STING: Spatio-Temporal Interaction Networks and Graphs for Intel Platforms

David Bader, Jason Riedy, Henning Meyerhenke, David Ediger, Timothy Mattson

29 August 2011
Outline

Motivation

Technical
  Overall streaming approach
  Clustering coefficients
  Connected components
  Community detection (in progress)

Related
  Pasqual, a scalable de novo sequence assembler

Plans
Exascale Data Analysis

- 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
Graphs are pervasive

- Sources of massive data: petascale simulations, experimental devices, the Internet, scientific applications.
- New challenges for analysis: data sizes, heterogeneity, uncertainty, data quality.

Astrophysics

- **Problem**: Outlier detection
- **Challenges**: Massive datasets, temporal variation
- **Graph problems**: Matching, clustering

Bioinformatics

- **Problem**: Identifying target proteins
- **Challenges**: Data heterogeneity, quality
- **Graph problems**: Centrality, clustering

Social Informatics

- **Problem**: Emergent behavior, information spread
- **Challenges**: New analysis, data uncertainty
- **Graph problems**: Clustering, flows, shortest paths
These are not easy graphs.

Yifan Hu’s (AT&T) visualization of the Livejournal data set

- There’s nothing you can do to make this look pretty. The graphs defy any geometry.
  - Think about those of us at this meeting.

- These graphs do not partition easily.
  - Forget simple divide and conquer algorithms for parallel analysis.
  - Even bottom-up algorithms encounter parallel problems.
  - Need flexible parallelism at every level to exploit any available structure.
Intel’s non-numeric computing program

Supporting massive, dynamic graph analysis across the spectrum of Intel platforms.

Interactions

- Workshop on Scalable Graph Libraries
  - Co-sponsored by Georgia Tech & PNNL
  - Hosted at Georgia Tech, 29-30 June 2011
  - Attended by 33 from sponsors, industry, and academia. (Timothy Mattson, Roger Golliver, Aydin Buluç, and John Gilbert)

- Hosting Intel-loaned Westmere server
  - Quad E7-8870 with 0.25TiB of memory
  - Active access by Guy Blelloch’s benchmark group, John Gilbert’s KDT group
  - Program-related access for PAPI counter support

- Graph analysis minisymposium at SIAM PP, Feb. 2012
Graph Partitioning and Graph Clustering

- Many application areas identify vertex subsets with many internal and few external edges. Problems addressed include:
  - What are the communities within an (online) social network?
  - How do I speed up a numerical simulation by mapping it efficiently onto a parallel computer?
  - How must components be organized on a computer chip such that they can communicate efficiently with each other?
  - What are the segments of a digital image?
  - Which functions are certain genes (most likely) responsible for?

- 12-13 February 2012, Atlanta, Georgia

- Paper deadline: 21 October 2011
- Co-sponsored by DIMACS, by the Command, Control, and Interoperability Center for Advanced Data Analysis (CCICADA); Pacific Northwest National Laboratory; Sandia National Laboratories; and Deutsche Forschungsgemeinschaft (DFG).

1. At the URL:
   - Definitions of objectives.
   - Collection of real-world and artificial graphs.
   - Measurement and judging standards.
Overall streaming approach


Jason’s network via LinkedIn Labs

Assumptions

1. LinkedIn is a wonderful example. Only some of a person’s “social network” appears.
2. Algorithms must cope with surprises.

1. A graph represents some real-world phenomenon.
   • But not necessarily exactly!
   • Noise comes from lost updates, partial information, ...
**Assumptions**

- We target massive, “social network” graphs.
  - Small diameter, power-law degrees
  - Small changes in massive graphs often are unrelated.

1. Methods do apply to non-power-law, non-social-network graphs. Those often behave *much* better than social networks, so the methods may look artificially good on physical networks.
Overall streaming approach

1. Batching is well-established in server contexts (e.g. SEDA).
2. We focus on relatively fast changes. The engineering choices for slower changes like those that (ideally) occur in power networks and scientific literature may be different. But if we can support networks that change quickly, slower changes become a natural consequence.
3. “Fast” changes need not be related. The LinkedIn global network changes often, but few peoples’ networks change frequently.

Assumptions

- The graph changes but we don’t need a continuous view.
  - We can accumulate changes into batches...
  - But not so many that it impedes responsiveness.
Difficulties for performance

- What partitioning methods apply?
  - Geometric? Nope.
  - Balanced? Nope.
  - Is there a single, useful decomposition? **Not likely.**

- Some partitions exist, but they don’t often help with balanced bisection or memory locality.

