Update on Triangle Counting on GPU

Carl Pearson*, Mohammad Almasri*, Omer Anjum*, Vikram S. Mailthody*, Zaid Qureshi§,
Rakesh Nagi‡, Jinjun Xiong†, and Wen-mei Hwu*
*ECE, §CS, ‡ISE, University of Illinois at Urbana-Champaign, Urbana, IL 61801
†Cognitive Computing Systems Research, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 10598
{pearson, almasri3, oanjum, vsm2, zaidq2}@illinois.edu,
nagi@illinois.edu, jinjun@us.ibm.com, w-hwu@illinois.edu

***Abstract*—This work presents an update to the triangle-counting portion of the subgraph isomorphism static graph challenge. This work is motivated by a desire to understand the impact of CUDA unified memory on the triangle-counting problem. First, CUDA unified memory is used to overlap reading large graph data from disk with graph data structures in GPU memory. Second, we use CUDA unified memory hints to solve multi-GPU performance scaling challenges present in our last submission. Finally, we improve the single-GPU kernel performance from our past submission by introducing a work-stealing dynamic algorithm GPU kernel with persistent threads, which makes performance adaptive for large graphs without requiring a graph analysis phase.**

***Index Terms*—GPU, graph algorithms, triangle counting**

I. Introduction

Triangles are a simple non-trivial structure of many graphs, and are used as foundational elements for graph analyses and transformations. The widespread adoption of GPUs in commodity, cloud-computing, and specialized supercomputing systems, provides substantial motivation for accelerating graph operations on GPUs, whether for stand-alone operations or as part of an extended pipeline. Of particular interest is whether large-scale graph operations can be effectively parallelized to multiple tightly-connected GPUs.

A key challenge of using GPUs for large-scale graph operations is the low compute intensity of those operations and the relatively low bandwidth between GPU memory and system memory. At the same time, GPU system integrators like IBM have developed and released programmer-friendly and tightly-integrated heterogeneous CPU/GPU systems, where GPUs have full access to expansive host memory and interconnects that have an order of magnitude higher bandwidth than previously available. In previous submissions, we presented our collaborative CPU/GPU approach to triangle counting and *k*-truss decomposition based on CUDA's zero-copy and unified memory for optimizations to our static graph challenge algorithms.

In this work, we examine the impact of programmer-friendly and tightly-integrated GPU programming APIs on triangle counting. We make the following contributions:

1) Triangle counting based on persistent-kernel dynamic algorithm selection;
2) Evaluation of performance impact of using unified memory for in-GPU-memory triangle counting;
3) Evaluation of performance impact of using unified memory for computation support, implicit graph partitioning, and multi-GPU scaling;
4) Unified memory for overlap of graph construction in GPU memory with disk I/O.

II. Design

The algorithm operates on an unweighted directed acyclic graph $G = (E, V)$, where each edge E is from a source vertex V_s to a destination vertex V_d. Each triangle is uniquely associated with a particular "base" edge. Each edge in the graph may be the base edge of zero or more triangles, and the triangle count for the graph is the sum of the triangle count of each base edge. Most datasets are provided by the GraphChallenge in a sorted edge-list file format. Similar to our previous submission, we follow an edge-oriented approach.

A. Graph Storage

In previous submissions, we used a hybrid "COO+CSR" (coordinate + compressed-sparse-row) format, where the CSR *colInd* array is paired with an equal-length *rowInd* array containing the non-zero row index. In a traditional CSR, each parallel task is typically assigned to a row, and then while processing that row additional parallelism may optionally be leveraged to access each entry in that row. Parallel tasks are not directly mapped to row entries, as the length of each row is not known until the CSR is accessed. The inclusion of the *rowInd* array makes it straightforward to parallelize across edges instead of rows in the CSR, as each parallel task can look up the source vertex of an edge without traversing the *rowPtr* array. We refer to the **rowPtr**, **rowInd**, and **colInd** arrays in the algorithms in Section II-D.

B. CUDA Unified Memory and Hints

CUDA unified memory provides a single coherent memory image available to all participating devices (in this case, CPUs and GPUs). At page granularity data is migrated to the memory of the device that has used it most recently. Since data access patterns for graphs are data-dependant, we hope to use this fine-grained on-demand data transfer system as a platform to implicitly partition graph data across GPUs and prevent unnecessary data transfers.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

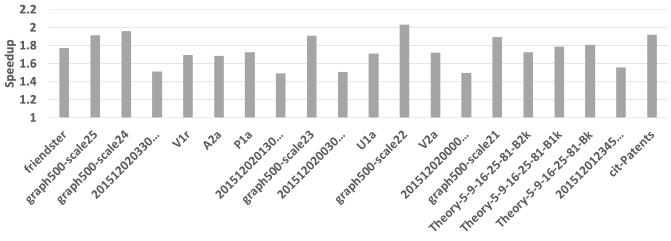

Figure 1. Speedup achieved by overlapping disk I/O with COO+CSR construction, over first reading the entire edge list and then constructing the COO+CSR representation.

Unified memory comes with a large performance cost of handling page faults as GPU threads access non-local pages. When a thread makes an access, the warp is stalled while the page is transferred to the demanding device and the device's page table is updated. Furthermore, if multiple devices all access the same page, the page may thrash between devices, bringing forward process to a near stand-still. A programmer with some knowledge of application data-access patterns can use hints to the system to affect the behavior of the page migration and reduce the performance costs.

- *cudaMemAdviseSetReadMostly*: a memory region is marked as read-mostly, and devices which read from this region get a read-only copy of the page. This prevents multiple devices that only read data from thrashing.
- *cudaMemPrefetchAsync*: This prefetches a memory region to a device, preventing the device from stalling while during a demand page migration, as the page will already be on the device.

C. COO+CSR Construction and Disk I/O Overlap

Thanks to unified memory, we are able to overlap CSR construction in the GPU address with the disk I/O for reading the data. Two threads are synchronized by a shared double-buffer, which one thread fills from the edge list file and the other consumes to create the COO+CSR in CUDA unified memory. **As such `total_time` includes the cost of disk I/O**; `gpu_time` does not. Figure 1 shows the speedup of overlapping these operations, compared with disk I/O alone followed by CSR construction. Both operations take approximately the same amount of time, leading to nearly a 2x speedup. For these measurements, the disk I/O is accelerated by the edge list file contents being in the operating system disk cache, as well as being in a binary format.

D. Triangle-Counting Kernel

We continue to approach the triangle-counting problem in terms of a per-edge count of common elements between source and destination node neighbor lists. In our previous submission, we used a single algorithm regardless of the incoming graph. Like many submissions, we continue to take advantage of the observation that converting the undirected graph into a DAG (directed acyclic graph) can cut the triangle counting work in half. By directing edges along a total ordering of nodes, the graph can be converted to a DAG. In this work, we direct all edges from lower to higher vertex id. Other works take a different approach, such as [1], [2] who order by degree. We did not take this approach, as we feel it requires a pass over the entire graph before the orientation of each edge can be known.

