High-Performance Triangle Counting on GPUs

Yang Hu[†*], Hang Liu[‡*], and H. Howie Huang[†]
[†]The George Washington University
[‡]University of Massachusetts Lowell
[*]Equal contributions

***Abstract*—Counting triangles in a network is a primary step toward making sense of social networks. For instance, a graph with a large number of triangles is regarded as a "tightly knit community" with high degree of trust, because in this case all the friends (neighbors) of one vertex are also friends (connected) to each other. This work focuses on using Graphics Processing Units (GPUs) to accelerate triangle counting. To accommodate large graphs, the stat-of-the-art GPU-based triangle counting project – TriCore– simply stores the entire graph in the secondary storage to achieve communication free multi-GPU triangle counting. Our key observation is that each modern GPU server (Table I) often installs multiple GPUs which can easily overwhelm the disk bandwidth. Therefore, this paper introduces a new design for workload balancing to partition the graph and the workload in order to buffer each partitioned data in the CPU memory for faster data provisioning. Taken together, this work is the first, to the best of our knowledge, to advance the rate of triangle counting beyond 10^9 traversed edges per second (TEPS), as well as the first project that achieves $> 10^8$ TEPS for graphs with more than ten billion edges.**

I. Introduction

The triangle counting algorithm intersects the neighbor lists of two endpoints from each query edge to identify the triangles in a graph. To achieve high-performance triangle counting on GPUs, neither of the following designs is dispensable:

1) In pursuit of workload balancing, we often adopt edge centric triangle counting algorithm [7], that is, we use the edge list format of the graph to provide the query edge while compressed sparse row (CSR) representation for the access of the neighbor lists of the two endpoints from the query edge.
2) To reduce the time complexity, orientation [12] is exploited to remove half of the edges from the undirected version of the graph. In short, for each pair of undirected edges, orientation deletes the one edge that leads to the larger difference in out degrees between the two endpoints, which proves to reduce the complexity [12], [10].

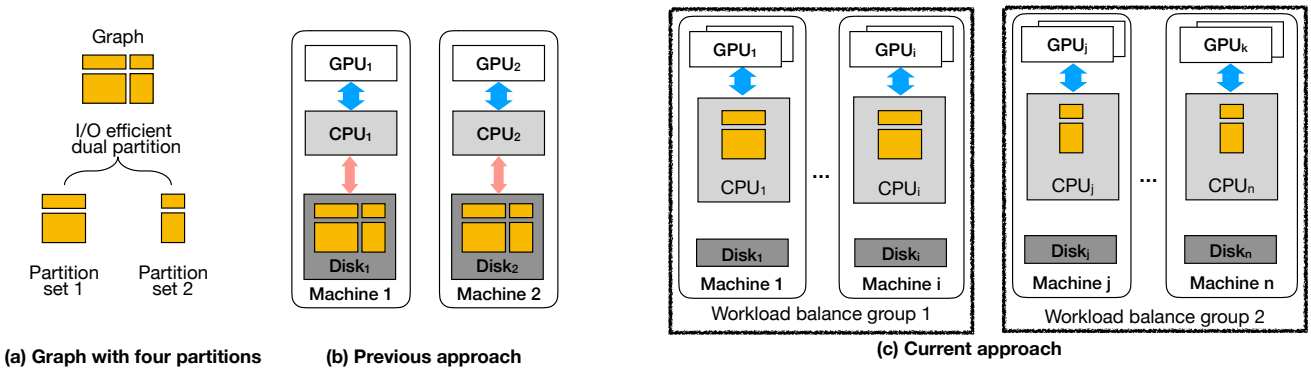
The state-of-the-art GPU-based triangle counting design [6], [7] comes with two techniques, that is, the binary search-based intersection, and the I/O efficient dual partition method. In particular, binary search-based intersection outperforms the mainstream merge path-based practice stemming from the fact that binary search-based intersection presents better memory access pattern atop GPUs' single instruction multiple data (SIMD) architecture. The second technique advances the traditional 2-D partition [9] for CSR representation of the graph for workload balancing. In addition, since most triangle counting work often exploits both CSR and edge list to count triangles, this dual partitioning method further splits the edge list to meet the partitions that are conducted atop CSR [7].

