ORNL/TM-2020/1716

Scale-Free Graph Networks with Trillions of Edges: Rapid Generation using 1000 GPUs

Maksudul Alam
Kalyan Perumalla

November 2020

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2020/1716

Computer Science and Mathematics Division

Scale-Free Graph Networks with Trillions of Edges: Rapid Generation using 1000 GPUs

Maksudul Alam
Kalyan Perumalla

November 2020

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-Battelle, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

CONTENTS

LIST OF FIGURES

ACRONYMS

GPU	Graphical Processing Unit
CUDA	Common Unified Data Architecture
SBA	Sequential Barabási–Albert
SCM	Sequential Copy Model
PPA	Parallel Preferential Attachment
cuPPA	CUDA Parallel Preferential Attachment

ABSTRACT

Synthetic networks are very useful in investigations of complex systems across the scientific spectrum, such as cyber-infrastructures, social networks, internet, and epidemiological contact networks. Scale-free networks are among the most common types of synthetic networks used for scientific investigation as formal approximations to complex real-life networks, which are technically characterized as exhibiting a power-law degree distribution. As real-life complex systems grow larger, generating networks that sufficiently mimic reality becomes computationally expensive. To address this issue, we present the latest results from efficiently scaled execution of our novel parallel algorithm for generating random scale-free networks using the preferential-attachment model. The algorithm, named **cuPPA**, is custom-designed to exploit multiple graphical processing units (GPUs) in distributed computing nodes. Our generator is useful in the development, debugging, and testing phases of high-performance computing (HPC) applications that use very large network data. During those phases, the overhead of moving actual network structure data can be avoided by generating the surrogate network structure data on the fly. The speed of the algorithm is sufficiently high to significantly reduce the computational overhead during the development phases of the network application. Furthermore, our generator provides determinism for the reproducibility of the HPC execution during development. Our algorithm generates extremely large scale-free networks of 4 trillion edges in less than 8 minutes using 1008 NVIDIA Volta GPUs of the Summit supercomputer. To our knowledge, this result represents the first achievement ever of graph network generation at this scale of parallel execution with over thousand GPUs. Moreover, it is equally applicable to offline processing as well as online stream processing – our algorithm is uniquely suitable for generating networks in a *streaming mode* without the need for explicitly storing (writing to disk) the entire network, and is suitable for targeting even larger scales with quadrillions of edges.

1. INTRODUCTION

Graph networks are prevalent in many complex systems, such as circuits, chemical compounds, protein structures, biological networks, social networks, the Web, and XML documents. Recently, there has been substantial interest in the study of a variety of synthetic (randomized) networks to serve as mathematical models of complex systems. Various network theories, metrics, topologies, and mathematical models have been proposed to understand the underlying properties and relationships of these systems.

Barabási and Albert [6] discovered a class of inhomogeneous networks, called scale-free networks, characterized by a power-law degree distribution $P(k) \propto k^{-\gamma}$, where k represents the degree of a vertex and γ is a constant. This class of network has multiple practical applications. For instance, the Barabasi-Albert model is useful in evaluating many real-world networks such as the North American electric grid with high reliability [7].

As these complex systems of today grow larger, the ability to generate progressively large random networks becomes all the more important. It is well known that the structure of larger networks is fundamentally different from that of small networks, and many patterns, such as communities, emerge only in massive networks [11].

Complementing the trend of increasing scales of network graphs, we also witness the trend of dramatically faster computing platforms, particularly based on accelerators called Graphical Processing Units (GPUs). GPUs are a cost-effective, energy-efficient, and widely available parallel processing platform. They are

highly parallel, multi-threaded, many-core processors that have greatly expanded beyond graphics operations and are now widely used for general purpose computing.

A novel set of GPU-based algorithms called `cuPPA` for generating scale-free networks was recently proposed by us [1, 2, 4]. In some of our previously largest experiments, `cuPPA` generated graphs of sizes ranging up to 16 billion edges. In this report, we significantly increase this capability by reporting additionally scaled algorithmic implementation and experiments taking the size from billions to trillions of edges. We present the challenges and run times in generating graphs with trillions of edges.

