A Fine Tuned Hybrid Implementation for Solving Shortest Path Problems using Bellman Ford

Gaurav Hajela
Department of Computer Science and Engineering
Maulana Azad National Institute of Technology
Bhopal, India

Manish Pandey
Department of Computer Science and Engineering
Maulana Azad National Institute of Technology
Bhopal, India

ABSTRACT

In this paper a hybrid implementation for Bellman-Ford to solve shortest path problems is proposed using OpenCL. Here first parallel implementation for Bellman-Ford for single source shortest path (SSSP) problem and all pair shortest path (APSP) are analyzed on CPU and GPU and based on this analysis work is divided among CPU and GPU and hybrid implementation is done. As proper resource utilization is done here we have termed it a fine tuned implementation. We have got considerable speedup of 2.88x over parallel implementation on GPU for SSSP and 3.3x over parallel implementation of Bellman-Ford for APSP on GPU.

Keywords

Shortest path problem , OpenCL , Graphical processing unit(GPU).

1. INTRODUCTION

In this world of fastest growing technology, computers are becoming more powerful than ever before. So it's a challenging task to make efficient utilization of all the resources within a machine. In early days only CPU are involved in programming but now a day GPU which are termed as General Purpose Graphical Programming Unit (GPGPU) are also available as one of the resource which can be equally utilized and can provide high performance at a reasonable cost. GPU are well suited for applications which involve the use of matrices due to its architecture.

One of the applications is shortest path problems on graph which deals with matrices. Shortest path problems finds lot of applications in real world like VLSI design, finding directions between physical locations like Google maps, wechat. This paper deals with Bellman-Ford algorithm to find shortest path in a graph. This algorithm works on negative edges also where majority of algorithms fail and is capable of detecting negative cycle in graph. So this paper aims to provide hybrid implementation of Bellman-Ford which will utilize both CPU and GPU using a standard framework called OpenCL.

OpenCL provides a way to utilize heterogeneous resources in a system and one can control the execution on particular hardware. CPU and GPU can be selected explicitly for execution using OpenCL. The piece of code which controls the execution is called host code and that which run in parallel on CPU or GPU is called kernel.

1.1 Bellman-Ford Algorithm

Bellman-Ford algorithm aims to find shortest path from a single source to all other vertices in graph. It can also be used to find APSP by applying it to all the vertices in graph. Bellman-Ford algorithm is shown in Algorithm 1.

```
Algorithm BellmanFord (s,Dist,Cost,n)
{
    1.  for i=1 to n do
    2.      Dist[i] = Cost[s,i];
    3.  End for
    4.          for k=1 to n-1 do
    5.              for each (u,v) in E do
    6.                  $Dist^k[v] = min (Dist^{k-1}[v], Dist^{k-1}[u] + Cost[u,v])$
    7.              End for
    8.          End for

}
```

Algorithm 1: Algorithm for Bellman-Ford.

Where *Cost* contains adjacency matrix representation of graph and is a n*n matrix for given graph G with n vertices,E the set of edges and V set of vertices. *Dist* matrix will initially contain direct edges from source vertex to all other vertices in the graph and after successful completion it will contain shortest path length from source to all the other vertices in set V. *$Dist^k[v]$* will contain shortest path length from 's' to 'v' going through atmost 'k' intermediate vertices.

Organization of the paper: In Section 2, the previous modified algorithms have been discussed along with the improvements made on Bellman Ford algorithm by different authors. In Section 3 hybrid implementations of Bellman-Ford algorithm for SSSP and APSP are explained along with host and kernel algorithm. Comparative analysis and results are shown in Section 4.

2. RELATED WORK

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.

The results of parallel implementation of Bellman-Ford for SSSP and APSP in [15] were analyzed and on the basis of that analysis the work between CPU and GPU is divided and hybrid implementation for Bellman-Ford is proposed. This paper aims at proper resource utilization by implementation and which is clearly illustrated by graphs that it is a fine tuned implementation.

3. HYBRID IMPLEMENTATION OF BELLMAN-FORD

3.1 Single source shortest path (SSSP) problem

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 table1.First we will analyze execution time of parallel implementation for SSSP on CPU and GPU. From table 1 it is clear that GPU implementation is nearly 4 times faster than that on CPU.

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 2.

Table 1: Speedup comparison with respect to serial implementation on specified CPU for SSSP.

