IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

A Review: Parallel Implementation of Shortest path Algorithm on GPGPU

Avadhoot K. Katkar[1], Dr. D. B. Kulkarni[2]

Department of Information Technology, Walchand College of Engineering, Sangli, MS, India[1,2]

Abstract: To find out the shortest paths from a single source to all other vertices's is a common problem in graph analysis. The Bellman¬Ford's algorithm is solves such a single¬source shortestpath (SSSP) problem and better applies to beparallelized for many¬core architectures. TheSequential Bellman-Ford Algorithm is that it very time consuming for large data set & it's time complexity is O(ve). Therefore, frontier based Bellman¬Ford's algorithm significantly reduces the number of relaxation operation, number of iterations and time required for search task. This paper presents a parallel implementation of the Bellman¬Ford algorithm based on frontier propagation. General Purpose Graphics Processing Units (GPGPU) based parallel computing is the best alternate way to speed up the search task. This paper gives detailed information about frontier based Bellman¬Ford's algorithm to solve optimization problem & improve the performance of GPU based computing.

Keywords: Bellman-Ford, SSSP, GPGPU, Parallel graph algorithms.

I. INTRODUCTION

Single source shortest path problem finds application in large domains of the scientific and real world. Common applications of these algorithms are in network routing, VLSI chip design, robotics and transportation, artificial intelligence, social networks, data mining they are also used for directions between physical locations like in google maps. All of these applications generally involve positive weights but some applications are there weights can be negative. Currency exchange arbitrage and some other areas, edge represents something else than the distance between two entities. In such application BellmanFord algorithm [2] can be used. BellmanFord operates on all vertices's independently. Each vertex maintains its distance to the source. on each iteration, a vertex checks each adjacent vertex, updating its own distance to the source if it finds any shortest path. This operation is repeated until the distances converge. BellmanFord algorithm is applicable on graphs with negative weights and can also detect negative cycles where the majority of algorithms fail (e.g Dijkstra's algorithm). BellmanFord's is also used in power allocation in wireless sensor networks, systems biology and regenerative braking energy for the railway vehicle.

The algorithm requires many iterations and each iteration is based on the previouslycomputed results so it is well suitable for parallelization. Recently several packages have been developed for processing large graphs on parallel architectures such as parallel Boost graph library, Pregel & Pegasus. The proposed solution in the paper for GPGPU[3]. Therefore, GPGPU based parallel computing is the best alternate way to speed up the search task. Experimental results have been conducted on graphs of different size to compare the proposed approach with the most representative sequential and parallel implementations for solving the SSSP problem. The time complexity is O(ve) where v=vertices's & e= edges.

II. LITERATURE REVIEW

Bellman-Ford was introduced by Richard Bellman and Lester Ford Jr. in 1958. Since then several modifications and improvements were made to this algorithm.

F. Busato and N. Bombieri [1] has presented a parallel implementation of the Bellman-Ford algorithm based on frontier propagation which is different from all other approaches in the literature. The idea behind is that it uses a frontier data structure in which all and only active nodes are processed in parallel. The parallel processing of active nodes does preserve the semantics of the algorithm.

P. J. Martine et. al. [4] has proposed different CUDA solutions for the SSSP problem by Considering adjacency lists and matrices. Probably, the most well-known algorithm solving this problem for the case of graphs with nonnegative edges was given by Dijkstra. Author's, has solved the problem by using Dijkstra's algorithm adjacency lists of a graph is made up of three arrays i.e vertices's, edges and weights. In the case of

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

adjacency lists,it is difficult to conceive a method to allow threads to collaborate when reading from global memory.
On the opposite, when adjacency matrices are used, threads must visit every element of each column or row, and so, threads can cooperate to bring elements of arrays to shared memory.

H. Ortega-Arranz et. al. [5] has solved SSSP problem by using Dijkstra's algorithm. The complexity time of this algorithm is O(v2). In this Dijkstra's approach it parallelizes the internal operations of the sequential Dijkstra algorithm. The idea behind is that to parallelization of a single sequential Dijkstra algorithm resides in the inherent parallelism of its loops. For each iteration of Dijkstra's algorithm, the outer loop selects a node to compute new distance labels. Inside this loop, the algorithm relaxes its outgoing edges in order to update the old distance labels, that is the inner loop. Parallelizing the inner loop implies to traverse simultaneously the outgoing edges of the frontier node.

