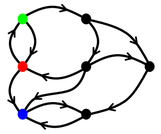
Distributed Triangle Counting in the Graphulo Matrix Math Library

Dylan Hutchison, University of Washington, dhutchis@cs.washington.edu

GRAPHULO

An open source library to orchestrate server-side graph processing in the Apache Accumulo database

- Graphulo uses server-side iterators (a la BigTable) to compute the GraphBLAS kernels on database tables:
  - Matrix Multiply
  - Element-wise Multiply
  - Matrix Reduce
  - Element-wise Add
  - Apply function
  - Matrix Assignment
  - Matrix Extraction (Row/Column filtering)

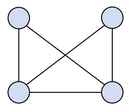


- Advantages of in-database computation include
  - Data Locality (Graphulo brings compute to data)
  - Infrastructure Reuse (no sharing hardware between systems)
  - Accumulo Features (Accumulo as a Big Index, Distributed Execution)

ACCUMULO

Static Graph Challenge

- Finding the k-Truss subgraph
  - Implemented in HPEC '16
- Triangle Counting
  - This poster



Example 3-truss consisting of 2 triangles

Two Triangle Counting Algorithms

Input: Upper triangle of unweighted adjacency matrix $\mathbf{A}$
Output: Number of triangles t

1 $\mathbf{T} = \mathbf{A}$ *// clone* $\mathbf{A}$ *to* $\mathbf{T}$
2 $\mathbf{T} = \mathbf{T} + \mathrm{triu}(\mathbf{A}^\mathsf{T}\mathbf{A})$ *// upper triangle of matrix multiply*
// custom multiply: $a \otimes b = 2$ *if* $a = b = 1$, *otherwise 0*
3 $\mathbf{T}(\mathbf{T} \,\%\, 2 == 0) = 0$ *// filter to odd entries*
4 $t = \mathrm{sum}((\mathbf{T} - 1)/2)$

Algorithm 2: Graphulo Adjacency-only triangle counting

Input: Lower triangle of unweighted adjacency matrix $\mathbf{A}$
Input: Unweighted incidence matrix $\mathbf{E}$
Output: Number of triangles t

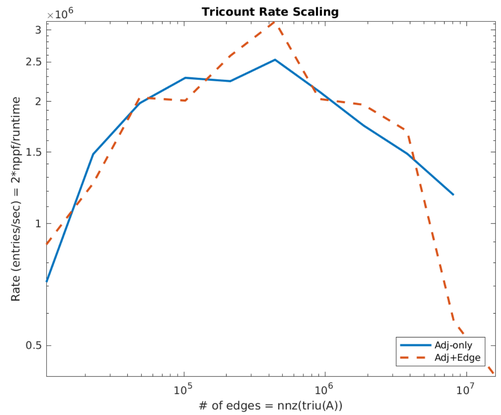
1 $\mathbf{T} = \mathrm{triu}(\mathbf{A}^\mathsf{T}\mathbf{E})$ *// upper triangle of matrix multiply*
2 $t = \mathrm{sum}(\mathbf{T} == 2)$ *// count the entries of* $\mathbf{T}$ *equal to 2*

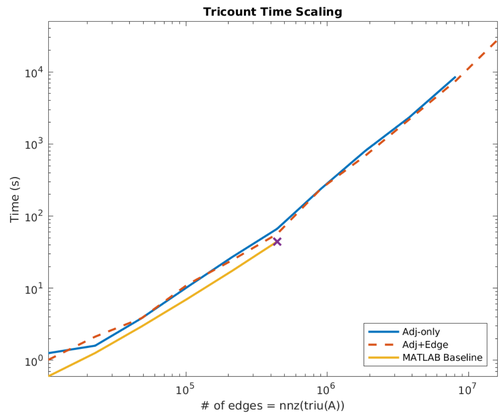
Algorithm 3: Graphulo Adj.+Incidence triangle counting

Triangle Counting Experiment on Power Law Graphs

Question: How do the two triangle counting implementations compare in performance?

Baseline: Single-node in-memory MATLAB implementation





Discussion

- MATLAB Baseline exceeds memory after scale 15 (520k edges); Baseline performs faster while it fits in memory
- Similar performance for both Graphulo algorithms
- Phase transition between scale 15 and 16 (520k and 1.05M edges)
  - Scale ≤ 15 – Reduction step is bottleneck
  - Scale ≥ 16 – Matrix multiply step is bottleneck
  - Peak processing rate achieved at the phase transition point
- Data skew (from power law graphs) limits performance
  - Poor load balancing; one machine receives a disproportionately large workload
  - Permuting rows and columns helps; however, some machine must receive the "super-node"—the highest degree row
- Proposal: A hybrid algorithm to address skew
  - Handle high-degree rows separately with an inner-product matrix multiply
  - Handle low-degree rows with an outer-product matrix multiply

Experiment Details

- m3.xlarge Amazon nodes, each 15 GB mem, 4 vCPU, 2x40 GB SSD
- 8 workers, 3 coordinators, 1 monitor
- Graph500 power law matrix generator, 2^{10} to 2^{20} rows, 16 nonzeros/row

Past Work

- Showed Graphulo faster than single-node in-memory LA packages on MxM (HPEC '15)
- Confirmed results for more complex I/O-bound, single-pass graph analytics (IPDPS '15, HPEC '16)
- Verified Graphulo scales with Accumulo as cluster size increases (HPEC '16)
- Implemented GraphBLAS, Jaccard, k-Truss; showed Graphulo is best for I/O-bound, single-pass analytics (HPEC '16)
- Built LaraDB atop Graphulo; compared to MapReduce on MxM (BeyondMR '17)

Future Work

- Measure how well a hybrid algorithm addresses data skew
- Explore the "export to an external system" strategy, as used in polystores such as Myria and BigDAWG
- Benchmark on real-world data, which likely has less skew
- Address programmability barriers to graph algorithms via code generation

UW DB · NSF · HPC