Fall 2017

Power-Efficient and Highly Scalable Parallel Graph Sampling using FPGAs

Usman Tariq
WMU, muhammadusman.tariq@wmich.edu

Umer Cheema
Intel

Fahad Saeed
Western Michigan University, fahad.saeed@wmich.edu

Power-Efficient and Highly Scalable Parallel Graph Sampling using FPGAs

Usman Tariq[1], Umer I. Cheema[2], and Fahad Saeed[*1]

[1]*Department of Electrical and Computer Engineering, Western Michigan University*
[2]*Technology & Manufacturing Group Intel Corporation*

Abstract

Energy efficiency is a crucial problem in *data centers* where big data is generally represented by directed or undirected graphs. Analysis of this big data graph is challenging due to volume and velocity of the data as well as irregular memory access patterns. Graph sampling is one of the most effective ways to reduce the size of graph while maintaining crucial characteristics. In this paper we present design and implementation of an FPGA based graph sampling method which is *both* time- and energy-efficient. This is in contrast to existing parallel approaches which include memory-distributed clusters, multicore and GPUs. Our strategy utilizes a novel graph data structure, that we call COPRA that allows time- and memory-efficient representation of graphs suitable for reconfigurable hardware such as FPGAs. Our experiments show that our proposed techniques are 2x faster and 3x more energy efficient as compared to serial CPU version of the algorithm. We further show that our proposed techniques give comparable speedups to GPU and multi-threaded CPU architecture while energy consumption is 10x less than GPU and 2x less than CPU.

1 Introduction

Extracting information through graph analysis is an essential process used in many real-world applications [18] [17]. Running simulations and analytics on massively huge graphs is impractical and cost-prohibitive [6]. Graph sampling allows reduction in the amount of data that needs to be processed while maintaining application specific properties e.g. degree distribution, betweenness centrality, degree exponent, rank exponent etc. Graph sampling literature is extensive and includes *Selection Based Sampling* [1], *Traversal Based Sampling* [9] [10] [19] and *Reduction Based Sampling* [12].

Sequential techniques are inadequate even for sampling of big data. Existing parallel sampling techniques are either application specific [3] [4] or they are too inaccurate [16] to be used in real applications. Allocating power hungry resources for sampling such as GPU or Intel Phi can result in significant cost for large data centers. In addition, conventional "One Size Fits All" devices like CPUs are no longer desired as they can lead to under utilization of resources which in turn can also waste a lot of energy [8]. Majority of large-scale efforts focuses on conventional devices since dedicated accelerators are difficult to program. In the past, FPGAs have had a draw back over other accelerators because of time consuming implementations of proposed architecture using HDLs like Verilog or VHDL [20].

In this paper, we present an FPGA based parallel version of the best performing reduction based algorithm presented in [12] geared towards power-law graphs. FPGA is our choice of accelerator since FPGAs are highly energy efficient, can provide speedups for correctly designed strategy when compared with CPUs, provide custom made solution and their reconfigurability allows them to be reprogrammed as per changing requirements. We chose a higher level approach by implementing our proposed strategy using Intel FPGA

---

[*]**Corresponding author e-mail**: fahad.saeed@wmich.edu

SDK for OpenCL. Using our novel graph data structure and careful parallel design we show that our proposed strategy is able to compete in terms of speed with both CPU and GPU based versions of the sampling algorithms while reducing the energy consumption by a factor of 3. We test our implementation on synthetic power-law graphs that closely resemble to real world graphs.

2   Background

Assume a graph $G(V, E)$ with $V(G)$ as its vertex set and $E(G)$ as its edge set. Sampling process will select a subset of vertices or edges from the original graph and construct a sub-graph induced by those vertices/edges. In reduction based sampling, we select a subset of vertices or edges to be removed from the original graph. Three parallel graph sampling algorithms (sequential version originally presented in [12]) that we implement are:

Delete Random Vertex (DRV): Randomly choose vertices and delete from the original graph.
Delete Random Edge (DRE): Randomly delete edges from the original graph.
Delete Random Vertex Edge (DRVE): Randomly select a vertex and then delete a random edge from set of edges incident to that vertex.

As with any graph sampling algorithm, small portion of vertices, edges or vertex-edges are removed and we keep the largest connected component and delete all other components. The whole process is repeated till we are left with graph of desired size.

Rest of this paper is organized as follows: In section 3, we describe our work in detail. We present our results and compare them with CPU and GPU versions in section 4. In section 5, we conclude this paper.

3   Proposed Methods

3.1   COPRA: Graph Data Structure for Parallel Processing

The sampling methodologies that we design and implement on FPGA require removing vertices and edges from the original graph in parallel. Therefore, we need a data structure which can deal with irregular memory access patterns of graphs and also allow multiple threads to access data points and concurrently make changes in parallel. Further, a data structure that can allow modification of the graph (as a result of sampling) without accessing external memory. Such a structure will be essential for efficient bandwidth utilization on an FPGA. Keeping these constraints in mind we introduce COPRA, a novel data structure, which is a modified version of Compressed Sparse Row (CSR) to represent graph in memory. Let us briefly discuss Compressed Sparse Row (CSR) before we get into the details of our modifications necessary for FPGA based strategy.

