student Author

As the Web graph is very large, containing over a billion nodes, PageRank is generally computed offline, during the preprocessing of the Web crawl, before any queries have been issued [3]. In this particular work, we parallelized a customized serial implementation of power method and a variant of it obtained through Aitken extrapolation [4] for computing PageRank.

The core component in the power method for computing PageRank is the Sparse Matrix-Vector product (SpMV) and certain vector-vector operations. The parallel implementation was done using the CUDA programing model, developed by NVIDIA. In this parallel implementation, we exploited the power of CUBLAS [5] and CUSPARSE [6] routines for the same. From our results, it is evident that the parallel implementation outperforms the serial equivalent in terms of both the factors viz. convergence time and number of iterations for specific values of the damping factor (α). We validated our work on publicly available datasets [7] [8], which are matrices that represent web graphs of sizes varying from 884 x 884 to 281903 x 281903.

The remainder of this paper is organized as follows. Section II reviews the related work done in this area. Section III, gives an overview about the power method and it's extrapolation variant for computing PageRank. Section IV discusses the methodology used to parallelize the serial implementation. The experimental results obtained are discussed in Section V. Finally we point to some conclusions and future work that can be undertaken in Section VI.

II. Related Work

Standard works in literature for computing PageRank in parallel focused on exploiting the power of large cluster based computation. In general, the main focus of any parallel implementaion of the PageRank algorithm was on reducing the communication overhead and efficient problem partitioning. Zhu et al. [9] used an iterative aggregation and disaggregation method to effectively speedup the PageRank computation in a distributed

environment. Gleich et al. [10] modeled the computation of PageRank as a linear system and evaluated the performance of a parallel implementation on a 70-node Beowulf cluster. A parallel implementation of the standard power method for computing PageRank was done by Wu et al. [11] on AMD GPUs using OpenCL programing model. This work essentially concentrates on parallelizing the sparse matrix vector product (SpMV) using the AMD GPUs. Another work in this regard by Cevahir et al. [12] focused on extending a CPU cluster based parallel implementation on to a GPU cluster, using upto 100 GPU nodes.

Our work is an attempt on measuring performance gains in computational costs for calculating PageRank using power method and it's extrapolation variant on a single NVIDIA GPU using the standard CUBLAS and CUSPARSE libraries. To the best of our knowledge, no work has been done on parallelizing PageRank computation using a single GPU.

III. Power Method for Computing PageRank

Finding Eigenvector for a given matrix is a numerical linear algebra problem that can be solved using several available methods in the literature. But, Power method is often the method of choice due to it's ability to calculate the dominant Eigen pair. This method is an apt choice for the PageRank problem because, we are interested only in the principal Eigenvector of the given matrix. Procedurally, this method is an iterative method that finds the vector $x^{\bar{k}+1}$ from $x^{\bar{k}}$ as $x^{\bar{k}+1} = \mathbf{A}x^{\bar{k}}$ until $x^{\bar{k}+1}$ converges to desired tolerance.

When Power method converges, the vector $x^{\bar{k}+1}$ is the Eigenvector for the given matrix corresponding to the dominant Eigenvalue (which, in this case is 1). The power method for computing PageRank can be elucidated as follows. Here, δ is the 1-Norm difference

Data: Matrix $\mathbf{A}$, Initial vector $\bar{x^0}$
Result: $x^{\bar{k}+1}$, the Eigenvector of Matrix $\mathbf{A}$
repeat
  $x^{\bar{k}+1} = \mathbf{A}x^{\bar{k}}$;
  $\delta = ||x^{\bar{k}+1} - x^{\bar{k}}||_1$;
until $\delta < \epsilon$;

Algorithm 1: Power method for computing PageRank

between the vectors in the k^{th} iteration and the $k + 1^{th}$ iteration and ϵ is the desired convergence threshold. Note that $\mathbf{A} = \alpha\mathbf{\bar{P^T}} + (1 - \alpha)\mathbf{E}$. Where, E is an stochastic matrix introduced to model the complete behaviour of a random surfer on the web. This reformulation is a consequence of the fact that the sparse matrix $\mathbf{P}$ alone do not capture the complete behavior of a random surfer. This introduces a new parameter α, also known as "Damping Factor", whose value ranges between 0 and 1. It is an empirically proven fact that the damping factor α signifies the sub-dominant Eigenvalue of the Google matrix [13]. Observe that $\mathbf{P}$ is a sparse matrix and the sparsity is lost by taking the Convex Combination.

