Linear Algebra-Based Triangle Counting via Fine-Grained Tasking on Heterogeneous Environments

(Update on Static Graph Challenge)

Abdurrahman Yaşar†, Sivasankaran Rajamanickam*, Jonathan Berry*,
Michael Wolf*, Jeffrey S. Young†, and Ümit V. Çatalyürek†
ayasar@gatech.edu, srajama@sandia.gov, jberry@sandia.gov,
mmwolf@sandia.gov, jyoung9@gatech.edu, and umit@gatech.edu
*Center for Computing Research, Sandia National Laboratories, Albuquerque, NM, U.S.A.
†College of Computing, Georgia Institute of Technology, Atlanta, GA, U.S.A.

***Abstract*—Triangle counting is a representative graph problem that shows the challenges of improving graph algorithm performance using algorithmic techniques and adopting graph algorithms to new architectures. In this paper, we describe an update to the linear-algebraic formulation of the triangle counting problem. Our new approach relies on fine-grained tasking based on a tile layout. We adopt this task based algorithm to heterogeneous architectures (CPUs and GPUs) for up to 10.8x speed up over past year's graph challenge submission. This implementation also results in the fastest kernel time known at time of publication for real-world graphs like twitter (3.7 second) and friendster (1.8 seconds) on GPU accelerators when the graph is GPU resident. This is a 1.7 and 1.2 time improvement over previous state-of-the-art triangle counting on GPUs. We also improved end-to-end execution time by overlapping computation and communication of the graph to the GPUs. In terms of end-to-end execution time, our implementation also achieves the fastest end-to-end times due to very low overhead costs.**

I. Introduction

With increased use of accelerators for achieving better performance, proposing algorithms that require ideally no architecture specific changes on code base is crucial for portability on heterogeneous environments. This paper addresses these two primary problems, using general purpose accelerators for the graph challenge and doing that in a portable manner.

In this paper, we focus on a triangle counting algorithm that utilizes both CPUs and GPUs on a compute node and uses a portable tiled layout that requires almost no (algorithmic or layout) change between different architectures. This paper improves our previous work [1], [2] by using the linear algebra based triangle counting algorithm on a tiled layout and exploiting multiple levels of shared-memory parallelism on CPUs and GPUs.

This algorithmic changes allow us to achieve the fastest times for real world graphs like twitter and friendster compared to past champions utilizing the GPUs. We achieve 3.7 seconds on twitter and 1.8 seconds on friendster graphs as opposed to 6.5 and 2.1 seconds by past champions [3] when the graph is GPU resident. However, we believe assuming the graph is on the GPU is not realistic for several use cases, therefore in this paper we also focus on an end-to-end time metric when the graph is not GPU resident. In this case, we are able to count triangles in twitter and friendster graphs in 4.6 and 3.1 seconds where data copy is overlapped with computation. In the following sections we only report end to end results. In this paper, we propose a new **linear algebraic formulation** for triangle counting problem that uses tiles and a **fine-grained parallel algorithm** that exploits multiple level of parallelism on different architectures. We implement a highly efficient **multi-core**, **multi-GPU** hybrid framework that outperforms state-of-the-art. Experimental results demonstrate that our codes achieve the fastest kernel times on real-world graphs when the graph resides on the GPU and the fastest end-to-end times when the graph is not on the GPU. The performance improvement is up to $10\times$ over our previous state-of-the-art implementation.

II. Background

A. 2017 Static Graph Challenge

We used a linear algebra-based triangle counting implementation, KKTri (previously designated TCKK) [1] in the the 2017 Static Graph Challenge [4]. That work, focused on efficient shared memory parallelism on top of a portable SpGEMM (called KK-MEM) [5] in the Kokkos Kernels library [6]. The primary focus of that work was on two linear-algebra based formulations of triangle counting:

1) $D = (L \times U). * L$: This formulation represented triangle counting in terms of sparse matrix-matrix multiplication followed by an element-wise matrix multiplication where L and U are the lower and upper triangle parts of the adjacency matrix for the graph.
2) $D = (L \times L). * L$: This formulation was used primarily for the 2017 Graph Challenge. This formulation follows the same logic as the previous method.

