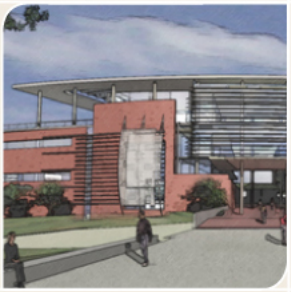
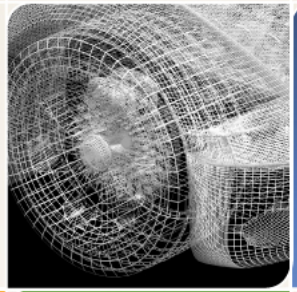
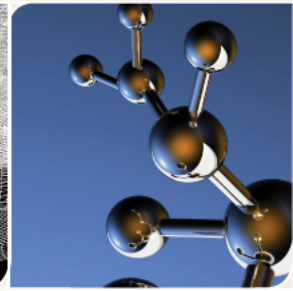
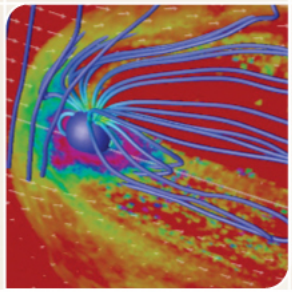
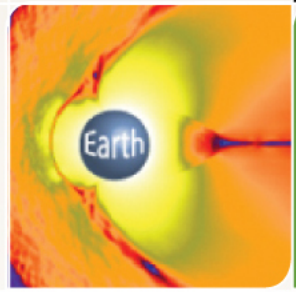


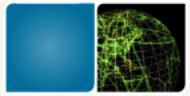
STING: Spatio-Temporal Interaction Networks and Graphs for Intel Platforms

David Bader, Jason Riedy,
David Ediger, Rob McColl,
Timothy G. Mattson*

24 July 2012

Georgia Tech College of Computing
Computational Science and Engineering

Outline


Motivation

STING for streaming, graph-structured data on Intel-based platforms

World-leading community detection

Community-building interactions

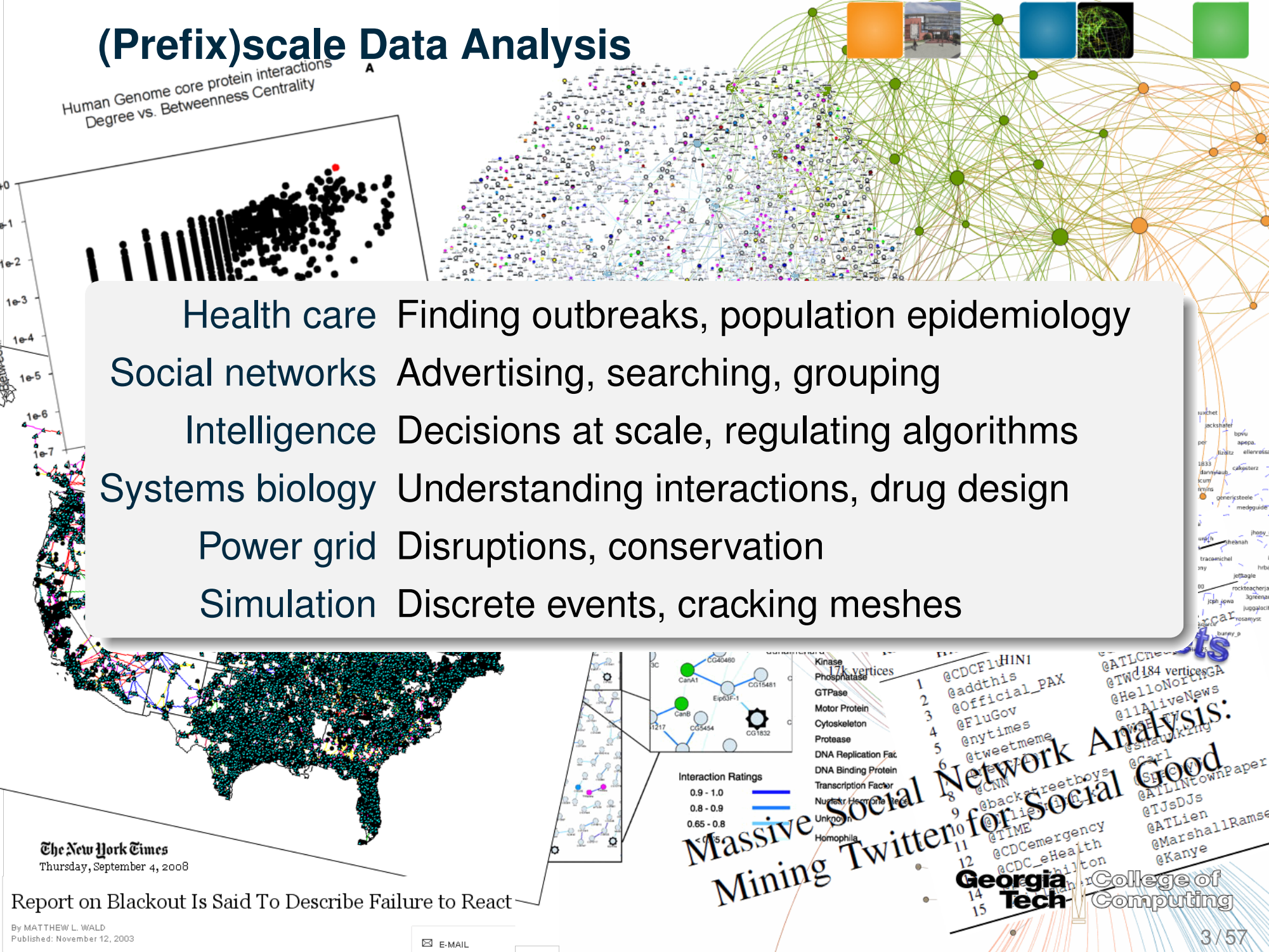
Plans and direction

Note: Without hard limits, academics will use at least all the time...

Georgia Tech College of Computing

(Prefix)scale 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

Graphs: things and relationships

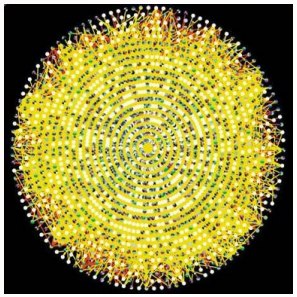
- Different kinds of things, different kinds of relationships, but graphs provide a framework for analyzing the *relationships*.
- New challenges for analysis: data sizes, heterogeneity, uncertainty, data quality.

Astrophysics

Problem Outlier detection
Challenges Massive data sets, temporal variation
Graph problems Matching, clustering

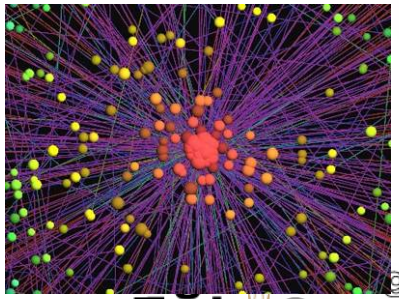
Bioinformatics

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

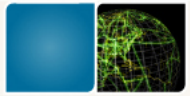


Social Informatics

Problem Emergent behavior, information spread
Challenges *New* analysis, data uncertainty
Graph problems Clustering, flows, shortest paths



No shortage of data...

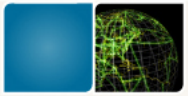

Existing (some out-of-date) data volumes

NYSE	1.5 TB generated daily, maintain a 8 PB archive
Google	"Several dozen" 1PB data sets (CACM, Jan 2010)
Wal-Mart	536 TB, 1B entries daily (2006)
EBay	2 PB, traditional DB, and 6.5PB streaming, 17 trillion records, 1.5B records/day, each web click is 50-150 details. `http://www.dbms2.com/2009/04/30/ebays-two-enormous-data-warehouses/`
Facebook	845 M users... and growing(?)

- All data is *rich* and *semantic* (**graphs!**) and changing.
- Base data rates include items and not *relationships*.

Georgia Tech College of Computing

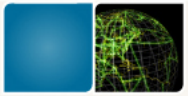
General approaches


- 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, ...
- 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, ...

Both very important to different areas.
GT's primary focus is on streaming.
Note: Not CS theory streaming, but analysis of streaming data.

Why analyze data streams?


Data volumes

NYSE 1.5TB daily

LHC 41TB daily

Facebook, *etc.* Who knows?

Data transfer

- 1 Gb Ethernet: 8.7TB daily at 100%, 5-6TB daily realistic
- Multi-TB storage on 10GE: 300TB daily read, 90TB daily write
- CPU $\leftrightarrow$ Memory: QPI,HT: 2PB/day@100%

Data growth

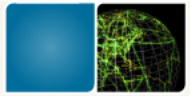
- Facebook: $> 2\times$/yr
- Twitter: $> 10\times$/yr
- Growing sources: Bioinformatics, μsensors, security

Speed growth

- Ethernet/IB/*etc.*: $4\times$ in next 2 years. Maybe.
- Flash storage, direct: $10\times$ write, $4\times$ read. Relatively huge cost.

Georgia Tech College of Computing

Intel's non-numeric computing program (our perspective)

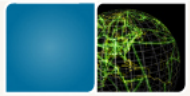

Supporting analysis of massive graph under rapid change across the spectrum of Intel-based platforms.

STING on Intel-based platforms

- Maintain a graph and metrics under large data changes
  - Performance, documentation, distribution
  - Greatly improved ingestion rate, improved example metrics
  - Available at `http://www.cc.gatech.edu/stinger/`.
- Algorithm development
  - World-leading community detection (clustering) performance on Intel-based platforms
    - Good for focused or multi-level analysis, visualization, ...
  - Developments on community *monitoring* in dynamic graphs

Georgia Tech College of Computing

On to STING...


Motivation

STING for streaming, graph-structured data on Intel-based platforms

- Assumptions about the data
- High-level architecture
- Algorithms and performance

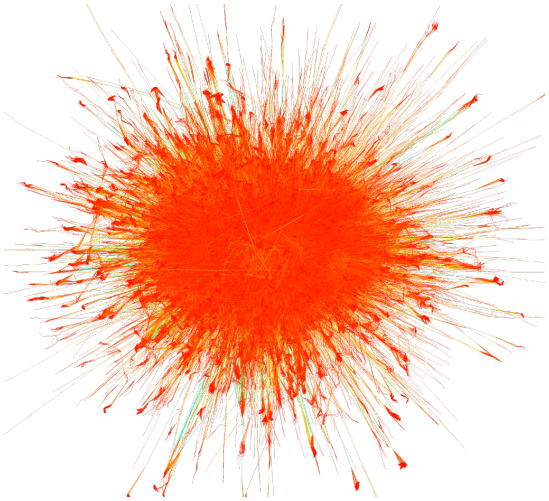
World-leading community detection

Community-building interactions

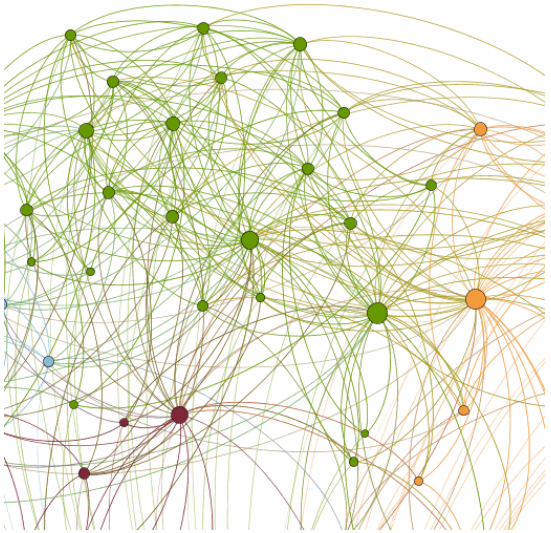
Plans and direction

Georgia Tech College of Computing

Overall streaming approach



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



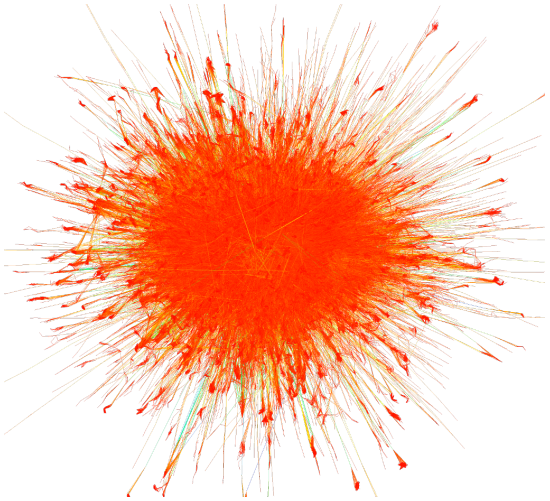
Jason's network via LinkedIn Labs

Assumptions

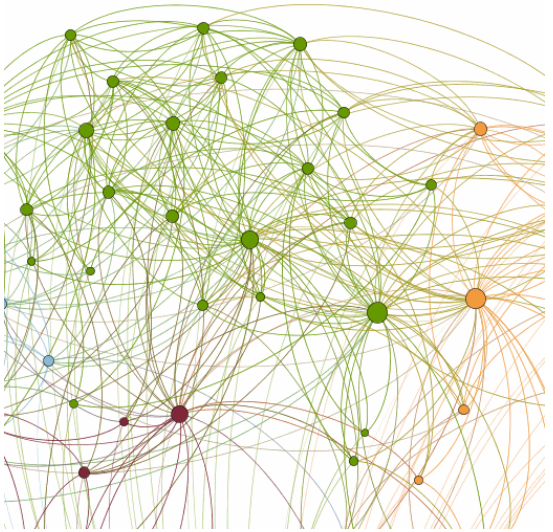
- A graph represents some real-world phenomenon.
  - But **not** necessarily exactly!
  - Noise comes from lost updates, partial information, ...

