Characterizing and Analyzing Massive Spatio-Temporal Graphs

David A. Bader, David Ediger,

Karl Jiang, & Jason Riedy

Georgia Tech College of Computing

Pacific Northwest NATIONAL LABORATORY

Outline

- Motivation
  - Explosion of Social and Other Networks
- GraphCT: A Massive Graph Characterization Toolkit
  - Provides summaries for graphs with **billions** of vertices & edges
  - Tuned for the Cray XMT
- A Design for Streaming Graph Analysis
  - STINGER: Data Structure for Changing Graphs
  - Initial Experiments with Streaming Clustering Coefficients
- Future Directions
  - Hierarchy of Interesting Temporal Graph Queries

Center for Advanced Supercomputing Software for Multithreaded Architectures (CASS-MT)

Objective

To design software for the analysis of massive-scale spatio-temporal interaction networks using multithreaded architectures such as the Cray XMT. The Center launched in July 2008 and is led by Pacific-Northwest National Laboratory.

Description

We are designing and implementing advanced, scalable algorithms for static and dynamic graph analysis, including generalized *k*-betweenness centrality and dynamic clustering coefficients.

Highlights

On a 64-processor Cray XMT, *k*-betweenness centrality scales nearly linearly (58.4x) on a graph with 16M vertices and 134M edges. Initial streaming clustering coefficients handle around 200k updates/sec on a similarly sized graph.

Pacific Northwest NATIONAL LABORATORY

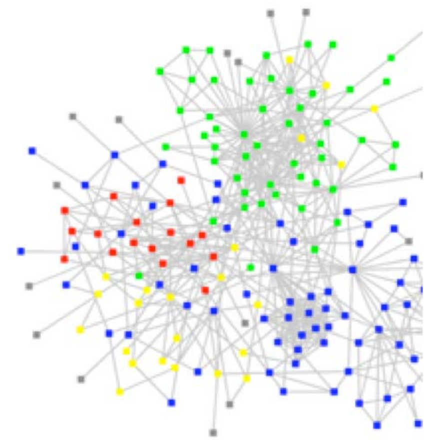


Image Courtesy of Cray, Inc.

Our research is focusing on temporal analysis, answering questions about changes in global properties (*e.g.* diameter) as well as local structures (communities, paths).

David A. Bader (PI)
David Ediger, Karl Jiang, Jason Riedy
Pacific Northwest National Laboratory

Georgia Institute of Technology

NSF Computing Research Infrastructure:
Development of a Research Infrastructure for Multithreaded Computing Community Using Cray Eldorado Platform

The Cray XMT system serves as an ideal platform for the research and development of algorithms, data sets, libraries, languages, tools and simulators for applications that benefit from large numbers of threads, massively data intensive, *sparse-graph* problems that are difficult to parallelize using conventional message-passing on clusters.

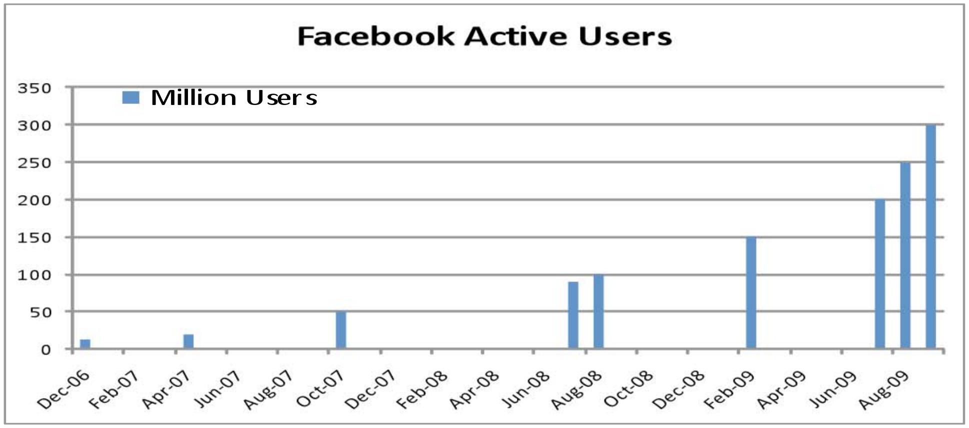
- A shared community resource capable of efficiently running, in experimental and production modes, complex programs with thousands of threads in shared memory
- Assembling software infrastructure for developing and measuring performance of programs running on the hardware
- Building stronger ties between the people themselves, creating ways for researchers at the partner institutions to collaborate and communicate their findings to the broader community

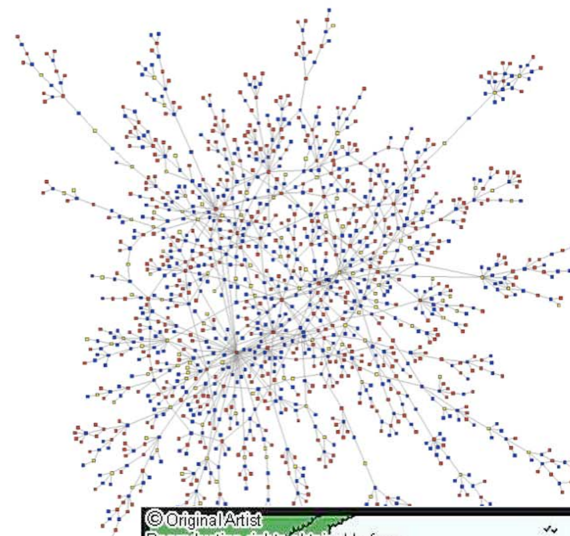
Collaborators include: University of Notre Dame, University of Delaware, University of California Santa Barbara, CalTech, University of California Berkeley and Sandia National Laboratories

David A. Bader (PI)
Jeffrey Vetter (co-PI)
NSF CNS-0708307

Massive Social Networks

facebook has more than 300 million active users





Suddenly, the flock became suspicious:
How come the newcomer wasn't shorn?

Traditional graph partitioning often fails:

- **Topology**: Interaction graph is low-diameter, and has no good separators
- **Irregularity**: Communities are not uniform in size
- **Overlap**: individuals are members of one or more communities

Sample queries:

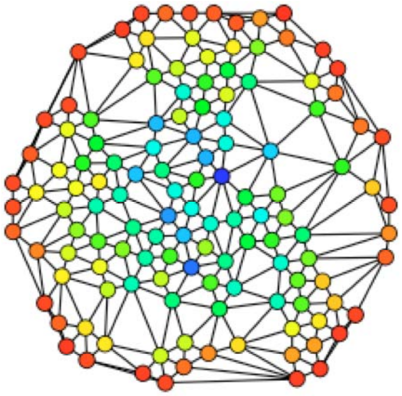
- **Allegiance switching**: identify entities that switch communities.
- **Community structure**: identify the genesis and dissipation of communities
- **Phase change**: identify significant change in the network structure

Limitations of Current Tools

- Graphs with millions of vertices are well beyond simple comprehension or visualization: **we need tools to summarize the graphs**.
- Existing tools: UCINet, Pajek, SocNetV, tnet
- Limitations:
  - Target workstations, **limited in memory**
  - No parallelism, **limited in performance**.
  - Scale only to low density graphs with a **few million vertices**
- We need a package that will easily accommodate graphs with several **billion** vertices and deliver results in a timely manner.
  - Need parallelism both for computational speed and memory!
  - The Cray XMT is a natural fit...

What is GraphCT?

- **Graph C**haracterization **T**oolkit
- Efficiently summarizes and analyzes static graph data
- Built for large multithreaded, shared memory machines like the Cray XMT
- Increases productivity by decreasing programming complexity
- Classic metrics & state-of-the-art kernels
- Works on many types of graphs
  - directed or undirected
  - weighted or unweighted



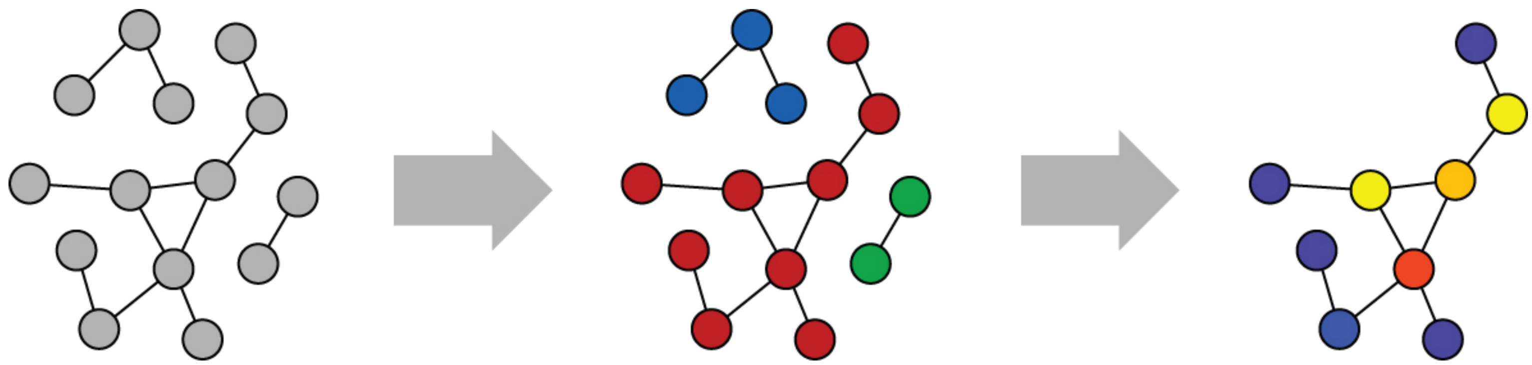
Dynamic spatio-temporal graph

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

Key Features of GraphCT

- Low-level primitives to high-level analytic kernels
- Common graph data structure
- Develop custom reports by mixing and matching functions
- Create subgraphs for more in-depth analysis
- Kernels are tuned to maximize scaling and performance (up to 64 processors) on the Cray XMT



Load the Graph Data — Find Connected Components — Run *k*-Betweenness Centrality on the largest component

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

GraphCT Functions

Name
RMAT graph generator
Degree distribution statistics
Graph diameter
Maximum weight edges
Connected components
Component distribution statistics
Vertex Betweenness Centrality
Vertex k-Betweenness Centrality
Multithreaded BFS
Edge-divisive Betweenness-based Comm Detection (pBD)
Lightweight Binary Graph I/O