An additional challenge arises when we consider whether the generated graph can include more than one edge between a pair of nodes. Those graphs that contain no more than one edge between a pair of nodes are called simple graphs. Those graphs that can have more than one edge between any pair of nodes are called multi-graphs. The graphs generated by the previous algorithms were restricted to simple graphs. However, the effort involved in assuring that a generated synthetic graph is in fact a simple graph can become large as the graph size is increased. On the other hand, the probability of duplicate edges (that is, having more than one edge between a pair of nodes) exponentially decreases [12] in scale-free networks with large number (as in trillions) of edges. Checking for presence of duplicate edges (and eliminating or avoiding) to produce simple graphs can take a significantly longer amount of time. As the number of duplicate edges is relatively small in a scale-free network , one can instead use a graph that contains duplicate edges often called a multi graph. For these reasons, here we also implement and evaluate an algorithm that generates multi graph.

The rest of the paper is organized as follows. Section 2. provides background material in terms of preliminary information, notations, an outline of the network generation problem, and the base algorithms. Section 3. covers the experimental study and performance results using `cuPPA`. Finally, Section 4. concludes with a summary and an outline of future directions.

2. BACKGROUND

2.1 Preliminaries and Notations

In the rest of this report, we use the following notations. We denote a network $G(V, E)$, where V and E are the sets of vertices and edges, respectively, with $m = |E|$ edges and $n = |V|$ vertices labeled as $0, 1, 2, \ldots, n-1$. For any $(u, v) \in E$, we say u and v are *neighbors* of each other. The set of all neighbors of $v \in V$ is denoted by $N(v)$, that is, $N(v) = \{u \in V | (u, v) \in E\}$. The degree of v is $d_v = |N(v)|$. If u and v are neighbors, sometimes we say that u is *connected* to v and vice versa.

We develop parallel algorithms using the CUDA (Compute Unified Device Architecture) framework on the GPU. A GPU contains multiple streaming multiprocessors (SMs). An SM is a group of core processors. Each core processor executes only one thread at a time. All core processors can execute their corresponding threads simultaneously. If some threads perform operations that have to wait for data fetches with high latencies, those are put into the waiting state and other pending threads are executed. Based on this mode of operation, GPUs increase throughput by keeping the processors busy. All thread management, including the creation and scheduling of threads, is performed entirely in hardware with negligible time overhead for launching work on the GPU. For these advantages, modern supercomputers, such as the Oak Ridge Leadership Computing Facility's Titan and Summit machines listed in the series of top supercomputers of the world, are built using GPUs augmenting conventional central processing units (CPUs).

We use abbreviations of units as **K**, **M**, **B**, and **T** to denote units of thousand, million, billion, and trillion, respectively. For example, 2 **B** stands for two billion.

2.2 cuPPA: A Preferential Attachment–Based Generator

The preferential attachment model is a model for generating randomly evolved scale-free networks using a preferential attachment mechanism. In this mechanism, a new vertex is added to the network and connected to certain existing vertices that are chosen preferentially based on specific properties of the vertices. In the most common method, preference is given to vertices with larger degrees: the higher the degree of a vertex, the higher is the probability of choosing it. cuPPA uses this degree-based preferential attachment mechanism. In the rest of the paper, by preferential attachment (PA) we mean degree-based preferential attachment.

cuPPA uses an algorithm called the *copy model* [9, 10] to generate preferential attachment–based graphs. The copy model works as follows. It starts with a small clique of $\hat{d}$ vertices and, in every time step, a new vertex t is added to the network to create $d \leq \hat{d}$ connections to existing vertices $F_\ell(t)$ for $1 \leq \ell \leq d$ with $F_\ell(t) < t$. For each connection $(t, F_\ell(t))$ from vertex t, the following steps are executed:

Step 1: First, a random vertex $k \in [0, t-1]$ is chosen with uniform probability.

