IN-PLACE RECURSIVE APPROACH FOR ALL-PAIRS SHORTEST PATHS PROBLEM USING OPENCL

Manish Pandey, Himanshu Pandey, Dr. Sanjay Sharma
Department of CSE, Maulana Azad National Institute of Technology
Bhopal, India
contactmanishpandey@yahoo.co.in, himanshu2mail@gmail.com,
sharmasanjay@manit.ac.in

Abstract

The all-pairs shortest paths (APSP) problem finds the shortest path distances between all pairs of vertices,and is one of the most fundamental graph problems. In this paper, a parallel recursive partitioning approach to APSP problem using Open Computing Language (OpenCL) for directed and dense graphs with no negative cyclesbased on R-Kleene algorithm, is presented, which recursively partitions dense adjacency matrix into sub-matrices and computes the shortest path. Graphics Processing Units (GPUs) are massively parallel in nature and provide high computational speedup at very low cost in comparison to other very costly High Performance Computing (HPC) systems. Most common technique for Graph representation is to store it in the form of adjacency matrix and GPUs are highly suitable for performing matrix computations in parallel. OpenCL is a framework which provides unified development environment for executing programs in heterogeneous platforms. Using OpenCL, we can execute program on GPUs and/or CPUs. Our implementation is mainly targeted towards executing OpenCL kernels on GPU. In designing effective OpenCL programs, data transfers between host and device memory should be minimized. Our approach is in-place in nature, so it does not require additional memory space while performing computation and entire data movement takes place in a bulk between host and device memory.

Keywords - All-pairs shortest path; OpenCL; GPU; Adjacency matrix; in-place.

1 INTRODUCTION

The all-pairs shortest path (APSP) problem is to find the minimum path distances for all source-destination pairs in a graph. The cost is the sum of the weights of edges composing the path. This problem is widely applicable in various areas like geographical information systems, intelligent transportation systems, IP routing [4].

There are many approaches to solve APSP problem like Floyd-Warshall (FW) algorithm or running single source shortest path (SSSP) algorithms for all vertices in the graph etc. FW algorithm requires O (N^3) time complexity where N is the number of vertices in a graph [7].

These sequential algorithms have very high time complexity and for very large graph involving millions of vertices, such algorithms become impractical. Parallel algorithms can achieve practical time for APSP problem for very large graph but hardware used in these is very expensive. Bader et al. [6] have used supercomputer CRAY MTA-2 to perform breadth-first search on very large graph.

Initially Graphics Processing Unit (GPU) was hardware designed to perform some graphics acceleration tasks like rendering, gaming, image processing etc. In recent years, GPU is evolved to perform general purpose computation and there exist GPU implementations [5] in various fields like linear algebra, image processing, computer vision, signal processing etc. GPU has become a cost-effective platform in comparison to other very expensive High Performance Computing(HPC) systems. GPU's massive parallel hardware is well suited for performing matrix operations [3].

Open computing Language (OpenCL) is framework for writing programs that execute in parallel on different compute devices such as CPUs and GPUs from different vendors AMD, Intel, ATI, Nvidia etc.

In this paper, we present OpenCL parallel implementation for recursive partitioning approach to APSP problem based on R-Kleene algorithm [1] which works by recursively partitioning the adjacency matrix of graph into sub-matrices and performing computation on that.

This paper is organized as follows: Section 2 describes related work. Section 3 shows description of OpenCL framework. Section 4 describes the APSP problem and OpenCL parallel implementation for recursive approach. In section 5 experimental results are presented. Section 6 presents Conclusion and future work.

2 RELATED WORK

Paolo et al. [1] have presented a recursive divide-and-conquer algorithm for the solution of the all-pair shortest-path problem for directed and dense graphs with no negative cycles. They have formulated a recursive algorithm where the result is computed in-place for dense adjacency matrices. They have presented a quantitative measure for performance of R-Kleene in terms of *millions of integer instructions per second*(MIPS) determined as N^3*/ (execution of algorithm in seconds)*.

This R-Kleene algorithm employs a different schedule of path computation rather than classical iterative FW algorithm.R-Kleene inherits the property of matrix-multiply (MM), and it is *cache oblivious*achieving optimal data cache utilization [2]. It shows efficient register utilization because of its recursive nature and a large number of operations can be run independently in parallel [1].

Our approach for APSP problem is OpenCL parallel implementation of R-Kleene based algorithm as OpenCL kernels, which are executed in parallel on massively parallel GPU threads. In OpenCL, these threads are referred as Work-items. Due to execution as parallel threads, it shows speedup over R-Kleene algorithm. Using MM property, our implementation targets mainly GPU as OpenCL device because GPUs are highly suitable for MM operations and it also increases data locality.