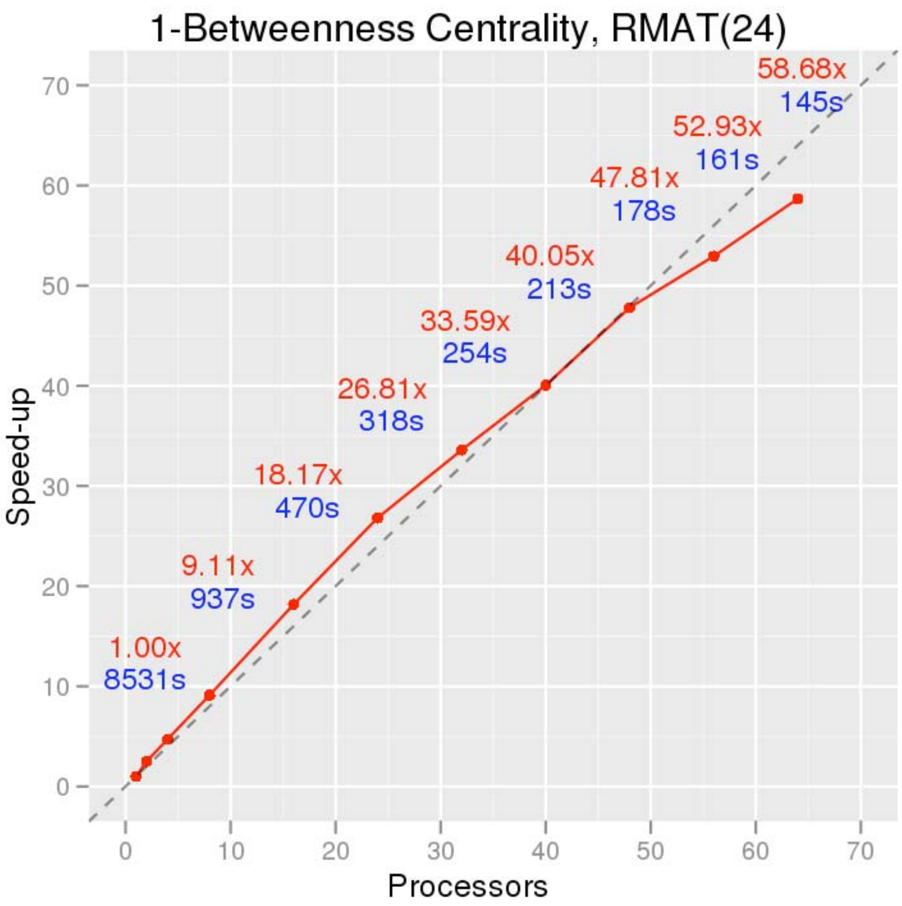
Name
Modularity Score
Conductance Score
st-Connectivity
Delta-stepping SSSP
Bellman-Ford
GTriad Census
SSCA2 Kernel 3 Subgraphs
Greedy Agglomerative Clustering
Minimum spanning forest
Clustering coefficients
DIMACS Text Input

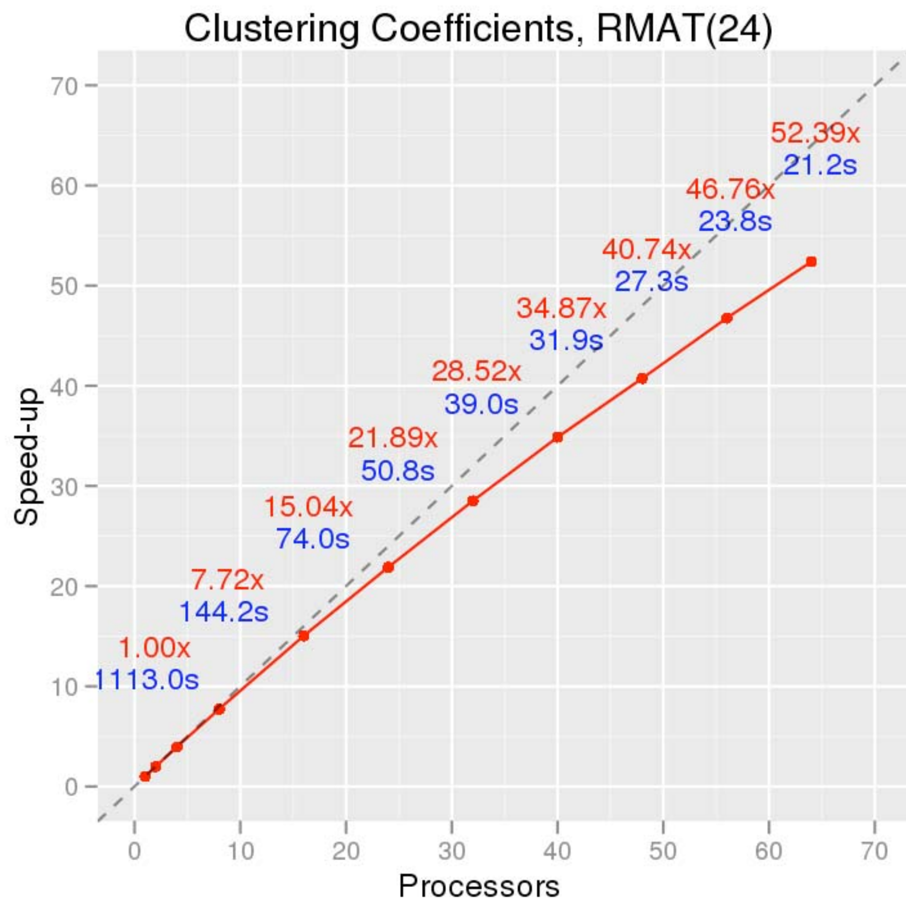
Key
Included
In Progress
Proposed/Available

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

GraphCT Performance





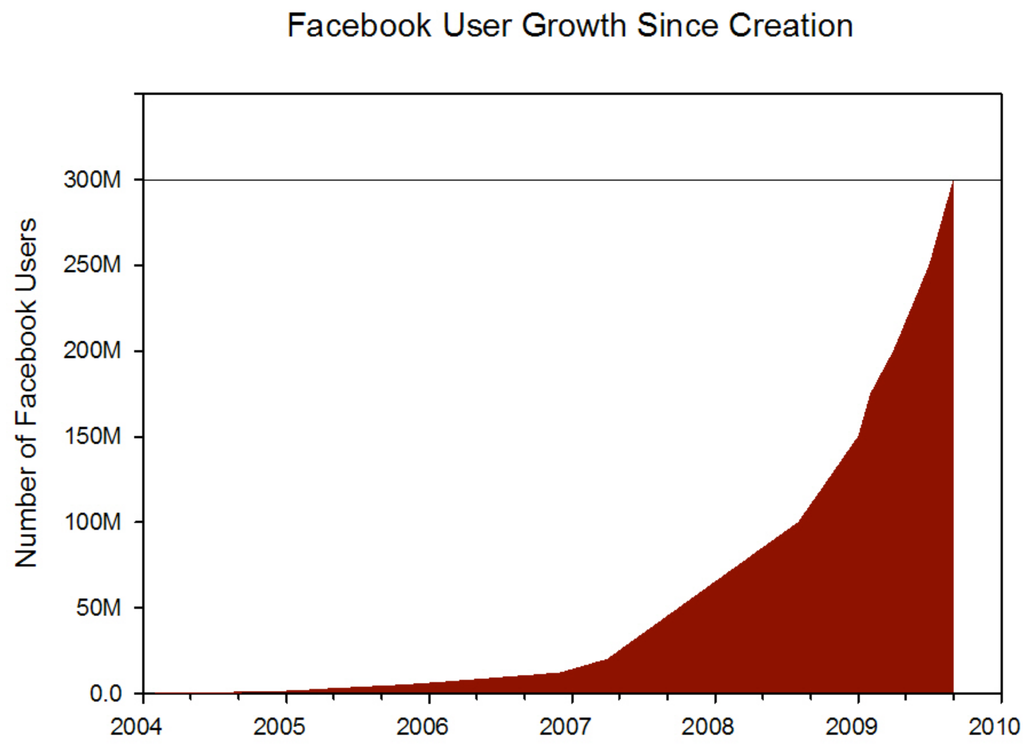
- RMAT(24) : 16.7M vertices, 134M edges
- RMAT(28) : 268M vertices, 2.1B edges
  - BC_1 : 2800s on 64P
  - CC : 1200s on 64P

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

Driving Forces in Social Network Analysis

- Note the graph is **changing** as well as growing.



300 million active Facebook users worldwide in September 2009

- What are this graph's properties? ***How do do they change?***

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Analysis of Graphs with Streaming Updates

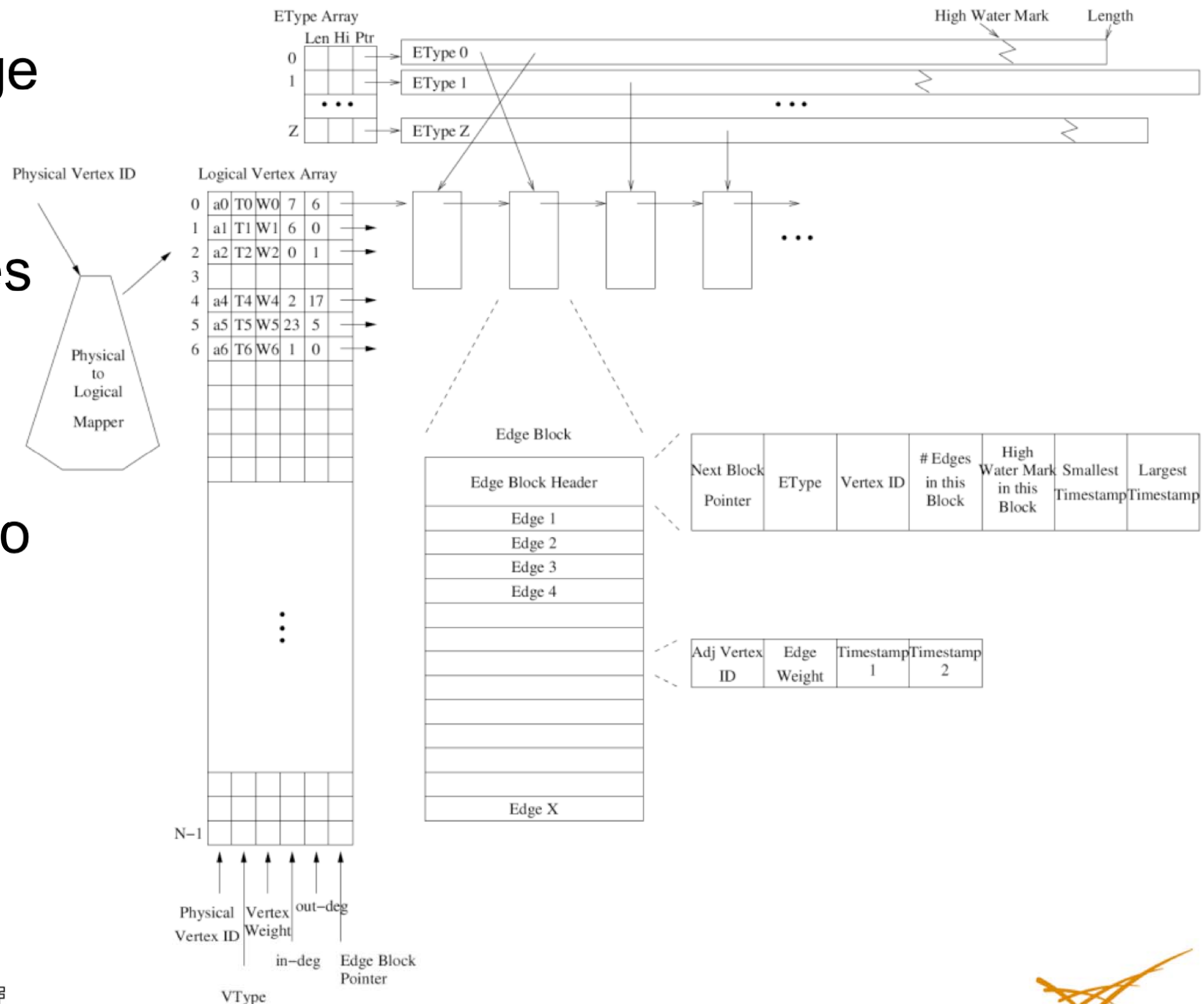
- STINGER: A Data Structure for Changing Graphs
  - Light-weight data structure that supports efficient iteration *and* efficient updates.
- Experiments with Streaming Updates to Clustering Coefficients
  - Working with bulk updates, can handle almost 200k per second

STING Extensible Representation

- Enhanced representation developed for dynamic graphs developed in consultation with David A. Bader, Johnathan Berry, Adam Amos-Binks, Daniel Chavarría-Miranda, Charles Hastings, Kamesh Madduri, and Steven C. Poulos.
- Design goals:
  - Be useful for the entire “large graph” community
  - Portable semantics and high-level optimizations across multiple platforms & frameworks (XMT C, MTGL, etc.)
  - Permit good performance: No single structure is optimal for all.
  - Assume globally addressable memory access
  - Support multiple, parallel readers and a single writer
- Operations:
  - Insert/update & delete both vertices & edges
  - Aging-off: Remove old edges (by timestamp)
  - Serialization to support checkpointing, etc.

