STINGER: Multi-threaded Graph Streaming

Jason Riedy and David Bader

Georgia Institute of Technology
Data analysis runs throughout.
Many problems can be phrased through *graphs*.
Any many are changing and *dynamic*. 

- **Health care**  Finding outbreaks, population epidemiology
- **Social networks** Advertising, searching, grouping
- **Intelligence** Decisions at scale, regulating algorithms
- **Systems biology** Understanding interactions, drug design
- **Power grid** Disruptions, conservation
- **Simulation** Discrete events, cracking meshes
Outline

Motivation: Graph Algorithms for Analysis

Graphs and Streaming Data

STING/STINGER Analysis Framework

Building Blocks for Streaming Graph Data
  - PageRank
  - Triangle Counting
  - Agglomerative Communities

Observations
General approaches

- **High-performance static graph analysis**
  - Develop techniques that apply to unchanging massive graphs.
  - Provides useful after-the-fact information, starting points.
  - Serves many existing applications well: market research, much bioinformatics, ...
  - Needs to be $O(|E|)$.

- **High-performance streaming graph analysis**
  - Focus on the dynamic changes within massive graphs.
  - Find trends or new information as they appear.
  - Serves upcoming applications: fault or threat detection, trend analysis, online prediction...
  - Can be $O(|\Delta E|)$? $O(\text{Vol}(\Delta V))$? Less data $\Rightarrow$ faster, efficient
Streaming graph data

Data Rates

From www.statisticsbrain.com:

- 58M posts per day on Twitter (671 / sec)
- 1M links shared per 20 minutes on Facebook

Other applications (e.g. network security) need to respond nearly at line rate, 81k-1.5M pps on gigabit ethernet.

Opportunities

- Do not need to analyze the entire graph.
- Different domains: Throughput & latency
  - Expose different levels of concurrency
- Can achieve ridiculous “speed ups.”
Streaming Queries

Different kinds of questions

- How are individual graph metrics (e.g. clustering coefficients) changing?
- What are the patterns in the changes?
  - Are there seasonal variations?
  - What are responses to events?
- What are *temporal anomalies* in the graph?
  - Do key members in clusters / communities change?
  - Are there indicators of event responses before they are obvious?
On to STING...

Motivation: Graph Algorithms for Analysis

Graphs and Streaming Data

STING/STINGER Analysis Framework

Building Blocks for Streaming Graph Data

Observations
STING’s focus

- STING: Spatio-Temporal Interaction Networks and Graphs
- STING manages queries against changing graph data.
  - Visualization and control often are application specific.
- Ideal: Maintain many persistent graph analysis kernels.
  - One current graph snapshot, kernels keep smaller histories.
  - Also (a harder goal), coordinate the kernels’ cooperation.
STING: High-level architecture

- OpenMP + sufficiently POSIX-ish
- Multiple processes for resilience
Initial considerations [Bader, et al.]

- Be useful for the entire “large graph” community
- Permit good performance: No single structure is optimal for all.
- Assume globally addressable memory access and atomic operations
- Not a *graph database*, but supports types, subsets
- Large graph $\Rightarrow$ rare conflicts
Building Blocks for Streaming Graph Data

Motivation: Graph Algorithms for Analysis

Graphs and Streaming Data

STING/STINGER Analysis Framework

Building Blocks for Streaming Graph Data
  - PageRank
  - Triangle Counting
  - Agglomerative Communities

Observations
Incremental PageRank

- PageRank: Well-understood method for ranking vertices based on random walks (related to minimizing conductance).
- Equivalent problem, solve \((I - \alpha A^T D^{-1})x = (1 - \alpha)v\) given initial weights \(v\).
- Goal: Use for seed set expansion, sparse \(v\).
- State-of-the-art for updating \(x\) when the graph represented by \(A\) changes? Re-start iteration with the previous \(x\).
- Can do significantly better for low-latency needs.
- Compute the change \(\Delta x\) instead of the entire new \(x\).
Incremental PageRank: Iteration

Iterative solver

Step $k \rightarrow k + 1$:

$$
\Delta x^{(k+1)} = \alpha (A + \Delta A)^T (D + \Delta D)^{-1} \Delta x^{(k)} +
\alpha [(A + \Delta A)^T (D + \Delta D)^{-1} - A^T D^{-1}] x
$$

- Additive part: Non-zero only at changes.
- Operator: Pushes changes outward.
Incremental PageRank: Limiting Expansion

Iterative solver

Step $k \to k + 1$:

$$\Delta \hat{x}^{(k+1)} = \alpha (A + \Delta A)^T (D + \Delta D)^{-1} \Delta \hat{x}^{(k)}_{\text{ex}} + \alpha \Delta \hat{x}_{\text{held}}$$

$$\alpha [(A + \Delta A)^T (D + \Delta D)^{-1} - A^T D^{-1}] \hat{x}$$

- Additive part: Non-zero only at changes.
- Operator: Pushes sufficiently large changes outward.
Incremental PageRank: Test Cases

