Comparative Study on Boruvka's Implementation on Hetrogenous Platform with Cache Analysis

Vivek Goyal, Angad Nandwani, J. Sairabanu, Sumithra Sriram[a]
School of Computing Science and Engineering (SCSE), VIT University, India
vivek.goyal2013@vit.ac.in, angad.nandwani2013@vit.ac.in, banu83@gmail.com, sumithrasriram@gatech.edu
[a]:Georgia Institute of Techonology,Atlanta

***Abstract*— To address the performance analysis of Boruvka's algorithms by analyzing different cache performance parameters in VTUNE Amplifier. In general, Minimum Spanning Tree [MST] is computed with various algorithms, Boruvka's being the oldest and efficient. MST is widely used in applications like search engines, pattern recognition, routing algorithms, network design etc. The paper presents the time comparative study of two different methods of Boruvka's algorithm on different architectures of multi core CPU-chips. It highlights the importance of GPU computing in the modern era and shows that the Compressed Sparse Row[CSR] format of implementation for Boruvka's algorithm outperforms in all the platforms for different readily available benchmarks.**

Keywords—performance;borukva's;cache; vtune; MST; oldest; time; architectures; gpu; csr;outperforms;benchmarks

I. INTRODUCTION

In the new era of modern computing, new types of heterogeneous computers have begun to emerge[15]. These system contain multiple processors with a powerful CPU and GPU core to accelerate the computation[1]. A survey of the top 10 supercomputers in the world shows that the systems contain an average of 182,035 processors running at an average frequency of 2.5GHz[2]. This frequency is too low to perform a task involving large computations, hence evolving the techniques of parallelization.

These systems provide an opportunity to developers, to design such programming languages[16] or software's which can efficiently distribute work between the two cores. Traditionally GPU operates in co-ordination with CPU where CPU offloads parallel parts of computation to GPU[4]. Each Kernel in GPU can generate thousands of threads to fully utilize the thread level parallelism available in GPU.

Today we require a minimum spanning tree solver for several application domains, like search engines, networking etc., therefore demanding an efficient algorithm for MST solver[9]. This technique of offloading the work from CPU to GPU, can be efficiently utilized in getting minimum spanning tree quicker[3]. We can efficiently parallelize the computation for the MST solver and reduce the time effectively in getting results. In this paper we have compared the difference in time it takes for the MST algorithm to run on both homogenous and heterogeneous system.

We have different methods for MST solver of which Boruvka's Algorithm is the oldest[5]. It has two variants. In first method[11] we solve the MST by Boruvka's approach. We approach the MST by grouping the nodes to find the super-vertices and representing each group of nodes by a representative id. In the second method[12], we use CSR(Compressed Sparse Row) format to represent the graph and then find the MST.

MST-solver algorithm is mostly implemented through CPU. But nowadays with the advent of GPU, the MST-solver algorithms[10] can be parallelized which can show a drastic reduction in time require to solve MST[20]

The primary job for graphic card is to compute 3-D Functions[13] but its ability to compute heavy calculation effectively has helped the computer to increase its efficiency. GPU is an electronic circuit used in embedded system, personal computers mainly for applications[14] like image processing and manipulation of computer graphics. It is also used to reduce the effective time consumption for programs like sorting, vector addition, vector multiplication of matrixes with size in order of thousands. If we use GPU for computationally intensive tasks, then this kind of implementation is called GPGPU[6]. It is used for performing complex mathematical opeartions in parallel for achieving low time complexity.

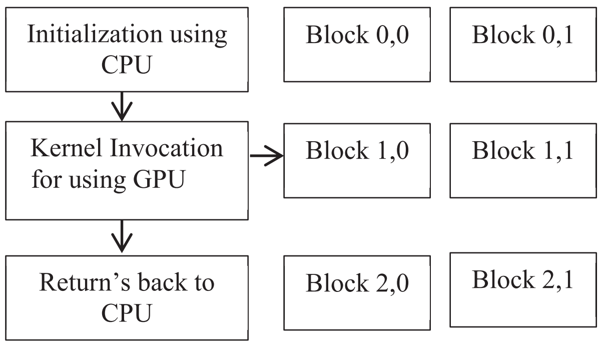

Fig 1. Execution of Code in GPU

Whenever the code is executed, the parallel code is separated from the serial code. The parallel part of code is executed by the GPU environment while the serial part of the code is executed by the CPU environment, as shown in Fig 1. We can also increase the performance of the MST Solver using cache. In GPU, cache is use to localize data during volume rendering [17], while in CPU, it is used to localized data during memory references. Using cache in CPU implementation, we will reduce the traversing time through graphs, increasing the performance ratio.