Three optimizations were used to achieve good performance: (1) in-place masked SpGEMM which reduced the memory needed for triangle counting; (2) data compression on the right hand side matrix that allowed using efficient bitwise operations (3) ordering of the vertices a common heuristic to reduce number of operations.

B. 2018 Static Graph Challenge

For the 2018 Static Graph Challenge [7], we designed a linear algebra-based triangle counting implementation KKTri-Cilk that inherited from the KK-SpGEMM algorithm [8] and

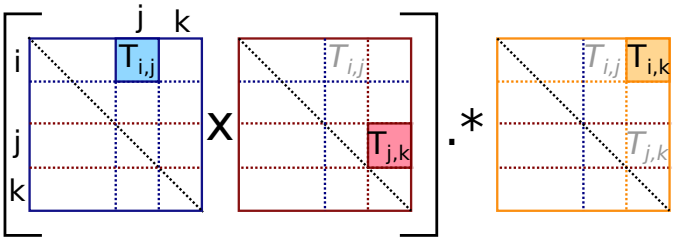

Fig. 1. Tiled Triangle Counting

improved load-balancing and efficient hyper-thread usage issues using Cilk based programming model and optimizations.

The parallelization strategy and the runtime system is the main difference between KKTri-Cilk and KKTri. KKTri used a very simple scheme, partitioning the matrix evenly into partitions of a fixed number of rows. To balance the work among the tasks, KKTri-Cilk uses an heuristic to find the partitions, creating partitions such that the number of non-zeros within each partition are approximately equal.

Compression of the right hand side matrix can decrease the problem size significantly, and allow using efficient bitwise operations. However, compression is not always successful because of the natural order of the matrices.

III. Approach

A lightweight 2D partitioning algorithm is implemented to create tiles. This algorithm partitions the graph in two dimensional space where diagonal tiles are required to be squares. This partitioning is known as symmetric generalized block distribution [9].

In Equation 1, we propose a new linear algebra based triangle counting formulation that uses tiles. Similar to LL and LU this formulation represents triangle counting in terms of sparse matrix-matrix multiplication followed by an element-wise matrix multiplication but with tiles. If number of tiles is one then this formulation is identical with LL Algorithm [1]. Figure 1 illustrates this formulation. In the context of this paper, a "task", t, counts the triangles in a given triple of tiles, $t = \{T_{i,j}, T_{j,k}, T_{i,k}\}$. However, counting triangles in all possible triple of tiles, ends up counting each triangle three times. Considering tasks, $t = \{T_{i,j}, T_{j,k}, T_{i,k}\}$ where $i \leq j \leq k$ avoids unnecessary counting. With this restriction, if a given graph partitioned into $p \times p$ tiles, then number of tasks, N_{task}, is defined as $N_{task} = \frac{p \times (p+1) \times (p+2)}{6}$. Tile based triangle counting can be formulated as:

$$D_{i,j,k} = (T_{i,j} \times T_{j,k}) . * T_{i,k} \quad (1)$$

Latapy *et al.* [10] proposes to use list intersection based counting for small degree vertices and a hashmap based intersection for the other. In this work, a similar approach is used; i.e. any task with sparse tiles will use list based intersection and denser tiles will use a dense hashmap accumulator.

We use both CPUs and GPUs to process tasks together. There is no architecture specific algorithmic change in the code-base. This will allow execution on any accelerator and CPU combination. However, because of architectural differences, the parallelization approach differs between CPU and GPU itself. While, CPU threads execute different tasks in parallel, GPU threads execute cooperatively to complete the same task in parallel. Assigning heavier tasks to GPUs is one of the primary optimizations that is being used in hybrid approaches [11] due to massive parallelism capabilities of these devices. Hence, we try to estimate and execute heavier tasks on GPUs.