STING Extensible Representation

- Semi-dense edge list blocks with free space
- Compactly stores timestamps, types, weights
- Maps from application IDs to storage IDs
- Deletion by negating IDs, separate compaction

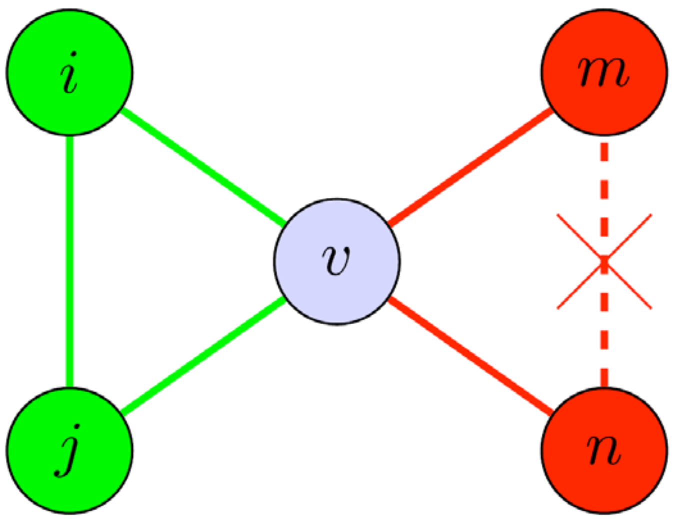


Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Testbed: Clustering Coefficients

- Roughly, the ratio of actual triangles to possible triangles around a vertex.



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

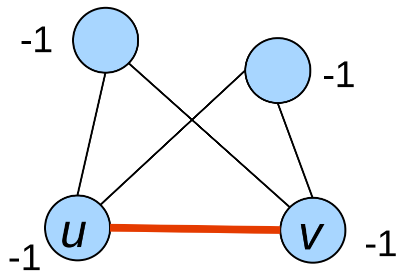
closed triplets / # all triplets

- Locally, count those around *v*.
- Globally, count across entire graph.
  - Multiple counting cancels (3/3=1)

Georgia Tech College of Computing
Computational Science and Engineering
Pacific Northwest NATIONAL LABORATORY

Streaming updates to clustering coefficients

- Monitoring clustering coefficients could identify anomalies, find forming communities, *etc*.
- Computations stay **local**. A change to edge <*u, v*> affects only vertices *u*, *v*, and their neighbors.



- Need a fast method for updating the triangle counts, degrees when an edge is inserted or deleted.
  - Dynamic data structure for edges & degrees: STINGER
  - Rapid triangle count update algorithms: exact and **approximate**
- "Massive Streaming Data Analytics: A Case Study with Clustering Coefficients." Ediger, David, Karl Jiang, E. Jason Riedy, and David A. Bader. Technical Report, Georgia Tech, Fall 2009.

Updating clustering coefficients

- Using RMAT as a graph and edge stream generator.
  - Mix of insertions and deletions
- Result summary for single actions
  - Exact: from 8 to 618 actions/second
  - Approx: from 11 to 640 actions/second
- Alternative: Batch changes
  - Lose some temporal resolution within the batch
  - Median rates for batches of size B:

Algorithm	B = 1	B = 1000	B = 4000
Exact	90	25 100	50 100
Approx.	60	83 700	193 300

- STINGER overhead is minimal; most time in spent metric.

Future Directions

- User interaction with GraphCT
  - What characteristics are of interest?
  - What output reports?
- **STING**, a framework for analyzing **S**patio-**T**emporal **I**nteraction **N**etworks and **G**raphs
  - Take current experimental infrastructure and generalize it.
  - Accept streaming data from outside the XMT.
    - (Frees up more memory for analyzing the data.)
  - Incorporate new, novel analysis techniques.
    - Update metrics, track statistically significant subgraphs (with Dr. Kamesh Madduri, LBNL), ...
  - And eventually, more complicated user queries.
  - (Transferring the analysis results back out is an open issue.)

Hierarchy of Interesting Analytics

- **Extend single-shot graph queries to include time.**
  - Are there *s*-*t* paths between time T_1 and T_2?
  - What are the important vertices at time *T*?
- **Use persistent queries to monitor properties.**
  - Does the path between *s* and *t* shorten drastically?
  - Is some vertex suddenly very central?
- **Extend persistent queries to fully dynamic properties.**
  - Does a small community stay independent rather than merge with larger groups?
  - When does a vertex jump between communities?
- **New types of queries, new challenges...**

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Recent Publications

- Jiang, Karl, David Ediger, and David A. Bader. "**Generalizing k-Betweenness Centrality Using Short Paths and a Parallel Multithreaded Implementation**." The 38th International Conference on Parallel Processing (ICPP 2009), Vienna, Austria, September 2009.
- Madduri, Kamesh, David Ediger, et al. "**A Faster Parallel Algorithm and Efficient Multithreaded Implementations for Evaluating Betweenness Centrality on Massive Datasets**." Third Workshop on Multithreaded Architectures and Applications (MTAAP), Rome, Italy, May 2009.
- Bader, David A., et al. "**STINGER: Spatio-Temporal Interaction Networks and Graphs (STING) Extensible Representation**." 2009.
- Ediger, David, Karl Jiang, E. Jason Riedy, and David A. Bader. "**Massive Streaming Data Analytics: A Case Study with Clustering Coefficients**," Technical Report, Georgia Tech, Fall 2009.

Backup

- *k*-Betweenness centrality details
- Clustering coefficients details
- GraphCT User's & Developer's Guide

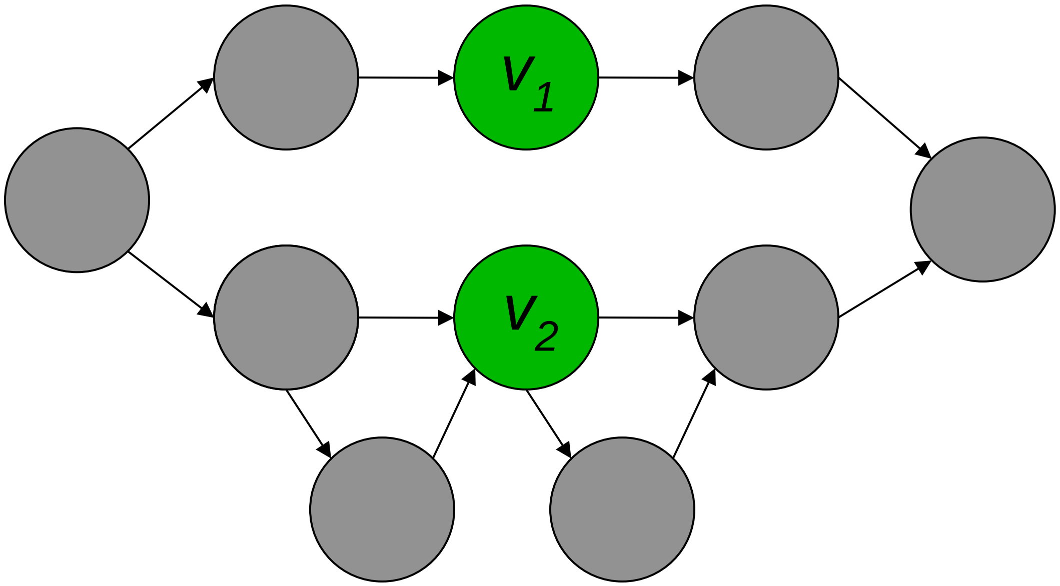
Outline: *k*-Betweenness Centrality, BC_k

- A new twist on betweenness centrality:
  - Count **short** paths in addition to **shortest** paths
  - Captures wider connectivity information
- Quick introduction and illustration
- Applying BC_k to the Notre-Dame WWW data set:
  - How do the scores behave with increasing *k?*
  - Which vertices have zero scores?
    - (Directed and undirected graphs are different.)
  - Can we approximating by BC_k random sampling?
- Scalability on the Cray XMT with RMAT graphs.

k-Betweenness Centrality

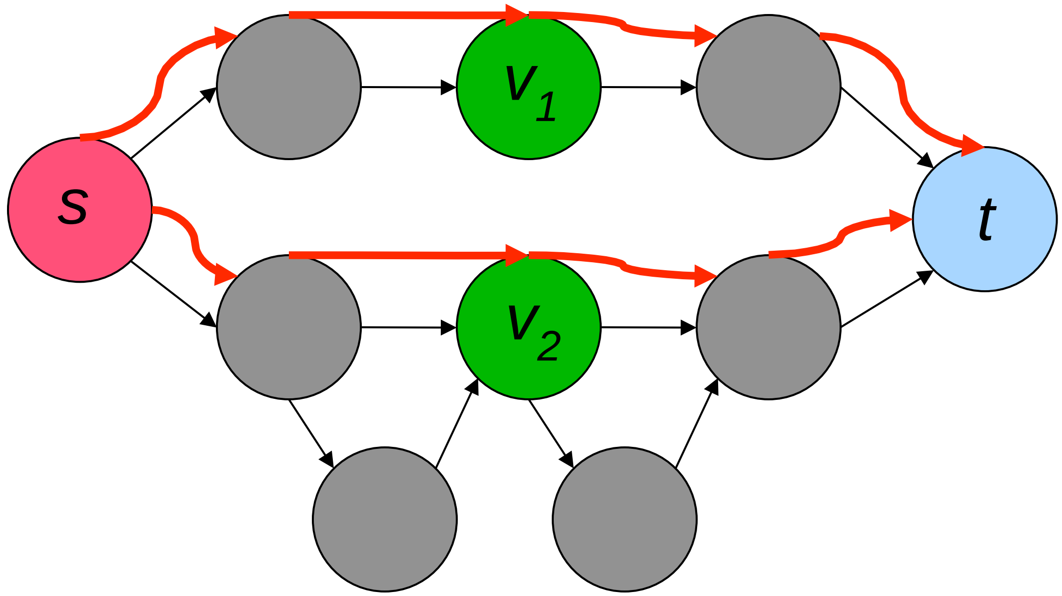
- Measure *centrality* of a vertex ***v*** by the number of paths passing through ***v*** between ***s*** and ***t*** relative to the number of paths connecting ***s*** and ***t***.
- High *betweenness centrality (BC):* many **shortest** paths
- High *k-betweenness centrality* (BC_k)*:* many **short** paths
  - All paths no longer than the shortest + parameter *k* counted.
  - 0-Betweenness centrality is simply betweenness centrality.
  - 1-BC also counts paths one step longer than the shortest.
- BC_k captures more connectivity information with *k*.
- Expensive to compute as *k* grows, but approximated...

