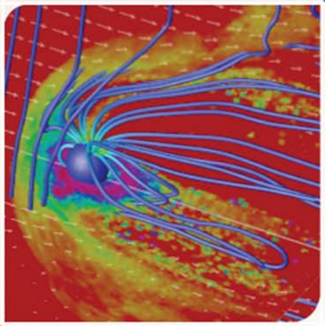
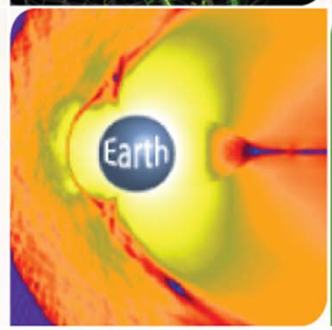
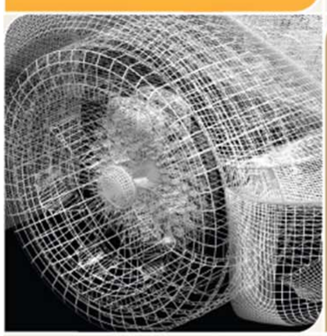


Massive Analytics for Streaming Graph Problems

David A. Bader



Georgia Tech
College of Computing
Computational Science and Engineering

Outline

- Overview
- Cray XMT
- Streaming Data Analysis
  - STINGER data structure
- Tracking Clustering Coefficients
- Tracking Connected Components
- Parallel Graph Frameworks

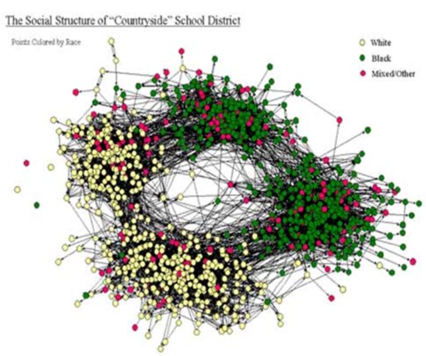
STING Initiative:
Focusing on Globally Significant Grand Challenges

- Many globally-significant grand challenges can be modeled by Spatio-Temporal Interaction Networks and Graphs (or “STING”).
- Emerging real-world graph problems include
  - detecting community structure in large social networks,
  - defending the nation against cyber-based attacks,
  - improving the resilience of the electric power grid, and
  - detecting and preventing disease in human populations.

- Unlike traditional applications in computational science and engineering, solving these problems at scale often raises new research challenges because of sparsity and the lack of locality in the massive data, design of parallel algorithms for massive, streaming data analytics, and the need for new exascale supercomputers that are energy-efficient, resilient, and easy-to-program.

Center for Adaptive Supercomputing Software

- WyomingClerk, launched July 2008
- Pacific-Northwest Lab
  - Georgia Tech, Sandia, WA State, Delaware
- The newest breed of supercomputers have hardware set up not just for speed, but also to better tackle large networks of seemingly random data. And now, a multi-institutional group of researchers has been awarded over $14 million to develop software for these supercomputers. Applications include anywhere complex webs of information can be found: from internet security and power grid stability to complex biological networks.



CASS-MT TASK 7: Analysis of Massive Social Networks

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.

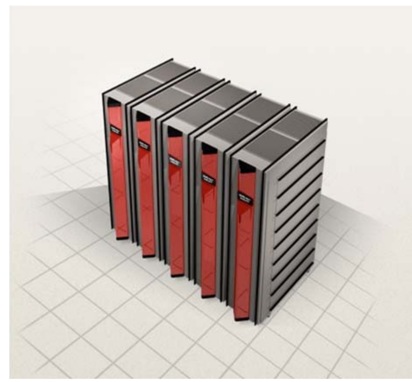
Description

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



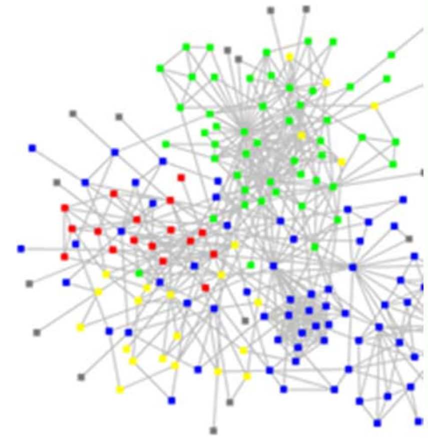


Image Courtesy of Cray, Inc.

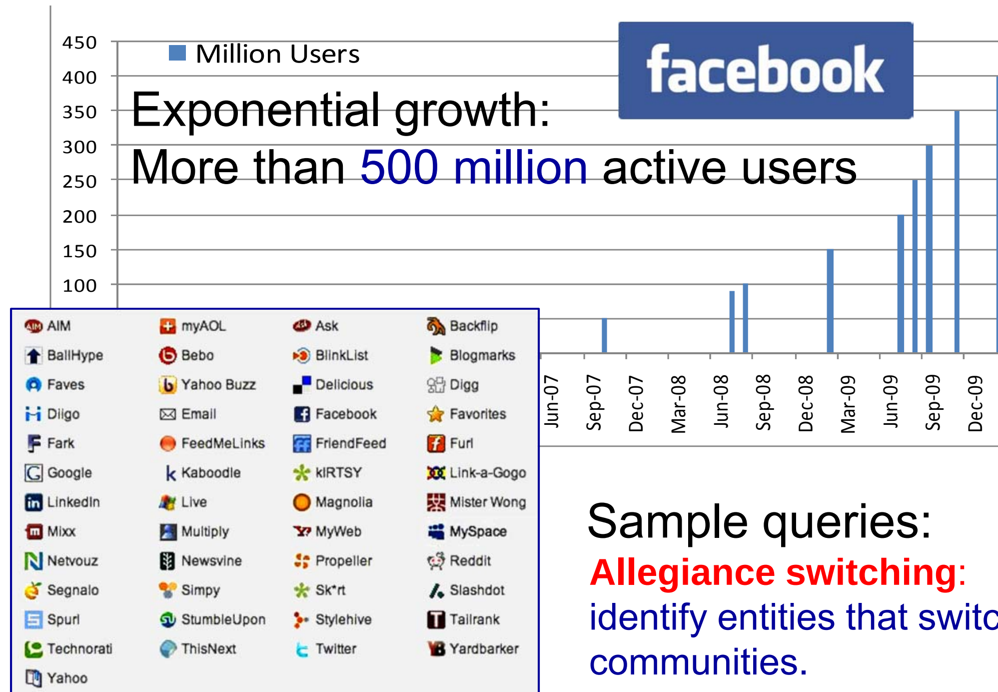
Our research is focusing on temporal analysis, answering questions about changes in global properties (*e.g.* diameter) as well as local structures (communities, paths).

David A. Bader (CASS-MT Task 7 LEAD)
David Ediger, Karl Jiang, Jason Riedy

Exascale Streaming Data Analytics: Real-world challenges

All involve analyzing massive streaming complex networks:

- Health care → disease spread, detection and prevention of epidemics/pandemics (e.g. SARS, Avian flu, H1N1 “swine” flu)
- Massive social networks → understanding communities, intentions, population dynamics, pandemic spread, transportation and evacuation
- Intelligence → business analytics, anomaly detection, security, knowledge discovery from massive data sets
- Systems Biology → understanding complex life systems, drug design, microbial research, unravel the mysteries of the HIV virus; understand life, disease,
- Electric Power Grid → communication, transportation, energy, water, food supply
- Modeling and Simulation → Perform full-scale economic-social-political simulations



Ex: discovered minimal changes in O(billions)-size complex network that could hide or reveal top influencers in the community

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

REQUIRES PREDICTING / INFLUENCE CHANGE IN REAL-TIME AT SCALE

Open Questions: Algorithmic Kernels for Spatio-Temporal Interaction Graphs and Networks (STING)

- Traditional graph theory:
  - Graph traversal (e.g. breadth-first search)
  - S-T connectivity
  - Single-source shortest paths
  - All-pairs shortest paths
  - Spanning Tree
  - Connected Components
  - Biconnected Components
  - Subgraph isomorphism (pattern matching)
  - ….

Hierarchy of Interesting Graph 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...**

Graph Analytics for Social Networks

- Are there new graph techniques? Do they parallelize? Can the computational systems (algorithms, machines) handle massive networks with millions to billions of individuals? Can the techniques tolerate noisy data, massive data, streaming data, etc. …
- Communities may overlap, exhibit different properties and sizes, and be driven by different models
  - Detect communities (static or emerging)
  - Identify important individuals
  - Detect anomalous behavior
  - Given a community, find a representative member of the community
  - Given a set of individuals, find the best community that includes them

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

Open Questions for Massive Analytic Applications

- How do we diagnose the health of streaming systems?
- Are there new analytics for massive spatio-temporal interaction networks and graphs (STING)?
- Do current methods scale up from thousands to millions and billions?
- How do I model massive, streaming data streams?
- Are algorithms resilient to noisy data?
- How do I visualize a STING with O(1M) entities? O(1B)? O(100B)? with scale-free power law distribution of vertex degrees and diameter =6 …
- Can accelerators aid in processing streaming graph data?
- How do we leverage the benefits of multiple architectures (e.g. map-reduce clouds, and massively multithreaded architectures) in a single platform?

Limitations of Current Analysis and Viz 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...

Architectural Requirements for the Efficient Graph Analysis (Challenges)

- Runtime is dominated by latency
  - Random accesses to global address space
  - Perhaps many at once
- Essentially no computation to hide memory costs
- Access pattern is data dependent
  - Prefetching unlikely to help
  - Usually only want small part of cache line
- Potentially abysmal locality at all levels of memory hierarchy

Architectural Requirements for the Efficient Graph Analysis (Desired Features)

- A large memory capacity
- Low latency / high bandwidth
  - For small messages!
- Latency tolerant
- Light-weight synchronization mechanisms
- Global address space
  - No graph partitioning required
  - Avoid memory-consuming profusion of ghost-nodes
  - No local/global numbering conversions

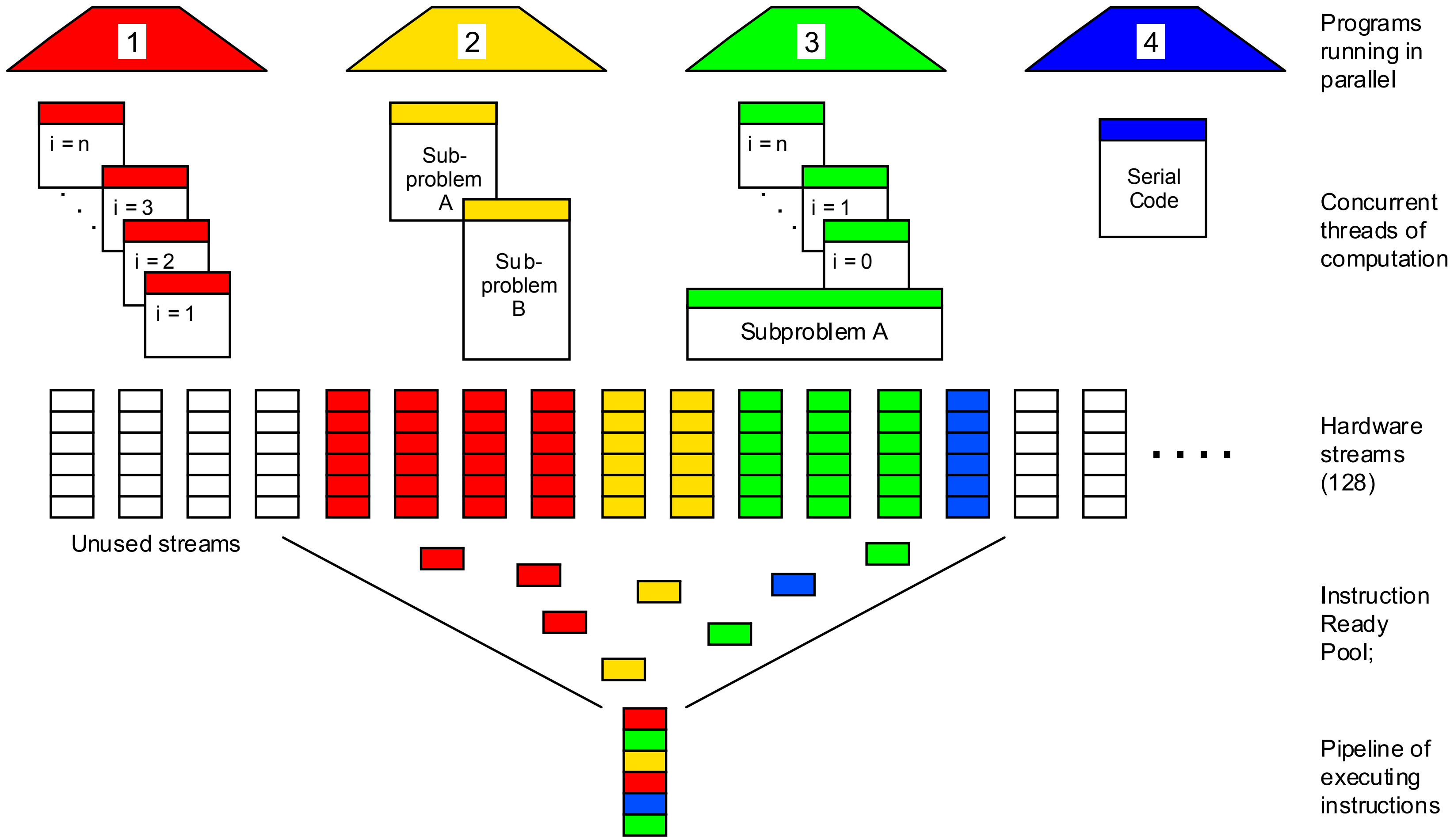
The Cray XMT

- 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

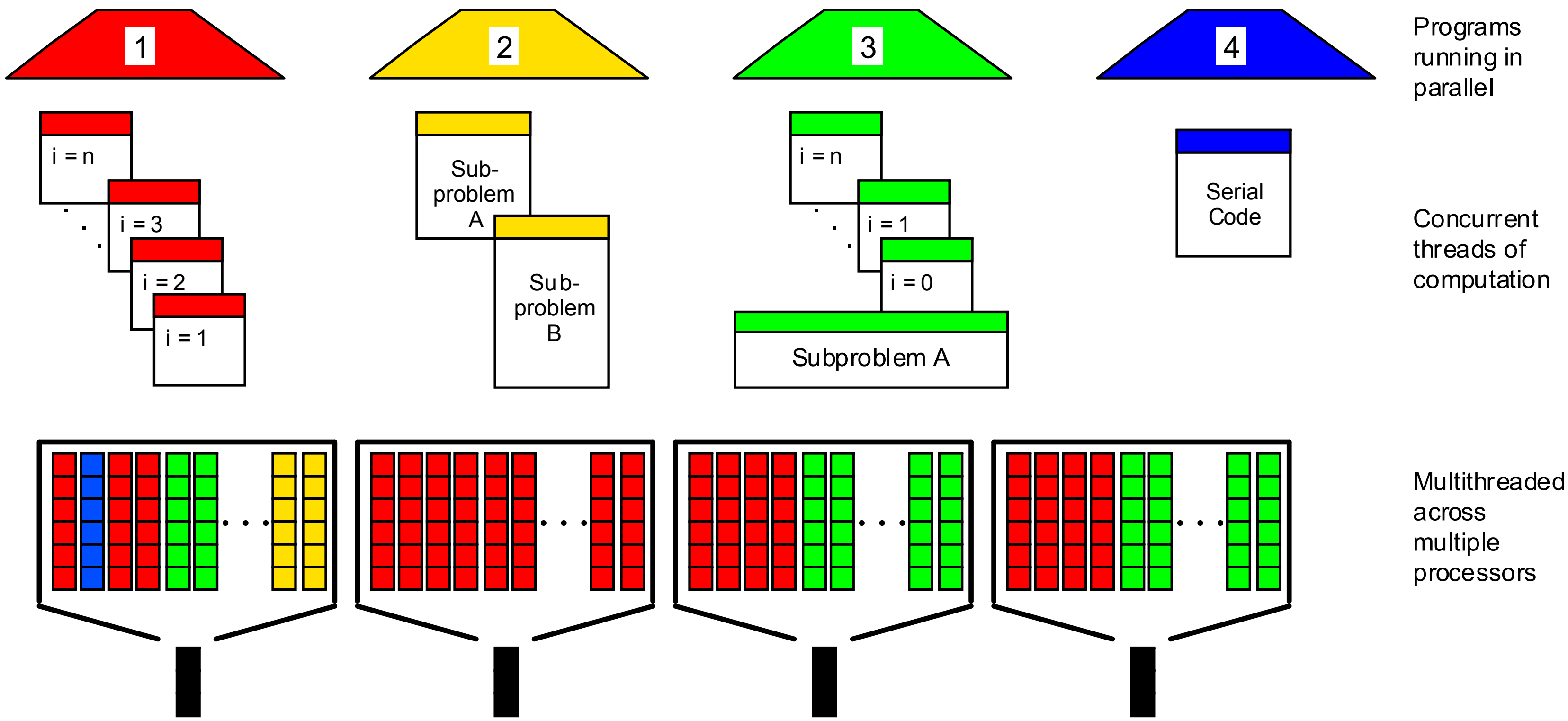
- GraphCT currently tested on a 128 processor XMT: 16K threads
  - 1 TB of globally shared memory
- → PilgrimShadow, SundryMaximal

XMT ThreadStorm Processor (logical view)



Slide Credit: Cray, Inc.

XMT ThreadStorm System (logical view)



Slide Credit: Cray, Inc.

What is not important on XMT

- Placing data near computation
- Modifying shared data
- Accessing data in order
- Using indirection or linked data-structures
- Partitioning program into independent, balanced computations
- Using adaptive or dynamic computations
- Minimizing synchronization operations

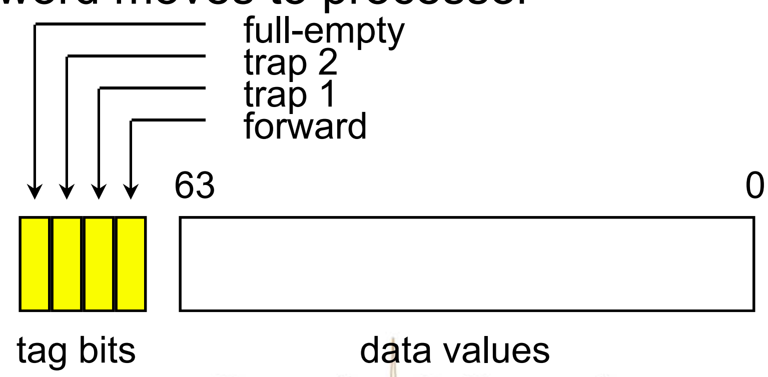
XMT memory

- Shared memory
  - Some memory can be reserved as local memory at boot time
  - Only compiler and runtime system have access to local memory

- Memory module cache
  - Decreases latency and increases bandwidth
  - No coherency issues

- 8 word data segments randomly distributed across the memory system
  - Eliminates stride sensitivity and hotspots
  - Makes programming for data locality impossible
  - Segment moves to cache, but only word moves to processor

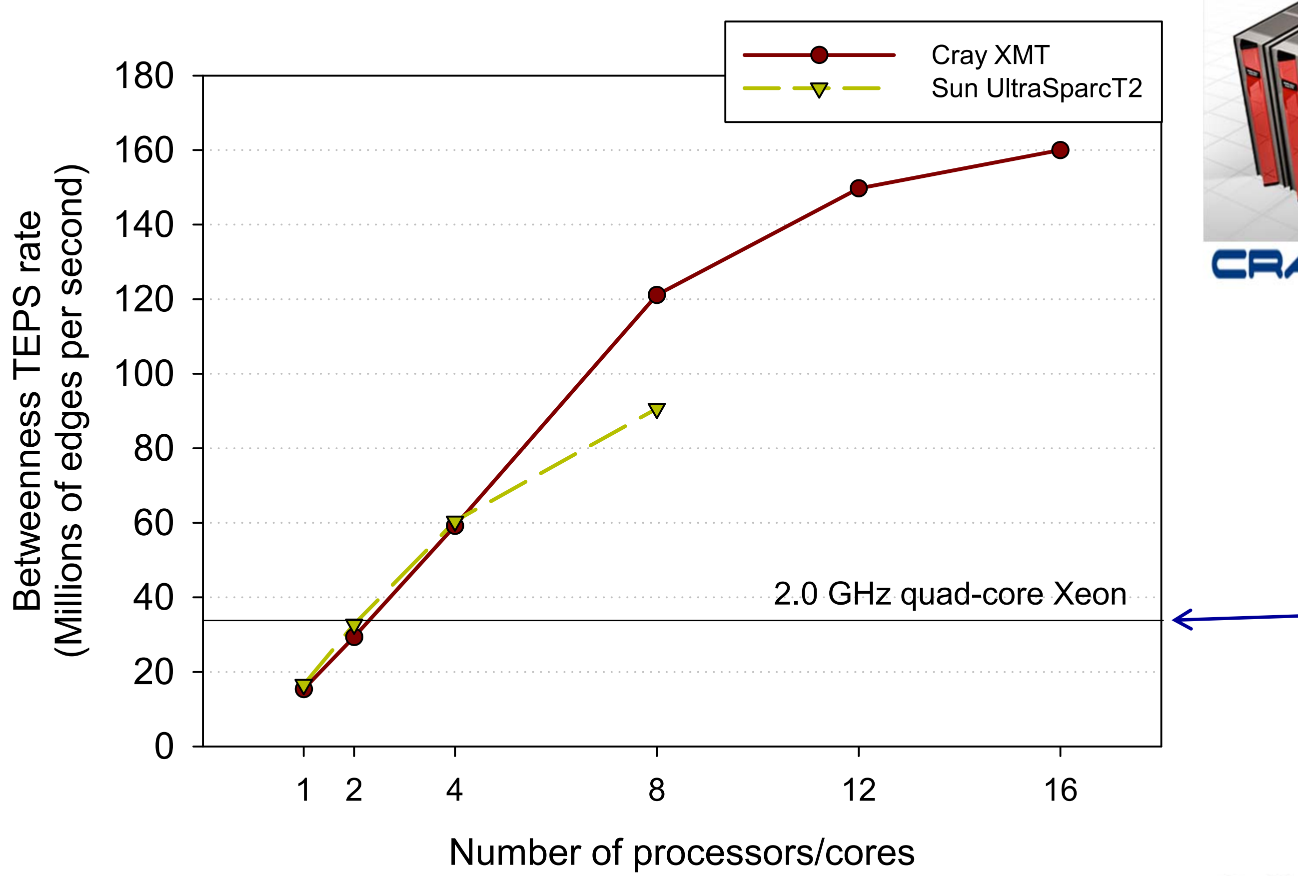
- Full/empty bits on all data words



Slide Credit: Cray, Inc.

