ISSN:0975-9646

Review of Graph Algorithms on GPU using CUDA Architecture

Trupti R. Desale

Student, Computer Engg. Department, MCOERC Nashik

Pune University, India

***Abstract*— In many practical applications include image processing, space searching, network analysis, graph partitioning etc. in that graphs structure are used to store data with involving millions of vertices. Graph algorithms are fundamental tools in this fields. Hence efficient graph processing is must for application performance. In ordered to increase the efficiency, Graphics Processing Unit (GPU) has been adopted to accelerate graph processing. This device can be treated as an array of Single Instruction Multiple Data (SIMD) processors using CUDA software interface by Nvidia. CUDA device has Massively Multithreaded architecture which makes various threads to run in parallel and hence available computation power of GPU optimally used . Various systems applies parallel algorithm with the use of GPU to accelerate application. Here mainly focus on BFS and Floyd-warshall graph algorithms. BFS algorithm run in parallel approach with different way in that parallelism achieved though vertex, queue are used to store result. Floyd-Warshall algorithm are used to find shortest path between all pair of vertices. Parallel Shortest path Algorithm is developed using block in GPU. Framework are used for graph processing having sequential interference. In this paper, I have focused on papers by various authors for various parallel methods carried out on GPU by using its multithreaded architecture for BFS & APSP .**

***Keywords* — GPU, BFS, Floyd-warshall agorithm, CUDA.**

I. INTRODUCTION

A. Graphs Needs Large Performance Improvement.

Graph processing is one of the integrative and important research area. Graphs are used as data structure in many application such as social networking, Chemistry, image processing, graph partitioning, data mining etc. In this algorithm developer s apply a series of operations on graph edges and vertices to get final result. The operation can be breadth first search(BFS), page rank, shortest path etc. For such graph operations, a sequential methods are available but it is not much efficient with respect to computing time. It is essential to have faster execution such graph operation to reduce complexity of problem. To get high performance of entire system it must require the efficiency of graph processing.

Here studied different parallel methods for graph operations on GPU using CUDA architecture.

B. COMPUTE UNIFIED DEVICE ARCHITECTURE (CUDA)

GPU stands for graphics processing unit which provides high computation power with low cost. They have multiple cores with very high memory bandwidth. For managing and issuing computations on GPU as parallel computing device needs Compute Unified Device Architecture (CUDA). nVIDIA developed CUDA architecture which use for only nVIDIA GPU.

Hardware Model of CUDA Architecture

CUDA Device is collection of various multiprocessors processors each (figure 1). Each multiprocessor works like a Single Instruction, Multiple Data architecture (SIMD). Each multiprocessor has shared memory that memory can accessible only processors which are inside a that multiprocessors. The processors within multiprocessors have set of 32-bit registers, texture and constant memory caches. Texture and constant caches are read only cached memory space and texture cache is optimized for texture fetching operations.

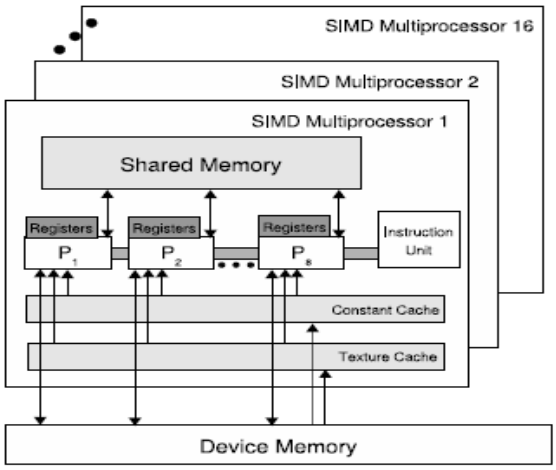

Fig 1: CUDA Hardware Model

Programming Model of CUDA architecture

A CUDA program is organized into a host program, consisting of one or more parallel kernels that are suitable for execution on a parallel processing device like the GPU.As a software interface, CUDA API can be defined as a set of library functions, which could be coded as an extension of the C language. Executable code for the CUDA device generated by a compiler.

