Parallelization of PageRank on Multicore Processors

Tarun Kumar[1], Parikshit Sondhi[2], and Ankush Mittal[3,*]

[1] Samsung Noida Mobile Center, Noida
tarun.krgtm2002@gmail.com
[2] Department of Computer Science University of Illinois at Urbana Champaign
sondhi1.uiuc@gmail.com
[3] College of Engineering Roorkee
dr.ankush.mittal@gmail.com

Abstract. PageRank is a prominent metric used by search engines for ranking of search results. Page rank of a particular web page is a function of page ranks of all the web pages pointing to this page. The algorithm works on a large number of web pages and is thus computational intensive. The need of hardware is currently served by connecting thousands of computers in cluster. But faster and less complex alternatives to this system can be found in multicore processors. In this paper, we identify major issues involved in porting PageRank algorithm on Cell BE Processor and CUDA, and their possible solutions. The work is evaluated on three input graphs of different sizes ranging from 0.35 million nodes to 1.3 million. Our results show that PageRank algorithm runs 2.8 times fast on CUDA compared to Xeon dual core 3.0 GHz.

Keywords: Cell BE Processor, CUDA, Multicore Processor, PageRank, Web Graph.

1 Introduction

PAGERANK of a webpage, first introduced by Google is a prominent characteristic used by search engines while ranking of search results [1]. PageRank first introduced in [2], exploits the link structure of web. It assigns a relative importance measure called rank of the page, to each web page.

Rank of a particular page depends upon the rank of the web pages linking to this page. Higher the page rank more important is the page. PageRank algorithm itself is computational intensive and it has to work upon billions of web pages. It takes time in order of days [3] to solve the PageRank. Web pages are updated, added, removed to and from WWW continually, therefore the frequent computation of rank of pages is required. Besides this, some applications of PageRank like topic sensitive search and personalized web search require large number of page rank scores recomputed to reflect the user preferences [4]. Thus, some new ways to calculate rank of web pages in minimum possible time are always sought. A recent breakthrough with introduction of the Multicore Processors has provided a new alternative for solving computational intensive algorithms in efficient ways in terms of time.

* The work presented in this paper was done while author was in IIT Roorkee.

R. Ramanujam and S. Ramaswamy (Eds.): ICDCIT 2012, LNCS 7154, pp. 129–140, 2012.
© Springer-Verlag Berlin Heidelberg 2012

In this paper, we identify the issues and their possible solutions of porting PageRank algorithm on cell BE Processor followed by the implementation of PageRank algorithm on Cell BE. We also provide an implementation of PageRank algorithm on CUDA. A comparison of Cell BE and CUDA is presented on the bases of time taken to compute PageRank algorithm for standard web graphs.

The rest of the paper is organized as follows: Section 2 describes the related work on PageRank algorithm. Section 3 provides the background information on multicore architecture and PageRank algorithm. Section 4 provides implementation of PageRank on Cell BE Processor and CUDA. Section 5 shows the results. Section VI concludes the work and suggests the future work.

2 Related Work

There has been a sincere effort to reduce the time of computation of PageRank algorithm. Chen et. al. [5] has proposed some I/O efficient technique to reduce the disk reads and writes. They analyzed the link structure of the web in detail and perform the preprocessing of the web graph and propose IO efficient algorithm. Their approach shows significant benefits over original PageRank algorithm when main memory of the system to be worked upon is very small of the order of MBs. But in real scenario main memory size has been increased very much therefore their approach becomes of no use.

Another technique for solving rank of web pages which exploits block structure of web was presented by Kamvar et. al. [6]. Web graph has majority of hyperlinks which link pages on a host to other pages on the same host, many of those that do not link pages to within the same domain. They exploited this structure of web and achieved a speedup of 2 times with this approach. Manaskasemsak et. al. [7] presented a parallel PageRank Computation on a Gigabit PC cluster and showed significant improvement.

PageRank is a highly computational intensive and Cell BE Processor is also designed for computational intensive algorithm. With this idea, Buehrer et. al. [8] implemented PageRank algorithm on Cell BE. But, because of large number of random memory writes, and data transfer between PPE and SPE required by PageRank algorithm, implementation took more time than on single processor Xeon. They also presented a comparison of time taken by different processors to calculate ranks of pages for a particular graph and found that their implementation on Cell BE is 22 times slow in comparison to Xeon processor.