Georgia Tech College of Computing

Overall streaming approach



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



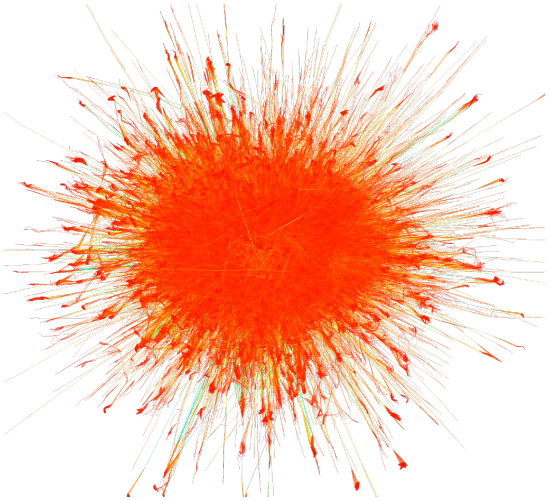
Jason's network via LinkedIn Labs

Assumptions

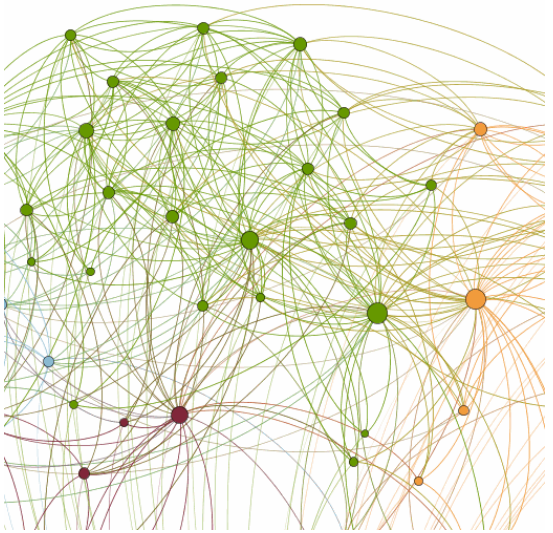
- We target massive, "social network" graphs.
  - Small diameter, power-law degrees
  - Small changes in massive graphs often are unrelated.

Georgia Tech College of Computing

Overall streaming approach



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



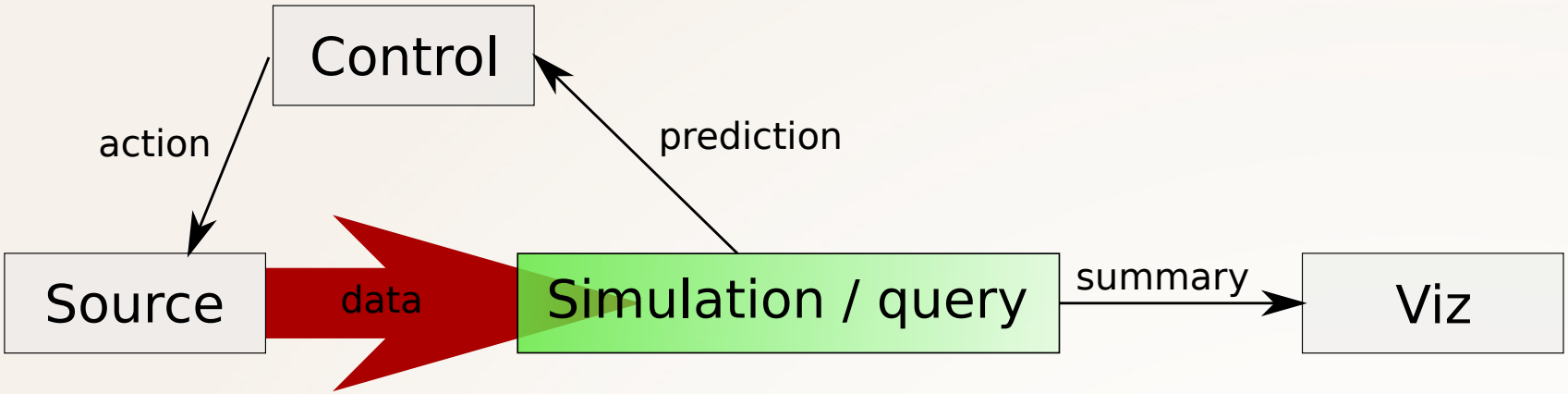
Jason's network via LinkedIn Labs

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.

Georgia Tech College of Computing

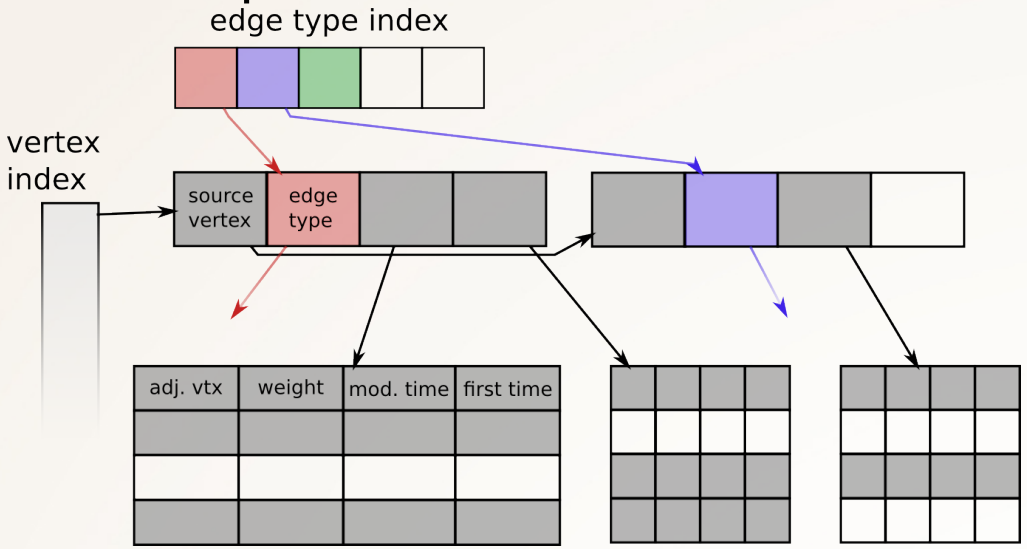
STING's focus



- STING manages queries against changing graph data.
  - Visualization and control often are application specific.
- Ideal: Maintain many persistent graph analysis kernels.
  - One current snapshot of the graph, kernels keep smaller histories.
  - Also (a harder goal), coordinate the kernels' cooperation.
- STING components:
  - Framework for *batching* input changes.
  - Lockless data structure, STINGER, accumulating a typed graph with timestamps.

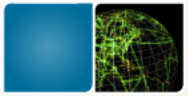
STINGER

STING Extensible Representation:



- 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.

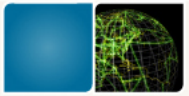
STING's year


Over the past year

- Improved performance on Intel-based platforms by 14$\times$[HPEC12]
- Evaluated multiple dynamic kernels across platforms[MTAAP12, ICASSP2012]
- Documentation and tutorials
- Assist projects with STING adoption on Intel-based platforms

STING in use


Where is it?

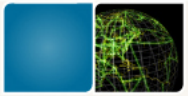
`http://www.cc.gatech.edu/stinger/`

Code, development, documentation, presentations...

Who is using it?

- Center for Adaptive Supercomputing Software – Multithreaded Architectures (CASS-MT) directed by PNNL
- PRODIGAL team for DARPA ADAMS (Anomaly Detection at Multiple Scales) including GTRI, SAIC, Oregon State U., U. Mass., and CMU
- DARPA SMISC (Social Media in Strategic Communication) including GTRI
- Well-attended tutorials given at PPoPP, in MD, and locally.

Experimental setup

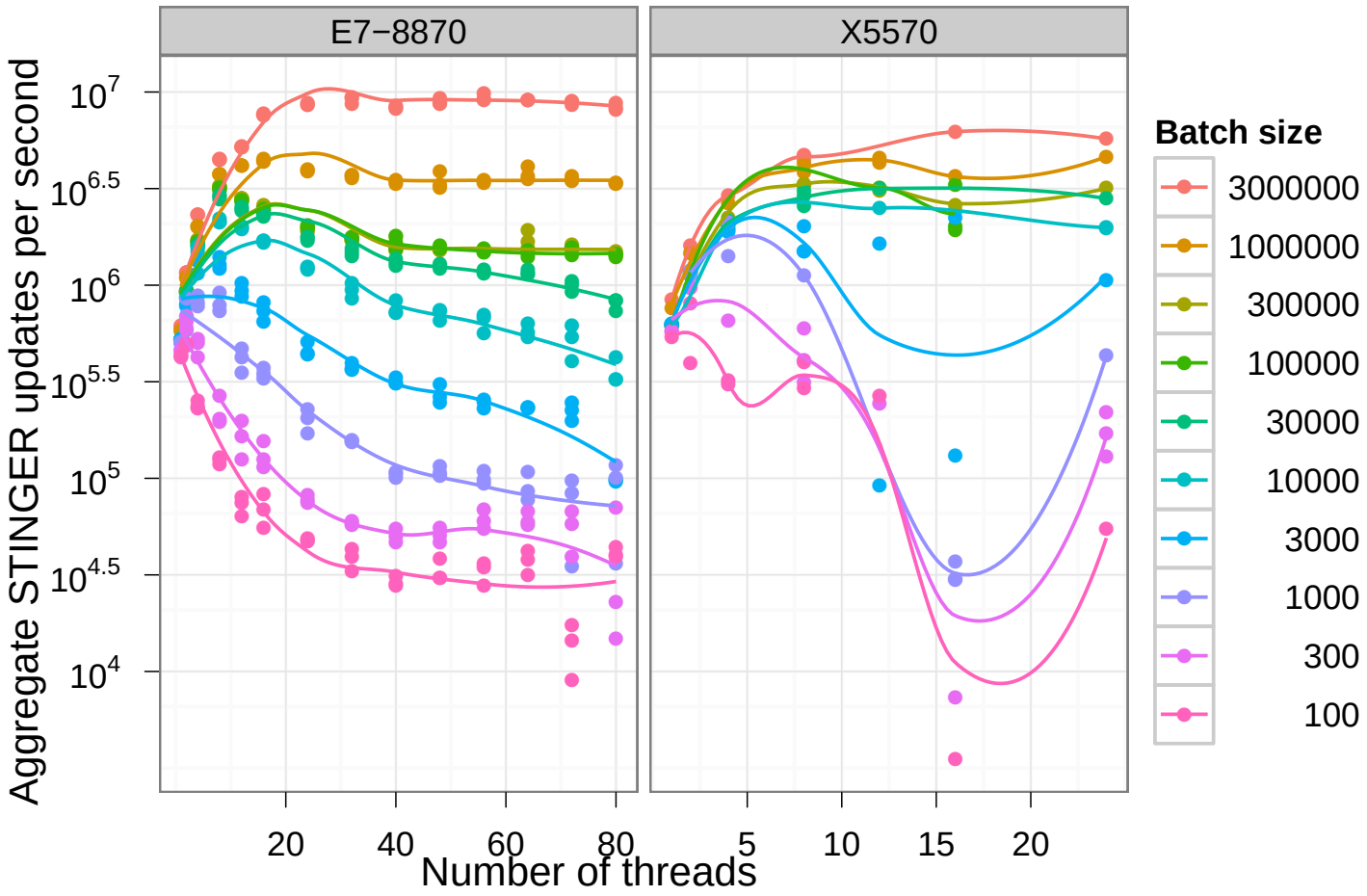

Unless otherwise noted

Line	Model	Speed (GHz)	Sockets	Cores
Nehalem	X5570	2.93	2	4
Westmere	X5650	2.66	2	6
Westmere	E7-8870	2.40	4	10

- Large Westmere, `mirasol`, loaned by Intel (*thank you!*)
- All memory: 1067MHz DDR3, installed appropriately
- Implementations: OpenMP, gcc 4.6.1, Linux $\approx$ 3.2 kernel
- Artificial graph and edge stream generated by R-MAT [Chakrabarti, Zhan, & Faloutsos].
  - Scale x, edge factor $f \Rightarrow 2^x$ 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*.
- Portable: OpenMP, Cray XMT family