Betweenness Centrality



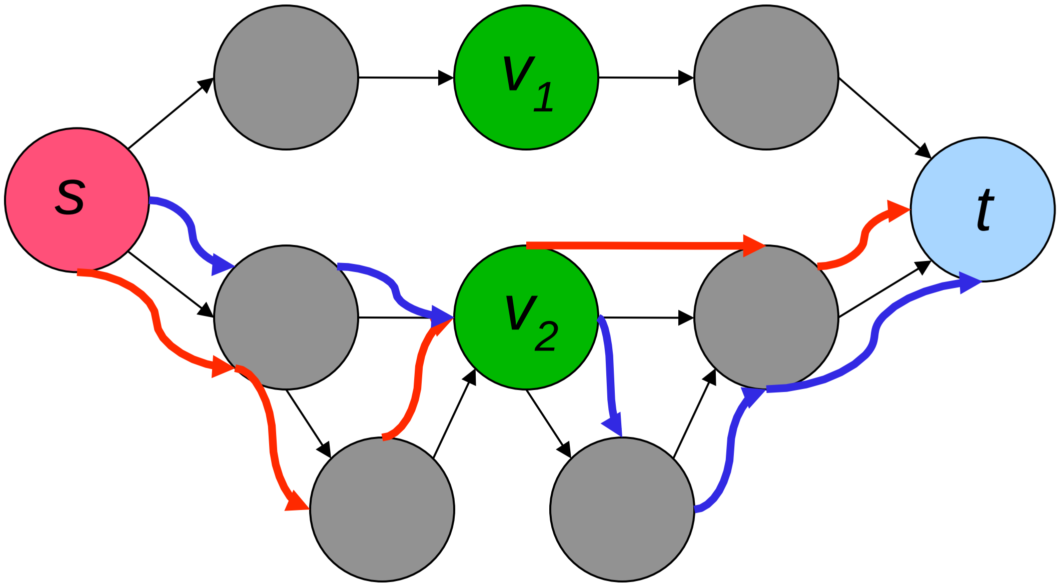
- How important are v_1 and v_2? Use betweenness centrality.
- The betweenness centrality of v_1, $BC(v_1)$:
  - Consider **shortest** paths between any two vertices *s*, *t* ≠ *v1.*
  - Sum over all such *s*, *t*: fraction of paths passing through v_1

BC: Need More Than the Shortest Path?



- Consider the view from a particular vertex pair *s*, *t*.
- Total of five paths, so the *st* contributions to v_1, v_2 = 1/5.
- But there is more redundancy through v_2, more nodes influence / are influenced by v_2...

k-Betweenness Centrality: Shortest + *k*



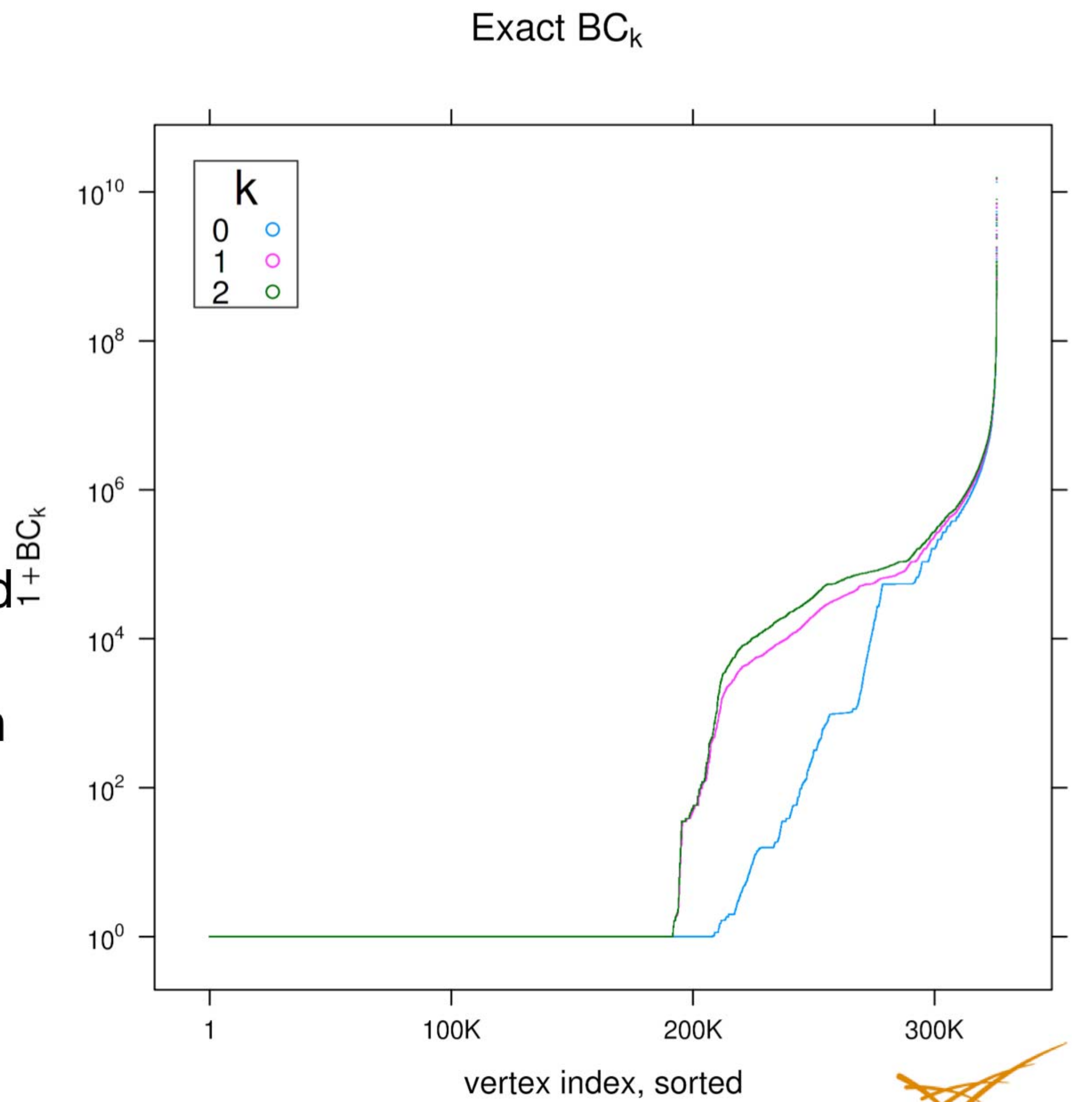
- Consider counting paths **one longer** than the shortest.
- Nothing new through v_1. Two new paths cross through v_2!
- *k*-Betweenness Centrality (BC_k):
  - Consider paths within *k* of the shortest path. Above is BC_1.
  - 0-Betweenneess centrality is regular BC, $BC_0(v) = BC(v)$.

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

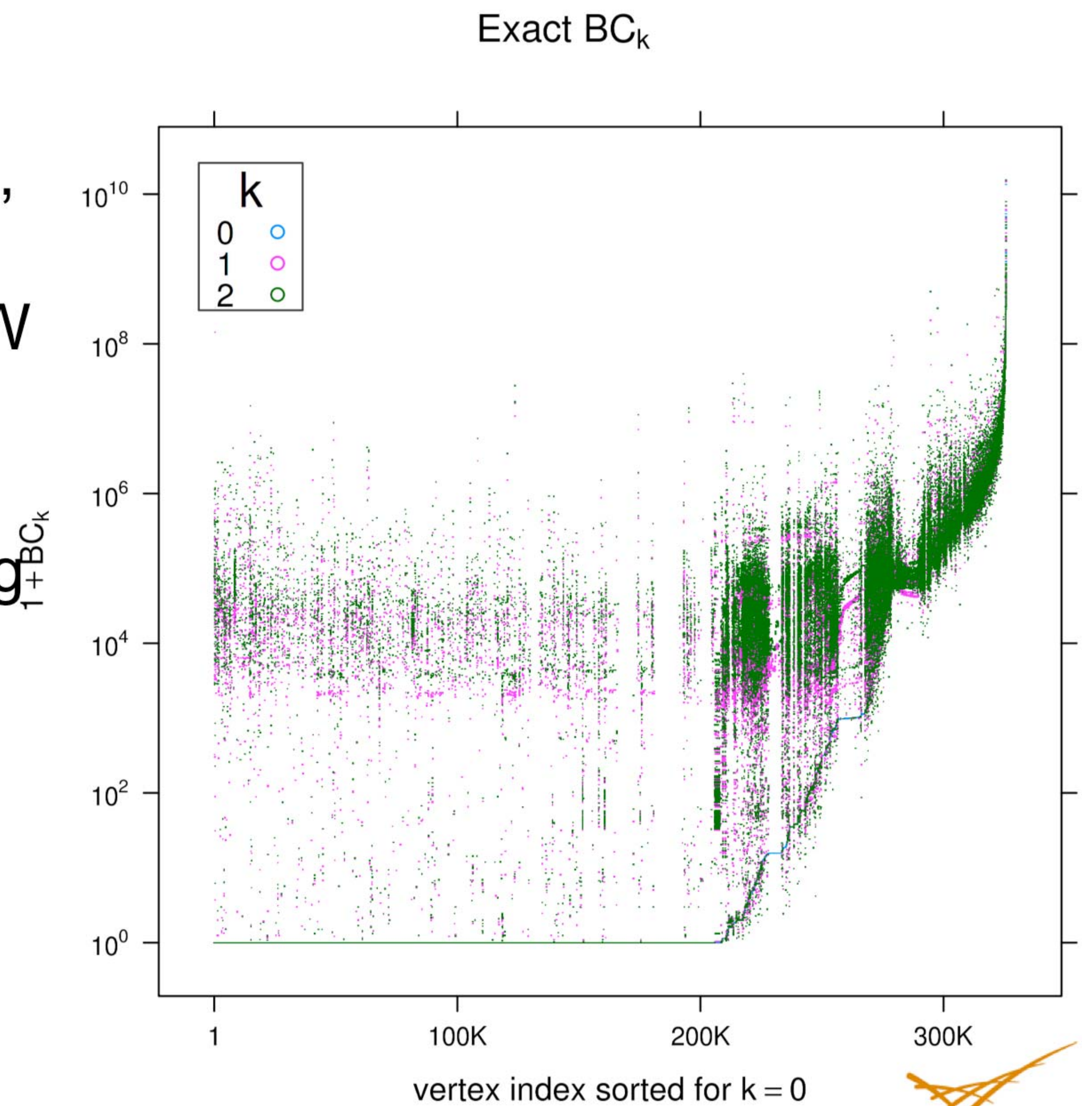
BC_k for k > 0: More Path Information

- Exact BC_k for k = 0, 1, 2
- On directed ND-WWW
- Vertices in increasing BC_k order (**independently**)
- Large difference going from k = 0 to k > 0
- Few additional paths found in k = 2
- ***k* > 0 captures more path information, somewhat converges**