3 Background

3.1 Multicore Processors

A multi-core processor is an integrated circuit to which two or more processors are attached for enhanced performance, reduced power consumption, and more efficient simultaneous processing of multiple tasks. In this section we describe two multi-core architectures- STI Cell BE and CUDA.

The Cell BE [9] is a heterogeneous multi-core chip that is significantly different from conventional multiprocessors. It consists of a central microprocessor called the

Power processing element (PPE), eight SIMD co-processing units called synergistic processor elements (SPE), a high speed memory controller, and a high bandwidth bus interface, all integrated on a single chip. It has a 128 registers of 128 bits. Serje et. al. [10] and Kurzak et. al. [11] show a significant improvement in their implementations.

General purpose computing on the GPU (Graphics Processing Unit) is an active area of research. The GPU contains hundreds of cores that work great for parallel implementation. CUDA (Compute Unified Device Architecture) allows GPUs to be programmed using a variation of C language. GPUs have been proved very efficient for highly computational algorithms [12, 13].

3.2 PageRank

PageRank is an algorithm to determine the relative ranking of web pages. The concept of PageRank is based on an idea that if a page v of interest has many other pages u with pointing to , then the pages u are implicitly conferring some importance to page v. Let C(u) be the number of links which page u points out, and let PR(u) be the rank of page u, then hyperlink $u \rightarrow v$ confers $PR(u)/C(u)$ units of rank to page v.

$$PR_i(A) = (1-d) + d * \left(\frac{PR_{i-1}(T_1)}{C(T_1)} + - - - - - - - - + \frac{PR_{i-1}(T_n)}{C(T_n)} \right) \quad (1)$$

Where, $PR_i(A)$ is the PageRank of page A calculated in i^{th} iteration. $PR_{i-1}(Ti)$ is the PageRank of page Ti which link to page A, calculated in $i\text{-}1^{th}$ iteration. C(Ti) is the number of outbound links on page Ti and d is a damping factor which can be set between 0 and 1.

(dest_id) (in_degree) (Source_nodes)

1	2	3 7
2	4	4 5 7 9
3	3	2 7 9
4	1	1
-	-	-----

Fig. 1. Structure of file containing Web graph

WWW can be considered as a directed graph where each web page is treated as a node of graph and hyperlinks as edges of graph which is known as web graph. Every node of web graph has some number of forward and backward links. A web graph is the input to the PageRank algorithm and stored into a text file as shown in figure 1.

4 Implementation Method

4.1 Implementation of PageRank on Cell BE Processor

Implementation Issues

1. PageRank operates on a huge amount of data. To achieve a better performance gain, all calculations should be done on SPEs. Since SPE's local store is small

(256 KB) and data to be worked upon is large and available at PPE, therefore a large number of DMA transfer need to be done between PPE and SPE producing a bottleneck in performance.
2. Rank of a particular node depends upon any number of nodes in the complete range of nodes. That means data to be worked upon is not continuous (rather scattered in memory arbitrarily). So on the direct input, data level parallelism is not possible. To achieve data level parallelism some modification are required.
3. Since DMA is done on sequential data while requirement in PageRank is of any random node. Thus many DMA operations may be required for smaller data.

Design

We provide data structures followed by algorithm. The data structure design is follows:

1. The web graph is read on PPE and stored into two arrays such that,
   a. Array1 (referred as Node array) contains
      i. First node followed by its in-degree, followed by the source nodes, then
      ii. Second node followed by its in-degree, followed by its source nodes then
      iii. Third node and so on.

      n1| deg1| s1|s2|s3.....|n2|deg2|s4|s5|s6|.....

      here n1, n2 are nodes.
      deg1, deg2 are in-degree of n1, n2 respectively and s1, s2 represent the source node to node n1.
   b. Array2 (referred as Degree array) contains the out-degree of nodes in the indices corresponding to source nodes in Array1

      n1|deg1|d1|d2|d3|....|n2|deg2|d4|d5|d6.....

      here n1 and n2 are same as in array1, deg1 and deg2 are same as in array1 but di represents the out-degree of node si (present in array1)
2. Two arrays V1 and V2 are maintained to keep rank of nodes at ith and (i+1)th. V1 is used as a reference array containing the page ranks as calculated from the previous iterations and used in calculation of V2.