Computer and dataset	#GPUs	CPU memory space (GB)
Amazon EC2	8 V100	418
Google Cloud	8 V100	624
Microsoft Azure	4 V100	448
SDSC Comet	4 P100	128
Kron30 dataset	-	64
Kron31 dataset	-	128
Gsh dataset	-	124

TABLE I: #GPUs and the corresponding CPU memory space of each server, as well as the big graph sizes. Note, the space consumption of the graph comes from both the edge list and CSR formats. We use 4-byte to store the vertex ID while 8-byte for the offsets.

Despite the existing designs significantly boost the performance of triangle counting on GPUs, the key observation of this paper is that contemporary GPU servers install a collection of extremely powerful GPUs on a single machine. Consequently, conventional designs [6], [7] face the looming crisis that loading graph partitions from secondary storage (e.g., disk) to the CPU memory and further to GPUs may fail to catch up the processing capability of such an array of powerful GPUs. Fortunately, as shown in Table I, those powerful servers also equip considerable CPU memories, which can hold virtually all the big graphs. Therefore, we propose a workload balance group based buffering mechanism that exploits the CPU memory to cache the set of graph partitions of interest in the CPU memory to meet the speed needs on data provisioning from those attached GPUs. The detailed descriptions are presented in Section II and Figure 1.

Performance Highlights. Our evaluation encompasses two hightlights, as shown in Figure 2. Firstly, this is the first work, to the best of our knowledge, that achieves beyond billion TEPS (❶) performance for triangle counting. In Figure 2, these datasets are amazon, cit-Patent and roadNet (i.e., CA,PA,TX). Second, this is also the first work that retains 10^8 TEPS rate for graphs at the scale of ten billion edges (❷). Note, the rate of triangle counting often drops when graph becomes larger, as suggested by [10], [15], [11]. In this context, achieving $> 10^8$ counting rate is extremely challenging. In



(a) Graph with four partitions (b) Previous approach (c) Current approach

Fig. 1: For (a) graph with four partitions, we group them into two partition sets. (b) Previous approach stores the entire graph in the disk for each GPU card. In contrast, (c) Current approach buffers each partition set in CPU memory to avoid slow disk to CPU memory data movement. Note, each server maintains a partition set instead of single partition for the purpose of ease of workload stealing.

particular, these three graphs are KR30, KR31 and Gsh [5]. Note Kron30 and Kron31 are generated by the Graph 500 Kronecker generator [3] with the average edge factor as 16 and vertex count to be 2^{30} and 2^{31}, respectively.

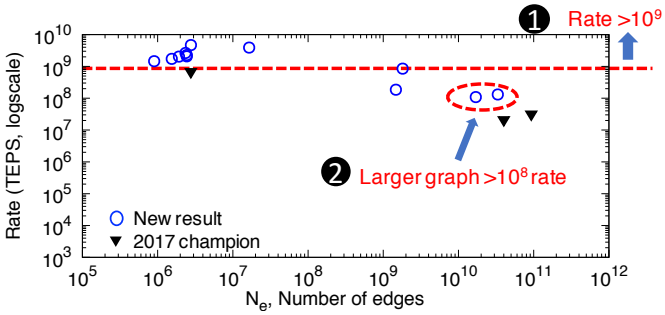


Fig. 2: Performance and graph size comparison between this work and the 2017 champion [11].

II. Workload Balance Group Design

Figure 1 compares the system designs of [6], [7] and our current approach. In particular, assuming the I/O efficient dual partition method splits the graph into four parts with roughly similar amount of edges. That is the yellow rectangles in Figure 1(a). Previous approach [6], as shown in Figure 1(b), simply stores the entire graph in the secondary storage disk for every participating server. During computation, each machine, on demand, loads the required partitions from disk to CPU memory and further to GPU in order to count triangle. Once workload imbalance surfaces, any GPU on any server can help tackle that problem resulting from the fact that each server maintains the entire copy of the graph. The issue of limited secondary storage bandwidth was not surfaced in TriCore [7] which only installs one relatively weak GPU on a server.

However, as shown in Table I, mainstream cloud providers and supercomputers tend to attach a collection of super powerful GPUs (e.g., NVIDIA V100 GPUs [1]) on each server. That is, the collective processing capabilities of all GPUs on one machine will reach a level that the aggregated throughput from the disks fails to satisfy. In this setting, disk bandwidth immediately becomes the major bottleneck for triangle counting which is also evident by ❶ from Figure 3.

