K-truss decomposition for Scale-Free Graphs at Scale in Distributed Memory

Roger Pearce Geoffrey Sanders
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
{rpearce, sanders29}@llnl.gov

***Abstract*—We update our prior 2017 Graph Challenge submission [11] on large scale triangle counting in distributed memory by extending it to compute the full k-truss decomposition [6] of large scale-free graphs. We build on heuristics to minimize 'wedge checks', by operating on an ordered directed graph, and describe an algorithm to 'unroll' triangle counts when they are scheduled for pruning by the k-truss decomposition. Our k-truss 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-world, scale-free graphs: a 128B edge web-graph and a 1.4B edge twitter follower graph. To our knowledge, the 128B edge web-graph is the largest real-world graph to have its k-truss decomposition computed.**

I. Introduction

We present a short summary of our work-in-progress towards k-truss decomposition in large scale-free graphs, motivated by the recent *Graph Challenge* [12]. We build on our previous work counting triangles in large distributed graphs [11] using heuristics to minimize the effects of high-degree vertices that create a quadratic number of wedges, that may or may not have a closing triangle. The key heuristic that dramatically reduces the number of *wedge checks* is based on directing the edges based on degree [5], [7], which has empirically been evaluated along with other orderings such as by k-core [2], [14], [15]. In this work, we extend this approach and compute a full k-truss decomposition. We demonstrate our approach, implemented in our HavoqGT[1] framework, by computing the k-truss decomposition on two large real-world graphs. To our knowledge, our work presents the largest real-world graph to have its full k-truss decomposition computed.

Recent work on parallel algorithms has let to multiple breakthroughs in capabilities for k-truss decomposition using shared-memory [8], [16] and accelerators [3], [17]. Our work targets a distributed memory environment, namely High-Performance Computing (HPC), with a goal of enabling processing of large datasets, larger than will likely fit in the memory of shared-memory systems. Other previous work on computing k-truss decompositions in distributed systems exist, both HPC on real-world graphs of up to 1B edges [18], and non-HPC on real-world graphs up to 3 million edges [4], [7].

A graph $G(V, E)$ is a set of $|V| = n$ vertices and $|E| = m$ edges, representing relationships between vertices of the form $(i, j) \in E$ for $i, j \in V$. Here our input graph is assumed *undirected*, meaning $(i, j) \in E$ if and only if $(j, i) \in E$. The number of edges incident to vertex i is its *degree*, written d_i. A *triangle* in G is any size three subset of vertices $\{a, b, c\}$ that is fully connected, $(a, b), (a, c), (b, c) \in E$.

A *truss* of order $k \in \{3, 4, ...\}$ for graph $G(V, E)$ is a connected subgraph $H(V', E')$, $V' \subset V$, $E' \subset E$, such that each edge is contained in at least $(k - 2)$ triangles, where each triangle counted is composed of edges/vertices also in $H(V', E')$ [6]. In this work, for a specific integer k, we represent to the set of all trusses of order k of G with H_k, a possibly disconnected subgraph. The k-*truss decomposition* is the family of nested subgraphs:

$$G \supset H_3 \supset H_4 \supset \cdots \supset H_{k_{max}}.$$

The truss decomposition may be computed as an integer label Θ_{ij} on each edge $(i, j) \in E$, where Θ_{ij} is the maximum k for which (i, j) is an edge in H_k.