We use Cuda streams to simultaneously execute several tasks on the GPUs. Four Cuda streams are created for each GPU in the node. Then, a CPU thread is created and made responsible for the operations on the stream. GPUs and CPUs compete for tasks and get a new one from a queue when they are ready. Tasks are ordered based on their size in a task queue. GPUs start to process tasks starting from the heaviest task and CPUs start to process tasks from the lightest tasks. This continues until all tasks have been executed.

TABLE I
Overview of the Architectures. Pinned: transfers using pinned memory. Page-able: transfers using page-able memory. H2D: host to device. D2H: device to host.

		DGX	Newell	Minsky
CPU		Intel, E5-2698	POWER9	POWER8-NVL
Cores		2×40	2×16	2×8
Host Memory		512 GB	320 GB	512 GB
L2-Cache		256 KB	512 KB	512 KB
L3-Cache		50 MB	10 MB	8 MB
GPU		V100-SXM2	V100-SXM2	P100-SXM2
GPU Memory		32 GB	32 GB	16 GB
Number of GPUs		8	2	4
Pageable	H2D	9.2 GB/S	12.2 GB/S	15.1 GB/S
	D2H	8.0 GB/S	14.2 GB/S	8.9 GB/S
Pinned	H2D	10.7 GB/S	60.0 GB/S	31.4 GB/S
	D2H	12.1 GB/S	60.0 GB/S	32.6 GB/S

IV. Experimental Evaluation

We present several experiments to identify the performance trade-offs of the proposed work. These experiments were carried out on three architectures with multicore processors and GPUs that are shown in Table I. GNU compiler (g++) version 7.2, Cuda runtime version 10.0 and OpenMP version 4.0 are used to compile and run the code on all the architectures.

A. Dataset and Copy Time Included Peak Rates

Table II lists 23 graphs that we used in our experiments along with the number of vertices ($|V|$), number of edges ($|E|$), number of triangles ($|T|$), size of the graph in memory (Raw Size), number of tiles and number of tasks in the graph. Note that when we partition the adjacency matrix into p intervals on rows and on columns in total $p \times p$ tiles are created which is reported in Table II. However, our algorithm only uses upper triangular part of the matrix, therefore, only $\frac{p \times (p+1)}{2}$ tiles are being used. In addition to 20 Challenge graphs for which triangle counting is particularly costly, 3 additional large real-world graphs [12], [13] are included in our experiments. We used the Graph Challenge procedure of symmetrizing the matrices (using undirected graphs). For all experiments, we

TABLE II

PROPERTIES OF THE DATASET. BEST OF THE MEDIANS OF THE COPY TIME INCLUDED EXECUTION TIMES IN SECONDS AND CORRESPONDING RATES ARE REPORTED. GREEN - INSTANCES THAT ARE MORE THAN 5× FASTER THAN LAST YEAR SUBMISSION. BLUE - THE FASTEST RATE FOR A GRAPH.