A. An Efficient Power method type Algorithm for the PageRank problem

We are dealing with matrices that represent the hyperlink structure of the web that are very huge in size. Considering $\mathbf{A}$ as such and applying the Power method will lead to memory overflow. This can be overcome by replacing the matrix vector product $\mathbf{A}\bar{x}$ as illustrated in the following Algorithm.

Data: Matrix $\mathbf{P}$, Initial vector $\bar{x^0}$, Damping factor α, Personalization Vector $\bar{v}$
Result: $x^{\bar{k}+1}$, the Eigenvector of Matrix $\mathbf{P}$
repeat
  $x^{\bar{k}+1} = \alpha\mathbf{P^T}x^{\bar{k}}$;
  $w = ||x^{\bar{k}}||_1 - ||x^{\bar{k}+1}||_1$;
  $x^{\bar{k}+1} = x^{\bar{k}+1} + w\bar{v}$;
  $\delta = ||x^{\bar{k}+1} - x^{\bar{k}}||_1$;
until $\delta < \epsilon$;

Algorithm 2: Modified Power method for computing PageRank

Where $\bar{v}$ is the Personalization vector which is $\frac{1}{n}\bar{e}$ ($\bar{e}$ is a column vector of all ones). The irreducibility of $\mathbf{A}$ implies that the dominant Eigenvalue is 1.

B. Extrapolation Techniques for accelerating Convergence of Power Method

Due to the sheer size of the web (over 50 billion links) PageRank computation by power method can take several days to complete. Speeding up this computation is highly critical.

Though, power method is an apt choice for the PageRank problem, the convergence of the method is a matter of concern when α goes closer to 1. Hence, there is a need to lookout for variants of the power method that can accelerate the computation of the desired Principal Eigenvector of the given Matrix.

1) Aitken Extrapolation Technique: The Aitken Extrapolation technique [4] for computing PageRank is an extension of the basic power method in which the component values of the intermediary vectors in the

power iterations are periodically pruned so that the resultant vectors obtained are refined in the direction of the desired vector. The algorithm can be elucidated as follows.

Data: Matix $\mathbf{A}$, Initial vector $\bar{x^0}$
Result: $\bar{x^{k+1}}$, the Eigenvector of Matrix $\mathbf{A}$
repeat
  $\bar{x^{k+1}} = \mathbf{A}\bar{x^k}$;
  $\delta = ||\bar{x^{k+1}} - \bar{x^k}||_1$;
  periodically x^k = **Aitken**($\bar{x}^{k-2}$,$\bar{x}^{k-1}$, $\bar{x}^k$);
until $\delta < \epsilon$;

Algorithm 3: Aitken Power method for computing PageRank

function $\bar{x}$ = **Aitken**($\bar{x}^{k-2}$,$\bar{x}^{k-1}$, $\bar{x}^k$);
for $i \leftarrow 1$ **to** n **do**
  $\bar{g_i}$ = (x_i^{k-1} - x_i^{k-2})^2;
  $\bar{h_i}$ = x_i^k - 2x_i^{k-1}+ x_i^{k-2};
  x_i = x_i^k - g_i/hi;
end

Algorithm 4: The Aitken Routine

2) Operation Count: In order for an extrapolation method such as Aitken Extrapolation to be useful, the overhead should be minimal i.e. the costs in addition to the cost of applying power iteration to generate iterates. The operation count of the loop in Algorithm 4 is O(n), where n is the number of pages on the web. The operation count of one extrapolation step is less than the operation count of a single iteration of the Power Method, and since Aitken Extrapolation may be applied only periodically, Aitken Extrapolation has minimal overhead. In the implementation, the additional cost of each application of Aitken Extrapolation is about 1% of the cost of a single iteration of the Power Method which is negligible.