Algorithm 1 Two-pointer intersection count

```
Input:  a, aSz, b, bSz: sorted arrays a, b and sizes
Output: c: number of common elements in a and b
 1: procedure THSEQCNT(a, aSz, b, bSz)
 2:     c = 0
 3:     ap = a; loadA = true
 4:     bp = b; loadB = true
 5:     while ap < a + aSz and bp < b + bSz do
 6:         if loadA then
 7:             aVal = *ap; loadA = false
 8:         end if
 9:         if loadB then
10:             bVal = *bp; loadB = false
11:         end if
12:         if aVal == bVal then
13:             c += 1
14:             ap += 1; loadA = true
15:             bp += 1; loadB = true
16:         else if aVal < bVal then
17:             ap += 1; loadA = true
18:         else
19:             bp += 1; loadB = true
20:         end if
21:     end while
22:     return c
23: end procedure
```

1) Two-Pointer Intersection Count: We continue to use a two-pointer intersection count. This approach is similar to the ones described by other finalists. Algorithm 1 shows our current two-pointer intersection count implementation. The expected number of accesses for each thread is $aSz + bSz$, where $aSz = aEnd - aBegin$ and $bSz = bEnd - bBegin$. Compared to our submission last year, this implementation prevents generating a load instruction when a pointer has not moved, and fixes an out-of-bounds access bug.

2) Binary Search Intersection Count: In addition to the linear intersection count, we considered a binary-search approach similar to 2017 finalist/2018 champion Hu[1] and 2018 finalist Fox[3], who motivate it in more detail. Each value in one array may be sought in the other array using a binary search. To summarize, the expected number of global memory accesses is $cSz \times log2(dSz)$, where cSz is the shorter of the two arrays and dSz is the longer. When the lengths of the arrays are severely mismatched, the binary search will require fewer memory accesses than the two-pointer search. Furthermore, every element of the shorter array can be sought in the longer array in parallel. This yields good cache performance, as adjacent threads' accesses in the binary search will all be to the same or similar locations.

E. Static Kernel

In our static kernels, we assign each thread to a single edge. Each thread loads the source and destination vertex from the `rowInd` and `colInd` arrays (the COO), then traverses the

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

CSR part of the COO+CSR to determine which entries in the `colInd` array correspond to the neighbor list of each vertex.

The number of common elements (triangles) between the two neighbor lists is counted using the two-pointer sequential intersection For the linear kernel, Algorithm 1is used for the "linear" kernel.

The binary kernel uses a single thread, which searches for values from the shorter neihbor list in the longer neighbor list using a binary search. At the end of the kernel execution, each thread block uses a CUB [4] BLOCKSUM to aggregate the count into thread 0, which uses a single ATOMICADD to contribute its final count to memory.

F. Dynamic Algorithm Selection

Fox et al. [3] presented a logarithmic radix binning approach, where different amounts of parallelism were provided to different edges based on an estimate of their workload. Their results showed that the performance overhead of the binning was non-trivial, so we attempt to capture a similar idea without the associated overhead.

Instead of providing a different amount of parallelism to each edge, we apply a different intersection-count approach to each edge based on the expected number of loads described in Sections II-D1 and II-D2. This should give the benefit of fast $\mathcal{O}(log_2)$ search through very long rows, or a fast sequential search through shorter rows.

Furthermore, instead of paying the price of binning edges by expected workload ahead of time, we implement a work-stealing approach at the granularity of 32-thread warps. In our static kernels, where the grid size is determined by the number of edges, a single long-running thread could prevent an otherwise idle block from finishing and freeing resources for other blocks to begin their work.

Instead of statically assigning threads to edges, we create enough GPU warps to fully occupy the device, and then allow these persistent warps to work-steal edges from each other until the entire triangle count has been completed. Modern GPUs still broadly follow the lockstep execution model for threads in a warp, so any warp that gets stuck with a long-running thread will still block the other threads in that warp from doing useful work. However, the other warps in the block will be free to claim additional edges to count. This is analogous to the work-stealing runtime used by [5] except at the granularity of edges instead of partitions. We choose groups of 32 edges because if the two-pointer algorithm is selected, the 32 threads in each warp can execute independently on those edges. If the binary algorithm is selected, the warp can collaboratively count triangles without synchronizing the entire block.

Algorithm 2 shows pseudocode for our implementation. WARPBCAST is a function that transmits the first parameter from the thread with the lane index of the second parameter to every thread in the warp. WARPSUM similarly collaboratively sums the values in the first parameter of every warp and returns the result to the thread with the lane index of the second parameter. WARPBINCNT is a warp-collaborative binary search algorithm, where each thread in the warp searches for an element of the shorter list in the longer list using a binary search. In line 1, each thread computes its lane index within the warp. Lines 4-7 lane 0 uses an atomic operation to work-steal a group of 32 edges and broadcasts the starting edge to every other warp in the group. Lines 8-10 terminates the warp if no thread has a valid edge to work on. Lines 11-14 traverse the CSR data to get the offset and size of the source and destination vertex neighbor lists. Lines 15-25 compute an aggregate cost of using either the linear or binary intersection algorithm for those 32 threads. The binary cost is scaled by a user-tunable sf factor. The effects of that factor are explored in Figure 2. Lines 26-36 use the threads in the warp to either count triangles for all 32 edges in parallel with the sequential approach, or collaboratively count triangles using the binary search approach for one edge at a time. Finally, lines 38-41 do a collaborative summation of all of the triangles discovered by this block and contribute them to the total count with an

Algorithm 2 Edge-oriented Dynamic Kernel

Input: sf: a tuning parameter for the selection heuristic
Input: ei: the starting edge index (0)
Output: c: the total count

```
 1: lx = THREADIDX.X % 32                          ▷ lane in warp
 2: thCnt = 0                                      ▷ this thread's count
 3: while true do
 4:     if 0 then == lx
 5:         wi = ATOMICADD(ei, 32)                 ▷ first edge in 32
 6:     end if
 7:     wi = WARPBCAST(wi, 0)
 8:     if wi > nnz + 32 then
 9:         break
10:     end if
11:     src = colInd[rowPtr[src]]
12:     srcSz = colInd[rowPtr[src+1]] - src
13:     dst = colInd[rowPtr[dst]]
14:     dstSz = colInd[rowPtr[dst+1]] - dst
15:     lCost = srcSz + dstSz
16:     lcost = WARPSUM(lcost, 0)
17:     lcost = WARPBCAST(lcost, 0)
18:     if srcSz < dstSz then
19:         bincost = sf × srcSz × log2(dstSz)
20:     else
21:         bincost = sf ×dstSz × log2(srcSz)
22:     end if
23:     bincost = WARPSUM(bincost, 0)
24:     bincost = WARPBCAST(bincost, 0)
25:     if lcost < bcost then
26:         thCnt = THSEQCNT(src, srcSz, dst, dstSz)
27:     else
28:         for j = wi; j < iw + 32 and j < nnz; ++j do
29:             eSrc = WARPBCAST(src, j - wi)
30:             eSrcSz = WARPBCAST(srcSz, j - wi)
31:             eDst = WARPBCAST(dst, j - wi)
32:             eDstSz = WARPBCAST(dstSz, j - wi)
33:             thCnt = WARPBINCNT(eSrc, eSrcSz, eDst, eDstSz)
34:         end for
35:     end if
36: end while
37: thCnt = BLOCKSUM(THCNT)
38: if thenthreadIdx.x == 0
39:     ATOMICADD(c, thCnt)
40: end if
```

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

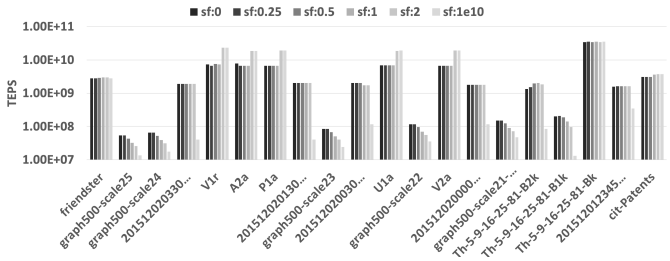

Figure 2. Performance of the dynamic selection kernel on the twenty largest graphs with a varying scale factor in the cost heuristic. A larger scale factor favors the two-pointer intersection algorithm over the binary intersection algorithm.

