IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 6, June 2018

A Comparative Analysis of Shortest Path Algorithms on GPU using OpenCL

Dharmendra Sansaniya[1], Sanjay Keer[2]

Abstract: Shortest path algorithms find applications in wide domains. But to provide result for complex graphs in real time is a challenging task. So in this paper four shortest path algorithms namely Dijkstra's algorithm, Floyd Warshall, Bellman Ford and Jhonsons algorithm are studied and analyzed to detect parallelism in them and the parallelized version of all three is implemented using parallel computing framework OpenCL. It is found that Bellman Ford and Floyd Warshall contains fine grained parallelism while Jhonsons has less parallelism.

Keywords: Bellman-Ford, Dijkstra, Floyd Warshall.

I. INTRODUCTION

The shortest path problem refers to the problem of finding the shortest path or minimal cost route from a specific source to a particular destination. Generally, graphs are most widely used for representation of such problems. Shortest Path problem basically deals with graphs and in specific, with problems belonging to weighted directed graph category. It finds applications in large number of domains such as, in network routing protocols, VLSI design, robotics and intelligent transportation systems. A graph is a finite set of vertices and edges represented as, G (V, E). Vertices are connected using edges. A directed graph is a graph in which each edge could be traversed only in a given direction. To calculate the distance between any two vertices (also called nodes), edges are given some values called weights (or cost). These weights measure the cost or distance between any two nodes via different possible edge sequence. The shortest path analysis is not restricted to the shortest distance between the source and destination, but also refers to other measurement units such as time, cost and the capacity of the link. There are number of optimization and high performance computing techniques like vectorization, loop unrolling using parallel heterogeneous computing platform. In this paper a heterogeneous computing platform which provided parallelism on multiple devices is explored for implementing shortest path algorithms.

OpenCL is an open source computing platform which performs following steps:
First it identifies platforms available on the device we have run the implementation.

- o In this paper intel and NVIDIA platform is used.
- Second it identifies devices available on available platforms.
- o In intel platform we have 2 devices available Intel i5 CPU and Intel graphics card.
- o In NVIDIA platform we have NVIDIA Geforce graphics card.
- Then for the devices we want to exploit a context is created. And devices in same context can only be synchronized. So if we want to synchronize 2 devices both need to be in same context.
- Then in fourth step command queues are created for every device independently. As we want to give parallel commands to both the devices. 2 command queues are created.
- Parallel kernels are created using OpenCL for both the algorithms. And each kernel is processed on separate devices in same context.

Flynn's Taxonomy: When looking at the world of parallel computing, there are several ways in which you can classify a parallel computing machine. These classes could be based on the hardware architecture of the machine. For example, Flynn (1972) presents a method of classifying a parallel computing machine based on its hardware architecture, and therefore, programmability. Flynn's classification is based on two separate dimensions, Instruction and Data. Furthermore, these dimensions are split into two states, Single or Single Instruction, Multiple Data (SIMD) processor, capable of performing thousands of identical instructions on any number of pieces of data. Single Instruction, Single Data (SISD) Only a Single Instruction is being executed by the CPU during any given clock cycle. Only a Single Data is being used as input for the current instruction during any given clock cycle. "Represents most conventional computing equipment available today" (Flynn, 1972).

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 6, June 2018

In recent years, the advent of GPGPU has popularized the use of parallel computing on the GPU in achieving significant performance gains on a relatively cheap hardware device. GPGPU is a method of using the GPU to perform computations that would usually be executed by the CPU, rather than performing calculations to handle computer graphics, as is their traditional use. When the GPU is used for GPGPU, it can be viewed as a coprocessor to the CPU, offloading complex tasks that the GPU can tackle in parallel. GPGPU provides an extremely cost effective alternative for parallel algorithms that would normally be exclusive to supercomputers, with a low-end CUDA enabled GPU costing approximately ↔25 compared to a super computer such as IBM's Blue Gene system at ☆1.3million. Multiple GPUs in a single system can be utilised for a single problem, often increasing the performance of parallel applications.