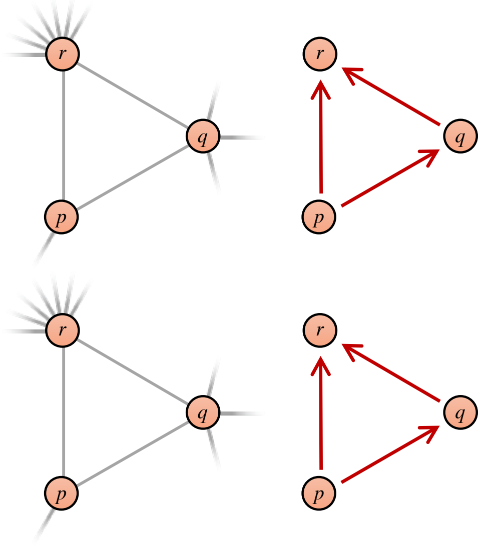


Fig. 1: A triangle in an undirected graph with $\tilde{d}_p < \tilde{d}_q < \tilde{d}_r$ (left) and in the associated degree-ordered directed graph (DODG) (right). Vertex p is the wedge pivot and edge (q, r) is the query edge, associated with triangle $\{p, q, r\}$.

II. Approach

Our approach operates on an augmented graph, discussed in our previous work on triangle counting [11], where the original

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

undirected edges are directed from low-degree to high-degree [5], [7]. Edges between vertices of equal degree are directed based on a simple hashed-based tie breaking. Each vertex is mapped to a unique value $h_i \in (0, 1)$ and the *total ordering* $\tilde{d}_i = d_i + h_i$ is employed to form directed edges $(i \rightarrow j)$ for each $(i, j) \in E$ such that $\tilde{d}_i < \tilde{d}_j$. Our k-truss approach works exclusively on this directed graph, referred to as *DODG* (Degree-Ordered Directed Graph). A limited subset of reverse edges are stored and utilized for k-truss computation.

Figure 1 illustrates an undirected triangle (left), and the transformation into a DODG (right); we will use the notation p to refer to a *wedge pivot vertices* for which one or more *wedges* $\{(p, q), (p, r)\}$ are created, and q refers to a *query vertex* for which a wedge closure (q, r) is queried. Additional metadata specific to our k-truss algorithm is stored for each edge and vertex in the *DODG*, this includes:

- $TC[]$ – Stored for each edge, the count of triangles;
- $qrC[]$ – Stored for each edge, the count of triangles in which the edge is a "qr edge";
- $pRefC[][]$ – Stored for each vertex, a reference count of all successful p vertices for each vertex that is a corresponding q in a triangle.

For clarity and terseness, what follows are descriptions of our k-truss subroutines as sequential algorithms, where distributed communication is not described in the pseudocode. However, these algorithms are actually implemented in our distributed vertex-centric framework as a series of vertex visitor subroutines, as we briefly describe in §II-A.

Algorithm 1: Compute k-truss

```
input  : DODG(V,E) Degree Ordered Directed Graph.
output: KTE[] K-Truss value per edge

TC[], qrC[], pRefC[] ← Initial Triangle
  Computation - Alg 2
K ← 3
do
    do
        Unroll Wedges - Alg 3
        Unroll $(q, r)$ Supported Triangles - Alg 4
        forall edges $e \in DODG(V, E)$ do
            if $TC[e] = 0$ then
                ▷ Edge count will be 0 when
                  all triangles properly
                  unrolled
                Remove $e$ from $DODG(V, E)$
                $KTE[e] \leftarrow$ K − 1
    while edges were cut from $DODG(V, E)$
    K++
while edges remain $\in DODG(V, E)$
```

The logic of our k-truss algorithm is outlined in Alg 1. Initially, all triangles are counted and the edge participation of triangles is tracked, shown in Alg 2. This is followed by an iterative loop that removes any edge (a, b) that is not part of the current H_k being computed, but before an edge can be removed, any triangle $\{(a, b), (a, c), (b, c)\}$ that (a, b) participates in must be decremented from the triangle counts of edges (a, c) and (b, c). We call the associated process *unrolling* triangles and list the serial version in Algs 3, 4.