To tackle this problem, we proposes to buffer the graph in the CPU memory so that moving data to GPU takes shorter time, thanks to the fact that contemporary servers also equip substantial amount of CPU memories that can hold a large portion of, if not the entire, graph.

Figure 1(c) illustrates our current design. In particular, we attempt to split the servers into workload balancing groups. As suggested by the name, this method allows each machine to address workload imbalance issues inside of the group. For instance, in Figure 1(c), we divide the graph into two partition groups. Next, the servers in workload balance group 1 are responsible for the partition group 1 of the graph while workload balance group 2 for partition group 2, and so on. This design can support graphs that are larger than the CPU memory because each server only needs to hold one partition group of the graph. If workload imbalance issue arises, all the servers from the same workload balance group can help alleviate the problem.

III. Results

We have implemented this project atop the prior HEPC work [6] with 800 extra lines of C++/CUDA code. We use CUDA toolkit 9.1 and G++ 5.4.0 with compilation flag as *O3* to compile the source code. Our test environment is the NVIDIA P100 GPUs from Comet supercomputers at San Diego Supercomputer Center (SDSC) [4] of the Extreme Science and Engineering Discovery Environment (XSEDE) [14].

We have examined all the graphs from Graph Challenge website, as plotted in Figure 1 and some extremely big graphs from Table III are downloaded from [2], [3]. In particular, we categorize the datasets into three categories, i.e., *small*, *medium* and *large*. For the first two categories, the vertex

Graph Name	$\|V\|$	$\|E\|$	#Triangles	Time (second)	TEPS
Friendster [13]	65,608,367	1,806,067,135	4,173,724,142	2.1	8.48E+08
Twitter [8]	41,652,230	1,468,365,182	34,824,916,864	6.5	2.26E+08
Scale25-16 [3]	33,554,432	523,611,003	22,535,831,016	2.5	2.06E+08
Twitter [8]: 2017 Champion [10] 256× machines	41,652,230	1,468,365,182	34,824,916,864	8.5	1.72E+08

TABLE II: Performance for medium size graphs on 8 GPUs.

Graph Name	$\|V\|$	$\|E\|$	#Triangles	Time (second)	TEPS
Kron30 [3]	1,073,741,824	17,022,115,838	1,074,908,326,232	156.7	1.08E+08
Kron31 [3]	2,147,483,648	34,101,759,806	2,306,560,594,152	586.8	5.81E+07
Gsh [5]	988,490,691	33,274,090,228	1,788,448,336,689	253.4	1.31E+08

TABLE III: Performance for large graphs on 32 GPUs (with partition algorithm).

and edge counts span from 4,040 to 65,608,367 and 12,572 and 1,806,067,135, respectively. For the last category, the vertex and edge counts are 988,490,691 – 2,147,483,648 and 17,022,115,838 – 33,274,090,228, respectively. Note, we use 32-bit and 64-bit to represent vertex ID and offset indexing, respectively.

We report three runtimes. For small size graphs with in-memory computation, we report *runtime* includes time from data ready in DRAM to the end, and *nomem runtime* with only GPU kernel time (excludes memory copy time between DRAM and GPU memory). For the results of big graphs with partition algorithm, we report the *total time*, includes the IO time, data copy to GPUs and computation time. We only report this because the MPI based load balancing scheme used for this implementation makes it hard to separate I/O, memory copy and GPU kernel time.

The following sections will report both the time consumption and corresponding traversed edges per second (TEPS) performance metrics for our evaluations. The time of performing graph partitioning is excluded. Besides, all the results are the average of 64 runs.

A. Small Graphs

For smaller graphs which contain less than 1 billion edges, we evaluate them on both a single and eight GPUs. As shown in Figure 3, for single GPU, we observe a collection of datasets from this category (i.e., P100 TEPS) achieve $> 10^8$ TEPS albeit including the data transferring data from CPU memory to GPU.

Once scaling to eight P100 GPUs, labeled as 8× P100 in Figure 3, we find the computing rate stays similar to 1×P100 case. We thus hypothesize moving data from CPU to GPU memory may dominate the entire time consumption given those datasets are extremely small. To address this issue, we directly store all the graphs in the GPU memory for these small graphs, which is labeled as *nonmem* in Figure 3. With this optimization, for 1×P100 nonmem, the majority of the graphs achieve beyond 10^8 computing rate. Observation ❶ from Figure 3 also backs our hypothesis.