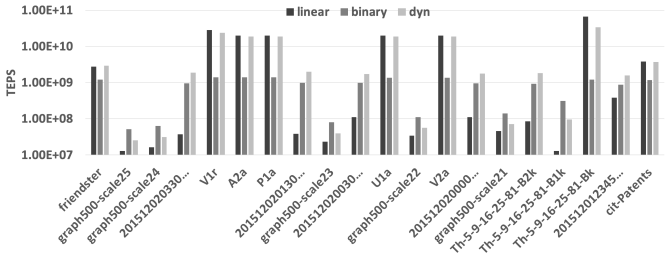

Figure 3. Achieved TEPS of various kernel implementations on the twenty graphs with the largest edge count.

ATOMICADD.

Figure 2 shows how the performance of the dynamic kernel varies with the tunable scale factor. The larger the scale factor, the more expensive the binary search is considered and the more likely that the linear approach is selected for any group of 32 edges. We can see that some graphs greatly favor the linear approach, particularly graphs where the average out degree is low and the variance in the out degree is low. The internet topology graphs performance drops sharply for the linear method, because they feature several extremely populous rows, with mostly empty rows. With the linear kernel, an entire warp may be blocked for a long time while a single thread works through the lists. When any binary search is allowed, binary search into the long rows prevents the performance degredataion. The `graph500` family of graphs exhibits gradually decreasing performance as the linear approach is favored. This suggests the performance model is not quite able to discriminate successfully in all cases. Based on these results, we select a scale factor of 2 for the final evaluation.

Figure 3 shows the achieved kernel TEPS (traversed edges per second, or edges divided by time), for the twenty largest graphs in the GraphChallenge dataset (by edge count). In most cases, the dynamic kernel matches the performance of either the linear or binary approach alone, without having to make a heuristic decision based on the graph properties before counting begins. This style of approach is promising for translating performance improvements to never-before-seen graphs. Only in the case of the `graph500` family does the dynamic kernel not match the performance of either other approach. This is consistent with the reduced performace for scale factor 2 in those graphs.

TABLE I
IBM AC922 "NEWELL" ARCHITECTURE SUMMARY.

CPU	2 × POWER9
System RAM	512 GB
GPU	4x NVidia V100 (16GB) [7]
CPU-CPU Interconnect	64 GB/s X-bus
NVLink Triad Interconnect	V2.0 x3 (150 GB/s)
CUDA Release	9.2.148
Nvidia Driver	396.44
Linux	4.14.0-49.13.1.el7a.ppc64le
Page Size	64 KB

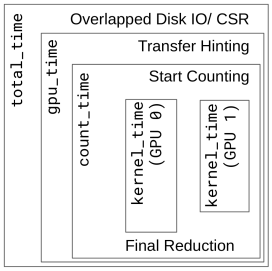

Figure 4. Diagram of what operations are included in measured `total_time`, `gpu_time`, `count_time`, and `kernel_time`. When multiple triangle kernels run in a multi-GPU execution, `kernel_time` is not reported.

G. Multi-GPU Approach

For the multi-GPU approach, we do a simple partition of the edges to each GPU. This causes substantial load imbalance, since our edge orientation approach causes low-index rows to be on average much longer than high-index rows. The CSR structures are marked with the `cudaMemAdviseReadMostly` hint (Section II-B to prevent thrashing when multiple GPUs access the same page (for example, if they read the same row).

III. RESULTS

A. Newell Evaluation Hardware

Triangle counting was evaluated on the same "Newell" platform [6] used in our previous submission. Table I summarizes the hardware and software environment of the evaluation. The systems comprise two triads, CPU0-GPU0-GPU1 and CPU1-GPU2-GPU3. Components within a triads are connected by the listed "triad interconnect," and the CPUs in each triad are connected by the "CPU-CPU interconnect."

B. Timing Methodology

Figure 4 summarizes what is included in the four reported times for each graph. In all cases, the clock is stopped once the final triangle count is available on the CPU. In *total_time*, the clock is started after the disk I/O is completed. In *gpu_time*, the clock is started after the CSR is constructed, before unified-memory hints are provided to the system. In *process_time*, the clock is started after unified memory hints have completed, but before counting operations begin. In *kernel_time*, only the actual triangle counting kernel is timed. The *process_time* is the time that includes all triangle counting operations,

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

but excludes all other operations. Table II summarizes the performance of our approach on all graphs.

C. Twitter Graph

We also evaluate performance on `twitter` [8], a follower-followee relationship graph from 2014. Since the twitter dataset is not provided by the Graph Challenge, we convert it to take a similar form.

D. Comparison with Past Submissions

1) Mailthody et al. (2018 Finalist): In our 2018 submission [9], we showed substantial performance degredation in some multi-GPU runs, up to several orders of magnitude in some cases. For the unified-memory case, this was due to unified memory thrashing. For the zero-copy case, zero-copy causes redundant host-to-device transfers because data is not cached in GPU memory. Through the use of unified-memory hints, we have brought the multi-GPU performance up to a more reasonable baseline, where performance is actually improved once the data is in the GPU. In addition, improvements to the sequential kernel code as well as the dynamic algorithmic selection have improved kernel performance.

2) Bisson & Fatica (2018 Champion): Due to differences in procedure, there is no good direct comparison between [10] (B&F) timing and ours. On the surface it seems natural to compare B&F's `total_t` with our `total_time`, but B&F suggest that CSR construction is done on the GPU, as total time starts after data has been copied "to device memory."

An ideal comparison would be between our `gpu_time` with their `process_t` plus matrix compaction, Such a comparison would ignore the differences in CSR construction time. B&F do not provide results cut that way. Since their kernel is typically substantially accelerated by their matrix processing, a direct kernel-only comparison may be interesting but not ultimately useful.

Finally, we designed our approach ultimately with scalability to large graphs in mind, and it contains no steps (other than the triangle counting) which require a full pass over the graph data. B&F require processing the entire graph for compaction, and again for triangle counting. If the graph does not fit in GPU memory, this approach may not be feasible. They reported requiring a GPU with 32GB instead of 16GB of memory in order to process the friendster graph.

3) Hu, Liu, & Huang (2018 Champion): Our `gpu_times` and `count_times` are substantially faster than [11]'s single-GPU times on the larger graphs where they report reults, though we evaluate on Nvidia V100s instead of P100s. For the large graphs, they do not report single GPU results, though their multi-GPU times are better than those achieved here.