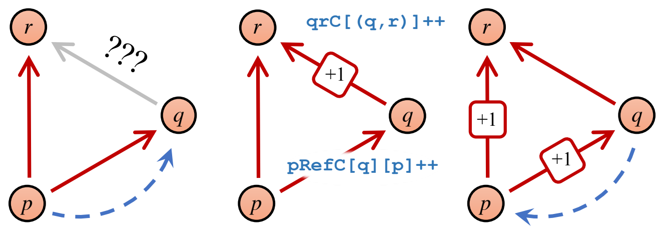


Fig. 2: Stages of distributed wedge check with modifications for performing the k-truss decomposition. First, vertex p asks vertex q if edge (q, r) exists (left). If so, q increments the triangle count of (q, r), and additionally augments/increments the extra data structures (center). Lastly, q reports to p, who increments triangle counts for (p, r) and (p, q), if edge (q, r) exists (right).

Algorithm 2: Initial Triangle Computation

```
input  : DODG(V,E) Degree Ordered Directed Graph.
input  : <operator(V,V) Less than compare
         operator between two Vertices, used by DOD
         Graph.
output: TC[] Triangle count of each edge.
output: qrC[] Count of triangles where edge is a "qr
         edge" per edge.
output: pRefC[][] Reference count of all 'p'
         vertices for closing triangles, stored for each
         'q' vertex.

forall vertices $p \in DODG(V, E)$ do
    forall pairs of incident vertices of $p$, $\{q, r\} \in$
      DODG, where $q < r$ do
        ▷ Creates ordered wedges for
          each vertex in DODG
        if $E(q, r) \in$ DODG then
            ▷ $E$(q,r) found, wedge is
              closed
            $TC[E(q, r)] + +$
            $TC[E(p, q)] + +$
            $TC[E(p, r)] + +$
            $qrC[E(q, r)] + +$
            $pRefC[q][p] + +$
```

The initial triangle sub-routine (Algorithm 2) used by our k-truss algorithm must track additional triangle statistics that are used by the triangle unroll process. These additional statistics were not required in our previous work that only counted the total number of triangles. These additional statistics are: *TC[]*, the count of triangles for each edge; *qrC[]*, the count of triangles in which an edge is a "qr edge"; *pRefC[][]*, a list of all successful p vertices for each vertex that is a corresponding q in a triangle. These additional fields are used and updated during the unrolling process. The first triangle unrolling setup is shown in Algorithm 3, where each wedge in *DODG* is tested if it should remain in the current H_k. If either edge of the wedge $\{(p,q),(p,r)\}$ fails to remain in H_k, its potential closing edge (q,r) is queried, and if found, all the counts are decremented, completely *unrolling* the triangle.

A second case that must be considered is when a (q,r) edge requires removal from H_k; this is outlined in Algorithm 4. When a (q,r) edge is identified for removal, all potential p vertices must be searched using the set of possibilities gathered in *pRefC[][]*. If one of the potential p vertices contains the wedge $\{(p,q),(p,r)\}$, then its counts are decremented, completely *unrolling* the triangle.

This approach using *pRefC[][]* is used because an explicit list of p's for each (q,r) edge would require $O(\#triangles)$ storage, which is often many orders of magnitude larger than the number of edges in the original graph. The size of *pRefC[q][]* is bounded by d_q, implying that the worst-case memory usage for all reference counts (*pRefC[][]*) is $O(E)$, the total number of edges in the original graph.

A. Distributed Graph Algorithm

A brief outline of our distributed vertex-centric implementation in HavoqGT is illustrated in Figures 2, 3, and 4. Because the $DODG$ is partitioned in distributed memory, each step of the algorithm requires explicit message passing, which is performed using a vertex-centric visitor abstraction. The $DODG$ is partitioned such that each vertex and its set of directed out-edges are uniquely assigned to a single processor (MPI rank). HavoqGT provides an asynchronous vertex-centric computation model, where *visitors* (small functions with associated state) are sent and executed on vertices in the distributed graph.