Graph Analysis Performance: Multithreaded (Cray XMT) vs. Cache-based multicore

- SSCA#2 network, SCALE 24 (16.77 million vertices and 134.21 million edges.)



STREAMING DATA ANALYSIS

Current Unserved Applications

- Separate the "good" from the "bad"
  - Spam. Frauds. *Irregularities*.
  - Pick news from world-wide events tailored to interests *as the events & interests change*.
- Identify and track changes
  - Disease outbreaks. Social trends. Utility & service changes during weather events.
- Discover new relationships
  - Similarities in scientific publications.
- Predict upcoming events
  - Present advertisements *before* a user searches.

Shared features: Relationships are abstract. Physical locality is only one aspect, unlike physical simulation.

Streaming Data Characteristics

- The data expresses unknown (*i.e.* unpredictable) relationships.
  - The relationships are not necessarily bound by or related to physical proximity.
  - Arranging data for *storage* locality often is equivalent to the desired analysis.
  - There may be temporal proximity… That is a question we want to answer!

Streaming Data Characteristics

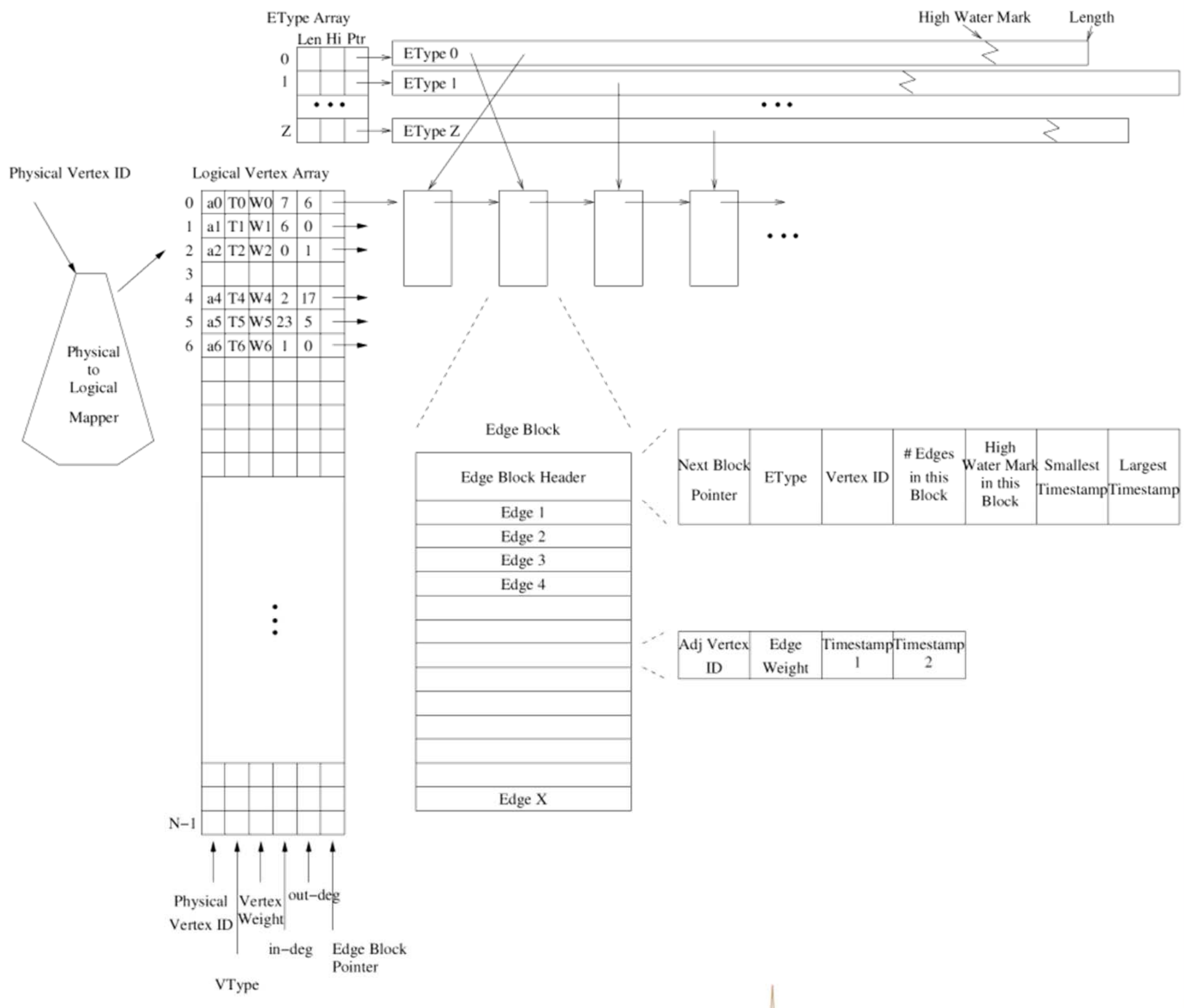
- The data expresses relationships *partially*.
  - Personal friendship is not the same as on-line "friendship."
  - Streams often are lossy or contain errors.
    - Real links may be dropped, false links added.
    - Time synchronization is difficult.
  - Need to determine error models…

STING Extensible Representation (STINGER)

- 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

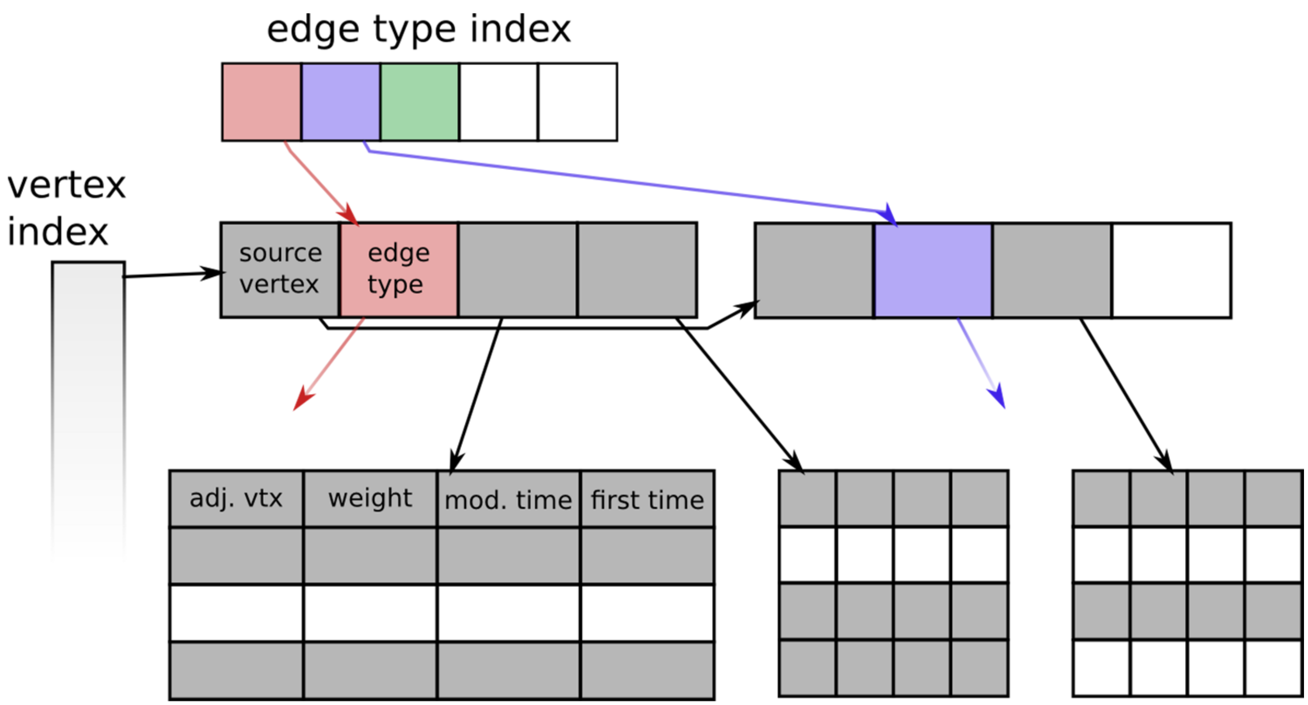


STINGER

- Georgia Tech implementation runs in parallel on Cray XMT and OpenMP/multicore desktop
- Shows little or no performance overhead for many kernels
- Recent publication using STINGER:
  - **David Ediger, Karl Jiang, Jason Riedy, and David A. Bader**, "Massive Streaming Data Analytics: A Case Study with Clustering Coefficients." MTAAP, Atlanta, GA, 2010.
  - Demonstrates good performance for small graphs on Intel Nehalem and large streaming datasets on the Cray XMT

STINGER: Extending the Hybrid

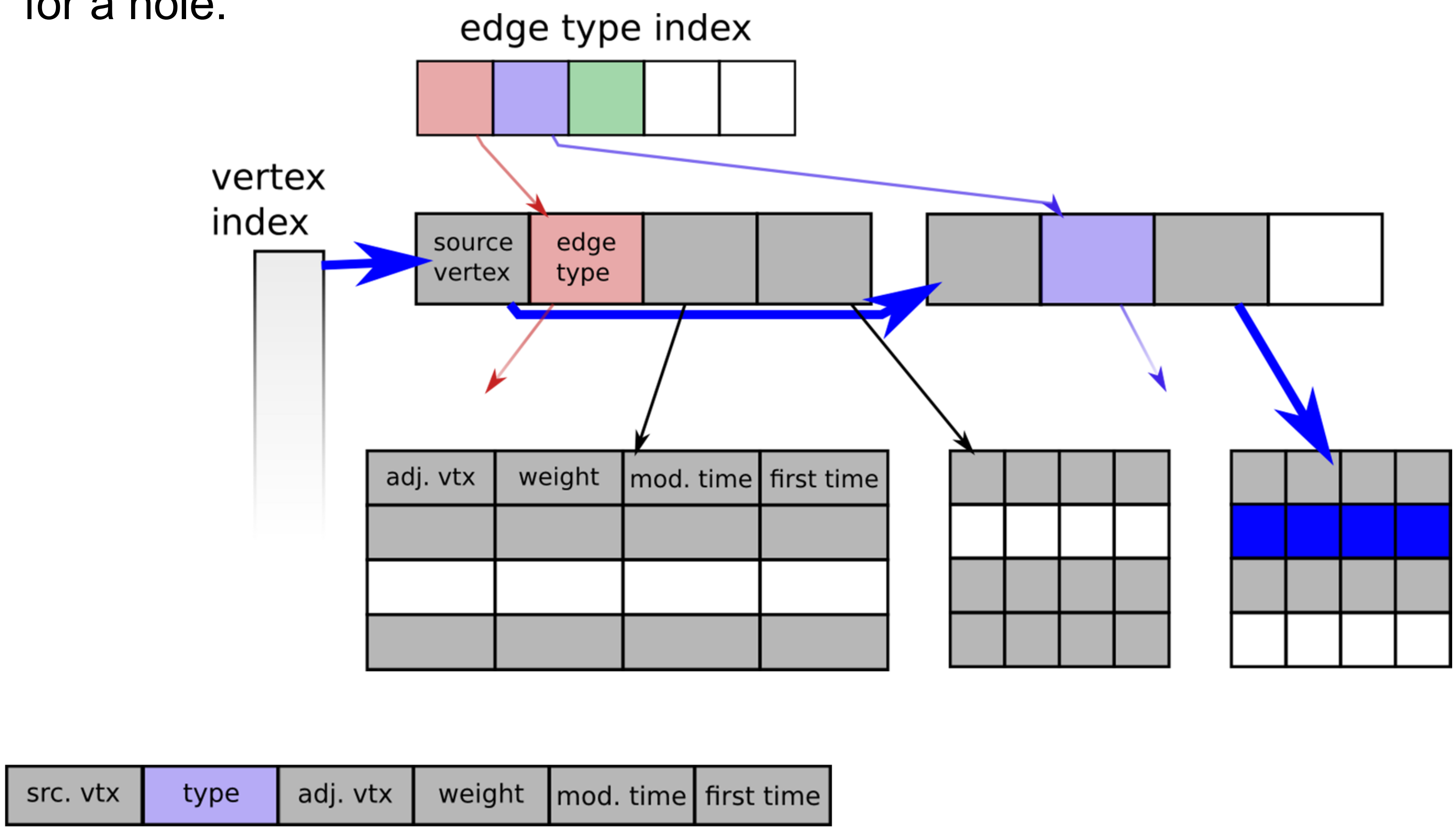
D. Bader, J. Berry, A. Amos-Binks, D. Chavarría-Miranda, C. Hastings, K. Madduri, S. Poulos, "STINGER: Spatio-Temporal Interaction Networks and Graphs (STING) Extensible Representation"



Many applications need different *kinds* of relationships / edges. The hybrid approach can accommodate those by separating different kinds' edge arrays. An additional level of indirection permits fast access by source vertex or edge type.

STINGER: Edge Insertion

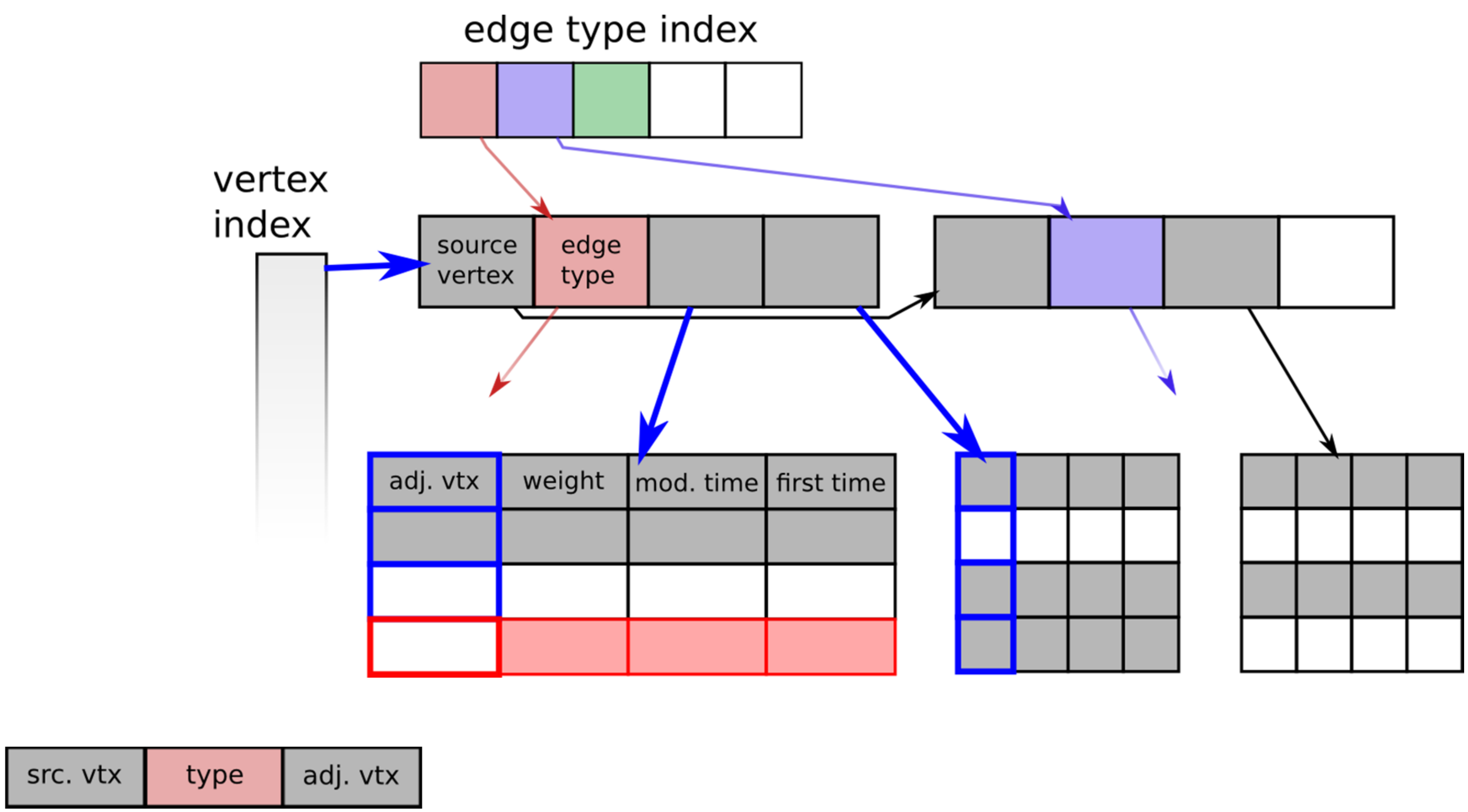
Insertion (best case): From the source vertex, skip to the edge type, then search for a hole.



Worst case: Allocate a new block and add to the list...

STINGER: Edge Removal

Removal: Find the edge. Remove by negating the adj. vertex. Atomic store.

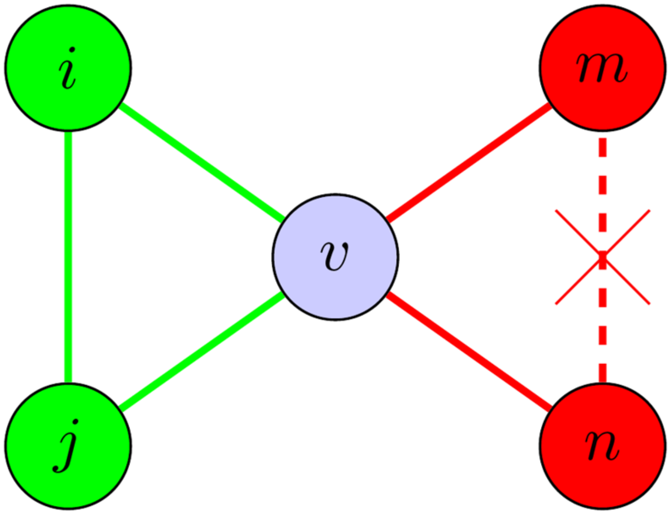


If insertion sets the adj. vertex > 0 after other updates, insertion will appear atomic.

TRACKING CLUSTERING COEFFICIENTS

Case Study: Clustering Coefficients

- Used as a measure of "small-worldness."
- Larger clustering coefficient → more inter-related
- Roughly, the ratio of actual triangles to possible triangles around a vertex.

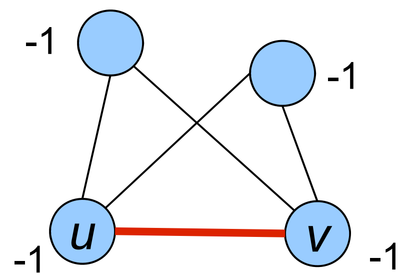


- Defined in terms of ***triplets***.
- *i-j-v* is a ***closed triplet*** (triangle).
- *m-v-n* is an ***open triplet***.
- Clustering coefficient

  \# closed triplets / # all triplets
- Locally, count around *v*.
- Globally, count across entire graph.
  - Multiple counting cancels (3/3=1)

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.