This project does not utilise multiple GPU however. The applications of GPGPU are far reaching and include some of the following: Graph Theory, Ray Tracing, Matrix and/or Vector Operations, Signal Processing, Image Processing, Speech Recognition., Physics Simulations, Medical Computation. Multiple GPGPU APIs exist to utilise the GPU for parallel computing. Popular APIs include NVIDIAs CUDA, OpenCL Khronos (2011) and DirectX's Direct Compute platform (Microsoft, 2012). This research focuses solely on NVIDIAs CUDA API. Each have their advantages and disadvantages, but all provide a solid parallel computing API to utilise the powerful hardware of modern GPU. Modern graphics cards have a specialised hardware architecture that can be represented as a parallel computer. Unlike traditional graphics cards, GPUs such as NVIDIAs 580GTX are equipped with 16 multiprocessors, each with 32 cores, providing an impressive 512 cores. Each core has access to a global bank of memory, much like the Random Access Memory (RAM) on a PC, as well as a block of shared memory per multiprocessor which provides fast storage that can be used to share data between parallel processes. The potential of GPGPU is extremely great, given this unique hardware architecture that can provide great performance benefits to algorithms at a relatively low cost.

II. INTRODUCTION TO OPENCL

OpenCL is an open standard framework for parallel programming composed of several computational resources (CPU, GPU and other processors). Thus one can achieve considerable acceleration in parallel processing if it utilises all the computational resources. The main advantage with OpenCL is its portability as it provides cross vendor software portability [11]. OpenCL framework comprises of following models:

A. OpenCL Platform Model

High level representation of heterogeneous system is demonstrated by Platform model as shown in Fig. 1. It consists of a host and OpenCL device. A host is any computer with a CPU and standard operating system. OpenCL device can be GPU, DSP or a multi-core CPU [11]. OpenCL device is collection of compute units which further composed of one or more processing elements.

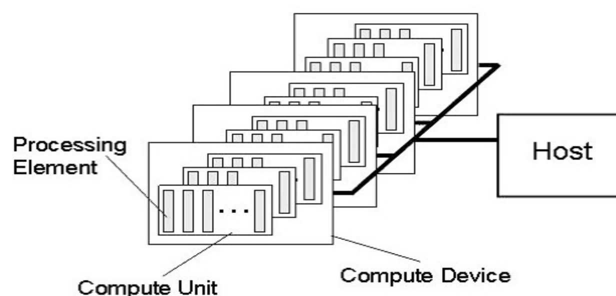


Figure 1: OpenCL Platform Model

Processing elements within a compute unit will execute same instruction sequence while compute units can execute independently. Different GPU vendors follow different architectures but all follow a similar design pattern which is illustrated in Fig. 2.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
IJARCCE
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 6, June 2018

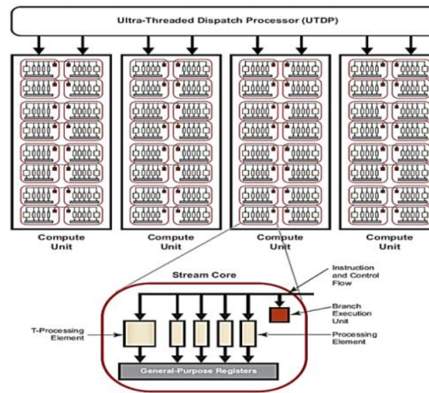


Figure 2: AMD GPU Compute Device

B. OpenCL Execution Model

OpenCL execution model define how the kernel execution takes place and how kernel interact with host and with other kernels and it comprises of two components: kernels and host program. Kernels are further categorized into two types: OpenCL kernels and Native Kernels. Kernels execute on OpenCL devices and host execute on CPU (host system).

Workgroups evenly divide the index space of ND Range in each dimension. And the index space within a workgroup is referred as local index space which is defined for each work item. Size of index space in each dimension is indicated with uppercase and ID is indicated using lowercase.