Figure 2 corresponds with Algorithm 2, and illustrates the communication (in blue) necessary to check for a closing wedge edge (q,r), increment counts when its found, and return an acknowledgment in order to increment edges (p,r) and (p,q). During the initial triangle counting phase, each processor generates wedges for all vertices in the local partition that have out degree $>= 2$. A message requesting a wedge check for edge (q,r) is sent to q (left). If q contains the edge (q,r), its metadata is updated (center): the triangle count TC and the qrC count for the edge (q,r) are incremented, and the reference count in $pRefC[q][p]$ is incremented. Lastly, an acknowledgment message is sent back to p (right) to update triangle counts of edges (p,r) and (p,q).

During the k-truss unrolling phase, each processor searches for edges that require removal due to an insufficient number of triangles. The triangles that are destroyed by removing each edge must be accurately and efficiently accounted. When an edge is identified for removal (e.g., (p,r) in Figure 3), the

Algorithm 3: Unroll Wedges

```
input : DODG(V,E) Degree Ordered Directed Graph.
input : <operator(V,V) Less than compare
        operator between two Vertices, used by DOD
        Graph.
input : TC[] Triangle count of each edge.
input : qrC[] Count of triangles where edge is a "qr
        edge" per edge.
input : pRefC[][] Reference count of all 'p'
        vertices for closing triangles, for each 'q'
        vertex.
input : k $k$-truss to decompose

forall vertices $p \in DODG(V,E)$ do
    forall pairs of incident vertices of p, $\{q, r\} \in$
      DODG, where $q < r$ do
        ▷ Creates ordered wedges for
          each vertex in DODG
        if $TC[E(p,q)] < k-2$ OR
          $TC[E(p,r)] < k-2$ then
            ▷ An edge of the wedge is not
              in $H_k$, attempt to unroll
            if $E(q,r) \in$ DODG then
                ▷ Unroll triangle
                $TC[E(\mathrm{q,r})]--$
                $TC[E(\mathrm{p,q})]--$
                $TC[E(\mathrm{p,r})]--$
                $qrC[E(\mathrm{q,r})]--$
                $pRefC[q][p]--$
```

wedges are *unrolled* by performing wedge-checks similar to the original triangle counting phase except that counts are decremented if the closing edge (q,r) is found. There are two important cases that are not mutually exclusive: when the edge to remove is part of the wedge that originally counted a triangle (a (p,r) or a (p,q) edge to a $\{p,q,r\}$ triangle), and when the edge to remove is the query edge for a triangle (a (q,r) edge to a $\{p,q,r\}$ triangle). Figure 3 corresponds with Algorithm 3, and illustrates the communication necessary to *unroll* a (p,r) edge. Figure 3, in effect rolls back the counts from Figure 2. The second case is depicted in Figure 4, corresponding with Algorithm 4, and illustrates the communication necessary to *unroll* a (q,r) edge that still supports a wedge. When a (q,r) edge is removed and $qrC[(q,r)] > 0$, all of the potential wedges it could support must be checked. Messages are sent to all p vertices with a reference count $pRefC[q][p] > 0$ to check for the wedge $\{(p,q),(p,r)\}$. When found, the triangle counts for (p,r) and (p,q) are decremented.

III. Results & Discussion

For our experimental study, we used LLNL's *Catalyst* cluster of 300 compute nodes. Each compute node has 24-

Algorithm 4: Unroll (q, r)-Supported Triangles

```
input : DODG(V,E) Degree Ordered Directed Graph.
input : TC[] Triangle count of each edge.
input : qrC[] Count of triangles where edge is a "qr
        edge" per edge.
input : pRefC[][] Reference count of all 'p'
        vertices for closing triangles, for each 'q'
        vertex.
input : K K-truss to decompose

forall edges $(q, r) \in DODG(V, E)$ do
    if $TC[(q, r)] < k - 2$ AND $qrC[(q, r)] > 0$ then
        ▷ Edge requires removal, but
          still supports other wedges
        forall vertices $p$ where $pRefC[q][p] > 0$ do
            if $E(p, q) \in G(V, E)$ AND
              $E(p, q) \in G(V, E)$ then
                ▷ Found supported wedge for
                  $e(q, r)$, unroll
                $TC[E(q, r)] - -$
                $TC[E(p, q)] - -$
                $TC[E(p, r)] - -$
                $qrC[E(q, r)] - -$
```

