International Journal of Computer Applications (0975 – 8887)
Volume 139 – No.13, April 2016

Cache Friendly Bellman-Ford algorithm using OpenCL

Lekha Jadhav Rahul Dubey Manish Shrivastava, PhD

ABSTRACT

Shortest path algorithms play a vital role in real world applications. In this paper a cache friendly implementation for Bellman Ford algorithm to solve single source shortest path and all pair shortest path algorithm is proposed. The proposed algorithm is compared with sequential algorithm in terms of execution time, cache hit, ALUPacking and ALUBusy. This algorithm is also tuned with execution environment to yield maximum performance. In this paper we have discussed all above factors in terms of framework called OpenCL.

Keywords

Bellman-Ford, OpenCL, ALUPacking, ALUBusy, Cache hit.

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 Processing 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 problem finds application in large domains of scientific and real world. Common applications of these algorithms are in network routing [6], VLSI design, robotics and transportation, they are also used for directions between physical locations like in Google maps. Here all the applications mentioned generally involve positive weights but some applications are there where weights can be negative like currency exchange arbitrage and some other areas where, edge represents something other than merely distance between two entities. In such application areas Bellman-Ford algorithm can be used. Bellman-Ford algorithm [12] is applicable on graphs with negative weights and can also detect negative cycles where majority of algorithms fail. Bellman-Ford is also used in wireless sensor networks and other ad hoc networks as distributed Bellman Ford [7] can be used there. Distributed Bellman-Ford is also used as first ARPANET routing algorithm in 1969 [14].

Most of the above application areas specified are real time applications and need results in a quick time so the performance of algorithm need to be improved so that it consume less power and time. Parallel computing on GPU is one of the technologies which are used for high performance computing at a reasonable cost and considerable speed up of performance. GPU is currently used for a variety of purposes apart from graphical processing and gaming. That’s why we refer GPU as General Purpose Graphical processing unit (GPGPU)[10] as it provides high performance computing can be programmed using standard frame work like OpenCL and CUDA. OpenCL [11] is a framework which is for all GPU while, CUDA is meant specifically for NVIDIA GPUs only. Thus, we will be using OpenCL for our GPU implementation due to its portability and open-ness. 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. In this paper a cache friendly hybrid implementation with vectorization for solving shortest path algorithm using Bellman-Ford algorithm by the use of OpenCL platform is proposed. Traditionally programs are intended to run on CPU with operating system controlling the execution of program. In such programs which are also referred as set of instructions to be executed; each instruction starts execution only when previous instruction is completed. Such programs are referred as sequential programs. But as now there are plenty of resources other than CPU in a system and its programmers responsibility to utilize all those resources new trend of programming has been introduces. This new era of programming is referred as parallel computing.

And Bellman Ford algorithm possess inherent parallelism and is applicable in wide domain as it is applicable for the graphs with negative edge weights and is also able to identify negative cycle in garphs.

2. LITERATURE REVIEW

Bellman Ford was introduced by Richard Bellman and Lester Ford Jr. in 1958. Since then several modifications and improvements were made on this algorithm. Yen et.al [5] modified the algorithm in 1970 which proved to be quite famous. In 1993, another modification which included topological scan algorithm for Bellman Ford [2] was developed. This gained attention due to its capability to outperform the standard algorithm in most of the cases. A hybrid implementation of Bellman Ford and Dijkstra’s algorithm is given in [7]. This implementation is asymptotically better than Bellman Ford. As Dijkstra’s algorithm does not work on negative dges, this algorithm was applied to graphs with sparse distribution of negative edges. 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 $O(n^2)$. STAR machine is a parallel processor of SIMD type and possess ability to perform vector processing. Michael J. Bannister and David Eppstein [1] proposed a randomized variant of algorithm in 2011. It was improved by a factor of 2/3 over Yen’s modification (1970) [4, 5]. This speedup was termed as randomized speedup. Due to GPU’s supportive architecture, several parallel implementations on GPU for SSSP algorithms were proposed. AydınBuluc, John R. Gilbert and CerenBudak [8] proposed parallel implementations for SSSP and APSP using CUDA. In [13], a CUDA implementation for Bellman Ford has been provided. A speedup of about 10x was obtained by making the algorithm suitable for parallelism. Recently, Andrew Davidson [9] presented several work efficient methods for SSSP problems. Considerable speedup was observed not only over serial implementation but over other traditional GPU implementations as well.

3. SHORTEST PATH ALGORITHMS

Shortest path problems deal with graphs. Out of a number of graphs the graphs with which shortest path problems deal belong to weighted directed graph category. Here the weights may be negative or may be positive. Majority of the algorithms deals with graphs having positive edge weights and only a few can deal with graphs having negative edges also.