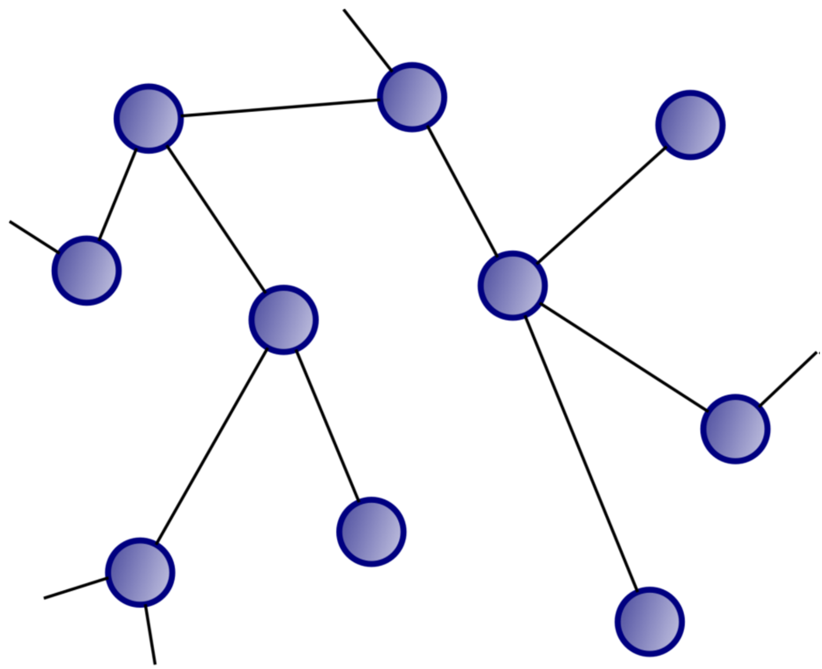
- Need a fast method for updating the triangle counts, degrees when an edge is inserted or deleted.
  - Dynamic data structure for edges & degrees: STINGER
  - Rapid triangle count update algorithms: exact and **approximate**
- "Massive Streaming Data Analytics: A Case Study with Clustering Coefficients." Ediger, David, Karl Jiang, E. Jason Riedy, and David A. Bader. MTAAP 2010, Atlanta, GA, April 2010.

Batching Graph Changes

- Individual graph changes for local properties will not expose much parallelism. Need to consider many actions at once for performance.
- Conveniently, batches of actions also amortize transfer overhead from the data source.
  - Common paradigm in network servers (*c.f.* SEDA: Staged Event-Driven Arch.)
- Even more conveniently, clustering coefficients lend themselves to batches.
  - Final result independent of action ordering between edges.
  - Can reconcile all actions on a single edge within the batch.

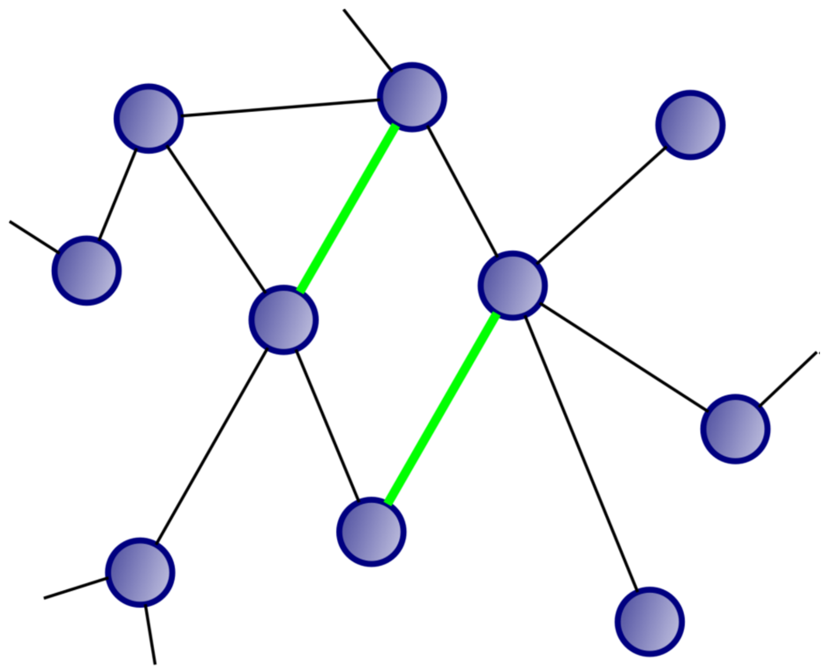
Updating Triplet Counts

Consider a starting graph:



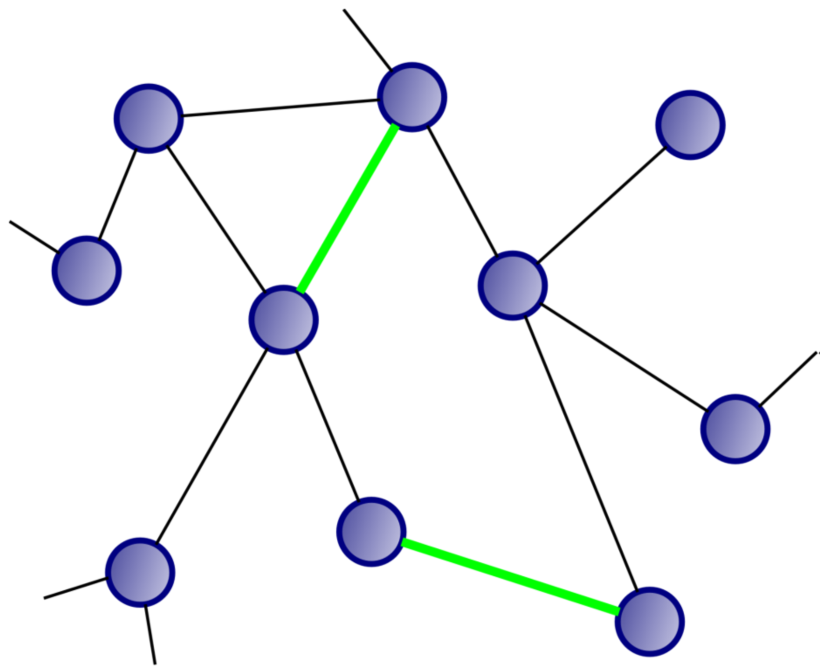
Updating Triplet Counts

Insert two edges (green):



Updating Triplet Counts

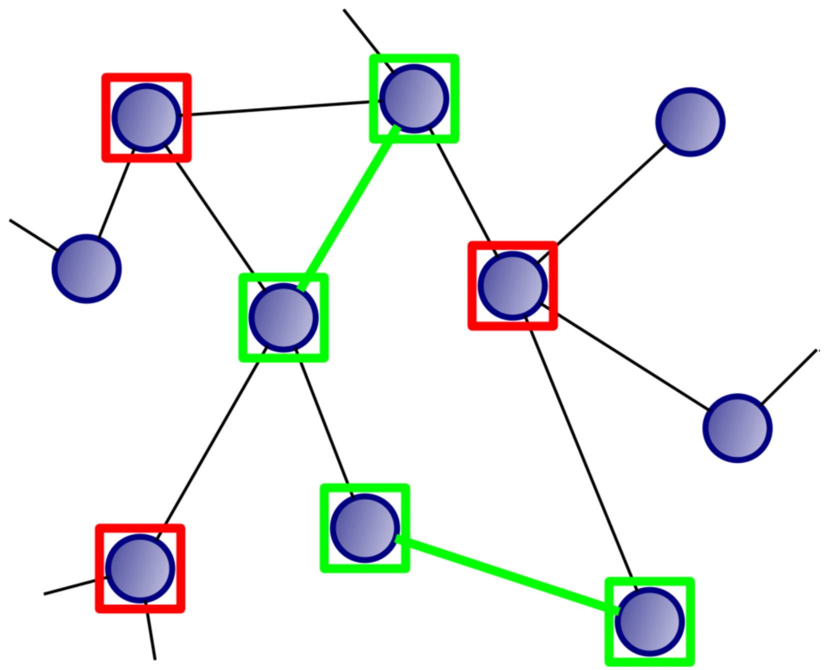
Consider adjacent vertices (green boxes):



The *open* triplet count is a function only of degree. Update the local *open* triplet count for each green boxed vertex.

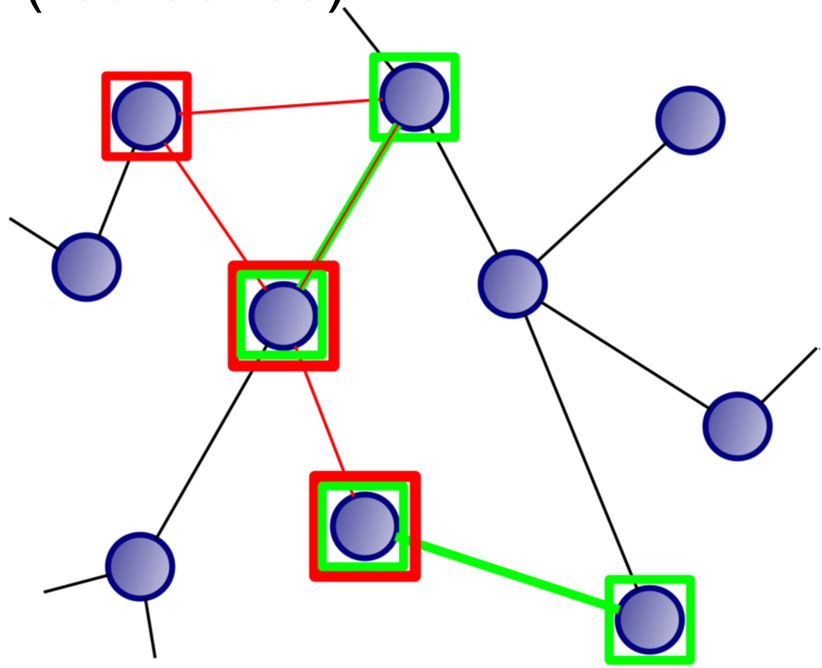
Updating Triplet Counts

Now examine all vertices adjacent to those:



Updating Triplet Counts

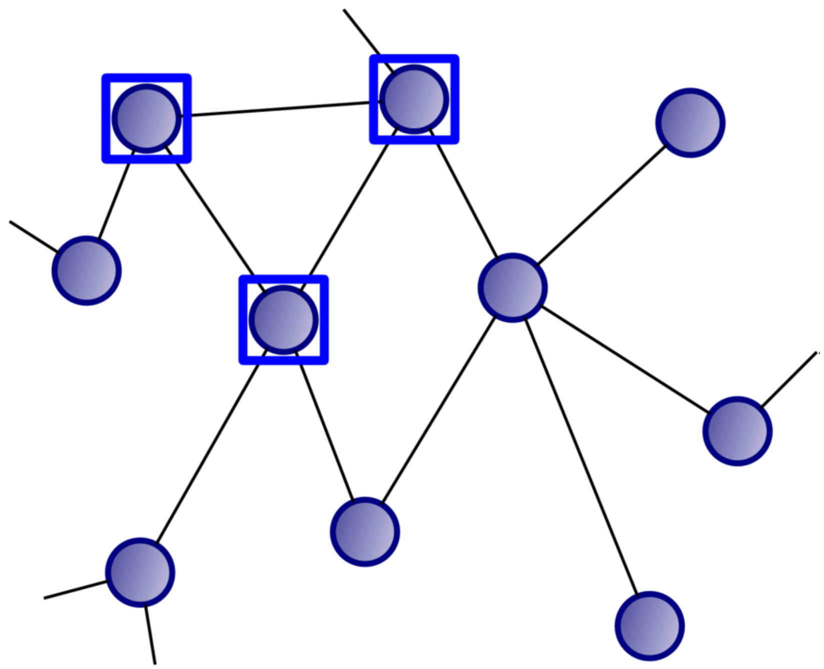
Prune consideration to vertices adjacent to *two* newly attached vertices (red boxes):



- Being adjacent to two newly joined edges is necessary for being part of a new closed triple (triangle) although not sufficient.
- From each red boxed vertex, search for a new edge opposite it. Only need to search the red edges.

Updating Triplet Counts

Update closed triplet (triangle) counts for found triangles (blue boxes):



- ***Note:*** Only accessed edges adjacent to the newly inserted edges. Batching *reduces* work over individual actions.
- Glossed over cases (two, three new edges in triangle); none need extra searches.
- Technique also handles edge removal.

Updating clustering coefficients

- Using RMAT as a graph and edge stream generator.
  - 16M vertices, 537M initial
  - 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:

64p Cray XMT

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

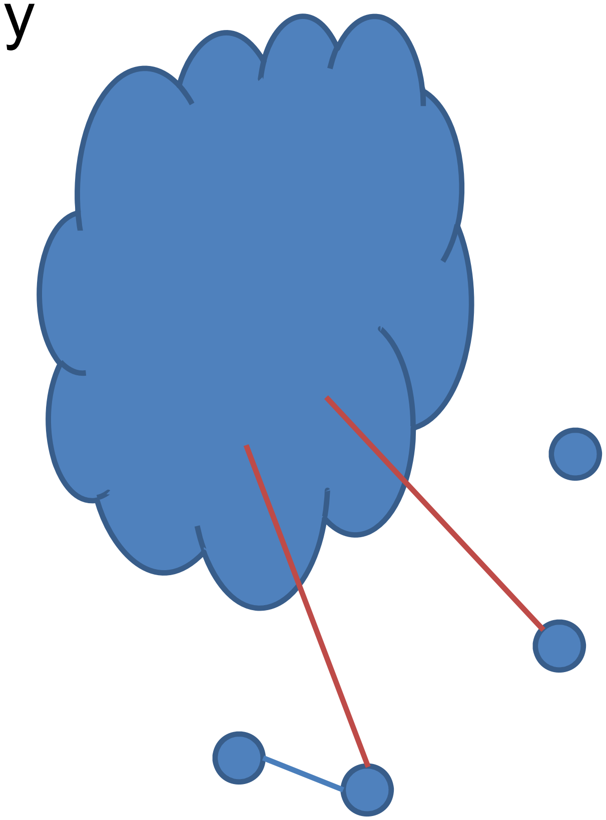
- Approx: Summarizes adj. structure with a Bloom filter, 100% accuracy in this test.
- STINGER overhead is minimal; most time in spent metric.

TRACKING CONNECTED COMPONENTS

Tracking connected components

Goals:

- Given a graph and a sequence of many edge insertions and fewer removals, track the components.
- Provide component membership information for *many* other kernels, including diameter, searches, etc.
- Evaluate STINGER's efficiency on the XMT.



Adapting for STINGER: Original static code

```
while (!is_empty(stack, &top)) {
   int64_t k, myStart, myEnd;
   u = pop(stack, &top);
   myStart = off[u];
   myEnd = off[u+1];
   for (k = myStart; k < myEnd; k++) {
           v = ind[k];
           if (int_fetch_add(marks + v, 1) == 0) {
               d[v] = my_root;
               push(v, stack, &top);
           } else {
               if (!(d[v]==d[my_root])) {
                   int64_t t = int_fetch_add(&cross_count, 1);
                   crossU[t] = u;
                   crossV[t] = v;
           } } } }
```

`ind[]` : end vertex array
`off[]` : vertex offset into `ind[]`

- **Leveraging GraphCT base...**

Georgia Tech College of Computing

Adapting for STINGER: static connected components

```
while (!is_empty(stack, &top)) {
    int64_t k, myStart, myEnd;
    size_t md;
    u = pop(stack, &top);
    deg_u = stinger_outdegree(S, u);
    myStart = stinger_int64_fetch_add(&head, deg_u);
    myEnd = myStart + deg_u;
    stinger_gather_typed_successors(S, 0, u, &md, &neighbors[myStart], deg_u);
    for (k = myStart; k < myEnd; k++) {
            v = neighbors[k];
            if (stinger_int64_fetch_add(marks + v, 1) == 0) {
                d[v] = my_root;
                push(v, stack, &top);
            } else {
                if (!(d[v]==d[my_root])) {
                    int64_t t = stinger_int64_fetch_add(&cross_count, 1);
                    crossU[t] = u;
                    crossV[t] = v;
            } } } }
```

S : STINGER data structure
`neighbors[]` : pre-allocated buffer
head : end pointer into `neighbors[]`

- **Assuming a pre-allocated buffer, neighbors.**

Adapting for STINGER: static connected components

```
while (!is_empty(stack, &top)) {
    int64_t k, myStart, myEnd;
    size_t md;
    u = pop(stack, &top);
    deg_u = stinger_outdegree(S, u);
    myStart = stinger_int64_fetch_add(&head, deg_u);
    myEnd = myStart + deg_u;
    stinger_gather_typed_successors(S, 0, u, &md, &neighbors[myStart], deg_u);
    for (k = myStart; k < myEnd; k++) {
            v = neighbors[k];
            if (stinger_int64_fetch_add(marks + v, 1) == 0) {
                d[v] = my_root;
                push(v, stack, &top);
            } else {
                if (!(d[v]==d[my_root])) {
                    int64_t t = stinger_int64_fetch_add(&cross_count, 1);
                    crossU[t] = u;
                    crossV[t] = v;
            } } } }
```

S : STINGER data structure
`neighbors[]` : pre-allocated buffer
head : end pointer into `neighbors[]`

- **Portable spelling for atomic operations.**

Adapting for STINGER: static connected components

```
while (!is_empty(stack, &top)) {
    int64_t k, myStart, myEnd;
    size_t md;
    u = pop(stack, &top);
    deg_u = stinger_outdegree(S, u);
    myStart = stinger_int64_fetch_add(&head, deg_u);
    myEnd = myStart + deg_u;
    stinger_gather_typed_successors(S, 0, u, &md, &neighbors[myStart], deg_u);
    for (k = myStart; k < myEnd; k++) {
            v = neighbors[k];
            if (stinger_int64_fetch_add(marks + v, 1) == 0) {
                d[v] = my_root;
                push(v, stack, &top);
            } else {
                if (!(d[v]==d[my_root])) {
                    int64_t t = stinger_int64_fetch_add(&cross_count, 1);
                    crossU[t] = u;
                    crossV[t] = v;
            } } } }
```

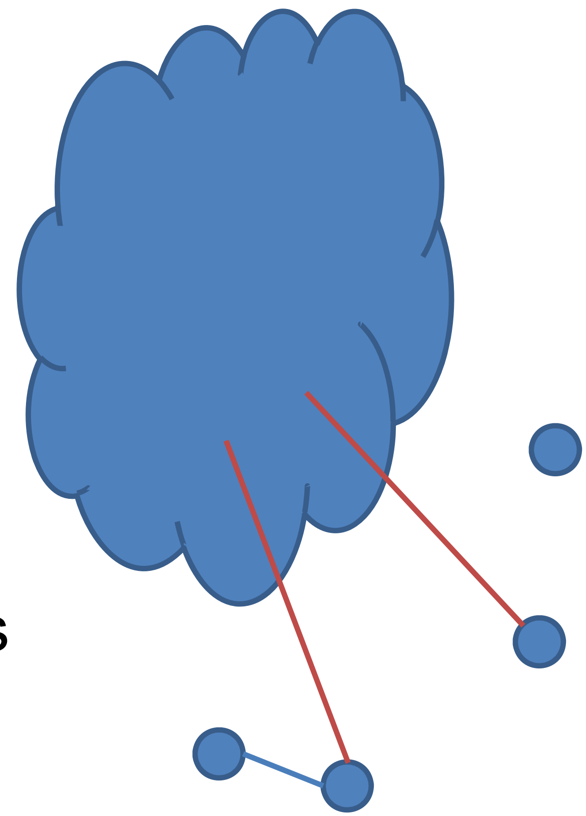
S : STINGER data structure
`neighbors[]` : pre-allocated buffer
head : end pointer into `neighbors[]`

- **Copying neighbors isolates from dynamic changes.**
- **Keeps compiler-optimizable loop structure.**

Tracking connected components

Assumptions:

- Scale-free network: most changes are within one large component.
- Edge additions: primarily merge small component into the one large component.
  - Do not need access to graph...
- Deletions: rarely disconnect components
  - Needs static connected components algorithm to look for changes
  - Heuristics may avoid the full run

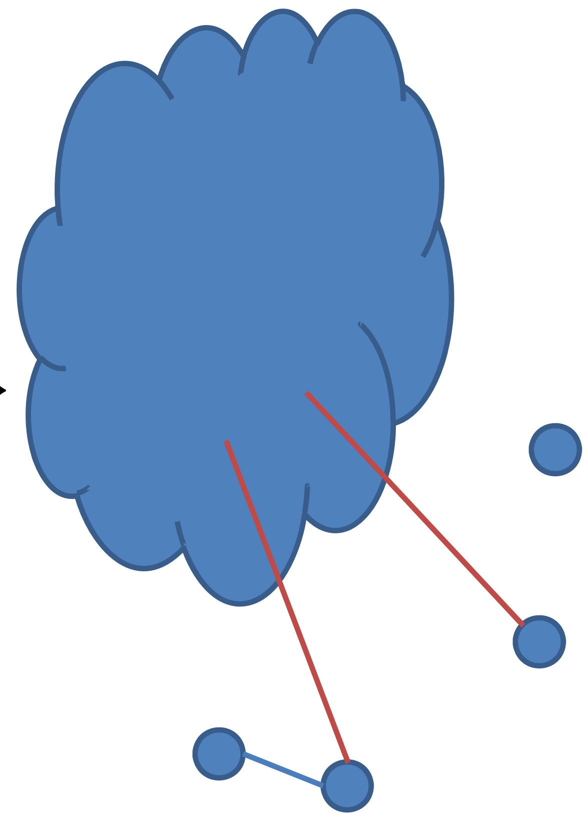


Georgia Tech College of Computing

Tracking connected components

Edge addition (in batches):

- Relabel batch of additions with component numbers.
- Collapse the graph, removing self-edges. Any edges that remain cross components.
- Compute components of component ↔ component graph. Relabel smaller into larger.
- Problem size reduces from number of changes to number of components
- Proceeds concurrently with STINGER modification

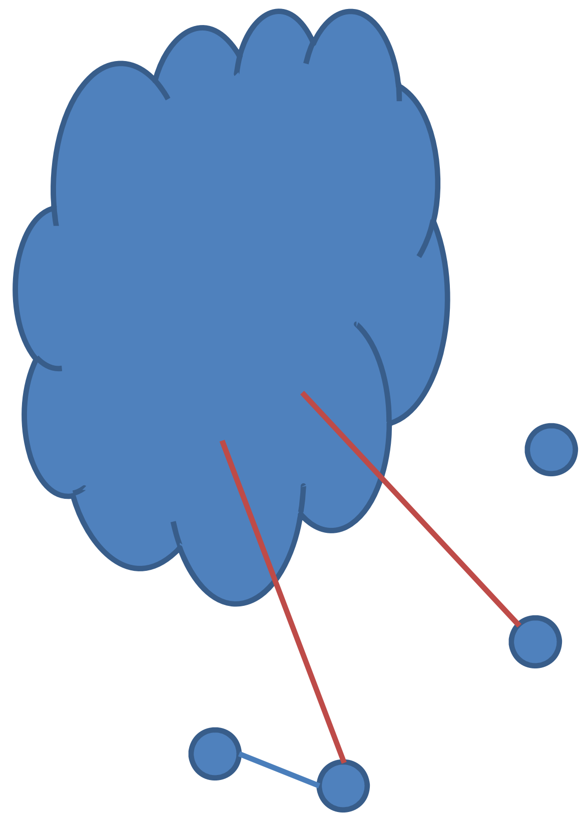


Georgia Tech College of Computing

Tracking connected components

Edge deletion:

- A single deletion in a batch will trigger static connected components
- *Heuristic:* Accumulate *n* deletions before recomputation
- *Heuristic:* Perform truncated breadth first search *k* steps away from each endpoint. Null intersection means recomputation.
- *Heuristic:* Deleted edges that provably do not form triangles within a batch can be ignored. (In progress.)
- Can tune heuristics for data



Performance Tuning Case Study

- From prior work, XMT can process ~100,000 updates/sec
- Initial implementation: 1100 updates/sec
    - Execution time scaled with batch size – not good
- Removed memory allocation & parallelized all loops
    - No observed change in performance
- Instrumented 13 loops & function calls for timing
- qsort(): majority of time & does not scale
    - (Note: qsort is standard, programmers expect it to work reasonably well.)
- Experimented with several parallel sorting algorithms
    - While we have experience optimizing parallel sorting on the MTA, our current need requires a different sort.