4) Yacsar et al. (2018 Champion): Cilk gives [12] a work-stealing runtime, an idea analogous to the work-stealing done between warps in our dynamic kernel implementation. Many of their peak rates are similar to ours for larger graphs, though typically once a graph in the GPU memory our rates are much higher.

IV. Extra Contributions to the Community

Our graph dataset handling code is available at https://github.com/cwpearson/graph-datasets2.

Some analysis of the Graph Challenge graph datasets has been made available online at https://graphchallenge-datasets.netlify.com.

Our triangle counting code is in the Pangolin library, currently available at https://github.com/c3sr/pangolin

V. Conclusion

We present an update to our 2018 submission, motivated by understanding whether CUDA unified memory can be used as a foundation for processing large graphs on high-bandwidth heterogeneous systems. We introduce a work-stealing dynamic algorithm GPU kernel with persistent threads, to make performance adaptive for large graphs without requiring pre-counting graph analysis. We utilize CUDA unified memory to allow simultaneous access of large graph data on disk with construction of graph data structures in GPU memory. We utilize CUDA unified memory hints to tackle some performance challenges when scaling to multiple GPUs. Finally, we improve the single-GPU performance from our 2018 submission, and greatly improve the multi-GPU performance. Based on these results, we believe this approach is a solid foundation for scaling performance to large graphs. Moving forward, we intend to look at techniques to improve the multi-GPU load balancing, improving the dynamic algorithm kernels to reflect advances in kernel performance achieved by other teams, and extend our work to graphs that do not fit in GPU memory.

Acknowledgments

This work is supported by IBM-ILLINOIS Center for Cognitive Computing Systems Research (C3SR) - a research collaboration as part of the IBM AI Horizons Network. This work utilizes resources supported by NSF-MRI program, grant #1725729, and University of Illinois at Urbana-Champaign

References

[1] Y. Hu, P. Kumar, G. Swope, and H. H. Huang, "Trix: Triangle counting at extreme scale," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–7.

[2] R. Pearce, "Triangle counting for scale-free graphs at scale in distributed memory," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, Sep. 2017, pp. 1–4.

[3] J. Fox, O. Green, K. Gabert, X. An, and D. A. Bader, "Fast and adaptive list intersections on the gpu," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–7.

[4] CUB library. [Online]. Available: https://nvlabs.github.io/cub

[5] A. Yaar, S. Rajamanickam, M. Wolf, J. Berry, and . V. atalyrek, "Fast triangle counting using cilk," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*, Sep. 2018, pp. 1–7.

[6] A. B. Caldeira, "IBM power system AC922 introduction and technical overview," *IBM Redbooks*, 2018.

[7] (2017) NVIDIA Tesla V100 GPU architecture. Nvidia. [Online]. Available: http://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[8] H. Kwak, C. Lee, H. Park, and S. Moon, "What is Twitter, a social network or a news media?" in *WWW '10: Proceedings of the 19th international conference on World wide web*. New York, NY, USA: ACM, 2010, pp. 591–600.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

TABLE II

TRIANGLE COUNTING PERFORMANCE SUMMARY. TOTAL_TIME INCLUDES THE OVERLAPPED DISK I/O AND CSR CONSTRUCTION. GPU_TIME INCLUDES COST OF TRANSFERRING DATA TO THE GPU. COUNT_TIME THE COST OF ALL COUNTING OPERATIONS. KERNEL_TIME INCLUDES ONLY THE TRIANGLE COUNTING KERNEL TIME.