A work-item can be uniquely identified by its global ID (g_x, g_y) or by the combination of its local ID(l_x, l_y) and work group ID (w_x, w_y) as shown in relation below :

$$g_x = w_x * L_x + l_x$$
$$g_y = w_y * L_y + l_y$$

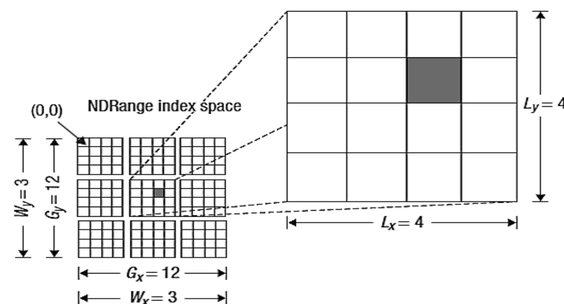


Figure 1: Relation between global ID and local ID, work-group ID in 2-D index space

NDRange index space of size G_x by G_y (12x12) is divided into 9 work-groups, each having size 3x3. The shaded block has a global ID of (g_x, g_y) = (6, 5) and a work-group plus local ID of (w_x, w_y) = (1, 1) and (l_x,l_y) = (2, 1).

C. OpenCL Memory Model

OpenCL memory model defines different regions of memory and how they are related to platform and different execution model; this is shown in Fig. 4. There are generally five different regions of memory:

Host memory : This memory is limited to host only and OpenCL only defines the interaction of host memory with OpenCL objects.

Global memory : All work items in all work groups have read/write access to this region of memory and can be allocated only by the host during the runtime.

Constant memory : Region of memory which stays constant throughout the execution of kernel. Work-items have read only access to this region.

Local memory : Region of memory is local to work group. It can be implemented dedicatedly on OpenCL device or may be mapped on to regions of Global memory.

Private memory : Region that is private for work-item.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
IJARCCE
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 6, June 2018

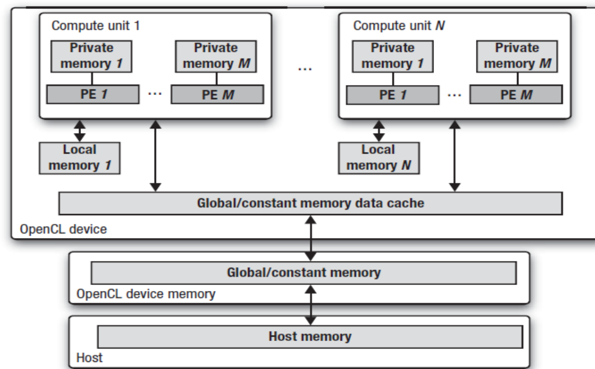
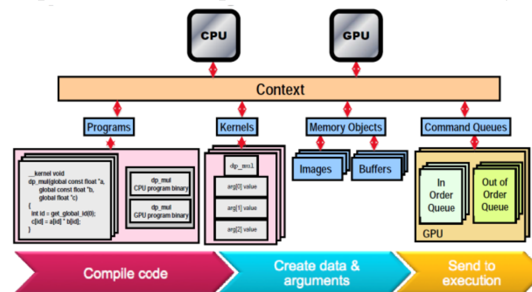


Figure 2 : OpenCL Memory Model

D. OpenCL Programming Model

A programmer can freely combine any of the programming models available in OpenCL. OpenCL is basically defined with two programming models: data parallel and task parallel model. However hybrid model can also be used.



III. INTRODUCTION TO GPU ARCHITECTURE

To make efficient utilization of resources one need to be fully acquaint with architecture of those resources especially like GPU. GPU comprises of one or more compute units and compute units further consists of stream core processors. Each stream core processor consists of some ALU's and special function unit. In VLIW5 architecture it consists of 4 ALUs and 1 special function unit. All the stream core processors within a compute unit share local memory and global memory is shared by all the compute units. The architecture of GPU on which implementations are tested i.e. AMD HD 6450 is shown in figure 5.