Size of V1 is equal to the size of array1 and array2 while size of V2 is equal to number of nodes. Here thing to be noted is that V1 contains rank of all nodes in the sequence same to the sequence of nodes of array1. That means there is redundancy of rank value of a particular node several times in V1. This is because, a particular node may be the source node of multiple nodes and hence present multiple times in array1. Algorithm proceeds in following way,

1. Array V1 is initialized to 1.
2. for each iteration,
   i. Array2 (or Degree array) and V1 are equally divided among number of SPEs.
   ii. Since number of nodes dedicated to an SPE is large so SPE reads array2 and V1 in parts. In one time SPE reads as many elements of arry2 and V1 as it can accommodate in its local store. Since a particular node, its in-degree, source nodes and their out-degree all are present in array2 and V1 so rank of node can

be calculated easily, and same section of V1 need not be read again for calculations of two nodes. As soon as the rank of nodes is calculated in one time it is sent back to the PPE where it is stored in V2.

iii. As soon as all SPEs calculate the rank of all nodes dedicated to them and V2 is updated at PPE, V1 is updated from V2.

Implementation Details

PPE Operations

Processing starts with PPE by reading the web graph and preparing data structures. PPE spawns pthreads equal in number to SPEs. Each pthread spawns an SPE thread. As soon as SPE thread is created, it starts calculating rank of nodes assigned to it. During this time PPE's pthread goes into a blocking wait giving control to other pthreads of PPE while waiting for a signal by SPE. Figure 2 shows the overall working of PPE and SPEs for one iteration of PageRank algorithm.

