2nd Int'l Conf. on Electrical Engineering and Information & Communication Technology (ICEEICT) 2015
Jahangirnagar University, Dhaka-1342, Bangladesh, 21-23 May 2015

A Novel Approach Toward Parallel Implementation of BFS Algorithm Using Graphic Processor Unit

Fahmid Al Farid[2,*], Md. Sharif Uddin[2], Shohag Barman[1], Amirhossein Ghods[2], Sowmitra Das[1], Md. Mehedi Hasan[1]
[1]Dept. of Computer Science & Engineering, University of Chittagong,
Chittagong 4331, Bangladesh.
[2]School of Electrical Engineering, University of Ulsan
93 Daehak-ro, Mugeo-dong, Nam-gu, Ulsan 680-749, South Korea.
[*]fahmid.farid@gmail.com

Abstract—**Graph related algorithms are significant to many of the research areas and disciplines. A very big graph with millions of vertices is common in scientific research works and in the implementations of engineering tasks. Many researcher have tried to implement graph algorithms in parallel architectures, where in this paper, authors have tried to accelerate this implementation in an efficient way. In this paper, a GPU implementation of breadth-first search (BFS) is introduced to accelerate graph algorithm implementation. First, a BFS algorithm is implemented in a sequential environment and then on GPU. Experimental results show that the GPU-based approach of BFS outperforms the same as sequential.**

Keywords—Graphics Processing Units; Breadth-first Search; Central Processing Units

I. INTRODUCTION

In order to improve performance in computing, systems are designed to exploit parallelism. Generally, a GPU (Graphics Processing Unit) can contribute more parallelism than Central Processing Unit (CPU). In case of heterogeneous computing systems, CPU and GPU are utilized both, together in a wide range for their variety of usages [1]. Compute Unified Device Architecture (CUDA) which is newly developed environment, allows to accomplish computational task of Graphics Processing Units (GPU) [2]. GPUs have an antagonistic programming model and are critical to use [3]. Breadth-first Search (BFS) is a basic graph theory algorithm which is largely used to solve various important computational problems. To evaluate the BFS scalability of each iteration with respect to a given graph can be determined by using some standard models [4]. Graphics card which offers massively parallel computing infrastructure having many core special processors is called Graphics Processor Unit (GPU). Primarily it was meant for hurrying computer graphics by unloading works from CPU (Central Processing Unit) and accomplish rendering. A paper implements parallel FEC algorithm on CUDA supported NVIDIA GeForce GTX580 GPU card and achieves performance comparison with CPU based sequential approach. Lastly, they established efficient collective communication functions for the 3D torus structural design of Blue Gene/L that also take benefit of the organization in the problem [5].

Graphics processing units (GPUs) propose a considerable alternative considering great computation power which results after their parallel execution units. Several significant problems in social network analysis, computational sciences, business analytics, and security, are data-intensive and advance themselves in the analyses of graph-theory. Investigation have done to the challenges involved in travelling very large graphs by modeling a breadth-first search (BFS) algorithm for innovative multi-core processors that are expected to become the building blocks of exa-scale systems in the future [6-8]. Researchers are very interested in the problem related to traversing large graphs. Accelerating different types of algorithms such as hamming code decoders, real time feature based algorithms, formant synthesis of haegeum sounds and some others parallel graph processing system on GPU are very prominent in the field of parallel processing [9-11, 12]. Any traversal denotes to an organized technique of discovering all the vertices and edges in a graph. In various graph problems the assembling of vertices using a "breadth-first" search (BFS) is of specific interest. Hypothetical investigation in the random access machine (RAM) model of computation specifies that the computational effort accomplished by an effective BFS algorithm would measure linearly with the quantity of vertices and edges. There are numerous renowned sequential and analogous BFS algorithms in the graph theories and literatures [12]. In the proposed approach we implement the BFS algorithm for large graphs and show the best performance analysis comparatively with other recent works.

The rest of this paper is organized as follows. Section II describes background study, section III presents CUDA implementation, section IV gives details about Mapping Parallel BFS on GPU, and V illustrates experiment result followed by conclusion in section VI.

II. BACKGROUND STUDY

GPU is mentioned to as the device whereas the CPU is the host. Afterwards the kernel, which is the block of code to be handled runs on the GPU, the handled data is returned to the CPU from where they were primarily copied. Figure 2 shows a single precision (SP) GPU platform in which different threads are processes on the blocks B1...Bn. When matched to multi-core CPU which exploits task level parallelism, GPU through its issuing multiprocessors (SMs) displays a transformation in that it uses a data analogous programming outline in which thousands of threads run simultaneously on dissimilar data pieces.