Data Set	$\|V\|$	$\|E\|$	$\|T\|$	Raw Size	Tiles	Tasks	Best Time (s)	Rates ($\times 10^8$) DGX	Newell	Minsky
cit-HepTh	27,770	352,285	1,478,735	2.2 MB	64	120	0.002	1.7	0.9	0.9
email-EuAll	265,214	364,481	267,313	9.5 MB	64	120	0.002	1.9	1.1	0.9
soc-Epinions1	75,879	405,740	1,624,481	3.9 MB	64	120	0.002	2.1	0.9	1.0
cit-HepPh	34,546	420,877	1,276,868	2.7 MB	64	120	0.002	2.0	1.6	1.1
soc-Slashdot0811	77,360	469,180	551,724	5.4 MB	144	364	0.002	2.9	0.9	1.3
soc-Slashdot0902	82,168	504,230	602,592	5.7 MB	144	364	0.002	3.1	1.3	1.2
flickrEdges	105,938	2,316,948	107,987,357	14 MB	144	364	0.016	1.5	1.3	1.1
amazon0312	400,727	2,349,869	3,686,467	28 MB	144	364	0.006	3.9	3.4	3.1
amazon0505	410,236	2,439,437	3,951,063	29 MB	144	364	0.007	3.7	3.4	2.8
amazon0601	403,394	2,443,408	3,986,507	28 MB	144	364	0.006	4.4	4.7	3.3
scale18	174,147	3,800,348	82,287,285	26 MB	256	816	0.021	1.8	1.4	1.4
scale19	335,318	7,729,675	186,288,972	56 MB	400	1540	0.041	1.9	1.5	1.4
as-Skitter	1,696,415	11,095,298	28,769,868	146 MB	400	1540	0.027	4.1	3.4	3.2
scale20	645,820	15,680,861	419,349,784	110 MB	400	1540	0.079	2.0	1.7	1.5
cit-Patents	3,774,768	16,518,947	7,515,023	352 MB	400	1540	0.038	4.4	3.6	3.4
scale21	1,243,072	31,731,650	935,100,883	216 MB	400	1540	0.144	2.2	1.8	1.7
soc-LiveJournal1	4,847,571	42,851,237	285,730,264	534 MB	400	1540	0.121	3.6	3.3	2.7
scale22	2,393,285	64,097,004	2,067,392,370	464 MB	576	2600	0.325	2.0	1.7	1.5
scale23	4,606,314	129,250,705	4,549,133,002	915 MB	576	2600	0.549	2.4	1.5	1.1
scale24	8,860,450	260,261,843	9,936,161,560	1.9 GB	784	4060	1.154	2.3	1.2	0.8
scale25	17,043,780	523,467,448	21,575,375,802	4.0 GB	1024	5984	2.400	2.2	0.9	0.6
twitter	61,578,414	1,202,513,046	34,824,916,864	13 GB	1296	8436	4.582	2.6	1.1	1.0
friendster	65,608,366	1,806,067,135	4,173,724,142	16 GB	1296	8436	3.133	5.8	2.6	2.2
							Geomean:	**2.6**	**1.7**	**1.5**

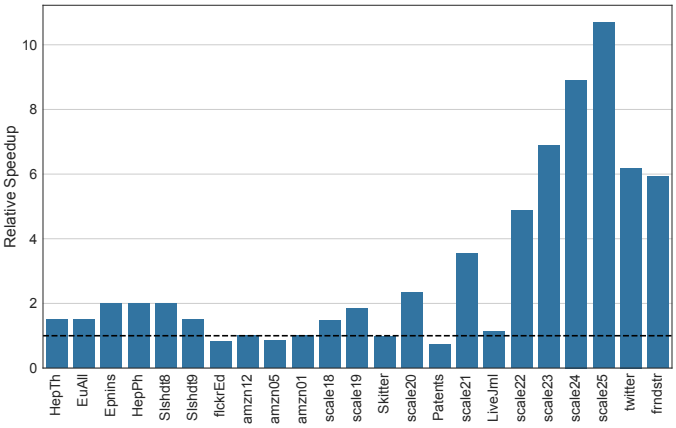

Fig. 2. Relative speedup between this work (on DGX architecture) and last year's submission. Black dashed line represents the baseline. Graphs are sorted on x-axis based on their number of edges.

report the best of the median time of five runs with different number of GPUs. Table II reports copy included best execution time and rates on different architectures for each graph. This work outperforms last year's submission by 6× on friendster graph and achieves 5.6×10^8 rate.

B. Relative Speedup comparisons to Last Year's Submission

Figure 2 presents relative speedup between this work and our last submission (KKTri-Cilk) [2]. This work outperforms KKTri-Cilk in 20 of 23 cases. In three small instances (flickrEdges, amazon0505 and cit-Patents) KKTri-Cilk performs better. This years work can achieve up to 11× speedup on large graphs in which GPUs become more useful.