				Elapsed Time (s)				TEPS				2-GPU Speedup		4-GPU Speedup	
graph	**edges**	**nodes**	**tris**	kernel	count	gpu	total	kernel	count	gpu	total	count	gpu	count	gpu
friendster_adj	1799999986	119432958	191716	6.11E-01	6.14E-01	1.37E+00	5.72E+01	2.94E+09	2.93E+09	1.31E+09	3.15E+07	1.33	0.97	2.02	0.62
twitter	1202513046	41652230	34824916864	2.97E+01	2.97E+01	3.02E+01	7.60E+01	4.05E+07	4.05E+07	3.98E+07	1.58E+07	1.11	1.10	1.78	1.70
graph500-scale25-ef16_adj	523467448	17043781	21575375802	2.06E+01	2.06E+01	2.08E+01	3.67E+01	2.54E+07	2.54E+07	2.51E+07	1.43E+07	1.52	1.51	2.75	2.61
graph500-scale24-ef16_adj	260261843	8860451	9936161560	8.34E+00	8.34E+00	8.46E+00	1.64E+01	3.12E+07	3.12E+07	3.08E+07	1.59E+07	1.58	1.56	2.87	2.63
201512020330.v226196185_e480047894	240023945	226196186	26	1.26E-01	1.27E-01	2.72E-01	8.09E+00	1.91E+09	1.88E+09	8.83E+08	2.97E+07	1.03	0.88	1.76	0.56
V1r	232705452	214005018	49	9.86E-03	1.17E-02	1.55E-01	7.44E+00	2.36E+10	2.00E+10	1.50E+09	3.13E+07	1.62	0.84	1.33	0.38
A2a	180292586	170728176	3858	9.25E-03	1.06E-02	1.21E-01	6.04E+00	1.95E+10	1.70E+10	1.49E+09	2.98E+07	1.34	0.83	1.19	0.37
P1a	148914992	139353212	3412	7.85E-03	9.61E-03	1.02E-01	5.11E+00	1.90E+10	1.55E+10	1.46E+09	2.91E+07	1.36	0.84	1.76	0.39
201512020130.v128568730_e270234840	135117420	128568731	10	6.71E-02	6.85E-02	1.55E-01	4.86E+00	2.01E+09	1.97E+09	8.73E+08	2.78E+07	1.01	0.92	1.43	0.55
graph500-scale23-ef16_adj	129250705	4606315	4549133002	3.23E+00	3.23E+00	3.29E+00	7.29E+00	4.00E+07	4.00E+07	3.93E+07	1.77E+07	1.44	1.42	2.52	2.30
201512020030.v68863315_e143414960	71707480	68863316	6	4.11E-02	4.25E-02	8.83E-02	2.59E+00	1.74E+09	1.69E+09	8.12E+08	2.77E+07	0.99	0.88	1.54	0.59
U1a	69389281	67716232	325	3.66E-03	5.11E-03	4.97E-02	2.39E+00	1.89E+10	1.36E+10	1.40E+09	2.90E+07	1.15	0.87	0.73	0.38
graph500-scale22-ef16_adj	64097004	2393286	2067392370	1.15E+00	1.15E+00	1.18E+00	3.12E+00	5.59E+07	5.58E+07	5.45E+07	2.05E+07	1.49	1.46	2.58	2.25
V2a	58608800	55042370	1443	3.06E-03	4.08E-03	4.20E-02	1.90E+00	1.91E+10	1.43E+10	1.39E+09	3.09E+07	0.82	0.80	0.74	0.40
201512020000.v35991342_e74485420	37242710	35991343	2	2.10E-02	2.28E-02	4.75E-02	1.35E+00	1.77E+09	1.64E+09	7.84E+08	2.75E+07	0.97	0.87	1.30	0.57
graph500-scale21-ef16_adj	31731650	1243073	935100883	4.38E-01	4.40E-01	4.55E-01	1.58E+00	7.24E+07	7.21E+07	6.98E+07	2.00E+07	1.52	1.48	2.62	2.22
Theory-5-9-16-25-81-B1k	28667380	2174641	66758995	2.99E-01	3.01E-01	3.14E-01	1.24E+00	9.57E+07	9.52E+07	9.12E+07	2.31E+07	1.13	1.11	1.74	1.55
Theory-5-9-16-25-81-B2k	28667380	2174641	155	1.36E-02	1.51E-02	2.88E-02	9.46E-01	2.10E+09	1.90E+09	9.97E+08	3.03E+07	0.96	0.91	0.97	0.58
Theory-5-9-16-25-81-Bk	23328000	2174641	0	6.67E-04	2.39E-03	1.35E-02	7.64E-01	3.50E+10	9.78E+09	1.73E+09	3.05E+07	0.90	0.87	0.36	0.42
201512012345.v18571154_e38040320	19020160	18571155	2	1.19E-02	1.34E-02	2.71E-02	6.92E-01	1.60E+09	1.42E+09	7.01E+08	2.75E+07	1.00	0.88	1.02	0.58
cit-Patents_adj	16518947	3774769	7515023	4.41E-03	6.11E-03	1.51E-02	5.39E-01	3.74E+09	2.70E+09	1.09E+09	3.06E+07	1.13	0.97	0.88	0.54
graph500-scale20-ef16_adj	15680861	645821	419349784	1.69E-01	1.71E-01	1.78E-01	6.78E-01	9.28E+07	9.19E+07	8.80E+07	2.31E+07	1.41	1.36	2.37	1.96
Theory-3-4-5-9-16-25-B1k	11080030	530401	35882427	9.61E-02	9.79E-02	1.04E-01	4.69E-01	1.15E+08	1.13E+08	1.06E+08	2.36E+07	1.05	1.05	1.16	1.05
Theory-3-4-5-9-16-25-B2k	11080030	530401	651	6.99E-03	8.37E-03	1.49E-02	3.75E-01	1.59E+09	1.32E+09	7.41E+08	2.96E+07	0.87	0.91	0.73	0.58
graph500-scale19-ef16_adj	7729675	335319	186288972	6.52E-02	6.69E-02	7.15E-02	3.16E-01	1.18E+08	1.15E+08	1.08E+08	2.45E+07	1.40	1.36	2.13	1.71
Theory-3-4-5-9-16-25-Bk	6912000	530401	0	2.06E-04	1.90E-03	6.10E-03	2.28E-01	3.36E+10	3.64E+09	1.13E+09	3.03E+07	0.86	0.86	0.29	0.38
graph500-scale18-ef16_adj	3800348	174148	82287285	2.04E-02	2.21E-02	2.56E-02	1.48E-01	1.86E+08	1.72E+08	1.48E+08	2.57E+07	1.30	1.13	1.46	1.26
roadNet-CA_adj	2766607	1965207	120676	1.21E-04	1.73E-03	5.21E-03	9.55E-02	2.28E+10	1.60E+09	5.31E+08	2.90E+07	0.53	0.84	0.51	0.62
Theory-9-16-25-81-B1k	2606125	362441	4059175	1.28E-02	1.40E-02	1.70E-02	1.05E-01	2.04E+08	1.86E+08	1.53E+08	2.47E+07	0.86	0.87	0.71	0.68
Theory-9-16-25-81-B2k	2606125	362441	35	6.75E-04	2.22E-03	4.11E-03	9.34E-02	3.86E+09	1.17E+09	6.35E+08	2.79E+07	0.59	0.64	0.34	0.38
amazon0601_adj	2443408	403395	3986507	6.97E-04	2.26E-03	4.10E-03	8.79E-02	3.50E+09	1.08E+09	5.97E+08	2.78E+07	0.72	0.66	0.42	0.42
amazon0505_adj	2439437	410237	3951063	5.62E-04	2.14E-03	5.06E-03	8.84E-02	4.34E+09	1.14E+09	4.82E+08	2.76E+07	0.54	0.72	0.60	0.62
amazon0312_adj	2349869	400728	3686467	6.03E-04	1.78E-03	4.73E-03	8.63E-02	3.90E+09	1.32E+09	4.97E+08	2.72E+07	0.55	0.76	0.30	0.43