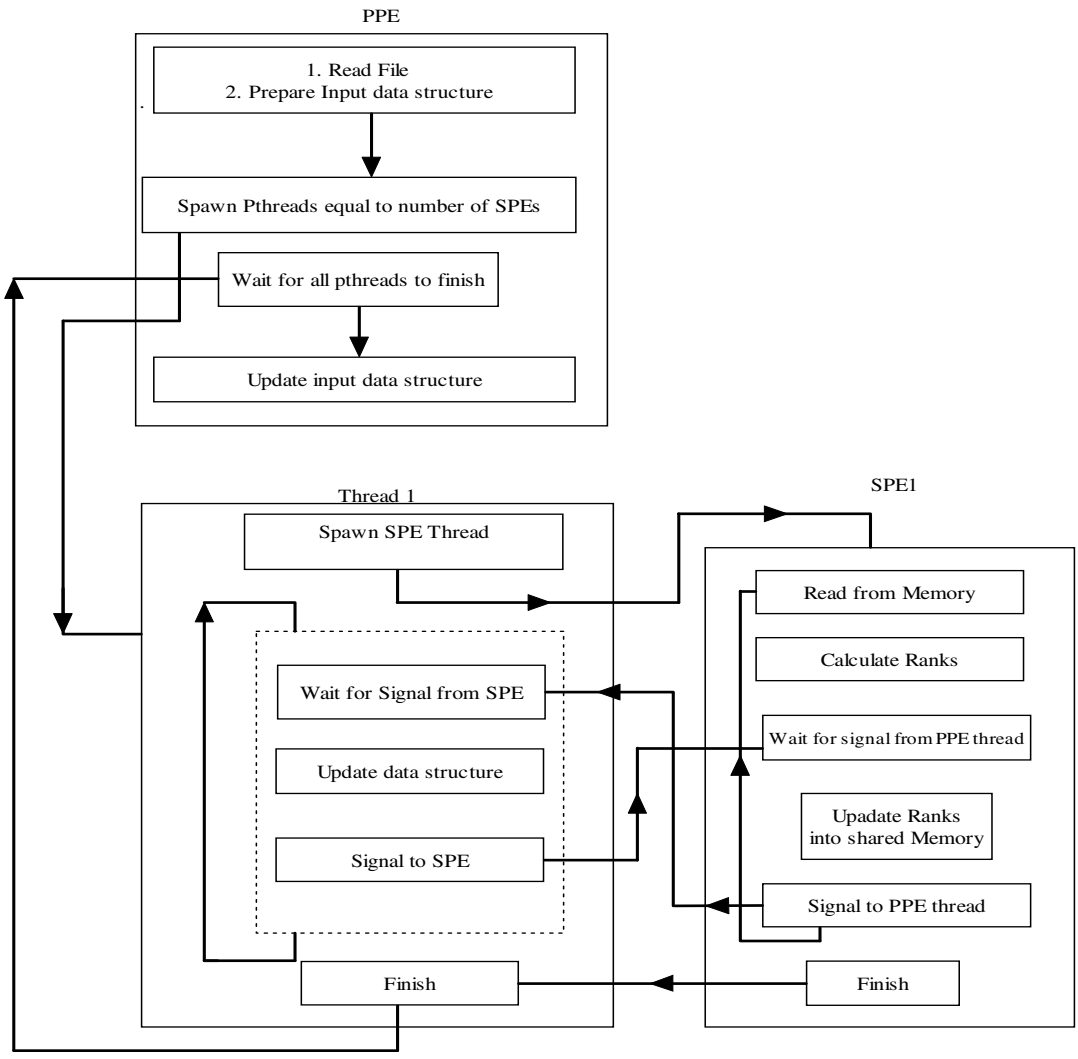


Fig. 2. Overall working of Cell BE processor for PageRank Algorithm

Pthreads update their data structures from the shared memory which is updated by corresponding SPE. Shared memory synchronization is required between pthread and SPE.

SPE Operations

As soon as the SPE thread is created by PPE, SPE starts reading two arrays of degree and rank. Since the task is equally divided among all SPEs so each SPE reads from a particular array location which is determined by the number of SPE. Each SPE starts reading at (SIZE/N)*i location where size of arrays is given by SIZE, N represents total number of SPEs and i is between 0 to N-1 for different SPEs. SPE reads data from memory through DMA operation. Since DMA transfers are limited by 16KB per transfer, therefore only 4096 integer elements can be brought at one time. Thus 4096 elements of degree array and 4096 elements of rank array are brought by two successive DMA transfer. The calculation of PageRank is done with SIMD operation. New rank of nodes present in these 4096 elements is calculated and sent back to PPE by writing into the shared memory. Before writing into shared memory SPE waits for a signal by PPE. After writing into the shared memory SPE sends a signal to PPE about the update of memory and start reading next data from input arrays.

Synchronization

Communication and shared memory synchronization between PPE thread and SPE is achieved with mailbox. The mailbox used is SPE write outbound mailbox. The SPE informs PPE each time after updating shared memory. The status of mailbox at PPE is 1 when mailbox is full and 0 when mailbox has been read by PPE while at SPE its value is 1 when there is no data in it and 0 when SPE writes data in mailbox. The communication synchronization between PPE and SPE is shown is figure 3.