IV. Scope for Parallelization

The power method for computing PageRank is highly parallelizable since most of the computation involves Sparse Matrix-Vector product (SpMV) and vector-vector operations. Since the matrix under consideration has a sparse structure, we used the CUSPARSE library provided by NVIDIA to store the matrix and perform the corresponding operations. The sparse matrix is constructed by reading an adjacency list, which represents the nonzero values in the matrix.

The list of operations involved in power method for computing PageRank and their equivalent CUSPARSE/CUBLAS library calls used in our implementation can be summarized as follows.

- $\bar{y} = \alpha\mathbf{P^T}\bar{x} \rightarrow$ cusparseDcsrmv(), to perform the sparse matrix P^T and dense vector $\bar{x}$ product
- $||\bar{x}||_1$ and $||\bar{y}||_1 \rightarrow$ cublasDasum(), to calculate the 1-Norm of the vectors (Here, both the vectors $\bar{x}$ and $\bar{y}$ are dense)
- $\bar{y} = \bar{y} + w\bar{v} \rightarrow$ cublasDaxpy(), to update the vector $\bar{y}$, a simple axpy operation
- $\delta = ||\bar{y} - \bar{x}||_1 \rightarrow$ Wrote a simple kernel in the GPU to calculate δ

A. Experimental Setup

For the experiment, our parallelized version of the power method was run on Tesla T20 based "Fermi" GPU. A serial implementation of the same was also run in Matlab R2010a on Intel(R) core(TM)2 Duo 3.00GHz processor and 3GB RAM.

B. Evaluation Dataset

The performance of the parallel implementation of power method for computing PageRank and it's corresponding extrapolation variant is evaluated on publicly available datasets [7] [8]. We also evaluated the performance on stanford.edu dataset. It consists of 281903 pages and 2382912 links. This link structure is represented as an adjacency list, which is read as an input to construct the sparse matrix P. Table I summarizes the details of the datasets used in our experimentation.

Name	Size
Computational Complexity (CC)	884 x 884
Abortion	2292 x 2292
Genetic	3468 x 3468
Stanford.edu	281903 x 281903

TABLE I
Dataset Description

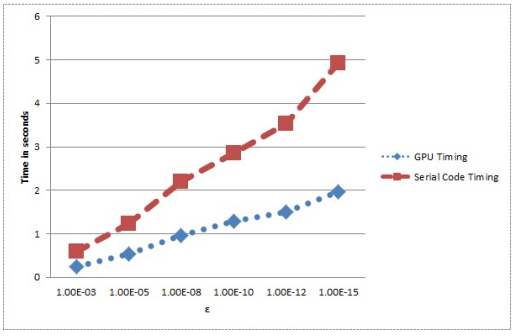

Fig. 1. Execution time for α = 0.75 on stanford.edu dataset

α	ϵ	Reduction in It(Serial)	Reduction in It(GPU)
0.75	1e-3	0	1
0.75	1e-8	3	3
0.75	1e-10	2	2
0.75	1e-12	1	2
0.80	1e-3	0	0
0.80	1e-8	1	4
0.80	1e-10	4	4
0.80	1e-12	3	3
0.99	1e-3	112	114
0.99	1e-8	116	129
0.99	1e-10	126	130

TABLE II
SAMPLE RESULTS FOR EXTRAPOLATION ON GENETIC DATASET

α	ϵ	Reduction in It(Serial)	Reduction in It(GPU)
0.85	1e-3	3	4
0.85	1e-8	4	5
0.85	1e-10	4	5
0.85	1e-12	5	5
0.95	1e-3	4	7
0.95	1e-8	5	7
0.95	1e-10	6	7
0.95	1e-12	7	7