And the most exciting achievements appear in 8×P100 nonmem case. In particular, we retain, on average, 7.3×10^8 computing rate with the minimum and maximum of 2.1×10^8 and 4.7×10^9 from roadNet-CA and as20000102 graphs, respectively. In total, eight datasets obtain $> 10^9$ TEPS computing rate. In Figure 3, we use ❷ to highlight the 10^9 computing rate. For ease of presentation, we also include the time consumption in Table IV. Our *nonmem* approach is similar to the 2017 graph challenge work that also reports the result excluding the memory copy time [15].

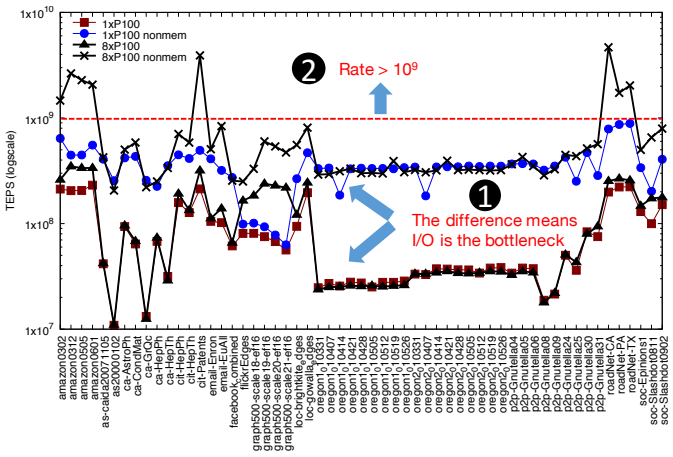


Fig. 3: TEPS for small graphs on one and eight GPUs, respectively. Corresponding time consumptions are listed in Table IV.

B. Medium Graphs

Table II presents the graph specifications, triangle counts, runtime and TEPS of three popular medium size graphs on eight P100 GPUs. These results are extremely close to 10^9 TEPS. Comparing to 2017 champion [10] which counts triangles for Twitter [8] datasets in 8.5 seconds with 256 machines, our design only needs 6.5 seconds on eight P100 GPUs.

C. Large Graphs

Table III further studies the performance of our design for extremely large graphs. In particular, we use 32 P100 GPUs to conduct the tests. Clearly, all the TEPS we achieve are beyond 10^8 rate. Note, larger graphs are often more challenging to compute triangles, as suggested by [10]. To the best of our