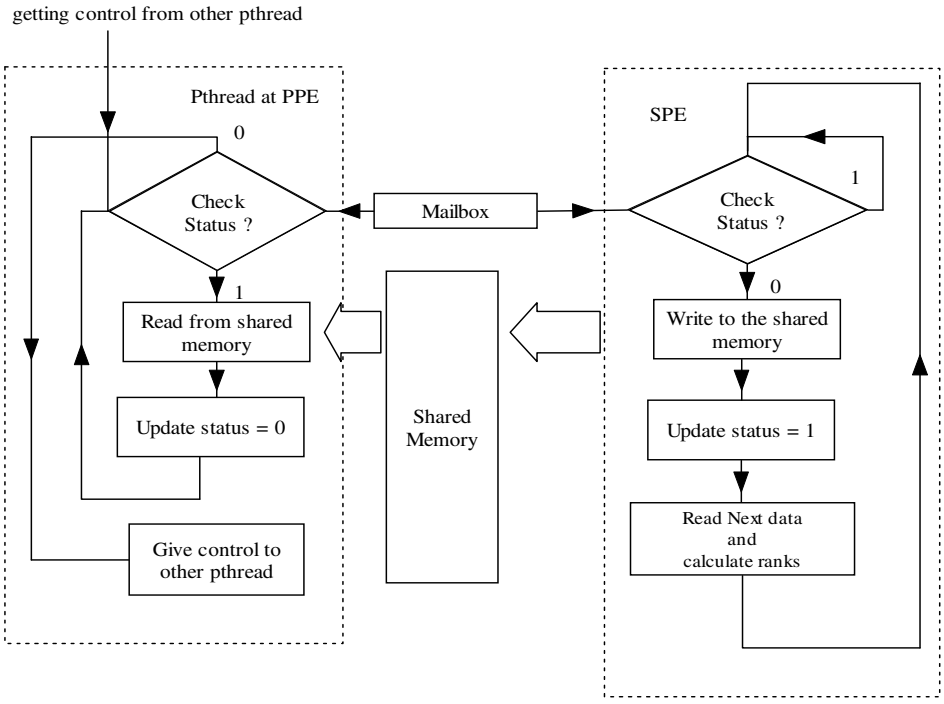


Fig. 3. Synchronization between PPE thread and SPE

4.2 Implementation of PageRank on CUDA

Implementation Issues

1. Architecture of CUDA requires threads of same code path to be running in parallel on a multiprocessor. Execution on CUDA takes place in form of warps. Warps are 32 thread units that are executed on a multiprocessor. CUDA stops all divergent threads within a warp. So if any branch statement is encountered the amount of parallelization gets reduced as divergent threads are no longer running in parallel. Real world scenario web graphs have varying number of in-degrees for nodes. Now processing in the PageRank algorithm works on every node. We have initiated one thread for every node. Since these nodes have varying number of in-degrees the amount of iterations performed by every thread is different which creates a number of divergent threads.
2. Memory synchronization constructs for global memory are not available in CUDA.
3. CUDA does not provide atomic statements for floating point values while PageRank works totally on floating point values.
4. PageRank calculation performs read operations from a wide range of memory area with very less localization of reference. This generates a lot of page faults.

Design

CUDA provides the facility of generating tens of thousands of threads at a time with no generation time. These threads can run on multiprocessors of GPU in parallel. CUDA threads run on different multiprocessors simultaneously, therefore there should not be any data dependency among threads.

Design 1

PageRank algorithm calculates rank of nodes such that rank of a particular node depends on rank of other nodes which have a link with that node. There is no data dependency present between any two nodes of web graph. We create threads equal to the number of nodes. CUDA's limitations disallow this direct approach of solving PageRank algorithm efficiently. The limitations of CUDA with respect to PageRank are as follows:

Number of nodes in a web graph is very large (of the order of billion) and such a large number of threads cannot be generated on GPU (limited by hardware).

Rank of a node may depend upon any number of nodes; therefore a loop to calculate rank of different nodes runs for different number of iterations and hence causing different code paths for threads. This means a large number of threads diverge and they cannot run in parallel.

The above stated problems were further eliminated in the following manner.

Threads corresponding to all nodes should not be spawned at the same time; therefore threads are created in multiple passes in a loop.

To avoid the problem of divergence an extra level of parallelism is added. Instead of calculating rank of a node on a single thread, rank of one node is calculated by as many threads as the in degree of node. We create threads equal to the total in-degree

of all the nodes. For each node, threads equal to its in-degree calculate parallely their respective shares of rank and add that share to the rank of node (which is kept 0 initially). This causes threads to have equal amount of work to be done and hence the code path is same for each thread. This approach requires the rank of a node modified by several threads running in parallel which causes the problem of synchronization among the threads. CUDA does not provide synchronization tools for global memory. Though it provides atomic operations (means once a thread is using a particular memory location no other thread can use that location) for integers only but PageRank requires floating point values. Thus this approach could not be used.

Design 2

The main problem with design 1 is that it hinders the performance because of the variable loop length of each thread. In order to avoid this problem, we run a fixed length loop (say N) on GPU for all threads. Value of N depends upon the web graph to be worked upon. Ideally we want most of the computations to be performed on GPU. GPU prefers threads of similar amount of computation. We select N in such a way that more GPU threads are similar in computation. The idea is to run GPU and CPU parallely such that while GPU is running loop of length N for all threads, CPU calculates partial rank of those nodes which have in-degree more than N by running a loop from N to in-degree of the node.

Figure 4 shows an example of small web graph. Value of N is kept 4. Rank of all nodes is calculated with 4 (or less) source nodes at GPU. Host at the same time calculates partial rank of nodes 4, 5, 7, 8, 11 with those source nodes participating that have index more than N (= 4) . For example for node 4 partial ranks with source nodes 2, 5, 6, 7 is calculated on GPU and partial rank with source nodes 8, 9, 12 is calculated at Host.