In general there are two classes of shortest path problems:

a) Single source shortest path (SSSP) problems.

b) All pair shortest path (APSP) problems.

3.1 Single source shortest path

In Single source shortest path problems single source is there and these algorithms aims to find shortest path from this single source vertex to all the other vertices in graph. Weight of the path is the sum of all the edges weights which constitutes the path and path with minimum weight is referred as shortest path.

Some of the single source shortest path algorithms are:

i. Bellman-Ford algorithm.

ii. Dijkstra's algorithm.

Our work is concerned about Bellman-Ford algorithm so we have discussed Bellman-Ford algorithm only.

3.1.1 Bellman-Ford Algorithm

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 'kth' 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.

```
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
}

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
```

INDENTIFICAITON OF PARALLELISM IN BELLMAN FORD:

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 end of Relax() procedure. This issue is referred as write-write consistency.

4. PERFORMANCE ENHANCEMENT IN OPENCL

In opencl there are many measures of kernel performance measurement. Some of the prominent factors that play an important role in kernel's performance are listed below :

ALU Optimization:[3][4]

There are two measurements that show utilization of single instruction multiple data (SIMD) unit these are ALUBusy and ALUPacking.[4][3]

ALUBusy :ALUBusy is defined as rate of instruction processed by SIMD units.

ALUPacking : utilization of 5 ALUs of single SIMD unit by instruction in case of VLIW5 architecture of GPU.

Low ALUBusy indicates either not enough work is scheduled or ALU units are stalled due to data latency. To improve ALUPacking, developers can structure their codes to use more vector operations.

Memory Optimization :[2]

Memory optimization comprises of two memory sections :

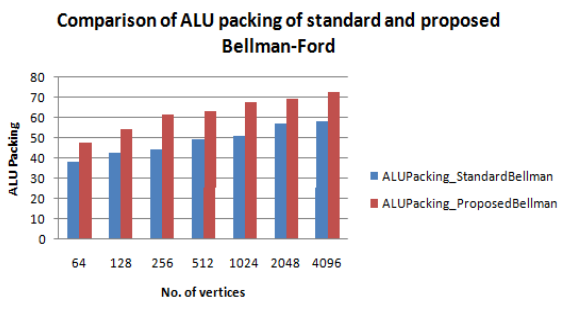
- Global Memory optimization.
- Local Memory Optimization.

Global Memory Optimization: cache utilization is the prominent factor that plays the most important role in global memory optimization.[3][2]

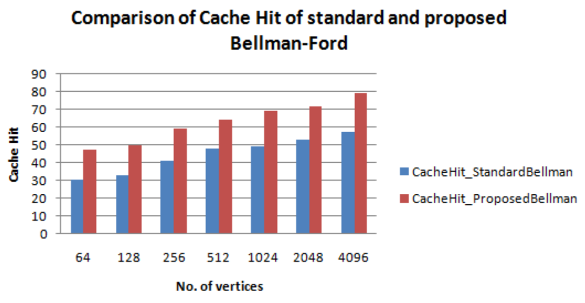
5. RESULTS AND ANALYSIS

In this paper Bellman-Ford is analyzed with previous implementation by different authors in terms of following:

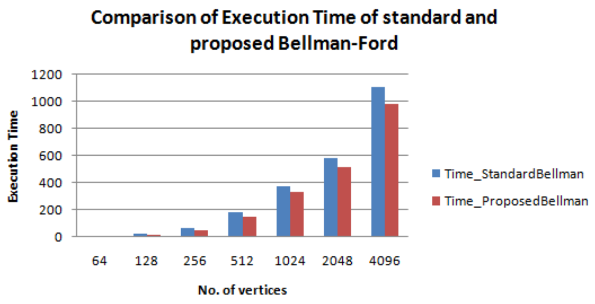
a. **ALUPacking**



b. Cache Hit



c. Execution Time



6. CONCLUSION

In this paper Bellman-Ford is implemented on OpenCL using principal of locality of reference. Here implementation is analyzed in terms of ALU Packing, Cache Hit and Execution time. It is found that after optimized implementation proposed implementation is better.

7. 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] A. Munshi, B. R. Gaster, T.G. Mattson, J. Fung, D. Ginsburg, “OpenCL Programming Guide”,Addison-Wesley pub.,2011.

[7] YefimDinitz ,RotemItzhak , Hybrid Bellman-Ford-Dijkstra Algorithm.

[8] AydınBuluc , John R. Gilbert and CerenBudak , “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.