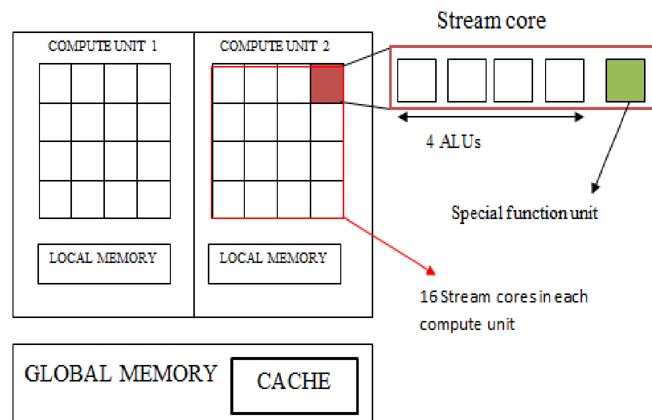


Figure 3 : Architecture of AMD HD 6450

Copyright to IJARCCE

IJARCCE
ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 6, June 2018

AMD HD 6450	
COMPUTE UNITS	2
STREAM CORES PER COPUTE UNIT	16
ALU PER STREAM CORE	4
SPECIAL FUNTION UNIT	1
LOCAL MEMORY	32KB
GLOBAL MEMORY	2GB

Table 1: Configuration of AMD HD 6450

IV. RELATED WORK

In [4] three parallel friendly and work-efficient methods to solve this Single-Source Shortest Paths (SSSP) problem: *Workfront Sweep*, *Near-Far* and *Bucketing*. All of these methods do much less work than traditional Bellman Ford method. All these techniques for shortest path algorithms are implemented on GPU. In this paper Dijkstra algorithm is compared with BellmanFord algorithm on which above mentioned three techniques are applied. All these algorithms are studied for different data structures and traversal techniques. In [10] Floyd Warshall and Dijkstra algorithm are compared by applying divide and conquer on FW to make use of multi GPU cluster using OpenCL. Bellman Ford is introduced by Richard Bellman and Lester Ford Jr. in 1958 since then several modifications and improvements were made on this algorithm. One of the famous modifications include Yen's modification in 1970 [5].Other modifications include topological scan algorithm for Bellman Ford [2] in 1993, which outperforms the standard algorithm in most of the cases. A hybrid implementation of Bellman Ford and Dijkstra's algorithm is given which is asymptotically better than Bellman Ford in [7]. In 2001, A.S. Nepomniaschaya presented a STAR procedure for Bellman Ford on a parallel system with vertical data processing (STAR- machine) [3] and managed to reduce the complexity to $O(n^2)$. In 2011, Michael J. Bannister and David Eppstein [1] proposed a randomized variant of algorithm which is improved by a factor of 2/3 over Yen's modification(1970) [4,5]; they have termed this speedup as randomized speedup.

Several parallel implementations on GPU for SSSP algorithms were proposed. Aydın Buluc, John R. Gilbert and Ceren Budak [8] have proposed parallel implementations for SSSP and APSP using CUDA. A CUDA implementation for Bellman_Ford is given in [13] and by making algorithm suitable for parallelism they have got speedup of about 10x.

Recently, Andrew Davidson [9] have presented several work efficient methods for SSSP problems and got considerable speedup over serial implementation and other traditional GPU implementations also.

V. BELLMAN FORD

Consider a graph G(n,E,V) where, n is the number of vertices, E is the set of edges and V is the set of vertices. Adjacency matrix representation of graph is used here, as it is well suited for GPU. Here, Cost is the adjacency matrix for graph. Initially, Dist will contain direct edges from the source's'. Afterwards, Dist[v] of 'k^{th}' iteration means distance from 's' to 'v' going through no more than 'k' intermediate edges. Finally, after successful completion of algorithm Dist will contain the shortest path to all the vertices 'v' in V from source's'. For each edge (u,v) in set E, Relax(u,v) is called (n-1) times. So, Relax () is called E (n-1) times, thus majority of time of the algorithm is spent in this procedure. The algorithm for Bellman Ford is illustrated in Algorithm 3.1.

```
Algorithm BellmanFord (s,Dist,Cost,n)
{
for i=1 to n do
Dist[i] = Cost[s,i];
End for
for k=1 to n-1 do
for each (u,v) in E do
Relax(u,v)
End for
End for    }
```