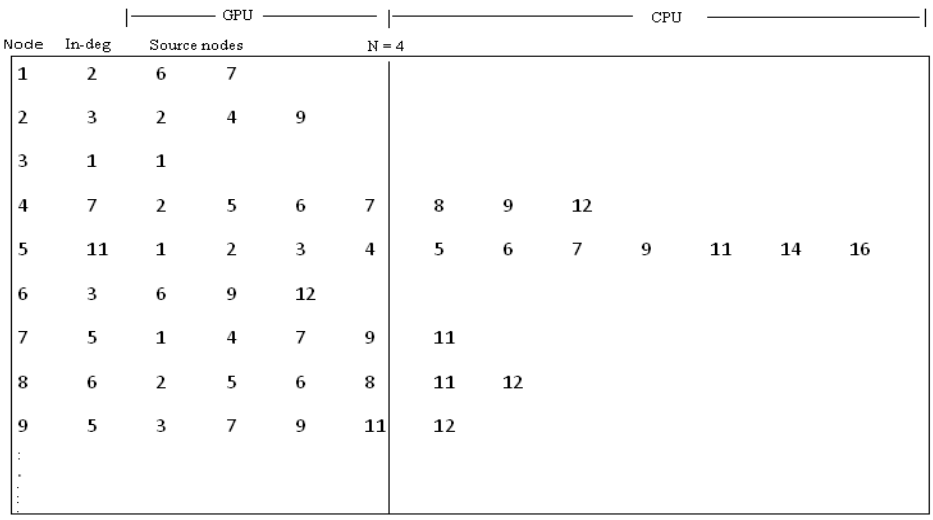


Fig. 4. An example showing the division of work between GPU and CPU

The work of PageRank calculation is divided onto GPU and Host in such a way that when GPU is calculating partial rank of all nodes with the help of N (or less) source nodes, Host at same time calculates partial rank of all nodes with remaining

source nodes (other than N nodes if any). GPU calculates the share of N source nodes which point to a destination node by running loop N times for each thread. Host calculates share of rest of the nodes by running a loop from N to their corresponding in degree. In this way host and GPU calculate partial rank of nodes. These partial ranks are then added and used for next iteration.

Implementation Details

This section presents implementation details of design 2 described in previous section. Program for calculating PageRank consists of mainly two parts, host and kernel. Figure 5 shows the overall control flow of PageRank calculation on CUDA. Execution of PageRank algorithm starts with host program.