Sorting edges for batching

- Two-valued sort of edges: 1st by source, then by destination
- Expecting 1,000 to 100,000 pairs (power law distribution)
- Recursive quicksort with futures
  - Not enough parallelism
- Sandia merge sort
  - Single batch is too small to take advantage
- Fastest: Bucket sort by source, then concurrent qsort()s
  - ~30 lines of code
  - Parallel loops, linear recurrences, reductions
- Result: improved from 1.1K updates/sec to 150K upd./sec

Experimental Results: Connected components

Synthetic, Power Law Input: 16M vertices, 135M edges

16 proc. on the Cray XMT (20 batches of 50,000) 6.25% deletions

	Updates / sec
Edge adds only	77,600
Edge adds + STINGER	54,000
Adds + Deletes + STINGER	5,900
Threshold 50K deletes*	46,500

*Threshold recomputes static connected components after 50,000 deletes are accumulated

Experimental Results on Intel Nehalem-EP

Synthetic, Power Law Input: **1M** vertices, **8M** edges **(SMALL)**

16 threads (batches of 1) 6.25% deletions

	Updates / sec
Edge adds only	2,770,000
Edge adds + STINGER	537,000
Adds + Deletes + STINGER	397,000
Truncated BFS-5*	440,000

*Performs breadth first search 5 steps from source and destination and recomputes when intersection is null. Note that the graph is very sparse, diameter much larger than 5.

STINGER findings

- Static code easy to convert
- Maximum graph size reduced by ~16x
  - 1 TB Cray XMT: 268M vertices → 16M vertices
  - 12 GB Intel: 16M vertices → 1M vertices
  - Metadata & **block overheads**

    •Blocks sized to store >100 edges, these examples have average <10 per vertex.

    •Reduction of these overheads in progress.
- With large batch sizes, running static connected components on XMT faster than many parallel truncated breadth first searches (heuristic)

PARALLEL GRAPH FRAMEWORKS

Parallel Graph Frameworks

- SNAP
  - Georgia Tech, Bader/Madduri
- Parallel Boost Graph Library
  - Indiana, Lumsdaine
- MultiThreaded Graph Library (MTGL)
  - Sandia, Berry
- GraphCT
  - Georgia Tech, Ediger, Riedy, Jiang, Bader
- STINGER
  - Georgia Tech, Bader, Riedy, Ediger, Jiang

SNAP: Small-world Network Analysis and Partitioning

- New parallel framework for small-world network analysis
- 10-100x faster than existing approaches
- Can process graphs with billions of vertices and edges
- Open-source
- [Bader/Madduri]

Interaction data

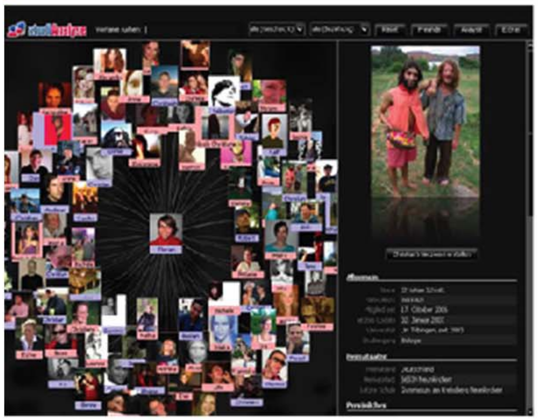
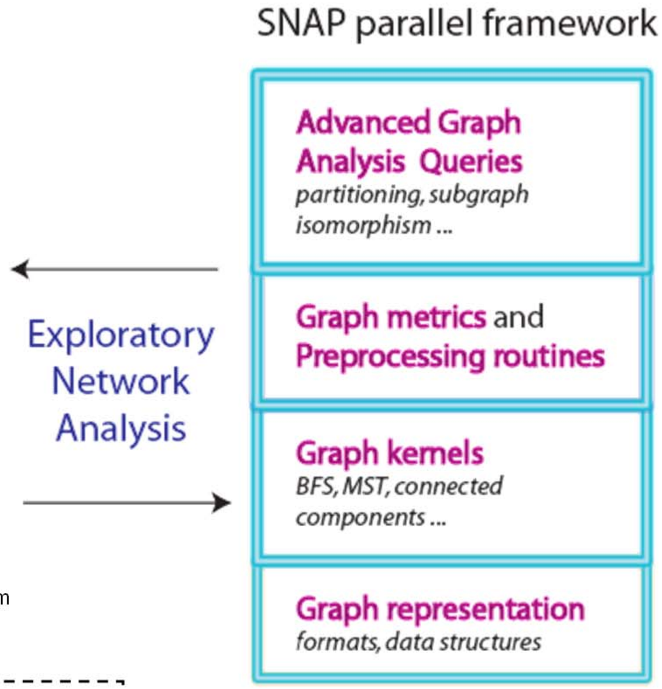


Image Source: visualcomplexity.com



snap-graph.sourceforge.net

Parallel Boost Graph Library

- C++ library for parallel & distributed graph computations
- Provides similar data structures and algorithms as sequential Boost Graph Library
- Developed by Indiana University in 2005
- Scales up to 100 processors for some algorithms on ideal graphs
  - see earlier slide on PBGL performance

http://www.osl.iu.edu/research/pbgl/

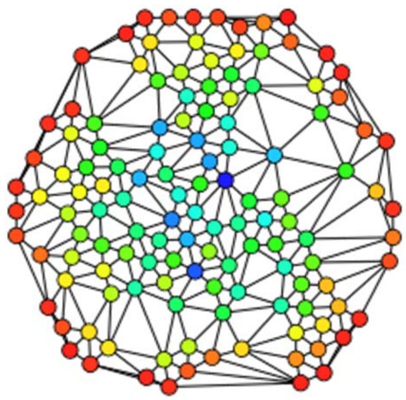
Multithreaded Graph Library (MTGL)

- Under development at Sandia National Labs
- Primitives for “visiting” a vertex
  - Get data about the vertex
  - Retrieve a list of all adjacencies
- Abstract connector to graph representation
- Tailored for Cray XMT, but portable to multicore using Qthreads
- Programmer must still understand code that is generated in order to get good performance on the XMT

https://software.sandia.gov/trac/mtgl

GraphCT (Georgia Tech)

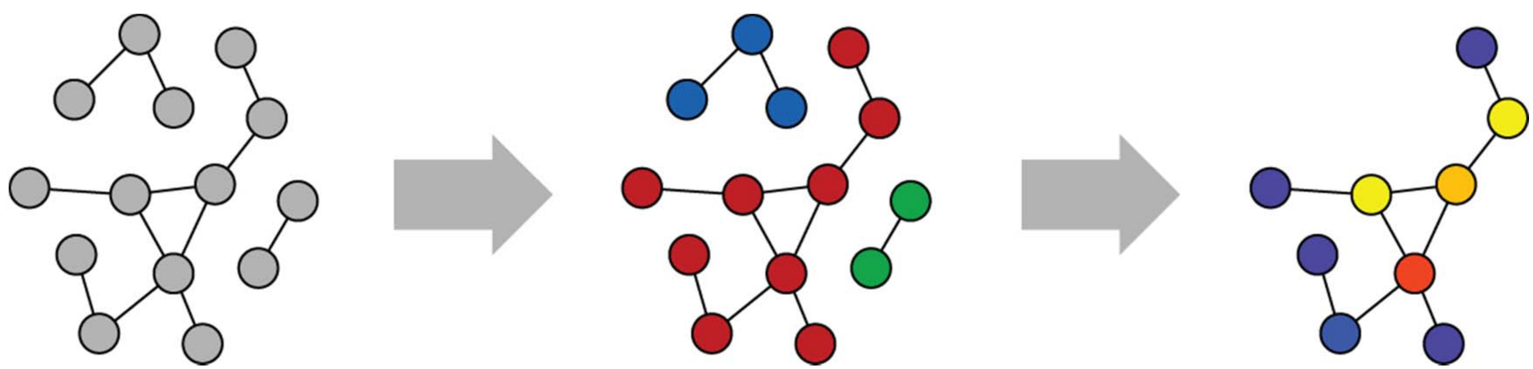
- Graph Characterization Toolkit
- Efficiently summarizes and analyzes static graph data
- Built for large multithreaded, shared memory machines like the Cray XMT
- Increases productivity by decreasing programming complexity
- Classic metrics & state-of-the-art kernels
- Works on many types of graphs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



Dynamic spatio-temporal graph

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 128 processors) on the Cray XMT