Graph	$\|V\|$	$\|E\|$	Triangle	Runtime (second)				Rate (TEPS)			
				1XP100	1XP100 nomem	8XP100	8XP100 nomem	1XP100	1XP100 nomem	8XP100	8XP100 nomem
amazon0302	262,112	899,792	717,719	4.25E-03	1.40E-03	3.46E-03	6.15E-04	2.12E+08	6.41E+08	2.60E+08	**1.46E+09**
amazon0312	400,728	2,349,869	3,686,467	1.15E-02	5.28E-03	6.76E-03	8.91E-04	2.05E+08	4.45E+08	3.48E+08	**2.64E+09**
amazon0505	410,237	2,439,437	3,951,063	1.18E-02	5.47E-03	7.27E-03	1.07E-03	2.06E+08	4.46E+08	3.36E+08	**2.28E+09**
amazon0601	403,395	2,443,408	3,986,507	1.06E-02	4.41E-03	7.28E-03	1.18E-03	2.31E+08	5.54E+08	3.36E+08	**2.07E+09**
as-caida20071105	26,476	53,381	36,365	1.28E-03	1.31E-04	1.27E-03	1.26E-04	4.16E+07	4.07E+08	4.21E+07	4.24E+08
as20000102	6,475	12,572	6,584	1.15E-03	4.94E-05	1.14E-03	6.10E-05	1.09E+07	2.55E+08	1.10E+07	2.06E+08
ca-AstroPh	18,773	198,050	1,351,441	2.11E-03	4.72E-04	2.06E-03	3.94E-04	9.37E+07	4.19E+08	9.60E+07	5.02E+08
ca-CondMat	23,134	93,439	173,361	1.46E-03	2.16E-04	1.37E-03	1.59E-04	6.41E+07	4.33E+08	6.82E+07	5.86E+08
ca-GrQc	5,243	14,484	48,260	1.10E-03	5.63E-05	1.16E-03	6.56E-05	1.32E+07	2.57E+08	1.25E+07	2.21E+08
ca-HepPh	12,009	118,489	3,358,499	1.74E-03	5.27E-04	1.61E-03	4.72E-04	6.79E+07	2.25E+08	7.35E+07	2.51E+08
ca-HepTh	9,878	25,973	28,339	8.23E-04	7.35E-05	8.98E-04	7.74E-05	3.16E+07	3.53E+08	2.89E+07	3.36E+08
cit-HepPh	34,547	420,877	1,276,868	2.66E-03	9.40E-04	2.19E-03	5.97E-04	1.58E+08	4.48E+08	1.92E+08	7.05E+08
cit-HepTh	27,771	352,285	1,478,735	2.78E-03	8.50E-04	2.63E-03	6.03E-04	1.27E+08	4.14E+08	1.34E+08	5.85E+08
cit-Patents	3,774,769	16,518,947	7,515,023	7.77E-02	3.34E-02	5.20E-02	4.22E-03	2.13E+08	4.94E+08	3.18E+08	**3.91E+09**
email-Enron	36,693	183,831	727,044	1.75E-03	4.48E-04	1.65E-03	3.63E-04	1.05E+08	4.10E+08	1.12E+08	5.06E+08
email-EuAll	265,215	364,481	267,313	3.56E-03	1.14E-03	2.62E-03	4.36E-04	1.02E+08	3.19E+08	1.39E+08	8.36E+08
facebook_combined	4,040	88,234	1,612,010	1.43E-03	3.23E-04	1.34E-03	3.47E-04	6.15E+07	2.73E+08	6.58E+07	2.55E+08
flickrEdges	105,939	2,316,948	107,987,357	2.87E-02	2.35E-02	1.41E-02	9.26E-03	8.07E+07	9.87E+07	1.65E+08	2.50E+08
graph500-scale18-ef16	174,148	3,800,348	82,287,285	4.70E-02	3.77E-02	2.07E-02	1.15E-02	8.09E+07	1.01E+08	1.84E+08	3.30E+08
graph500-scale19-ef16	335,319	7,729,675	186,288,972	1.03E-01	8.29E-02	3.24E-02	1.29E-02	7.52E+07	9.32E+07	2.39E+08	6.01E+08
graph500-scale20-ef16	645,821	15,680,861	419,349,784	2.32E-01	2.01E-01	6.85E-02	2.91E-02	6.75E+07	7.79E+07	2.29E+08	5.38E+08
graph500-scale21-ef16	1,243,073	31,731,650	935,100,883	5.65E-01	5.04E-01	1.45E-01	6.73E-02	5.62E+07	6.30E+07	2.19E+08	4.71E+08
loc-brightkite_edges	58,229	214,078	494,728	2.27E-03	8.04E-04	1.78E-03	3.85E-04	9.43E+07	2.66E+08	1.21E+08	5.56E+08
loc-gowalla_edges	196,592	950,327	2,273,138	4.86E-03	2.03E-03	3.90E-03	1.17E-03	1.96E+08	4.68E+08	2.44E+08	8.09E+08
oregon1_010331	10,671	22,002	17,144	8.90E-04	6.62E-05	9.24E-04	7.57E-05	2.47E+07	3.32E+08	2.38E+07	2.91E+08
oregon1_010407	10,730	21,999	15,834	8.11E-04	6.55E-05	8.81E-04	7.49E-05	2.71E+07	3.36E+08	2.50E+07	2.94E+08
oregon1_010414	10,791	22,469	18,237	8.78E-04	1.21E-04	8.91E-04	7.18E-05	2.56E+07	1.86E+08	2.52E+07	3.13E+08