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

IJARCCE

International Journal of Advanced Research in Computer and Communication Engineering

ISO 3297:2007 Certified

Vol. 7, Issue 6, June 2018

```
Relax (u,v)
{
if Dist[v]> Dist[u] + Cost[u,v]
Dist[v] = Dist[u] + Cost[u,v]

}
```

Time complexity of above algorithm if adjacency matrix representation is used will be $O(n^3)$.
All pair shortest path using bellman ford algorithm could also be calculated if above algorithm for all the vertices in the graph is called.

```
For each s in V
Call BellmanFord(s,Dist,Cost,n);
End for
```

Identified Parallelism:
The only issue arises here is how to calculate minimum of all these 'n' values. So rather than calculating the minimum which will increase the time of algorithm we will synchronize the write operations on $Dist^k[v]$ for all 'u' such that minimum value resides in $Dist^k[v]$ at the and of Relax() procedure. This issue is referred as write-write consistency.

VI. FLOYD WARSHALL

APSP is a fundamental problem in graph theory. Floyd-Warshall (FW) is a well known algorithm for its solution. FW sequential implementation uses three nested loops. Consider a weighted graph G (V, E) stored using adjacency matrix representation by a weight matrix W of order N*N where N is number of vertices in G.where, $w_{ij} \epsilon$ W for all (i,j) ϵ E. This matrix W contains zero for diagonal elements as both corresponds to same vetex. And infinity for the vertices which are not connected directly and weight is there for vertices which are connected directly or edges available in graph.

Floyd algorithm:

```
For(int k=0;k<N;k++)
For (int i=0;i<N;i++)
For(int j=0;j<N;j++)
```
$$d_{ij}^{(k)} \leftarrow \min\left(d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$

Identified parallelism:
As it is clear from the above algorithm that value of kth iteration depends on k-1 so this loop contains dependency so it cannot be removed to perform parallelism. But rest 2 loops can be called in parallel for N2 threads using OpenCL.

VII.DIJKSTRA'S ALGORITHM

Dijkstra algorithm ia a single source shortest path algorithm and it can only be applied to connected graphs having positive edge weights.
The algorithm consists of following steps :

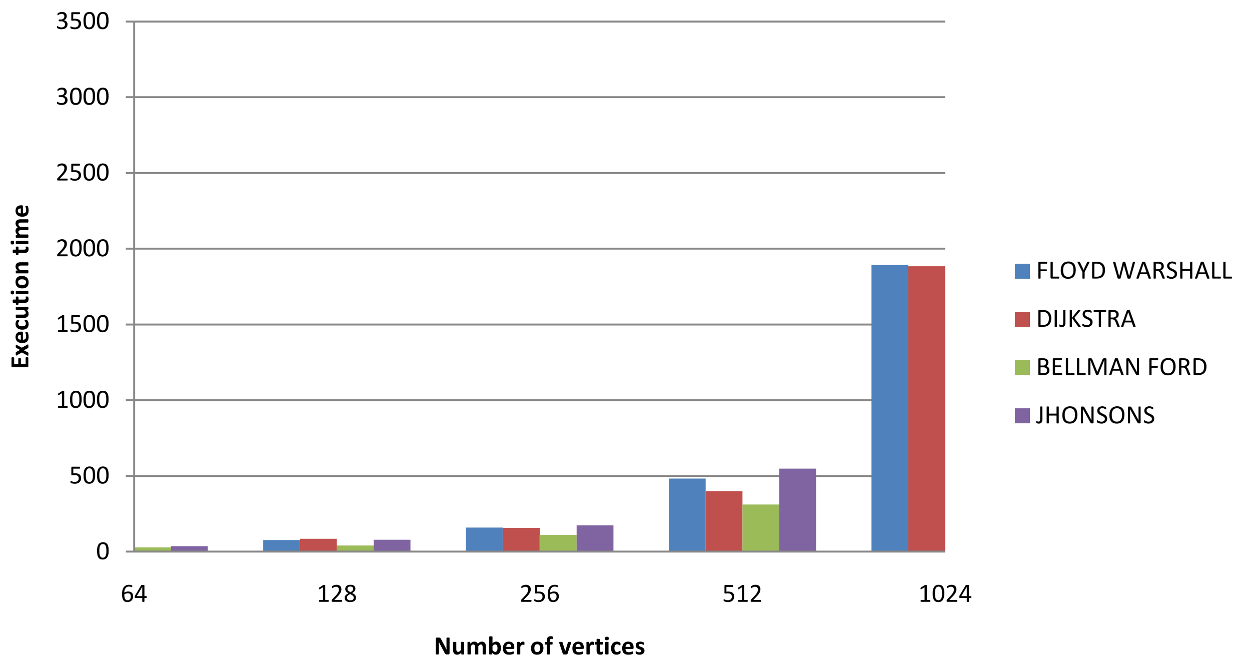
- Distance to source vertex is set to zero.
- Set all other distances to infinity.
- S is set of visited vertices which is empty initially.
- Q is the queue which initially contains all the vertices.
- Then until Q is empty an element is selected from Q with minimum distance.
- And then this u is added to visited vertex list.
- If new shortest path found is shortest among all it is set as new shortest path till this step.