- Performance needs new approaches, not just standard scientific computing methods.

Jason’s network via LinkedIn Labs


- All categories overlap. (No, really. Not saying how.)
  - And LinkedIn gets some categories wrong...

- Any localizing partitioning helps with performance. But how can we find that partitioning?
STING manages queries against changing graph data.

- Visualization and control often are application specific.
- Ideal: Maintain many persistent graph analysis kernels.
  - Keep one current snapshot of the graph resident.
  - Let kernels maintain smaller histories.
  - Also (a harder goal), coordinate the kernels' cooperation.
• Batches provide two levels of parallelism.
  • Busy loci of change: Know to share the busy points.
  • Scattered changes: Parallel across (likely) independent changes.

• The massive graph is maintained in a data structure named STINGER.
STING Extensible Representation:

1. Fundamentally an adjacency list representation using fixed-size array chunks within the lists.
2. Supports semantic graphs through edge and vertex types.
3. Many low-level aspects still in flux.
4. Cache coherence needs are an open question. Some inconsistency may be similar to errors already in the data.
5. Here *inconsistency* refers to the data being represented and not data structure integrity.

- Rule #1: No explicit locking.
  - Rely on atomic operations.
- Massive graph: Scattered updates, scattered reads rarely conflict.
- Use time stamps for some view of time.
Initial results

Prototype STING and STINGER

Monitoring the following properties:

1. clustering coefficients,
2. connected components, and
3. community structure (in progress).

High-level

- Support high rates of change, over 10k updates per second.
- Performance scales somewhat with available processing.
- Gut feeling: Scales as much with sockets as cores.

1. Yes, we do regret picking two with the same acronym.
2. Working on low-level microbenchmarks and experiments to verify scaling limits.

http://www.cc.gatech.edu/~bader/code.html
Experimental setup

Unless otherwise noted

<table>
<thead>
<tr>
<th>Line</th>
<th>Model</th>
<th>Speed (GHz)</th>
<th>Sockets</th>
<th>Cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nehalem</td>
<td>X5570</td>
<td>2.93</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Westmere</td>
<td>E7-8870</td>
<td>2.40</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

- Westmere loaned by Intel (*thank you!*)
- All memory: 1067MHz DDR3, installed appropriately
- Implementations: OpenMP, gcc 4.6.1, Linux ≈ 3.0 kernel
- Artificial graph and edge stream generated by R-MAT [Chakrabarti, et al.]
  - Scale \( x \), edge factor \( f \) ⇒ \( 2^x \) vertices, \( \approx f \cdot 2^x \) edges.
  - Edge actions: 7/8th insertions, 1/8th deletions
  - Results over five batches of edge actions.
- Caveat: No vector instructions, low-level optimizations yet.

- Dual-socket, six-core machines to be added soon.
- Not ported to distributed/PGAS environments like SCC yet.
- R-MAT has known issues, but at least they’re known.
- We only delete edges known to exist.
- Still trying to define best practices for these experiments. Feedback is welcome!

<table>
<thead>
<tr>
<th>Line</th>
<th>Memory request buffer depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nehalem</td>
<td>64</td>
</tr>
<tr>
<td>Westmere</td>
<td>88</td>
</tr>
</tbody>
</table>
Clustering coefficients

- Used to measure “small-world-ness” [Watts and Strogatz] and potential community structure
- Larger clustering coefficient ⇒ more inter-connected
- Roughly the ratio of the number of actual to potential triangles

- Defined in terms of triplets.
  - \( i - v - j \) is a closed triplet (triangle).
  - \( m - v - n \) is an open triplet.
- Clustering coefficient:
  \[ \frac{\text{# of closed triplets}}{\text{total # of triplets}} \]
- Locally around \( v \) or globally for entire graph.

- Multiple possibilities for the global clustering coefficient, or for directed graphs (transitivity).
- Keeping to undirected and local for discussion.
Updating triangle counts

Given Edge \{u, v\} to be inserted (+) or deleted (-)

Approach Search for vertices adjacent to both \(u\) and \(v\), update counts on those and \(u\) and \(v\)

Three methods

- **Brute force** Intersect neighbors of \(u\) and \(v\) by iterating over each, \(O(d_u d_v)\) time.

- **Sorted list** Sort \(u\)'s neighbors. For each neighbor of \(v\), check if in the sorted list.

- **Compressed bits** Summarize \(u\)'s neighbors in a bit array. Reduces check for \(v\)'s neighbors to \(O(1)\) time each. Approximate with Bloom filters. [MTAAP10]

All rely on atomic addition.