In this paper we have done a comparative study of Boruvka's Algorithm and its implementation on CPU as well as on GPU. We have also analyze the processor's throughput using VTUNE's Cache Analysis to reduce stall cycles and identify potential performance issues.

II. BORUVKA'S IMPLEMENTATION

Boruvkas algorithm is the earliest known solution for minimum spanning tree. They provide a quick MST for different undirected graphs G(V,E), where V denote the number of the vertices and E denote the number of edges present in graph(Fig 2). Any edge is denoted by three components E(u,v,w) where u is source vertex, v is the destination vertex and w is the weight of that edge. It is the algorithm which gives least complexity of O(m) [5] as compare to other MST algorithms[18].

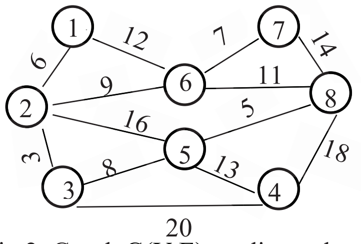

Fig 2. Graph G(V,E), undirected graph with eight vertices and thirteen edges. Each vertex is given a vertex id represented in the circle.

The algorithm is implemented in three major steps. The prior step is to find the minimum weighted edge for each node. To find minimum weighted edge, loop across each node and select the minimum weighted edge. To find minimum weighted edge we have used flag array, where 1 indicates starting of each segment node where 0 indicates all the edges which are part of that segment While finding the minimum weighted edge for each node, there may be cases when two particular nodes have the same minimum weighted edge.

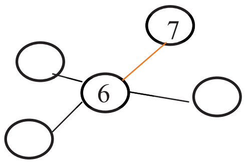

Fig 3.A

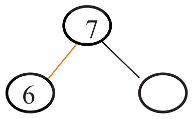

Fig 3.B

For vertex shown in both the figures above, they have the common minimum weighted edge i.e the edge with weight 7. So, to include it only once, we replace the edge of node with lower vertex to self-loops. In this case, edge of node with vertex id 6 will be replaced by self-loop at 6. The self-loop edges will help us to exclude that source vertex during the formation of super-vertex.

The second step is to form super vertex out of the connected minimum weighted edges and assign different representative id's to them. For the graphs, these are the super vertex formed from the minimum weighted edge of each vertex, and each given a representative id. The third step is the repetition of the above procedure i.e. finding the minimum weighted edge between these super-vertices formed and continuing the procedure till we get the one super-vertex. The super-vertex obtained will be the minimum spanning tree for the given graph.

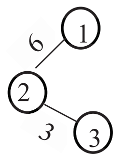

Fig 4.A Id:1

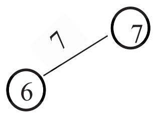

Fig 4.B Id:2

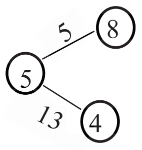

Fig 4.C Id:3

The disadvantage for the above procedure is that we cannot trace back the results obtained. We are replacing each time the super-vertex obtained with a new id, losing the original identity of the vertex. We will only be able to obtain a path i.e. the minimum spanning tree but with a new identity, making it difficult to trace back.

The major time taken in the above procedure was to assign flags, for indicating the starting of the segment. Each time with recursive approach, the segment needs to be identified with a flag array, which took a major portion of time. To reduce implementation time, we used GPU to parallelize the assignment of flag array each time in recursive approach. This brought a major change in reducing time to find a minimum spanning tree.