Step 2: Then, $F_\ell(t)$ is determined as follows:

$$F_\ell(t) = k \text{ with prob. } p \qquad \text{(Direct edge)} \tag{1}$$

$$= F_l(k) \text{ with prob. } (1-p) \qquad \text{(Copy edge)} \tag{2}$$

where l is a random outgoing connection from vertex k.

We also denote $\mathbb{F}(t) = \{F_1(t), F_2(t), \ldots, F_d(t)\}$ to be the set of outgoing vertices from vertex t.

cuPPA exploits a distributed memory-based algorithm [3, 5] to generate graphs on the GPU. Here, the set of vertices V is partitioned into disjoint subsets of vertices and processed by concurrent GPU threads. The algorithm works in two phases.

1. In the first phase of the algorithm (called *Execute Copy Model*), the copy model for all vertices is executed in parallel (using all threads). This phase creates all the direct edges and some of the "copy" edges. However, many copy edges might not be fully processed due to the dependencies. The incomplete copy edges are put in a waiting queue.
2. In the second phase of the algorithm (called *Resolve Incomplete Edges*), the incomplete edges from the waiting queue are resolved and the copy edges are finalized.

2.3 Graph Representation

We use one array of nd elements to represent and store the entire graph. Each vertex u connects to d existing vertices. The neighbors of u are stored between the indices inclusive from ud to $(u+1)d - 1$ that represent the other end-point vertices. We call these indices the *outgoing vertex list* for vertex u. The initial network consists of the d^2 vertices from the start of the array, representing a clique of d nodes. For any edge u, v where $u > v$ and $u, v > d$, the edge is represented by storing v in one of the d items in the *outgoing*

vertex list of u. Note that the graph's edge set E contains exactly nd edges as defined by the Barabási–Albert or the copy model. Any vertex with an index $0 \leq i < nd$ of the array G denotes the (t mod d)-th end-point of the vertex $\frac{i}{d}$.

2.4 Parallel and Distributed Implementation

For parallel execution, the graph vertices are allocated to computing nodes using a block partitioning scheme. One rank is spawned per computing node (that is, one rank per Q GPUs of each node), with R ranks across R nodes. The total number of GPUs used is $P = QR$. The number of vertices assigned to each GPU is given by $\Delta N = \lceil \frac{nd}{QR} \rceil$ (except for the last GPU, which has fewer). The GPU numbered q in rank r, $0 \leq q < Q$ of rank $0 \leq r < R$, is assigned vertices $[I, I + \Delta N)$, where $I = Q \times r \times \Delta N$.

On the Summit supercomputer, there are six GPUs per computing node (that is, $Q = 6$). For an experiment using $P = 1008$ GPUs, we launch an MPI run with $R = \frac{P}{Q} = \frac{1008}{6} = 168$ ranks, one rank per node. Within each rank, we initiate asynchronous launches of one kernel invocation per GPU. After all kernels are launched, their completion is synchronized on the CPU. Using our cuPPA-Hash multi-GPU algorithm [4], we process the block of nodes assigned to each GPU in parallel. At the end of this parallel execution across all the GPUs, a `cudaDeviceSynchronize()` operation is performed to serve as a barrier for all GPUs within a computing node. After the GPUs are synchronized, a `MPI_Barrier()` operation is performed to synchronize all MPI ranks to mark the completion of the graph generation.

Note that the number of edges being processed at any given time within the GPU memory is smaller than the total number of vertices assigned to that GPU. Based on the hardware capabilities of the GPU (for example, the number of CUDA threads), the number is varied. This number is optimized based on experimental observations [4] to use 512 threads.

3. EXPERIMENTAL RESULTS

In this section, we evaluate our algorithm and its performance by experimental analysis. We demonstrate the accuracy of our algorithm by showing that our algorithm produces very large networks with power law degree distribution as desired. We also compare the run time of our algorithm against other sequential and parallel algorithms.