- Expand to batches by grouping changes by least-degree vertex.
  - High-degree, high-churn vertices: work collapses
  - Nearby changes end up accessing same edge information.

- Why not a dynamic, sorted data structure for edge arrays?
  - Overhead is very high for common edge-walking uses [Madduri and Bader]

- Assumes the graph isn’t changing while a batch is processed.
  - One alternative is to re-count around all changes rather than increment (decrement) counts.
  - Another: Use time stamp ranges to check if an edge is / is not present when kernel begins.
On some platforms, Bloom filters outperform sorted intersections.

Needs more optimization, but up to real-world rates.

The 2.93 GHz Nehalem starts below the 2.40 GHz Westmere.

But at 16 threads... We’re working on a performance bottleneck analysis.

Could benefit from low-level operations, different bit summaries (FastBit?).
Different batch sizes

- Variability increases as batch size decreases.
  - Used `omp parallel for` without scheduling clause.
  - gcc’s default is dynamic scheduling with a chunk size of 1.
- Performance appears to level off with larger batches.
Connected components

- Maintain a mapping from vertex to component.
- *Global property*, unlike triangle counts
- In "scale free" social networks:
  - Often one big component, and
  - many tiny ones.
- Edge changes often sit *within* components.
- Remaining insertions merge components.
- Deletions are more difficult...

1. Useful in sampling algorithms.
2. Future work: consider components from thresholds.
Connected components

- Maintain a mapping from vertex to component.
- **Global property**, unlike triangle counts
- In “scale free” social networks:
  - Often one big component, and
  - many tiny ones.
- Edge changes often sit *within* components.
- Remaining insertions merge components.
- Deletions are more difficult...

1. Useful in sampling algorithms.
2. Future work: consider components from thresholds.
3. And you never even look at the graph itself for insertions, only the components.
The difficult case

- **Very few deletions matter.**
- Determining *which* matter may require a large graph search.
  - Re-running static component detection.
  - (Long history, see related work in [MTAAP11].)
- Coping mechanisms:
  - **Heuristics.**
  - Second level of batching.

1. Will discuss heuristics next.
2. Accumulate remaining deletions into a second batch. Update from this batch less often. Produces an *approximation* until the deletions are handled, then *exact* result again.
Rule out effect-less deletions

- Use the spanning tree by-product of static connected component algorithms.

- Ignore deletions when one of the following occur:
  1. The deleted edge is not in the spanning tree.
  2. If the endpoints share a common neighbor*.
  3. If the loose endpoint can reach the root*.

- In the last two (*), also fix the spanning tree.

Rules out 99.7% of deletions.
1. Insertion-heavy load (7/8ths), so performance increases with batch size.
2. Maintaining components in a weighted view may be different.
3. Deletions handled “within the batch,” but only two batches have deletions after the heuristics.
4. Westmere’s hyperthreading is more stable in this test.
Community detection (work in progress)

Greedy, agglomerative partitioning
- Partition to maximize modularity, minimize conductance, ...

Seed set expansion
- Grow an optimal / "relevant" community around selection.
- (Work with Jonny Dimond of KIT.)

1. Modularity: A measure of internal edge counts relative to a random graph
2. Conductance: A normalized edge cut measure
3. Think of recommender algorithms, caching, visualization.
5. We have initial streaming work in each, but very preliminary.
Agglomerative community detection

Parallel greedy, agglomerative partitioning [PPAM11]

- Score edges by optimization criteria.
- Chose a maximal, heavy-weight matching.
  - Negate edge scores if minimizing conductance.
- Contract those edges.
- Mimics sequential optimizers, but produces different results.
Performance

- R-MAT on right.
- Livejournal
  - 15M vertex, 184M edge
  - 6-12 hours on E7-8870
- Highly variable performance.
- Algorithm under development.

1. R-MAT doesn’t show community structure, so times are optimistic.
2. These results are from single runs of a non-deterministic algorithm.
3. From Stanford’s SNAP graph data.
4. We expect variable performance from discrete optimization of an irregular function.
5. Defining “scalable” needs to focus on the data and not the processing.
6. Heavy performance tail: Very few edges increase modularity at each iteration.
Related: Pasqual

A scalable de novo assembler for next-gen gene sequencing

Work by David Bader, Henning Meyerhenke, Xing Liu, Pushkar Pande.