3 OPENCL FRAMEWORK

OpenCL is an open royalty-free standard for general purpose parallel programming across CPUs, GPUs and other processors, giving software developers portable and efficient access to the power of these heterogeneous processing platforms [10]. OpenCL framework is divided in following four models: Platform model, Execution model, Memory model and Programming model.

3.1 OpenCL Platform model

“Fig. 1” shows the OpenCL platform model. It consists of a hostconnected to one or more OpenCL devices. An OpenCL device is divided into one or morecompute units (CUs) which are further divided into one or more processing elements (PEs).Computations on a device occur within the processing elements.

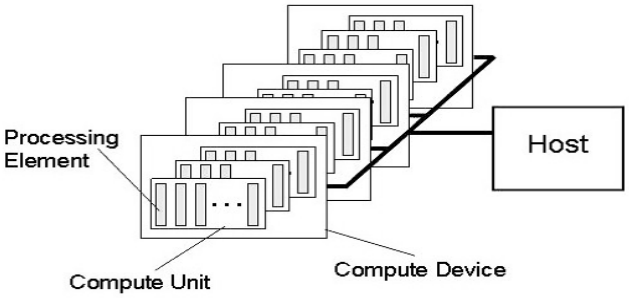

Fig.1. OpenCL Platform Model

TheOpenCL application submits commands from the host to execute computations on theprocessing elements within a device. The processing elements within a compute unit execute asingle stream of instructions as SIMD units.

The definition of compute unit is different for different vendors. In AMD, each compute unit contains many*Stream Cores* (or SIMD Engines), and stream cores contain individual processing elements. In NVIDIA, *Stream Multiprocessors* (SMs) are called compute units.

“Fig. 2” shows a simplified diagram of an AMD GPU compute device. Different GPU compute devices have different characteristics (such as the number of compute units), but follow a similar design pattern [8].

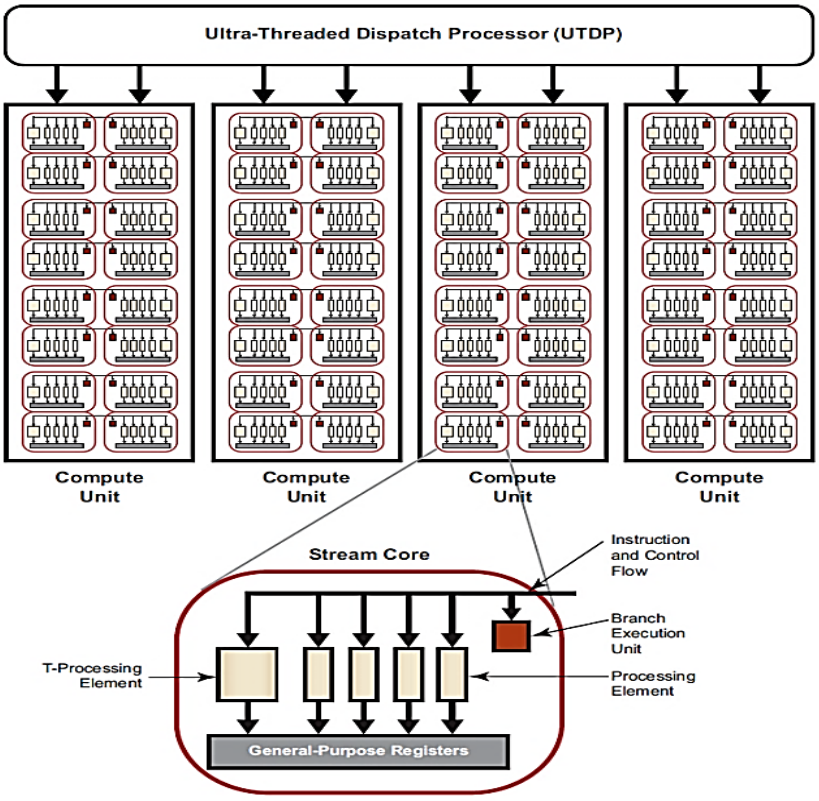

Fig.2. AMD GPU compute device [8]

3.2 OpenCL Execution Model

The OpenCL execution model comprises two components: kernels and host programs. Kernels are the basic unit of executable code that runs on one or more OpenCL devices. The host program executes on the host system, defines devices context, and queues kernel execution instances using command queues. Kernels are queued in-order, but can be executed in-order or out-of-order [13].

