Shared-memory Graph Truss Decomposition

Humayun Kabir
Kamesh Madduri
Penn State

December 19, 2017
Acknowledgments

NSF grants ACI-1253881, CCF-1439075
A graph truss

Introduced by J. Cohen in 2008

A cohesive subgraph for social network analysis

A $k$-truss is a maximal non-trivial single-component subgraph such that every edge is contained in at least $k-2$ triangles.

Similar to $k$-dense, triangle $k$-core, and $k$-community formulations
Truss decomposition

Enumerate all $k$-trusses in the graph

Requires computing the *trussness* of every edge

An edge has trussness $l$ if it belongs to an $l$-truss but not an $(l+1)$-truss
Example: $k$-trusses and trussness

Example graph (6 vertices, 8 edges)

One 2-truss

One 3-truss

# of triangles each edge participates in

Trussness values
Our contributions

PKT, a shared-memory algorithm for computing trussness of every edge

Work-efficient
Level-synchronous parallelism
Memory use proportional to number of edges
Does not use a hash table to maintain edges
Uses a fast triangle counting subroutine
Truss decomposition algorithms

Two main approaches

Bottom-up peeling-based
- Work-efficient (each triangle processed only once)
- Parallelization requires fine-grained synchronization or $O(\#\text{triangles})$ memory

$h$-index based
- Not work-efficient
- Parallelization is simpler
Our algorithm

Based on the bottom-up peeling strategy

Uses fine-grained synchronization to update triangle counts (instead of enumerating all triangles)

Number of barrier synchronizations are proportional to $t_{\text{max}}$ (maximum trussness)
High-level overview of our algorithm

Compute support (\# of triangles an edge participates in), store in array $T$

$k = 2$

while (edges remain to be processed)

1. $S \leftarrow$ set of edges with support $k-2$
2. Update support of edges that form triangles with edges in $S$
3. Remove edges in $S$ from the graph
4. Empty $S$, go to step 1.

Increment $k$
Example: two iterations of PKT

Compute support of all edges

k = 2: identify edges with support 0

k = 2: update support of other edges

k = 3: identify edges with support 1

k = 3: update support of other edges

k = 3: identify edges with support 1
Support update step in PKT

The trickiest part of the algorithm.

We do not explicitly list triangles when updating support. We use atomics instead.

See Algorithm 5 and “Concurrent triangle processing” subsection in paper.
Empirical evaluation

15 large sparse graphs with triangle counts varying from 10 million to 48 billion

Dual-socket Intel system with 128 GB memory. 12-core 2.2 GHz Xeon E5-2650 v4 (Broadwell) processors.

Source code available on GitHub: humayunk1/PKT
Breakdown of execution time
24-core parallel speedup
Trussness and execution time distribution (uk-2002 graph)

- Trussness $k$
  - Percentage of edges with trussness $\leq k$
  - Percentage of total elapsed time
New parallel graph analysis competition (http://graphchallenge.mit.edu/, 2017)

Two challenge problems

Static: triangle counting, truss decomposition
Streaming: stochastic block partition clustering

6 parallel truss decomposition-related submissions
Please see HPEC 2017 proceedings for more details
Conclusions

We present PKT, a new shared-memory parallel algorithm for truss decomposition of graphs

On a 24-core Intel system, PKT achieves a 10X parallel speedup (average, 15 test graphs)

PKT uses a bottom-up, peeling-based approach and is memory-efficient
Possible Improvements

Explore memory use-synchronization tradeoffs

On-the-fly triangle counting instead of precomputing support

For higher values of $k$, extract $k$-cores and then search for trusses
Thank you!

Questions?