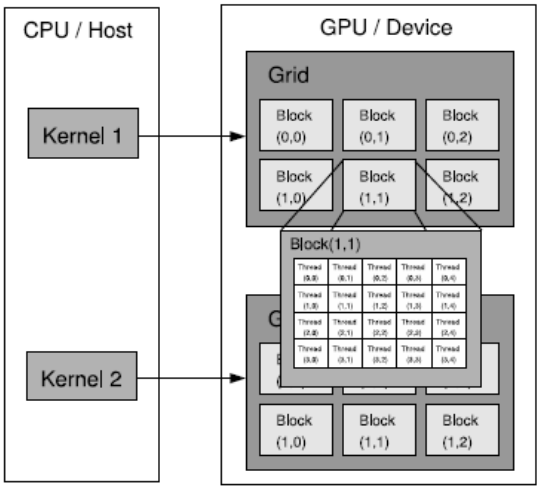

Fig 2: CUDA Software Model

II. Sequence Graph Processing

Sequential fundamental graph algorithms are exist in fast implementation having of number order of vertices and edges. But, for very large graphs, this kind of algorithms becomes impractical. Then, Parallel algorithms are used, for the achieving basic graph operations within practical times but for that required high hardware cost. Bader et al.[1] in this use CRAY supercomputer to perform BFS and single pair shortest path on very large graphs. While using CRAY supercomputer for graph processing this method is fast, but very expensive hardware are used. Venkataraman et al[2]. proposed Floyd's all-pairs shortest-path algorithm in that used more complex blocked versionbto better utilize the cache for large graphs and achieved a speedup of 1.6 and 1.9 over unblocked basic implementation.

III. Parallel Approaches for Breadth-first search

The Breadth first search (BFS) has number of applications in different areas. These include image processing, space searching, network analysis, graph partitioning, automatic theorem proving etc. The BFS problem is, given an undirected, unweighted graph G(V,E) and a it has source vertex S, the BFS aims to find out the minimum number of edges required to reach each vertex V in G from source vertex S. The best time complexity reported for sequential algorithm is O(V+E).

A cost effective parallel platform provided by using graphics hardware to solve many general problems. Many problems are benefited from GPU in speed and parallel processing. Harish and Narayanan proposed accelerate large graph algorithm using CUDA[3]. This method is capable of handling large graphs, unlike previous GPU implementation. Here in BFS, give one thread to every vertex. Frontier and visited, F and X respectively and also integer array, cost, c stores the minimum count of edges of every vertex from the source vertex S. each vertex looks at frontier array if true, then update the cost c of its and neighbors. But some cases like scale free graphs BFS works slower because of the large degree at few vertices, loop inside the kernel which causes the more lookups to device memory and slowing down the kernel execution time.

Vibhav et al. [4] their BFS implementation used vertex compaction process with the help of prefix sum i.e. assign threads only for active vertices. For removing unnecessary threads vertex compaction process is very useful. At particular time,small number of vertices may be active. They carried out experiments on various types of graphs and compared the results with the best sequential implementation of BFS and experiment shows lower performance on low degree graphs. Lijuan luo[5] they proposed effective GPU implementation of Breadth-First Search. They used a hierarchical technique to efficiently implemented a queue structure on the GPU. To reduce synchronization overhead they used a hierarchical kernel arrangement. Their experimental result shows it has same computational complexity as fastest CPU version and achieved up to 10 times speedup.

Hong, kim they implemented a novel wrap-centric [6] programming method that reduces the inefficiency in an intuitive but effective way that exposes the traits of underlying GPU architecture to users. Their experimental result showed significant speedup against pervious studied GPU implementations as well as a multithreaded CPUs.

IV. Parallel Approaches for all pair shortest path

In all pairs shortest path problem (APSP), given an weighted graph G(V, E, W) with positive weights, and that aim is to find out least minimum weighted path from each & every vertex to every other vertex. Floyd-Warshall's, the well known APSP algorithm.

Micikelvicius[7] proposed to solve all pair shortest path using graphics hardware. In that unique distance matrix entry corresponded to each pixel, so to perform Floyd-warshall algorithm used fragment shader. But this algorithm can not work on large graph.

The Harish and Narayanan[3] proposed graph algorithm that is Floyd-warshall's all pair shortest path algorithm requires O(V^3) time and O(V^2) space. Here used a adjacency matrix for graphs and Floyd warshall algorithm implemented using O(V) threads, each running a loop same size inside it. This approach is slower because of sequential access of entire vertex array by every thread. Other approach is to running single source path to every vertex. This methods require O(V) threads where Floyd warshall's algorithm require O(V^2) threads and which creates extra overheads for context switching for threads.

Gary J. Katz and Joseph T. Kider proposed all pair shortest path for large graph[8]. Here graph size problem

due to memory availability is solved. Their approach handles graph size larger than GPU on board available memory this achieved through breaking the graphs into blocks. Convert into blocks in nontrivial on-chip shared memory cache to increase the performance in efficient manner. The algorithm is implemented by blocked formulation. The basic idea for implemented algorithm is revise original Floyd warshall algorithm into a hierarchically parallel methods which can be distributed across on GPU with multiple processors. Matrix is divided into sub blocks with equal size then processed. This implementation of algorithm provides 60-130x speedup over a standard CPU solution O($|V|^3$). 45-100x speedup to blocked CPU implementation that specified by Venkataraman et al. [2] also this methods provides speedup of 5.0-6.5x compared to standard GPU implementation [3]

