LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-CONF-737552

Triangle Counting for Scale-Free Graphs at Scale in Distributed Memory

R. Pearce

August 28, 2017

2017 IEEE High Performance Extreme Computing Conference (HPEC)
Waltham, MA, United States
September 12, 2017 through September 14, 2017

Disclaimer

Triangle Counting for Scale-Free Graphs at Scale in Distributed Memory

Roger Pearce
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
rpearce@llnl.gov

***Abstract*—Triangle counting has long been a challenge problem for sparse graphs containing high-degree "hub" vertices that exist in many real-world scenarios. These high-degree vertices create a quadratic number of *wedges*, or 2-edge paths, which for brute force algorithms require closure checking or *wedge checks*. Our work-in-progress builds on existing heuristics for pruning the number of wedge checks by ordering based on degree and other simple metrics. Such heuristics can dramatically reduce the number of required wedge checks for exact triangle counting for both real and synthetic scale-free graphs. Our triangle counting algorithm is implemented using HavoqGT, an asynchronous vertex-centric graph analytics framework for distributed memory. We present a brief experimental evaluation on two large real scale-free graphs: a 128B edge web-graph and a 1.4B edge twitter follower graph, and a weak scaling study on synthetic Graph500 RMAT graphs up to 274.9 billion edges.**

I. Introduction

We present a short summary of our work-in-progress towards triangle counting in large scale-free graphs, motivated by the recent *Graph Challenge* [9]. Such graphs are particularly challenging for triangle counting due to the presence of high-degree vertices that create a quadratic number of wedges, that may or may not have a closing edge creating a triangle. Over the years, a number heuristics have been developed to reduce the number of wedges that require checking while maintaining an exact count of the triangles. A series of heuristics that dramatically reduce the number of *wedge checks* is based on directing the edges based on degree [4], [5], which has empirically been evaluated along with other orderings such as by k-core [10], [2], [11]. Our work builds on these heuristics, and we present a new empirical evaluation for large scale-free graphs in distributed memory implemented in our HavoqGT[1] framework.

The key technique that our approach leverages is based on creating an augmented *Degree-Ordered Directed* (DOD) graph, where the original undirected edges are directed from low-degree to high-degree [4], [5]. Edges between vertices of equal degree are directed based on a simple hashed-based tie breaking. Figure 1 illustrates an undirected triangle (left), and the transformation after degree-based ordering (right). An important outcome of this transformation is that the distribution of outgoing directed edges is concentrated on the low-degree vertices, removing many edges from the original high-degree vertices. This dramatically reduces the number of wedges checked during triangle counting. The original undirected degree distributions and the DOD directed degree distributions are shown in Figures 2, 3, for the two real graphs used in this study. In these two cases, the maximum out-degree in the DOD graph is multiple orders of magnitude smaller than the maximum original undirected degree. An additional heuristic we apply to reduce wedge checks and distributed communication is to track which vertices have zero outgoing edges in the DOD graph. Consider the DOD graph illustrated

[1]HaovqGT is available open source: http://software.llnl.gov/havoqgt/

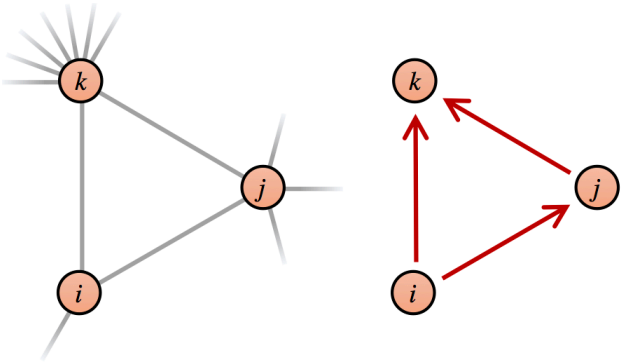


Fig. 1: A triangle in an undirected graph with $d_i < d_j < d_k$ (left) and the associated degree-based ordering (right).