- Next-generation sequencers produce mountains of small gene sequences.
- Assembling into a genome: Yet another large graph problem.
- Pasqual forms a compressed overlap graph and traces paths.
- **Only scalable and correct shared-memory assembler.**
  - Faster and uses less memory than other existing systems.
  - Evaluation against the few distributed assemblers is ongoing.
- Algorithm extends to metagenomics.

http://www.cc.gatech.edu/pasqual/
### Similar speed, better results

<table>
<thead>
<tr>
<th>Length</th>
<th>Code</th>
<th>Time (min)</th>
<th>N50 (bp)</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>Velvet</td>
<td>&gt;12h</td>
<td>1355</td>
<td>683</td>
</tr>
<tr>
<td></td>
<td>Edena</td>
<td>451.15</td>
<td>1375</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>ABYSS</td>
<td>56.52</td>
<td>1412</td>
<td>521</td>
</tr>
<tr>
<td></td>
<td>SOAPdenovo</td>
<td>15.62</td>
<td>1470</td>
<td>485</td>
</tr>
<tr>
<td>100</td>
<td>Pasqual</td>
<td>15.50</td>
<td>1451</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>Velvet</td>
<td>132.68</td>
<td>6635</td>
<td>175</td>
</tr>
<tr>
<td></td>
<td>Edena</td>
<td>466.12</td>
<td>6545</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>ABYSS</td>
<td>92.92</td>
<td>6229</td>
<td>136</td>
</tr>
<tr>
<td></td>
<td>SOAPdenovo</td>
<td>18.05</td>
<td>6879</td>
<td>142</td>
</tr>
<tr>
<td></td>
<td>Pasqual</td>
<td>19.15</td>
<td>7712</td>
<td>6</td>
</tr>
</tbody>
</table>

- Artificial data, timings on dual-socket Nehalem.
- Take a known sequence and re-sample as if it were being sequenced.
- Length: How long are the reads (samples)?
- Coverage: How much to the reads overlap?
- N50: Measure of quality, higher is better.
- Errors: Compared to known sequence.
Zebrafish, 61Mbp (cov 30)

Far better speed and results

<table>
<thead>
<tr>
<th>Length</th>
<th>Code</th>
<th>Time (min)</th>
<th>N50 (bp)</th>
<th>Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Velvet</td>
<td>256.02</td>
<td>4045</td>
<td>799</td>
</tr>
<tr>
<td></td>
<td>Edena</td>
<td>&gt;12h</td>
<td>2661</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ABYSS</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>SOAPdenovo</td>
<td><strong>31.83</strong></td>
<td>3998</td>
<td>725</td>
</tr>
<tr>
<td></td>
<td><strong>Pasqual</strong></td>
<td>57.27</td>
<td><strong>4535</strong></td>
<td>0</td>
</tr>
<tr>
<td>200</td>
<td>Velvet</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Edena</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>ABYSS</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>SOAPdenovo</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td><strong>Pasqual</strong></td>
<td><strong>38.07</strong></td>
<td><strong>7911</strong></td>
<td>0</td>
</tr>
</tbody>
</table>

STING

Related

Pasqual, a scalable de novo sequence assembler

Zebrafish, 61Mbp (cov 30)
Performance v. SOAPdenovo

1. On the Zebrafish data, length 100, quad-socket Westmere.
Community detection  Improving the algorithm, pushing into streaming by de-agglomerating and restarting.

Seed set expansion  Maintaining not only one expanded set, but multiple for high-throughput monitoring.

Microbenchmarks  Expand on initial promising work on characterizing performance by peak number of memory operations achieved, find bottlenecks by comparing with microbenchmarks.

Distributed/PGAS  STINGER fits a PGAS model well (think SCC). Interested in exploring distributed algorithms.

Packaging  Wrap STING into an easily downloaded and installed tool.
Bibliography I

D. Ediger, K. Jiang, E. J. Riedy, and D. A. Bader.
Massive streaming data analytics: A case study with clustering coefficients.

D. Ediger, E. J. Riedy, and D. A. Bader.
Tracking structure of streaming social networks.
In Proceedings of the Workshop on Multithreaded Architectures and Applications (MTAAP’11), May 2011.

K. Madduri and D. A. Bader.
Compact graph representations and parallel connectivity algorithms for massive dynamic network analysis.
In 23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS), Rome, Italy, May 2009.
Detecting communities from given seeds in social networks.

E. J. Riedy, H. Meyerhenke, D. Ediger, and D. A. Bader.
Parallel community detection for massive graphs.
D. Chakrabarti, Y. Zhan, and C. Faloutsos.
R-MAT: A recursive model for graph mining.
In Proc. 4th SIAM Intl. Conf. on Data Mining (SDM), Orlando, FL, Apr. 2004. SIAM.

D. J. Watts and S. H. Strogatz.
Collective dynamics of ‘small-world’ networks.