V. IMPLEMENTATION OF GRAPH ALGORITHM USING FRAMEWORK

Sungpack Hong and Hassan Chafi [9] proposed Green-Marl, a domain-specific language (DSL) in that allow developers to construct their graph analysis algorithms using high level language. But in a Green-Marl in the algorithm must expose the data-level parallelism inherent. also present Green-Marl compiler in which translation of high-level algorithmic described in Green-Marl into an efficient C++ implementation by exploitation of this exposed data level parallelism.

jianlong Zhong and bingsheng he proposed a software framework named Medusa[10] to simplify programming graph processing algorithms on the GPU. Medusa framework provide sequential interference to developer. It provide six device code APIs for developer to write GPU graph processing algorithm. Medusa hides a GPU specific programming details with a small set of system provided APIs. Medusa front end automatically transforms definition device code APIs and user defined data structure into compatible CUDA kernels. Medusa storage component allow developers to initialize the graph structure through the use of system APIs like AddEdge and AddVertex then Medusa runtime component which is responsible for executing the user-defined APIs in parallel on GPU. This Medusa BFS implementation having difference in wrap-centric[6] method, the Medusa applies L threads to vertex has L edges, while wrap-centric methods applies a virtual wrap to vertex. This results, Medusa incurs more memory accesses. Their experimental result shows that On large diameter graphthe performance of Medusa-based algorithm is reduced than that of basic implementation.

VI. CONCLUSIONS

In the paper presented review of the graph algorithms like BFS, APSP those are implemented using CUDA on GPU in parallel approach .It is very important to use efficient data structure for storing the input graph and the results of the algorithms,. Form the overview of the algorithm and experimental result it shows that the for large graph, GPU implementation achieves very great speed up over CPU implementation. There is different way to implement graph algorithm though the use of multithreading. It is very important how programmers make optimum use of multithreading that can be possible on CUDA device which improve the performance of algorithm. GPU has memory hierarchy though the use of different memory that can optimized graph processing in future will be work on.

ACKNOWLEDGMENT

The author wish to thank Matoshri college of Engineering and Research Centre Nasik, HOD of computer department, guide and parents for supporting and motivating for this work because without their blessing this was not possible.

REFERENCES

[1] A. Bader and Kamesh Madduri. Parallel algorithms for evaluating centrality indices in real-world networks. In ICPP ’06: Proceedings of the 2006 International Conference on Parallel Processing, pages 539–550, Washington, DC, USA, 2006. IEEE Computer Society.

[2] VENKATARAMAN G., SAHNI S., MUKHOPADHYAYA S.: A blocked all-pairs shortest-paths algorithm. J. Exp. Algorithmics 8 (2003), 2.2.

[3] P. Harish and P.J. Narayanan, Accelerating Large Graph Algorithms on the GPU Using CUDA, in Proc. HiPC, 2007, pp. 197-208.

[4] Vibhav Vineet and P. J. Narayanan, 2009 “Large graph algorithms for massively multithreaded architecture”

[5] R. E. Sorace, V. S. Reinhardt, and S. A. Vaughn, “High-speed digital- L. Luo, M. Wong, and W.-M. Hwu, An Effective GPU Implementation of Breadth-First Search, in Proc. DAC, 2010, pp. 52-55.to-RF converter,” U.S. Patent 5 668 842, Sept. 16, 1997.

[6] S. Hong, S.K. Kim, T. Oguntebi, and K. Olukotun, Accelerating CUDA Graph Algorithms at Maximum Warp, in Proc. PPoPP, 2011, pp. 267-276M. Shell. (2002) IEEEtran homepage on CTAN. [Online]. Available:http://www.ctan.org/texarchive/macros/latex/contrib/supported/IEEEtran/

[7] MICIKEVICIUS P.: General parallel computation on commodity graphics hardware: Case study with the al lpairs shortest paths problem. In Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA ’04, June 21-24,2004, Las Vegas, Nevada, USA, Volume 3 (2004), CSREA Press, pp. 1359–1365.

[8] G.J. Katz and J.T. Kider Jr., All-Pairs Shortest-Paths for Large Graphs on the GPU, in Proc. Graph. Hardware, 2008, pp. 47-55.

[9] S. Hong, H. Cha_, E. Sedlar, and K. Olukotun, Green-Marl: A DSL for Easy and Efficient Graph Analysis, in Proc. ASPLOS, London, U.K., 2012, pp. 349-362.

[10] Jianlong Zhong and Bingsheng He,"Medusa: Simpli_ed Graph Processing on GPUs".IEEE Transaction on parallel and distributed system, Vol. 25, NO. 6, JUNE 2014