Consider the graph shown in Fig. 1 (left) and its Compressed Sparse Row (CSR) representation (right). In CSR, vertices and edges are stored in two separate arrays V and E. Vertex numbers are represented by the indices of V while the value stored at each index of V points to the starting location of edges of that vertex. In edge array E, for each edge, we store the vertex number at its other end. In its simplest form space complexity of CSR is $O(V + 2E)$.

Even though CSR representation is simple and memory efficient, vertex and edge removal is not possible in its original format. Using CSR will require multiple updates to the data at external memory which will result in inefficient on-chip memory and bandwidth usage. Such a scheme will be inefficient both in terms of time and energy. Other graph data structure such as adjacency matrix of adjacency list also pose similar challenges i.e. higher memory complexity $O(V^2)$ and sequential memory accesses respectively. In order to tackle these challenges, we introduce two major changes in CSR to allow graph modification in an efficient manner (vertex removal and edge removal). The design of COPRA is discussed below.
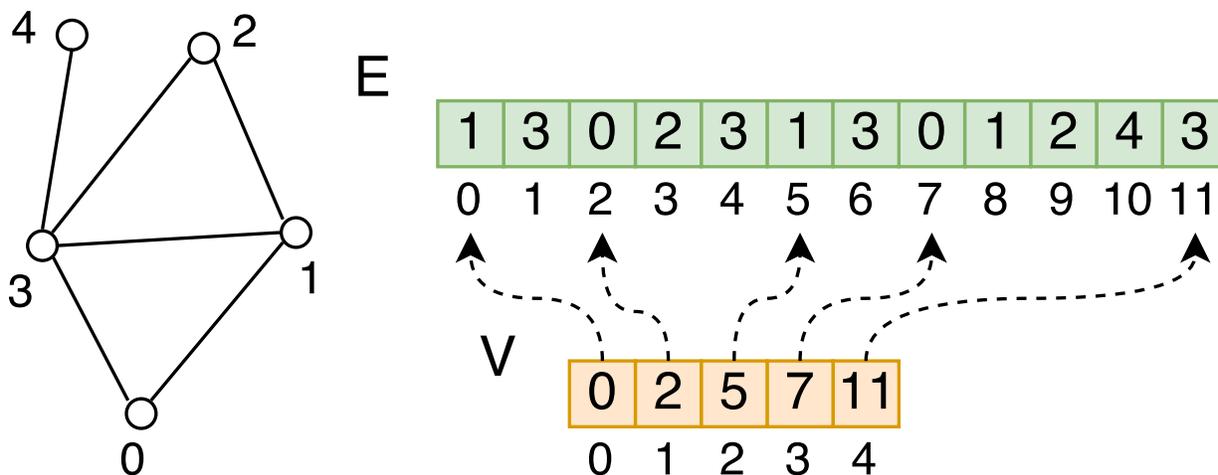
Figure 1: Graph G and its CSR Representation

3.1.1 Vertex Removal

When trying to delete a vertex say vertex "1", we'll lose some other information as to where edges of vertex "0" end in E. This information comes in handy when we are trying to traverse neighbors of a vertex especially if most of the vertices from the original graph have been removed. To solve this problem we introduce another vertex array and rename the two vertex arrays as *V-Start* and *V-End* as they keep track of starting and ending positions of their edges in E as shown in Fig. 2.

3.1.2 Edge Removal

The second problem is with edge removal or more specifically random edge removal. As each edge has two entries in E, one for each vertex it's incident on, in original CSR format (Fig. 1) we don't have any information available to find the other end of randomly selected edge. Suppose, we were to randomly delete the edge that connects vertex "1" and "3" (In reality we'll pick either index 4 or 8 and search for the other value). Say, the randomly selected index was 4. Looking at the value we instantly know that one side is connected to vertex "3" but to find out the vertex on the other side of this edge we'll have to traverse the vertex array and look for the reference to array E in V that's nearest to index 4. This will be very time consuming and become impractical in large graphs. To overcome this problem, we add additional values in edge array, i.e. at the beginning of each edge set (edges incident to one vertex) we add the vertex number (blue values). To indicate this value is not an edge we add another flag (red values) before the vertex entry which will help us get correct vertex. Completed version is shown in Fig. 3.

3.2 Parallel Removal of Vertices, Edges and Vertex-Edges

First step of sampling process is to remove small portion of Vertices, Edges or Vertex-Edges, depending on the sampling strategy, from the original graph. Our method is highly scalable as each thread removes one vertex (edge or vertex-edge) in parallel. Even if multiple threads end up accessing the same memory location the end result of sampling still remains the same. Here we discuss three algorithms that remove vertices, edges and vertex-edges from graphs respectively.