- Initial, high-level implementation via sparse matrices in Julia.
- Test graphs from the 10th DIMACS Implementation Challenge.
- Add uniformly random edges... worst case.
- Up to 100k in different batch sizes.
- One sequence of edge actions per graph shared across experiments.
- Conv. & hold base threshold: $10^{-12}$

| Graph                 | $|V|$  | $|E|$  | Avg. Deg. | Size (MiB) |
|-----------------------|-------|-------|-----------|------------|
| caidaRouterLevel      | 192,244 | 609,066 | 3.17      | 5.38       |
| coPapersCiteseer      | 434,102 | 1,603,6720 | 36.94     | 124.01     |
| coPapersDBLP          | 540,486 | 15,245,729 | 28.21     | 118.38     |
| great-britain.osm     | 7,733,822 | 8,156,517 | 1.05      | 91.73      |
| PGPgiantcompo         | 10,680  | 24,316  | 2.28      | 0.23       |
| power                 | 4,941   | 6,594   | 1.33      | 0.07       |
Incremental PageRank: Throughput v. latency

Percent of edge traversals relative to re-started iteration:

![Graph showing percent of edge traversals relative to re-started iteration](image-url)
Triangle Counting

**Current version**

- Count all the triangles around each graph vertex.
- Used in clustering coefficients (numerator), *etc*.
  - Up to 130,000 graph updates per second on X5570 (Nehalem-EP, 2.93GHz)
  - 2000× speed-up over static recomputation
- Main algorithm, for each vertex \( v \):
  - Sort its adjacency list.
  - For each neighbor \( w \),
    - search for \( w \)'s neighbors in the sorted list.
- *Could* compute diag(\( A^3 \)), more or less...
Triangle Counting: Small Batches

Low-latency case

- In general, diag($A^3$) is a silly option.
- But $A^3 \Delta x$, a BFS, to count around a few vertices...
- Brute force (MTAAP10)
  - Roughly 4× slower with moderate batches, and
  - less than 2× slower with small batches.
- Could be reasonable for a quick hack.

Small changes (low latency) may find more applications of linear algebra-like primitives.
Community Detection

What do we mean?

- **Partition** a graph’s vertices into disjoint communities.
- Locally optimize some metric, e.g. modularity, conductance
- Try to capture that vertices are *more similar* within one community than between communities.
- **Modularity**: More internal edges than expected.

Jason’s network via LinkedIn Labs
Parallel Agglomerative Method

- Use a matching to avoid a serializing queue.
  - Simple greedy algorithm.
  - *Would require smuggling data and communication in through element operators...*
- Highly scalable, $5.8 \times 10^6 - 7 \times 10^6$ edges/sec, $8\times - 16\times$ speed-up on 80-thread E7-8870 (thanks Intel!)
- Extends to dynamic community maintenance
  - Extract vertices from communities, re-agglomerate
  - *Matrix triple product-ish*
Parallel Agglomerative Method

- Use a matching to avoid a serializing queue.
  - Simple greedy algorithm.
  - *Would require smuggling data and communication in through element operators...*

- Highly scalable, $5.8 \times 10^6 - 7 \times 10^6$ edges/sec, $8 \times - 16 \times$ speed-up on 80-thread E7-8870 (thanks Intel!)

- Extends to dynamic community maintenance
  - Extract vertices from communities, re-agglomerate
  - *Matrix triple product-ish*
Parallel Agglomerative Method

- Use a matching to avoid a serializing queue.
  - Simple greedy algorithm.
  - *Would require smuggling data and communication in through element operators...*

- Highly scalable, $5.8 \times 10^6 - 7 \times 10^6$ edges/sec, $8\times - 16\times$ speed-up on 80-thread E7-8870 (thanks Intel!)

- Extends to dynamic community maintenance
  - Extract vertices from communities, re-agglomerate
  - *Matrix triple product-ish*
Seed Set Expansion

**Problem**

- Given a small number of vertices, and
- find a region of interest around them.

- Start with a subset consisting of the selection.
- Evaluate the change in *modularity* around the current subset.
- Absorb all vertices that...
  - may increase modularity by a significant amount, or
  - are within the top 10% of changes, or...
- Repeat until the set is large enough.
- *Step-wise guided expansion doesn’t fit current primitives.*
Observations

- Throughput / latency trade offs:
  - Different levels of parallelism and optimizations
  - Larger batches ⇒ higher throughput, more collisions
  - Small batches ⇒ lower latency, more scattered
  - Impact optimizations similarly to direction-optimized BFS

- Can build proposed building blocks against STINGER

- Many algorithms are not naturally expressed:
  - Matching
  - Guided set expansion by changing criteria
  - Streaming versions of these...

- Targets for version 2?
STINGER: Where do you get it?

www.cc.gatech.edu/stinger/

Gateway to

- code,
- development,
- documentation,
- presentations...

(working on usage and development screencasts)

Remember: Still academic code, but maturing.

Users / contributors / questioners:

Georgia Tech, PNNL, CMU, Berkeley, Intel, Cray, NVIDIA, IBM, Federal Government, Ionic Security, Citi
Acknowledgment of support