Shown in Figure 1, the flowchart algorithm regarding usage of GPU in parallel process is shown. First the inputs are given to system and then the mathematical process are internally repeated.

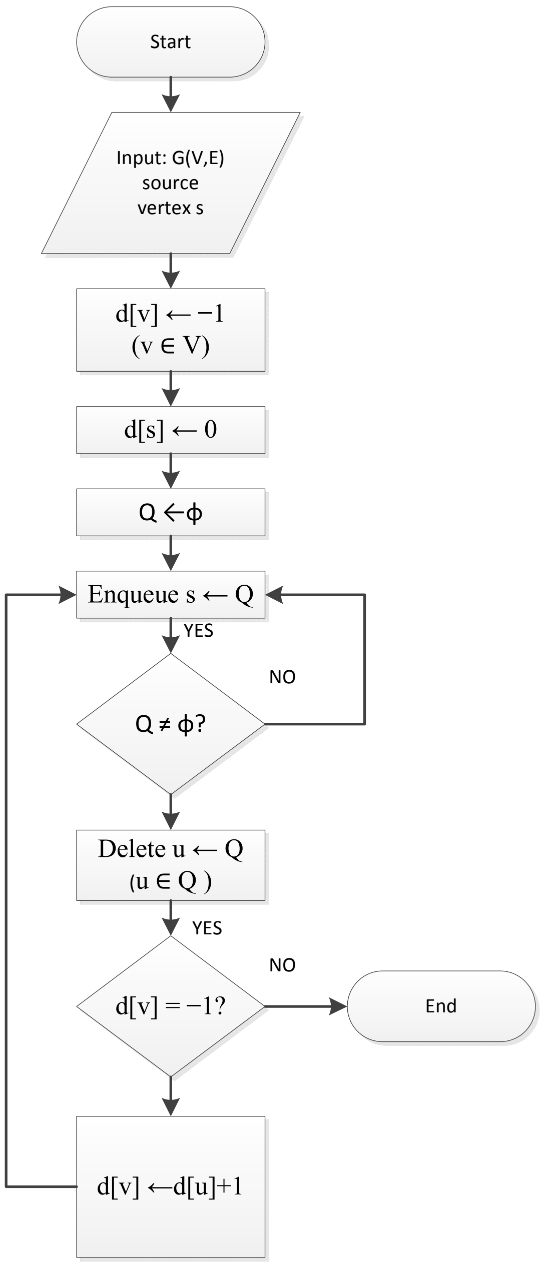

Fig.1. Flowchart Algorithm.

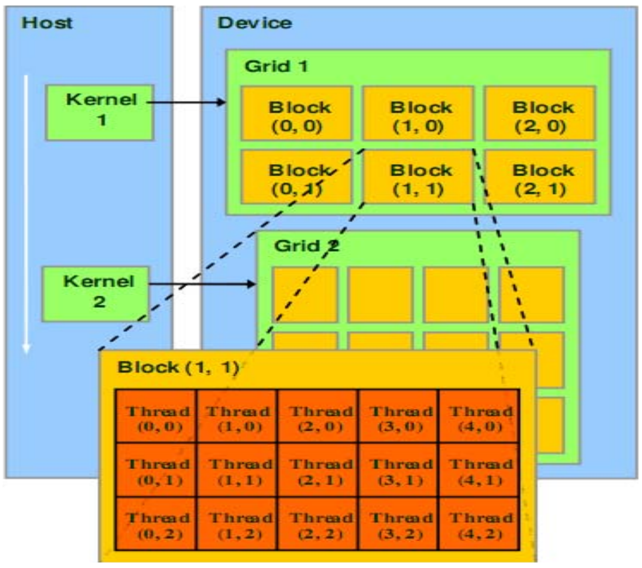

Fig.2. GPU internal hierarchy.

III. CUDA IMPLEMENTATION

In order to evaluate the performance of the BFS CUDA implementation, we run the BFS C program in a CPU and the CUDA program of BFS on a GPU device GeForce GTX 580, respectively. We used CUDA driver CUDART version 4.2 to run CUDA C program in the nvcc compiler. Detailed parameters of the host and the device are given in Table 1 and Table 2.