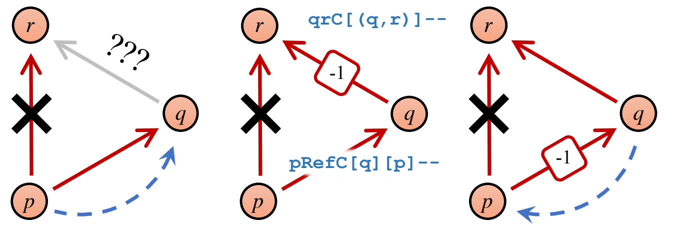


Fig. 3: Unrolling wedges: stages of removing a (p, r) edge of a $\{p, q, r\}$ triangle. First, for every wedge $\{(p, r), (p, q)\}$ involving (p, r), vertex p asks vertex q if edge (q, r) exists (left). If so, q decrements the triangle count of (q, r), and additionally decrements the extra data structures (center). Lastly, q reports to p, who decrements triangle counts for (p, q), if edge (q, r) existed (right). The process of removing a (p, q) edge is similar.

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 *DODG* graph was created and processed in DRAM. We have experimented with two large real-world graphs: *WDC 2012* [1] is a large webgraph with 3.5 billion vertices, 128 billion edges, 95 million maximum degree, and 9.6 trillion triangles; *Twitter Follower* [10] is a graph with 41 million vertices, 1.4 billion edges, 3 million maximum degree, and 34 billion triangles.

Statistics of the k-truss decomposition for WDC and Twitter

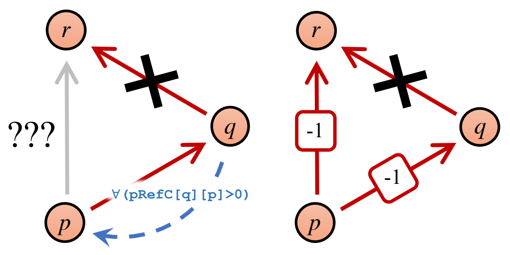


Fig. 4: Unrolling (q, r)-supporting edges: Stages of removing a (q, r) edge of a $\{p, q, r\}$ triangle, only when $qrC[(q, r)] > 0$. First, vertex q asks all vertices p where $\texttt{pRefC}[q][p] > 0$ if edge (p, r) exists (left). If so, p decrements the triangle counts of (p, q) and (p, r) (right).

are shown in Figure 5. Due to its large memory requirement, the WDC graph required 256 compute nodes and the full k-truss decomposition took 55.6 hours. The Twitter graph is shown using 128 compute nodes and took 2.2 hours. To our knowledge, this is the first k-truss analysis of the WDC 2012 graph, and is the largest real graph to date to have a k-truss decomposition computed.