3.1 Hardware and Software

We used the Summit machine that offers a few thousand compute nodes containing a mixture of CPUs and GPUs. Each node contains two IBM POWER9 processors (two CPU sockets) and six NVIDIA Volta V100 accelerators (six GPUs) connected by high speed NVLINK connections. Each node has 512 GB of DDR4 memory for use by the CPUs, 96 GB of High Bandwidth Memory (HBM2) for use by the GPUs, and 1.6TB of non-volatile memory. We used C/C++ and CUDA for programming the algorithm on CPU and GPU respectively. The `CUDA Compilation Tools V11` were used for the GPU code along with `nvcc` compiler.

3.2 Degree Distribution

We generated three big networks of 1 trillion edges, 2 trillion edges, and 4 trillion edges by executing the generation algorithm on 252, 504, and 1008 GPUs respectively. The degree distributions of these networks are shown in Figure 1 in a log-log scale. The number of nodes is set to 2 billion ($n = 2\mathbf{B}$) vertices and $d = 500, 1000, 2000$ for 252, 504, and 1008 GPUs respectively. All the vertices are stored in the GPU memory. Note that, using a 32-bit node identifier, all the GPU memory banks operate near their full capacity of 16 GB memory for each of the three different configurations.

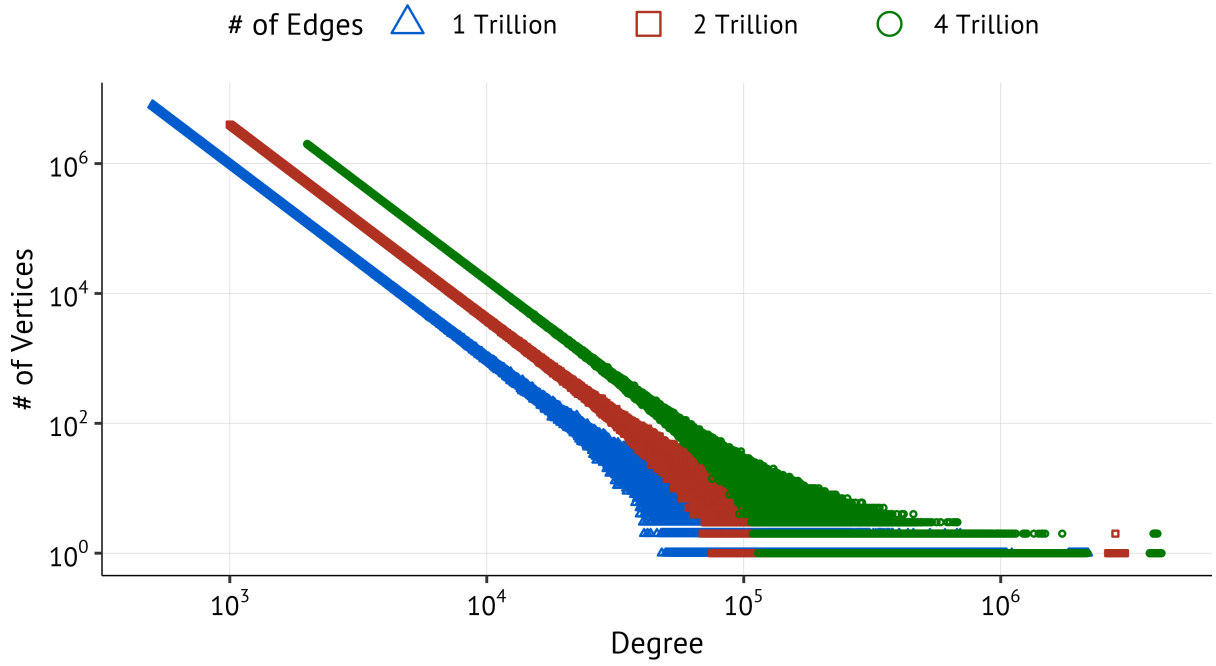

Figure 1. Validation of scale-free property (straight lines in log-log **scale degree distributions) of the generated networks for** $n = 2\mathbf{B}$, $d = 500, 1000, 2000$

The distribution is heavy-tailed, which is a distinct feature of the power-law networks. The exponent γ of the power-law degree distribution is measured to be 2.7. This supports the fact that for the finite average degree of a scale-free network, the exponent should be $2 < \gamma < \infty$ [8]. We also examined the degree distribution of the generated simple graphs and multigraphs. Based on the examination, we found that the degree distribution of simple and multigraphs for each configuration are quite similar in nature.