TABLE I. HARDWARE SPECIFICATION OF CPU

Parameter	Value
Processor	Intel Xeon x5690
Number of Core	6
Thread	12
Clock Frequency	3.46GHz

TABLE 2. HARDWARE SPECIFICATION OF GPU

Parameter	Value
Device	GeForce GTX 580
Total amount of global memory	3072 Mbytes
Total number of CUDA cores	512
GPU clock speed	1.54 GHz
Memory clock rate	2004.00 MHz
Memory bus Width	384bit
L2 cache size	786432 bytes
Total amount of constant memory	65536 bytes
Total amount of shared memory per	49152

block	
Total number of registers per block	1024
Warp size	32
Maximum number of threads per block	1024
Maximum sizes of each dimension of a block	1024x1024x 1024
Maximum sizes of each dimension of a grid	65535x 65535x 65535

IV. MAPPING PARALLEL BFS ON GPU

Heterogeneous computing system that contains a host and a device optimized to mutually process data, a scheduling scheme that maps the appropriate portion of the data for each of the host and the device is needed. A good way to achieve this is to examine the execution time history and the remaining time to achieve the task at hand. Many benchmark applications such as matrix multiplication, two-point angle correlation function and image processing which involves repetitive multi-tasking with input data range above 512 Mb are best executed in the GPU. With the aid of these benchmark applications, the CPU is able to estimate the amount of time needed to complete the best edge for a large graph and will hence decide to export the graph onto the GPU for the execution of the BFS algorithm. The parallel portion of the algorithm is executed on the GPU as kernels, where one kernel is executed at a time and a number of threads execute each kernel. The difference between the mode of instruction execution on GPU and the CPU is as shown in Figure 3 and Figure 4 respectively.

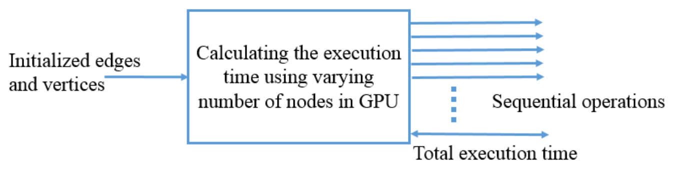
Initialized edges and vertices
Calculating the execution time using varying number of nodes in GPU
Sequential operations
Total execution time

Fig. 3. Work done on BFS by the CPU.

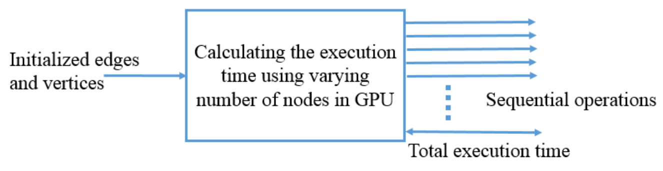
Initialized edges and vertices
Calculating the execution time using varying number of nodes in GPU
Sequential operations
Total execution time

Fig. 4. Work done on BFS by the GPU.

V. EXPERIMENTAL RESULTS

Table 3 shows execution time of the GPU-based approach of BFS and the same CPU-based implementation. The GPU-based approach outperforms the same CPU-based sequential approach in terms of execution time. From the graphical representation it is very clear to understand. In case of speedup here shows some fluctuations. This is because when the number of thread for the parallel execution is similar with the required parallel execution then the speedup is maximum. When the number of required execution in parallel is little bit higher than the number of threads then speedup decrease for some vertices whereas again continuing in increasing with proportion of increasing vertices. Figure 5 and Figure 6 are representing execution time for CPU versus GPU in seconds and milliseconds respectively.

TABLE III. EXECUTION TIME OF CPU AND NVIDIA GPU

Number of vertices	CPU	GPU	Speedup
50	0.00490sec	0.00088sec	5.54x
100	0.01000sec	0.00135sec	7.37x
150	0.03250sec	0.00207sec	27.44x
200	0.04650sec	0.00169sec	33.76x
250	0.06850sec	0. 00202sec	33.76x
300	0.11000sec	0.00231sec	47.45x
350	0.17948sec	0.00433sec	41.43x
400	0.23486sec	0.00602sec	38.95x
450	0.27870sec	0.01024sec	27.21x
500	0.34sec	0.01906sec	17.83x

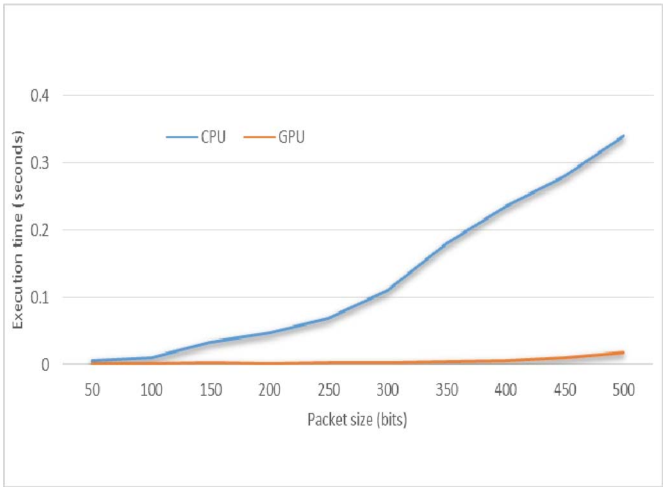

Fig. 5. Execution time for CPU versus GPU in seconds.