3.2.1 Parallel Vertex Removal

Kernel pseudo-code for removing vertices is given in Algorithm 1. In the outer-loop of our algorithm we traverse the edges of the vertex v being removed. After removing one side of an edge $e = (v, u)$ we traverse the edges of the neighboring vertex u in the inner loop to remove the other side.
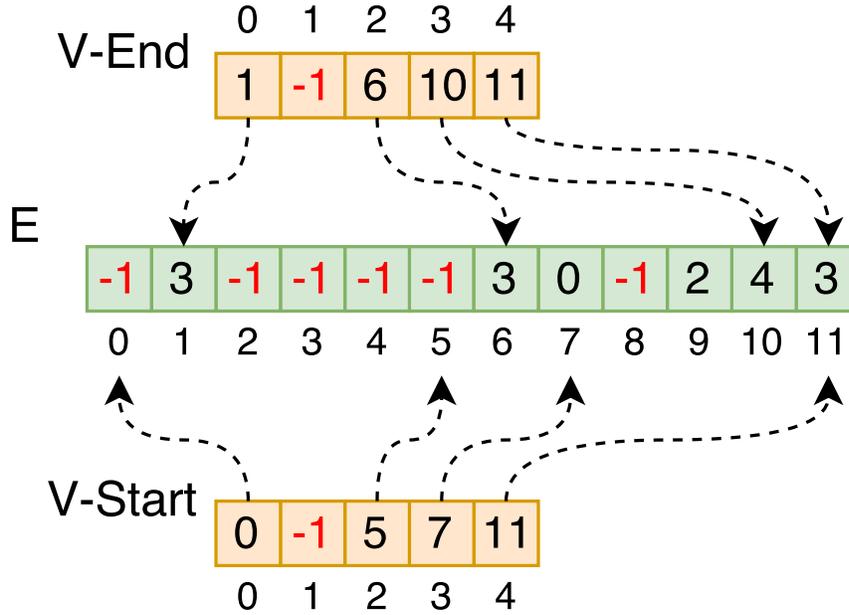
Figure 2: Introducing first modification in CSR i.e. adding another vertex array *V-End* to keep track of ending position of edges of each vertex in array *E*. In this way, we can remove any vertex from the original graph without worrying about lose of information i.e. we still know the degree of vertex "0" even after we have removed vertex "1" as opposed to original CSR structure.

Algorithm 1 Remove Random Vertices

Input: $vStart, vEnd, edges, rands$

1: $tid \leftarrow$ thread-id
2: $v \leftarrow rands[tid]$
3: $sPos \leftarrow vStart[v], vStart[v] \leftarrow -1$
4: $ePos \leftarrow vEnd[v], vEnd[v] \leftarrow -1$
5: **for** $i = sPos$ to $ePos$ **do**
6:     $nv \leftarrow edges[i]edges[i] \leftarrow -1$
7:     $sPos1 \leftarrow vStart[nv]$
8:     $ePos1 \leftarrow vEnd[nv]$
9:     **for** $j = sPos1$ to $ePos1$ **do**
10:       **if** $edges[j] == v$ **then**
11:         $edges[j] = -1$
12:       **end if**
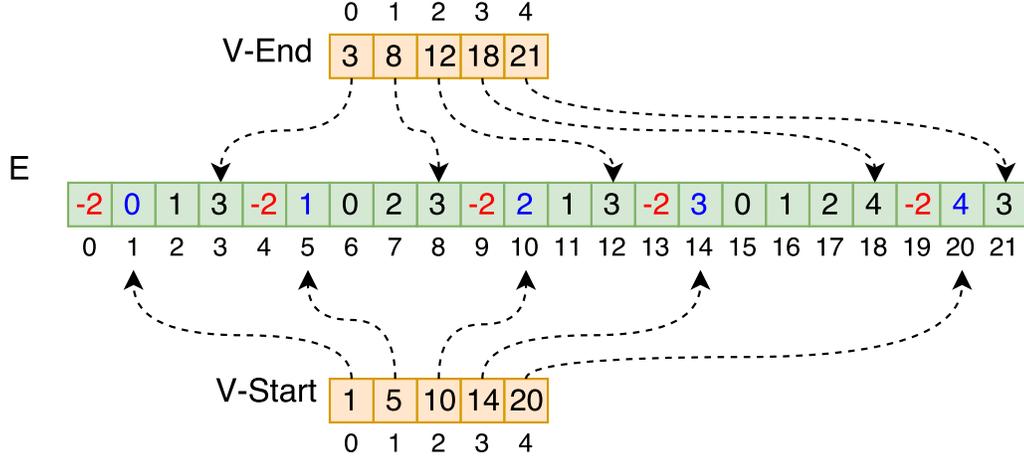13:   **end for**
14: **end for**