Theory-9-16-25-81-Bk	2332800	362441	0	8.96E-05	1.65E-03	4.15E-03	8.56E-02	2.60E+10	1.41E+09	5.62E+08	2.73E+07	0.46	0.60	0.25	0.38
flickrEdges_adj	2316948	105939	107987357	3.81E-03	4.77E-03	6.52E-03	8.66E-02	6.08E+08	4.86E+08	3.55E+08	2.67E+07	0.89	0.78	0.67	0.59
Theory-25-81-256-B1k	2132284	547925	2102761	7.93E-03	9.53E-03	1.21E-02	8.83E-02	2.69E+08	2.24E+08	1.76E+08	2.41E+07	0.91	0.89	0.78	0.70
Theory-25-81-256-B2k	2132284	547925	7	6.82E-04	2.26E-03	5.16E-03	8.14E-02	3.13E+09	9.42E+08	4.13E+08	2.62E+07	0.67	0.88	0.62	0.70
Theory-25-81-256-Bk	2073600	547925	0	7.73E-05	1.63E-03	4.43E-03	7.47E-02	2.68E+10	1.27E+09	4.68E+08	2.78E+07	0.93	1.06	0.48	0.59
roadNet-TX_adj	1921660	1379918	82869	9.43E-05	8.61E-04	2.68E-03	6.50E-02	2.04E+10	2.23E+09	7.18E+08	2.96E+07	0.28	0.42	0.25	0.29
Theory-4-5-9-16-25-B1k	1582861	132601	3548463	1.69E-02	1.78E-02	1.93E-02	7.33E-02	9.36E+07	8.88E+07	8.21E+07	2.16E+07	0.84	0.80	0.75	0.71
Theory-4-5-9-16-25-B2k	1582861	132601	155	1.73E-03	2.50E-03	4.10E-03	5.85E-02	9.16E+08	6.32E+08	3.86E+08	2.71E+07	0.45	0.55	0.31	0.36
roadNet-PA_adj	1541898	1088093	67150	7.39E-05	1.62E-03	3.33E-03	5.58E-02	2.09E+10	9.53E+08	4.62E+08	2.76E+07	0.66	0.66	0.49	0.46
Theory-4-5-9-16-25-Bk	1152000	132601	0	4.60E-05	1.58E-03	3.72E-03	4.50E-02	2.51E+10	7.29E+08	3.10E+08	2.56E+07	0.52	0.71	0.24	0.39
loc-gowalla_edges_adj	950327	196592	2273138	1.43E-03	2.32E-03	4.65E-03	3.87E-02	6.64E+08	4.09E+08	2.04E+08	2.46E+07	0.59	0.83	0.53	0.64
amazon0302_adj	899792	262112	717719	1.16E-04	1.66E-03	3.94E-03	3.60E-02	7.78E+09	5.43E+08	2.28E+08	2.50E+07	0.67	0.86	0.50	0.65
soc-Slashdot0902_adj	504230	82169	602592	6.61E-04	1.80E-03	2.59E-03	2.26E-02	7.63E+08	2.80E+08	1.94E+08	2.24E+07	0.87	0.65	0.52	0.45
soc-Slashdot0811_adj	469180	77361	551724	6.30E-04	1.75E-03	2.98E-03	2.24E-02	7.44E+08	2.69E+08	1.57E+08	2.10E+07	0.84	0.95	0.51	0.47
cit-HepPh_adj	420877	34547	1276868	2.58E-04	1.78E-03	3.59E-03	2.06E-02	1.63E+09	2.37E+08	1.17E+08	2.04E+07	0.52	0.80	0.55	0.65
soc-Epinions1_adj	405740	75880	1624481	7.09E-04	1.44E-03	3.20E-03	1.98E-02	5.72E+08	2.81E+08	1.27E+08	2.04E+07	0.44	0.69	0.42	0.57
email-EuAll_adj	364481	265215	267313	1.16E-03	2.24E-03	4.17E-03	1.95E-02	3.14E+08	1.63E+08	8.75E+07	1.86E+07	0.58	0.80	0.57	0.67
cit-HepTh_adj	352285	27771	1478735	4.77E-04	1.95E-03	3.62E-03	1.87E-02	7.38E+08	1.81E+08	9.72E+07	1.88E+07	1.05	0.92	0.56	0.65
Theory-256-625-B1k	320881	160883	160000	9.48E-04	2.46E-03	3.69E-03	1.91E-02	3.38E+08	1.30E+08	8.68E+07	1.68E+07	0.67	0.69	0.65	0.61
Theory-256-625-B2k	320881	160883	1	1.56E-04	1.25E-03	2.74E-03	1.78E-02	2.06E+09	2.56E+08	1.17E+08	1.81E+07	0.69	0.69	0.40	0.51
Theory-256-625-Bk	320000	160883	0	2.34E-05	1.53E-03	2.03E-03	1.63E-02	1.37E+10	2.09E+08	1.57E+08	1.96E+07	0.43	0.44	0.48	0.37
Theory-3-4-5-9-16-B1k	217255	20401	465427	2.44E-03	3.95E-03	4.73E-03	1.52E-02	8.89E+07	5.50E+07	4.60E+07	1.43E+07	0.94	0.78	0.69	0.61
Theory-3-4-5-9-16-B2k	217255	20401	155	3.09E-04	1.18E-03	2.38E-03	1.31E-02	7.03E+08	1.84E+08	9.12E+07	1.66E+07	0.42	0.64	0.35	0.44
loc-brightkite_edges_adj	214078	58229	494728	2.37E-04	1.76E-03	2.88E-03	1.31E-02	9.03E+08	1.22E+08	7.43E+07	1.63E+07	0.58	0.57	0.54	0.53
email-Enron_adj	183831	36693	727044	3.64E-04	1.89E-03	3.05E-03	1.25E-02	5.05E+08	9.71E+07	6.02E+07	1.47E+07	0.88	0.99	0.58	0.56
Theory-5-9-16-25-B1k	175873	26521	264799	1.32E-03	2.85E-03	3.22E-03	1.26E-02	1.34E+08	6.17E+07	5.46E+07	1.39E+07	1.05	0.68	0.65	0.52
Theory-5-9-16-25-B2k	175873	26521	35	9.99E-05	1.17E-03	2.31E-03	1.15E-02	1.76E+09	1.51E+08	7.61E+07	1.53E+07	0.43	0.57	0.38	0.44
p2p-Gnutella31_adj	147892	62587	2024	3.99E-05	1.07E-03	2.26E-03	1.01E-02	3.71E+09	1.38E+08	6.54E+07	1.46E+07	0.76	0.96	0.33	0.42
Theory-5-9-16-25-Bk	144000	26521	0	1.72E-05	1.52E-03	3.39E-03	1.17E-02	8.40E+09	9.47E+07	4.25E+07	1.23E+07	1.02	1.28	0.48	0.69
Theory-3-4-5-9-16-Bk	138240	20401	0	1.13E-05	6.83E-04	1.77E-03	9.73E-03	1.23E+10	2.02E+08	7.80E+07	1.42E+07	0.34	0.56	0.22	0.35
Theory-16-25-81-B1k	137164	36245	133321	1.21E-03	2.74E-03	3.11E-03	1.15E-02	1.13E+08	5.00E+07	4.41E+07	1.19E+07	1.04	0.82	0.66	0.49
Theory-16-25-81-B2k	137164	36245	7	2.59E-04	9.45E-04	2.83E-03	1.12E-02	5.29E+08	1.45E+08	4.85E+07	1.22E+07	0.31	0.73	0.29	0.54
Theory-16-25-81-Bk	129600	36245	0	1.30E-05	1.15E-03	2.96E-03	1.07E-02	9.95E+09	1.13E+08	4.38E+07	1.21E+07	0.48	0.93	0.38	0.60
ca-HepPh_adj	118489	12009	3358499	1.58E-04	1.25E-03	2.50E-03	9.57E-03	7.50E+08	9.46E+07	4.74E+07	1.24E+07	0.68	0.71	0.40	0.49
ca-CondMat_adj	93439	23134	173361	6.71E-05	7.52E-04	2.24E-03	8.58E-03	1.39E+09	1.24E+08	4.17E+07	1.09E+07	0.55	0.86	0.24	0.41