BC_k for k > 0: More Path Information

- Exact BC_k for k = 0, 1, 2
- On directed ND-WWW
- Vertices in increasing BC_k order **(by k = 0)**
- Large difference going from k = 0 to k > 0
- Few additional paths found in k = 2
- **Note how many vertices jump from BC_0 = 0 to BC_k > 0!**

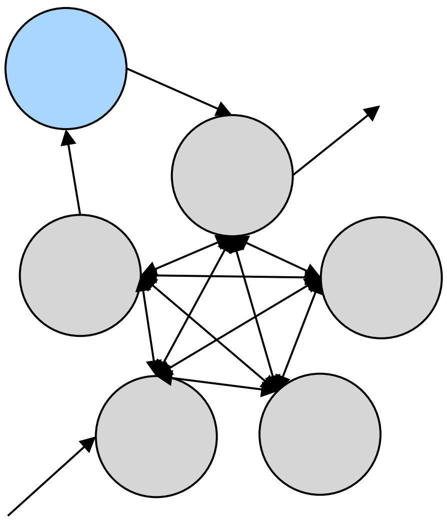


Georgia Tech College of Computing
Computational Science and Engineering

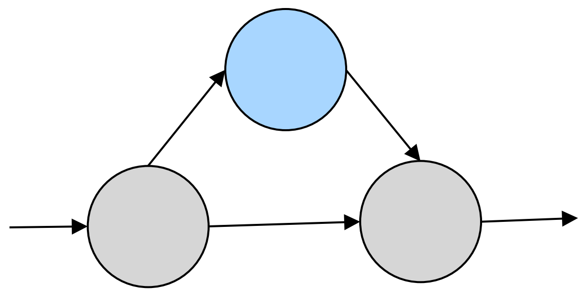
Pacific Northwest NATIONAL LABORATORY

Which Vertices Become Central with *k* > 0?

For *k* = 0 only



Neighbors form a clique

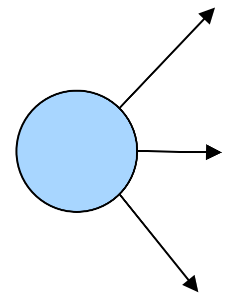


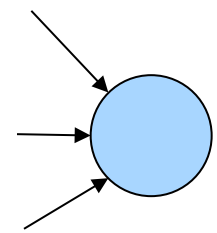
One step out of a path

More?

(Different than undirected.)

For all *k*





0 in- or out-degree

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Exact BC_k: Too Expensive, So Approximate...

- ND-WWW graph: 325K vertices, 1.4M edges (smallish)
- 64 processor XMT @ PNNL, 16 proc. runs
- Timings (more caveats mentioned later):
  - Approximate BC_k with 256 source vertices v. exact BC_k
  - Not parallel between samples. Limits scalability, but wasn't obvious until the code was optimized (by a factor of 11x).
  - Exact timings are older code on the 16-proc. XMT. Too slow to run often.

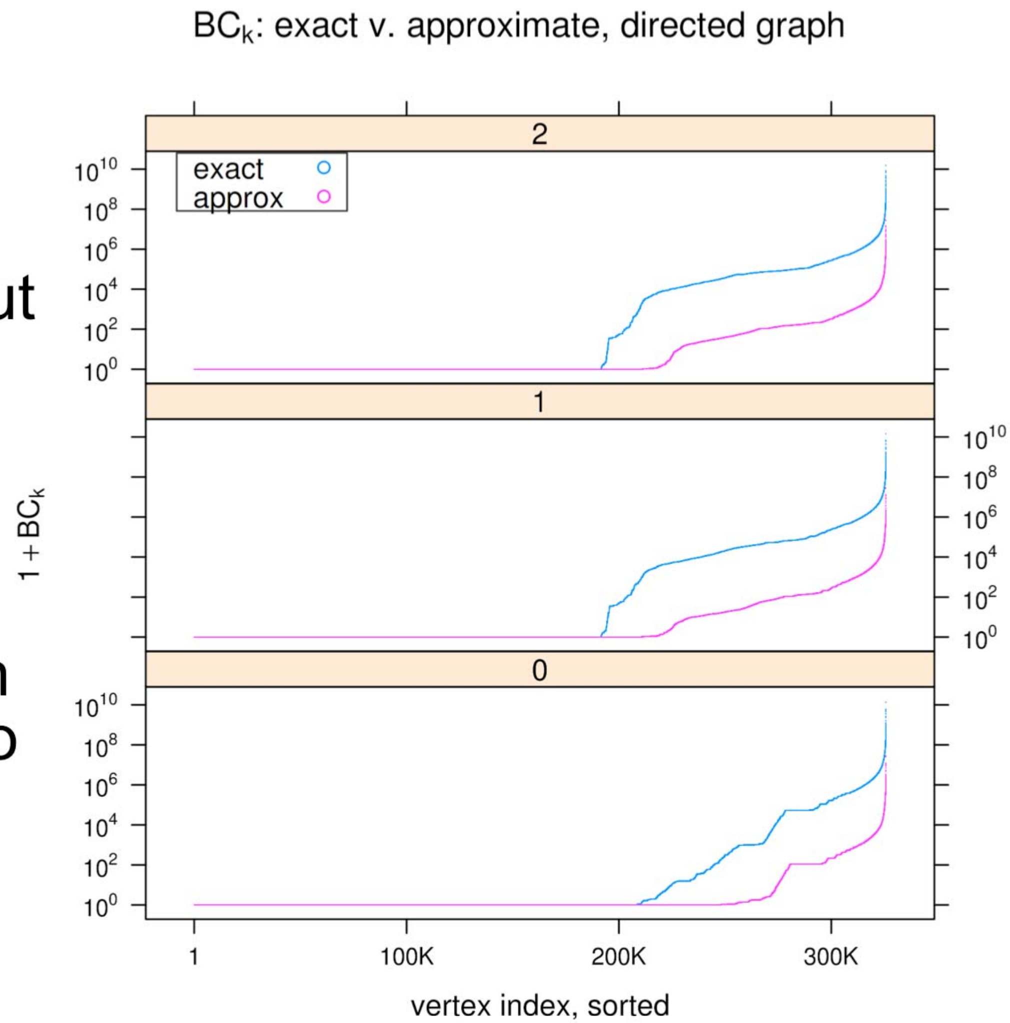
k	Approx.	Exact (old)
0	34s	43m
1	73s	13h
2	123s	43h

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

Approximating BC$_k$ by Sampling

- No approximation theory yet for directed graphs...
- Poor normalization, but captures much of the shape.
- Percentiles are better quality judge.
- Current approximation renders too many zero scores, **undersampling**.
- Missing a handful of vertices in top 5%.

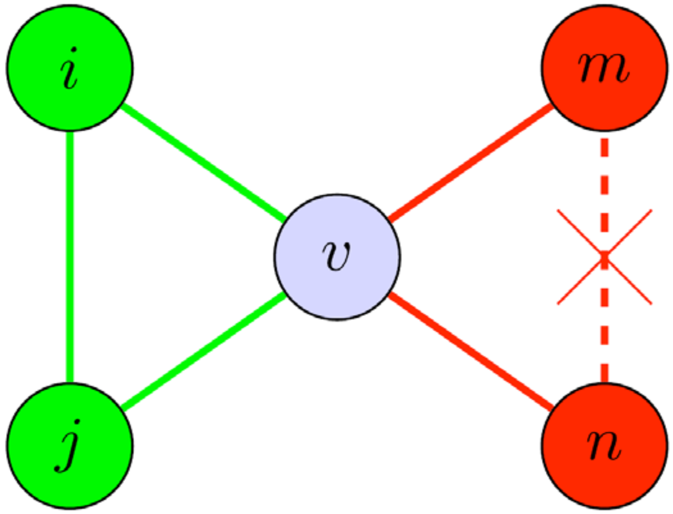


Outline: Clustering coefficients

- Quickly define clustering coefficients.
  - We're not going into interpretation, just computation.
- Performance within GraphCT
  - Static graph, scalable performance.
- Performance in a streaming framework
  - Update clustering coefficients as new data arrives.
  - Performance for adding edges 1-by-1 and in batches.

Clustering coefficients, undirected graphs

- Roughly, the ratio of actual triangles to possible triangles around a vertex.



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

closed triplets / # all triplets

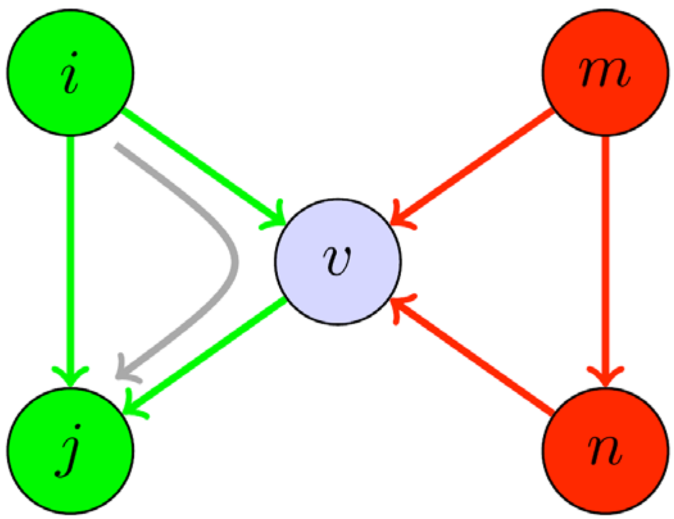
- Locally, count those around *v*.
- Globally, count across entire graph.
  - Multiple counting cancels (3/3=1)

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

Transitive coefficients, *directed* graphs

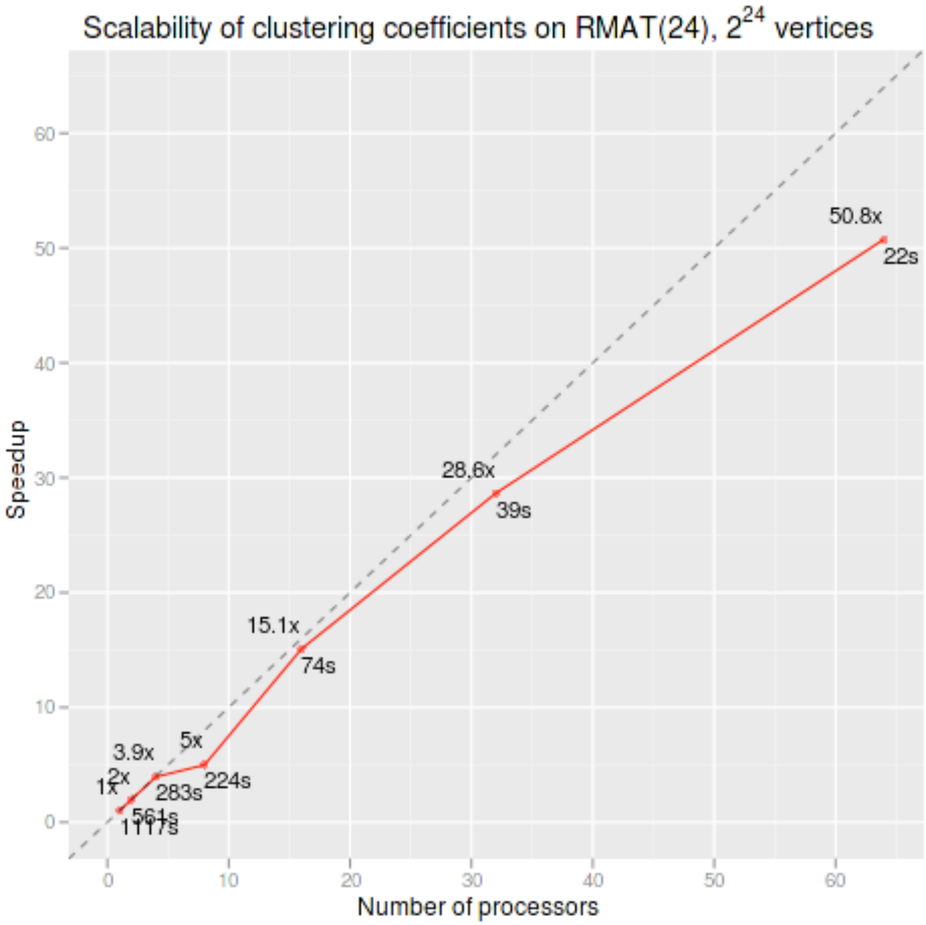
- Roughly, the ratio of actual triangles to possible triangles around a vertex. **But what counts as a triangle?**