OpenCL also allows grouping of work-items together into work-groups. The size of each work-group is defined by its own local index space. OpenCL only assures that the work-items within a work-group execute concurrently. Synchronization is only possible between work-items within a work-group. The index space spans an N-dimensioned range of values and thus is called an NDRange. N in this N-dimensional index space can be 1, 2, or 3 only.

A work-item can be identified by its global ID (g_x, g_y) or by the combination of its local ID (l_x, l_y) and work-group ID (w_x, w_y). A relation between global ID and local ID can be defined as following:

$$g_x = w_x * L_x + l_x$$

$$g_y = w_y * L_y + l_y$$

Where L_x and $L_y$are work-group size in *x*-direction and *y*-direction respectively.

“Fig. 3” shows how the global IDs, local IDs, and work-group indices are related for a two-dimensional NDRange. NDRange index space of size G_x by G_y (12x12) is divided into 9 work-groups, each having size 3x3. The shaded block has a global ID of (g_x, g_y) = (6, 5) and a work-group plus local ID of (w_x, w_y) = (1, 1) and (l_x, l_y) = (2, 1).

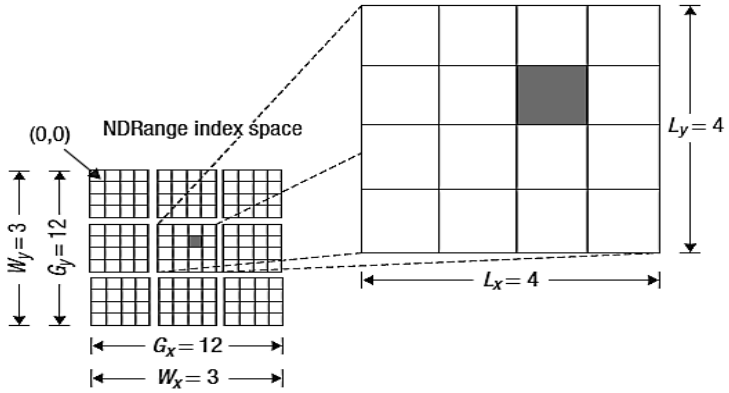

Fig.3. Relation between global ID and local ID, work-group ID in 2-D index space [9]

3.3 OpenCL Memory Model

OpenCL memory model defines four regions of memory accessible to work-items when executing a kernel."Fig. 4" shows how memory regions are related with platform model.OpenCL Memory model is divided in following memories:

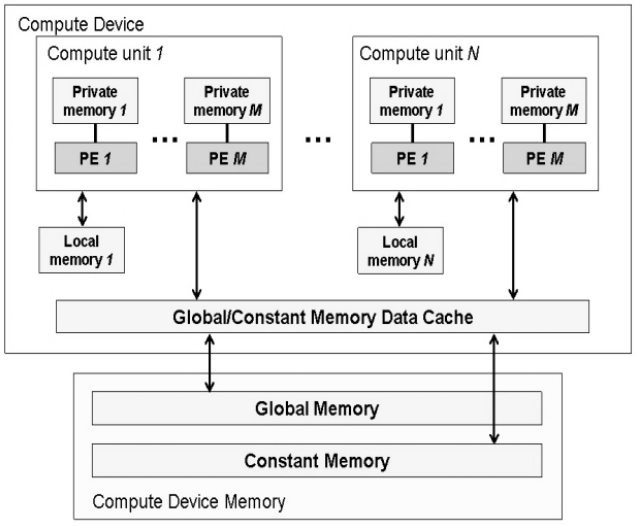

Fig.4. OpenCL Memory model

Global memory is a memory region in which all work-items and work-groups have read and write access on both the compute device and the host. This region of memory can be allocated only by the host during runtime.

Constant memory is a region of global memory that stays constant throughout the execution of the kernel. Work-items have only read access to this region.

Local memory is a region of memory used for data-sharing by work-items in a work-group. All work-items in the same work-group have both read and write access.

Private memory is a region that is accessible to only one work-item.

3.4 OpenCL Programming Model

The OpenCL execution model supports data parallel and task parallel programming models, as well as supporting hybrids of these two models. The primary model driving the design of OpenCL is data parallel [10].

4 OPENCL IMPLEMENTATION FOR SOLVING APSP PROBLEM USING IN-PLACE RECURSIVE APPROACH

4.1 APSP Problem

The APSP is a fundamental graph problem. Given a weighted directed graph $G = (V, E)$ with a weight function $w : E \rightarrow R$, the problem is to find shortest path length from a vertex $u \in V$ to vertex $v \in V$ for every pair of vertices (u, v), where the path length is the sum of the weights of its constituent edges.