oregon1_010421	10,860	22,747	19,108	8.15E-04	6.90E-05	8.83E-04	6.86E-05	2.79E+07	3.30E+08	2.58E+07	3.32E+08
oregon1_010428	10,887	22,493	17,645	8.21E-04	6.71E-05	8.81E-04	7.49E-05	2.74E+07	3.35E+08	2.55E+07	3.00E+08
oregon1_010505	10,944	22,607	17,597	9.02E-04	6.79E-05	8.78E-04	7.48E-05	2.51E+07	3.33E+08	2.58E+07	3.02E+08
oregon1_010512	11,012	22,677	17,598	8.16E-04	6.82E-05	8.97E-04	7.62E-05	2.78E+07	3.33E+08	2.53E+07	2.98E+08
oregon1_010519	11,052	22,724	17,677	8.16E-04	6.88E-05	8.84E-04	5.81E-05	2.78E+07	3.30E+08	2.57E+07	3.91E+08
oregon1_010526	11,175	23,409	19,894	8.18E-04	6.91E-05	8.97E-04	7.62E-05	2.86E+07	3.39E+08	2.61E+07	3.07E+08
oregon2_010331	10,901	31,180	82,856	9.40E-04	9.08E-05	9.32E-04	9.68E-05	3.32E+07	3.43E+08	3.34E+07	3.22E+08
oregon2_010407	10,982	30,855	78,138	9.37E-04	1.68E-04	9.24E-04	1.01E-04	3.29E+07	1.83E+08	3.34E+07	3.06E+08
oregon2_010414	11,020	31,761	88,905	8.51E-04	9.24E-05	9.25E-04	9.96E-05	3.73E+07	3.44E+08	3.43E+07	3.19E+08
oregon2_010421	11,081	31,538	82,129	8.51E-04	9.14E-05	8.92E-04	7.99E-05	3.71E+07	3.45E+08	3.54E+07	3.94E+08
oregon2_010428	11,114	31,434	78,000	8.60E-04	9.10E-05	9.24E-04	9.78E-05	3.65E+07	3.45E+08	3.40E+07	3.21E+08
oregon2_010505	11,158	30,943	72,182	8.49E-04	8.81E-05	9.19E-04	9.54E-05	3.64E+07	3.51E+08	3.37E+07	3.24E+08
oregon2_010512	11,261	31,303	72,866	9.21E-04	9.02E-05	9.16E-04	9.75E-05	3.40E+07	3.47E+08	3.42E+07	3.21E+08
oregon2_010519	11,376	32,287	83,709	8.52E-04	9.19E-05	9.15E-04	1.02E-04	3.79E+07	3.51E+08	3.53E+07	3.18E+08
oregon2_010526	11,462	32,730	89,541	8.55E-04	9.34E-05	9.35E-04	1.02E-04	3.83E+07	3.50E+08	3.50E+07	3.21E+08
p2p-Gnutella04	10,877	39,994	934	1.18E-03	1.09E-04	1.23E-03	1.10E-04	3.40E+07	3.67E+08	3.26E+07	3.64E+08
p2p-Gnutella05	8,847	31,839	1,112	8.41E-04	8.62E-05	9.02E-04	7.45E-05	3.79E+07	3.69E+08	3.53E+07	4.27E+08
p2p-Gnutella06	8,718	31,525	1,142	8.44E-04	8.60E-05	9.16E-04	8.99E-05	3.74E+07	3.67E+08	3.44E+07	3.51E+08
p2p-Gnutella08	6,302	20,777	2,383	1.10E-03	6.48E-05	1.16E-03	7.25E-05	1.88E+07	3.21E+08	1.79E+07	2.86E+08
p2p-Gnutella09	8,115	26,013	2,354	1.21E-03	7.40E-05	1.18E-03	7.95E-05	2.15E+07	3.51E+08	2.20E+07	3.27E+08
p2p-Gnutella24	26,519	65,369	986	1.30E-03	1.54E-04	1.29E-03	1.46E-04	5.01E+07	4.25E+08	5.08E+07	4.47E+08
p2p-Gnutella25	22,688	54,705	806	1.52E-03	2.17E-04	1.27E-03	1.26E-04	3.61E+07	2.52E+08	4.30E+07	4.35E+08
p2p-Gnutella30	36,683	88,328	1,590	1.06E-03	1.88E-04	1.10E-03	1.71E-04	8.36E+07	4.69E+08	8.01E+07	5.15E+08
p2p-Gnutella31	62,587	147,892	2,024	1.97E-03	5.18E-04	1.57E-03	2.62E-04	7.52E+07	2.85E+08	9.41E+07	5.65E+08
roadNet-CA	1,965,207	2,766,607	120,676	1.39E-02	3.51E-03	1.09E-02	5.92E-04	1.99E+08	7.87E+08	2.53E+08	**4.67E+09**
roadNet-PA	1,088,093	1,541,898	67,150	6.96E-03	1.78E-03	5.77E-03	8.92E-04	2.22E+08	8.67E+08	2.67E+08	**1.73E+09**
roadNet-TX	1,379,918	1,921,660	82,869	8.61E-03	2.16E-03	7.48E-03	9.46E-04	2.23E+08	8.88E+08	2.57E+08	**2.03E+09**
soc-Epinions1	75,880	405,740	1,624,481	3.11E-03	1.20E-03	2.77E-03	8.13E-04	1.30E+08	3.38E+08	1.47E+08	4.99E+08
soc-Slashdot0811	77,361	469,180	551,724	4.70E-03	2.33E-03	2.71E-03	7.18E-04	9.97E+07	2.02E+08	1.73E+08	6.54E+08
soc-Slashdot0902	82,169	504,230	602,592	3.34E-03	1.24E-03	2.86E-03	6.36E-04	1.51E+08	4.06E+08	1.77E+08	7.93E+08