GraphCT: 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
```

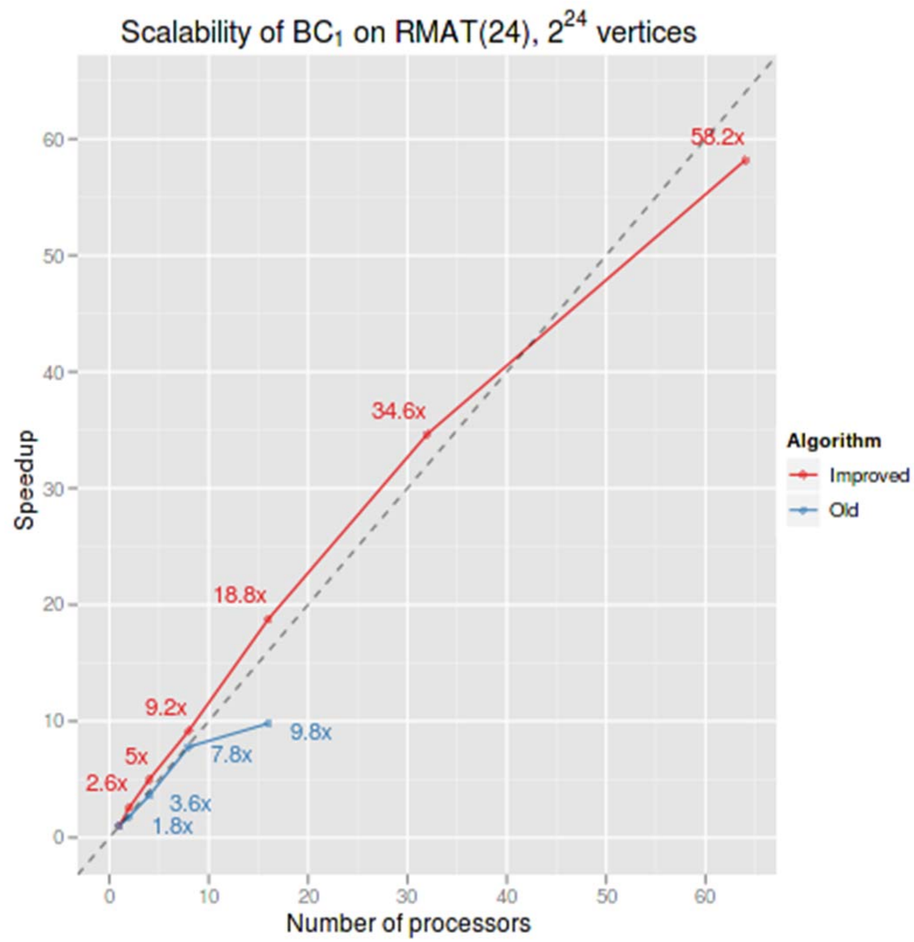
GraphCT Functions

Name	Name
RMAT graph generator	Modularity Score
Degree distribution statistics	Conductance Score
Graph diameter	st-Connectivity
Maximum weight edges	Delta-stepping SSSP
Connected components	Bellman-Ford
Component distribution statistics	GTriad Census
Vertex Betweenness Centrality	SSCA2 Kernel 3 Subgraphs
Vertex k-Betweenness Centrality	Greedy Agglomerative Clustering
Multithreaded BFS	Minimum spanning forest
Edge-divisive Betweenness-based Comm Detection (pBD)	Clustering coefficients
Lightweight Binary Graph I/O	DIMACS Text Input

Key
Included
In Progress
Proposed/Available

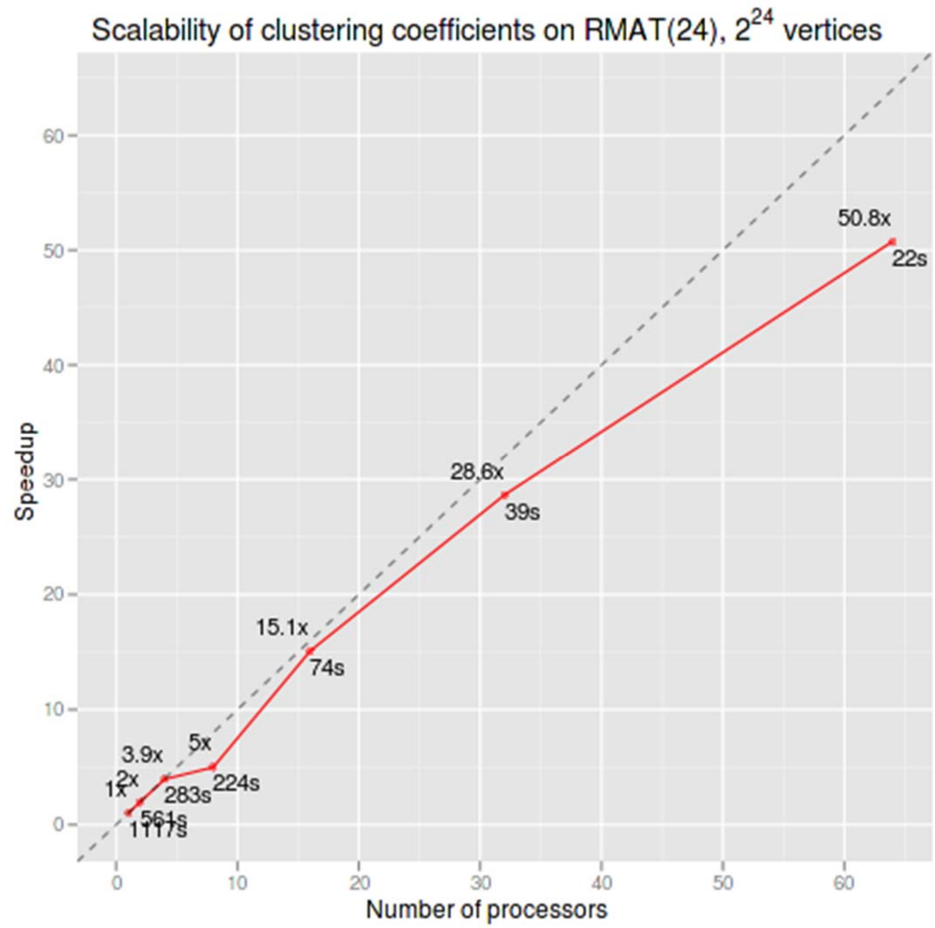
Scalability of *k*-Betweenness Centrality in GraphCT

- 58x speed-up on 64 processor XMT
- Synthetic power-law graph with 16M vertices & 135M edges
- Able to run 20 breadth-first searches in parallel



Clustering Coefficients in GraphCT

- A measure of the connectivity of the network
- Used in the definition of "small world"
- 51x speed-up on 64 processor XMT
- Total time: 22 secs for 16M vertices



Bader, Related Recent Publications (2005-2008)

- D.A. Bader, G. Cong, and J. Feo, “On the Architectural Requirements for Efficient Execution of Graph Algorithms,” *The 34th International Conference on Parallel Processing* (ICPP 2005), pp. 547-556, Georg Sverdrups House, University of Oslo, Norway, June 14-17, 2005.
- D.A. Bader and K. Madduri, “Design and Implementation of the HPCS Graph Analysis Benchmark on Symmetric Multiprocessors,” *The 12th International Conference on High Performance Computing* (HiPC 2005), D.A. Bader *et al.*, (eds.), Springer-Verlag LNCS 3769, 465-476, Goa, India, December 2005.
- 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.
- D.A. Bader and K. Madduri, “Parallel Algorithms for Evaluating Centrality Indices in Real-world Networks,” *The 35th International Conference on Parallel Processing* (ICPP 2006), Columbus, OH, August 14-18, 2006.
- K. Madduri, D.A. Bader, J.W. Berry, and J.R. Crobak, “Parallel Shortest Path Algorithms for Solving Large-Scale Instances,” *9th DIMACS Implementation Challenge -- The Shortest Path Problem*, DIMACS Center, Rutgers University, Piscataway, NJ, November 13-14, 2006.
- K. Madduri, D.A. Bader, J.W. Berry, and J.R. Crobak, “An Experimental Study of A Parallel Shortest Path Algorithm for Solving Large-Scale Graph Instances,” *Workshop on Algorithm Engineering and Experiments* (ALENEX), New Orleans, LA, January 6, 2007.
- J.R. Crobak, J.W. Berry, K. Madduri, and D.A. Bader, “Advanced Shortest Path Algorithms on a Massively-Multithreaded Architecture,” *First Workshop on Multithreaded Architectures and Applications* (MTAAP), Long Beach, CA, March 30, 2007.
- D.A. Bader and K. Madduri, “High-Performance Combinatorial Techniques for Analyzing Massive Dynamic Interaction Networks,” *DIMACS Workshop on Computational Methods for Dynamic Interaction Networks*, DIMACS Center, Rutgers University, Piscataway, NJ, September 24-25, 2007.
- D.A. Bader, S. Kintali, K. Madduri, and M. Mihail, “Approximating Betewenness Centrality,” The *5th Workshop on Algorithms and Models for the Web-Graph* (WAW2007), San Diego, CA, December 11-12, 2007.
- David A. Bader, Kamesh Madduri, Guojing Cong, and John Feo, “Design of Multithreaded Algorithms for Combinatorial Problems,” in S. Rajasekaran and J. Reif, editors, *Handbook of Parallel Computing: Models, Algorithms, and Applications*, CRC Press, Chapter 31, 2007.
- Kamesh Madduri, David A. Bader, Jonathan W. Berry, Joseph R. Crobak, and Bruce A. Hendrickson, “Multithreaded Algorithms for Processing Massive Graphs,” in D.A. Bader, editor, *Petascale Computing: Algorithms and Applications*, Chapman & Hall / CRC Press, Chapter 12, 2007.
- D.A. Bader and K. Madduri, “SNAP, Small-world Network Analysis and Partitioning: an open-source parallel graph framework for the exploration of large-scale networks,” *22nd IEEE International Parallel and Distributed Processing Symposium* (IPDPS), Miami, FL, April 14-18, 2008.

Bader, Related Recent Publications (2009-2010)

- S. Kang, D.A. Bader, "An Efficient Transactional Memory Algorithm for Computing Minimum Spanning Forest of Sparse Graphs," 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), Raleigh, NC, February 2009.
- Karl Jiang, 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), Vienna, Austria, September 2009.
- Kamesh Madduri, David Ediger, Karl Jiang, David A. Bader, Daniel Chavarría-Miranda. "A Faster Parallel Algorithm and Efficient Multithreaded Implementations for Evaluating Betweenness Centrality on Massive Datasets." 3rd Workshop on Multithreaded Architectures and Applications (MTAAP), Rome, Italy, May 2009.
- David A. Bader, et al. "STINGER: Spatio-Temporal Interaction Networks and Graphs (STING) Extensible Representation." 2009.
- David Ediger, Karl Jiang, E. Jason Riedy, and David A. Bader. "Massive Streaming Data Analytics: A Case Study with Clustering Coefficients," Fourth Workshop in Multithreaded Architectures and Applications (MTAAP), Atlanta, GA, April 2010.
- Seunghwa Kang, David A. Bader. "Large Scale Complex Network Analysis using the Hybrid Combination of a MapReduce cluster and a Highly Multithreaded System:," Fourth Workshop in Multithreaded Architectures and Applications (MTAAP), Atlanta, GA, April 2010.
- David Ediger, Karl Jiang, Jason Riedy, David A. Bader, Courtney Corley, Rob Farber and William N. Reynolds. "Massive Social Network Analysis: Mining Twitter for Social Good," The 39th International Conference on Parallel Processing (ICPP 2010), San Diego, CA, September 2010.
- Virat Agarwal, Fabrizio Petrini, Davide Pasetto and David A. Bader. "Scalable Graph Exploration on Multicore Processors," *The 22nd IEEE and ACM Supercomputing Conference* (SC10), New Orleans, LA, November 2010.

Collaborators and Acknowledgments

- Jason Riedy, Research Scientist, (Georgia Tech)
- Graduate Students (Georgia Tech):
  - Seunghwa Kang
  - David Ediger
  - Karl Jiang
  - Pushkar Pande
- Bader PhD Graduates:
  - Kamesh Madduri (Lawrence Berkeley National Lab)
  - Guojing Cong (IBM TJ Watson Research Center)
- John Feo and Daniel Chavarría-Miranda (Pacific Northwest National Laboratory)
- Jon Berry, Bruce Hendrickson (Sandia National Labs)
- Jeremy Kepner (MIT Lincoln Laboratory)

Acknowledgment of Support

NATIONAL INSTITUTES OF HEALTH

DEPARTMENT OF ENERGY · UNITED STATES OF AMERICA

NASA

DEPARTMENT OF DEFENSE · UNITED STATES OF AMERICA

DARPA

OAK RIDGE National Laboratory

Pacific Northwest NATIONAL LABORATORY

Sandia National Laboratories

IBM®

Microsoft® Research

NVIDIA.

SONY

intel®

CRAY

Sun microsystems

XILINX®

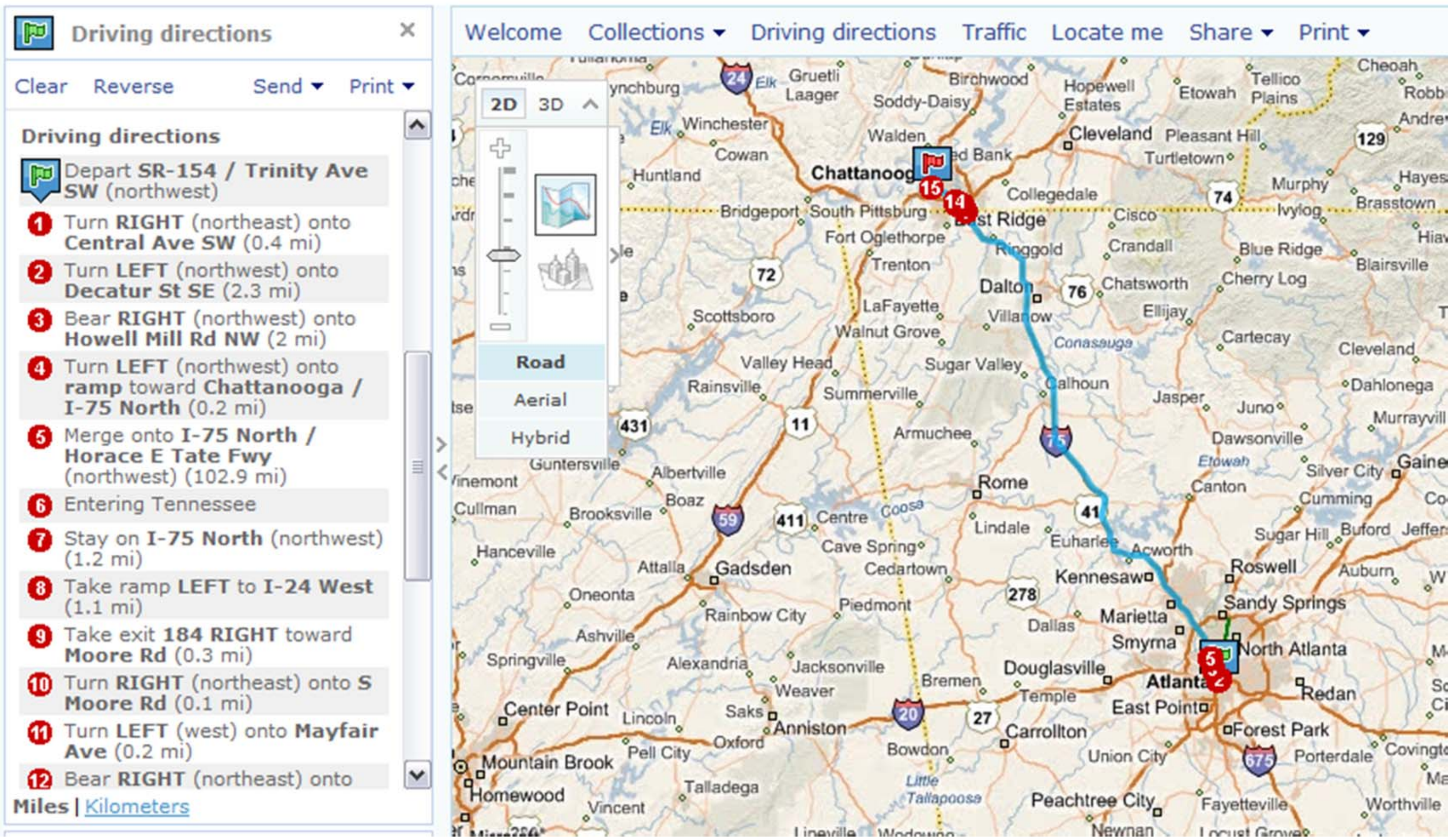
TOSHIBA

Backup Slides

MOTIVATION: NETWORK ANALYSIS

Routing in transportation networks

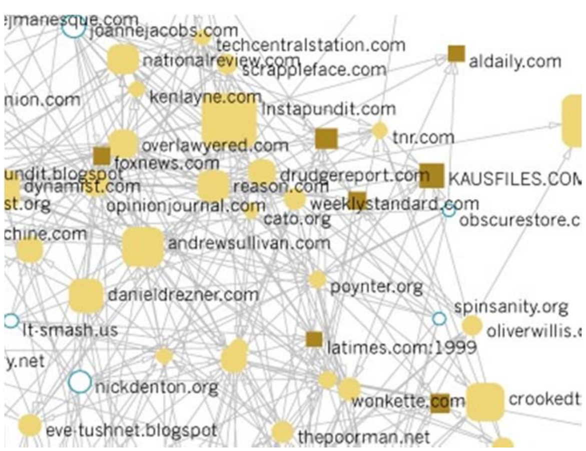
Road networks, Point-to-point shortest paths: 15 seconds (naïve) → 10 microseconds

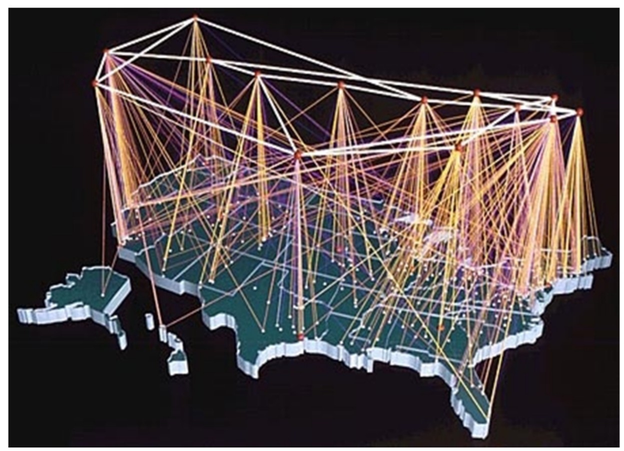


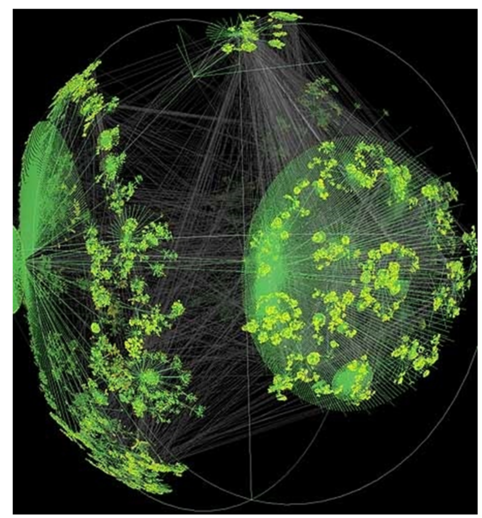
H. Bast et al., “Fast Routing in Road Networks with Transit Nodes”, Science 27, 2007.

Internet and the WWW

- The world-wide web can be represented as a directed graph
  - Web search and crawl: traversal
  - Link analysis, ranking: Page rank and HITS
  - Document classification and clustering
- Internet topologies (router networks) are naturally modeled as graphs







"Google, Citing Attack, Threatens to Exit China"

- This article was reported by Andrew Jacobs, Miguel Helft and John Markoff and written by Mr. Jacobs.
- BEIJING — Google said Tuesday that it would stop cooperating with Chinese Internet censorship and consider shutting down its operations in the country altogether, citing assaults from hackers on its computer systems and China's attempts to "limit free speech on the Web."

Elizabeth Dalziel/Associated Press

The New York Times

12 January 2010

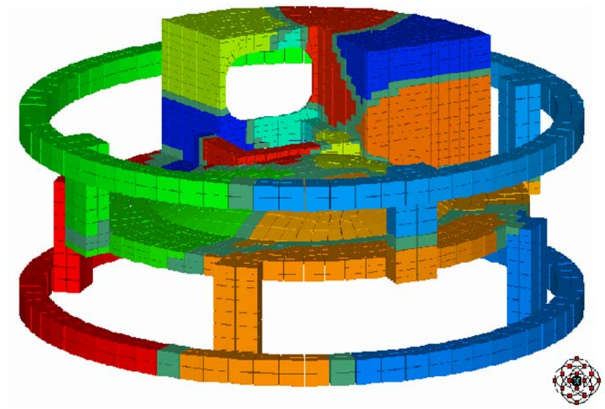
"95% of User Generated Content is spam or malicious"

- Covering the last six months of 2009, the report is based upon the findings of the ThreatSeeker Network which is used to discover, classify and monitor global Internet threats and trends courtesy of something called the Internet HoneyGrid.
- Scanned 40M sites, 10M email messages
  - 13.7% of searches for trending news/buzz words (as defined by Yahoo Buzz & Google Trends) led to malware.
  - 71% of Web sites with malicious code are legitimate sites that have been compromised.
  - 95% of user-generated posts on Web sites are spam or malicious.
  - Consistent with previous years, 51% of malware still connects to host Web sites registered in the United States.
  - China remains second most popular malware hosting country with 17%, but during the last six months Spain jumped into the third place with 15.7% despite never having been in the top 5 countries before.
  - 81% of emails during the second half of the year contained a malicious link.
  - 85.8% of all emails were spam.
  - 35% of malicious Web-based attacks included data-stealing code.
  - 58% of all data-stealing attacks are conducted over the Web.

Source: Davey Winder, 6 February 2010, http://www.daniweb.com/news/story258407.html

Scientific Computing

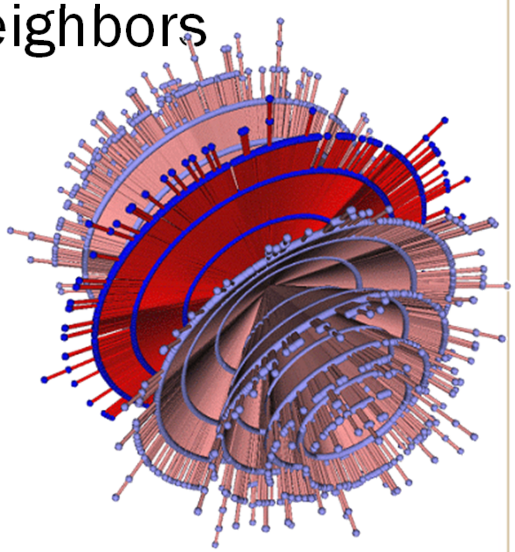
- Reorderings for sparse solvers
  - Fill reducing orderings
    - partitioning, traversals, eigenvectors
  - Heavy diagonal to reduce pivoting (matching)
- Data structures for efficient exploitation of sparsity
- Derivative computations for optimization
  - Matroids, graph colorings, spanning trees
- Preconditioning
  - Incomplete Factorizations
  - Partitioning for domain decomposition
  - Graph techniques in algebraic multigrid
    - Independent sets, matchings, etc.
  - Support Theory
    - Spanning trees & graph embedding techniques



B. Hendrickson, “Graphs and HPC: Lessons for Future Architectures”, http://www.er.doe.gov/ascr/ascac/Meetings/Oct08/Hendrickson%20ASCAC.pdf

Informatics Graphs are Even Tougher

- Very different from graphs in scientific computing!
  - Graphs can be enormous
  - Power-law distribution of the number of neighbors
  - Small world property – no long paths
  - Very limited locality, not partitionable
  - Highly unstructured
  - Edges and vertices have types



Six degrees of Kevin Bacon
Source: Seokhee Hong

- Experience in scientific computing applications provides only limited insight.

Graphs are pervasive in large-scale data analysis

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

Astrophysics

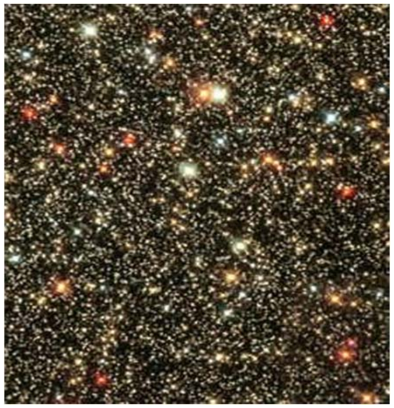
Problem: Outlier detection.
Challenges: massive datasets, temporal variations.
Graph problems: clustering, matching.

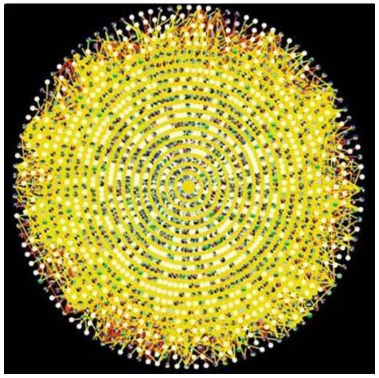
Bioinformatics

Problem: Identifying drug target proteins.
Challenges: Data heterogeneity, quality.
Graph problems: centrality, clustering.

Social Informatics

Problem: Discover emergent communities, model spread of information.
Challenges: new analytics routines, uncertainty in data.
Graph problems: clustering, shortest paths, flows.





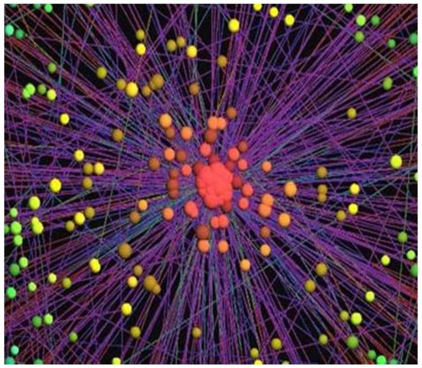


Image sources: (1) http://physics.nmt.edu/images/astro/hst_starfield.jpg
(2,3) www.visualComplexity.com

Data Analysis and Graph Algorithms in Systems Biology

- Study of the interactions between various components in a biological system
- Graph-theoretic formulations are pervasive:
  - Predicting new interactions: modeling
  - Functional annotation of novel proteins: matching, clustering
  - Identifying metabolic pathways: paths, clustering
  - Identifying new protein complexes: clustering, centrality

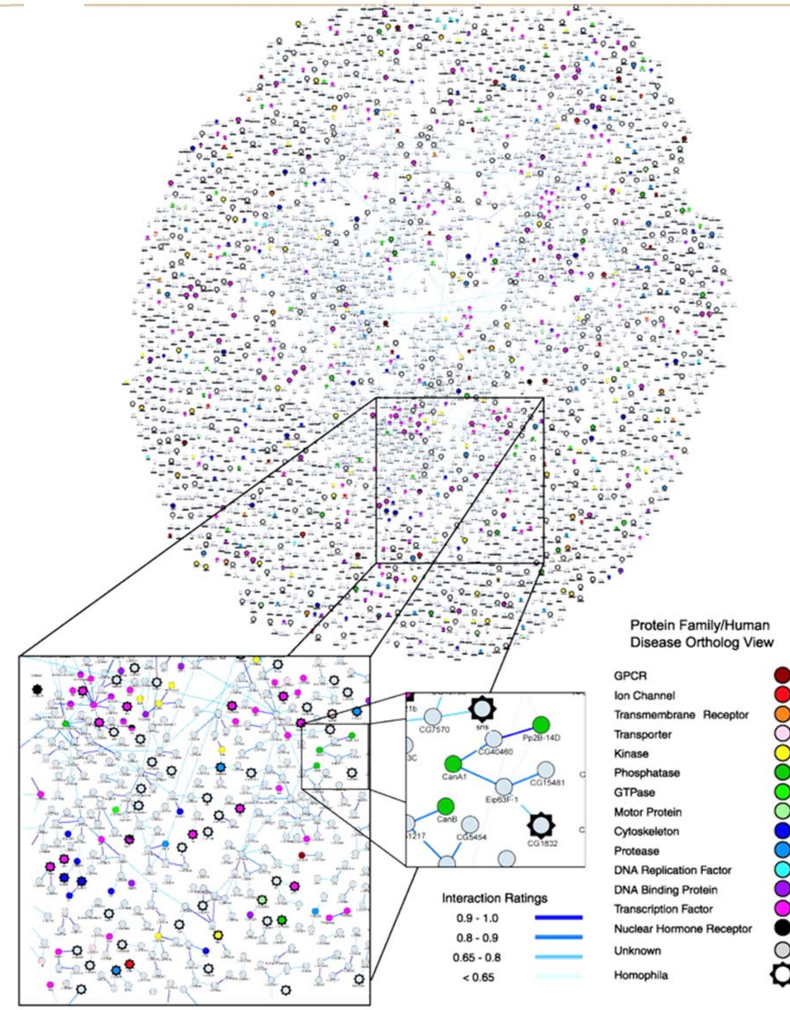


Image Source: Giot et al., "A Protein Interaction Map of *Drosophila melanogaster*", *Science 302*, 1722-1736, 2003.

Graph –theoretic problems in social networks

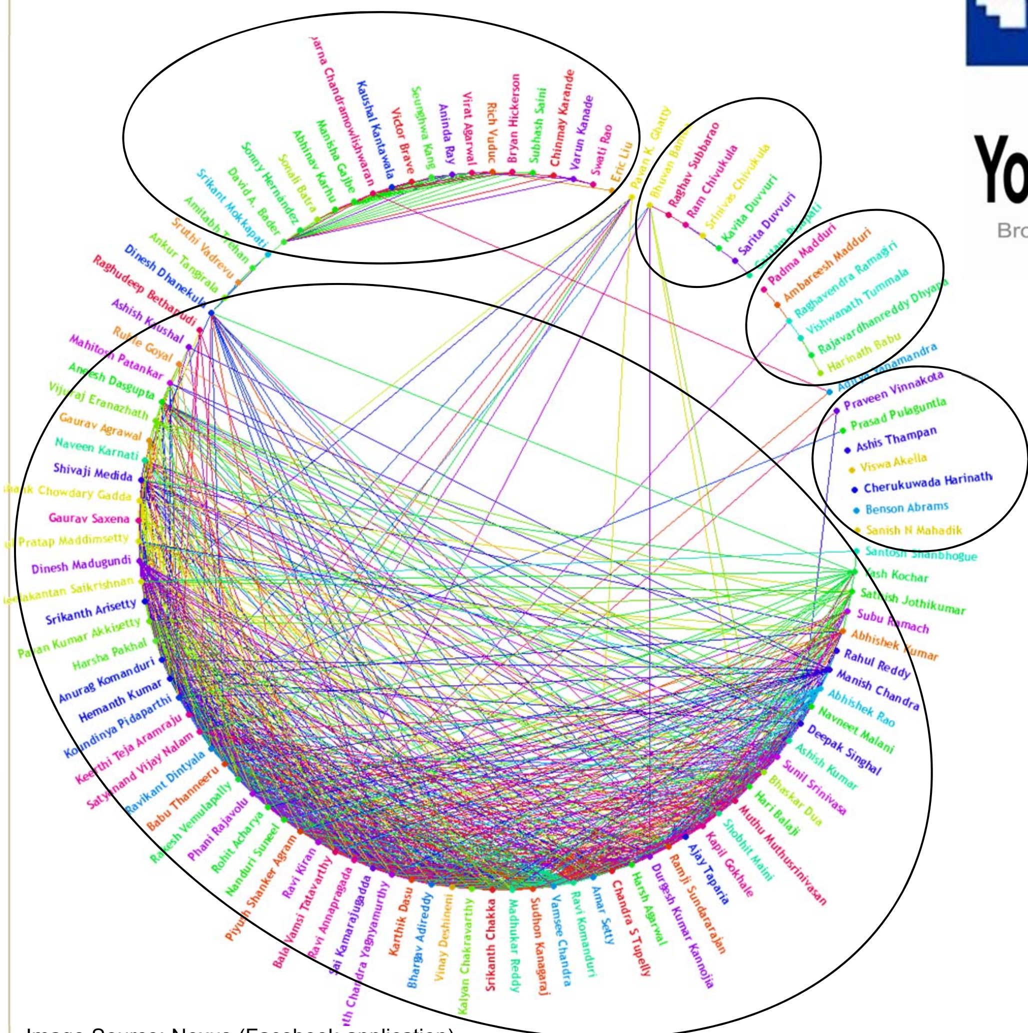


Image Source: Nexus (Facebook application)

- Community identification: clustering
- Targeted advertising: centrality
- Information spreading: modeling

Network Analysis for Intelligence and Survelliance

- [Krebs ’04] Post 9/11 Terrorist Network Analysis from public domain information
- Plot masterminds correctly identified from interaction patterns: centrality

- A global view of entities is often more insightful
- Detect anomalous activities by exact/approximate graph matching

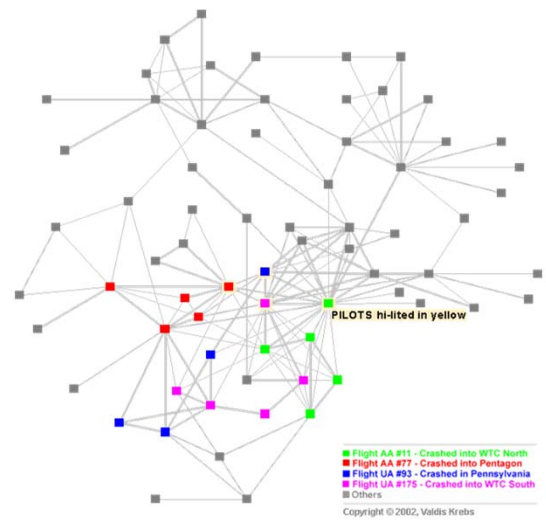


Image Source: http://www.orgnet.com/hijackers.html

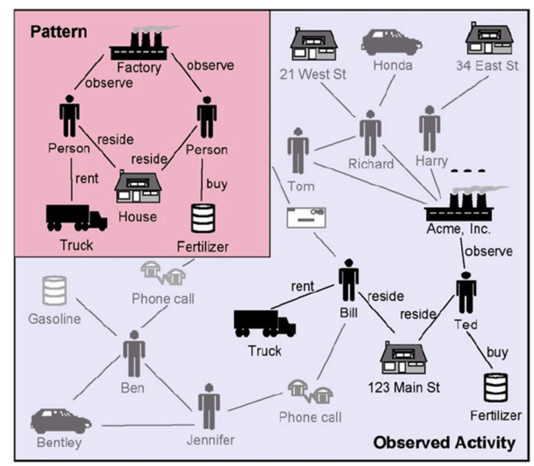


Image Source: T. Coffman, S. Greenblatt, S. Marcus, Graph-based technologies for intelligence analysis, CACM, 47 (3, March 2004): pp 45-47

Characterizing Graph-theoretic computations

Input: Graph abstraction

Problem: Find ***

- paths
- clusters
- partitions
- matchings
- patterns
- orderings

Graph algorithms

- traversal
- shortest path algorithms
- flow algorithms
- spanning tree algorithms
- topological sort

.....

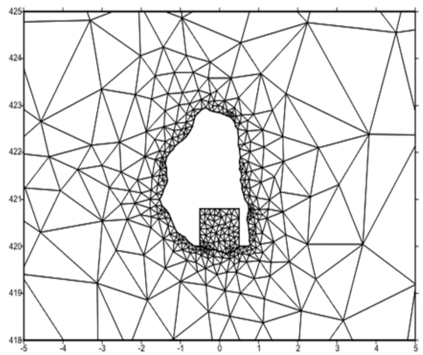
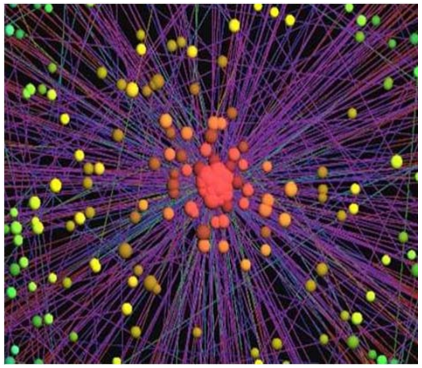
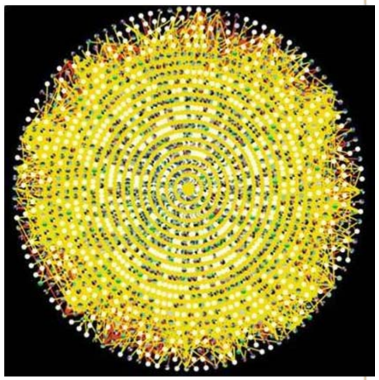
Factors that influence choice of algorithm

- graph sparsity (m/n ratio)
- static/dynamic nature
- weighted/unweighted, weight distribution
- vertex degree distribution
- directed/undirected
- simple/multi/hyper graph
- problem size
- granularity of computation at nodes/edges
- domain-specific characteristics

Graph problems are often recast as sparse linear algebra (e.g., partitioning) or linear programming (e.g., matching) computations

Massive data analytics in Informatics networks

- Graphs arising in Informatics are very different from topologies in scientific computing.

Static networks,
Euclidean topologies

Emerging applications: dynamic,
high-dimensional data

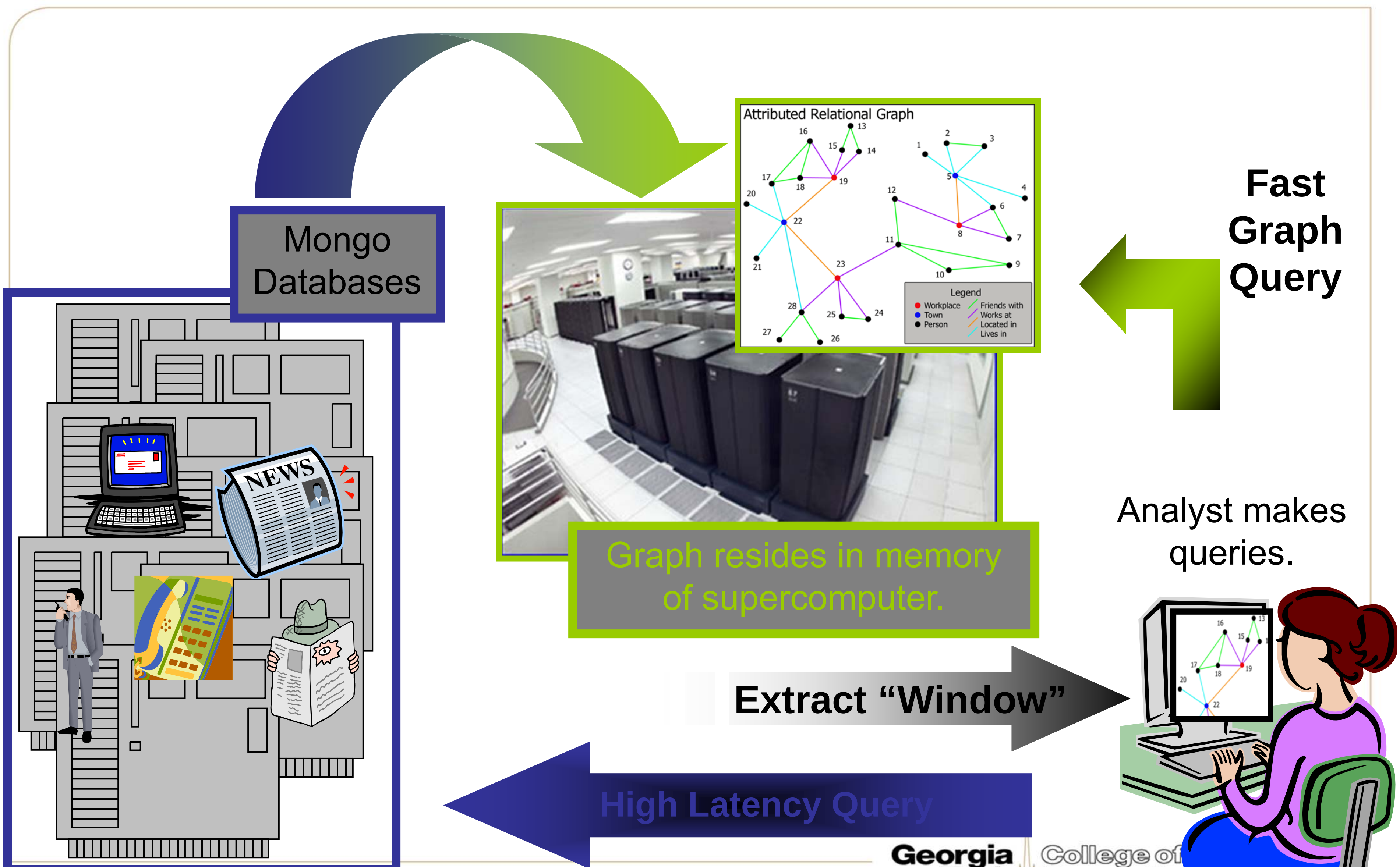
- We need new data representations and parallel algorithms that exploit topological characteristics of informatics networks.

What we'd like to infer from Information networks

- What are the degree distributions, clustering coefficients, diameters, etc.?
  - Heavy-tailed, small-world, expander, geometry+rewiring, local-global decompositions, ...
- How do networks grow, evolve, respond to perturbations, etc.?
  - Preferential attachment, copying, HOT, shrinking diameters, ..
- Are there natural clusters, communities, partitions, etc.?
  - Concept-based clusters, link-based clusters, density-based clusters, ...
- How do dynamic processes – search, diffusion, etc. – behave on networks?
  - Decentralized search, undirected diffusion, cascading epidemics, ...
- How best to do learning, e.g., classification, regression, ranking, etc.?
  - Information retrieval, machine learning, ...

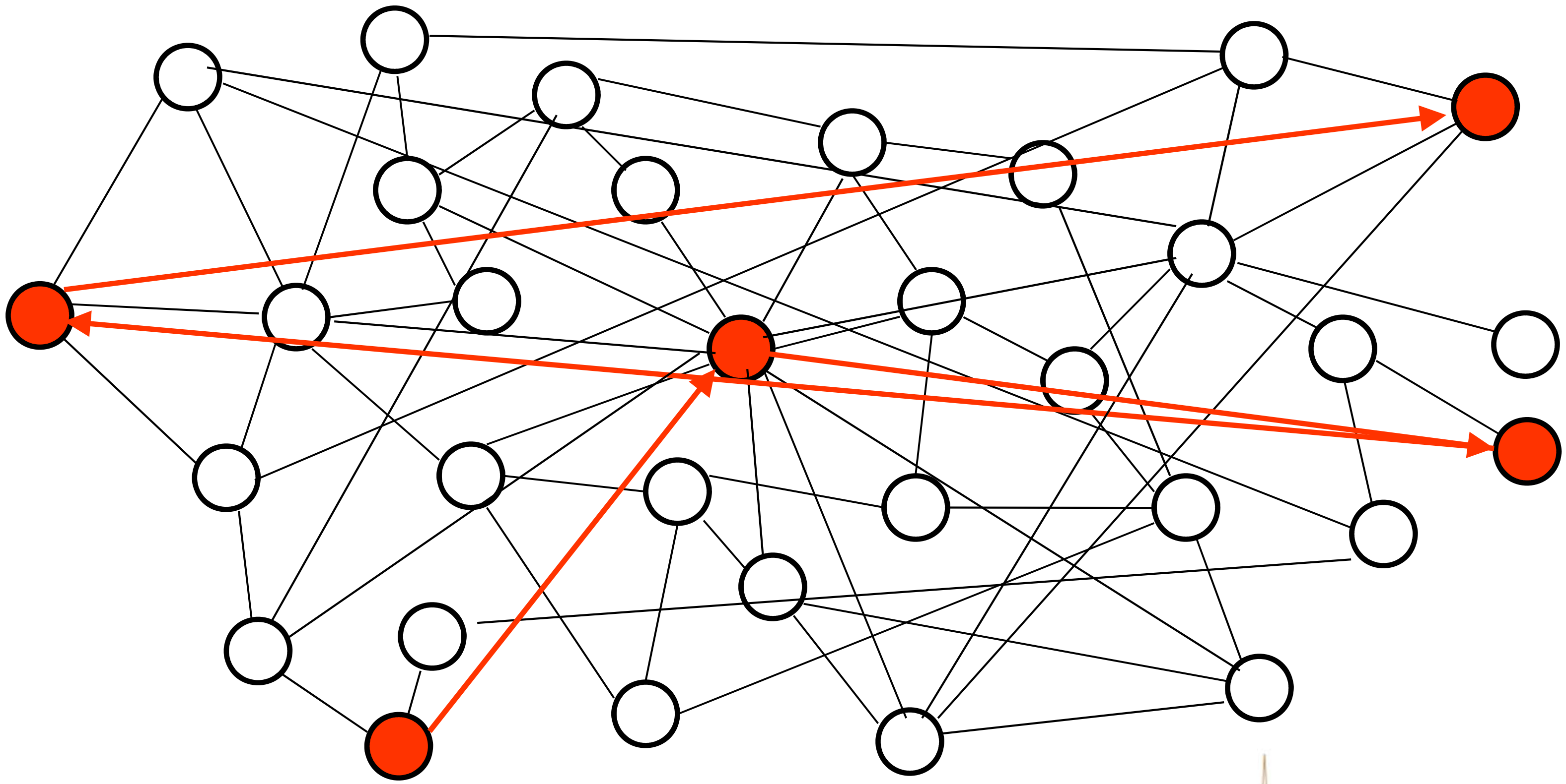
Slide credit: Michael Mahoney, Stanford

The Big Picture



Graph Analysis

- Irregularly traversing large (multi-billion vertices with up to a trillion edges) graph data.



ALGORITHMS:
SOCIAL NETWORK ANALYSIS (CENTRALITY)

Finding "central" entities is a key graph analytics routine

- Centrality: Quantitative measure to capture the importance of a vertex/edge in a graph.
  - Application-specific: can be based on degree, paths, flows, eigenvectors, …

Intelligence
Problem: Unraveling terrorist networks.

Bioinformatics
Problem: Identifying drug target proteins, metabolic pathways.

US power transmission grid
Problem: Contingency analysis

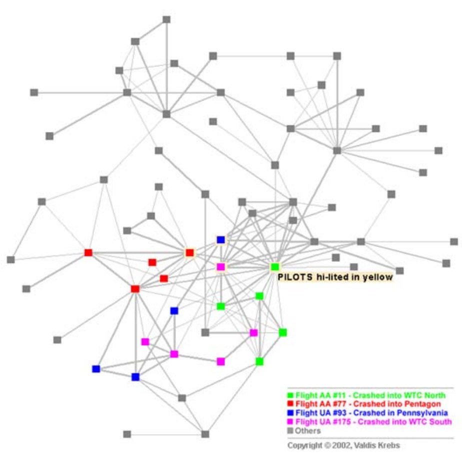


Image Source: http://www.orgnet.com/hijackers.html

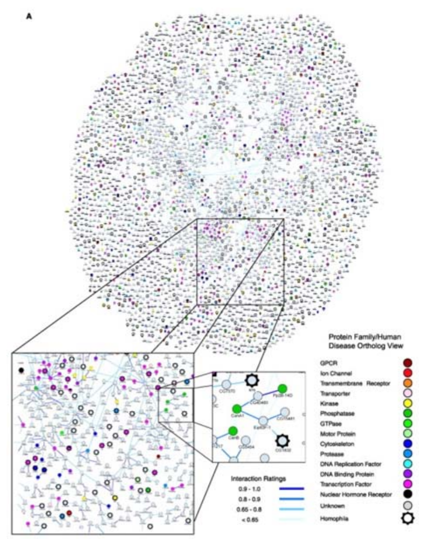


Image Source: Giot et al., "A Protein Interaction Map of *Drosophila melanogaster*", *Science 302*, 1722-1736, 2003.

Online Social networks
Problem: Discover emergent communities, identify influential people.

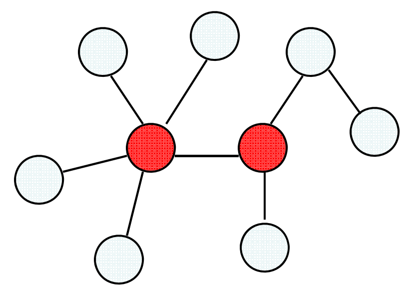
Centrality in Massive Social Network Analysis

- Centrality metrics: Quantitative measures to capture the importance of person in a social network
  - Betweenness is a global index related to shortest paths that traverse through the person
  - Can be used for community detection as well
- Identifying *central* nodes in large complex networks is the key metric in a number of applications:
  - Biological networks, protein-protein interactions
  - Sexual networks and AIDS
  - Identifying key actors in terrorist networks
  - Organizational behavior
  - Supply chain management
  - Transportation networks
- Current Social Network Analysis (SNA) packages handle 1,000's of entities, our techniques handle BILLIONS (6+ orders of magnitude larger data sets)

Betweenness Centrality (BC)

- Key metric in social network analysis
  [Freeman '77, Goh '02, Newman '03, Brandes '03]

$$\boxed{BC(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}}$$



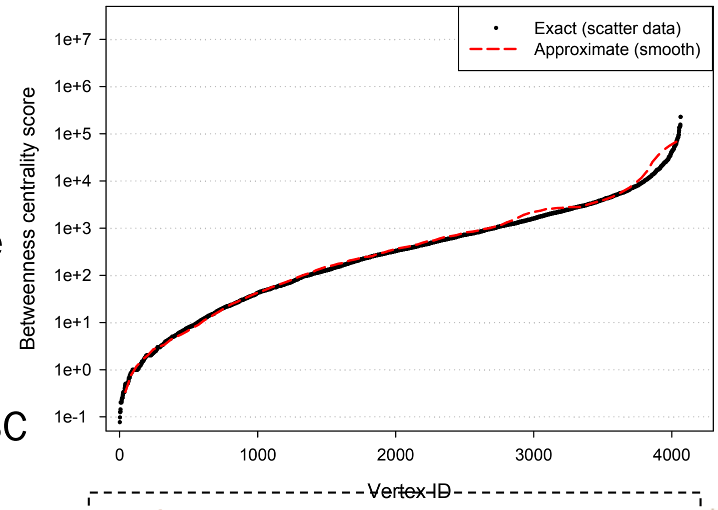
- σ_{st} : Number of shortest paths between vertices *s* and *t*
- $\sigma_{st}(v)$: Number of shortest paths between vertices *s* and *t* passing through *v*

- Exact BC is compute-intensive

BC Algorithms

- Brandes [2003] proposed a faster sequential algorithm for BC on sparse graphs
  – $O(mn + n^2 \log n)$ time and $O(n)$ space for weighted graphs
  – $O(mn)$ time for unweighted graphs
- We designed and implemented the first parallel algorithm:
  – [Bader, Madduri; ICPP 2006]

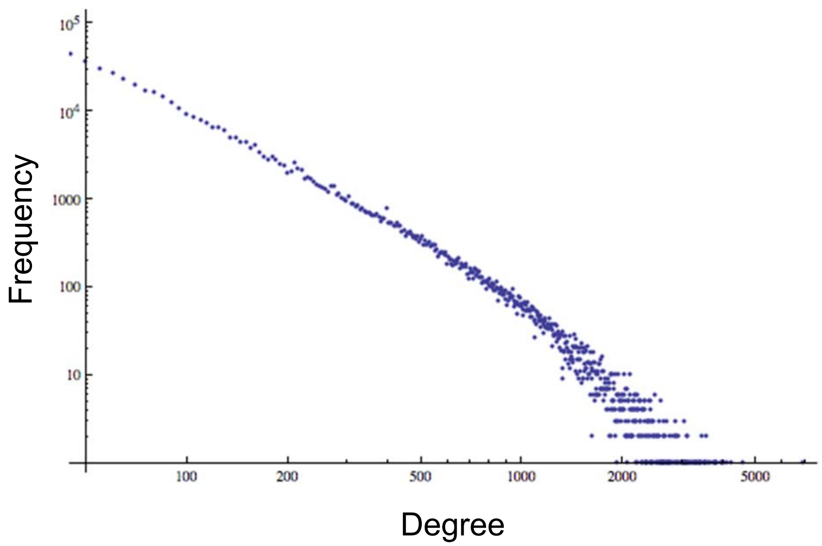
- Approximating Betweenness Centrality [Bader Kintali Madduri Mihail 2007]
  – Novel approximation algorithm for determining the betweenness of a *specific vertex or edge* in a graph
  – *Adaptive* in the number of samples
  – Empirical result: At least 20X speedup over exact BC

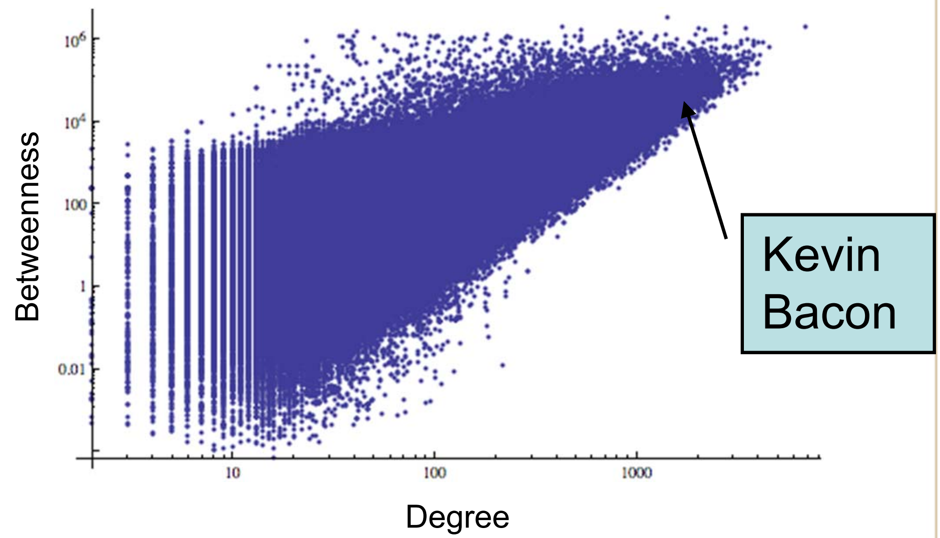


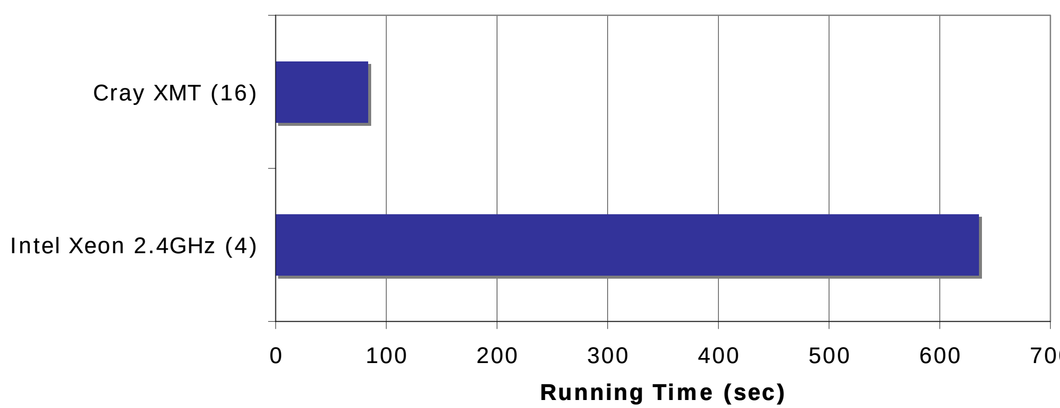
Graph: 4K vertices and 32K edges,
System: Sun Fire T2000 (Niagara 1)

IMDB Movie Actor Network (Approx BC)

An undirected graph of 1.54 million vertices (movie actors) and 78 million edges. An edge corresponds to a link between two actors, if they have acted together in a movie.







Fine-grained Parallel BC Algorithm

- Consider an undirected, unweighted graph
- High-level idea: Level-synchronous parallel Breadth-First Search augmented to compute centrality scores
- Exact BC computation
  - *n* source vertices (iterations)
  - Each iteration:
    - traversal and path counting
    - dependency accumulation

$$\delta(v) = \sum_{v \in P(w)} \frac{\sigma(v)}{\sigma(w)} \left(1 + \delta(w)\right)$$

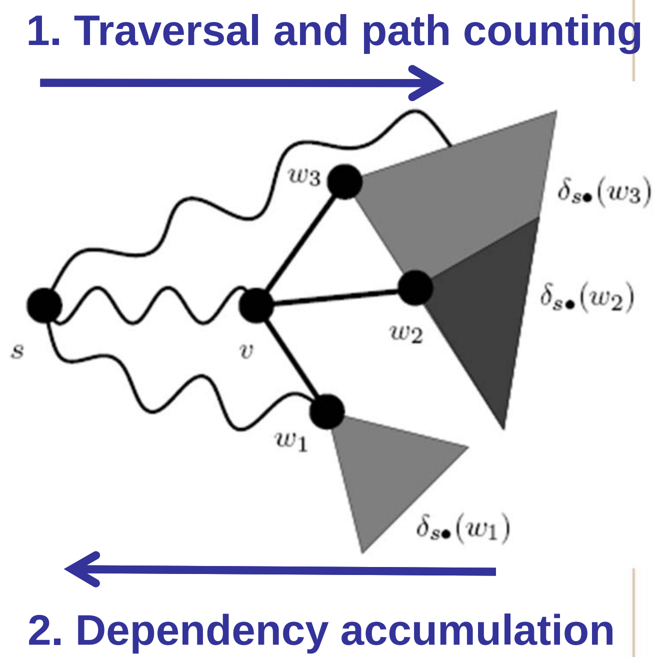
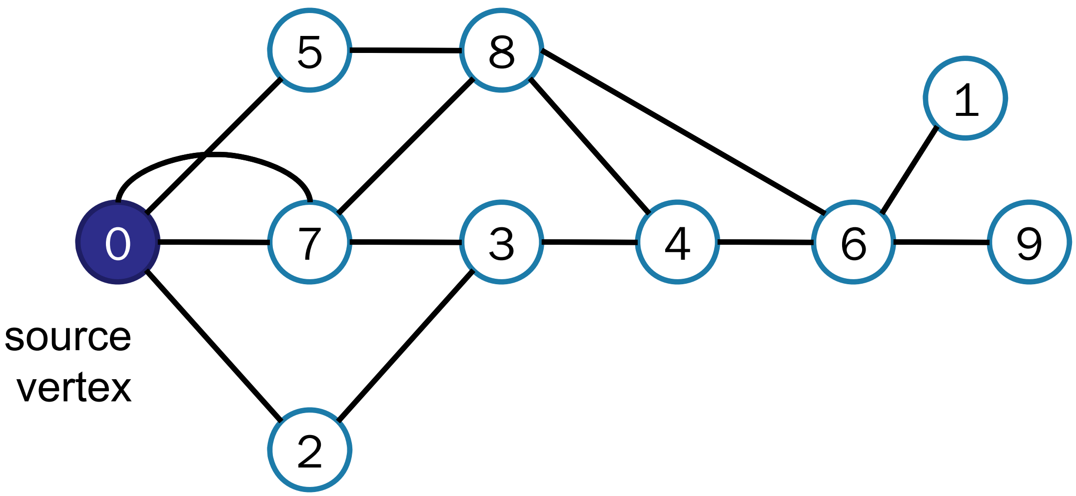


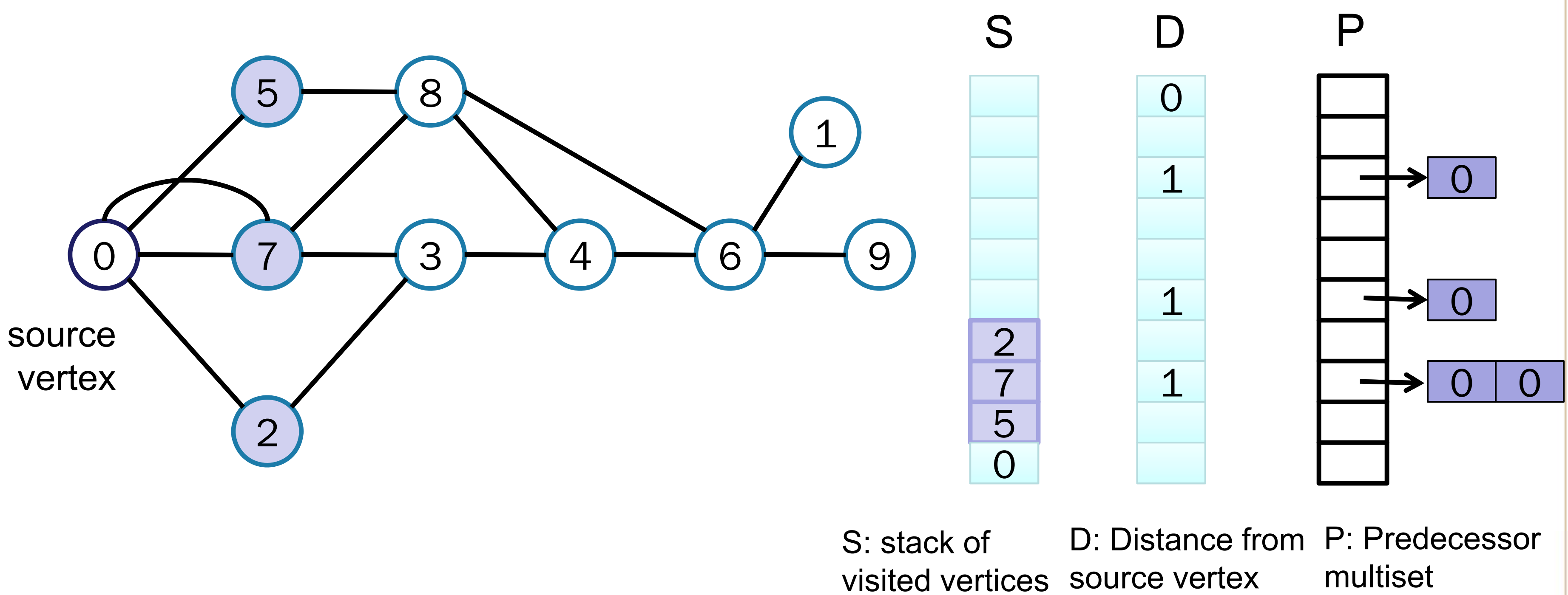
Illustration of Parallel BC (pBC-Old)

1. Traversal step: visit adjacent vertices, update distance and path counts.



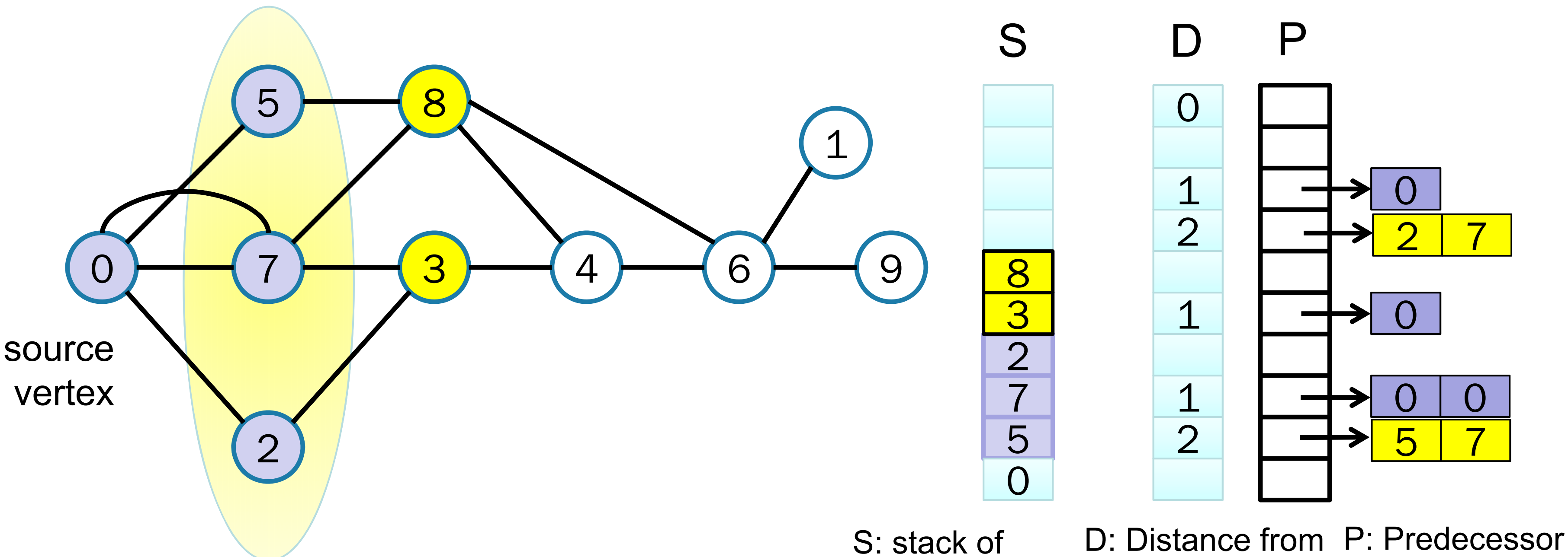
Step 1 (traversal) Illustration

1. Traversal step: visit adjacent vertices, update distance and path counts.



Step 1 Illustration

1. Traversal step: visit adjacent vertices, update distance and path counts.



S: stack of visited vertices

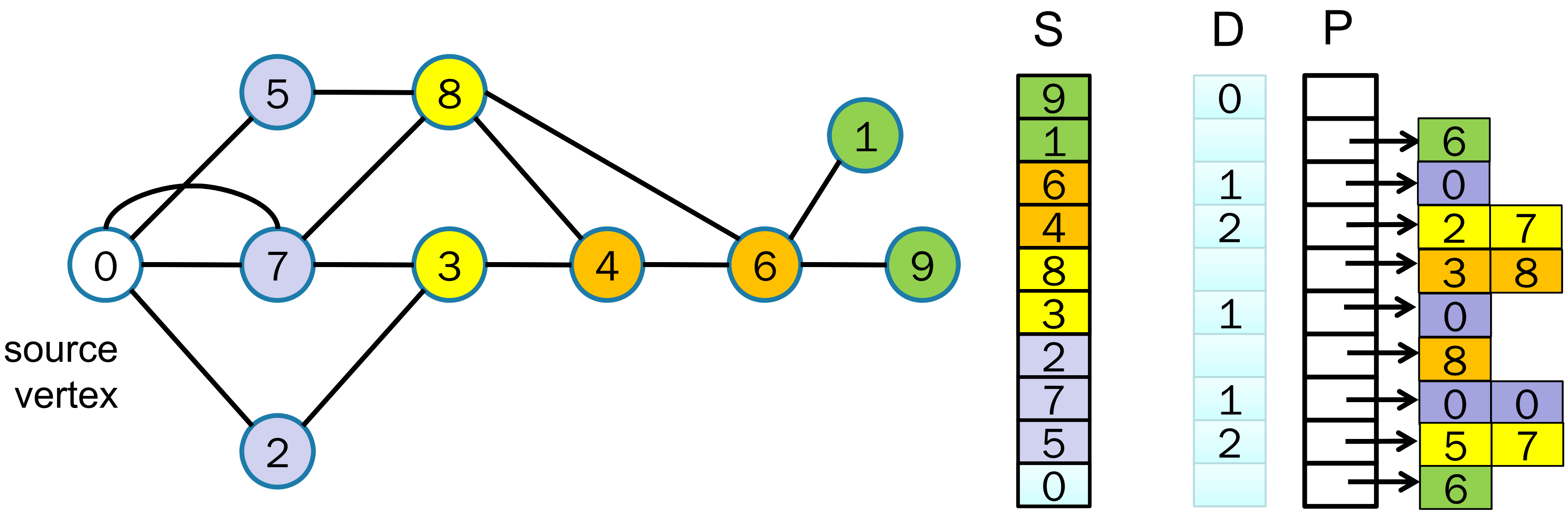
D: Distance from source vertex

P: Predecessor multiset

Level-synchronous approach: The adjacencies of all vertices in the current frontier can be visited in parallel

Step 1 Illustration

1. Traversal step: at the end, we have all reachable vertices, their corresponding predecessor multisets, and D values.



S: stack of visited vertices

D: Distance from source vertex

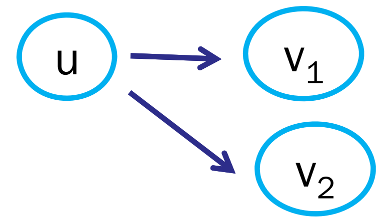
P: Predecessor multiset

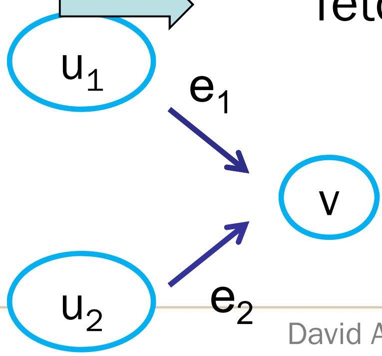
Level-synchronous approach: The adjacencies of all vertices in the current frontier can be visited in parallel

Step 1 pBC-Old pseudo-code