Figure 3: Modified CSR Representation: Final version of our proposed data structure COPRA. Edge array "E" is indexed by two vertex arrays "V-Start" and "V-End" and we also have corresponding vertex number before each edge set. Space complexity of this structure is $O(4V + 2E)$

Algorithm 2 Remove Random Edges

Input: $vStart, vEnd, edges, rands$

1: $tid \leftarrow$ thread-id
2: $e \leftarrow rands[tid]$
3: $v \leftarrow edges[e], edges[e] \leftarrow -1$
4: $i \leftarrow e - 1$
5: **while** $edges[i] \neq -2$ **do**
6: $\quad i \leftarrow i - 1$
7: **end while**
8: $sPos \leftarrow vStart[v]$
9: $ePos \leftarrow vEnd[v]$
10: **for** $j = sPos$ to $ePos$ **do**
11: $\quad$ **if** $edges[j] == edges[i + 1]$ **then**
12: $\quad\quad edges[j] = -1$
13: $\quad$ **end if**
14: **end for**

In OpenCL for FPGAs, nested loops can degrade performance and should be avoided if possible. In our kernel, we used loop unrolling to reduce number of iterations of inner loop. We get the maximum performance gain by unrolling the inner loop 16 times. This loop unrolling doesn't add any overhead in terms of memory accesses since the burst size of global memory used (DDR3) is 64 bytes.

3.2.2 Parallel Edge Removal

Parallel Edge Removal consists of two parts: 1) Remove one side of the edge and find the vertex on the other side, 2) Remove the other side of the vertex. First part is done by traversing the edge array backwards till we find -2. After getting the vertex connected to the other side we traverse its edges and delete the reference to first vertex. This is explained in Algorithm 2 Both loops are unrolled 16 times to get the maximum performance.

Algorithm 3 Remove Random Vertex-Edges

Input: $vStart, vEnd, edges, randV, randE$
1: $tid \leftarrow$ thread-id
2: $v \leftarrow randV[tid]$
3: $sPos \leftarrow vStart[v]$
4: $ePos \leftarrow vEnd[v]$
5: $d \leftarrow ePos - sPos$ {vertex degree}
6: $e \leftarrow randE[tid] \mod d$
7: $vn \leftarrow edges[sPos + e]$ {neighbor of v}
8: $sPos1 \leftarrow vStart[vn]$
9: $ePos1 \leftarrow vEnd[vn]$
10: **for** $j = sPos1$ to $ePos1$ **do**
11:    **if** $edges[j] == v$ **then**
12:       $edges[j] = -1$
13:    **end if**
14: **end for**

3.2.3   Parallel Vertex-Edge Removal

This technique includes picking random vertex and then deleting one of the random edges from that vertex. To achieve this we need two arrays containing random values, we call these "randV" and "randE". The size of "randE" is determined based on the maximum degree D of the graph. Once, we pick a vertex v at random, we pick a value from "randE" depending on which thread we're in. After this, deleting the other end of this edge is straight forward since we already know which vertex v this edge e belongs to. Pseudo code is given in Algorithm 3. Loop is unrolled 16 times for maximum performance gain.

There is no guarantee that no two threads will access the same memory location (rather it's very likely that two or more vertices will end up accessing the same address). We ensure that data integrity using our strategy is maintained by a simple observation that the value being written to any location is -1 (to indicate removal). Whenever a thread reads a value from memory, it can check that the value is not -1. If it is, we know that it is invalid i.e. the corresponding vertex, edge or vertex-edge has already been removed and we won't perform any actions on that data.

3.3   Sampling Algorithm

After having a memory efficient data structure to represent and update graph structures, we discuss design and implementation details of our sampling strategy on FPGA. Complete algorithm is described in algorithm 4:

Algorithm 4 Parallel Graph Sampling

1: **while** Graph Size > Required Sample Size **do**
2:    Launch Kernel to Remove Vertices, Edges or Vertex-Edges.
3:    $compSize \leftarrow 0$
4:    **while** $compSize <$ Required Sample Size **do**
5:       **while** $continue == true$ {Run BFS} **do**
6:          $continue \leftarrow false$
7:          Launch BFS kernel described in Algorithm 5.
8:       **end while**
9:    **end while**
10:    Launch Kernel to remove all vertices that are not in Largest Connected Component.
11: **end while**