Dist[S] ← 0
For all $v \epsilon V - \{S\}$
Do dist[v] ← ∞
S ← φ
Q ←V

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
IJARCCE
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 6, June 2018

While Q ≠ φ
Do u ← min(Q,dist)
S ← S U {u}
For all v ϵ neighbours[u]
Do if dist[v] > dist[u] + W(u,v)
Then d[v] ← d[u] + W(u,v)
Return dist

Identified parallelism :
For all the vertices in Q we can execute below steps in parallel and find the vertex using write-write consistency which holds minimum distance value. So that the distance returned will be the least of all the vertices processed in parallel.



VIII. JHONSONS ALGORITHM

Johnson's algorithm is a way to find the shortest paths between all pairs of vertices in a sparse, edge weighted, directed graph. It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist. It works by using the Bellman–Ford algorithm to compute a transformation of the input graph that removes all negative weights, allowing Dijkstra's algorithm to be used on the transformed graph.[1][2] It is named after Donald B. Johnson, who first published the technique in 1977.

Single source shortest path algorithm finds shortest path from a source vertex to all other vertices in the graph. For a graph of 'n' vertices there can be n*n possible pair of vertices including same vertex set (v,v) which will be zero for simple graph as self loop will not be there. So workgroup of {n,n} will be suitable for SSSP. Each work item in workgroup will represent a pair (u,v) where where u and v ϵ V.

As illustrated in [15] the parallel algorithm of Bellman-Ford for SSSP got the speedup as shown in table 5.1.First we will analyze execution time of parallel implementation for SSSP on CPU and GPU. From table 5.1 it is clear that GPU implementation is nearly 4 times faster than that on CPU.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940
IJARCCE
International Journal of Advanced Research in Computer and Communication Engineering
ISO 3297:2007 Certified
Vol. 7, Issue 6, June 2018

COMPARATIVE ANALYSIS

ALGORITHM	TYPE	COMPLEXITY	GRAPH	PARALLELISM
Dijkstra	SSSP and APSP	$O(n^2)$ and $O(n^3)$	Positive edge weights only	Coarse grained parallelism
Bellman Ford	SSSP and APSP	$O(n^3)$ and $O(n^4)$	• Negative and positice both. • Can detect negative cycle also. • But doesn't works for graph which contains negative cycle.	Fine grained parallelism
Floyd Warshall	APSP	$O(n^3)$	• Negative and positice both. • But doesn't works for graph which contains negative cycle.	Fine grained parallelism
Jhonsons Algorithm	SSSP and APSP	$O(n^2)$ and $O(n^3)$	• Uses Bellman Ford algorithm to remove negative weights from graph • And then applies Dijkstra on transformed graph	Coarse grained parallelism