STINGER insertion / removal rates

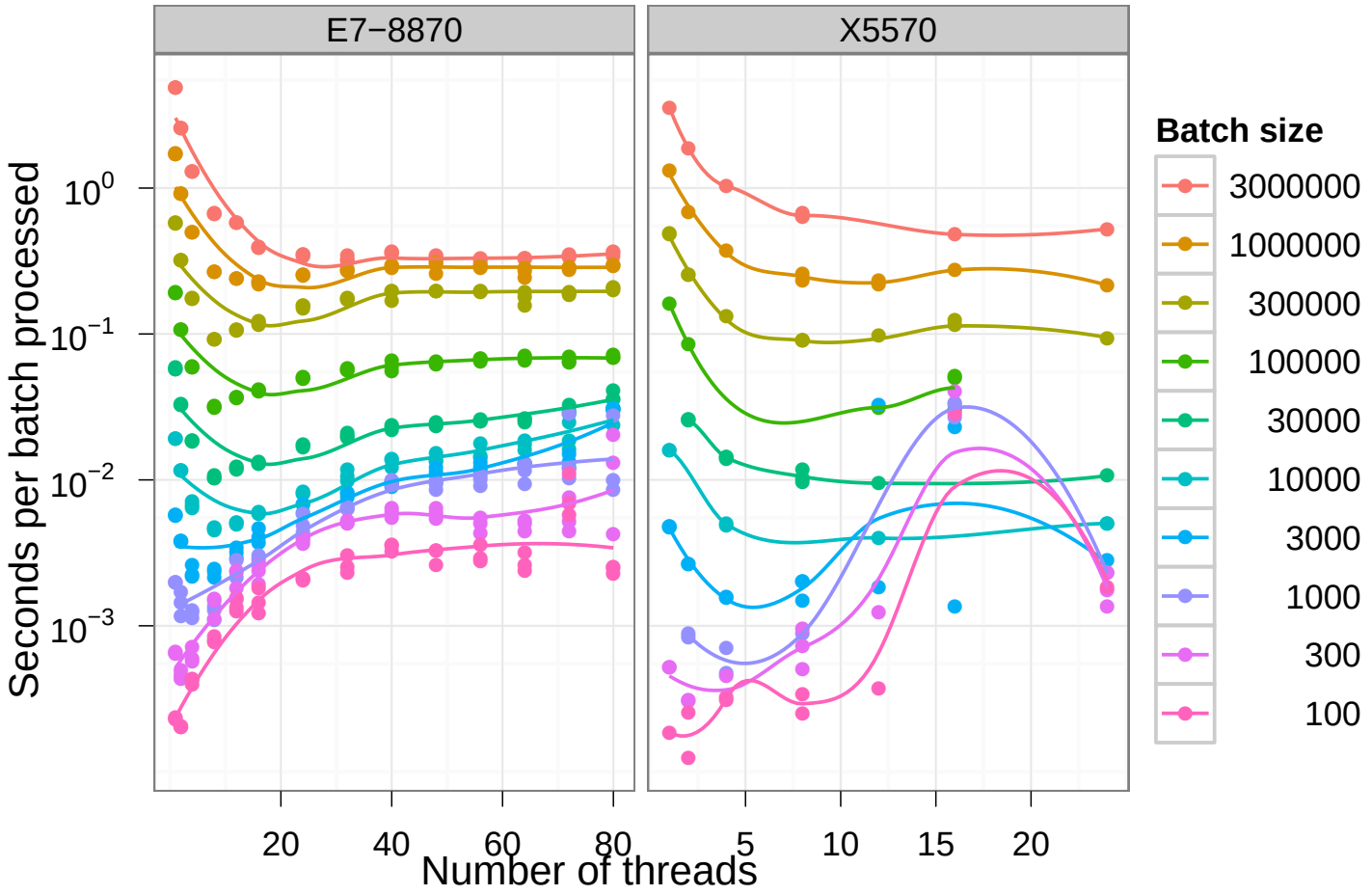
Edges per second



Georgia Tech College of Computing

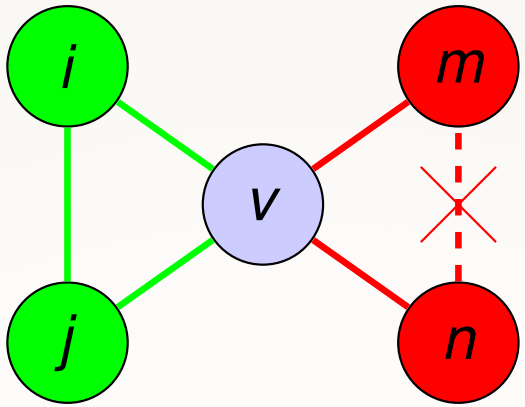
STINGER insertion / removal rates

Seconds per batch, “latency”



Clustering coefficients

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

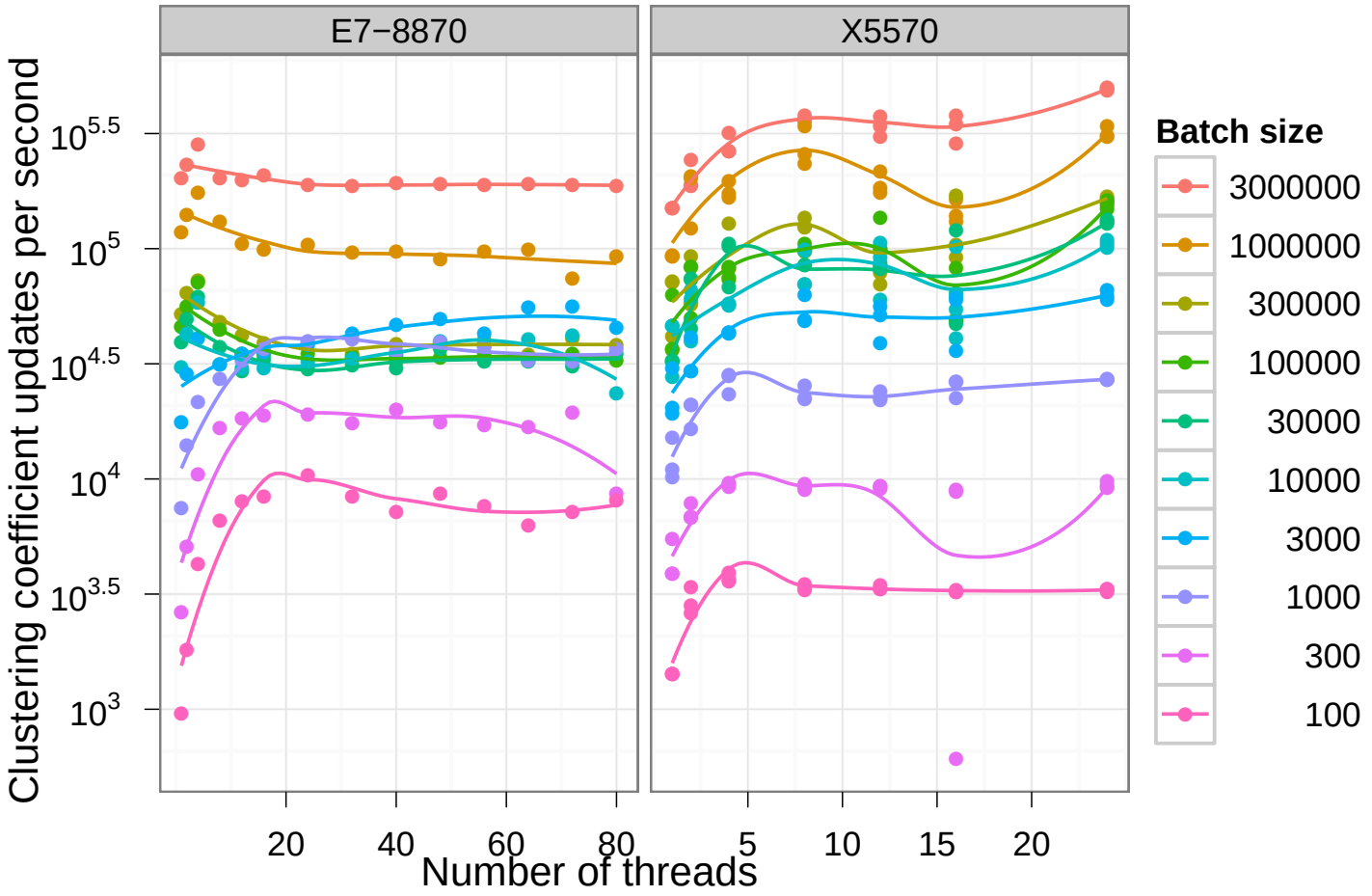


- Defined in terms of **triplets**.
- $i - v - j$ is a **closed triplet** (triangle).
- $m - v - n$ is an **open triplet**.
- Clustering coefficient:

  # of closed triplets / total # of triplets
- Locally around v or globally for entire graph.

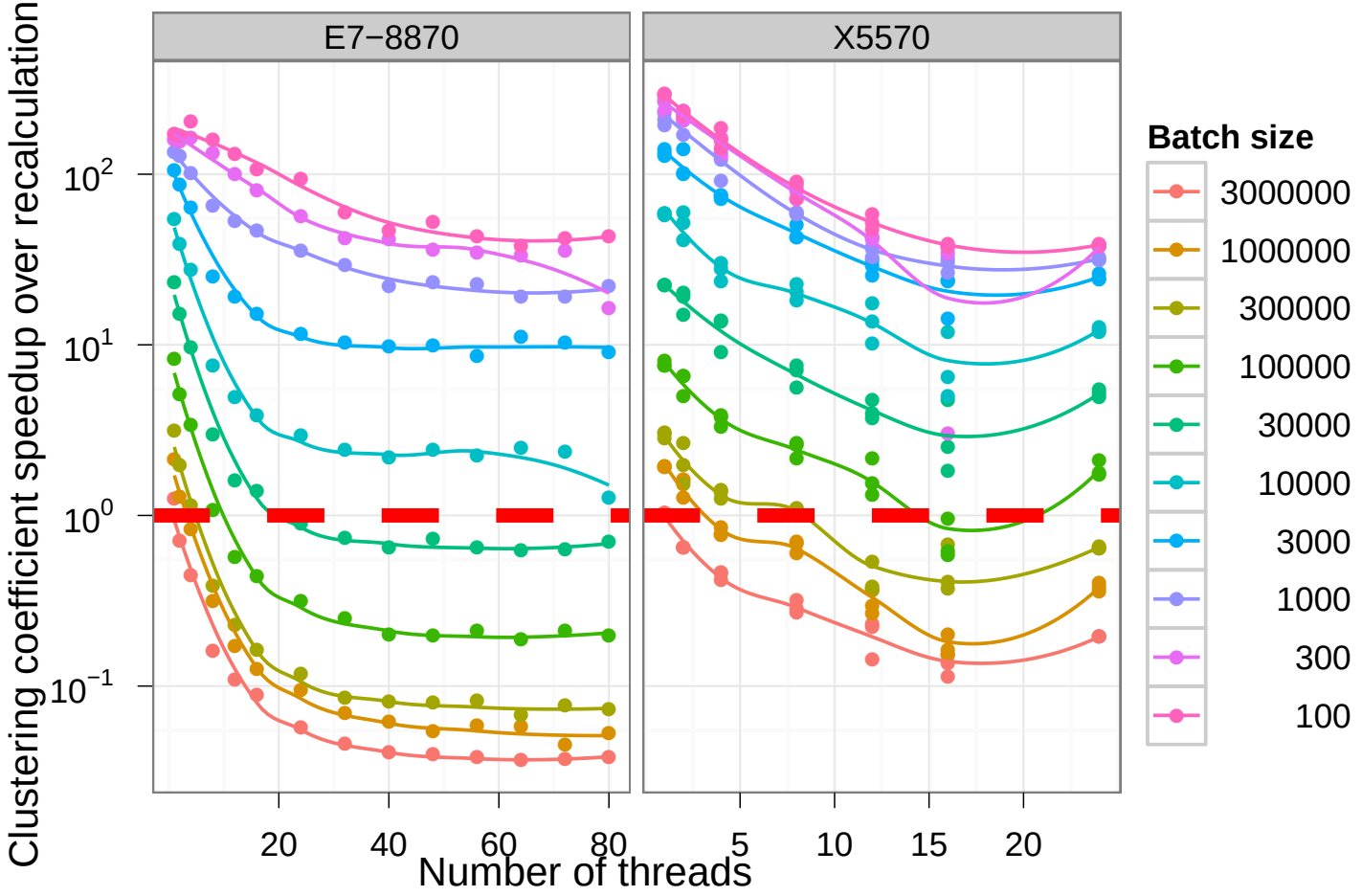
STINGER clustering coefficient rates

Edges per second



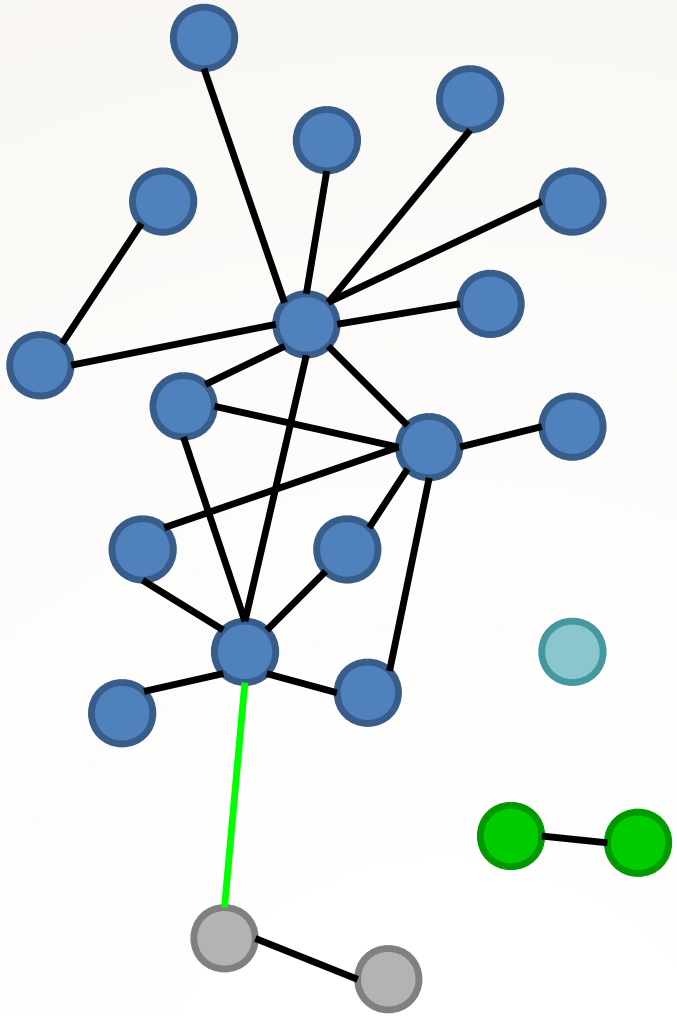
STINGER clustering coefficient rates

Speed-up over static recalculation



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...

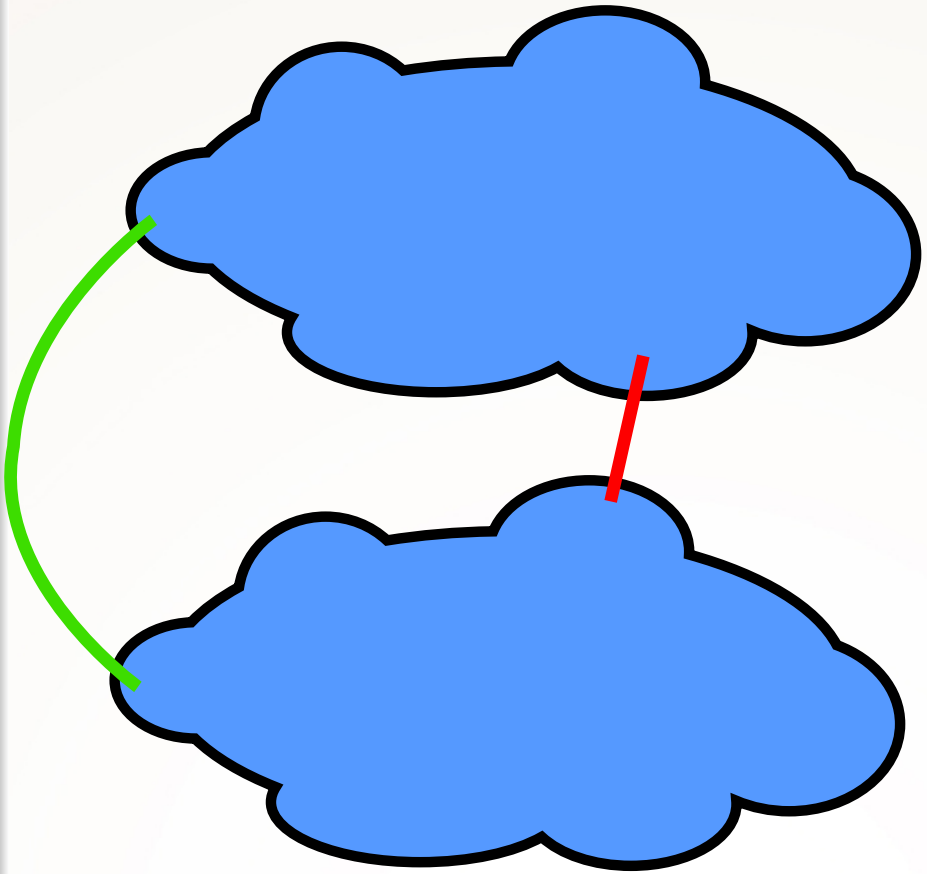


Georgia Tech College of Computing

Connected components: Deleted edges

The difficult case

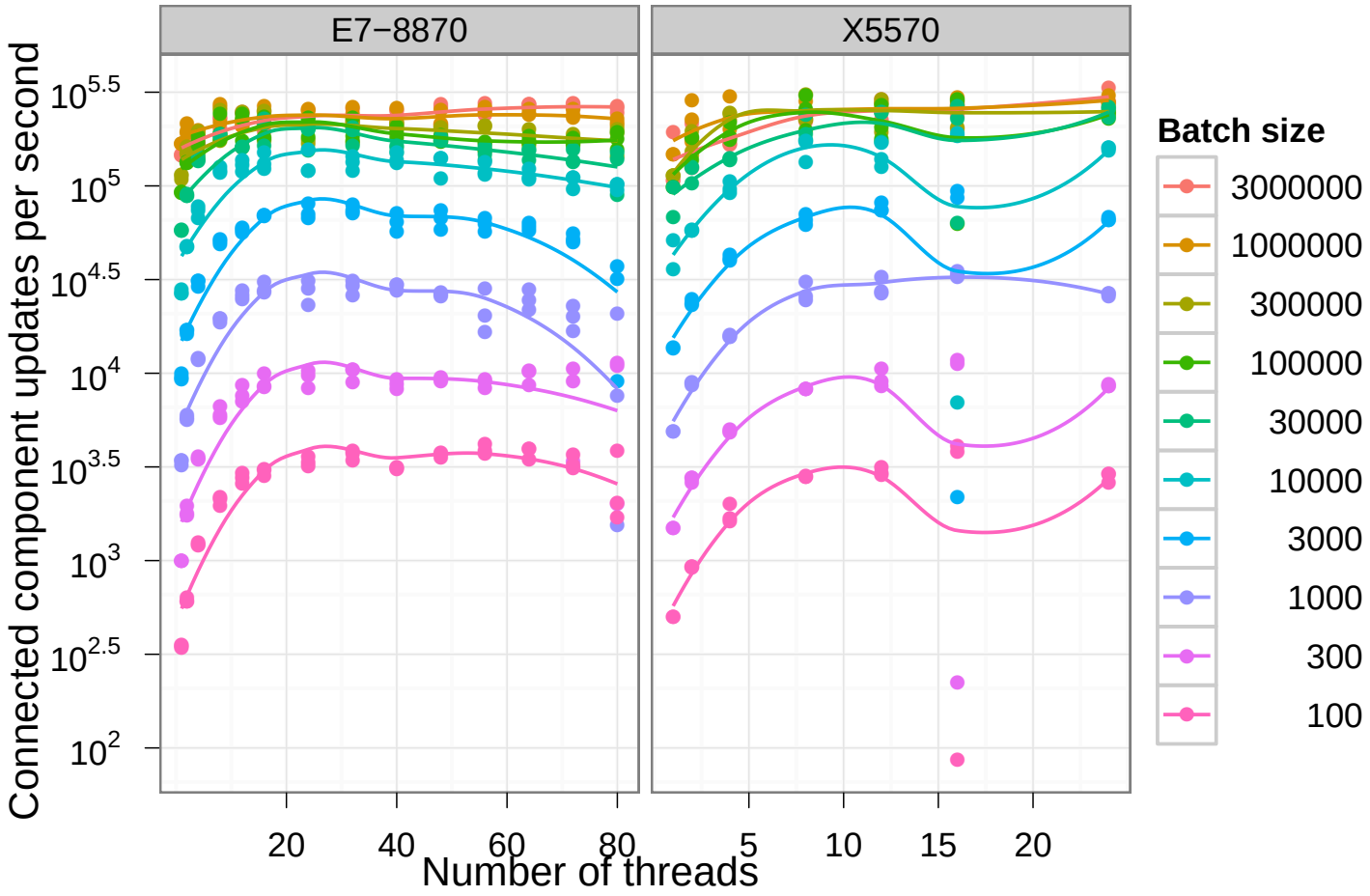
- Very few deletions matter.
- Maintain a spanning tree, & ignore deletion if
  1. not in spanning tree,
  2. endpoints share a common neighbor*,
  3. loose endpoint reaches root*.
- In the last two ($*$), also fix the spanning tree.
- Rules out 99.7% of deletions.
- (Long history, see related work in [MTAAP11].)