p2p-Gnutella30_adj	88328	36683	1590	2.55E-05	1.11E-03	2.61E-03	8.76E-03	3.46E+09	7.97E+07	3.39E+07	1.01E+07	0.41	0.69	0.36	0.53
facebook_combined_adj	88234	4040	1612010	2.02E-04	1.72E-03	2.85E-03	9.16E-03	4.37E+08	5.14E+07	3.10E+07	9.64E+06	0.96	0.85	0.55	0.59
p2p-Gnutella24_adj	65369	26519	986	3.58E-05	1.55E-03	1.85E-03	7.01E-03	1.83E+09	4.22E+07	3.54E+07	9.33E+06	0.90	0.73	0.48	0.42
p2p-Gnutella25_adj	54705	22688	806	2.84E-05	9.36E-04	1.24E-03	6.25E-03	1.93E+09	5.84E+07	4.40E+07	8.75E+06	0.44	0.42	0.31	0.26
Theory-81-256-B1k	41809	21075	20736	1.38E-04	1.65E-03	2.70E-03	7.68E-03	3.03E+08	2.54E+07	1.55E+07	5.45E+06	0.98	1.09	0.53	0.56
Theory-81-256-B2k	41809	21075	1	5.17E-05	1.09E-03	1.70E-03	6.42E-03	8.08E+08	3.84E+07	2.47E+07	6.51E+06	0.44	0.51	0.35	0.39
Theory-81-256-Bk	41472	21075	0	8.96E-06	6.91E-04	2.47E-03	7.42E-03	4.63E+09	6.00E+07	1.68E+07	5.59E+06	0.47	1.06	0.22	0.49
p2p-Gnutella04_adj	34305	10877	738	4.03E-05	1.56E-03	2.60E-03	7.38E-03	8.51E+08	2.19E+07	1.32E+07	4.65E+06	0.54	0.70	0.48	0.50
oregon2_010526_adj	32730	11462	89541	2.02E-04	8.92E-04	2.35E-03	6.18E-03	1.62E+08	3.67E+07	1.39E+07	5.30E+06	0.24	0.45	0.28	0.59
oregon2_010519_adj	32287	11376	83709	2.01E-04	8.96E-04	1.32E-03	5.16E-03	1.61E+08	3.60E+07	2.45E+07	6.26E+06	0.53	0.63	0.28	0.25
p2p-Gnutella05_adj	31839	8847	1112	3.85E-05	7.59E-04	1.81E-03	6.34E-03	8.27E+08	4.19E+07	1.76E+07	5.02E+06	0.43	0.60	0.24	0.43
oregon2_010414_adj	31761	11020	88905	1.31E-04	1.65E-03	3.08E-03	7.06E-03	2.43E+08	1.92E+07	1.03E+07	4.50E+06	0.91	1.01	0.52	0.61
oregon2_010421_adj	31538	11081	82129	1.92E-04	1.29E-03	2.64E-03	6.43E-03	1.64E+08	2.45E+07	1.20E+07	4.90E+06	0.50	0.72	0.40	0.51
p2p-Gnutella06_adj	31525	8718	1142	3.53E-05	1.55E-03	3.09E-03	7.06E-03	8.92E+08	2.03E+07	1.02E+07	4.47E+06	0.72	1.04	0.47	0.70
oregon2_010428_adj	31434	11114	78000	1.95E-04	1.72E-03	2.74E-03	6.53E-03	1.61E+08	1.83E+07	1.15E+07	4.81E+06	0.52	0.59	0.51	0.60
oregon2_010512_adj	31303	11261	72866	1.91E-04	8.81E-04	1.92E-03	6.17E-03	1.64E+08	3.55E+07	1.63E+07	5.07E+06	0.58	0.77	0.27	0.39
oregon2_010331_adj	31180	10901	82856	1.90E-04	8.82E-04	2.04E-03	5.82E-03	1.64E+08	3.54E+07	1.53E+07	5.35E+06	0.29	0.43	0.28	0.41
Theory-4-5-9-16-B1k	31036	5101	45013	2.15E-04	1.33E-03	2.86E-03	7.09E-03	1.44E+08	2.34E+07	1.09E+07	4.38E+06	0.70	1.11	0.41	0.54
Theory-4-5-9-16-B2k	31036	5101	35	1.03E-04	1.63E-03	2.37E-03	6.65E-03	3.01E+08	1.90E+07	1.31E+07	4.67E+06	0.56	0.62	0.51	0.51
oregon2_010505_adj	30943	11158	72182	1.52E-04	8.45E-04	2.59E-03	6.28E-03	2.03E+08	3.66E+07	1.19E+07	4.93E+06	0.29	0.70	0.27	0.52
oregon2_010407_adj	30855	10982	78138	1.86E-04	1.29E-03	2.82E-03	7.35E-03	1.66E+08	2.39E+07	1.09E+07	4.20E+06	0.42	0.68	0.41	0.67
p2p-Gnutella09_adj	26013	8115	2354	3.89E-05	7.81E-04	2.19E-03	5.99E-03	6.68E+08	3.33E+07	1.19E+07	4.34E+06	0.28	0.57	0.25	0.50
ca-HepTh_adj	25973	9878	28339	3.01E-05	1.11E-03	2.31E-03	6.54E-03	8.62E+08	2.33E+07	1.12E+07	3.97E+06	0.51	0.77	0.35	0.46
oregon1_010526_adj	23409	11175	19894	1.36E-04	1.55E-03	3.03E-03	6.65E-03	1.72E+08	1.51E+07	7.73E+06	3.52E+06	0.62	0.78	0.50	0.64
Theory-4-5-9-16-Bk	23040	5101	0	1.46E-05	1.79E-03	2.95E-03	7.00E-03	1.58E+09	1.29E+07	7.80E+06	3.29E+06	1.03	1.06	0.58	0.61
oregon1_010421_adj	22747	10860	19108	1.03E-04	1.18E-03	1.81E-03	5.74E-03	2.22E+08	1.93E+07	1.26E+07	3.96E+06	0.42	0.46	0.38	0.35
oregon1_010519_adj	22724	11052	17677	1.31E-04	1.65E-03	2.40E-03	5.91E-03	1.74E+08	1.38E+07	9.47E+06	3.85E+06	1.13	0.75	0.54	0.48
oregon1_010512_adj	22677	11012	17598	1.25E-04	1.64E-03	2.02E-03	5.54E-03	1.81E+08	1.38E+07	1.13E+07	4.09E+06	0.92	0.83	0.53	0.39
oregon1_010505_adj	22607	10944	17597	1.35E-04	1.29E-03	2.02E-03	5.63E-03	1.67E+08	1.75E+07	1.12E+07	4.02E+06	0.35	0.41	0.40	0.46
oregon1_010428_adj	22493	10887	17645	1.11E-04	7.94E-04	1.48E-03	5.01E-03	2.03E+08	2.83E+07	1.52E+07	4.49E+06	0.44	0.45	0.25	0.34
oregon1_010414_adj	22469	10791	18237	1.15E-04	1.63E-03	2.27E-03	5.87E-03	1.96E+08	1.38E+07	9.88E+06	3.83E+06	0.57	0.49	0.50	0.46
oregon1_010331_adj	22002	10671	17144	1.10E-04	1.63E-03	2.64E-03	6.24E-03	1.99E+08	1.35E+07	8.33E+06	3.52E+06	0.96	0.96	0.51	0.50
oregon1_010407_adj	21999	10730	15834	1.08E-04	1.64E-03	2.73E-03	6.28E-03	2.03E+08	1.34E+07	8.05E+06	3.50E+06	0.56	0.67	0.50	0.55
p2p-Gnutella08_adj	20777	6302	2383	3.92E-05	1.14E-03	1.58E-03	5.30E-03	5.30E+08	1.83E+07	1.31E+07	3.92E+06	0.49	0.43	0.36	0.32
Theory-9-16-25-B1k	15988	4421	15169	2.05E-04	8.95E-04	1.21E-03	4.50E-03	7.78E+07	1.79E+07	1.32E+07	3.55E+06	0.31	0.27	0.28	0.26
Theory-9-16-25-B2k	15988	4421	7	6.53E-05	1.58E-03	2.23E-03	5.48E-03	2.45E+08	1.01E+07	7.18E+06	2.92E+06	0.67	0.56	0.48	0.41