For the adjacency-matrix representation of a graph G = (V, E), we assume that the vertices are numbered 1, 2, . . . , |V| in some arbitrary manner. Then the adjacency-matrix representation of a graph G consists of an $n \times n$ weight matrix $W = (w_{ij})$, where w_{ij} is the weight of directed edge (i, j). If $i = j$, then $w_{ij} = 0$ and if there is no edge between vertices i and j, then $w_{ij} = \infty$.

The standard algorithm for APSP problem is Floyd-Warshall (FW) algorithm [7]. The pseudocode for FW algorithm is given in "Fig. 5". The solution of APSP problem is given by matrix$D^{(n)} = (d_{ij}^{(n)})$, where $d_{ij}^{(k)}$be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set {1, 2, . . . , k}. Initially $D^{(0)} = W$.

ALGORITH FLOYD-WARSHALL(W)

```
1   n← rows(W)
2   D^(0)← W
3   for k from 1 to n do
4     for i from 1 to n do
5       for j from 1 to n do
6       d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
7       end for
8     end for
9   end for
```

Fig.5. FW Algorithm pseudo code

In FW algorithm, every element (i, j) of adjacency matrix ink^{th} iteration is updated by adding elements (i, k) & (k, j) and comparing with (i, j), where $1 \leq k, i, j \leq n$. Since k, i, j can take any value between 1 and n, so it makes impossible to partition data (adjacency matrix).

4.2 In-place Recursiveapproach

Our approach for solving APSP problem using OpenCL is based on R-Kleene algorithm [1]. It works by recursively partitioning adjacency matrix into sub-matrices. "Fig. 6" shows the pseudocode for R-Kleene algorithm. Here original matrix is divided into 4 equal sub-matrices A, B, C, D and this algorithm is recursively called for sub-matrix. Then several MMs are performed on sub-matrices. Matrix multiplication X += YZ (or X = X + YZ) is formally defined as $x_{i,j} += \Sigma_{k=0}^{n-1} y_{i,k} * z_{k,j}$, where scalar addition is minimum of two numbers, i.e. $a + b = \min(a, b)$ and scalar multiplication of two numbers is addition, i.e., $a * b = a + b$. These MMs are defined in a closed semi-ring.

Tropical semiring ($R^+ \cup \{\infty\}$, min, +, ∞, 0) is a closed semiring defined over the set of non-negative numbers R^+. For APSP problem, elements (weight values) in adjacency matrix can have any value in the set R^+ and min & + are two binary operations. MMs in our approach are defined on this closed semiring.

MM operation in R-Kleene algorithm resembles with the matrix-multiply operation in multiplication of two matrices, where product (*) & sum (+) are replaced by sum (+) & min operation respectively in MM routine of R-Kleene.

ALGORITH R-KLEENE(J)

```
        /*      | A B | */
        /* J = | C D | */

1    A ← R−Kleene(A)
2    B ← B + A*B
3    C ← C + C*A
4    D ← D + C*B
5    D ← R−Kleene(D)
6  B ← B + B*D
7    C ← C + D*C
8    A ← A + B*C
```

Fig.6. Pseudocode for R-Kleene sequential algorithm

4.3 OpenCL Parallel Implementation

"Fig. 7" shows OpenCL parallel algorithm for our approach based on R-Kleene. In this algorithm, an *n*-by-*n* adjacency matrix J of dense graph is recursively partitioned into equal-sized *n*/2-by-*n*/2 sub-matrices A, B, C and D. Recursive calls are made to sub-matrices A & D denoted by step 6, 16 respectively in Fig.7. OpenCL kernel for MM module OCL_KERNEL_RKMM(X, Y, Z, *n*) shown in "Fig. 10" is called on these sub-matrices. This kernel is executed by n^2 processing elements (PEs) or threads in parallel, where each thread is computing a value for an *n*-by-*n* matrix element. In our implementation, we have designed 2D kernels.