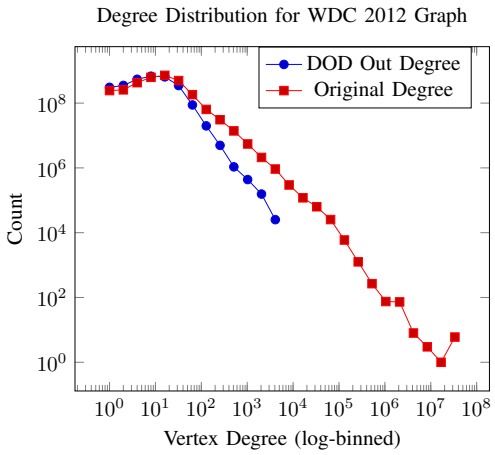


Fig. 2: Degree distributions for WDC 2012 [1] graph.

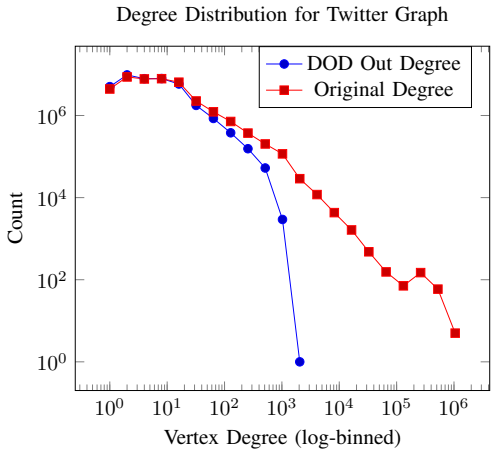


Fig. 3: Degree distributions for Twitter [7] follower graph.

in Figure 1(right); if a vertex has zero out-degree, it cannot be a j vertex in a triangle, eliminating the need to query j for a closing edge to k.

This paper presents a snapshot of our work-in-progress towards triangle counting for large graphs in distributed memory[2]. Strong and weak scaling studies are presented for real and synthetic graphs, including some of the largest graphs used for triangle counting to date.

II. Approach

Our approach begins with an undirected graph, that has been partitioned with a vertex-cut method in HavoqGT called *distributed delegates* [8] to partition the high-degree vertices. The following steps are applied, which are also illustrated in Figure 4:

1) A 2-core decomposition is performed, eliminating all vertices and edges in the 1-periphery that cannot be a member of a triangle. See Figure 4(a-b).
2) A *Degree-Ordered Directed* (DOD) graph is constructed where each edge is directed from low-degree to high-degree. See Figure 4(b-c). For edges between vertices of equal degree, ties are broken based on a hash of the vertex identifiers. A key atribute of this DOD graph is that the maximum out-degree is significantly smaller than the input undirected graph. Figures 2 and 3 compare the degree distribution of the original undirected graph to the out-degree distribution of the DOD graph for our two real test graphs. Another implication of this reduction in maximum out-degree is that a simple 1D partitioning of the DOD graph is now feasible, without the use of vertex-cut *delegates* (for the sparse graphs we have tested). A 1D partitioning is preferred for step 3's wedge creation.
3) Finally, $\binom{out-degree}{2}$ wedges are created for each vertex in the DOD graph, and each wedge-check query is performed using HavoqGT's vertex-centric programming model to check existence of the closing edge.

[2]Interested readers may contact the author for updated manuscripts of this work.

III. Results & Discussion

For our experimental study, we used LLNL's *Catalyst* cluster with 300 compute nodes. Each compute nodes has 24-cores (2x *Intel(R) Xeon(R) CPU E5-2695 v2 @ 2.40GHz*), 128GB DRAM, and 800GB NVRAM (Intel SSD 910). For all experiments, the input undirected graph was stored on NVRAM, and the DOD graph was created and processed in DRAM. There was sufficient DRAM to process both of the real graphs we tested, but not for the synthetic graphs. For consistency, we chose to process all input undirected graphs directly from NVRAM. Informative statistics about our test graphs are shown in Table I, including the *max degree* of the input undirected graphs and the *DOD max degree* after degree order transformation.