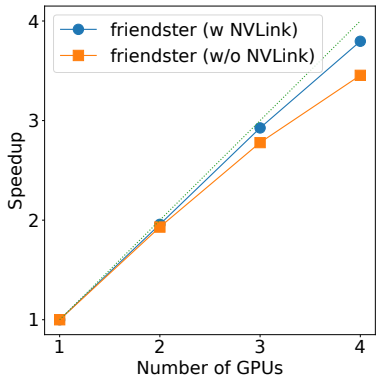

Fig. 3. Effect of bandwidth

C. Effect of bandwidth on speedup

Figure 3 presents strong scaling of the friendster graph up to 4 GPUs on two machines; with NVLink and without NVLink. We observe from this figure that with NVLink enabled architecture we get better scaling. Hence, we believe that if we would have a server with 8 Volta GPUs with NVLink, we could get even better execution times than the reported ones.

V. CONCLUSIONS

We developed a fine-grained tasking based multi-core, multi-GPU, triangle counting method. This linear algebra implementation is up to 10.8× faster than our previous submission. This implementation results in the fastest end-to-end time known at time of publication due to very low overhead costs.

Acknowledgments: We thank Simon Hammond, Cynthia Phillips, and Stephen Olivier for helpful discussions. Sandia

National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

REFERENCES

[1] M. M. Wolf, M. Deveci, J. W. Berry, S. D. Hammond, and S. Rajamanickam, "Fast linear algebra-based triangle counting with kokkoskernels," in *IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–7.

[2] A. Yaşar, S. Rajamanickam, M. M. Wolf, J. W. Berry, and Ü. V. Çatalyürek, "Fast triangle counting using cilk," in *IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–7.

[3] Y. Hu, H. Liu, and H. H. Huang, "High-performance triangle counting on gpus," in *IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2018, pp. 1–5.

[4] S. Samsi, V. Gadepally, M. Hurley, M. Jones, E. Kao, S. Mohindra, P. Monticciolo, A. Reuther, S. Smith, W. Song, D. Staheli, and J. Kepner, "Static graph challenge: Subgraph isomorphism," in *IEEE High Performance extreme Computing Conference (HPEC)*. IEEE, 2017.

[5] M. Deveci, C. Trott, and S. Rajamanickam, "Performance-portable sparse matrix-matrix multiplication for many-core architectures," in *Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2017, pp. 693–702.

[6] KokkosKernels. [Online]. Available: https://github/kokkos/kokkoskernels

[7] S. Samsi, V. Gadepally, M. Hurley, M. Jones, E. Kao, S. Mohindra, P. Monticciolo, A. Reuther, S. Smith, W. Song, *et al.*, "GraphChallenge. org: Raising the Bar on Graph Analytic Performance," *arXiv preprint*, 2018.

[8] M. Deveci, C. Trott, and S. Rajamanickam, "Multithreaded sparse matrix-matrix multiplication for many-core and gpu architectures," *Parallel Computing*, pp. 33–46, 2018.

[9] M. Grigni and F. Manne, "On the complexity of the generalized block distribution," in *International Workshop on Parallel Algorithms for Irregularly Structured Problems*. Springer, 1996, pp. 319–326.

[10] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theoretical Computer Science*, pp. 458–473, 2008.

[11] G. Teodoro, T. D. R. Hartley, Ü. V. Çatalyürek, and R. Ferreira, "Run-time optimizations for replicated dataflows on heterogeneous environments," in *IEEE International Symposium on High Performance Distributed Computing (HPDC)*, 2010, pp. 13–24.

[12] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, 2017.

[13] T. A. Davis and Y. Hu, "The University of Florida sparse matrix collection," *ACM Transactions on Mathematical Software (TOMS)*, p. 1, 2011.