ALGORITH OPENCL-PARALLEL-RK(J, *n*)

```
/* J is n-by-n matrix */
1    if (base case)
2      call OPENCL-PARALLEL-FW(J, n)
3    else
4      divide matrix J in matrices A, B, C & D
5      n ← n/2
6      call OPENCL-PARALLEL-RK(A, n)
7      for all n^2 matrix elements in parallel do
8        call OCL_KERNEL_RKMM(B, A, B, n)
9      end for
10     for all n^2 matrix elements in parallel do
11       call OCL_KERNEL_RKMM(C, C, A, n)
12     end for
13     for all n^2 matrix elements in parallel do
14       call OCL_KERNEL_RKMM(D, C, B, n)
15     end for
16     call OPENCL-PARALLEL-RK(D, n)
17     for all n^2 matrix elements in parallel do
18       call OCL_KERNEL_RKMM(B, B, D, n)
19     end for
20     for all n^2 matrix elements in parallel do
21       call OCL_KERNEL_RKMM(C, D, C, n)
22     end for
20     for all n^2 matrix elements in parallel do
21       call OCL_KERNEL_RKMM(A, B, C, n)
22     end for
```

Fig.7. Pseudocode for OpenCL parallel implementation of in-place recursive approach

Recursion in this algorithm is stopped when matrix size (i.e. *n*) becomes equal to (or smaller than) a threshold value. This is the base case for this algorithm. We have considered base case when matrix can be fitted in a single workgroup (or block) of size 16x16 (or lesser i.e. like 8x8). When this base case condition is true, then OpenCL parallel FW iterative algorithm is called, which calls a kernel OCL_KERNEL_FW(A, *k*) shown in "Fig. 8" for all matrix elements in each k^{th} iteration. This parallel FW iterative algorithm can also be called for large matrices in similar fashion.

ALGORITH OPENCL-PARALLEL-FW(A, *n*)

```
1    for k from 1 to n do
2      for all elements in matrix A, where 1≤ i, j ≤ n in parallel do
3call OCL_KERNEL_FW(A, k)
4      end for
5    end for
```

Fig.8. Pseudocode for OpenCL parallel FW algorithm

KERNEL OCL_KERNEL_FW(A, *k*)

```
1    (i, j) ← getThreadID
2    A[i, j] ← min(A[i, j], A[i, k] + A[k, j])
```

Fig.9. Pseudocode for FW kernel in OpenCL

KERNEL OCL_KERNEL_RKMM(X, Y, Z, *n*)

```
1    (i, j) ← getThreadID
2    for k from 1 to n do
3      X[i, j] ← min(X[i, j], Y[i, k] + Z[k, j])
4    end for
```

Fig.10. Pseudocode for OpenCL MM kernel used in algorithm in "Fig. 8"

OpenCL does not support recursion, so we have written a recursive function in the host program that calls OpenCL kernels in each of its recursive calls. It also increases data reuse ratio because of its natural cache locality exploitation as it is recursive in nature.

Memory access to local memory is faster than global memory access in compute device (GPU or CPU). So by exploiting memory hierarchy we can obtain significant performance gain, but it requires a very important attention from programmer that maximum memory size for local or private memory should not exceed. We have utilized local memory in OpenCL kernel for MM module.

Our OpenCL parallel implementation for APSP is completely in-place in nature. It means that additional memory space is not required in GPU global memory during computation. While dividing matrix J into A, B, C & D, we perform logical partitioning. It means sub-matrices are not stored separately in global memory, only start and end positions of sub-matrices are stored.

5 EXPERIMENTAL RESULTS

We have tested OpenCL parallel implementation for our approach in various GPUs and Intel CPU. Details of devices on which tests are performed, are given as follows:

- **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.

- **NVIDIA GeForce GT 630M (GPU):** 2 Compute units, 950 MHz clock, 1023MB Global Mem., 48 KB Local Mem., 1024 work group size on a system having Intel Core i5 CPU-3210M @ 2.5GHz and 4096MB RAM with NVIDIA GPU computing SDK 4.2.
- **AMD Radeon HD 6850(GPU):** 12 Compute units, 860 MHz clock, 1024MB Global Mem., 32KB Local Mem., 256 work group size on a system having Intel Core i3 CPU 530 @ 2.93 GHz and 4096MB RAM with AMD APP SDK v 2.8.
- **Intel Core i3-2310M (CPU):** 4 Compute units, 2095 MHz clock, 2048MB Global Mem., 32KB Local Mem., 1024 work group size with AMD APP SDK v2.8.

Applications are written in C++ using Visual Studio 2010 with OpenCL SDK (already specified). We have also implemented serial code for FW iterative & R-K algorithm in C++ running on a single core using Visual Studio 2010 on a system having Intel Core i3-2310M@ 2095MHz clock CPU & 3072 MB RAM for comparison with parallel implementations. We have tested our results on various randomly generated dense graphs having edges of the order of $O(n^2)$. Random weight values between 1 to 10 is assigned to edges of graph.