```
for all vertices u at level d in parallel do
    for all adjacencies v of u in parallel do
        dv = D[v];
        if (dv < 0) // v is visited for the first time
            vis = fetch_and_add(&Visited[v], 1);
            if (vis == 0)  // v is added to a stack only once
                D[v] = d+1;
                pS[count++] = v; // Add v to local thread stack
            fetch_and_add(&sigma[v], sigma[u]);
            fetch_and_add(&Pcount[v], 1); // Add u to predecessor list of v
        if (dv == d + 1)
            fetch_and_add(&sigma[v], sigma[u]);
            fetch_and_add(&Pcount[v], 1); // Add u to predecessor list of v
```

Atomic updates: performance bottlenecks!

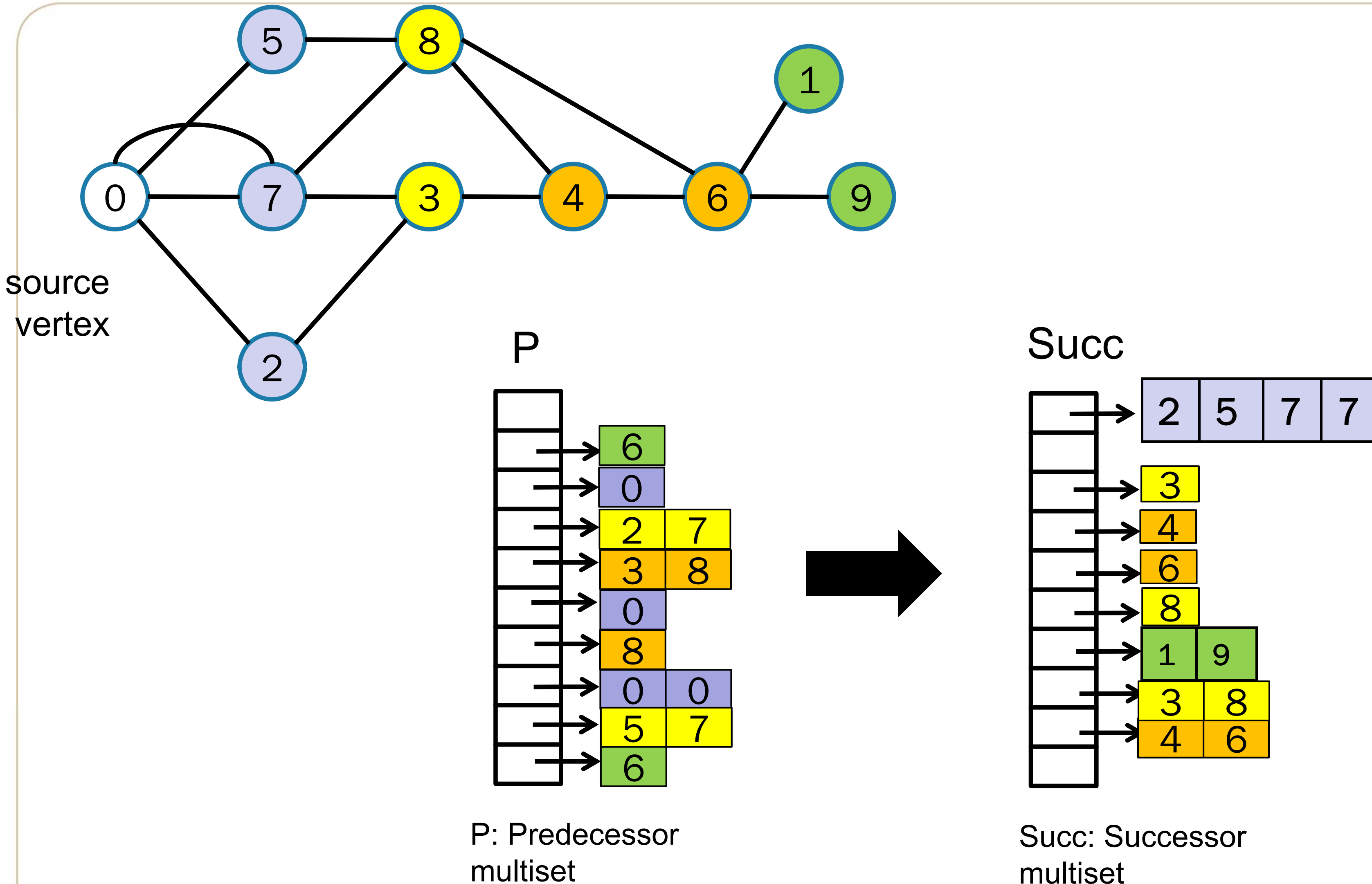




Step 1 analysis

- Exploit concurrency in exploration of current frontier and visiting adjacencies, as the graph diameter is low: O(log n) or O(1).
- Potential performance bottlenecks: atomic updates to predecessor multisets, atomic increments of path counts
- New contribution: Data structure change to eliminate storage of "predecessor" multisets. We store successor edges along shortest paths instead.
  - simplifies the accumulation step
  - Eliminates two atomic operations in traversal step
  - cache-friendly!

pBC-LockFree change in data representation



Step 1 pBC-LockFree Locality Analysis