III. BORUVKA'S IMPLEMENTATION USING CSR FORMAT

Another implementation of Borukvas algorithm is parallel approach, where we use the CSR format to reduce effective time in solving MST. Using GPU was only effective with a large number of vertices, which is not possible in every case. Overhead generation while implementing algorithm in GPU for smaller number of vertices extends the total time of implementation than that taken by CPU. Thus, parallel approach can help us to obtain efficiency in these cases.

CSR format is representation of the graph using four arrays (Table 1) that provide us an effective method to implement MST. The four arrays are

- Destination Array- It notes destination vertex for each source vertex.
- Weight Array- It notes the weight corresponding to each vertex in destination array.

- Outdegree Array- It notes the total number of edges coming out from each source vertex.
- First edge- It notes the position of first edge for each vertex.

The algorithm includes five major steps. The prior step is to find the minimum edge for each source vertex. The second step is to remove cycles by replacing each time the edge of node of lower vertex id to self-loop. Third, we assign flags to each segment. Forth, using the flags, we find exclusive prefix sum to assign representative id's to each segment.

TABLE 1. The four arrays Destination, Weight , Outdegree and First edge for the above graph in Fig 1.

Source	1	1	2	2	2	2	3	3	3	4	4	4	5	5	5	5	6	6	6	6	7	7	8	8	8	8
Destination	2	6	1	6	5	3	2	4	5	3	5	8	2	3	4	8	1	2	7	8	6	8	6	7	5	4
Weight	6	12	6	9	16	3	3	20	8	20	13	18	16	8	13	5	12	9	7	11	7	14	11	14	5	18

Source	1	2	3	4	5	6	7	8
Outdegree	2	4	3	3	4	6	2	4
First Edge	0	2	6	9	12	16	22	24

Fifth, we recursively form all the four arrays for each segment assuming them as a new vertex.

To find the minimum edge, we loop across each vertex in weight array using outdegree array and first edge array to find minimum weighted edge for each source, as shown in Table 2

TABLE 2. Minimum edge selection between the source and destination mentioned.

Source	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Destination	**2**	**3**	**2**	**5**	**8**	**7**	**6**	**5**

In Table 3, we have three cycles formed, to remove them we replace the destination vertex of source with lower vertex id depicting self-loops.

TABLE 3. Removing CYCLES

Source	1	**2**	3	4	**5**	**6**	7	8
Destination	2	**2**	2	5	**5**	**6**	6	5

Each segment in the graphs, has a vertex with a self-loop. These vertices will be assigned a flag, and these flags will be used to calculate exclusive prefix sum for each vertex which will provide a new vertex id to each segment. In Table 4, the three segment formed has three representative id's as shown in the exclusive prefix sum.

TABLE 4. Assigning new vertex Id

Source	1	**2**	3	4	**5**	**6**	7	8
Flag	0	**1**	0	0	**1**	**1**	0	0
Exclusive Prefix Sum	1	**1**	2	2	**2**	**3**	4	4

Then, we will be again finding the four arrays (Table 5) for the new vertices in the graph. These four arrays will be put to recursion providing us with the single super-vertex in the end resulting in minimum spanning tree.

IV. RESULTS

In Table 8, we have analyze the time comparison between Boruvka's algorithm and its CSR variant approach on three different processors . The results shown are for two set s of nodes[22], one with less number of vertices and other with greater (3353 and 67966 vertices). Through results, we observe that the time required in obtaining MST with I7 processor is less as compared to time required by the other processors (Table 6).

TABLE 5. The four arrays Destination, Weight , Outdegree and First edge for the next iteration

Vertex Id	1	2	3
Outdegree	5	5	4
First Edge	0	5	10

Source	1	1	1	1	1	2	2	2	2	2	3	3	3	3
Destination	3	3	2	2	2	1	1	3	3	1	1	1	2	2
Weight	12	9	16	8	20	16	8	11	14	4	12	9	11	14

This is due to increase in the number of cores, more cache, higher clock speed and hyper-threading.in the I7 processor.

We also observe that initially for less number of vertices the time taken by GPU was more as compared to CPU. This increase in time is just because of the overhead of copying data from CPU to GPU and then from GPU to CPU. For less number of vertices, the overhead time is more than the time for computation of MST and hence, resulting in more time in case of GPU. But as we increase the number of vertices, the total time for computation of MST reduces significantly as compared to CPU (Table 7).