A. Davidson et al. [6] have proposed three different work efficient solution to solve SSSP problem: Workfront Sweep, Near-Far Pile, and Bucketing. Each method has a different approach. By using Workfront Sweep implements queue based Bellman-Ford's algorithm. The main advantage of this method is that it reduces redundant work due to duplicate vertices's during the frontier propagation. Near-Far Pile method in which splits the work queue of vertices's into two sets known as one is Near Set with distances less than iΔ to be processed next, and second with distance outside that range i.e Far Pile, differed for later processing. Where as incremental weight is denoted as Delta (Δ). First go with the near set. In which traverse all edges from the vertices's in the near set and split the resulting vertices's that have been updated into two piles. It will append elements outside of our range to the end of the far pile, and begin the next iteration only processing the near set, after that check the far pile for valid elements outside of our range to the end of the far pile, and begin the next iteration only processing the near pile. Once run out of elements to process in the near set, then check the far pile for valid elements, compact all duplicates, and run another split with an updated range ((i+1)).

In all Δ cases, the split primitives can be performed quite cheaply with a simple scan and modified compaction routine. The main thing of this method's efficiency is that in many cases, unprocessed work in the Far Pile can be discarded as closer vertices's are processed, therefore minimizing the number of times we must touch data in the Far Pile. At the time of performing the split we merely append data to the end of the pile, requiring no extra data movement. Thus the Far Pile data is only touched when it will run out of work queue items. There are two costs to adding this functionality. First they are reducing the amount of available parallelism within the work queue. Second, they add the overhead of a split on every iteration. Bucketing method used to implement the Delta-stepping algorithm. But the Delta-stepping algorithm is not supported for SIMD architecture because it requires dynamic data structures for buckets. Bucketing method is slower than other two methods.

K. Kelley and T. Schardl [8] has proposed parallel Gabow's scaling algorithm [9]. The proposed algorithm performs well in practice on random graphs, outperforming a simple Dijkstra implementation on multi-core CPU's.

J. Crobak et al. [9] has presented a multi- threaded implementation of Thorup's algorithm for undirected graphs to solve SSSP problem. Thorup's algorithm is naturally suited for multi- threaded machines since many computations can share a data structure within the same process.

Table1 shows that the time complexity of each algorithm.

Algorithm	Time Complexity
Dijkstra	O(v2)
Bellman-Ford	O(ve)

Table1: time complexity SSSP algorithms. v represents number of vertices's and e represents number of edges.

II. COMPUTATIONAL RESULTS

In paper [1], Author has used SSSP as a combinatorial optimization problem. The objective of SSSP is to find out the shortest path starting from single source and visit all the vertices's in the graph such that the distance should be minimized. Author's experimented their results with different dataset likes 1000EWD, rome99, 10000EWD, NYC. All the experiment conducted on PC with RAM 8GB, NVIDIA GeForce GTX 730 device, which has 384 CUDA Cores, Compute capability 3.5, Intel i7 Core Processor.

IJARCCE

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

Table 2 shows result analysis of frontier propagation approaches of SSSP problem.

Dataset	Number of Vertices	Number of Edges	Frontier propagation
1000EWD	1K	16K	0.0004 s
Rome99	3K	8K	0.004s
10000EWD	10K	1.2M	0.003 s
NYC	2.64M	7.33M	0.3476 s

Table 2. Performance of frontier propagation approache, time measured in terms of seconds (s) From Table 2. Author have calculated approximation values and analysed its efficiency with different approaches like Work-Front Sweep, Near-Far Pile and Frontier propagation. i.e is shown in the Table 3. However, large number of iteration required to reach optimal solutions for large datasets. It can be obtained with minimum number of iteration with parallelization. Therefore, shortest path would get in the reasonable amount of time. Now a day's SSSP can be applied to a large number of data instances than used in this paper.

Table 3. Performance comparisons of different Approaches, time measured in terms of second (s).