NO. OF VERTICES	PARALLEL_CPU	PARALLEL_GPU
64	3	12
128	2.451	10.857
256	3.449	8.827
512	3.638	16.398
1024	3.755	17.378
2048	3.671	17.611
AVERAGE	3.335	13.843

```
Algorithm OpenCL_Hybrid_BellmanFord_SSSP
 {
    1. For k from 1 to n-1 do
    2.    For all v in V such that (u,v) belongs to E and v ranges from 0 to n/3 -1 in parallel do
    3.       Call KERNEL_BELLMAN_SSSP_CPU(Cost,Dist,k,,W1)
    4.    For all v in V such that (u,v) belongs to E and v ranges from n/4 to n in parallel do
    5.       Call KERNEL_BELLMAN_SSSP_GPU(Cost,Dist,k,W2)
    6. End for
}
```

Algorithm 2: Algorithm for host code of Hybrid Bellman Ford for SSSP

Here W1 represent workgroup size which is {n,n/3} so ‘u’ will vary from 0 to n-1 and v from 0 to n/3 – 1 because we want to assign only n/3 vertices to CPU. Algorithm for CPU is shown in Algorithm 3.

```
KERNEL_BELLMAN_SSSP_CPU(Cost,Dist,k,W1)
{
   u = get_global_id(0)
   v = get_global_id(1)

   if k is odd then
        // synchronization is done here
        Dist[1][v] = min(Dist[0][v], combine(Dist[0][u] + Cost[u][v]))

   if k is even then
        // synchronization is done here
        Dist[0][v] = min(Dist[1][v], combine(Dist[1][u] + Cost[u][v]))

}
```

Algorithm 3: Algorithm for kernel of Bellman Ford for SSSP

While W2 represent workgroup size for GPU which is {n,2n/3} and algorithm for GPU implementation is represented in algorithm 4. As first n/3 are handled by CPU so on GPU we have taken n/3 as offset for v because W2 will vary from 0 to 2n/3 -1.

```
KERNEL_BELLMAN_SSSP_GPU(Cost,Dist,k,W2)
{
   u = get_global_id(0)
   v = n/3 + get_global_id(1)

   if k is odd then
        // synchronization is done here
        Dist[1][v] = min(Dist[0][v], combine(Dist[0][u] + Cost[u][v]))

   if k is even then
        // synchronization is done here
        Dist[0][v] = min(Dist[1][v], combine(Dist[1][u] + Cost[u][v]))
}
```

Algorithm 4: Algorithm for kernel of Bellman Ford on GPU for SSSP

3.2 All pair shortest path (APSP) problem

To calculate APSP using Bellman-Ford, SSSP will be called for all the vertices in graph each will act as a source once.So 3D workgroup is used for solving APSP in [15].

As illustrated in [15] Bellman-Ford for APSP will got the following speedup as shown in table 2.

Table 2: Speedup comparison with respect to serial implementation on specified CPU for APSP

NO. OF VERTICES	PARALLEL_CPU	PARALLEL_ GPU
64	4.218	12
128	5.103	14.933
256	5.292	17.406
512	6.100	22.101
1024	7.300	26.300
AVERAGE	5.602	18.548

Similarly it is clear from the table 2 that GPU is nearly 3 times faster than CPU. So we have divided the algorithm accordingly in ratio 1:3 among CPU and GPU on the basis of equation explained in SSSP section. Host algorithm is shown in algorithm 5.

```
Algorithm OpenCL_Hybrid_BellmanFord_APSP

{

1.   For k  from 1 to n-1 do

2.      For all v in G such that (u,v) belongs to G and v
        ranges from 0 to n/3 -1 in parallel do

3.   Call KERNEL_BELLMAN_APSP_CPU(Cost,Dist,k,W1)

4.   For all v in G such that (u,v) belongs to G and v ranges
     from n/3 to n  in parallel do

5.   Call KERNEL_BELLMAN_APSP_GPU(Cost,Dist,k,W2)

6.      End for

7.   End for

 }
```

Algorithm 5: Algorithm for host code of Bellman Ford for APSP

Here W1 represent workgroup size which is {n,n,n/4} so 'u' and 'v' will vary from 0 to n-1 and s from 0 to n/4 – 1 because we want to assign only n/4 vertices to CPU. Algorithm for CPU is shown in Algorithm 6.

```
KERNEL_BELLMAN_APSP_CPU(Cost,Dist,k,W1)
{
   u = get_global_id(0)
   v = get_global_id(1)
   s = get_global_id(2)
   offset = s*2*n
   if k is odd then
        // synchronization is done here
      Dist[1][v] = min(Dist(offset + [0][v]),
combine(Dist(offset + [0][u]) + Cost[u][v]))

   if k is even  then
        // synchronization is done here
        Dist[0][v] = min(Dist(offset + [1][v]),
combine(Dist(offset + [1][u]) + Cost[u][v]))
}
```