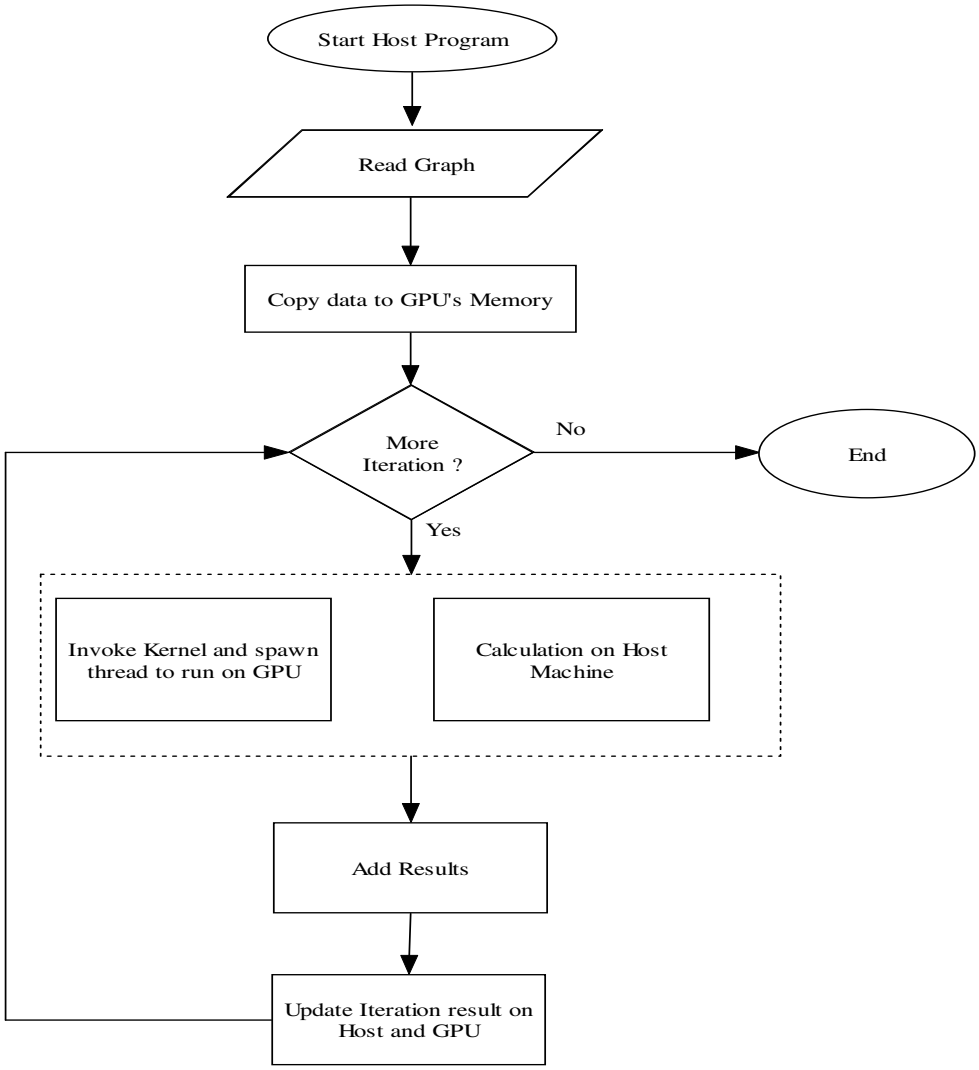


Fig. 5. Program flow of PageRank calculation on CUDA

Host program reads the web graph and copies it to the GPU's memory. Host invokes the kernel to be run on GPU for calculating partial rank with N source nodes. GPU starts processing. Since kernel calls are non-blocking for host, therefore Host

also starts rank calculation of nodes having in-degree more than N. As soon as calculation on both GPU and Host are finished, partial rank of all nodes from Host is brought into GPU memory and added with partial rank calculated at GPU. Thus the new rank of all nodes is calculated and input vectors for next iteration is updated both at GPU and Host.

5 Results

The PageRank algorithm works upon a large web graph in practice therefore; the web graphs which are used for experiments are EU – 2005, CNR-2000 and In-2004 [14]. EU-2005 graph contains 862664 nodes and 19235140 links. CNR-2000 contains 325557 nodes and 3216152 links. Graph In-2004 contains 1382908 nodes and 18534900 links. These graphs are prepared with ubi-Crawler [15].The Cell processor used for execution and testing results is Cellbuzz provided by Georgia Tech. University [16]. GPU used for experiments is GTX 280. Figure 6 shows the comparison of execution times between XEON dual core 2.0 GHz and CELL BE for graphs EU-2005 and In-2004.

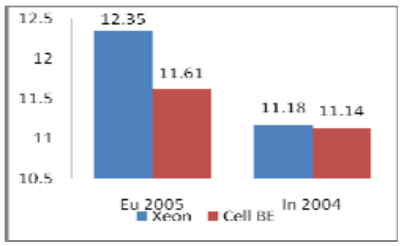


Fig. 6. Comparison of time between XEON and CELL BE processor for graph EU-2005 and In-2004

The speed up obtained by Cell BE over Xeon does not show a marked improvement but when compared to Brehrer et. al. [8]'s implementation of PageRank algorithm on Cell BE, our implementation of algorithm is 22 times faster. We also compare the time taken by Xeon dual core 3.0 GHz and CUDA. It is observed that implementation on CUDA is nearly 2.8 times fast for graph EU-2005. Figure 7 shows the comparison of timing on Xeon and CUDA for graph EU-2005 and CNR-2000.

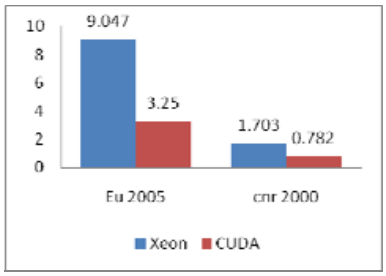


Fig. 7. Comparison of timing on Xeon and CUDA processors for graph EU 2005 and CNR 2000

We also compare the implementation on CUDA with Cell BE, it is found that CUDA performs much better. Figure 8 shows a comparison of timing on Xeon dual core 2.0 GHz, Cell BE and CUDA processor. It shows that implementation on CUDA is 2.6 times faster than implementation on Cell BE.