TABLE 6: Time Comparison between different processors on graphs with different number of vertices

Processors	Boruvka's Implementation using CPU(ms.)	
	3353	67966
I3	1205.21	183054.14
I5	763.05	116826.01
I7	645.93	97884.67

TABLE 7: Time Comparison between CPU & GPU on I5 processor for graphs with different number of vertices

Boruvka's Implementation using CPU(ms.)		Boruvka's Implementation using GPU(ms.)	
3353	67966	3353	67966
763.05	116826.01	1768.228	74238.74

TABLE 8: *Time Comparison between two algorithms*

Processors	Boruvka's Implementation (ms.)	Boruvka's Implementation using CSR format(ms.)	Boruvka's Implementation using GPU(ms.)
	3353	3353	
I3	1205.21	522.09	
I5	763.054	325.87	
I7	645.93	264.67	

V. CACHE ANALYSIS

Processing data at high clock rates requires an effective fast access to memory. With increase in the processor speed, memory speed was the bottleneck for computations [6], resulting cache to evolve. Cache are small, fast memory that match up to the speed of processors in providing data from memory to processes. It improves the processor throughput by reducing stall cycles due to memory activities.

VTUNES Amplifier is an application for performance analysis[19] of 32 & 64-bit x86 based machines. It assists in various kind of code profiling including stack, thread and hardware event sampling. It details you the time spent by processes in each sub-routine making analysis faster and easier. It helps you to understand multi-threading concept, parallel programming models effectively by collecting effective data and identifying potential performance issues. It is a tool that helps the developer to achieve high level of efficiency by graphically visualizing the memory reference patterns of an algorithm over time. Thus, the analysis of cache has become a key to boost the overall system performance.

Using VTUNES, we have analyze the following parameters[21] for the two algorithms (based on time)

1. CPU_CLK_UNHALTED.REF_TSC

This event count the number of references cycles when the event is not at halt state. It is not effected by core frequency changes and also has the elapsed time for which the core was not in halt state.

2. CPU_CLK_UNHALTED.THREAD

It count the number of thread cycles for which the thread was not in halt state.

3. INST_RETIRED.ANY

It counts the number of instructions that retire from execution. The programmable counter counts the instructions. It is an architectural performance event.

In Table 9, we observe that I7 processors make less number of reference as compare to I5, making the algorithm more optimized. We also conclude that the algorithm with CSR format of Boruvka's implementation takes less time for all the cache parameters, and hence more effective than normal method of implementation.

VI. CONCLUSION

With this paper, we portrayed the importance of GPU in computation of MST. It brought into light the significance of the Boruvka's algorithm i.e. the oldest method of MST computation. It concludes that the CSR format of Boruvka's implementation is more effective with respect to time than the normal implementation of Boruvka's. It highlights that in order to avoid overhead delay in GPU, the graph should have vertices greater than the set threshold. We also observe that implementing these algorithms over higher processors like I7, we can effectively reduce the obtaining time of MST. Using VTUNE's, cache analysis, we also identified potential performance issues with our algorithms.

TABLE 9. CACHE PERFORMANCE ANALYSIS ON TWO ALGORITHMS WITH DIFFERENT PROCESSORS

	I5		I7	
	Boruvka's Implementation (ms.)	Boruvka's Implementation using CSR format(ms.)	Boruvka's Implementation (ms.)	Boruvka's Implementation using CSR format(ms.)
CPU_CLK_UNHALTED.REF_TSC	1792002688	780001170	1750000265	778001167
CPU_CLK_UNHALTED.THREAD	2016003024	858001287	2008003012	896001344
INST_RETIRED.ANY	4312006468	1800002700	4300006450	1800002700

VII. REFERENCES

[1] Li Luo, Chao Yang, Yubo Zhao: A scalable hybrid algorithm based on domain decomposition and algebraic multigrid for solving partial differential equations on a cluster of CPU/GPUs, Chinese Academy of Sciences Shenzhen 518055, P. R. China

[2] Kit Barton, Amy Wang, Steven Perron, Priya Unnikrishnan: Challenges for Parallel Computing, IBM Canada Ltd.