ca-GrQc_adj	14484	5243	48260	3.38E-05	1.55E-03	2.96E-03	6.68E-03	4.28E+08	9.33E+06	4.90E+06	2.17E+06	0.56	0.75	0.48	0.58
Theory-9-16-25-Bk	14400	4421	0	9.34E-06	7.44E-04	2.06E-03	5.74E-03	1.54E+09	1.94E+07	7.01E+06	2.51E+06	0.45	0.77	0.24	0.41
as20000102_adj	12572	6475	6584	2.04E-04	1.77E-03	2.76E-03	5.75E-03	6.15E+07	7.10E+06	4.55E+06	2.19E+06	0.70	0.68	0.58	0.58
Theory-3-4-5-9-B1k	6583	1201	9107	1.50E-04	8.87E-04	1.20E-03	4.10E-03	4.39E+07	7.42E+06	5.48E+06	1.61E+06	0.50	0.49	0.26	0.24
Theory-3-4-5-9-B2k	6583	1201	35	3.25E-05	7.36E-04	1.72E-03	4.34E-03	2.02E+08	8.95E+06	3.84E+06	1.52E+06	0.29	0.48	0.23	0.33
Theory-3-4-5-9-Bk	4320	1201	0	1.33E-05	1.52E-03	2.82E-03	5.35E-03	3.24E+08	2.84E+06	1.53E+06	8.07E+05	0.55	0.74	0.49	0.67
Theory-25-81-B1k	4156	2133	2025	5.14E-05	7.71E-04	2.21E-03	4.88E-03	8.09E+07	5.39E+06	1.88E+06	8.51E+05	0.21	0.47	0.25	0.45
Theory-25-81-B2k	4156	2133	1	2.40E-05	1.54E-03	3.07E-03	6.10E-03	1.73E+08	2.69E+06	1.35E+06	6.81E+05	0.89	1.13	0.49	0.64
Theory-25-81-Bk	4050	2133	0	1.35E-05	1.53E-03	2.64E-03	5.32E-03	3.00E+08	2.65E+06	1.53E+06	7.62E+05	0.62	0.67	0.47	0.53
Theory-5-9-16-B1k	3448	1021	3149	5.48E-05	7.79E-04	2.15E-03	4.35E-03	6.30E+07	4.43E+06	1.60E+06	7.93E+05	0.42	0.59	0.25	0.44
Theory-5-9-16-B2k	3448	1021	7	3.05E-05	7.14E-04	2.14E-03	4.42E-03	1.13E+08	4.83E+06	1.61E+06	7.81E+05	0.44	0.79	0.22	0.43
Theory-5-9-16-Bk	2880	1021	0	1.27E-05	1.52E-03	2.15E-03	4.37E-03	2.27E+08	1.90E+06	1.34E+06	6.60E+05	1.03	0.74	0.49	0.52
Theory-4-5-9-B1k	940	301	821	4.74E-05	1.56E-03	3.21E-03	4.87E-03	1.98E+07	6.02E+05	2.93E+05	1.93E+05	0.68	1.10	0.51	0.79
Theory-4-5-9-B2k	940	301	7	1.93E-05	7.48E-04	1.68E-03	3.96E-03	4.87E+07	1.26E+06	5.59E+05	2.38E+05	0.43	0.60	0.24	0.43
Theory-16-25-B1k	841	443	400	3.53E-05	7.16E-04	1.24E-03	3.01E-03	2.38E+07	1.17E+06	6.77E+05	2.80E+05	0.42	0.37	0.23	0.27
Theory-16-25-B2k	841	443	1	1.42E-05	7.05E-04	9.09E-04	2.81E-03	5.93E+07	1.19E+06	9.25E+05	3.00E+05	0.41	0.38	0.22	0.21
Theory-16-25-Bk	800	443	0	8.83E-06	1.20E-03	1.61E-03	3.45E-03	9.06E+07	6.66E+05	4.98E+05	2.32E+05	0.74	0.57	0.38	0.39
Theory-4-5-9-Bk	720	301	0	8.42E-06	1.02E-03	2.00E-03	4.24E-03	8.56E+07	7.03E+05	3.59E+05	1.70E+05	0.76	0.65	0.34	0.42
Theory-3-4-5-B1k	346	121	287	2.99E-05	6.28E-04	1.68E-03	3.56E-03	1.16E+07	5.51E+05	2.06E+05	9.72E+04	0.20	0.38	0.20	0.40
Theory-3-4-5-B2k	346	121	7	1.63E-05	1.90E-03	2.97E-03	4.80E-03	2.13E+07	1.82E+05	1.16E+05	7.20E+04	1.18	1.30	0.60	0.71
Theory-9-16-B1k	313	171	144	3.03E-05	8.36E-04	2.58E-03	4.43E-03	1.03E+07	3.74E+05	1.21E+05	7.06E+04	0.26	0.64	0.27	0.62
Theory-9-16-B2k	313	171	1	1.68E-05	7.24E-04	1.43E-03	3.24E-03	1.86E+07	4.32E+05	2.19E+05	9.66E+04	0.51	0.49	0.19	0.27
Theory-9-16-Bk	288	171	0	1.36E-05	1.49E-03	1.81E-03	3.32E-03	2.12E+07	1.93E+05	1.59E+05	8.67E+04	0.54	0.42	0.40	0.34
Theory-3-4-5-Bk	240	121	0	1.26E-05	6.12E-04	2.03E-03	3.53E-03	1.90E+07	3.92E+05	1.18E+05	6.80E+04	0.22	0.46	0.18	0.39
Theory-5-9-B1k	104	61	45	1.98E-05	6.13E-04	1.54E-03	2.83E-03	5.24E+06	1.70E+05	6.74E+04	3.67E+04	0.45	0.75	0.18	0.36
Theory-5-9-B2k	104	61	1	1.37E-05	1.01E-03	1.70E-03	3.74E-03	7.61E+06	1.03E+05	6.10E+04	2.78E+04	0.49	0.54	0.31	0.35
Theory-5-9-Bk	90	61	0	1.28E-05	6.10E-04	1.53E-03	2.74E-03	7.01E+06	1.48E+05	5.87E+04	3.28E+04	0.37	0.65	0.18	0.36
Theory-4-5-B1k	49	31	20	1.53E-05	6.18E-04	1.31E-03	2.83E-03	3.20E+06	7.94E+04	3.73E+04	1.73E+04	0.44	0.46	0.16	0.28
Theory-4-5-B2k	49	31	1	1.40E-05	1.49E-03	2.90E-03	4.39E-03	3.51E+06	3.29E+04	1.69E+04	1.12E+04	0.70	1.03	0.45	0.69
Theory-4-5-Bk	40	31	0	1.38E-05	1.50E-03	2.03E-03	3.12E-03	2.91E+06	2.67E+04	1.97E+04	1.28E+04	1.05	0.86	0.44	0.44
Theory-3-4-B1k	31	21	12	1.35E-05	6.12E-04	1.86E-03	3.00E-03	2.29E+06	5.07E+04	1.66E+04	1.03E+04	0.44	0.74	0.18	0.38
Theory-3-4-B2k	31	21	1	1.31E-05	6.08E-04	7.67E-04	2.08E-03	2.37E+06	5.10E+04	4.04E+04	1.49E+04	0.30	0.28	0.17	0.16
Theory-3-4-Bk	24	21	0	1.29E-05	1.51E-03	2.84E-03	4.24E-03	1.86E+06	1.58E+04	8.45E+03	5.67E+03	0.74	0.94	0.46	0.58

[9] V. S. Mailthody, K. Date, Z. Qureshi, C. Pearson, R. Nagi, J. Xiong, and W.-m. Hwu, "Collaborative (cpu+ gpu) algorithms for triangle counting and truss decomposition," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–7.
[10] M. Bisson and M. Fatica, "Update on static graph challenge on gpu," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–8.
[11] Y. Hu, H. Liu, and H. H. Huang, "High-performance triangle counting on gpus," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–5.
[12] A. Yaşar, S. Rajamanickam, M. Wolf, J. Berry, and Ü. V. Çatalyürek, "Fast triangle counting using cilk," in *2018 IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–7.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE