International Journal of Technical Research and Applications e-ISSN: 2320-8163,
www.ijtra.com Volume 4, Issue 3 (May-June, 2016), PP. 488-491

A FAST OPTIMIZED PARALLEL GRAPH ALGORITHM ON GPGPU

C.P.Mogal [1], C.R. Barde[2]
Dept. of Computer Engineering, G.E.S.'s R.H. Sapat College of Engineering, Management Studies and Research, Nasik Affiliated to Savitribai Phule Pune University
[1]mogalcp.31@gmail.com
[2]erchandubarde@gmail.com

***Abstract*— Numerous Classic and emerging applications uses graph processing as core component. NVIDIA proposed CUDA (Computed unified device architecture) which increases the computation substantially with power of GPU. Graph algorithm is simplest way to improve the graph processing performance. In spite of improve programmability of GPU, writing efficient and correct program is very difficult and the task become more challenging and complex in case of graph due to irregular structures and large size of graph data. We proposed few fundamental graph algorithms like BFS, shortest path algorithm which efficiently accelerate and optimized graph processing with GPGPU (general Purpose graphical processing unit) technique.**

***Index Terms*-CUDA, GPGPU, Graph processing, BFS, Shortest path.**

I. INTRODUCTION

Graph algorithms form fundamental to many disciplines and are common in scientific and engineering applications. The need for high computation power and low price results into areas such as Graphics processing units (GPU) [20]. The sectors like defence, medicine, GIS widely using applications that are depend on graph processing. This paper deals with how to use GP-GPUs efficiently for graph algorithms and efficient GPU implementations for the various problems thus faced. The algorithms focus on minimizing irregularity at both algorithmic and implementation level. It also deals with analysis of all pair shortest path algorithm by performing on different memories of GPU. Literature survey shows that the graph processing is complex because of the irregularity of graph structure and the large size. The CUDA programming simplifies and improves the performance of graph processing with optimized fundamental graph algorithms still it is difficult to write the effective program. In this paper we proposed the graph algorithms which are BFS (Breadth-First Search) and APSP (All pair Shortest Path) through use of GPU. The proposed work will help to reduce data transfer rate CPU to GPU, reduced access of global memory by using shared memory and memory management.

II. RELATED WORK

Substantial amount of work has been done in graph domain.

Recently P.Harish and P. Narayanan proposed the fast graph algorithm on large graphs. Medusa implementation of GPGPU program simplifies graph processing. It proposed optimized technique which improves system performance It Provides different APIs to write GPU graph algorithm. It hides a GPU programming with help of APIs. Medusa L_threads to vertex and having L_edges to edge. Medusa uses APIs like add Edge and add Vertex to initialized graph structure. It shows result by comparing with other manually tunes algorithms, and hence claim for accelerated and enhanced performance in simplified way. [1] P. Harish and P.J.Narayanan Algorithms for large graph. They proposed the graph algorithm like BFS, SSSP, and APSP on large graph which is faster than previous work.[2] It finds that for scientific application is lack of higher precision than regular gamming applications.[7]Work achieves a speed up of 9 to 12 times over the best sequential CPU implementation. For instance, implementation finds connected components of a graph of millions of nodes and equivalent edges in about few milliseconds on a GPU, given a random edge list. [6] has presented a BFS implementation on the GPU. It is most suitable for accelerating sparse and near regular graphs, which are widely seen in the field of EDA. Edge-Message-Vertex (EMV) for fine-grained processing on vertices and edges proposed by [19]. EMV is specifically tailored for parallel graph processing on the GPU. [8] Proposed APSP algorithm to calculate shortest path but fails to achieve optimal result for sparse graph. [3] [11] primarily explained parallel algorithms and implementations for solving the single source shortest path problem on large-scale graph instances. [9][10] presented parallelization of BFS tailored to the GPU's requirement for large amounts of fine-grained, bulk-synchronous parallelism. Merrill and Garland, presented Scalable GPU graph traversal. Also demonstrated that GPUs are well-suited for sparse graph traversal. [22] explain basic graph algorithm in parallel.

III. PROPOSED SYSTEM

In the normal world, numerous unpredictable, interrelated occasions are occurring in the meantime, yet inside a fleeting grouping. Despite the fact that CUDA with GPGPU engineering enhance execution of chart registering measures composing powerful program for unpredictable extensive

diagram structure is perplexing errand. The Proposed implementation of a graph processing for improving performance of application like data mining ,network analysis, vehicle routing etc. where there is large graphs are used .

A. Architecture diagram

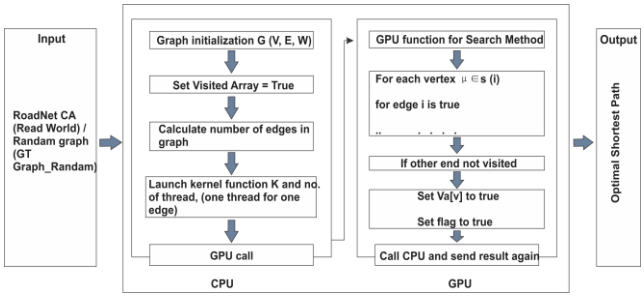

Fig 1: - System Architecture Diagram

The figure 1 demonstrates framework of system architecture. In first stage host designate device memory for graph information. Graph chart Data then exchange from CPU primary memory to GPU memory. In the second stage CPU teach the GPU for preparing, in that CUDA Kernel capacity characterized for BFS portion work that execute the edge-driven and in All Pair most limited way calculation subgroup the edges that execute in parallel methodology. In the third stage string execution administrator of GPU executes portion capacity in all centres of GPU in parallel. At the last gather result from all threads and exchange to CPU.

CUDA Memory Organization an expansive segment of processing time on the device is spent on information development, particularly the perusing of information into the individual thread. Since there are several number-crunching units on the GPU, the memory transfer speed of the PC chip is frequently the bottleneck or significant time shopper. With four sorts of memory accessible on the GPU, picking the right memory sort and using it accurately is essential for greatest velocity. [20]

B. Dataset

Our experimental dataset includes two categories of graphs: real-world and random graph. We use the GTgraph graph generator to generate Random Graph. For APSP Dataset of random graph is used and RoadNet CA is used for BFS with detailed as follows [16].

1) Dataset Name - RoadNet-CA
Dataset statistics
Nodes 1965206
Edges 5533214
2) Dataset Name –Random graph / RoadNet-CA
Dataset statistics
Nodes 65536
3) Dataset Name –WikiTalk
Dataset statistics
Nodes 4096

TABLE I
DATASET ROADNET-CA DETAILS

Additional Info

number of rows	1,971,281
number of columns	1,971,281
nonzeros	5,533,214
# strongly connected comp.	8,713
explicit zero entries	0
nonzero pattern symmetry	symmetric
numeric value symmetry	symmetric
type	binary
structure	symmetric

Above dataset will gives details about dataset used for carry experiment.

C. BFS (Breadth First Search) Algorithm

It is fundamental block for many of high level graph analysis and is use find out reachable from given source. To increase the performance of applications must have to increase the performance of graph algorithms. [11] [8] In this system BFS is first graph algorithm, BFS algorithm achieve parallelism through edge wise.

1: Start
2: Create vertex array V and edge array E for given G(V, E)
3: Create visited array & frontier array Va and Ea of size V
4: Initialize Va and Ea to false
5: Select source = = 0
6: Create cost array Ca and set to 0
7: Calculate no of blocks and no of threads
8: Allocate host memory & allocate device memory
9: Copy G data from CPU to GPU
10: Set the source node S as true
11: Copy visited node, nodelist, edgelist to device memory
12: Allocate result array size and copy on device memory
13: While Ea not empty do
14: for each vertex / edge V, E in parallel
15: Call CUDA kernel
16: CUDA thread sync( )
17: Free all memory and display cost Re-assign number

CUDA parallel function ( )
1: Start
2: calculate thread_Id
3: If (tid < No. of node and graph visited [ ]
4: for (all the edges of current vertex check already visited) if not then add to mask array
5: If (! visited [graph])
6: Cost= cost + [source to visited vertex]
7: E[cnt ++]
8: Return to CPU call
9: End

D. APSP(All Pair Shortest Path) Algorithm

The All Pair shortest path is used to find shortest distance between all pair of vertices having intermediate node k where, k is number of vertices [8]. In proposed algorithm graph is converted to adjacent matrix and with help of different GPU memories like global, texture and shared memory the program execution is carried out. For APSP, problem input is in the form of adjacency matrix and processing data are stored into shared memory to reduce latency for improving performance. [24].

1: Start
2: Read V no. of vertex and E no. of edges of graph G (V,E)
3: Create array V[ ], E[ ] for vertex and edge
4: Create adjacency matrix A from given graph G
5: Allocating size of CPU &GPU variable using CUDAmalloc()
6: Transfer data from CPU to GPU usingCUDA memcpy()
7: Start computing on Host()
8: Calculate no. of block and thread for parallel processing
//Parallel CUDA code to find out Min[i,j] i.e
// A[i,j]= Min(A[i,j]+A[k,j]) using global memory

--global cuda( )—
Step 1: Calculate thread_Id.x
Step 2: For each i < No matrix esteem do
Step 3: if (i < N && j < N)
Step 4: Find Min (i,j) using k=1
Step 5: A[i,j]=(Min[i,j]+A[k,j])
Step 6: Update A[i,j]
Step 7: Do this for all number of vertices
Step 8: Copy result back to CPU
Step 9: Free CPU memory
Step 10: End

IV. EXPERIMENTAL SETUP AND RESULT SETS

A. Software and Hardware

Languages used: CUDA

Software Description Interface: A single machine with CUDA capable GPU, Ubuntu and NVIDIA CUDA Toolkit is required for running the application

We test the performance of the serial implementation of APSP algorithm and parallel implementation on graph that having large number of vertices. Following tables illustrate the result set of results of proposed system.

Table II and Table III illustrate Proposed BFS algorithmic time taken on Fermi and Kepler architecture

TABLE II
BFS Algorithm run time using 256 Block/Threads

Dataset	No of Vertex	Algortihm Run time Gt740m	Algortihm Run time Gtx680	Algortihm Run time Quadro2000	MAX THREAD/Block
WikiTalk	4096	14.707712ms	9.63ms	7.13ms	256
roadNet-CA.txt	65536	147.012573 ms	70.75ms	69.98ms	256
roadNet-CA.txt	5533214	1997.463300 ms	852.144ms	937.57ms	256

TABLE III
BFS Algorithm run time using 1000 Block/Threads

Dataset	No of Vertex	Algortihm Run time Gt740m	Algortihm Run time Gtx680	Algortihm Run time Quadro2000	MAX THREAD/Block
WikiTalk	4096	9.250784 ms	10.30ms	5.04ms	1000
roadNet-CA.txt	65536	102.761436 ms	71.16ms	69.39ms	1000
roadNet-CA.txt	5533214	1208.164103 ms	911.129ms	962.35ms	1000

Following Table IV shows algorithm of APSP on GT740M

TABLE IV
APSP Algorithm run time Speedup

GT740M						
MatrixSize	Threads	CPU Time	GPU-Global Memory Time	GPU-Pinned Memory	SpeedUp using GlobalMemory	Speedup using PinnedMemory
4*4	4	0.0816ms	0.0676 ms	0.0788 ms	1.21	1.04
40*40	1000	8.91 ms	10 ms	7.25 ms	0.89	1.23
400*400	1000	8.11e+03 ms	1.11e+03 ms	1.11e+03 ms	7.28	7.32
4000*4000	1000	6.28e+06 ms	1.51e+05 ms	7.18e+05 ms	41.5	8.74

Following figure 2 illustrate comparison between existing system and Proposed System

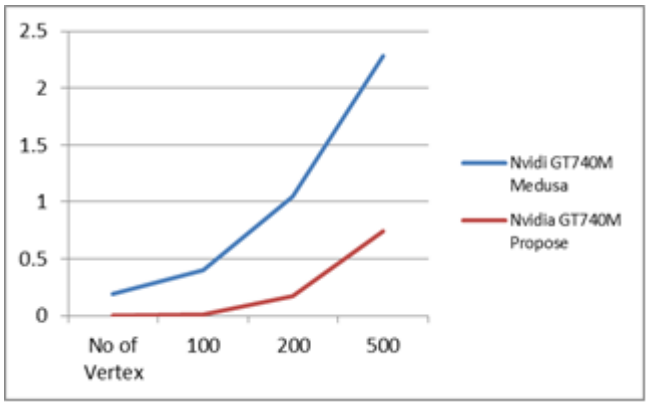

Fig 2-Performance comparison exiting system Vs Proposed system

V. CONCLUSIONS

The BFS and All pair shortest path algorithms with edge centric methodology of computation proposed in paper. The results shows there is substantial performance improvable.The result are same as actual results. As future work, this application can be ported to multiple GPU devices that will run in parallel. As the number of GPU cards used increases, a proportional speed up of the application is expected. The idea of parallelizing the pattern matching algorithm can be extended to parallelizing the pre-processing part. This process is expected to produce enormous speed as all the costly computations can be offloaded to the GPU. By using overlap kernel execution with data transfers there can be increase speed of execution.

VI. ACKNOWLEDGMENT

We are glad to express our sentiments of gratitude to all who rendered their valuable guidance to us. We would like to express our thanks to All Staff of Computer Department of G.E.S.'s R.H. Sapat COE., Nashik. We are also present our gratitude towards all the researcher for their valuable work in domain of graph computing and for encouraging new one for further research and study. We thank the anonymous reviewers for their comments.

REFERENCES

[1] Jianlong Zhong and Bingsheng He ,"Medusa-Simplified Graph Processing on GPUs".IEEE Transaction on parallel and distributed system, Vol. 25, NO. 6, June 2014.

[2] J. P. Harish and P. J. Narayanan, "Accelerating large graph algorithms on the GPU using cuda," in Proceedings of the 14th international conference on High performance computing, HiPC'07, (Berlin, Heidelberg), pp. 197-208, Springer-Verlag, 2007.

[3] David A. Bader and Kamesh Madduri,"Designing Multithreaded Algorithms for Breadth-First Search and connectivity on the Cray MTA-2," ICPP, pp. 523-530, 2006.

[4] J.D. Owens, D. Luebke, N.K. Govindaraju, M. Harris, J. Kruger, "A Survey of General-Purpose Computation on Graphics Hardware,"in Proc Eurographics, State Art Rep., pp. 21-51, 2005.

[5] V. Vineet and P. Narayanan, "CUDA cuts: Fast graph cuts on the GPU,"in Computer Vision and Pattern Recognition Workshops, 2008.CVPRW'08. IEEE Computer Society Conference on, pp. 1-8, IEEE,2008.

[6] L. Luo, M. Wong, and W.-M. Hwu, "An Effective GPU Implementation of Breadth-First Search," in Proc. DAC , pp. 52-55, 2010.

[7] J. Soman, K. Kishore, and P J Narayanan ,"A Fast GPU Algorithm for Graph Connectivity", IEEE Transaction on parallel and distributed system, Vol. 2, June 2012.

[8] *F G.J. Katz and J.T. Kider, "All-Pairs Shortest-Paths for Large Graphs on the GPU ", in Proc. Graph. Hardware, 2008, pp. 47-55.*

[9] D. Merrill, M. Garland, and A. Grimshaw, "Scalable GPU graph traversal," in Proceedings of the 17th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming, PPoPP '12, pp. 117-128, 2012.

[10] Merrill, D. et al. 2011. High Performance and Scalable GPU Graph Traversal. Technical Report CS2011-05. Department of Computer Science, University of Virginia.

[11] J. Bader, D.A. et al. On the Architectural Requirements for Efficient Execution of Graph Algorithms. 2005 International Conference on Parallel Processing (ICPP'05) (Oslo, Norway), 547-556.

[12] E. Dekel, D. Nassimi, and S. Sahni, "Parallel matrix and graph algorithms," SIAM Journal on computing, vol. 10, no. 4, pp. 657-675, 1999.

[13] "10th DIMACS implementation challenge,"http://www.cc.gatech.edu/dimacs10/index.shtml accessed on oct 20th, 2015

[14] Nvidia CUDA official site for developer website https://developer.nvidia.com/cuda-zone

[15] CUDA Zone. Official webpage of the nvidia cuda api. Websitehttp://www.nvidia.com/object/cuda_home_new.html

[16] GTgraph: A suite of synthetic random graph generators: https://sdm.lbl.gov/kamesh/software/GTgraph/Accessed: 2011-07-11

[17] Cormen, Leiserson,Rivest, And Stein, " Introduction to Algorithms", 2012,,Book

[18] NVIDIA, "Cuda: Compute unified device architecture programming guide ". Technical report, NVIDIA, 2014.

[19] J. Zhong and B. He, "Parallel graph processing on graphics processors made easy," Proc. VLDB Endow., vol. 6, pp. 1270-1273, Aug. 2013

[20] C.P.Mogal and C.R..Barde ,"Review Paper on Optimised and accelerated Parallel Graph Algoritm on GPGPU " IJCSMC, Vol. 4,Issue. 12, December 2015, pg.103 – 106.

[21] J. Zhong and B. He, "Kernelet: High-throughput gpu kernel executions with dynamic slicing and scheduling," IEEE Transactions on Parallel and Distributed Systems, vol. 99, no. Pre-Prints, p. 1, 2013.

[22] M. J. Quinn and N. Deo, "Parallel graph algorithms," ACM Computing Surveys (CSUR), vol. 16, no. 3, pp. 319-348.

[23] M. Makulla and R. Berrendor," Evaluating Parallel Breadth-First Search Algorithms for Multiprocessor Systems ",iariajournals,vol 7,2014

[24] C.P.Mogal and C.R.Barde ,"A fast and optimized graph algorithm on GPGPU,"Cpgcon2016 conference, PCCOE, Savitribai phule Pune University,24 March 2016