TABLE III
SAMPLE RESULTS FOR EXTRAPOLATION ON CC DATASET

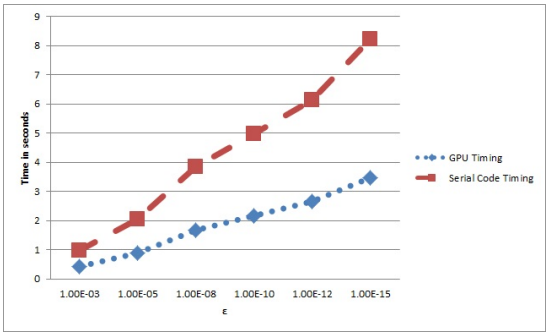

Fig. 2. Execution time for α = 0.85 on stanford.edu dataset

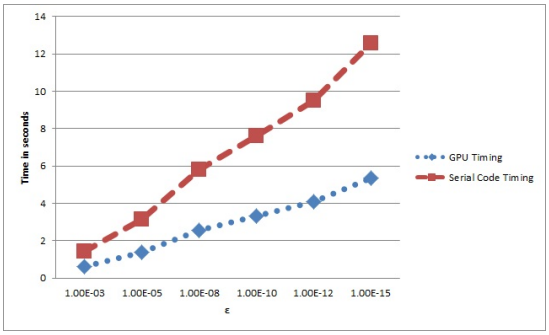

Fig. 3. Execution time for α = 0.90 on stanford.edu dataset

V. RESULTS[1]

In this section, we quantify the performance of our parallel implementation of the power method and it's extrapolation variant on datasets of varying sizes. The performance of the same is evaluated for varying values of α and tolerance factor ϵ. For our experimentation, the α values ranged from 0.75 to 0.99 and the ϵ values ranged from $1e-3$ to $1e-15$. The results of the datasets titled computational complexity and Genetic are tabulated in Tables II and III. Here, "It" refers to the total number of iterations taken for the power method to converge and "Ti" refers to the time taken for convergence in seconds.

From our findings, we observed substantial performance gains on all the datasets considered for experimentation and it is more pronounced in the case of stanford.edu dataset. As expected, the parallel implementation outperformed the serial implementation in terms of time taken for convergence for all the datasets. We also found a potential gain in terms of the number of iterations for convergence as α became closer to 1 and for higher tolerance levels. For instance, for α = 0.99 and ϵ = 1e-15, the serial implementation, converged in 18,234 iterations, whereas, the parallel implementation took merely 3013 iterations for convergence. Thus, we obtained a performance improvement of 16.52%.

We obtained positive results even in the case of Aitken Extrapolation in terms of the number of iterations taken for convergence. We measured the improvement in terms of reduction in the number of iterations taken for convergence. We summarize our findings in Tables II and III. Specifically, for the Genetic dataset, for α = 0.99, and for varying ϵ values, on an average, we observed that, the parallel implementation reduced the number of iterations for convergence by a difference of 6.

[1]Due to the page limit constraints, we could present only a sample of our obtained results

VI. CONCLUSIONS AND FUTURE WORK

PageRank computation that forms the core component of Google's search technology has been successfully parallelized using the CUDA programing model on a single GPU. We performed an extensive evaluation of our parallel implementation for different input parameter values and on datasets of varying sizes. From our results, it is clearly evident that our work is very relevant as it not only to minimizes the time involved in computing PageRank, but also reduces the number of iterations for convergence.