Georgia Tech College of Computing

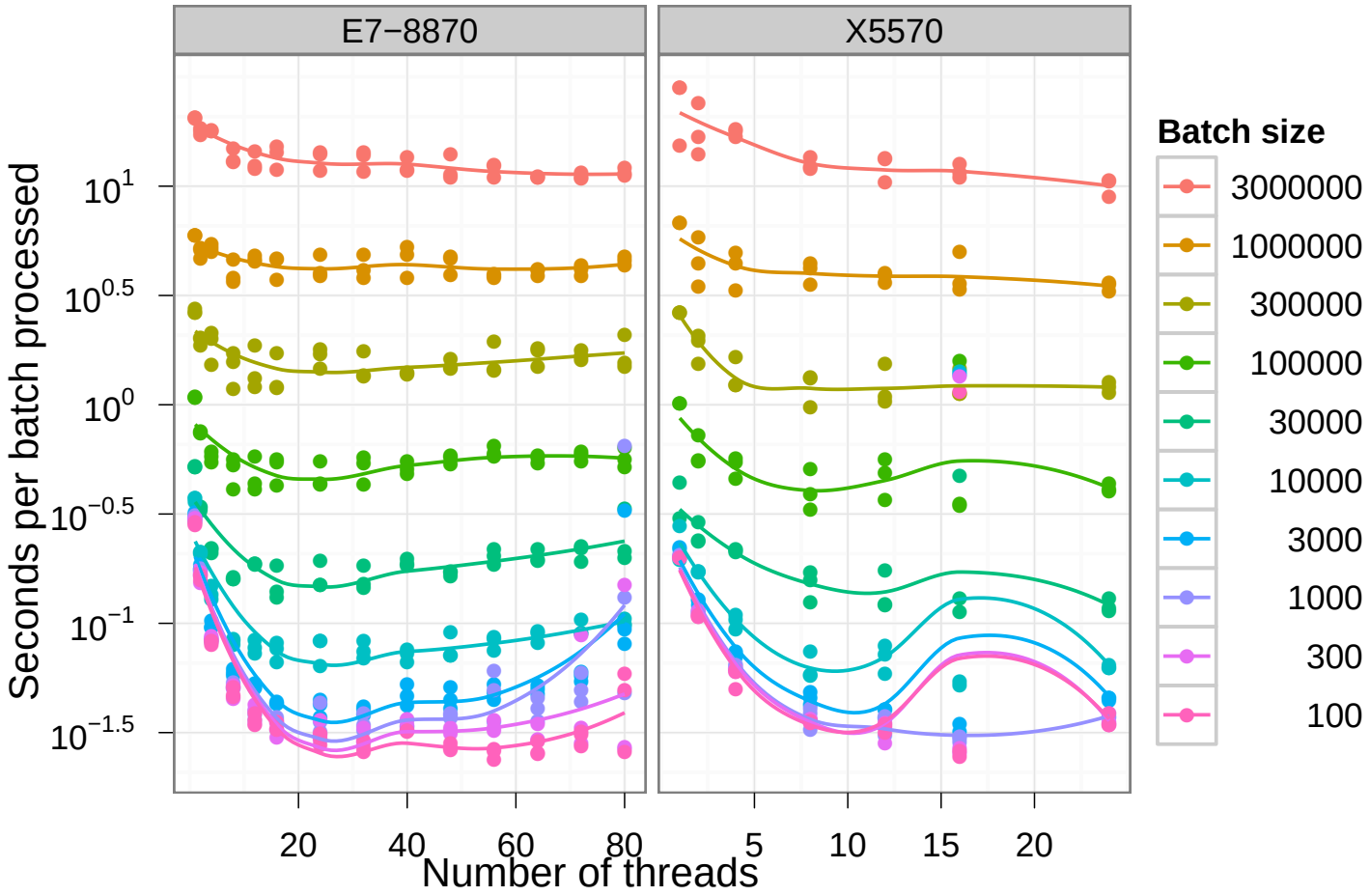
STINGER component monitoring rates

Edges per second



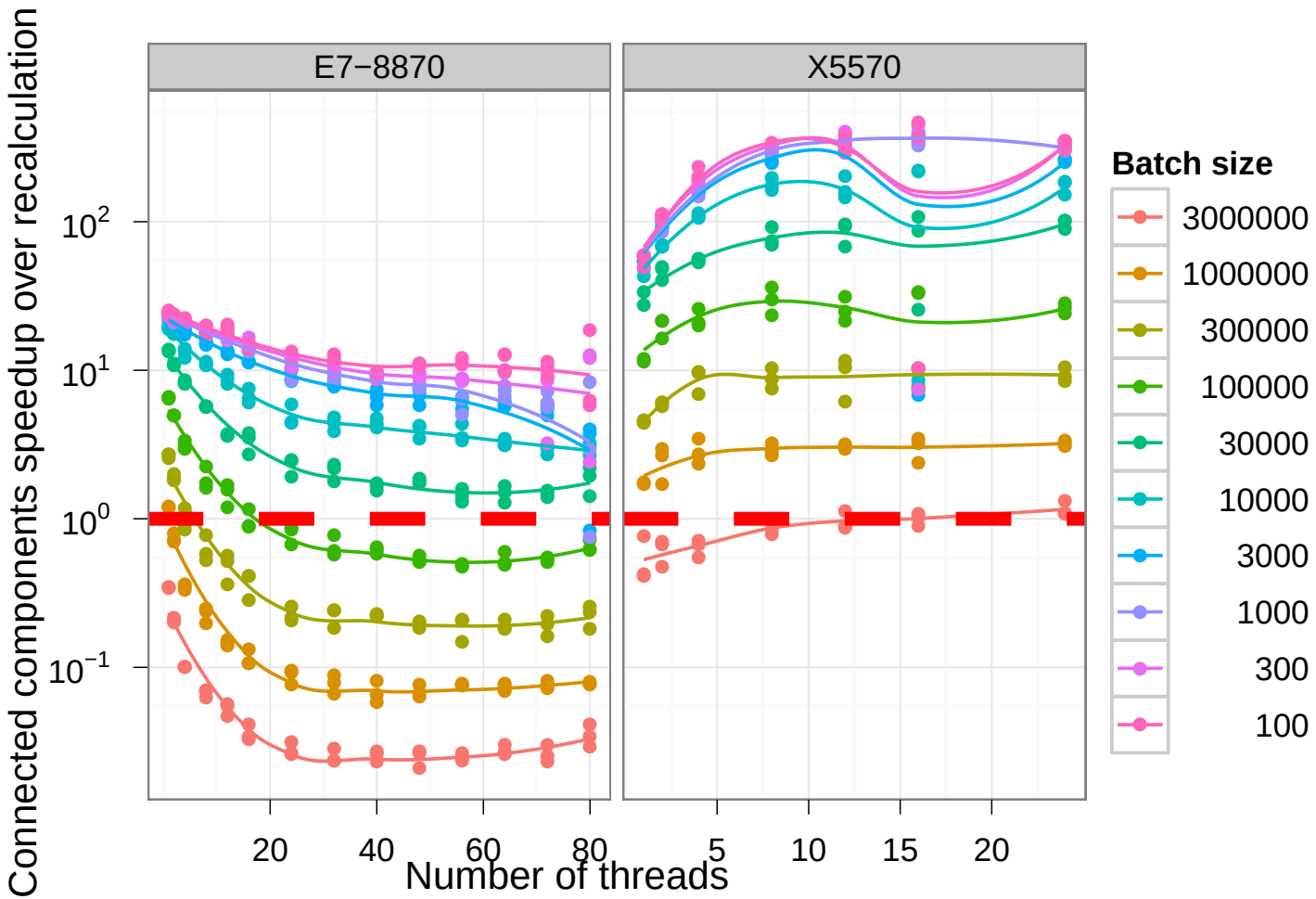
STINGER component monitoring rates

Seconds per batch, “latency”

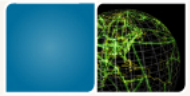


STINGER component monitoring rates

Speed-up over static recalculation



Community detection


Motivation

STING for streaming, graph-structured data on Intel-based platforms

World-leading community detection

- Defining community detection
- Our parallel, agglomerative algorithm
- Performance

Community-building interactions

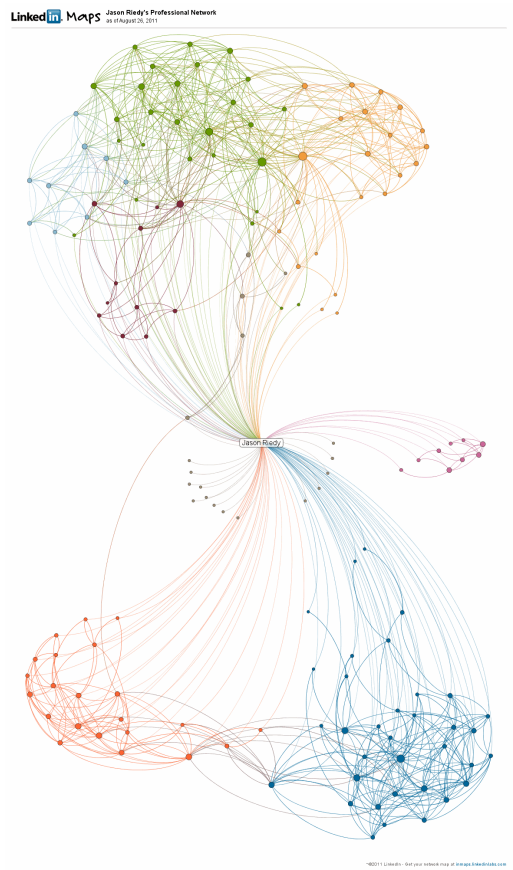
Plans and direction

Georgia Tech College of Computing

Community detection

What do *we* mean?

- Partition a graph's vertices into disjoint communities.
- A community locally maximizes some metric.
    - Modularity, conductance, ...
- Trying to capture that vertices are *more similar* within one community than between communities.



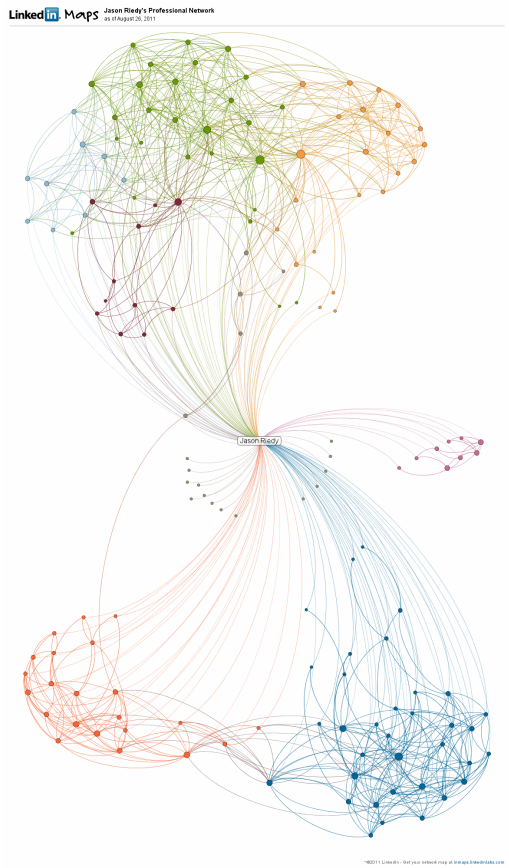
Jason's network via LinkedIn Labs

Georgia Tech College of Computing

Community detection

Assumptions

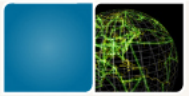
- Disjoint partitioning of vertices.
- There is no one unique answer.
  - Many metrics: NP-complete to optimize [Brandes, et al.].
  - Graph is lossy representation.
- Want an adaptable detection method.
- For rest, assume modularity [Newman].
- Important: Edge local scoring.



Jason's network via LinkedIn Labs

Georgia Tech College of Computing

Multi-threaded algorithm design points

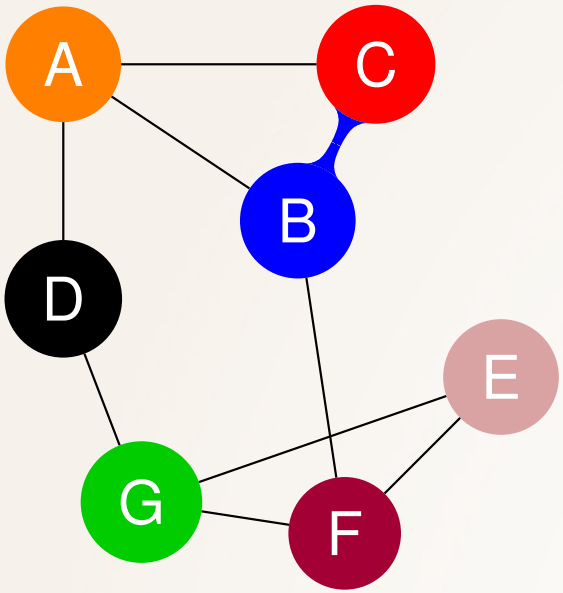

Targeting *massive* graphs

Multi-threading over shared memory, up to 2 GiB on Intel-based platforms.

A scalable multi-threaded graph analysis algorithm

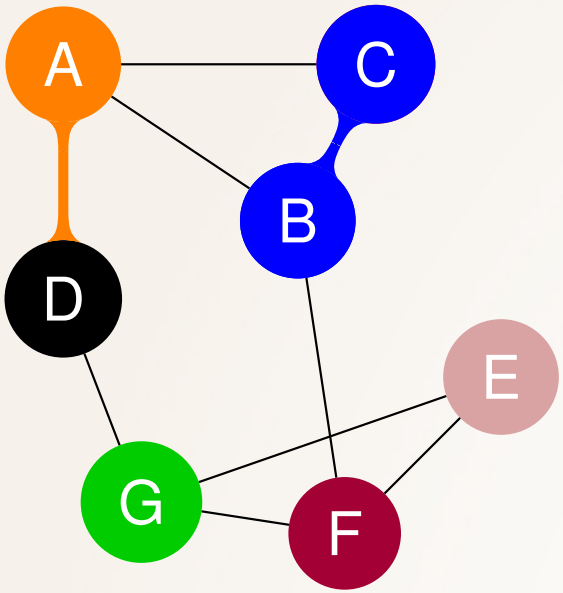
- ... avoids global locks and frequent global synchronization.
- ... distributes computation over edges rather than only vertices.
- ... works with data as local to an edge as possible.
- ... uses compact data structures that agglomerate memory references.

Sequential agglomerative method



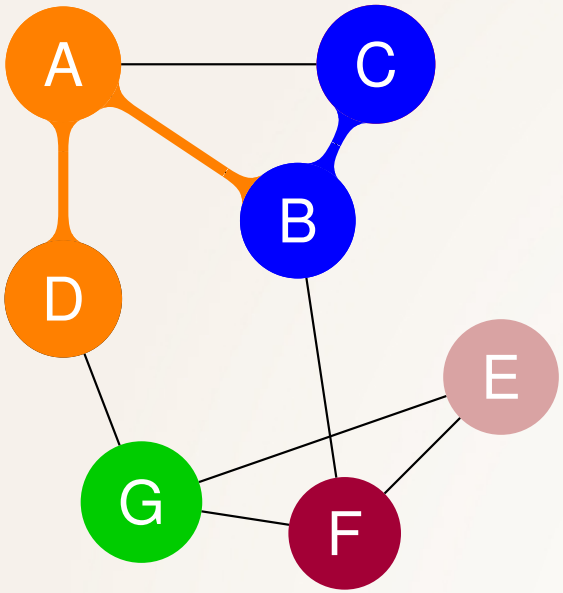
- A common method (*e.g.* [Clauset, et al.]) *agglomerates* vertices into communities.
- Each vertex begins in its own community.
- An edge is chosen to contract.
  - Merging maximally increases modularity.
  - *Priority queue.*
- Known often to fall into an $O(n^2)$ performance trap with modularity [Wakita and Tsurumi].