So we can say if for a particular value of 'N' GPU will take 't' time then CPU will take '4t'. Then for N.x CPU will take approximate $x^3.4t$ time as algorithm is of the order of $O(n^3)$ and GPU will take approximately $(1-x)^3.t$ time. The main objective of fine tuned implementation is to take such a value of 'x' so that both the time becomes comparable to each other that is:

$$x^3.4t = \alpha. (1-x)^3.t$$

Both can be finely tuned if α reaches nearby 1. After testing the implementation for different values of 'x', best results are obtained when vertices are divided in the ratio 1:3. So we will divide the work in ratio 1:3 among CPU and GPU so both will take comparable time in parallel and overall execution time will depend on the one which completes last. So here for n vertices n/3 will be handled by CPU and 2n/3 will be handled by GPU. Host algorithm is shown in algorithm.

IX. CONCLUSION

In this paper it is found Bellman Ford algorithm has more parallelism as compared to other algorithms. So In this paper all three algorithms are studied and parallelism is identified. Results illustrate that Bellman-Ford algorithm performs better than the other discussed algorithms.

REFERENCES

[1] Michael J. Bannister and David Eppstein , *"Randomized Speedup of the Bellman Ford Algorithm"* in arXiv:1111.5414v1 [cs.DS] 23 Nov 2011.
[2] Andrew V. Goldberg, Tomasz Radzik , *A Heuristic improvement of the Bellman Ford algorithm.Appl. Math. Lett.*Vol. 6, No. 3, pp. 3-6, 1993.
[3] A.S. Nepomniaschaya, *An Associative Version of the Bellman-Ford Algorithm for Finding the Shortest Paths in Directed Graphs*, V. Malyshkin (Ed.): PaCT 2001, LNCS 2127, pp. 285–292, 2001.
[4] J. Y. Yen., *An algorithm for finding shortest routes from all source nodes to a given destination in general networks*. Quarterly of Applied Mathematics 27:526-530, 1970.
[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. Problem 24-1: *Yen's improvement to Bellman Ford*. Introduction to Algorithms, 2nd edition, pp. 614-615. MIT Press, 2001.
[6] R. Bellman. On a routing problem. Quarterly of Applied Mathematics 16:87-90,1958.
[7] Yefim Dinitz , Rotem Itzhak , *Hybrid Bellman-Ford-Dijkstra Algorithm.*
[8] Aydın Buluc , John R. Gilbert and Ceren Budak , "*Solving Path Problems on the GPU" , Journal Parallel Computing Volume 36 Issue 5-6, June,2010 Pages 241-253.*
[9] Andrew Davidson , Sean Baxter, Michael Garland , John D. Owens , "*Work-Efficient Parallel GPU Methods for Single-Source Shortest Path " in International Parallel and Distributed Processing Symposium, 2014*
[10] Owens J.D., Davis, Houston, M., Luebke, D., Green, S., "*GPU Computing*", in: Proceedings of the IEEE, Volume: 96 , Issue: 5 , 2008.
[11] A. Munshi, B. R. Gaster, T.G. Mattson, J. Fung, D. Ginsburg, "*OpenCL Programming Guide*", Addison-Wesley pub., 2011.
[12] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, Second Edition. The MIT Press, Sep. 2001.
[13] Kumar, S.; Misra, A.; Tomar, R.S. ,"A modified parallel approach to Single Source Shortest Path Problem for massively dense graphs using CUDA" in *Computer and Communication Technology (ICCCT), 2011 2nd International Conference on* , vol., no., pp.635,639, 15-17 Sept. 2011.
[14] Atul Khanna, John Zinky , "The Revised ARPANET Routing Metric", in 1969 ACM.
[15] Hristo Djidjev and Sunil Thulasidasan,Guillaume Chapuis, Rumen Andonov, and Dominique Lavenier "Efficient Multi-GPU Computation of All-Pairs Shortest Paths" in 2014 IEEE 28th International Parallel & Distributed Processing Symposium.