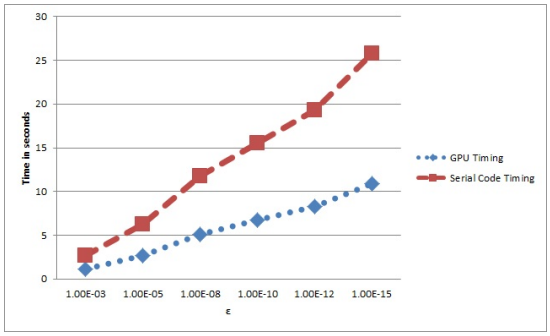

Fig. 4. Execution time for α = 0.95 on stanford.edu dataset

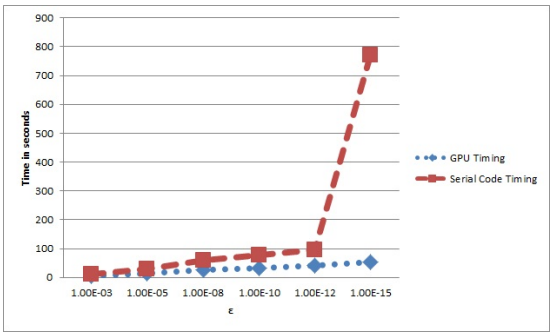

Fig. 5. Execution time for α = 0.99 on stanford.edu dataset

We intend to extend this parallel implementation on larger datasets like stanford-berkely.edu [14] that reflect the size of the World Wide Web in greater magnitude. All the datasets that are considered for experimentation are large but sill fit within the limits of the GPU memory. The single GPU approach for computing PageRank may not be scalable for larger matrices as the on-board GPU memory is a major constraint. Therefore, we look out for alternatives to accommodate large scale matrices. We also look out for higher performance gains through better optimization techniques at various levels of the algorithm. Though Aitken extrapolation gave positive results, it suffers a performance loss because the pruning of the component values of the iterates may not be optimal. In this regard, we also intend to parallelize another variant of power method obtained using quadratic extrapolation technique, which prunes the values in an optimal way.

ACKNOWLEDGMENT

We would like to express our deepest sense of gratitude to our Founder Chancellor Bhagwan Sri Sathya Sai Baba. Without His grace, this work would have been impossible. We also acknowledge NVIDIA Pune division for providing the GPU infrastructure and support.

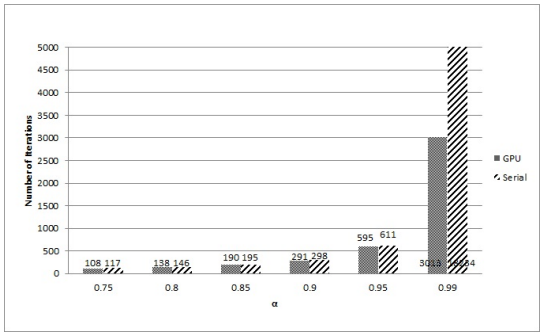

Fig. 6. Number of iterations for convergence on stanford.edu dataset (Power Method), ϵ = 1e-15

REFERENCES

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

[2] Kurt Bryan and Tanya Leise, "The $25,000,000,000 eigenvector: the linear algebra behind google," *SIAM Review*, vol. 48, pp. 569–581, 2006.

[3] Sergey Brin and Lawrence Page, "The anatomy of a large-scale hypertextual web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 107 – 117, 1998, Proceedings of the Seventh International World Wide Web Conference.

[4] Sepandar Kamvar, Taher Haveliwala, Christopher Manning, and Gene Golub, "Extrapolation methods for accelerating pagerank computations," in *In Proceedings of the Twelfth International World Wide Web Conference*. 2003, pp. 261–270, ACM Press.

[5] *CUDA CUBLAS Library, PG-05326-040V01*, NVIDIA Corporation, April 2011.

[6] *CUDA CUSPARSE Library, PG-05329-040V01*, NVIDIA Corporation, January 2011.

[7] Sepandar D. Kamvar, "Test datasets for link analysis algorithms, http://kamvar.org/assets/data/stanford-web.tar.gz," Accessed Online.

[8] University of Toronto Department of Computer Science, "Datasets for evaluatiing link analysis algorithms, www.cs.toronto.edu/ tsap/experiments/datasets/index.html," Accessed Online.

[9] Yangbo Zhu, Shaozhi Ye, and Xing Li, "Distributed PageRank computation based on iterative aggregation-disaggregation methods," in *Proceedings of the 14th ACM international conference on Information and knowledge management*.

[10] David Gleich, "Fast parallel PageRank: A linear system approach," Tech. Rep., 2004.

[11] Tianji Wu, Bo Wang, Yi Shan, Feng Yan, Yu Wang, and Ningyi Xu, "Efficient PageRank and spmv computation on AMD gpus," *Parallel Processing, International Conference on*, vol. 0, pp. 81–89, 2010.

[12] Ata Turk B. Barla Cambazoglu Akira Nukada Ali Cevahir, Cevdet Aykanat and Satoshi Matsuoka, "Efficient PageRank on GPU clusters," *18th Workshop for the evaluation of high-performance computing architecture*, 2009.

[13] Taher H. Haveliwala, Sepandar D. Kamvar, and Ar D. Kamvar, "The second eigenvalue of the google matrix," 2003.

[14] Sepandar Kamvar, Taher Haveliwala, and Gene Golub, "Adaptive methods for the computation of PageRank," *Linear Algebra and its Applications*, vol. 386, no. 0, pp. 51 – 65, 2004.