In "Fig. 11", log-log plot of execution time in milliseconds and no. of nodes in a graph is presented. OpenCL parallel implementation for RK based approach is tested on various GPU devices and also on CPU device. Timings for RK & FW iterative sequential implementation are also shown. Speedup for RK based OpenCL implementation is shown in "Fig. 12"&"Fig. 13" w.r.t. RK sequential & FW iterative sequential implementation respectively. OpenCL parallel implementation shows a significant speedup up to 475x over RK & FW iterative serial CPU implementation running on a single core.

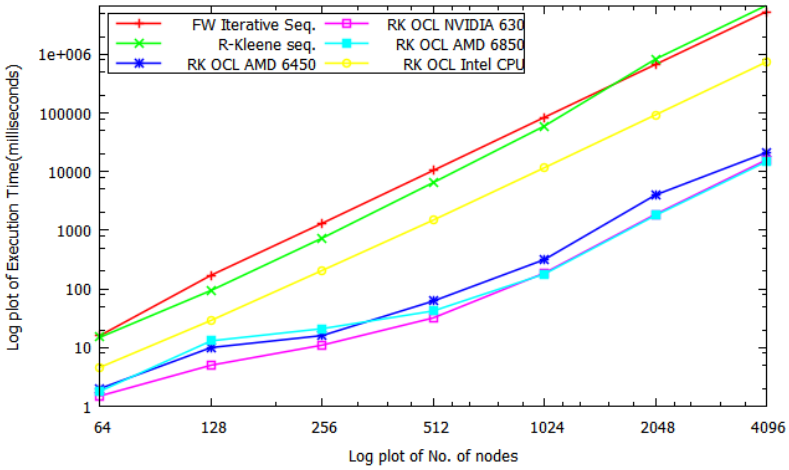

Fig.11. Timings for R-Kleene(RK) based OpenCL parallel implementation in various devices and for RK & FW iterative sequential implementation

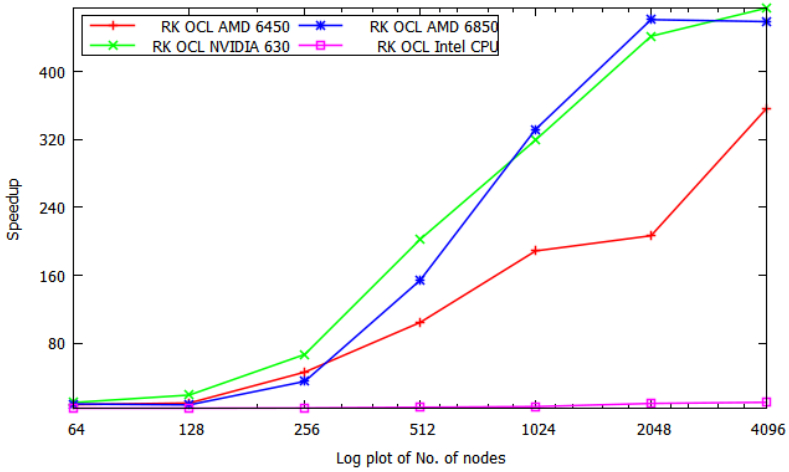

Fig.12. Speedup for RK based OpenCL parallel implementation w.r.t. RK sequential implementation

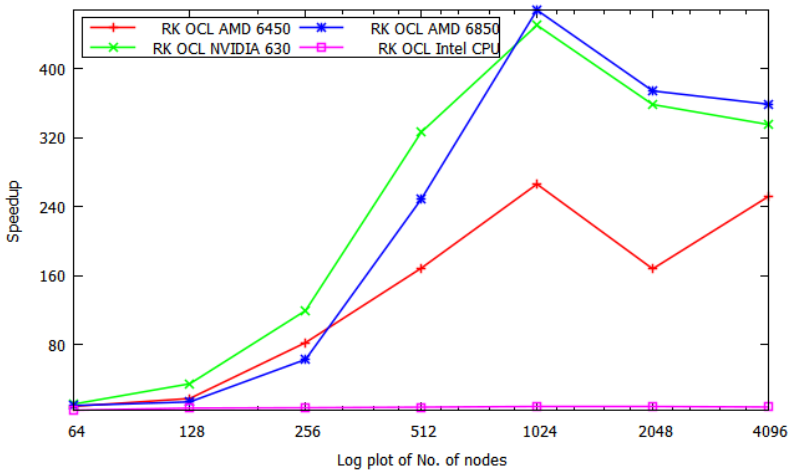

Fig.13. Speedup for RK based OpenCL implementation w.r.t. FW iterative Sequential implementation

In “Table 1”, speedup comparison for RK based OpenCL parallel implementation on various devices over RK serial CPU implementation is shown.