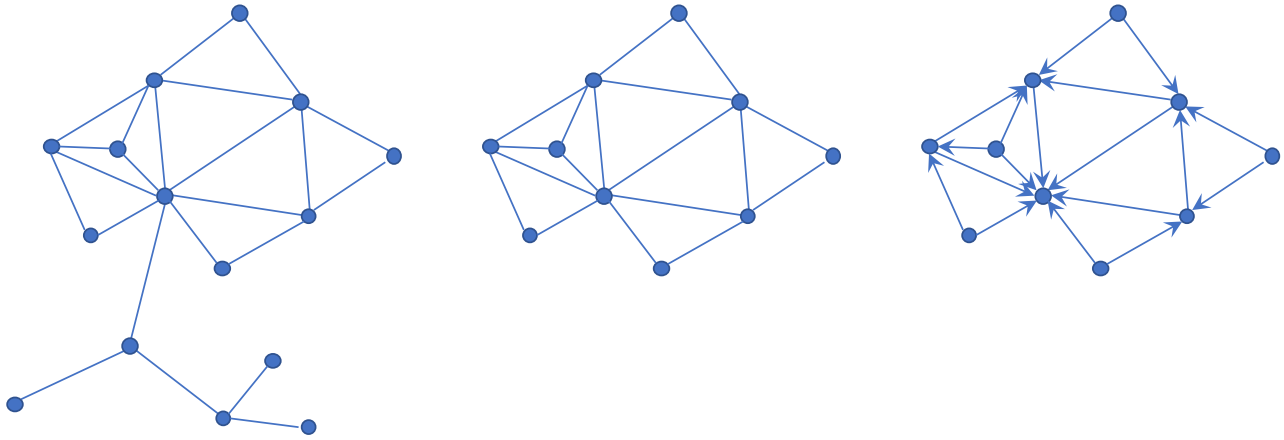
A. Strong Scaling on Real Graphs

Our initial strong scaling experiments have been performed on two large scale-free graphs: Twitter follower graph [7] and WDC 2012 Webgraph [1]. The 2012 Webgraph is the largest known open-source real-graph. HavoqGT is primarily designed for distributed graph processing, so we have limited our studies to graphs large enough to warrant distributed memory.

The strong scaling performance results on the Twitter and WDC 2012 graphs are shown in Figures 5 and 6 respectively. Overall, near-linear strong scaling was observed. Of the three phases required for our algorithm, computing the 2-core and creating the DOD graph are insignificant compared to the cost of wedge checking.

To our knowledge, this is the first result to count the triangles in the WDC 2012 graph, and also the largest real-graph for triangle counting. Using 256 compute nodes, 9.65 trillion triangles were counted in 808.7 seconds.

The Twitter graph has been perviously used in triangle counting studies on large shared-memory machines. Shun, et al. [11] presented the fastest prior result on this dataset, to our knowledge. Using a 4-socket, 40-core Intel shared-memory system, Shun, et al. reported counting the triangles in the Twitter graph in 55.9 seconds. Our experiments using 256 compute nodes of Catalyst improves this to 11.9 seconds (4.7x faster). The approximate break-even point using Catalyst is at 16-nodes (384-cores), where our approach takes 50.3 seconds. Direct comparisons between shared-memory and distributed-memory algorithms and systems are difficult to make, and the CPU generations of these systems are different. However, as the debate beween shared and distributed memory is common among the graph analytics community, this result serves as another example of a medium-sized graph analytic strong scaling in distributed-memory beyond the performance achieved by the best known shared-memory algorithm.



(a) (b) (c)

Fig. 4: Example transformation of (a) input undirected graph; (b) 2-core decomposition that eliminates vertices that cannot be members of triangles; (c) Degree Ordered Directed (DOD) graph from which wedges are computed.