Dataset	Number of Vertices	Number of Edges	Work-Front Sweep/ Near-Far Pile	Frontier propagation
asia.osm	12.0M	25.4M	12.7 s	0.0002 s
msdoor	415K	20.6M	0.206 s	0.0045 s
usa-road-d.cal	1.9M	4.7M	4.6 s	0.0031 s
Circuit5M_dc	3.5M	19.2M	0.240 s	0.3476 s

The graphical representation of table 3 is shown in following figure 1.

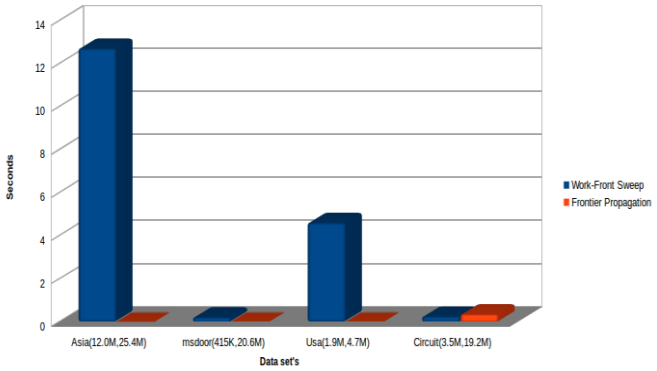

Fig 1. analysis of different approaches of SSSP problem.

III.CONCLUSION

This study reveals the effectiveness of proposed algorithm over as mentioned in literature solutions for solving the SSSP problem. For combinatorial optimization problems such as SSSP, many of the sequential approaches have shown the ability to obtain good results. Recently, many researchers build parallel approaches to gain more improvement in obtaining solutions over sequential approaches in order to handle large scale graphs. Based on experimental results of the above papers, GPU-based computing provides high-quality results than other parallel architectures. Parallel implementation of frontier based Bellman-Ford's Algorithm gives better result on GPU platform. Experimental results will be conducting on graphs of different size to compare the proposed approach with the most representative sequential and parallel implementations for solving the SSSP problem.

REFERENCES

[1] S. F. Busato and N. Bombieri, "An Efficient implementation of the Bellman-Ford Algorithm for Kepler GPU Architectures," vol. 27, no. 8, pp. 2222–2233, 2016.
[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, Introduction to Algorithms. Cambridge, MA,USA: MIT Press, 2009.
[3] S. Che, M. Boyer, J. Meng, D. Tarjan, J.W. Sheaffer, and K.Skadron, "A Performance Study of General-Purpose Applications on Graphics Processors Using Cuda," J. Parallel Distributed Computing, vol. 68, no. 10, pp.1370-1380, 2008.

ISSN (Online) 2278-1021
ISSN (Print) 2319 5940

International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 6, Issue 5, May 2017

[4] P. J. Martin, R. Torres, and A. Gavilanes, "CUDA solutions for the SSSP problem," in Proc. 9th Int. Conf. Comput. Sci.: Part I, 2009, pp. 904–913.
[5] H. Ortega-Arranz, Y. Torres, D. Llanos, and A.Gonzalez-Escri-bano, "A new GPU-based approach to the shortest path problem," in Proc. Int. Conf.High Perform. Comput. Simul. 2013, pp. 505–511.
[6] A. Davidson, S. Baxter, M. Garland, and J. Owens,"Work- efficient parallel GPU methods for Single-source shortest paths," in Proc. IEEE 28th Int. Parallel Distrib. Process. Symp. 2014, pp. 349–359.
[7] P. Harish and P. Narayanan, "Accelerating large graph algorithms on the GPU using cuda," in Proc.14th Int. Conf. High Perform. Comput., 2007, pp.197–208.
[8] K. Kelley and T. B. Schardl, "Parallel single-source shortest paths," MIT computer science and artificial intelligence laboratory, internal report, 2010.
[9] J. R. Crobak, J. W. Berry, K. Madduri, and D.A. Bader, "Advanced shortest paths algorithms on a massively-multithreaded architecture," in Proc. IEEE Int. Parallel Distrib. Process. Symp., 2007,pp. 1–8.

BIOGRAPHIES

Avadhoot K. Katkar Post-graduating student for master degree for information technology in Walchand College of Engineering, Shivaji University, MH, India. Researching in High-Performance Computing.

Dr. D.B. Kulkarni Professor of information technology in Walchand College of Engineering, Shivaji University,MH, India interested in High- Performance Computing.