Table 1. Speedup comparison for various devices

No. of nodes	AMD 6450 GPU	NVIDIA GT 630M GPU	AMD 6850 GPU	Intel CPU
64	7.5	10	8.33	3.26
128	9.4	18.8	7.23	3.24
256	45.81	66.64	34.9	3.56
512	104.42	202.31	154.14	4.36
1024	188.64	319.47	331.96	5.13
2048	206.6	441.71	461.27	8.91
4096	356.56	474.91	458.96	10.19

In “Fig. 14”, log-log plot for FW OpenCL parallel implementation on various devices and for RK & FW iterative serial implementation is shown. In this timings in milliseconds and no. of nodes in graph are represented.

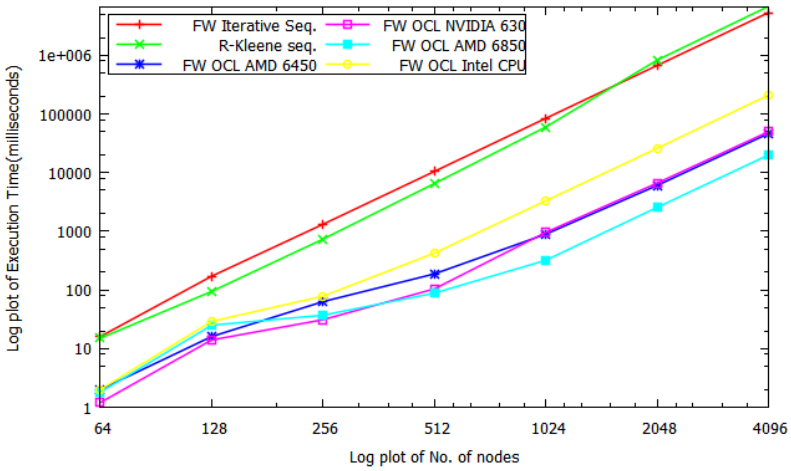

Fig.14. Timings for FW OpenCL parallel implementation on various devices and for RK & FW iterative sequential implementation

"Fig. 15" shows comparison between RK based OpenCL parallel and FW OpenCL parallel implementation on various GPUs. Our RK based OpenCL implementation takes lesser time in comparison to FW OpenCL implementation on same GPU device.