Sequential agglomerative method



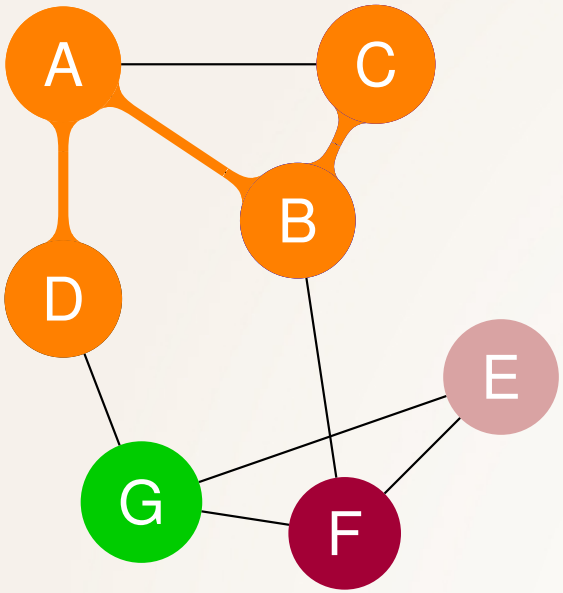
- A common method (*e.g.* [Clauset, et al.]) *agglomerates* vertices into communities.
- Each vertex begins in its own community.
- An edge is chosen to contract.
  - Merging maximally increases modularity.
  - *Priority queue.*
- Known often to fall into an $O(n^2)$ performance trap with modularity [Wakita and Tsurumi].

Sequential agglomerative method



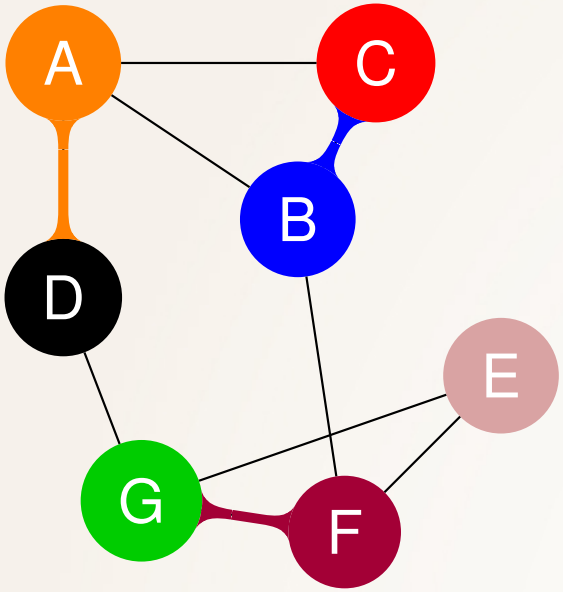
- A common method (*e.g.* [Clauset, et al.]) *agglomerates* vertices into communities.
- Each vertex begins in its own community.
- An edge is chosen to contract.
  - Merging maximally increases modularity.
  - *Priority queue.*
- Known often to fall into an $O(n^2)$ performance trap with modularity [Wakita and Tsurumi].

Sequential agglomerative method



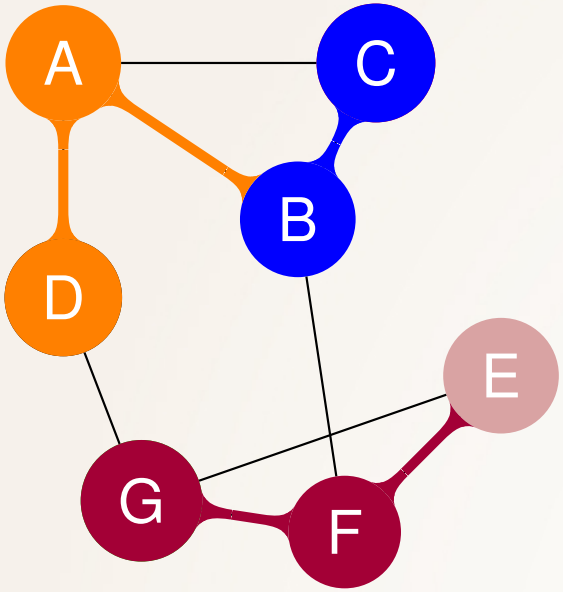
- A common method (*e.g.* [Clauset, et al.]) *agglomerates* vertices into communities.
- Each vertex begins in its own community.
- An edge is chosen to contract.
  - Merging maximally increases modularity.
  - *Priority queue.*
- Known often to fall into an $O(n^2)$ performance trap with modularity [Wakita and Tsurumi].

Parallel agglomerative method



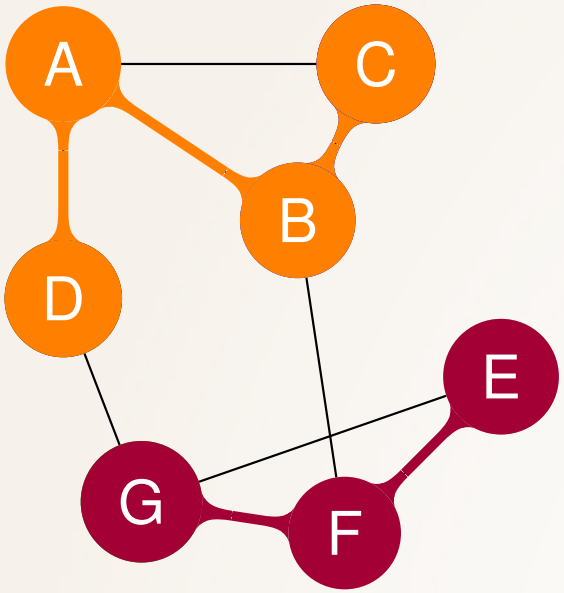
- We use a matching to avoid the queue.
- Compute a heavy weight, large matching.
  - Simple greedy algorithm.
  - Maximal matching.
  - Within factor of 2 in weight.
- Merge all matched communities at once.
- Maintains some balance.
- *Produces different results.*
- Agnostic to weighting, matching...
  - Can maximize modularity, minimize conductance.
  - Modifying matching permits easy exploration.

Parallel agglomerative method



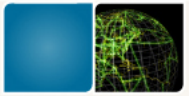
- We use a matching to avoid the queue.
- Compute a heavy weight, large matching.
  - Simple greedy algorithm.
  - Maximal matching.
  - Within factor of 2 in weight.
- Merge all matched communities at once.
- Maintains some balance.
- *Produces different results.*
- Agnostic to weighting, matching...
  - Can maximize modularity, minimize conductance.
  - Modifying matching permits easy exploration.

Parallel agglomerative method



- We use a matching to avoid the queue.
- Compute a heavy weight, large matching.
  - Simple greedy algorithm.
  - Maximal matching.
  - Within factor of 2 in weight.
- Merge all matched communities at once.
- Maintains some balance.
- *Produces different results.*
- Agnostic to weighting, matching...
  - Can maximize modularity, minimize conductance.
  - Modifying matching permits easy exploration.

Implementation: Data structures

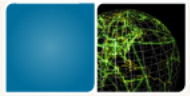




Extremely basic for graph $G = (V, E)$

- An array of $(i, j; w)$ weighted edge pairs, each i, j stored only once and packed, uses $3|E|$ space
- An array to store self-edges, $d(i) = w$, $|V|$
- A temporary floating-point array for scores, $|E|$
- A additional temporary arrays using $4|V| + 2|E|$ to store degrees, matching choices, offsets...

- Weights count number of agglomerated vertices or edges.
- Scoring methods (modularity, conductance) need only vertex-local counts.
- Storing an undirected graph in a symmetric manner reduces memory usage drastically and works with our simple matcher.

Implementation: Data structures

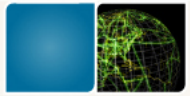




Extremely basic for graph $G = (V, E)$

- An array of $(i, j; w)$ weighted edge pairs, each i, j stored only once and packed, uses $3|E|$ space
- An array to store self-edges, $d(i) = w$, $|V|$
- A temporary floating-point array for scores, $|E|$
- A additional temporary arrays using $4|V| + 2|E|$ to store degrees, matching choices, offsets...

- Original ignored order in edge array, killed OpenMP.
- Roughly bucket edge array by first stored index. Non-adjacent CSR-like structure [MTAAP12].
- Hash i, j to determine order. Scatter among buckets [MTAAP12].

Implementation: Routines


Three primitives: Scoring, matching, contracting

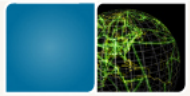
Scoring Trivial.

Matching Repeat until no ready, unmatched vertex:

1. For each unmatched vertex in parallel, find the best unmatched neighbor in its bucket.
2. Try to point remote match at that edge (lock, check if best, unlock).
3. If pointing succeeded, try to point self-match at that edge.
4. If both succeeded, yeah! If not and there was some eligible neighbor, re-add self to ready, unmatched list.

(Possibly *too* simple, but...)

Implementation: Routines

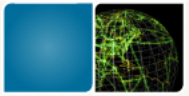




Contracting

1. Map each i, j to new vertices, re-order by hashing.
2. Accumulate counts for new i' bins, prefix-sum for offset.
3. Copy into new bins.

- Only synchronizing in the prefix-sum. That could be removed if I don't re-order the i', j' pair; haven't timed the difference.
- Actually, the current code copies twice... On short list for fixing.
- Binning as opposed to original list-chasing enabled Intel/OpenMP support with reasonable performance.

Performance summary

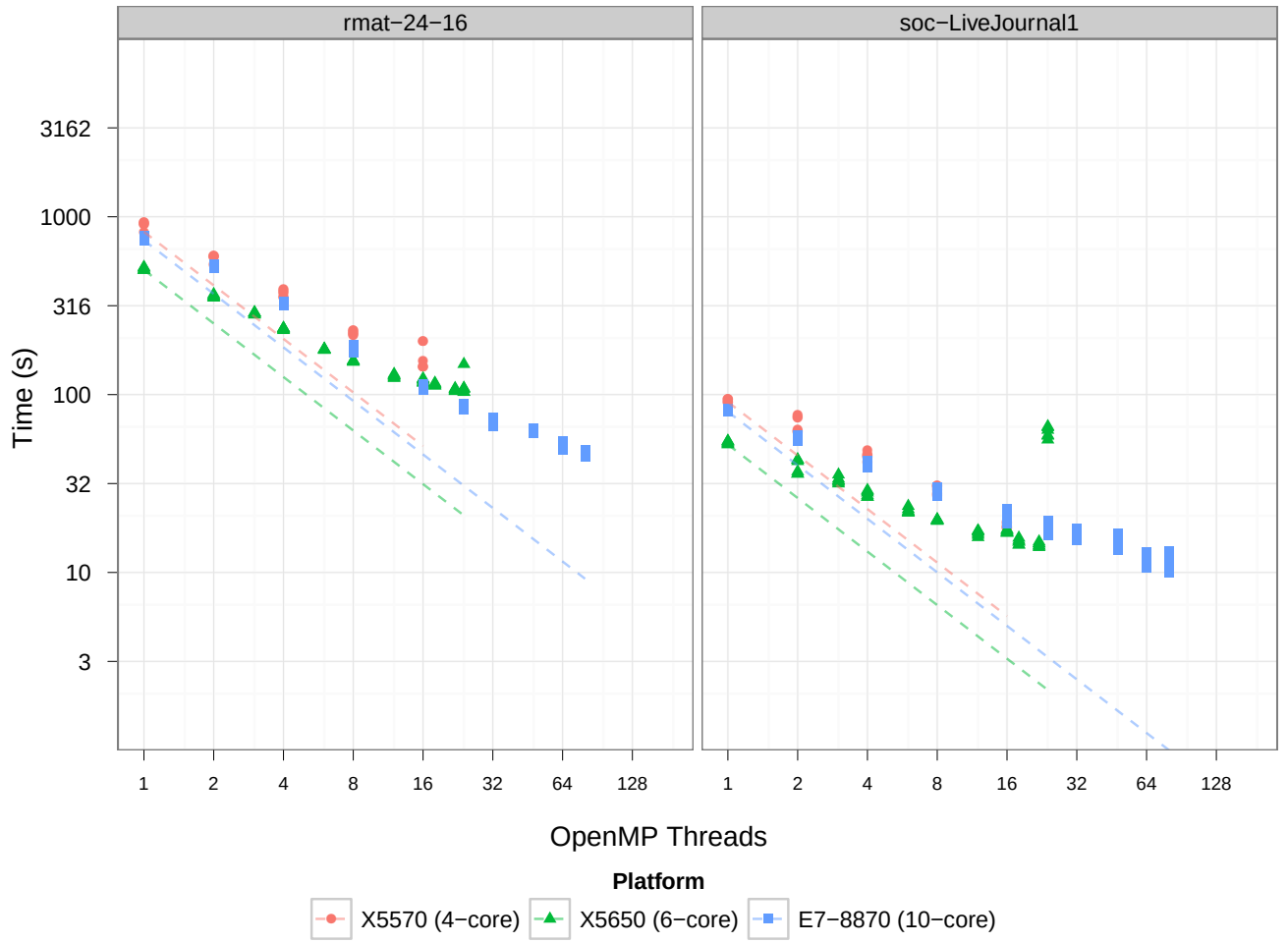

Two moderate-sized graphs, one large

Graph	$\|V\|$	$\|E\|$	Reference
rmat-24-16	15 580 378	262 482 711	[Chakrabarti, et al.] [Bader, et al.]
soc-LiveJournal1	4 847 571	68 993 773	[Leskovec]
uk-2007-05	105 896 555	3 301 876 564	[Boldi, et al.]