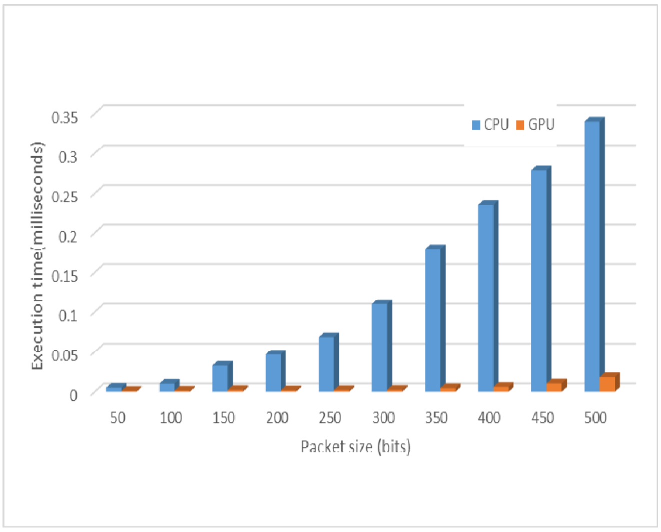

Fig. 6. Execution time for CPU versus GPU in milliseconds.

VI. Conclusions

In this paper, we presented the parallel implementation of breadth-first search which is a highly computational intensive algorithm in the case of large graph. We implemented the BFS algorithm using GPU and compared with the same CPU-based implementation. Experimental results showed that the GPU-based approach achieves higher speedup than the same sequential implementation using CPU for various number of vertices

References

[1] H.J. Choi, D.O. Son, S.G. Kang, J.M Kim, H.H Lee, and C. Hong, "An efficient scheduling scheme using estimated execution time for heterogeneous computing systems," Journal of Supercomputing, pp.1-17, 2013.

[2] M. Sharma, and R.C. Joshi, "Design and Implementation of Cover Tree Algorithm on CUDA-Compatible GPU," International Journal of Computer Applications, pp.163-174, 2010.

[3] H. Pawan, and P.J. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA," Springer Berlin Heidelberg, pp.197-208, 2007.

[4] X. Yinglong, and V.K. Prasanna, "Topologically adaptive parallel breadth-first search on multicore processors," 21st Intl. Conf. IASTED, pp.91-91, 2009.

[5] A. Yoo, E. Chow, K. Henderson, W. McLendon, B. Hendrickson, and Catalyurek, "A scalable distributed parallel breadth-first search algorithm on BlueGene/L," ACM/IEEE Intl. Conf. In Supercomputing, pp.25-25, 2005.

[6] V. Agarwal, F. Petrini, D. Pasetto, and D.A. Bader, "Scalable graph exploration on multicore processors," IEEE Int. Conf. High Performance Computing, Networking, Storage and Analysis, pp.1-11, 2010.

[7] J. Uddin, E. Oyekanlu, C.H. Kim, and J.M. Kim, "High Performance Computing for Large Graphs of Internet Applications using GPU," 2014.

[8] V. Agarwal, F. Petrini, D. Pasetto, and D. A. Bader, "Scalable graph exploration on multicore processors," In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis', IEEE Computer Society, pp. 1-11 2010.

[9] M.S. Islam, and K.I.M Jong-Myon, "Accelerating Extended Hamming Code Decoders on Graphic Processing Units for High Speed Communication," IEICE Transactions on Communications, 97(5), pp. 1050-1058, 2014.

[10] M. Kang, S. Islam, R. Islam, and J.M. Kim, "Accelerating the formant synthesis of haegeum sounds using a general-purpose graphics processing unit," Multimedia Tools and Applications, pp. 1-15, 2014.

[11] J.M Ready, and C.N. Taylor, "Gpu acceleration of real-time feature based algorithms," In Motion and Video Computing, 2007, WMVC'07, IEEE Workshop on IEEE, pp. 8-8, 2007.

[12] J. Zhong, and B.He, "Medusa: A Parallel Graph Processing System on Graphics Processors," ACM SIGMOD Record, 43(2), pp. 35-40, 2014.

[13] A. Buluç, and K. Madduri, "Parallel breadth-first search on distributed memory systems", In Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 65. ACM, 2011.