[3] Scott Rostrup, Shweta Srivastava, and Kishore Singhal: Fast and Memory-Efficient Minimum Spanning Tree on the GPU. 700 East Middlefield Rd Mountain View, CA

[4] M. Papadrakakis, G. Stavroulakis, A. Karatarakis: A new era in scientific computing: Domain decomposition methods in hybrid CPU–GPU architectures, National Technical University of Athens, Zografou Campus, Athens 15780, Greece

[5] Jaroslav Nesetril , Eva Milkova, Helena Nesetrilova: Otakar Boruvka on minimum spanning tree problem - Translation of both the 1926 papers, comments, history, Prague, Czech Republic

[6] André R. Brodtkorb, Trond R. Hagen, Martin L. Sætra: Graphics processing unit (GPU) programming strategies and trends in GPU computing, Oslo, Norway

[7] Effi Levi1, Roi Reichart, Ari Rappoport1: Edge-Linear First-Order Dependency Parsing with Undirected Minimum Spanning Tree Inference, Institute of Computer Science, The Hebrew University IE&M faculty, Technion, IIT.

[8] Ning Kang: Using the Cache Analysis Tool to Improve I-Cache Utilization on C55x Targets, December 2003

[9] Graham, R.L., Hell, Pavol, On the history of the minimum spanning tree problem, Annals of the History Computing, 7(1) (1985), 43-57.

[10] Ardhendu Mandal, Jayanta Dutta, S.C. Pal, A New Efficient Technique to Construct a Minimum Spanning Tree, Vol. 2 Issue 10, (2012).

[11] Vibhav Vineet, Pawan Harish, Suryakant Patidar, P.J. Narayanan, Fast Minimum Spanning Tree for Large Graphs on the GPU, International Institute of Information Technology, Hyderabad, India

[12] Cristiano da Silva Souse, Artur Matriano, Alberto Proenca, A Generic and Highly Efficient Parallel Variant of Boruvka's Agorithm, 23rd Euromicro International Conference on Parallel, Distrubuted, and Network-Based Processing, 2015.

[13] Ta-Wei Liu:Grpahic Card, Publication number: US 20080018653 A1, Jan 24, 2008

[14] Wookhyun Han, Hoon Sung Chwa, Hwidong Bae, Hyosu Kim, Insik Shin: GPU-SAM: Leveraging multi-GPU split-and-merge execution for system-wide real-time support, School of Computing,KAIST, Daejeon, South Korea.

[15] Pasqua D'Ambra ,Salvatore Filippone: A parallel generalized relaxation method for high-performance image segmentation on GPUs, Rome, Italy.

[16] Jun Sui , Chang Xua , S.C. Cheungc , Wang Xi , Yanyan Jiang , Chun Cao , Xiaoxing Ma , Jian Lu: Hybrid CPU–GPU constraint checking: Towards efficient context consistency, china.

[17] Yuki Sugimoto,Fumihiko Ino,Kenichi Hagihara: Improving cache locality for GPU-based volume rendering, Japan.

[18] Jaroslav Nesetril, A few remarks on the history of MST problem, Archivum Mathematicum, Brno, 33 (1997), 15-22. Prelim. version in KAM Series, Charles University, Prague,(1997), 97-338

[19] Jack Donnell: Java Performance Profiling using the VTune™ Performance Analyzer

[20] Boruvka, O. jistem problemu minimalnim, Prace Moravske Prirodovedecke Spolecnosti, Vol. 3, 1926, 37-58.

[21] Markus Geimer, Michael Gerndt. Sameer Shende, Bert Wesarg, Brian Wylie: Hands-on Practical Hybrid Parallel Application Performance Engineering, EuroMPI Conference, Vienna, Austria, September 23, 2012

[22] 11th DIMACS Implementation Challenge in Collaboration with ICERM:Steiner Tree Problems

Available at: http://dimacs11.cs.princeton.edu/downloads.html