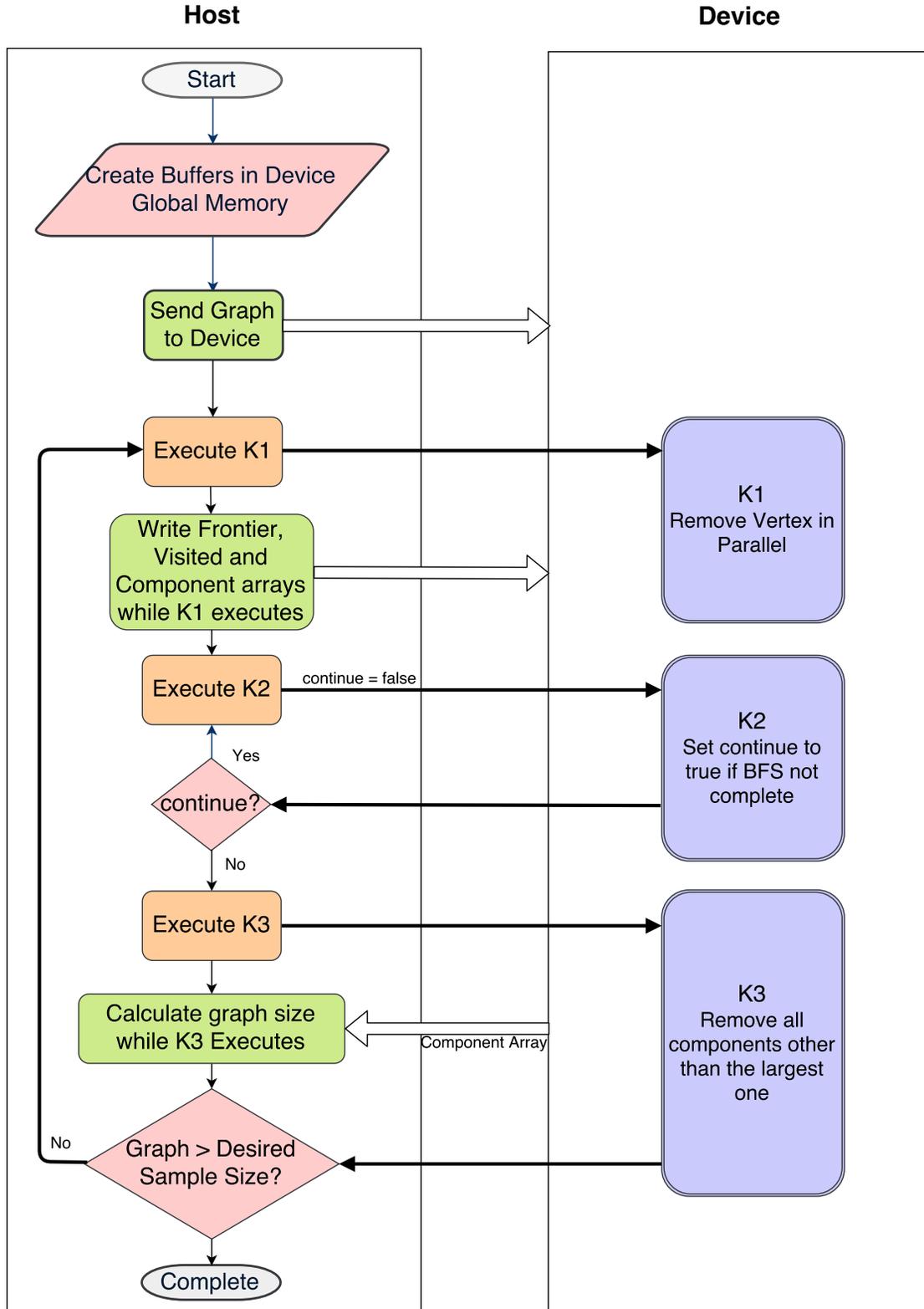The flow chart of the entire algorithm is given in Fig 4.
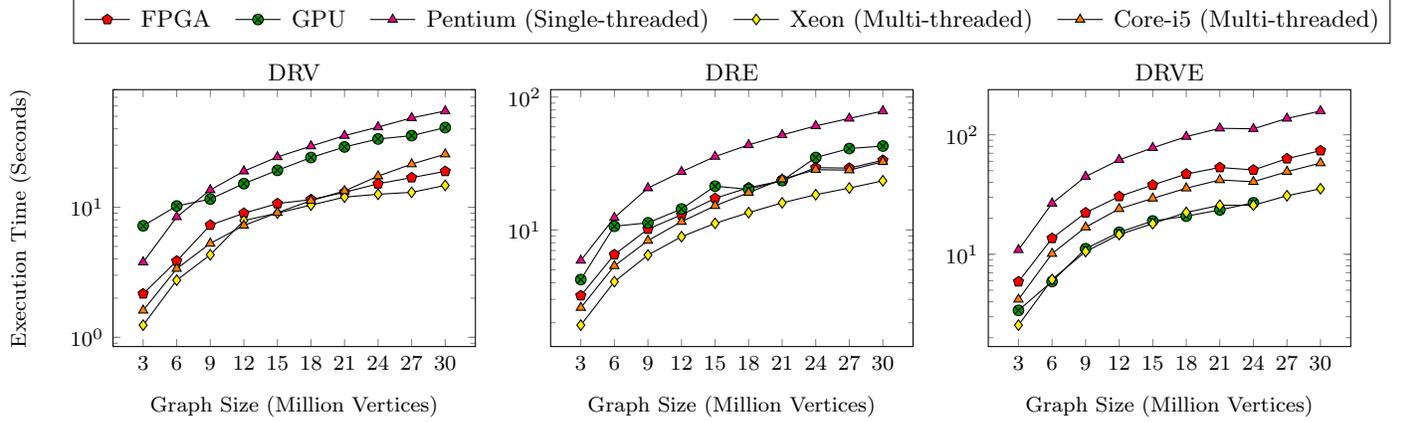

Figure 4: Parallel Graph Sampling

Figure 5: Execution time for different sampling techniques: DRV, DRE, DRVE with increasing graph size, executed on FPGA, GPU and CPU.
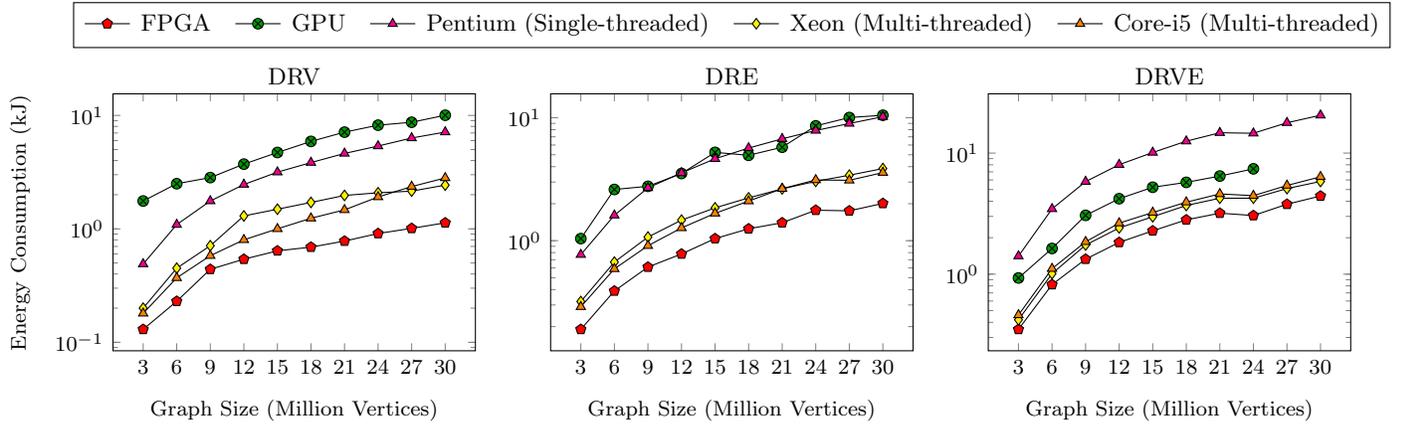

Figure 6: Energy consumption for different sampling techniques: DRV, DRE, DRVE with increasing graph size, executed on FPGA, GPU and CPU.

4 Results

4.1 Data Generation and Experimental Setup

The techniques discussed in this paper are targeted towards sampling of power-law graphs i.e. the graphs where number of vertices of degree x is proportional to $x^{-\alpha}$, where $\alpha > 0$ is a constant. Since vast majority of real world graphs follows power-law degree distribution; it makes our proposed technique more widely applicable. Different studies have shown that degree sequence of World Wide Web [14] [13] [11] and Internet router graph [7] follow power-law distributions among other applications [5] [2] [15]. For evaluation of our proposed strategy we generated synthetic power-law graphs using parameters to match the real world graphs that follow power-law i.e. graphs of up to 30M vertices with power-law degree distribution of exponent of 2.71 and average degree of 5.

Table 1: Specifications of FPGA, GPU and CPU used in our experiments.

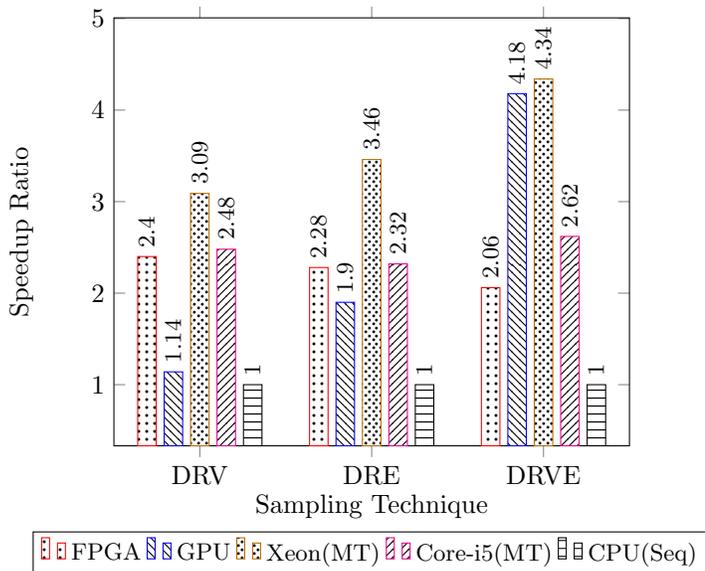|	DE5-Net Stratix V	GeForce GTX 480	Xeon W3565	Core-i5
Base Frequency	-	700 MHz	3.20 GHz	3.3 GHz
# Logical Cores	1	480	8	4
Memory	4 GB	1536 MB	6 GB	6 GB

8

Figure 7: Speedup ratio for different sampling techniques when executed on FPGA, GPU and CPU (Multi-Threaded) w.r.t CPU sequential version.

Figure 8: Energy Efficiency in term of "Vertices Processed per Second per Watt" for different sampling techniques when executed on FPGA, GPU and CPU.

We implemented the sampling algorithms on Pentium(R) and parallel version is implemented using OpenCL 1.2 on Intel Xeon W3565, Core-i5 2600, DE5-Net Stratix V GX FPGA Development Kit and GeForce GTX 480 GPU. Specifications of these devices are shown in Table 1.

4.2 Timing Analysis

We implemented sampling strategies on CPU, FPGA and GPU where sequential version of the algorithm runs on Pentium(R) while FPGA, GPU, Xeon and Core-i5 run the parallel version. The results in Fig. 5 show that our FPGA based technique is faster by more than 2x as compared to algorithms running on the CPU for all three sampling strategies. For FPGA vs GPU timing is comparable for DRV and DRE but GPU outperforms FPGA for DRVE. The relative speed up of FPGA and GPU with respect to CPU is shown in Fig. 7.

4.3 Energy Efficiency Analysis

To measure power efficiency we need to measure power consumed during execution of algorithms on each device. Power consumption for each device was measured experimentally using a *watt meter*. For sequential version we use Pentium(R) to minimize power consumed by idle cores. To measure power, we disconnect both GPU and FPGA from the system and run the algorithm with minimum (constant) number of background applications running. Average power consumed by CPU for our three implementations is around 70 Watts. The average power consumed by the system for the execution of FPGA code is around 60 Watts while average power consumed by the system during execution of GPU code is around 245 Watts. Note that we only use Pentium CPU as host for FPGA and GPU versions as only a small fraction of host computational resources are required to schedule kernels. For Xeon and Core-i5 we can directly measure power as no extra devices are connected to them. Fig 8 shows the power efficiency comparison of different sampling strategies.
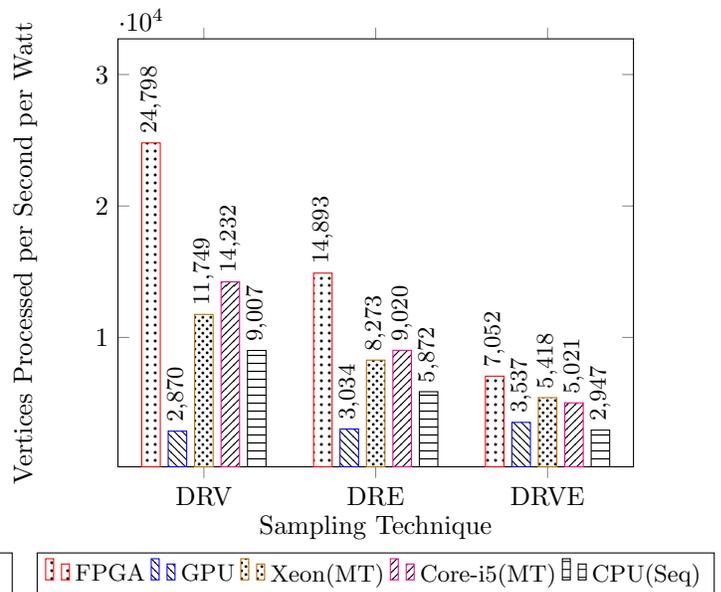
4.4 Quality Assessment

Our proposed FPGA based strategy is only accurate if it produces results similar to the sequential algorithms described in [12]. We run sequential and parallel version of three sampling algorithms on the largest graph (30M vertices) and measure characteristics like *Average Degree*, *Degree Exponent* and *Rank Exponent*. Rank Exponent and Degree Exponent are defined as follows [12]:

Rank Exponent is defined as the slope of log-log plot of node degrees vs their rank, where a node of kth highest degree will have a rank of k.

Degree Exponent is defined the slope of log-log plot of degree frequency vs degree.

Our experiments show that, for sampling percentages ranging from 10% to 70%, our proposed FGPA based strategy exhibits similar results with maximum difference of only 0.3% which is due to randomness involved in removal process.

5 Conclusion

The goal of the paper is to develop a graph sampling technique using FPGAs which can sample large graphs in parallel, is highly scalable and energy-efficient. We develop parallel FPGA-based sampling strategies that include DRV, DRE and DRVE. Our proposed FPGA based design not only exploits fine grained parallelism for sampling process but is also highly energy efficient when compared with CPU and GPU versions. Our extensive experiments indicate that our proposed technique compared with sequential (CPU) version provides 2x speedup and is 3x more energy efficient. When compared with GPU version, our FPGAs based strategy provides comparable execution run-times while having 10 times better energy efficiency. We also show that our FPGAs based strategy is more energy efficient when compared with parallel versions of CPUs. We also measure graph characteristics of sampled graph and compare them with the original graph and those produced by sequential sampling algorithms. These characteristics are within the bounds of originally proposed algorithms [12] and closely match with the sequential version.

Acknowledgement

References

[1] Nesreen Ahmed, Jennifer Neville, and Ramana Rao Kompella. Network sampling via edge-based node selection with graph induction. 2011.

[2] Xuelian Cai. The improved weighted evolution model of the as-level internet topology. In *Information Technology and Intelligent Transportation Systems: Volume 1, Proceedings of the 2015 International Conference on Information Technology and Intelligent Transportation Systems ITITS 2015, held December 12-13, 2015, Xian China*, pages 499–508. Springer, 2017.

[3] Kathryn Dempsey, Kanimathi Duraisamy, Hesham Ali, and Sanjukta Bhowmick. A parallel graph sampling algorithm for analyzing gene correlation networks. *Procedia Computer Science*, 4:136–145, 2011.

[4] Kathryn Dempsey, Kanimathi Duraisamy, Sanjukta Bhowmick, and Hesham Ali. The development of parallel adaptive sampling algorithms for analyzing biological networks. In *Parallel and Distributed*

Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, pages 725–734. IEEE, 2012.

[5] Xiaotie Deng, Zhe Feng, and Christos H Papadimitriou. Power-law distributions in a two-sided market and net neutrality. In *International Conference on Web and Internet Economics*, pages 59–72. Springer, 2016.

[6] Xenofontas A Dimitropoulos and George F Riley. Creating realistic bgp models. In *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on*, pages 64–70. IEEE, 2003.

[7] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, volume 29, pages 251–262. ACM, 1999.

[8] Jeremy Fowers, Greg Brown, John Wernsing, and Greg Stitt. A performance and energy comparison of convolution on gpus, fpgas, and multicore processors. *ACM Transactions on Architecture and Code Optimization (TACO)*, 9(4):25, 2013.

[9] Leo A Goodman. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

[10] Douglas D Heckathorn. Respondent-driven sampling: a new approach to the study of hidden populations. *Social problems*, 44(2):174–199, 1997.

[11] Jon M Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S Tomkins. The web as a graph: measurements, models, and methods. In *International Computing and Combinatorics Conference*, pages 1–17. Springer, 1999.

[12] Vaishnavi Krishnamurthy, Michalis Faloutsos, Marek Chrobak, Jun-Hong Cui, Li Lao, and Allon G Percus. Sampling large internet topologies for simulation purposes. *Computer Networks*, 51(15):4284–4302, 2007.

[13] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Extracting large-scale knowledge bases from the web. In *VLDB*, volume 99, pages 639–650, 1999.

[14] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew Tomkins. Trawling the web for emerging cyber-communities. *Computer networks*, 31(11):1481–1493, 1999.

[15] M Olmedilla, M Rocío Martínez-Torres, and SL Toral. Examining the power-law distribution among ewom communities: a characterisation approach of the long tail. *Technology Analysis & Strategic Management*, 28(5):601–613, 2016.

[16] Amir Hassan Rasti, Mojtaba Torkjazi, Reza Rejaie, Nick Duffield, Walter Willinger, and Daniel Stutzbach. Respondent-driven sampling for characterizing unstructured overlays. In *INFOCOM 2009, IEEE*, pages 2701–2705. IEEE, 2009.

[17] Fahad Saeed, Jason D Hoffert, and Mark A Knepper. Cams-rs: clustering algorithm for large-scale mass spectrometry data using restricted search space and intelligent random sampling. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 11(1):128–141, 2014.

[18] Fahad Saeed, Trairak Pisitkun, Mark A Knepper, and Jason D Hoffert. An efficient algorithm for clustering of large-scale mass spectrometry data. In *Bioinformatics and biomedicine (BIBM), 2012 IEEE International Conference on*, pages 1–4. IEEE, 2012.

[19] Matthew J Salganik and Douglas D Heckathorn. Sampling and estimation in hidden populations using respondent-driven sampling. *Sociological methodology*, 34(1):193–240, 2004.

[20] Deshanand Singh. Implementing fpga design with the opencl standard. *Altera whitepaper*, 2011.