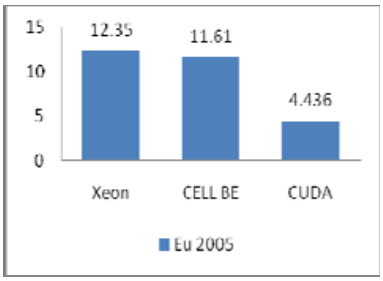


Fig. 8. Comparison of timing on Xeon and CUDA and Cell BE processors on graph EU 2005

6 Conclusion and Future Work

In this paper, we identified the major issues of porting PageRank algorithm on Cell Processor. Possible solutions to these issues were presented. Previous implementation of PageRank on Cell BE resulted in poor performance because of the high data transfer operation between PPE and SPE. A new approach is implemented which reduces the data transfer between PPE and SPE drastically and leads to a better performance. We also presented issues of porting PageRank algorithm on CUDA followed by its implementation on CUDA. It was found that implementation of PageRank on CUDA is performing much better than on Cell.

A better performance in current implementation can be found by dividing the graph in small blocks and then determine the value of N (number of iterations to be run on GPU for a thread) for each block. Performance of PageRank algorithm on multicore processor can be improved by analyzing the web graph in detail and preprocess it according to the restrictions and features of multicore architecture. Another possible solution that can be used for improving performance is to sort the web graph on the bases of in-degree of nodes.

References

1. Brin, S., Page, L.: The Anatomy of a Large-Scale Hypertextual Web Search Engine. In: Proceedings of the 7th International World Wide Web Conference, Brisbane, Australia, pp. 107–117 (April 1998)
2. Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank citation ranking: Bringing order to the web. Stanford Digital Library Working Paper (1998)
3. PageRank Google's Original Sin, `http://www.google-watch.org/pagerank.html`
4. Haveliwala, T.H.: Topic Sensitive PageRank. IEEE Transactions on Knowledge and Data Engineering 15(4), 784–796 (2003)

5. Chen, Y.Y., Gan, Q., Suel, T.: I/O-efficient techniques for computing PageRank. In: Proceedings of the Eleventh International Conference on Information and Knowledge Management, McLean, Virginia, USA, pp. 549–557 (2002)
6. Kamvar, S.D., Haveliwala, T.H., Manning, C.D., Golub, G.H.: Exploting the block Structure of the Web for Computing PageRank. Technical Report CSSM-03-02, Computer Science Department, Stanford University (2003)
7. Manaskasemsak, B., Rungsawang, A.: Parallel PageRank Computation on gigabit PC Cluster. In: Proceedings of 18th International Conference on Advanced Information Networking and Applications AINA, Fukuoka Japan, vol. 1, pp. 273–277 (March 2004)
8. Buehrer, G., Parthasarathy, S., Goyder, M.: Data mining on the cell broadband engine. In: Proceedings of the 22nd Annual International Conference on Supercomputing, Island of Kos, Greece, pp. 26–35 (June 2008)
9. Cell Broadband Engine – An introduction. Cell Programming Workshop, IBM System and Technology Group, April 14–18 (2007)
10. Sarje, A., Aluru, S.: Parallel Genomic Alignments on the Cell Broadband Engine. IEEE Transactions on Parallel and Distributed Systems, December 09 (2008)
11. Kurzak, J., Buttari, A., Dongarra, J.: Solving Systems of Linear Equations on the CELL Processor Using Cholesky Factorization. IEEE Transactions on Parallel and Distributed Systems 19(9), 1175–1186 (2008)
12. Liu, W., Schmidt, B., Voss, G., Muller-Wittig, W.: Streaming Algorithms for Biological Sequence Alignment on GPUs. IEEE Transactions on Parallel and Distributed Systems 18(9), 1270–1281 (2007)
13. Garland, M., Le Grand, S., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., Volkov, V.: Parallel Computing Experiences with CUDA. IEEE Micro 28(4), 13–27 (2008)
14. Laboratory for Web Algorithmics, `http://law.dsi.unimi.it/index.php?option=com_include&Itemid=65`
15. Boldi, P., Codenotti, B., Santini, M., Vigna, S.: UbiCrawler: A Scalable Fully Distributed Web Crawler. Journal of Software: Practice & Experience 34, 711–726 (2004)
16. User guide, Cell buzz, `http://wiki.cc.gatech.edu/cellbuzz/index.php/User_Guide`