TABLE I: Graph Dataset Information & Results

Dataset	$\|V\|$	$\|E\|$	Max degree	DOD max degree	# Wedges	# Triangles	Fastest time (sec)	Triangles per sec
Twitter [7]	41.65 M	1.47 B	3.1 M	4,158	147.8 B	34.82 B	8.52	4.1 GTPS (256 nodes)
WDC 2012 [1]	3.56 B	128.74 B	95 M	10,683	12.26 T	9.65 T	808.7	11.9 GTPS (256 nodes)
G500 Scale 34	17.18 B	274.9 B	69.8 M	23,896	246 T	50.58 T	36,178	1.4 GTPS (64 nodes)

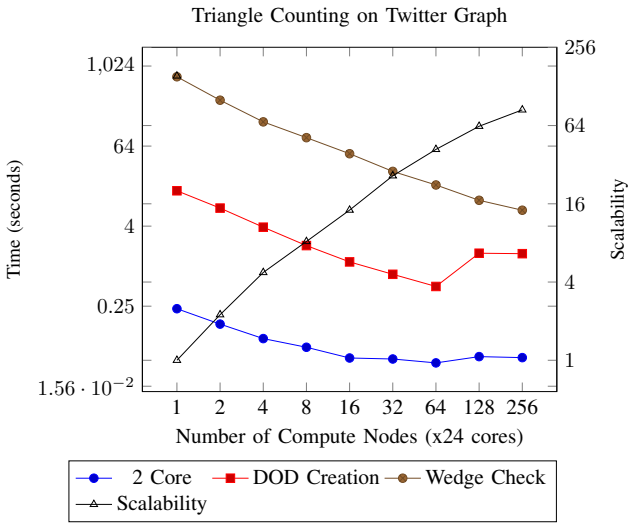


Fig. 5: Triangle Counting results on 1.4 billion edge Twitter follower graph.

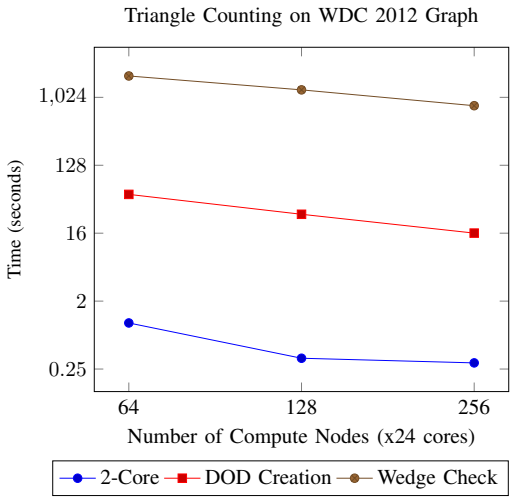


Fig. 6: Triangle Counting results on 128 billion edge WDC 2012 Webgraph. This dataset requires a minimum of 64 compute nodes on Catalyst due to large memory requirements.

B. Weak Scaling on Synthetic Graphs

Additionally, a weak scaling study was performed using undirected synthetic RMAT [3] graphs from the Graph500 [6] challenge, shown in Figure 7. Graph500 scales 28-34 are shown, the largest having 274.9 billion edges, using 1-64 compute nodes of Catalyst. While the graph size (number of vertices and edges) is weak-scaled, the amount of work (number of triangles and wedge checks) grows super-linearly, leading to a super-linear increase in running time. The total time to count the triangles tracks the increase in the number of wedges checked, as the graph size is increased. The rate of wedges and triangles processed per second is shown in Figure 8, where a near-linear scaling is observed.

At the largest scale tested, Scale 34, the graph contained 50.58 trillion triangles, and was processed in 36,178 seconds using 64 compute nodes. To our knowledge, this is the largest graph for triangle counting to date.