3.3 Graph Generation Time

Next we examined the time required to generate simple graphs and multigraphs with trillions of edges. The run times of the experiments are summarized in Table 1. At the fullest GPU memory capacity, the run time for multigraph is almost the similar for all three graph sizes. However, the time required for generating simple graph requires a heavy computational load of checking for duplicates, which affects the graph

generation time. For larger graph sizes and larger d, the effect on run time is more pronounced.

Table 1. Generation Time of Big Graphs

Graph Size	Number of GPUs	Generation Time (Seconds)		Million Edges per Second per GPU	
		Simple Graph	Multigraph	Simple Graph	Multigraph
1 trillion edges	252	29.42	2.42	134.88	1641.27
2 trillion edges	504	89.85	2.46	44.17	1616.23
4 trillion edges	1008	426.68	2.41	9.30	1649.96

We also investigated the time taken by each GPU in generating the networks for each configuration. These are shown in Figures 2, 3, and 4.

Simple Graph versus Multigraph: Note that generating a simple graph (by finding and eliminating duplicate edges) requires significantly more time than generating a multigraph (without checking for duplicate edges). This time effect is reflected among all GPUs. However, for generating a simple graph, the first GPU (with rank 0) requires significantly more time than other GPUs. This is due to the fact that, in preferential attachment, earlier vertices have fewer number of vertices to choose unique d edges and therefore require more frequent execution of the copy model due to more probable collisions. This phenomenon was experimentally observed in our earlier work as well [5].

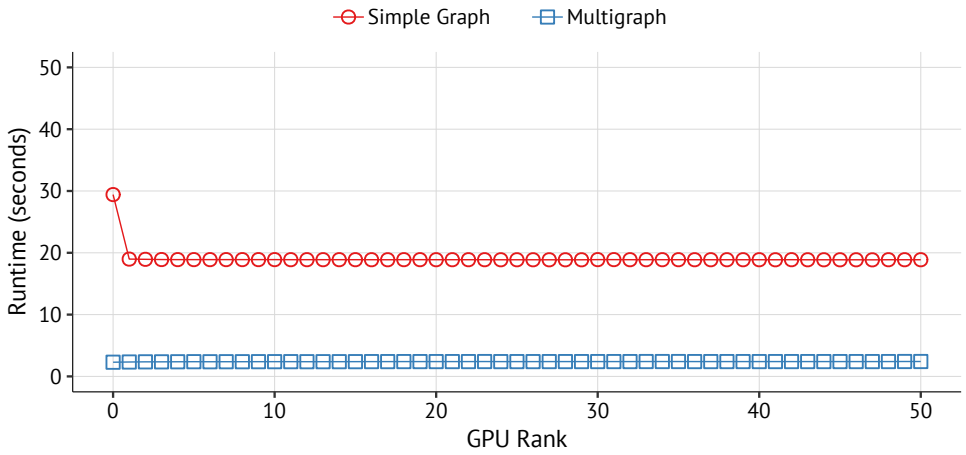

Figure 2. Distribution of run time in different GPU ranks for $n = 2$B, $d = 500$, and 252 GPUs (only the first 50 GPUs are shown)

Note that there is a processing cost for the verification (and elimination) for every edge in a batch of edges generated within each GPU. A verification kernel is executed on all the edges generated such that exactly d unique edges remain at the end of the kernel. Using a binary search tree to aid in searching and detecting duplication, the computational cost is kept low. For a batch of h edges generated in each vertex, there are $O(h \log d)$ comparison operations with the binary tree, since $h \geq d$ edges are consulted against d edges. This constitutes a (relatively) fixed cost in the computational time of simple graph, which is added to the time to generate the multigraph. This difference in times is observed in Figures 2, 3, and 4.