Algorithm 6: Algorithm for kernel of Bellman Ford for APSP for CPU

```
KERNEL_BELLMAN_APSP_GPU(Cost,Dist,k,W2)
{
   u = get_global_id(0)
   v = get_global_id(1)
   s = n/3 + get_global_id(2)
   offset = s*2*n
   if k is odd then
        // synchronization is done here
      Dist[1][v] = min(Dist(offset + [0][v]),
combine(Dist(offset + [0][u]) + Cost[u][v]))

   if k is even  then
        // synchronization is done here
        Dist[0][v] = min(Dist(offset + [1][v]),
combine(Dist(offset + [1][u]) + Cost[u][v]))
}
```

Algorithm 6: Algorithm for kernel of Bellman Ford for APSP for GPU

While W2 represent workgroup size for GPU which is {n,n,3n/4} and algorithm for GPU implementation is represented in algorithm 7. As first n/4 are handled by CPU so on GPU we have taken n/4 as offset for v because W2 will vary from 0 to 3n/4 -1.

4. COMPARATIVE ANALYSIS AND RESULTS

All the OpenCL parallel implementations are tested on following GPU and CPU:
AMD Radeon HD 6450(GPU): 2 Compute units, 625 MHz clock, 2048MB Global Mem., 32KB Local Mem., 256 work group size on a system having Intel Core i5 CPU 650 @ 3.2 GHz and 2048MB RAM with AMD APP SDK v2.8.

Implementations are done using using Visual Studio 2010 with OpenCL SDK 1.2. We have tested serial implementations on the above specified CPU. We have tested all our implementations on randomly generated graphs having edge weights between -10 to 10 where bellman ford has successfully detected negative cycles if present in graph. Rest of the results are taken on graphs without negative cycle in which we have considered kernel execution time only.
Comparative analysis of execution time of different variants of algorithm is shown in Figure 1 and Figure 2. For parallel

implementations only kernel execution time is considered. Speedup of different variants for SSSP and APSP is shown in Table 3 and Table 4 respectively.

Table 3: Speedup comparison of hybrid implementation with respect to parallel implementation on specified GPU for SSSP

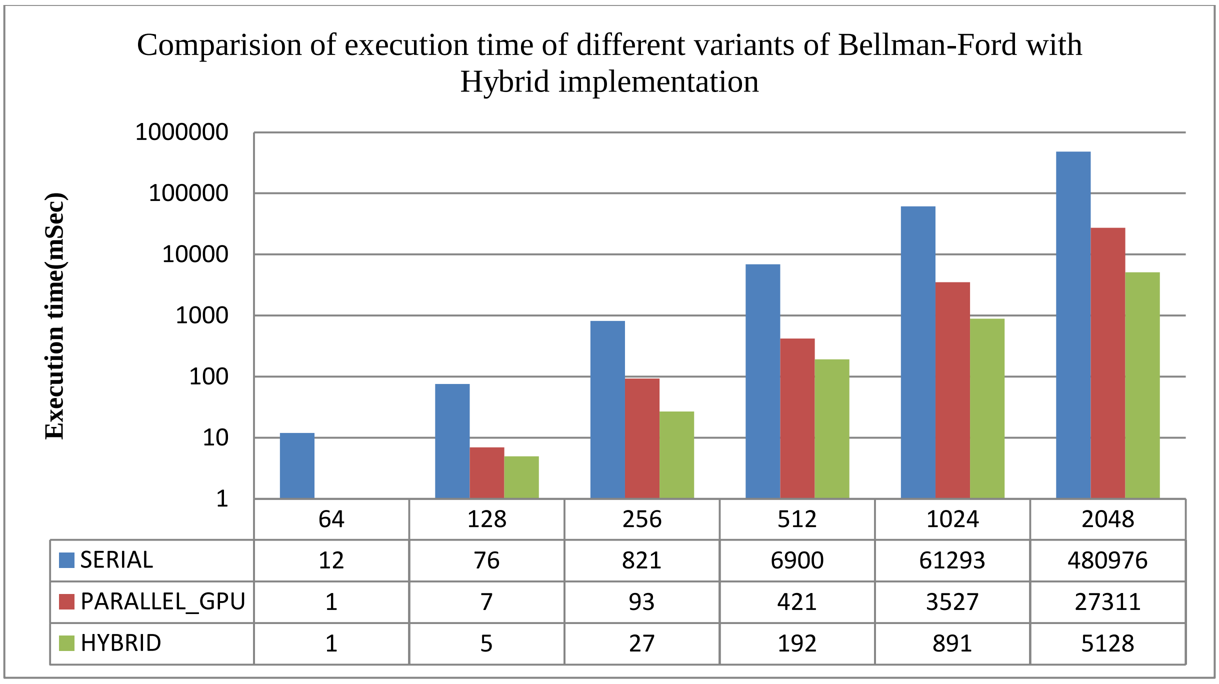
NO. OF VERTICES	SPEED UP
64	1
128	1.4
256	3.44
512	2.19
1024	3.95
2048	5.32
AVERAGE	2.88