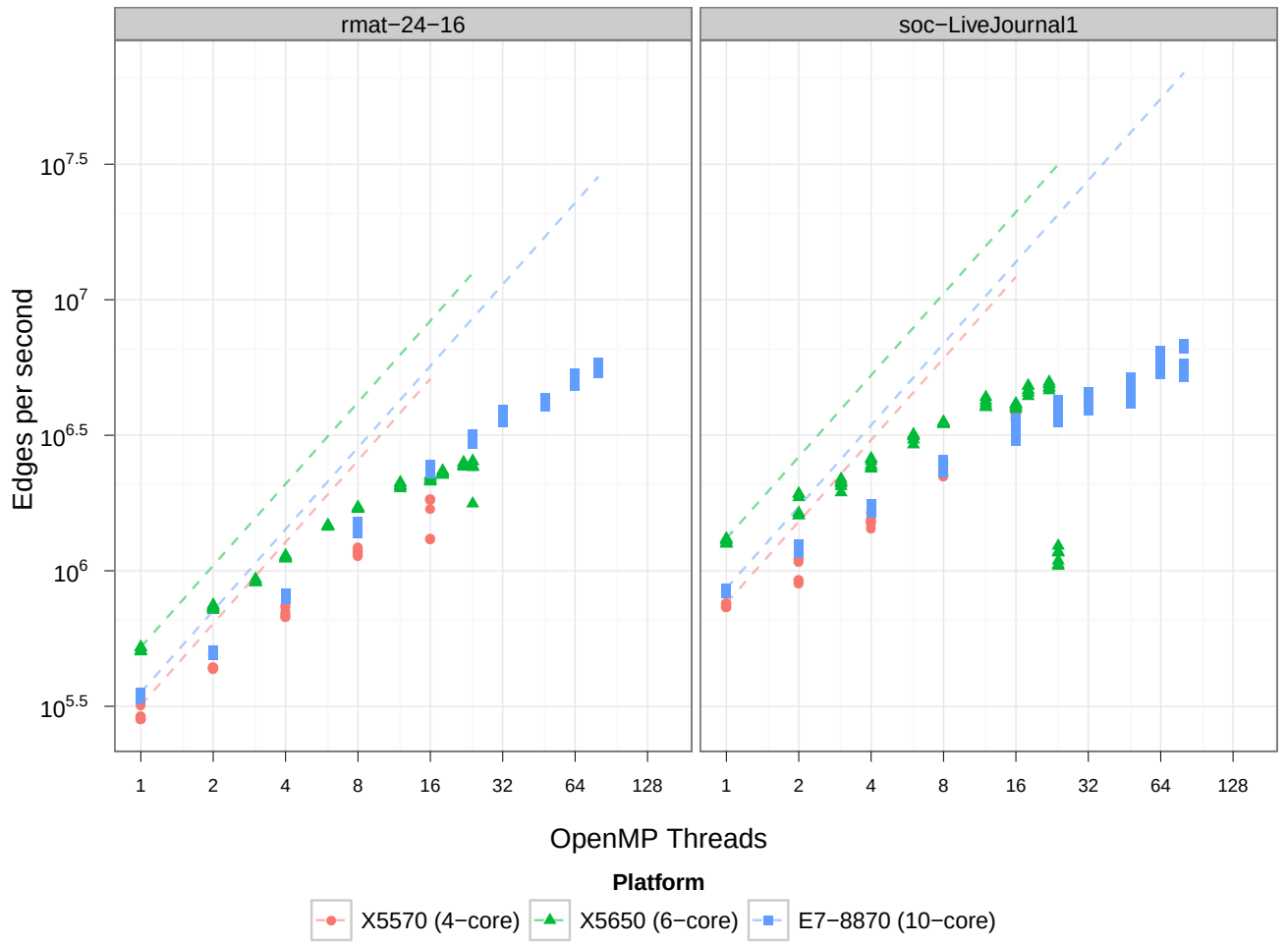
Peak processing rates in edges/second

Platform	rmat-24-16	soc-LiveJournal1	uk-2007-05
X5570	1.83×10^6	3.89×10^6	
X5650	2.54×10^6	4.98×10^6	
E7-8870	5.86×10^6	6.90×10^6	6.54×10^6
XMT	1.20×10^6	0.41×10^6	
XMT2	2.11×10^6	1.73×10^6	3.11×10^6

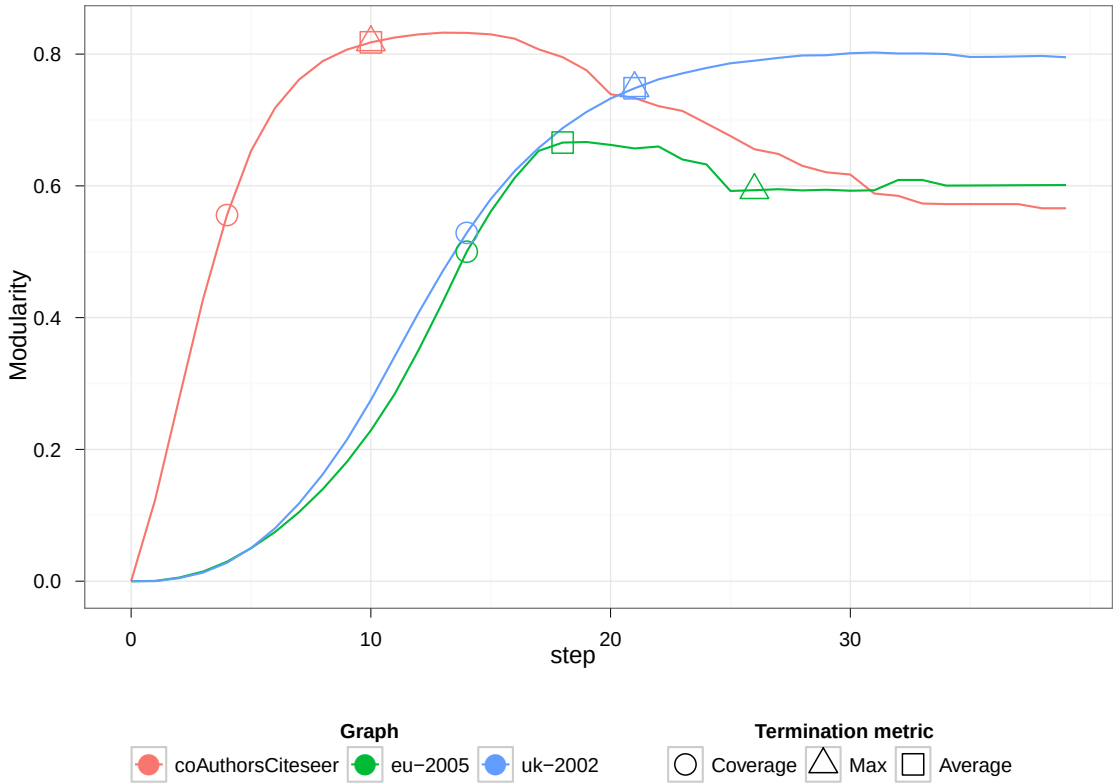
Performance: Time to solution



Performance: Rate (edges/second)

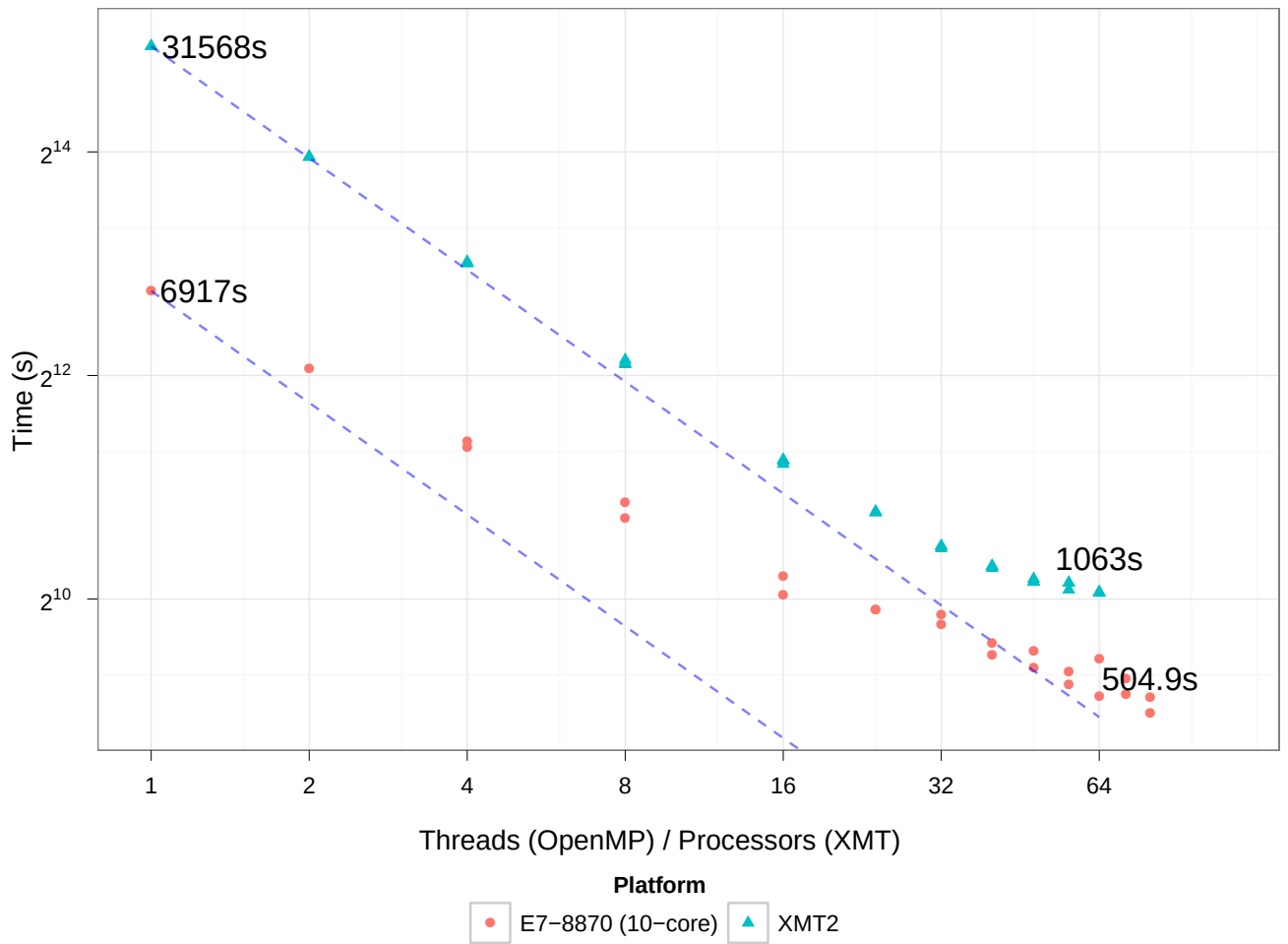


Performance: Modularity

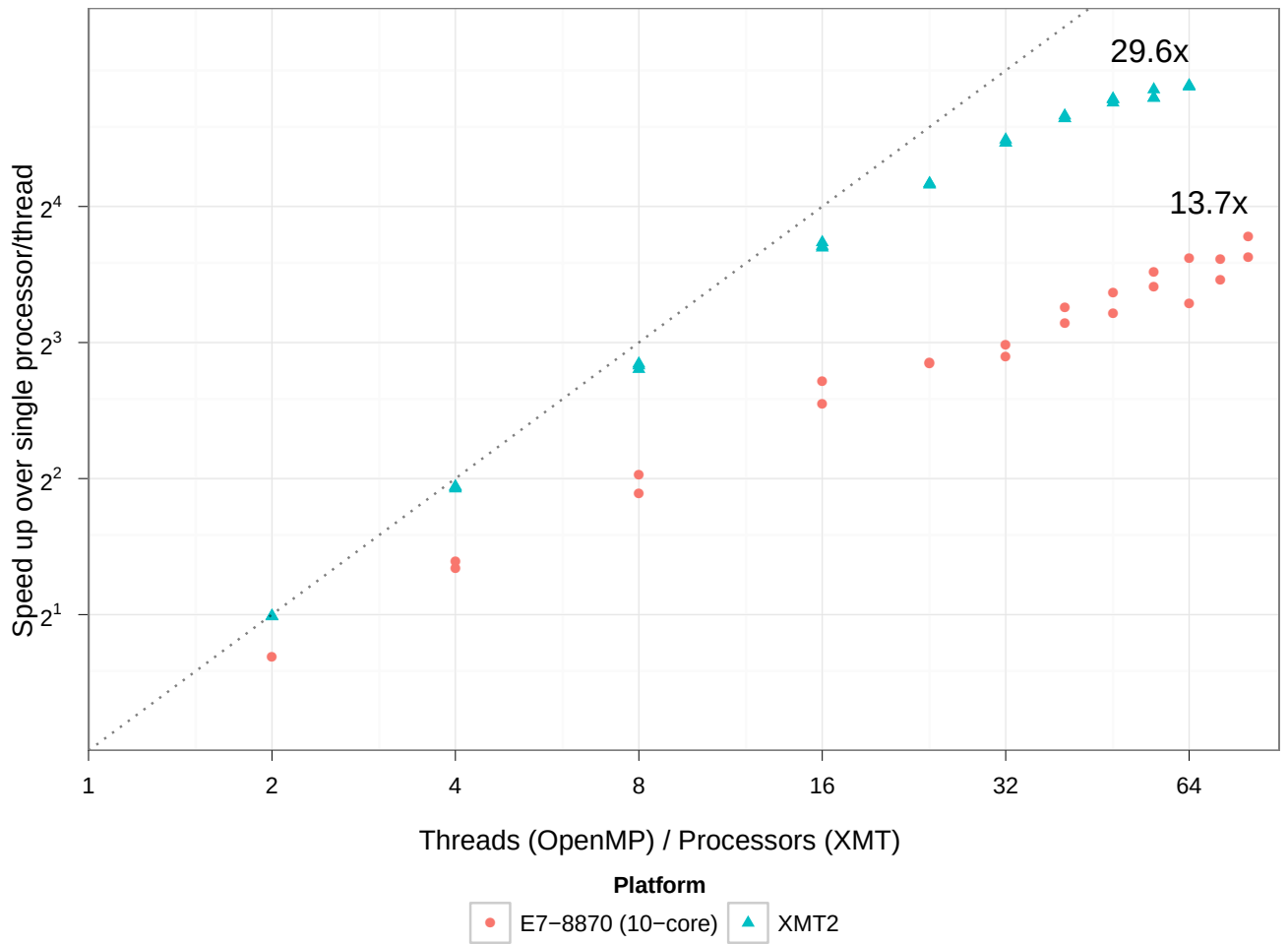


- Timing results: Stop when coverage ≥ 0.5 (communities cover $1/2$ edges).
- More work $\Rightarrow$ higher modularity. Choice up to application.