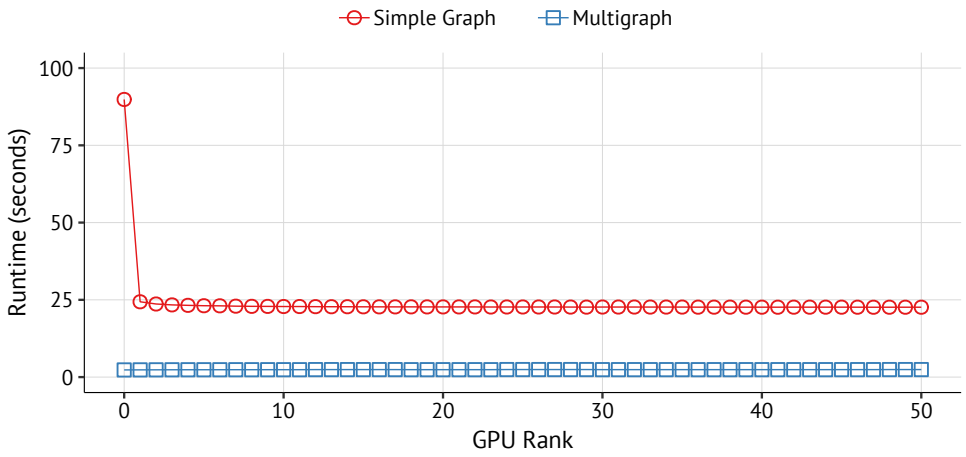

Figure 3. Distribution of run time in different GPU ranks for $n = 2\mathbf{B}$, $d = 1000$, and 504 GPUs (only the first 50 GPUs are shown)

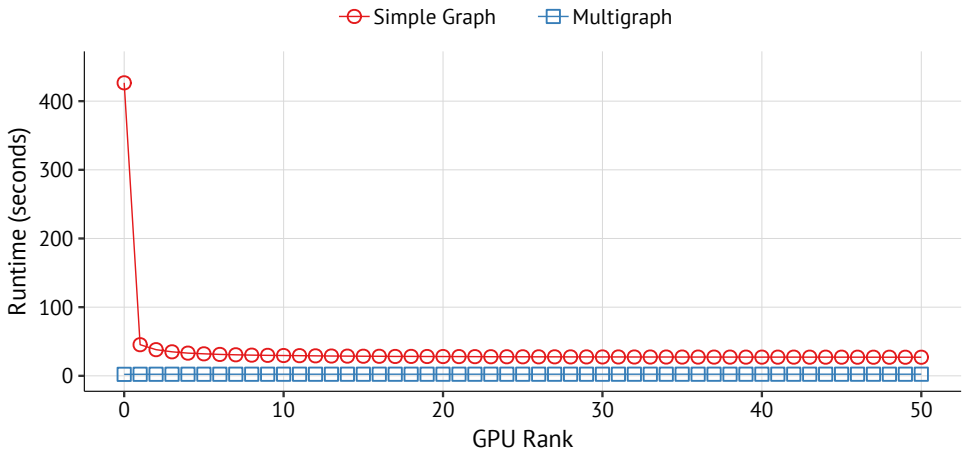

Figure 4. Distribution of run time in different GPU ranks for $n = 2\mathbf{B}$, $d = 2000$, and 1008 GPUs (only the first 50 GPUs are shown)

3.4 Strong Scaling

In this experiment we examine the strong scaling performance of cuPPA for generating graphs with trillions of edges. In strong scaling experiments, the size of the problem is fixed while more hardware resources are added to reap parallel processing performance.

Here, we fixed the number of nodes at 2 billion and degree at 500 ($n = 2\mathbf{B}$, $d = 500$) for generating one trillion edges ($1\mathbf{T}$). Note that 252 is the minimum number of GPUs needed to generate $1\mathbf{T}$ edges using 32-bit vertex identifiers. We then increased the number of GPUs to 504 and 1008 for the experiment, to observe the speed increase in generation. As 252 is the minimum number of GPUs required to generate trillion edges, we used it as a baseline to generate the (relative) strong scaling plot shown in Figure 5.

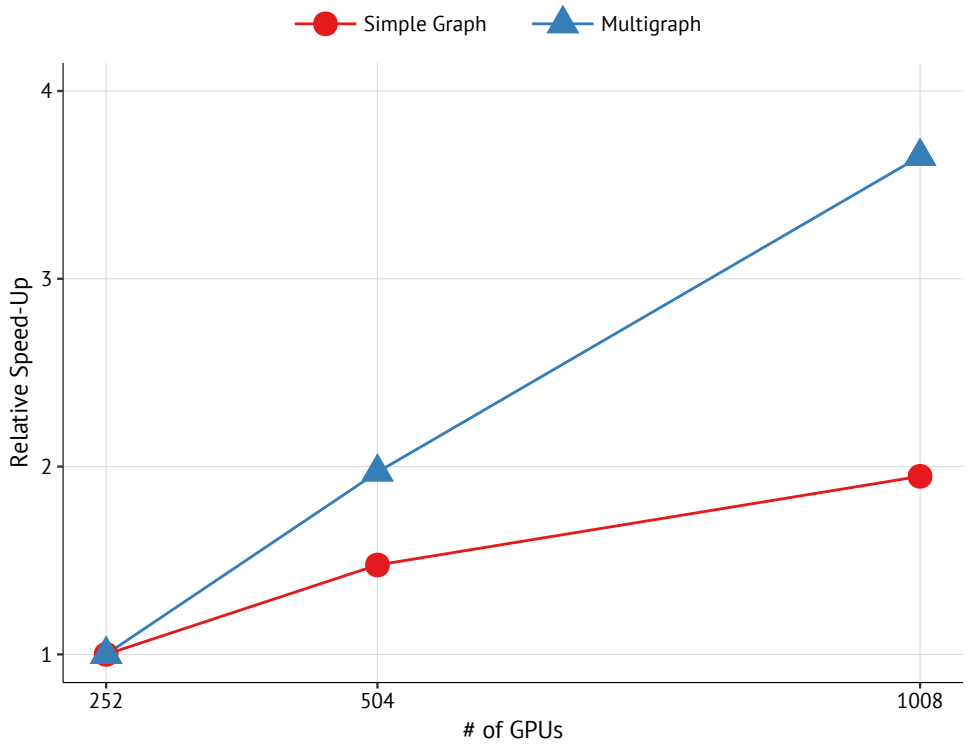

Figure 5. Relative speedup of cuPPA for generating Simple and Multigraph for $n = 2$B and $d = 500$ with varying number of GPUs

The relative speedup of generating multigraph is nearly close to the optimal level. When the number of GPUs is increased by a factor of 4, we achieved a speed increase by a factor of 3.65, which is 91% of the optimal value. However, the corresponding relative speedup for generating a simple graph is 1.95, which is approximately 49% of optimal value.

4. CONCLUSION

Synthetic graph networks of large sizes are useful for experiment-based understanding in applications such as scalable social networks. Here, we report one of the largest results in GPU-based parallel and distributed implementations of synthetic graph network generation using our `cuPPA` algorithm. Graph networks conforming to scale-free properties are generated with trillions of edges by executing our GPU-based algorithm on 1008 GPUs of a supercomputer. Our implementation is based on MPI-based communication for the distributed part of the execution, and uses CUDA for the multi-GPU parallel execution. Block partitioning of graph vertices is used across ranks and across GPUs within each rank. The issue of resolving the presence of parallel edges is addressed by resolving multigraph to simple graph during edge generation at runtime. A performance study of the run time and scaling is presented using scaled execution performed on the Summit supercomputer with over one thousand GPUs. Additional future work involves the development of a hybrid CPU-GPU execution in order to utilize unused cores of multi-core CPUs of the computing platform. Finally, our work can be extended to convert our internal, GPU-based graph

representation to other popular network formats for usability. In-place processing needs to be explored with graph algorithms that utilize the large scale-free network generated by our algorithm that stores the entire network in a distributed fashion across GPU memories.

5. REFERENCES

[1] Maksudul Alam and Kalyan S. Perumalla. Generating billion-edge scale-free networks in seconds: Performance study of a novel gpu-based preferential attachment model. Technical Report ORNL/TM-2017/486, Oak Ridge National Laboratory, 2017.

[2] Maksudul Alam and Kalyan S. Perumalla. Gpu-based parallel algorithm for generating massive scale-free networks using the preferential attachment model. In *IEEE International Conference on Big Data (Big Data)*, pages 3302–3311, 2017. doi: 10.1109/BigData.2017.8258315.

[3] Maksudul Alam, Maleq Khan, and Madhav V. Marathe. Distributed-memory parallel algorithms for generating massive scale-free networks using preferential attachment model. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12. ACM Press, 2013. ISBN 9781450323789. doi: 10.1145/2503210.2503291. URL http://doi.acm.org/10.1145/2503210.2503291.

[4] Maksudul Alam, Kalyan S. Perumalla, and Peter Sanders. Novel parallel algorithms for fast multi-gpu-based generation of massive scale-free networks. *Data Science and Engineering*, 4(1): 61–75, 2019. doi: 10.1007/s41019-019-0088-6.

[5] Maksudul Alam, Maleq Khan, Kalyan S. Perumalla, and Madhav Marathe. Generating massive scale-free networks: Novel parallel algorithms using the preferential attachment model. *ACM Trans. Parallel Comput.*, 7(2), May 2020. ISSN 2329-4949. doi: 10.1145/3391446. URL https://doi.org/10.1145/3391446.

[6] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286 (5439):509–12, 1999. ISSN 1095-9203. doi: 10.1126/science.286.5439.509. URL http://www.sciencemag.org/cgi/doi/10.1126/science.286.5439.509.

[7] David P. Chassin and Christian Posse. Evaluating north american electric grid reliability using the barabási-albert network model. *Physica A*, 355(2-4):667–677, 2005. ISSN 03784371. doi: 10.1016/j.physa.2005.02.051. URL http://dx.doi.org/10.1016/j.physa.2005.02.051.

[8] Sergey N. Dorogovtsev and José Fernando Ferreira Mendes. Evolution of networks. In *Advances in Physics*, volume 51, pages 1079–1187, 2002. doi: 10.1080/00018730110112519. URL http://www.tandfonline.com/doi/abs/10.1080/00018730110112519.

[9] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The web as a graph: Measurements, models, and methods. In *Annual International Conference on Computing and Combinatorics*, pages 1–17, Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 3-540-66200-6. URL http://dl.acm.org/citation.cfm?id=1765751.1765753.

[10] Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, D. Sivakumar, Andrew Tomkins, and Eli Upfal. Stochastic models for the web graph. In *Annual Symposium on Foundations of Computer Science*, pages 57–65. IEEE Comput. Soc, 2000. ISBN 0-7695-0850-2. doi: 10.1109/SFCS.2000.892065. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=892065.

[11] Jure Leskovec and Eric Horvitz. Planetary-scale views on a large instant-messaging network. In *International Conference on World Wide Web*, page 915. ACM Press, 2008. ISBN 9781605580852. doi: 10.1145/1367497.1367620. URL http://portal.acm.org/citation.cfm?doid=1367497.1367620.

[12] Peter Sanders and Christian Schulz. Scalable generation of scale-free graphs. *Information Processing Letters*, 116(7):489–491, 2016.