TABLE IV: Time consumption (second) and computing rate (TEPS) for various small graphs on one and eight GPUs.

knowledge, this is the first GPU-based triangle counting that can tackle graphs with $> 10^{10}$ edges, and more importantly, achieve a performance of $> 10^8$ TEPS.

IV. Conclusion

This work breaks the billion TEPS performance ceiling for the triangle counting algorithm. In particular, we use the CPU memory to buffer the graph data in order to rapidly move data from CPU to GPU, thereby, achieving fast triangle counting on GPUs. To the best of our knowledge, this is the first work which advances triangle counting beyond 10^9 TEPS. Furthermore, this is also the first GPU work that achieves $> 10^8$ TEPS rate for large graphs ($> 10^{10}$ edges).

V. Acknowledgment

This work was partially supported by National Science Foundation CAREER award 1350766 and grants 1618706 and 1717774 at George Washington University. This research used resources from XSEDE and Amazon AWS research credits at University of Massachusetts Lowell.

References

[1] NVIDIA TESLA V100 GPU ACCELERATOR, http://www.nvidia.com/content/pdf/volta-datasheet.pdf.

[2] Paolo Boldi, Andrea Marino, Massimo Santini, and Sebastiano Vigna. BUbiNG: Massive crawling for the masses. In *Proceedings of the Companion Publication of the 23rd International Conference on World Wide Web*, pages 227–228. International World Wide Web Conferences Steering Committee, 2014.

[3] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In *SDM*, 2004.

[4] Comet Supercomputer at SDSC. http://www.sdsc.edu/support/user_guides/comet.html, 2018.

[5] Gsh dataset from WebGraph. http://law.di.unimi.it/webdata/gsh-2015/, 2015.

[6] Yang Hu, Pradeep Kumar, Guy Swope, and H Howie Huang. Trix: Triangle counting at extreme scale. In *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*, pages 1–7. IEEE, 2017.

[7] Yang Hu, Hang Liu, and H Howie Huang. Tricore: Parallel triangle counting on gpus. In *High Performance Computing, Networking, Storage and Analysis, 2018 SC-International Conference for*. IEEE, 2018.

[8] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *WWW*, 2010.

[9] Hang Liu and H Howie Huang. Graphene: Fine-grained io management for graph computing. In *FAST*, pages 285–300, 2017.

[10] Roger Pearce. Triangle counting for scale-free graphs at scale in distributed memory. In *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*, pages 1–4. IEEE, 2017.

[11] Siddharth Samsi, Vijay Gadepally, Michael Hurley, Michael Jones, Edward Kao, Sanjeev Mohindra, Paul Monticciolo, Albert Reuther, Steven Smith, William Song, et al. Graphchallenge. org: Raising the bar on graph analytic performance. *arXiv preprint arXiv:1805.09675*, 2018.

[12] Julian Shun and Kanat Tangwongsan. Multicore triangle computations without tuning. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2015.

[13] SNAP: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data/.

[14] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, et al. Xsede: accelerating scientific discovery. *Computing in Science & Engineering*, 16(5):62–74, 2014.

[15] Chad Voegele, Yi-Shan Lu, Sreepathi Pai, and Keshav Pingali. Parallel triangle counting and k-truss identification using graph-centric methods. In *High Performance Extreme Computing Conference (HPEC), 2017 IEEE*, pages 1–7. IEEE, 2017.