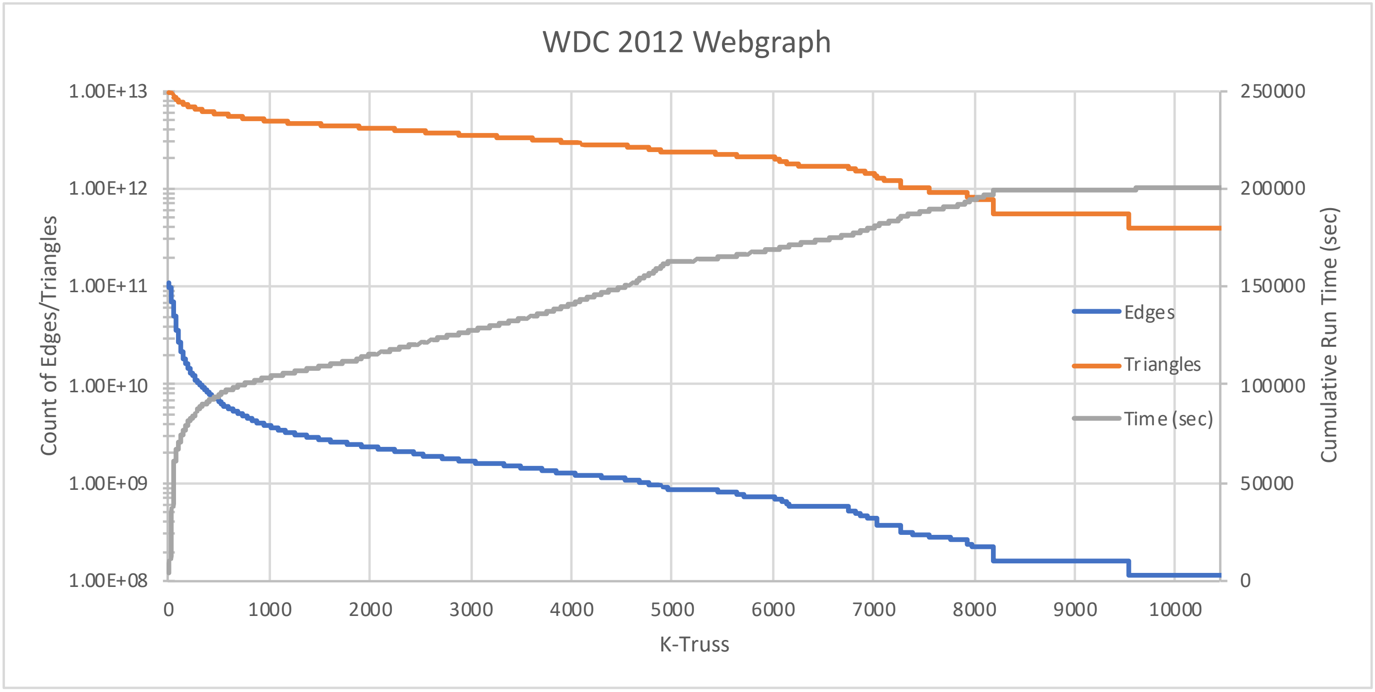
We are currently investigating ways to validate our implementation at this scale, which is challenging without an independent code to compare with that works at this scale. One avenue we will pursue is is to generate non-stochastic Kroncecker graphs [9] with known truss decomposition [13]. To date, we have validated our implementation against an independent code on numerous small datasets. We are also investigating ways to synthetically generate large graphs with more realistic known ground truth k-truss structure as an alternative validation process.