- Possibility: ***transitive coefficients***
  - *i-v-j* is a ***closed triplet***, i-v-j has a transitive shortcut, i-j.
  - *m-v-n* is an ***open triplet***.
- Very sensitive to the direction of edges.
- Temporal heuristic: the reverse edges often appear, delayed.
- Many variations exist in the literature. Computing each is similar; need application requests...

Performance of static clustering coefficients

- GraphCT supports basic clustering coefficients and transitivity coefficients
- Performance roughly the same for all versions
- Nice, inexpensive characterization kernel
- Being extended to handle streaming data
  - Multiple approaches:
    - Exact: Count locally
    - Approx: Bloom filters



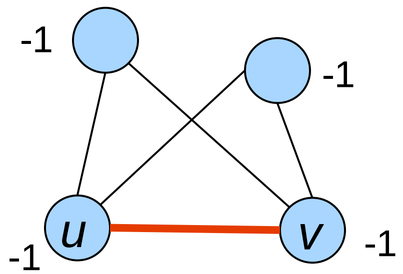
Global clustering coeff: Speed-up of 51x on 64p and RMAT(24)

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Streaming updates to clustering coefficients

- Monitoring clustering coefficients could identify anomalies, find forming communities, *etc*.
- Luckily, computations stay local. A change to edge <*u, v*> affects only vertices *u*, *v*, and their neighbors.



- Need a fast method for updating the triangle counts, degrees when an edge is inserted or deleted.
  - Dynamic data structure for edges & degrees: STINGER
  - Rapid triangle count update algorithms: exact and approximate
- Technical Report: Ediger, David, Karl Jiang, E. Jason Riedy, and David A. Bader. “Massive Streaming Data Analytics: A Case Study with Clustering Coefficients.”

Updating clustering coefficients

- Update local & global clustering coefficients while edges <*u, v*> are inserted and deleted.
- Exact and approximate approaches:
  - Exact: Explicitly count triangle changes by doubly-nested loop
    - *O(du * dv)*, where *dx* is the degree of *x* after insertion/deletion
  - Exact: Sort one edge list, loop over other and search with bisection.
    - *O((du + dv) log (du))*
  - Approx: Summarize one edge list with a Bloom filter. Loop over other, check using *O(1)* **approximate** lookup. May count too many, never too few.
    - *O(du + dv)*
- Expect issues near high degree vertices (hubs).

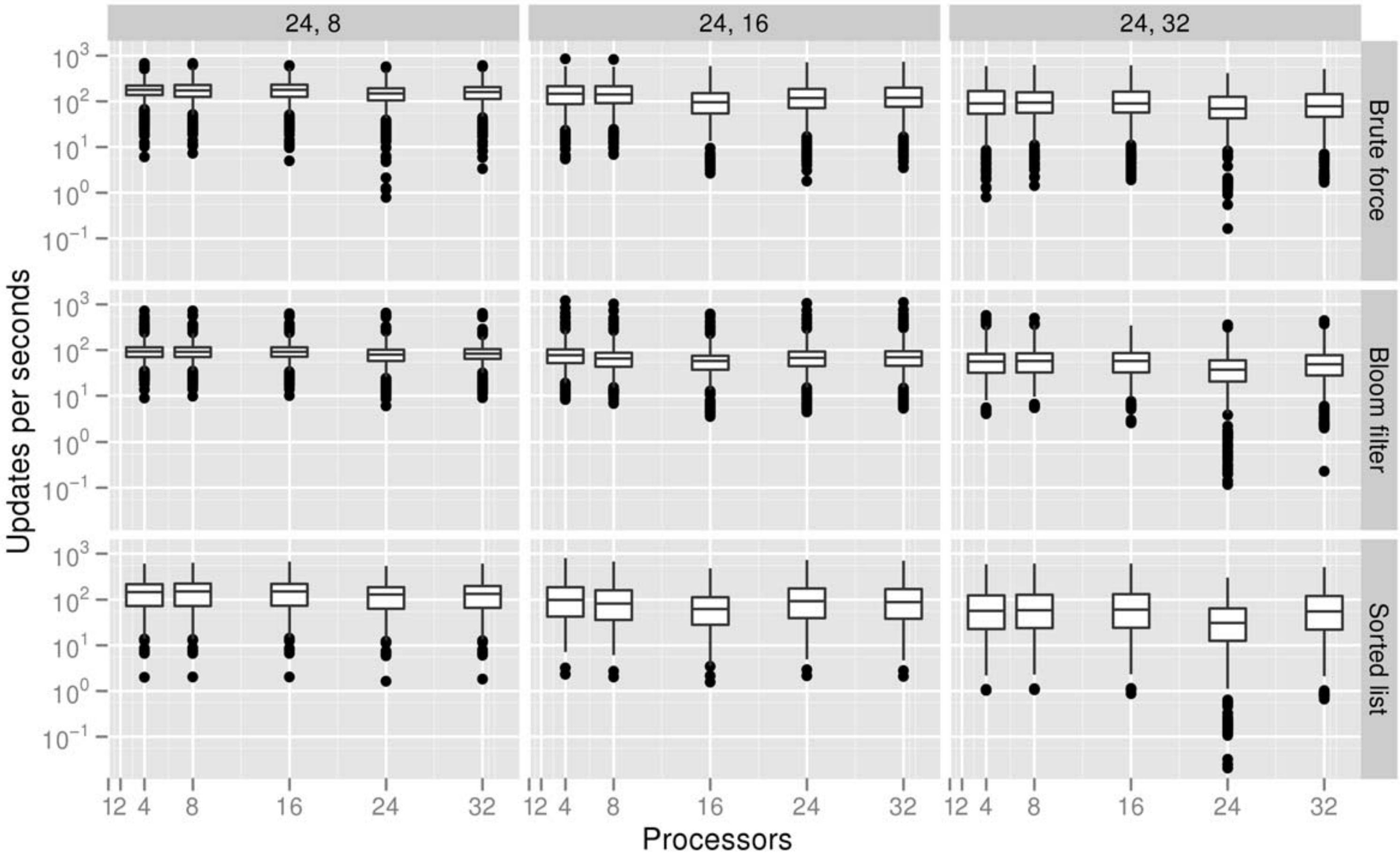
Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Updating clustering coefficients

- Using RMAT as a graph and edge generator.
- Generate graph with scale S and edge factor F, $2^S F$ edges.
  - Scale 24: 17 million vertices
  - Edge factors 8 to 32: 134 to 537 million edges
- Generate 1024 actions.
  - Deletion chance 6.25% = 1/16
  - Same RMAT process, will prefer same vertices.
- Start with an exact triangle count, run individual updates.
- **Result summary**
  - Exact: from 8 to 618 actions/second
  - Approx: from 11 to 640 actions/second

Updating clustering coefficients one-by-one



Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Updating clustering coefficients in a batch

- Start with an exact triangle count, run individual ***batched*** updates:
  - Consider *B* updates at once.
  - Currently loses some temporal resolution within a batch. Changes to the same edge are collapsed.
- **Result summary**

Algorithm	B = 1	B = 1000	B = 4000
Exact	90	25 100	50 100
Approx.	60	83 700	193 300

- More analysis in progress...

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

CASS-MT Task #7 - Georgia Tech

GraphCT: A Graph Characterization Toolkit

David A. Bader, David Ediger, Karl Jiang & Jason Riedy

October 26, 2009

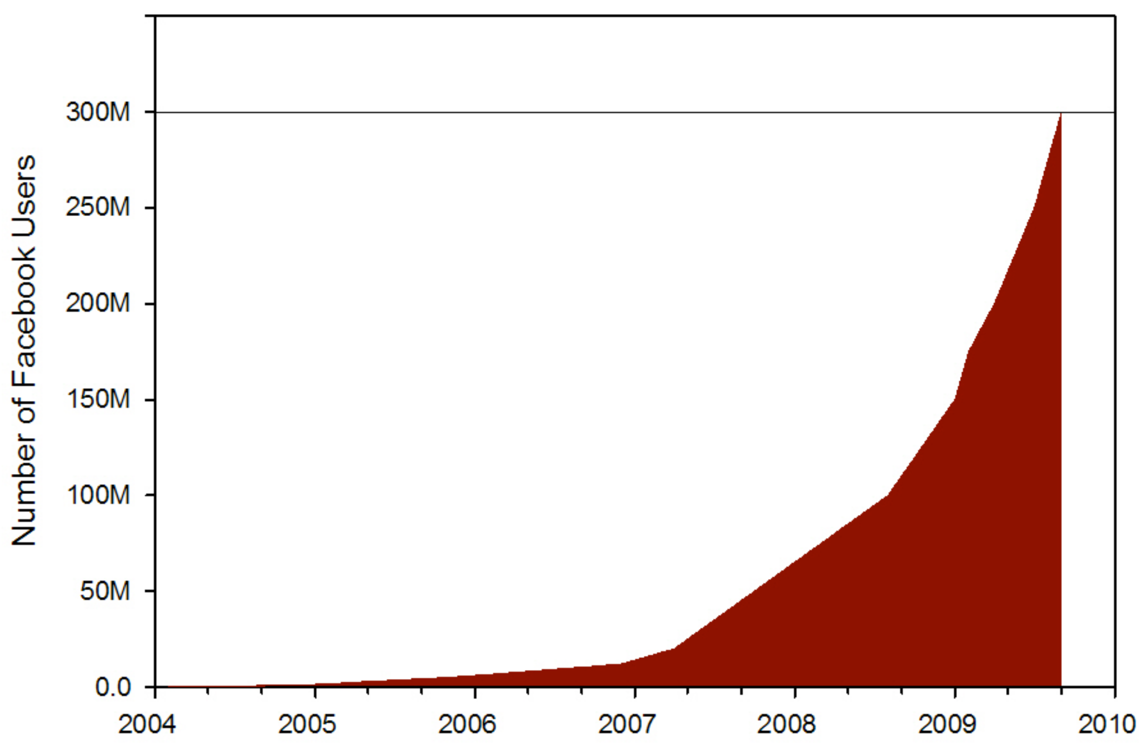
Outline

- Motivation
- What is GraphCT?
  - Package for Massive Social Network Analysis
  - Can handle graphs with **billions** of vertices & edges
- Key Features
  - Common data structure
  - A “buffet” of functions that can be combined
- Using GraphCT
- Future of GraphCT
- Function Reference

Driving Forces in Social Network Analysis

- An explosion of data!