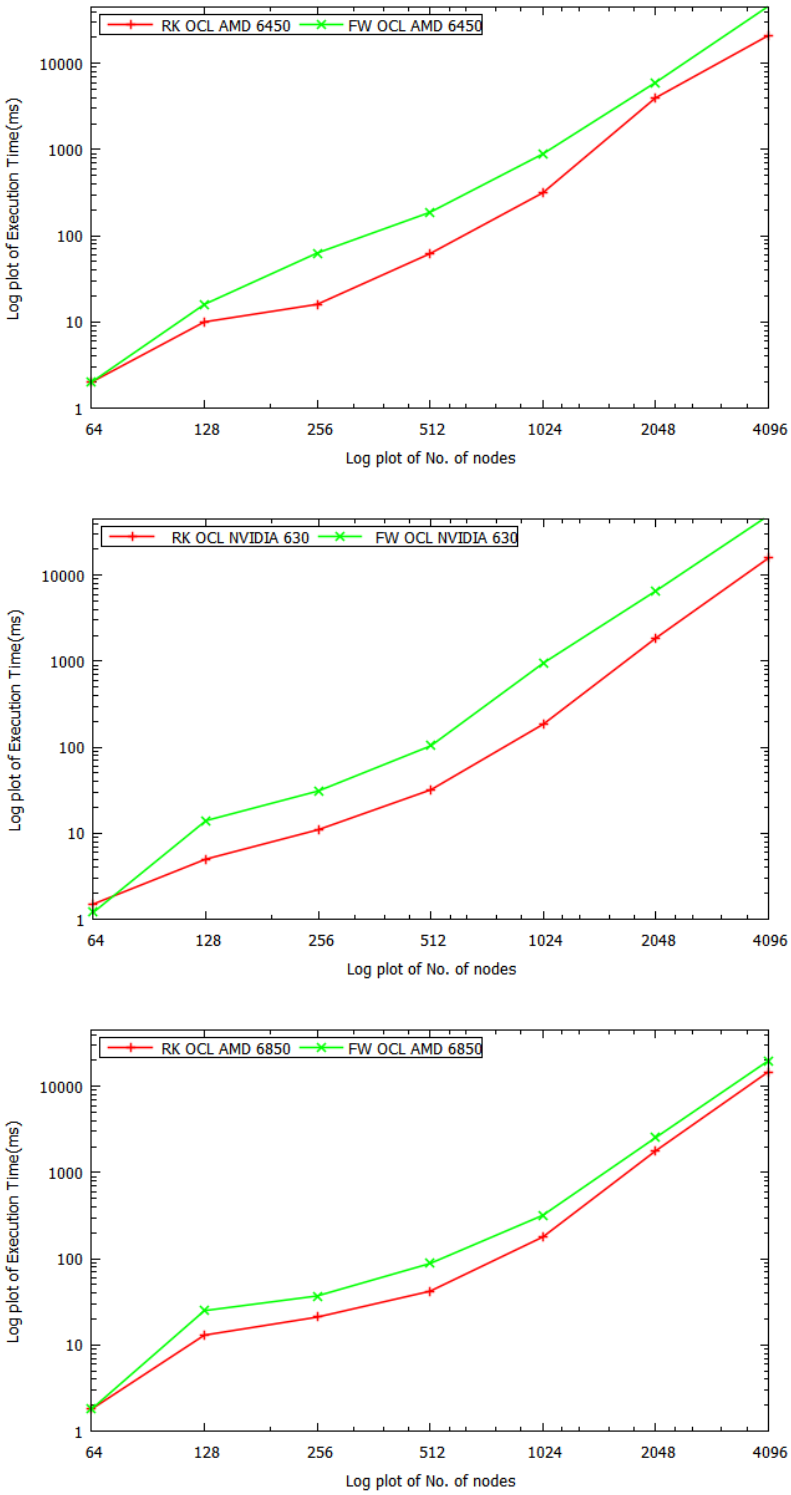

Fig.15. Comparison between RK based OpenCL parallel and FW OpenCL parallel implementation on same GPU device

6 CONCLUSION AND FUTURE WORK

In our paper, we have presented OpenCL parallel implementation for APSP problem based on R-Kleene algorithm. This algorithm is completely in-place and recursive in natures that makes it to better exploit the capabilities of GPU and it also utilizes data locality. We have utilized local memory in OpenCL implementation. Our implementation shows a significant speedup up to 475x over sequential FW and R-Kleene implementation running on a single core CPU.

Our future work includes designing Hybrid CPU-GPU implementation for APSP problem for very large graphs and also performing optimization like memory coalescing, vectorization for OpenCL implementation.

References

[1] Paolo D'Alberto, A. Nicolau, "R-Kleene: a high-performance divide-and-conquer algorithm for the all-pair shortest path for densely connected networks", Algorithmica 47 (2) (2007) pp. 203–213.

[2] Park, J., Penner, M., Prasanna, V.,"Optimizing graph algorithms for improved cache performance". In:Proc. of International Parallel and Distributed Processing Symposium, 2002.

[3] K. Fatahalian, J. Sugerman, P. Hanrahan, "Understanding the efficiency of GPU algorithms for matrix–matrix multiplication", in: HWWS '04: Proceedings of the ACM SIGGRAPH/ EUROGRAPHICS Conference, ACM, New York, 2004, pp. 133–137.

[4] S. Mu, X. Zhang, N. Zhang, J. Lu, Y. S. Deng, and S. Zhang. "IP routing processing with graphic processors". In DATE '10, March 2010.

[5] P. Harish, P. J. Narayanan, "Accelerating large graph algorithms on the GPU using CUDA", In: Proc. of 14th Int'l Conf. High Performance Computing (HiPC'07),Dec. 2007.

[6] David A. Badar, K. Madduri, "Designing multithreaded algorithms for breadth-first search and st-connectivity on the Cray MTA-2", In: ICPP, pages 523-530, 2006.

[7] Thomas H.Cormen, Charles E. Leiserson, Ronald L. Rivest and Clittord Stein, "An Introduction To Algorithms", McGraw-Hill Book Publication, First Edition,1990.

[8] AMD Inc., "AMD Acclerated Parallel Processing OpenCL Programming Guide", July 2012.

[9] A. Munshi, B. R. Gaster, T.G. Mattson, J. Fung, D. Ginsburg, "OpenCL Programming Guide",Addison-Wesley pub.,2011.

[10] OpenCL Specification, http://www.khronos.org/ registry/cl/specs/opencl-1.2.pdf.

[11] Opencl 1.2 reference pages, KHRONOS, 2012. http://www.khronos.org/registry/cl/sdk/1.2/docs/man/xhtml.

[12] K. Boydstun, "Introduction opencl" ,TAPIR, California Institute of Tech., http://www.tapir.caltech.edu/ ˜kboyds/OpenCL/opencl.pdf.

[13] AMD Inc., "Introduction to OpenCL programming training guide", May,2010