```
for all vertices u at level d in parallel do
    for all adjacencies v of u do
        dv = D[v];
        if (dv < 0)
            vis = fetch_and_add(&Visited[v], 1);
            if (vis == 0)
                D[v] = d+1;
                pS[count++] = v;
            fetch_and_add(&sigma[v], sigma[u]);
            Scount[u]++;
        if (dv == d + 1)
            fetch_and_add(&sigma[v], sigma[u]);
            Scount[u]++;
```

All the vertices are in a contiguous block (stack)

All the adjacencies of a vertex are stored compactly (graph rep.)

Non-contiguous memory access

Non-contiguous memory access

Non-contiguous memory access

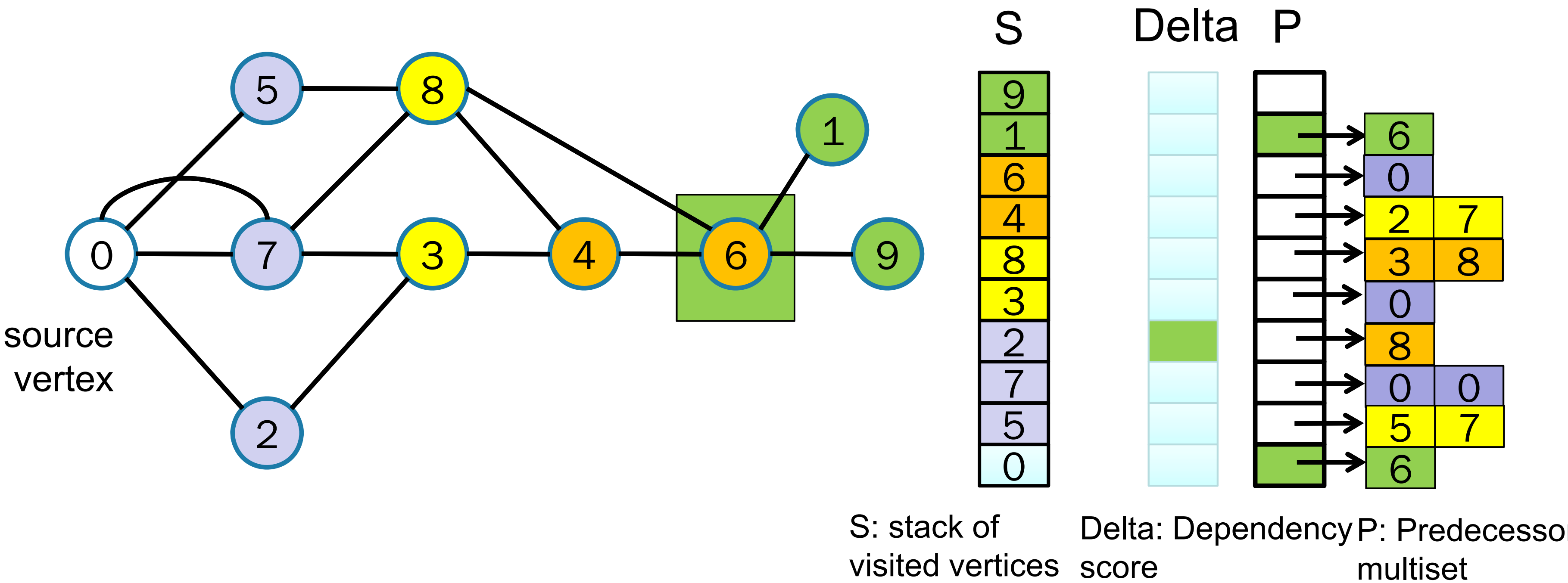
Indicates store to S[u]

Store D[v], Visited[v], sigma[v] contiguously for better cache locality.

Step 2 Dependence Accumulation Illustration

2. Accumulation step: Pop vertices from stack, update dependence scores.

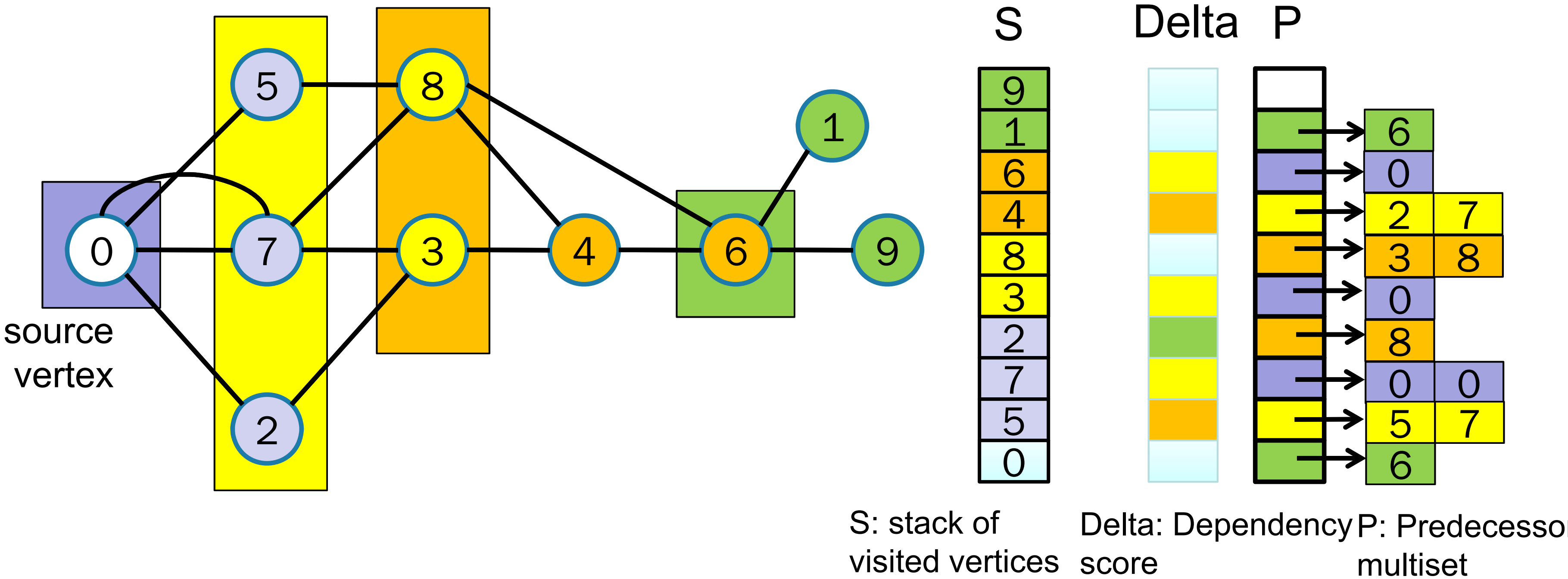
$$\delta(v) = \sum_{v \in P(w)} \frac{\sigma(v)}{\sigma(w)}\left(1+\delta(w)\right)$$



Step 2 Dependence Accumulation Illustration

2. Accumulation step: Can also be done in a level-synchronous manner.

$$\delta(v) = \sum_{v \in P(w)} \frac{\sigma(v)}{\sigma(w)}\left(1 + \delta(w)\right)$$



Step 2 pBC-Old pseudo-code

Locks: performance bottleneck!