Facebook User Growth Since Creation



300 million active Facebook users worldwide in September 2009

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Current Social Network Packages

- UCINet, Pajek, SocNetV, tnet
- Written in C, Java, Python, Ruby, R
- Limitations
  - Runs on workstation
  - Single-threaded
  - Several thousand to several million vertices
  - Low density graphs

- We need a package that will easily accommodate graphs with several **billion** vertices on large, parallel machines

The Cray XMT

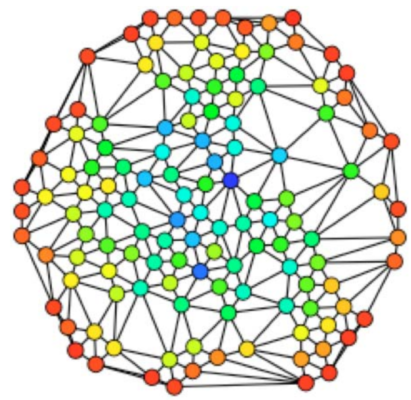
- Tolerates latency by massive multithreading
  - Hardware support for 128 threads on each processor
  - Globally hashed address space
  - No data cache
  - Single cycle context switch
  - Multiple outstanding memory requests
- Support for fine-grained, word-level synchronization
  - Full/empty bit associated with every memory word
- Flexibly supports dynamic load balancing

Image Source: cray.com

- GraphCT currently tested on a 64 processor XMT: **8192 threads**
  - **512 GB** of globally shared memory

What is GraphCT?

- **Graph C**haracterization **T**oolkit
- Efficiently summarizes and analyzes static graph data
- Built for large multithreaded, shared memory machines like the Cray XMT
- Increases productivity by decreasing programming complexity
- Classic metrics & state-of-the-art kernels
- Works on all types of graphs
  - directed or undirected
  - weighted or unweighted



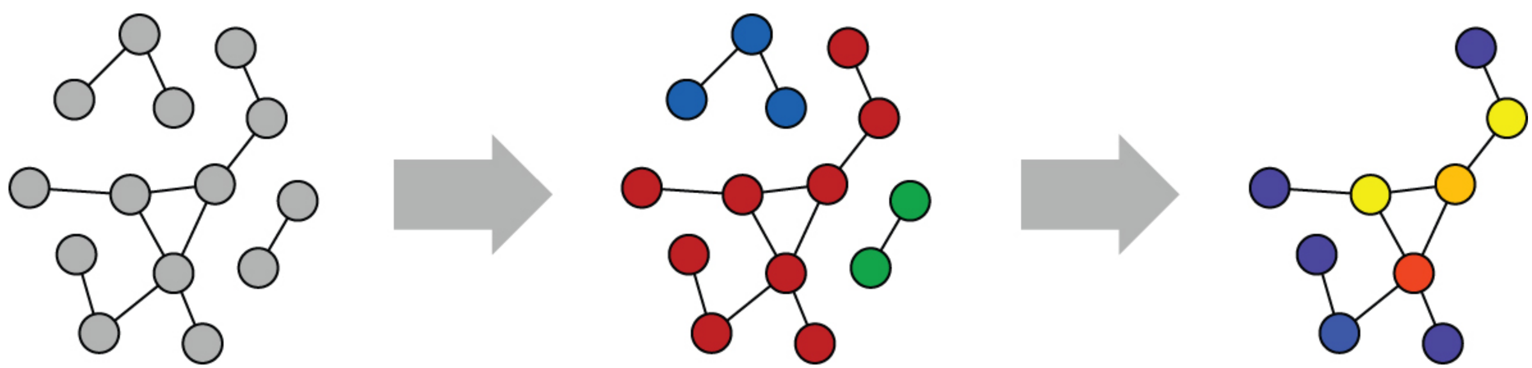
Dynamic spatio-temporal graph

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Key Features of GraphCT

- Low-level primitives to high-level analytic kernels
- Common graph data structure
- Develop custom reports by mixing and matching functions
- Create subgraphs for more in-depth analysis
- Kernels are tuned to maximize scaling and performance (up to 64 processors) on the Cray XMT



Load the Graph Data

Find Connected Components

Run *k*-Betweenness Centrality on the largest component

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

Static graph data structure

```
typedef struct {
   int numEdges;
   int numVertices;
   int startVertex[NE]; /* start vertex of edge,
                           sorted, primary key */
   int endVertex[NE];   /* end vertex of edge,
                           sorted, secondary key */
   int intWeight[NE];   /* integer edge weight */

   int edgeStart[NV];   /* per-vertex index into
                           endVertex array */
   int marks[NV];      /* common array for marking
                           or coloring of vertices */
} graph;
```

Using GraphCT

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

Usage options

- Operations on input graphs can be specified in 3 ways:
  - Via the command line
    - Perform a single graph operation
    - Read in graph, execute kernel, write back result
  - Via a script **[in progress]**
    - Batch multiple operations
    - Intermediate results need not be written to file (though they can be)
  - Via a developer's API
    - Perform complex series of operations
    - Manipulate data structures
    - Implement custom functions

The command line interface

1. Command line parameters

Example: `./GraphCT-CLI –i patents.txt –t dimacs –o result.txt –z kcentrality 1`

`-i`: Input file

`-t`: Graph type, can currently be either ‘dimacs’ or ‘binary’. ‘binary’ type is binary compressed row format generated by GraphCT

`-o`: Output file

`-z`: Kernel type (see following sections):

Georgia Tech College of Computing Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

2. Kernel types (index)

- Specified after `–z` flag
  - `kcentrality k Vs`
  - `degree`
  - `conductance`
  - `modularity`
  - `components`
  - `clustering`
  - `transitivity`
  - `diameter n`

3. Degree distribution & graph diameter

- Diameter can only be ascertained by repeatedly performing breadth first searches different vertices.
  - The more breadth first searches, the better approximation to the true diameter
  - `-z diameter <P>`
    - Does breadth first searches from P percent of the vertices, where P is an integer

- Degree distribution:
  - `-z degree:` gives
    - Maximum out-degree
    - Average out-degree
    - Variance
    - Standard deviation

4. Conductance and modularity

```
-z conductance, -z modularity
```

- Defined over colorings of input graph
  - Describe how tightly knit communities divided by a cut are
  - Not very meaningful in command line mode
  - In batch mode a coloring can be followed by conductance/ modularity calculation

- In batch mode:
  - Finds connected components
  - Modularity uses component coloring as a partition
  - Conductance uses the largest component as the cut

5.Vertex k-Betweenness Centrality

```
-z kcentrality k Vs
```

- Vs: number of source vertices (of breadth first search)
  - Set equal to NV (number of vertices) for exact computation
- k: count shortest path length + k
- Outputs file with k-BC scores ordered by vertex number

- *Note: Set k equal to 0 for betweenness centrality*

K. Jiang, D. Ediger, and D.A. Bader, "Generalizing k-Betweenness Centrality Using Short Paths and a Parallel Multithreaded Implementation," *The 38th International Conference on Parallel Processing* (ICPP 2009), Vienna, Austria, September 22-25, 2009.

6. Transitivity/clustering coefficient

`-z transitivity`

- Writes output file with local transitivity coefficient of each vertex
  - Measures number of transitive triads over total number of transitive triples

`-z clustering`

- Writes output file with local clustering coefficient of each vertex
  - Number of triangles formed by neighbors over number of potential triangles
  - Gives sense of how close vertex is to belonging to a clique

Tore Opsahl and Pietro Panzarasa. "Clustering in weighted networks," *Social Networks*, 31(2):155-163, May 2009.

7. Component statistics

`-z components`

- Statistics about connected components in graph
  - Number of components
  - Largest component size
  - Average component size
  - Variance
  - Standard deviation
- Writes output file with vertex to component mapping

Writing a script file [in progress]

1. Example script

```
read dimacs patents.txt => binary_pat.bin
print diameter 10
save graph
extract component 1 => component1.bin
print degrees
kcentrality 1 256 => k1scores.txt
kcentrality 2 256 => k2scores.txt
restore graph
extract component 2
print degrees
```

2. Script fundamentals

- Work on single ‘active graph’
- Can save and restore graphs at any point, like memory feature on pocket calculator
- Operations can:
  - Output data to the screen (e.g. degree information)
  - Output data to file (e.g. kcentrality data)
  - Modify the active graph (extract subgraph, component)

3. Example breakdown

`read dimacs patents.txt => binary_pat.bin`

- Two operations: reads in ‘patents.txt’ as a dimacs graph file, and writes the resulting graph back out as a binary file called ‘binary_pat.dat’
  - Binary graph is usually smaller and quicker to load
  - `=> filename` always takes the output of a particular command and writes it to the file ‘filename’
  - Current graph formats are ‘dimacs’ and ‘binary’

`print diameter 10`

- `print` command is used to print information to the screen
  - Shows the estimated diameter based on BFS runs from 10% of vertices

3. Example breakdown (cont.)

`save graph`

- Retain the current active graph for use later

`extract component 1 => component1.bin`

- `extract` command is used to use a coloring to extract a subgraph from the active graph
  - `component 1` colors the largest connected component
- Writes resulting graph to a binary file

`print degrees`

- Any kernel from the previous section may be used
- If output is a graph or per-vertex data, it cannot be printed

3. Example breakdown (cont.)

`kcentrality 1 256 => k1scores.txt`

- Calculates k=1 betweenness centrality based on breadth first searches from 256 source vertices
  - Result stored in ‘k1scores.txt’, one line per vertex
  - `kcentrality` result cannot be printed to screen since it is per-vertex data

`restore graph`

- Restore active graph saved earlier
- Can restore same graph multiple times

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

3. Example breakdown (cont.)

```
extract component 2
```

- Extract the second largest component of the graph

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Graph parsers

DIMACS graph parser

```
c comments
c here
p max n m
e v1 v2 w
```

- DIMACS file:
  - c = comment
  - p = problem line: n = number of vertices, m = number of edges
  - e = edge: indicates an edge from v1 to v2 of weight w
- Use standalone parser or read directly into GraphCT
  - Standalone parser outputs binary format graph file
    - Good if graph will be used multiple times to reduce I/O time

From data to analysis

- GraphCT produces a simple listing of the metrics most desired by the analyst
- At a glance, the size, structure, and features of the graph can be described
- Output can be custom tailored to show more or less data
- Full results are written to files on disk for per-vertex kernels
  - k-Betweenness Centrality
  - Local clustering coefficients
  - BFS distance
- Excellent for external plotting & visualization software