ACKNOWLEDGMENT

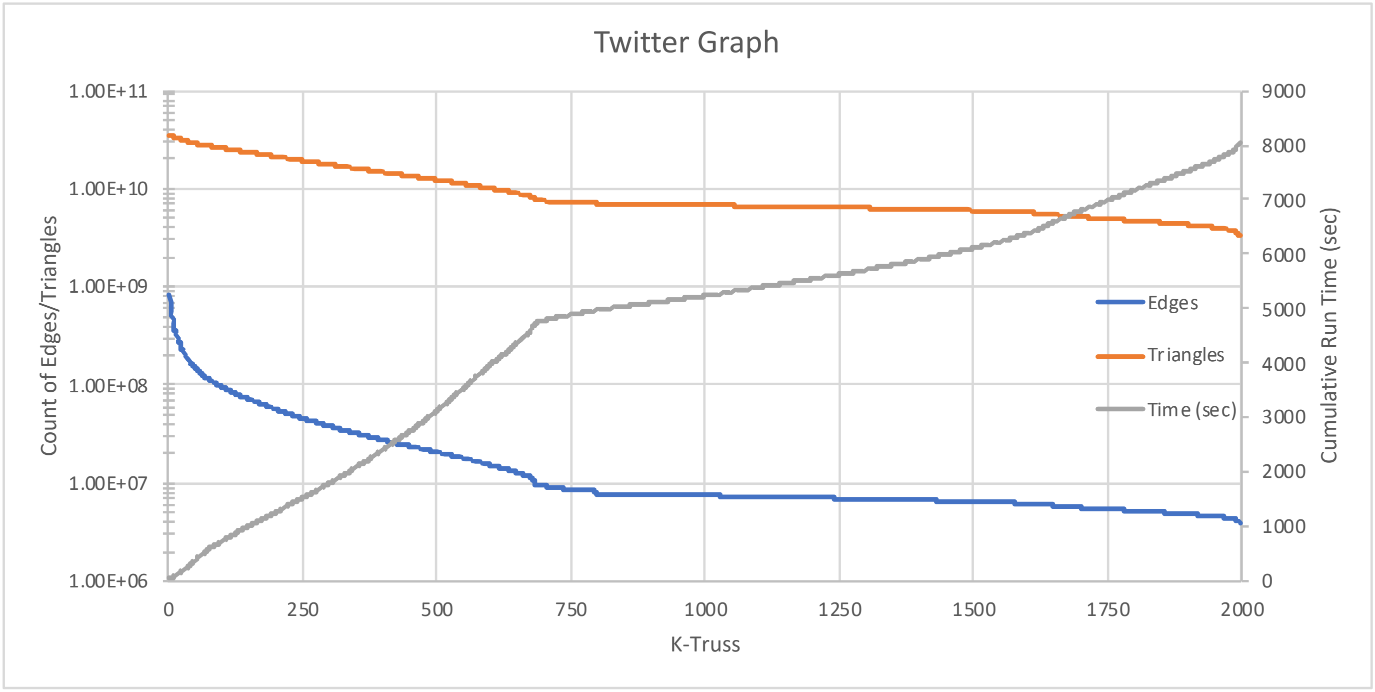
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-757957). Experiments were performed at the Livermore Computing facility.

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] M. Bisson and M. Fatica. Static graph challenge on gpu. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–8, Sept 2017.
[4] P.-L Chen, Chung-Kuang Chou, and M.-S Chen. Distributed algorithms for k-truss decomposition. In *Proceedings - 2014 IEEE International Conference on Big Data, IEEE Big Data 2014*, pages 471–480, 01 2014.
[5] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.
[6] Jonathan Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 2008.
[7] Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29–41, 2009.



(a)



(b)

Fig. 5: K-truss results on (a) 128 billion edge WDC 2012 Webgraph using 256 compute nodes and (b) 1.4 billion edge Twitter graph using 128 compute nodes, running on Catalyst at LLNL. The x-axis specifies the k-truss level. The count of edges and triangles remaining in H_k are shown in log-scale on the left-hand y-axis. The cumulative run time is shown on linear scale on the right-hand y-axis.

[8] H. Kabir and K. Madduri. Shared-memory graph truss decomposition. In *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*, pages 13–22, Dec 2017.

[9] Jeremy Kepner, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Tim Davis, Vijay Gadepally, Michael Houle, Matthew Hubbell, Hayden Jananthan, et al. Design, generation, and validation of extreme scale power-law graphs. In *Graph Algorithms Building Blocks (GABB)*, 2018.

[10] 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.

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

[12] 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.

[13] Geoffrey Sanders, Roger Pearce, Timothy La Fond, and Jeremy Kepner. On large-scale graph generation with validation of diverse triangle statistics at edges and vertices. In *Graph Algorithms Building Blocks (GABB)*, 2018.

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

[15] 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.

[16] S. Smith, X. Liu, N. K. Ahmed, A. S. Tom, F. Petrini, and G. Karypis. Truss decomposition on shared-memory parallel systems. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–6, Sept 2017.

[17] C. Voegele, Y. S. Lu, S. Pai, and K. Pingali. Parallel triangle counting and k-truss identification using graph-centric methods. In *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, Sept 2017.

[18] Jia Wang and James Cheng. Truss decomposition in massive networks. *Proc. VLDB Endow.*, 5(9):812–823, May 2012.