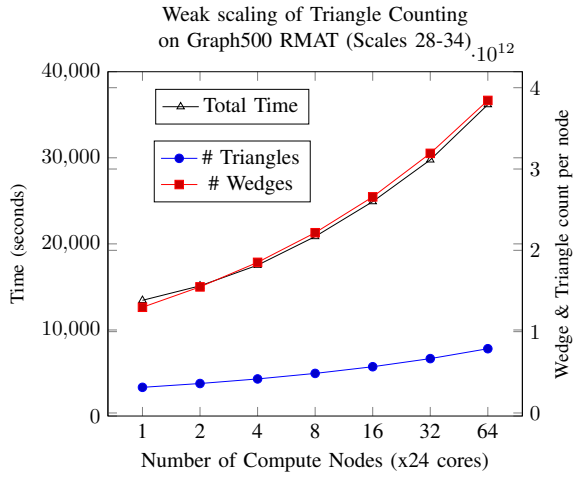


Fig. 7: Triangle Counting results for Graph500 RMAT graphs weak-scaled from Scale 28 (1 compute node) to Scale 34 (64 compute nodes). The total time tracks the number of wedge checks performed, which grows super-linearly.

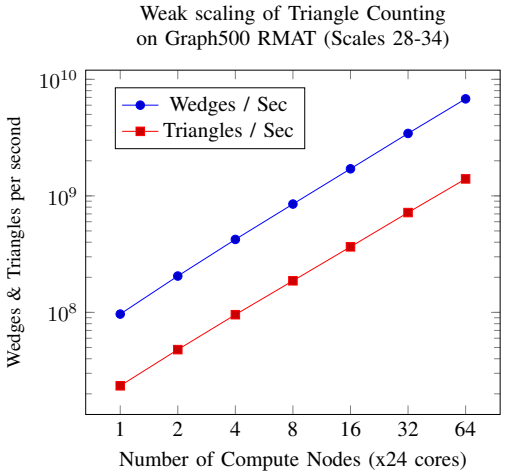


Fig. 8: Rates of Triangles and Wedges processed per second for Graph500 RMAT graphs weak-scaled (Scales 28-34).

IV. Summary

In summary, we present our work to date on triangle counting for large scale-free graphs in HavoqGT. Our approach scales to the largest known real graph in the open-source, WDC 2012, containing 128 billion edges. At peak scaling, our approach can count the 9.65 trillion triangles in this graph in 808.7 seconds using 256 compute nodes of Catalyst. The 1.4 billion edge Twitter follower graph containing 34 billion triangles was computed in 11.2 seconds, improving the prior fastest result for this graph by 4.7x. Finally, a quarter-trillion edge Graph500 RMAT graph was processed in 36,178 seconds, the largest graph used for triangle counting to our knowledge.

Acknowledgment

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-737552). Experiments were performed at the Livermore Computing facility.

References

[1] Web Data Commons webgraph. http://webdatacommons.org/hyperlinkgraph/, 2012.

[2] Ariful Azad, Aydin Buluç, and John Gilbert. Parallel triangle counting and enumeration using matrix algebra. In *Parallel and Distributed Processing Symposium Workshop (IPDPSW), 2015 IEEE International*, pages 804–811. IEEE, 2015.

[3] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In *Fourth SIAM International Conference on Data Mining*, April 2004.

[4] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[5] Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29–41, 2009.

[6] Graph 500 Steering Committee. The graph500 benchmark. http://www.graph500.org, 2010.

[7] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 591–600, New York, NY, USA, 2010. ACM.

[8] Roger Pearce, Maya Gokhale, and Nancy M Amato. Faster parallel traversal of scale free graphs at extreme scale with vertex delegates. In *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*, pages 549–559. IEEE, 2014.

[9] Siddharth Samsi, Vijay Gadepally, Michael Hurley, Michael Jones, Edward Kao, Sanjeev Mohindra, Paul Monticciolo, Albert Reuther, Steven Smith, William Song, Diane Staheli, and Jeremy Kepner. Static graph challenge: Subgraph isomorphism. In *IEEE HPEC*, 2017.

[10] Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *WEA*, pages 606–609. Springer, 2005.

[11] Julian Shun and Kanat Tangwongsan. Multicore triangle computations without tuning. In *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*, pages 149–160. IEEE, 2015.