Implementing graph traversal algorithms on parallel computing architecture with GPU

Resham Jhangiani
Department of Computer Science
California State University, Fullerton
Fullerton, USA
reshamjhangiani@csu.fullerton.edu

***Abstract*—This paper considers the importance of graph data, breadth first search, depth first search tree traversal algorithms and current day approach of GPU programming to implement them in on large graphs. How does Google maps provide the best route for getting to the destination, how does an internet router forward data packets avoiding network traffic and minimizing delay?**

Keywords*—Graph, BFS; DFS; GPU; Parallel computing.

I. Introduction

How does Google maps provide the best route for getting to the destination? How does an internet router forward data packets avoiding network traffic and minimizing the delay? How does an ecommerce website recommend products or Amazon track its shipment? How does Facebook and other social websites maintain user data. Almost all present day existing applications rely on graph data analysis. Graph have evolved as powerful and versatile data structures.

Complex dynamic relationship in highly connected data is well represented by graphs. A graph is a collection of vertices and edges where vertices are representatives of real world entities and edges are the relationship between the entities. Affordable and scalable performance data analytics on huge data can be achieved using multicore CPUs. However multicore CPU architecture deployed in large cluster for data analytics on bigdata is still unaffordable and bottlenecked. Demanding need for high-performance, continuous scalability, real-time analytics and faster memory access conclude ‘Parallelism’ as the ideal solution.

A cost-effective optimum approach for implementing parallel algorithms on bigdata is with the use of Graphical Processing Unit (GPU). Massive multi-threading architecture of GPU support parallel computing. This survey paper discusses the approach of scalable processing on graph data, using popular graph traversal algorithms Breadth first search (BFS), Depth first search (DFS) on a single GPU computing architecture. GPU cores are designed for performing repeated, similar instructions in parallel. Graph traversal algorithm are sequential in nature, they involve more data transfer then computation in contras GPU are suited for parallel repeated computation tasks. Thus, implementing these traversal algorithms on GPU demand a strategic approach and memory management. The survey paper also highlights the recent work aimed at obtaining optimum efficiency for executing graph traversal with GPU architecture.

The survey paper is organized as: a brief introduction to parallel bfs, dfs algorithm; memory optimization for graph traversal for gpu architecture; understanding graph properties to speed up the execution.

Basic graph, and adjacency list, matrix

II. PARALLEL BFS IMPLEMENTATION

BFS starts from tree root/ source and explores neighbors level by level. It uses queue data structure to keep track of the visited nodes of graph. Designate gpu threads equal to the number of edges in the graph for parallel implementation of bfs. The most efficient way to represent a graph is using adjacency list; for CUDA programming compact adjacency list using array data structure is preferable. Queue implementation for maintaining a record of non-visited nodes is an overhead in GPU; maintaining an additional array index is an overhead and re-configuring grid at every kernel iteration will be computational expensive.

Single source shortest path, for a graph G (V, E) with positive weights the shortest path from a source vertex S to destination vertex V_d can be acquired by implementation of the breadth first search. Dijkstra’s Algorithm is a sequential solution to the single source shortest path problem with a time complexity of O (V log

V + E) it optimizes to this time complexity partly due to implementation of BFS at intermediate level. Performance of the Dijkstra's algorithm can be optimized with implementation of parallel bfs algorithm with GPU architecture.

A. *Algorithm*

III. Parallel DFS Implementation

Depth First Search(DFS) is a tree/graph traversal algorithm that begins at the root node and explores every element down the branch before backtracking. DFS is a classic backtracking algorithm and thus is inherently sequential and recursive in nature. DFS uses a stack data structure for keeping track of the traversed nodes of a graph. Due to sequentially recursive nature of DFS it is unsuited for GPU implementation. The overhead for data access is far more than the computation of traversing which makes it unsuitable for GPU architecture. In NVIDIA's effort to implement parallel DFS [ref], an approach of implementing DFS on directed acyclic graph (DAG) and Directed Tree (DT) is implementing an algorithm with BFS-like approach, executing thrice or less to obtain pre-order and post-order DFS. This GPU based algorithm proves to outperform sequential DFS by six times. The time complexity is declared to be reduced from O ($\log^2 n$) to O (n log n) for n is the no of processors.

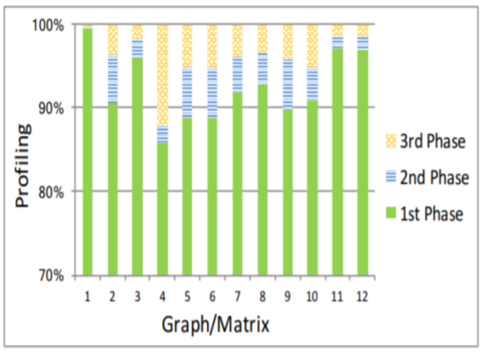
The citation states that assuming DFS implementation unsuited for GPU is false; and deeper knowledge of the type of graph the order DFS traversal is important factor that also needs to be considered. Traversal of nodes in DFS fashion can be unordered or lexicographic, and class of the graph can be directed, planar or general. For DAG and planar graph lexicographic DFS traversal is NC whereas unordered DFS traversal for general graph is RNC. The lexicographic DFS traversal approach is to iterate graph in parallel BFS-like manner to traverse the parents before any children. On obtaining the history of possible edges to parent node; it is easier to get the DFS pre- and post- order for every node. Top-down approach (roots to leaf node) on a DT the recursive nature of pre- and post- order can be maintained.

A. *Approach 1: Prallel DFS with custom data structure*

A data structure to store all the possible paths for the graph and or tree is needed; the execution of this data structure depends on the type of graph, tree and the size. Stack implementation may not be the best optimized data structure for parallel implementation of DFS. A customized data structure depending on the requirements is recommended. A linked list of blocks of memory is proposed in []. The first two elements are used to store parent and the tail size. Later part of the block is used to store the path. If the path fits in the given block size than the first element of the block denotes the size. If the path does not fit the given size, a new block is used. The first node of this new block points to the parent block. This chaining of block enables memory optimization. The comparison of path in parallel is possible with this data structure.

B. *Algorithm 1: Prallel DFS with custom data structure*

1. Let Graph G (V, E) and its adjacency matrix A
2. If DAG convert to DT, Else continue- PHASE 1
3. Compute Sub graph- PHASE 2
4. Compute pre-order, post-order- PHASE 3
5. Result: parent, pre-order, post-order for all node.



Time consumed by the three phases.

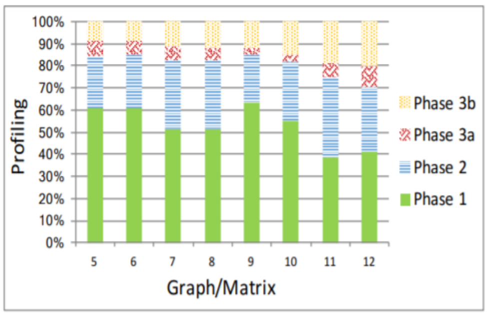
C. *Approach 2: Parallel DFs with SSSP technique.*

This approach overcomes the limitation of the approach one i.e. the time consumed for converting DAG to DT. Implementing single source shortest path (SSSP) for every node in graph results precisely in the path selected by DFS. Thus, the DT is obtained from DAG implicitly by executing SSSP. Custom SSSP is more significant in terms of speedup as there is access to queue of the nodes at every iteration.

D. *Algorithm 2: DFS with SSSP*

1. Let Graph G (V, E) and adjacency matrix A

2. Compute DAG Edge Weights
3. Transform DAG to DT
4. Obtain shortest node with SSSP



Time consumed by the three phases

Thus, the success of parallel implementation of DFS on graphs depends on multiple factors the most important being the connectivity of nodes for the DAG. Sparsity and density of graph will affect the efficiency of algorithm. Customized data structure, or customizes intermediate algorithm like SSSP can help achieve parallel implementation of the DFS with GPU.

IV. Challenges

Large graphs with highly dense network and diverse topology faces memory challenges, as graph traversal algorithms are pre-dominantly data access procedures than computing ones. Random access at frequent rates leads to poor GPU memory efficiency. Implementing shared memory with GPU architecture is not an easy implementation either. For enhancing efficiency processing and organizing graph data is one possible solution. Vertex centric, edge centric, data centric, path centric and matrix based are few processing models that can be used. Of these models vertex centric and edge centric are preferred as these models are effective for any type of graph. Vertex centric modeling creates good spatial locality when accessing nodes, the accessing of edges is however random. Edge centric enables sequential access to edges; however, vertices access is random. A general approach is to pre-process the graphs with CPU as they are stored in CPU memory. Later initialize vertex attribute array and load into GPU memory. Partition the edges into groups and invoke the kernel function, the kernel than launches number of threads corresponding to the number of edges in each partition. Clearly there is a data transfer overhead.

A. *Active vertices/ Dynamic frontiers*

The number of active vertices vary dramatically for a graph depending on the traversal algorithm used. BFS, DFS begin from the root vertex and dynamically keep adding new frontiers (neighbors) as it keeps traversing. Keeping track of these active vertices is a challenge. Using a Boolean array data structure is a the most general approach. However, this is associated with few overhead like; at every super step the array needs to be updated in CPU memory, we assume that this array will fit in the GPU memory this might not be the case.

V. Algorithms for Big Data Uncertainty

A. *Identify the Headings*

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include ACKNOWLEDGMENTS and REFERENCES, and for these, the correct style to use is “Heading 5.” Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract,” will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1,” “Heading 2,” “Heading 3,” and “Heading 4” are prescribed.

B. *Figures and Tables*

1) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1,” even at the beginning of a sentence.

TABLE I. TABLE STYLES

Tabl e Head	**Table Column Head**		
	Table column subhead	***Subhe ad***	***Subhe ad***
copy	More table copy[a]		

[a.] Sample of a Table footnote. *(Table footnote)*

[b.]

Fig. 1. Example of a figure caption. *(figure caption)*

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity "Magnetization," or "Magnetization, M," not just "M." If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write "Magnetization (A/m)" or "Magnetization

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi resolution TIFF or EPS file with all fonts embedded) because this method is somewhat more stable than directly inserting a picture.

To have non-visible rules on your frame, use the MSWord "Format" pull-down menu, select Text Box > Colors and Lines to choose No Fill and No Line.

(A (m(1)," not just "A/m." Do not label axes with a ratio of quantities and units. For example, write "Temperature (K)," not "Temperature/K."

VI. Conclusion

Acknowledgment *(Heading 5)*

The preferred spelling of the word "acknowledgment" in America is without an "e" after the "g." Avoid the stilted expression "one of us (R. B. G.) thanks ...". Instead, try "R. B. G. thanks...". Put sponsor acknowledgments in the unnumbered footnote on the first page.

References

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

Number footnotes separately in superscripts. Place the actual footnote at the bottom of the column in which it was cited. Do not put footnotes in the reference list. Use letters for table footnotes.

Unless there are six authors or more give all authors' names; do not use "et al.". Papers that have not been published, even if they have been submitted for publication, should be cited as "unpublished" [4]. Papers that have been accepted for publication should be cited as "in press" [5]. Capitalize only the first word in a paper title, except for proper nouns and element symbols.

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] Ward JS, Barker A (2013) Undefined by data: a survey of big data definitions. Tech. Rep. https://arxiv.org/abs/1309.5821

[2] Qiu J, Wu Q, Ding G, Xu Y, Feng S (2016). A Survey of machine learning for big data processing. EURASIP J adv Signal Proc 2016(1):67

[3] Zhou L, Pan S, Wang J, Vasilakos AV (2017). Machine learning on big data: opportunities and challenges. Neurocomputer 237:350-361.

[4] Nielsen FA (2008). Clustering of scientific citations in wikipedia. Tech. Rep. https://arxiv.org/abs/0805.1154

[5] Arora S, Ge R, Kannan R, Moitra A (2016). Computing a non-negative matrix factorization-provably. SIAM J Comput 45(4):1582-1611

[6] Moitra A (2016). An almost optimal algorithm for computing non-negative rank. SIAM J Comput 45(1):156-173

[7] Cygan M, Formin FV, Kowalik L, Lokshtanov D, Marx D, Piliczuk M, Saurabh S (2015). Parameterized algorithms. Springer, Cham

[8] Czumaj A, Sohler C (2007). Sublinear-time approximation algorithms for clustering via random sampling. Random Struct Algorithms 30(1-2):226-256

[9] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[10] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[11] M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.

[12]