Table 4: Speedup comparison of hybrid implementation with respect to parallel implementation on specified GPU for APSP

NO. OF VERTICES	SPEED UP
64	1.09
128	2.23
256	4.29
512	3.81
1024	5.06
AVERAGE	3.30

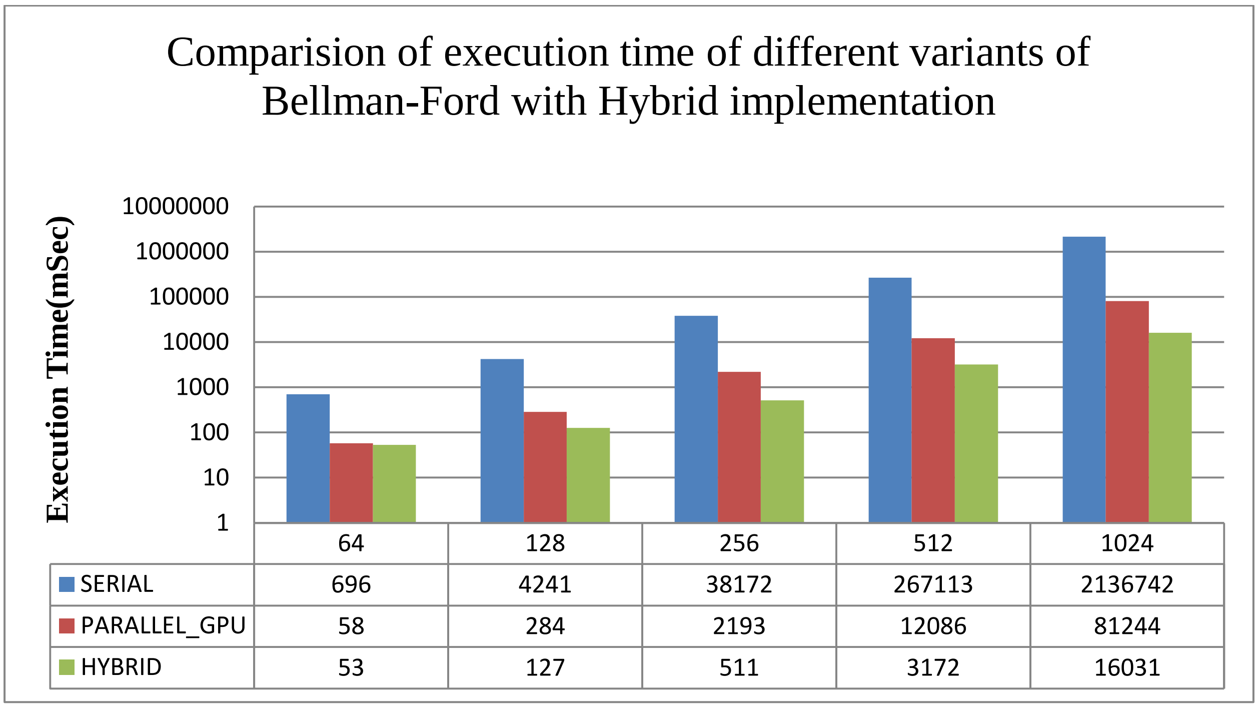
5. CONCLUSION AND FUTURE WORK

Fine tuned hybrid implementation proposed in this paper has got considerable speedup of 2.88x over parallel implementation on GPU for SSSP as both are assigned work accordingly and have taken equivalent amount of time. Similarly hybrid implementation for APSP has got speedup of about 3.3x over parallel implementation on GPU. In future focus will be to partition the algorithm for hybrid implementation rather than dividing the vertices among CPU and GPU. Partitioning algorithm among CPU and GPU can be more efficient.



	64	128	256	512	1024	2048
🟦 SERIAL	12	76	821	6900	61293	480976
🟥 PARALLEL_GPU	1	7	93	421	3527	27311
🟩 HYBRID	1	5	27	192	891	5128

Figure 1: Comparative analysis of execution time of different variants of Bellman Ford for SSSP



	64	128	256	512	1024
🟦 SERIAL	696	4241	38172	267113	2136742
🟥 PARALLEL_GPU	58	284	2193	12086	81244
🟩 HYBRID	53	127	511	3172	16031

Figure 2: Comparative analysis of execution time of different variants of Bellman Ford for APSP.

6. 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] Gaurav Hajela, Manish Pandey, “Parallel implementations for solving shortest path problem using Bellman-Ford” in IJCA June 2014 edition