Terminal — ssh — 62x30

```
Maximum kBC Vertices
--------------------
# 1:  1043959 - 51185011.876094
# 2:     8514 - 23861002.706084
# 3: 10921177 - 21248010.604545
# 4: 12485450 - 21230627.310697
# 5:   639597 - 18581325.025557
# 6: 11455618 - 17163034.556793
# 7:   716348 - 17118163.282853
# 8:  5262636 - 17098394.527765
# 9:  9519290 - 16162145.184488
#10: 14898754 - 15169313.436984
Time taken for kernel X is 215.862593 sec.

Calculating out-degree distributions...
Maximum out-degree is:  3982
Average out-degree is:  7.918640
Expected value of X^2:  502.054685
Variance is:  439.349821
Standard deviation:  20.960673
Time taken is  0.040067 sec.

Finding connected components...
There are 1966802 components in the graph.
Maximum # of vertices:  14712027
Average # of vertices:  8.530201
Expected value of X^2:  110048566.522184
Variance:  110048493.757858
Standard deviation:  10490.400076
Time taken is 43.270031 sec.
```

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

The Future of GraphCT

- Additional high-level tools
  – Divisive betweenness-based community detection
  – Greedy agglomerative clustering (CNM)
  – Hybrid techniques
  – Additional subgraph generators
- Helper functions
  – Data pre-processing
  – Support for common graph formats
- Extension to support dynamic graph data
  – STINGER example

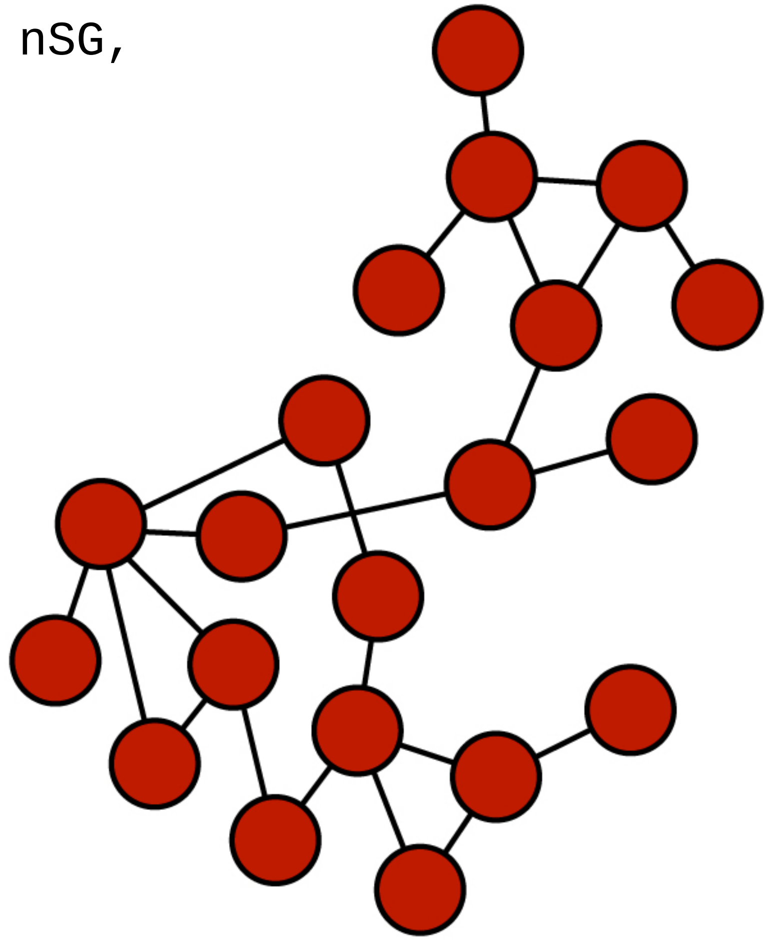
Experimental Kernels

Random walk subgraph extraction

```
void findSubGraphs(graph *G, int nSG,
int subGraphPathLength)
```

- Choose a number of random starting vertices nSG
- Perform a BFS of length `subGraphPathLength` from each source vertex
- Extract the subgraph:

```
subG = genSubGraph(G, NULL, 1);
```



Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Developer's Notes:

A Programming Example

1. Initialization & graph generation

```
// I want a graph with ~270 million vertices
getUserParameters(28);

// Generate the graph tuples using RMAT
SDGdata  = (graphSDG*) malloc(sizeof(graphSDG));
genScalData(SDGdata, 0.57, 0.19, 0.19, 0.05);

// Build the graph data structure
G = (graph *) malloc(sizeof(graph));
computeGraph(G, SDGdata);
```

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

2. Degree distribution & graph diameter

```
// Display statistics on the vertex out-degree
calculateDegreeDistributions(G);

// Find the graph diameter exactly
calculateGraphDiameter(G, NV);
// This will require 270M breadth first searches!

// Estimate the graph diameter
calculateGraphDiameter(G, 1024);
// This only does 1024 breadth first searches
```

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

3. Mark & summarize connected components

```
// run connected components & store the result in the graph
numComp = connectedComponents(G);

// display component size statistics based on colors
calculateComponentDistributions(G, numComp, &max, &maxV);
```

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

4. Find 10 highest 2-betweenness vertices

```
BC = (double *) malloc(NV * sizeof(double));

// k=2, 256 source vertices
kcentrality(G, BC, 256, 2);

printf("Maximum BC Vertices\n");
for (j = 0; j < 10; j++) {
      maxI  = 0; maxBC = BC[0];
      for (i = 1; i < NV; i++)
          if (BC[i] > maxBC) {maxBC = BC[i]; maxI = i;}
      printf("#%2d: %8d - %9.6lf\n", j+1, maxI, maxBC);
      BC[maxI] = 0.0;
}
```

Function Reference

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

Initialize default environment

```
void getUserParameters(int scale)
```

- Sets a number of application parameters
- `scale`: determines size of graph generation
  - $\log_2$ Number of Vertices

Load external graph data

```
int graphio_b(graph *G, char *filename)
```

- Load from a binary data file containing compressed data structure using 4-byte integers
- Format:
  - Number of Edges (4 bytes)
  - Number of Vertices (4 bytes)
  - Empty padding (4 bytes)
  - edgeStart array (NV * 4 bytes)
  - endVertex array (NE * 4 bytes)
  - intWeight array (NE * 4 bytes)

Scalable data generator

```
void genScalData(graphSDG*, double a, double b,
  double c, double d)
```

- Input:
  - RMAT parameters A, B, C, & D
  - Must call `getUserParameters( )` prior to calling this function
- Output:
  - graphSDG data structure (raw tuples)

- *Note: this function should precede a call to* `computeGraph()` *to transform tuples into a graph data structure*

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, April 2004. SIAM.

Graph construction

```
void computeGraph(graph *G, graphSDG *SDGdata)
```

- Input:
  - graphSDG data structure
- Output:
  - graph data structure

Directed graph -> undirected

```
graph * makeUndirected(graph *G)
```

- Input:
  - graph data structure
- Output:
  - Returns an undirected graph containing bidirectional edges for each edge in the original graph. Duplicate edges are removed automatically.

Generate a subgraph

```
graph * genSubGraph(graph *G, int NV, int color)
```

- Input:
  - graph data structure (marks[] must be set)
  - NV should always be set to NULL
  - color of vertices to extract
- Output:
  - Returns a graph containing only those vertices in the original graph marked with the specified color

Georgia Tech College of Computing Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

K-core graph reduction

```
graph * kcore(graph *G, int K)
```

- Input:
  - graph data structure
  - minimum out-degree K

- Output:
  - Returns a graph containing only those vertices in the original graph with an out-degree of at least K

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Vertex k-Betweenness Centrality

```
double kcentrality(graph *G, double BC[], int Vs,
   int K)
```

- Vs: number of source vertices
  - Set equal to G->NV for an exact computation
- K: count shortest path length + K
- BC[ ]: stores per-vertex result of computation

- *Note: Set K equal to 0 for betweenness centrality*

K. Jiang, D. Ediger, and D.A. Bader, "Generalizing k-Betweenness Centrality Using Short Paths and a Parallel Multithreaded Implementation," *The 38th International Conference on Parallel Processing* (ICPP 2009), Vienna, Austria, September 22-25, 2009.

Degree distribution statistics

```
void calculateDegreeDistributions(graph*)
```

- Input:
  - graph data structure
- Output:
  - Maximum out-degree
  - Average out-degree
  - Variance
  - Standard deviation

Component statistics

```
void calculateComponentDistributions (graph *G,
  int numColors, int *max, int *maxV)
```

- Input:
  - graph data structure
  - numColors: largest integer value of the coloring
- Output:
  - max: size of the largest component
  - maxV: an integer ID within the largest component

Modularity score

```
double computeModularityValue(graph *G,
  int membership[], int numColors)
```

- Input:
  - graph data structure
  - `membership[]`: the vertex coloring (partitioning)
  - `numColors`: the number of colors used above
- Output:
  - Modularity score is returned

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest
NATIONAL LABORATORY

Conductance score

```
double computeConductanceValue(graph *G,
    int membership[])
```

- Input:
  - graph data structure
  - `membership[]`: a binary partitioning
- Output:
  - Conductance score is returned

Connected components

```
int connectedComponents(graph *G)
```

- Input:
  - graph data structure
- Output:
  - `G->marks[]` : array containing each vertex's coloring where each component has a unique color
  - Returns the number of connected components

Breadth first search

```
int * calculateBFS(graph *G, int startV, int mode)
```

- Input:
  - graph data structure
  - startV: vertex ID to start the search from
  - mode:
    - mode = 0: return an array of the further vertices where the first element is the number of vertices
    - mode = 1: return an array of the distances from each vertex to the source vertex
- Output:
  - Returns an array according to the mode described above

D.A. Bader and K. Madduri, "Designing Multithreaded Algorithms for Breadth-First Search and st-connectivity on the Cray MTA-2," *The 35th International Conference on Parallel Processing* (ICPP 2006), Columbus, OH, August 14-18, 2006.

Graph diameter

```
int calculateGraphDiameter(graph *G, int Vs)
```

- Input:
  - graph data structure
  - Vs: number of breadth-first searches to run
- Output:
  - Returns the diameter (if Vs = NV) or the length of the longest path found

- *Note: this can be used to find the exact diameter or an approximation if only a subset of source vertices is used*

Global transitivity coefficient

```
double calculateTransitivityGlobal(graph *G)
```

- Input:
  - graph data structure
- Output:
  - Returns the global transitivity coefficient (for both directed and undirected graphs)

Tore Opsahl and Pietro Panzarasa. “Clustering in weighted networks,” *Social Networks*, 31(2):155-163, May 2009.

Georgia Tech College of Computing
Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Local transitivity coefficient

```
double * calculateTransitivityLocal(graph *G)
```

- Input:
  - graph data structure
- Output:
  - Returns the local transitivity coefficient for each vertex in an array

Tore Opsahl and Pietro Panzarasa. “Clustering in weighted networks,” *Social Networks*, 31(2):155-163, May 2009.

Georgia Tech College of Computing Computational Science and Engineering

Pacific Northwest NATIONAL LABORATORY

Local clustering coefficient

```
double * calculateClusteringLocal(graph *G)
```

- Input:
  - graph data structure
- Output:
  - Returns the local clustering coefficient for each vertex in an array

Tore Opsahl and Pietro Panzarasa. “Clustering in weighted networks,” *Social Networks*, 31(2):155-163, May 2009.