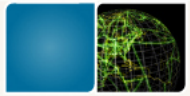
Performance: Large-scale time



Performance: Large-scale speedup



Building a community around streaming, graph-structured data


Motivation

STING for streaming, graph-structured data on Intel-based platforms

World-leading community detection

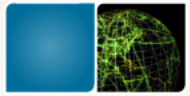
Community-building interactions

- External STING users∗
- Graph500 (synergistic)
- Conferences, DIMACS Challenge, etc.

Plans and direction

Georgia Tech College of Computing

Graph500


Not part of *this* project, but...

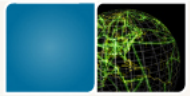
To keep people apprised:

- Adding single-source shortest paths benchmark.
- Re-working the generator to be more scalable, vectorized, *etc*.
- Addressing some generator parameter issues. Fewer duplicate edges.
- Improving shared-memory reference implementation performance...
- (Possibly adding an OpenCL reference implementation.)

`mirasol` is very useful for development and reference tuning.

Georgia Tech College of Computing

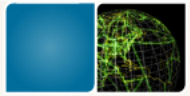
10th DIMACS Implementation Challenge


Graph partitioning and clustering

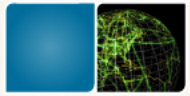
- Georgia Tech assisted in organizing the 10th DIMACS Implementation Challenge.
- Co-sponsored by Intel.
- Community detection implementation on `mirasol` won the Mix Challenge combining speed and multiple clustering criteria.
  - One of only **two** implementations that tackled `uk-2007-05` with 106 million vertices and 3.3 *billion* edges.
  - Other implementation required multiple days across a cluster.
- Next closest for performance used Intel's TBB (Fagginger Auer & Bisseling) or CUDA (small examples).

Conferences and minisymposiums


- GT hosted the bi-annual SIAM Parallel Processing conference.
  - Included a well-attended minisymposium organized by Riedy & Meyerhenke on “Parallel Analysis of Massive Social Networks,” including KDT work from UCSB.
- Submitting a massive graph minisymposium for SIAM CSE 2013 (25 Feb – 1 Mar)... Looking for speakers.
- List of presentations appended to slides, too many to include.
  - 10 conference presentations / invited talks / posters
  - 3 tutorials
  - 5 accepted (refereed) publications, 1 in submission

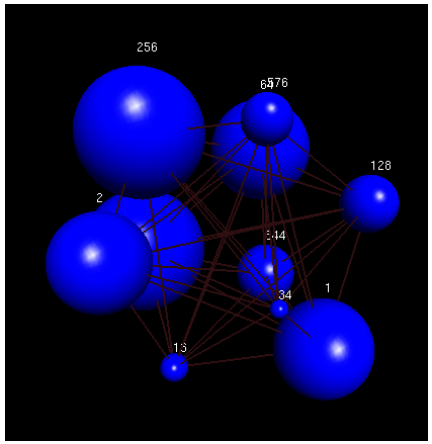
Plans and direction

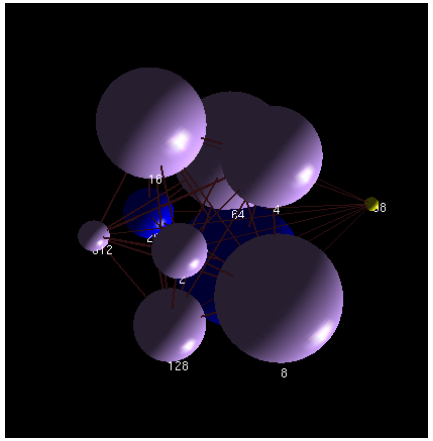




Georgia Tech College of Computing

Plans

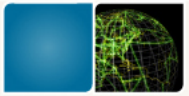
- Finish the split into a client/server model, including subgraph visualization, looking into KDT interface.





- Integrate performance counters, build a performance model / monitor.
  - Evaluate Intel features for bottlenecks (vectors, better NUMA allocation, *etc.*)
- Incorporate community detection, seed set expansion.

STING on-line resources


Home page `http://www.cc.gatech.edu/stinger/`

Download `.../downloads.php`

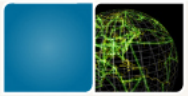
Reference guide `.../doxygen/`

General tutorial `(URL too long, linked from above)`

The current release of STING includes the following static and dynamic analysis kernel examples:

- multi-platform benchmarks for edge insertion and removal (dynamic),
- breadth-first search (static),
- clustering and transitivity coefficients (static and *dynamic*), and
- connected components (static and *dynamic*).

Community detection

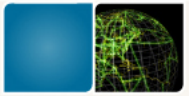

Home page `http://www.cc.gatech.edu/~jriedy/community-detection/`

Git source repository `http://www.cc.gatech.edu/~jriedy/gits/community-el.git`

- "Experimental" and subject to extreme change.
- Current version packaged as a separate, easy to use / import executable.
- Being adapted into STING.

Georgia Tech College of Computing

Presentations and tutorials I

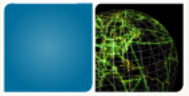




David A. Bader.
Opportunities and challenges in massive data-intensive computing,.
In *9th International Conference on Parallel Processing and Applied Mathematics (PPAM11)*, September 2011.
(invited keynote).

David A. Bader.
Opportunities and challenges in massive data-intensive computing.
Workshop on Parallel Algorithms and Software for Analysis of Massive Graphs (ParGraph), December 2011.
(invited keynote).

Georgia Tech College of Computing

Presentations and tutorials II

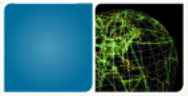




Jason Riedy, David Ediger, David A. Bader, and Henning Meyerhenke.
Tracking structure of streaming social networks.
2011 Graph Exploitation Symposium hosted by MIT Lincoln Labs, August 2011.
(presentation).

David A. Bader.
Fourth Graph500 results.
Graph500 Birds of a Feather, 27th International Supercomputing Conference (ISC), June 2012.
(presentation).

Georgia Tech College of Computing

Presentations and tutorials III

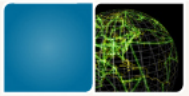




David A. Bader.
Massive data analytics using heterogeneous computing.
In *21st International Heterogeneity in Computing Workshop (HCW 2012), held in conjunction with The International Parallel and Distributed Processing Symposium (IPDPS 2012)*, May 2012.
(invited keynote).

David A. Bader.
Opportunities and challenges in massive data-intensive computing.
Distinguished Lecture, May 2012.
(invited presentation).

Georgia Tech College of Computing

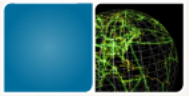
Presentations and tutorials IV


David A. Bader.
Opportunities and challenges in massive data-intensive computing.
From Data to Knowledge: Machine-Learning with Real-time and Streaming Applications, May 2012.
(invited presentation).

David A. Bader.
Opportunities and challenges in massive data-intensive computing.
NSF Workshop on Research Directions in the Principles of Parallel Computation, June 2012.
(invited presentation).

Presentations and tutorials V

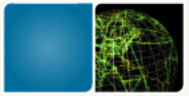

David Ediger, Jason Riedy, Rob McColl, and David A. Bader.

Parallel programming for graph analysis.
In *17th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP)*, New Orleans, LA, February 2012.
(tutorial).

Henning Meyerhenke, E. Jason Riedy, and David A. Bader.
Parallel community detection in streaming graphs.
SIAM Parallel Processing for Scientific Computing, February 2012.
(presentation).

Georgia Tech College of Computing

Presentations and tutorials VI


E. Jason Riedy, David Ediger, Henning Meyerhenke, and David A. Bader.
Sting: Software for analysis of spatio-temporal interaction networks and graphs.
SIAM Parallel Processing for Scientific Computing, February 2012.
(poster).

E. Jason Riedy and Henning Meyerhenke.
Scalable algorithms for analysis of massive, streaming graphs.
SIAM Parallel Processing for Scientific Computing, February 2012.
(presentation).

Georgia Tech College of Computing

Presentations and tutorials VII

Anita Zakrzewska and Oded Green.
Stinger tutorial.
Presented to GTRI and SAIC software engineers, July 2012.
(tutorial).

Publications I

David Ediger, Rob McColl, Jason Riedy, and David A. Bader.

Stinger: High performance data structure for streaming graphs.
In *IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, MA, September 2012.
(to appear).

Xin Liu, Pushkar Pande, Henning Meyerhenke, and David A. Bader.
Pasqual: A parallel de novo assembler for next generation genome sequencing.
IEEE Transactions on Parallel and Distributed Systems, 2012.
(to appear).

Publications II

E. Jason Riedy, David A. Bader, and Henning Meyerhenke.
Scalable multi-threaded community detection in social networks.
In *6th Workshop on Multithreaded Architectures and Applications (MTAAP), held in conjunction with The International Parallel and Distributed Processing Symposium (IPDPS 2012)*, May 2012.
(9/15 papers accepted, 60% acceptance).

E. Jason Riedy, Henning Meyerhenke, David Ediger, and David A. Bader.
Parallel community detection for massive graphs.
In *10th DIMACS Implementation Challenge - Graph Partitioning and Graph Clustering*. (workshop paper), Atlanta, Georgia, February 2012.
Won first place in the Mix Challenge and Mix Pareto Challenge.

Georgia Tech College of Computing

Publications III

Jason Riedy, Henning Meyerhenke, David A. Bader, David Ediger, and Timothy G. Mattson.
Analysis of streaming social networks and graphs on multicore architectures.
In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Kyoto, Japan, March 2012.

References I


D. Bader, J. Gilbert, J. Kepner, D. Koester, E. Loh, K. Madduri, W. Mann, and T. Meuse.
HPCS SSCA#2 Graph Analysis Benchmark Specifications v1.1.
Jul. 2005.


D.A. Bader and J. McCloskey.
Modularity and graph algorithms.
Presented at UMBC, September 2009.


J.W. Berry., B. Hendrickson, R.A. LaViolette, and C.A. Phillips.
Tolerating the community detection resolution limit with edge weighting.
CoRR **abs/0903.1072** (2009).

References II


P. Boldi, B. Codenotti, M. Santini, and S. Vigna.
Ubicrawler: A scalable fully distributed web crawler.
Software: Practice & Experience, vol. 34, no. 8, pp. 711–726, 2004.


U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner.
On modularity clustering.
In *IEEE Trans. Knowledge and Data Engineering*, vol. 20, no. 2, pp. 172–188, 2008.


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.

Georgia Tech College of Computing

References III

A. Clauset, M. Newman, and C. Moore.
Finding community structure in very large networks.
Physical Review E, vol. 70, no. 6, p. 66111, 2004.

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.

J. Leskovec.
Stanford large network dataset collection.
At `http://snap.stanford.edu/data/`, Oct. 2011.

M. Newman.
Modularity and community structure in networks.
In *Proc. of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

Georgia Tech College of Computing

References IV

K. Wakita and T. Tsurumi.
Finding community structure in mega-scale social networks.
CoRR, vol. abs/cs/0702048, 2007.

D. J. Watts and S. H. Strogatz.
Collective dynamics of ‘small-world’ networks.
Nature, 393(6684):440–442, Jun 1998.

Georgia Tech College of Computing