```
for level d = GraphDiameter to 2 do
    for all vertices w at level d in parallel do
        for all v in P[w] do
            acquire_lock(v);
            delta[v] = delta[v] + (1 + delta[w]) * sigma(v)/sigma(w);
            release_lock(v);
        BC[v] = delta[v]
```

$$\boxed{\delta(v) = \sum_{v \in P(w)} \frac{\sigma(v)}{\sigma(w)}\left(1 + \delta(w)\right)}$$

Step 2 pBC-LockFree

for level d = GraphDiameter-2 to 1 do

for all vertices v at level d in parallel do

for all w in S[v] in parallel do reduction(delta)

```
delta_sum_v = delta[v] + (1 + delta[w]) * sigma[v]/sigma[w];

BC[v] = delta[v] = delta_sum_v;
```

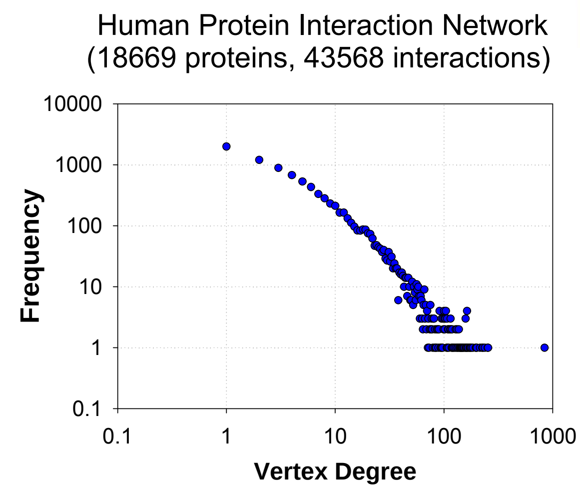
No atomic operation, reduction instead

Only floating point operations in code

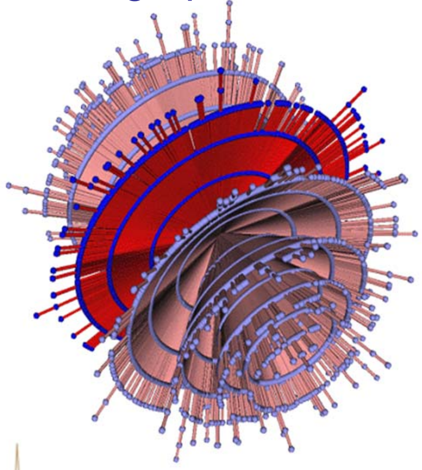
New parallel BC algorithm works well for massive “small-world” networks

- Low graph diameter.
  - Key source of concurrency in graph traversal.
- Skewed (“power law”) degree distribution of the number of neighbors.
  - Inner loop easier to parallelize after elimination of successor multisets. Preprocess for balanced partitioning of work among processors/threads.
  - High-degree vertices can be processed in parallel, separately.
- Dynamic network abstractions, from diverse data sources; massive networks (billions of entities).
  - Data representations and structures are space-efficient, support edge attributes, and fast parallel insertions and deletions.

Skewed degree distribution



Low graph diameter



Performance Results: Experimental Setup

Cray XMT

- Latency tolerance 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
- 16 x 500 MHz processors, 128 GB RAM

DARPA HPCS SSCA#2 Graph Analysis benchmark

- Representative of graph-theoretic computations in real-world networks.
  http://www.graphanalysis.org
- Approximate betweenness centrality is a key kernel.
- Synthetic R-MAT networks generated based on Kronecker products.
- Performance measure: Traversed edges per second (TEPS) rate.

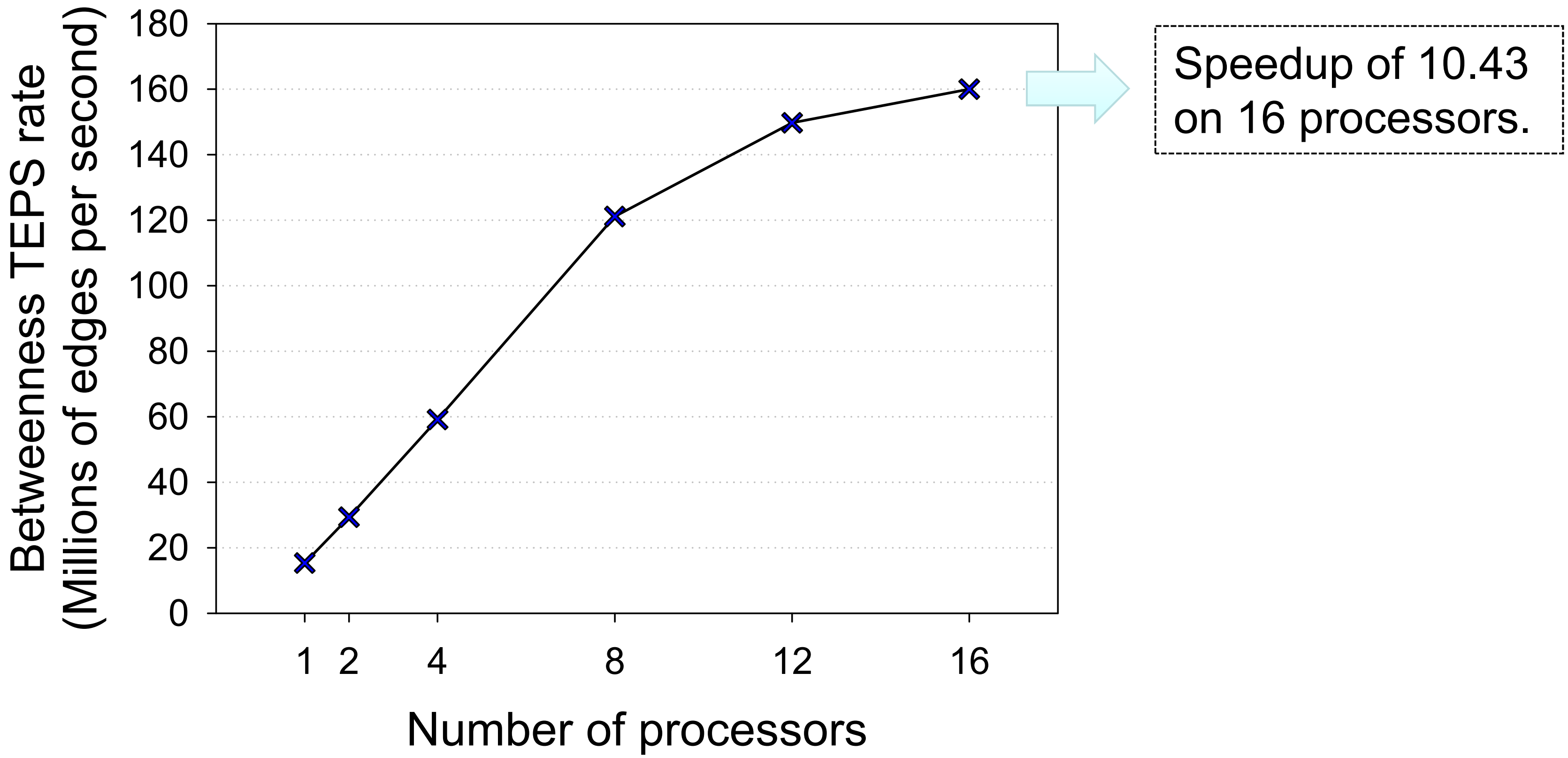
$$\mathrm{BC\ TEPS\ rate} = \frac{7n \cdot 2^{K4Approx}}{t} \ \mathrm{edges/second}$$

IMDb actors network

- Real-world social network constructed from IMDb data.
- Undirected network: 1.54 million vertices (actors) and 78 million edges (edge → two actors co-starring in a movie).

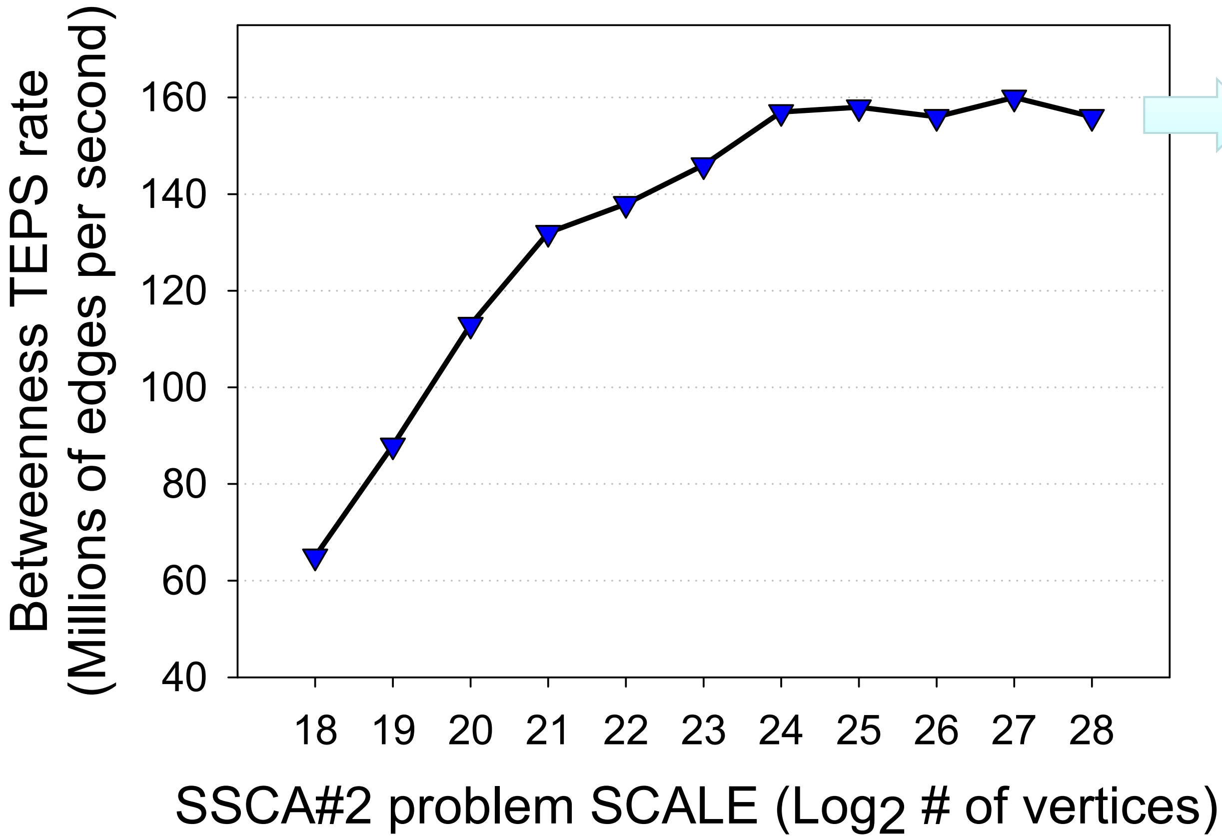
Cray XMT Parallel Performance

- Synthetic network with 16.77 million vertices and 134.21 million edges (SCALE 24), K4Approx = 8.



Cray XMT Performance vs. Problem size

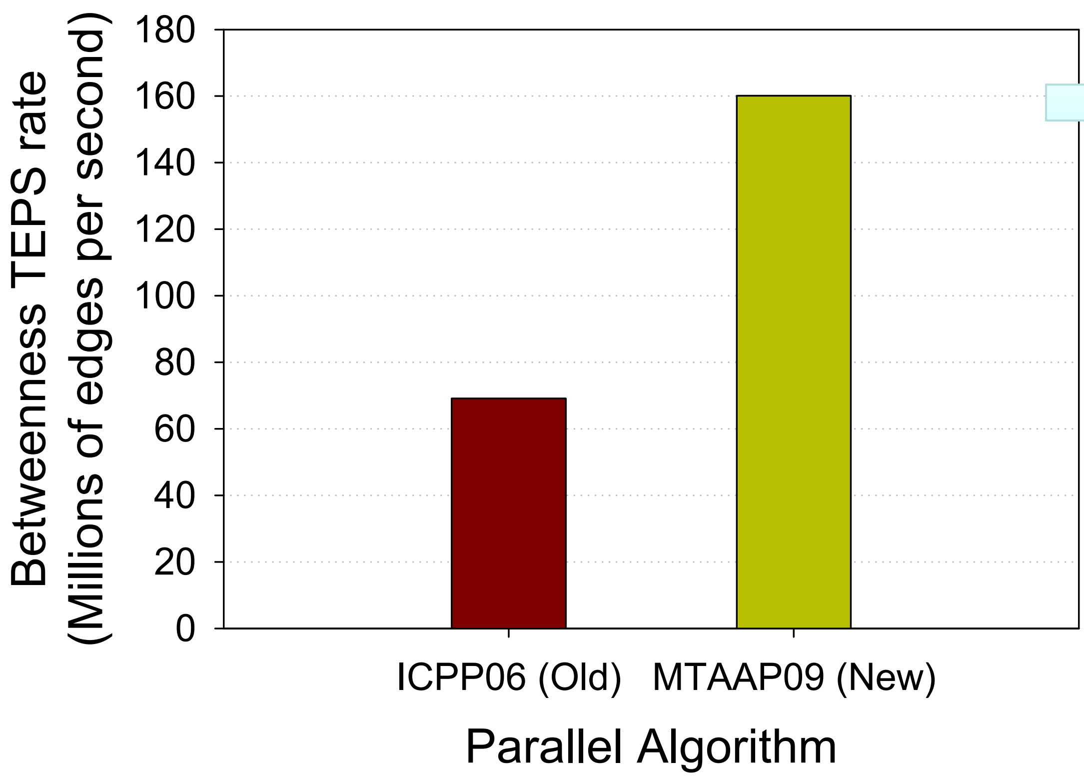
- SSCA#2 networks, $n = 2^{SCALE}$ and $m = 8n$.



Sufficient concurrency on 16 processors for problem instances with SCALE > 24.

Performance compared to previous algorithm

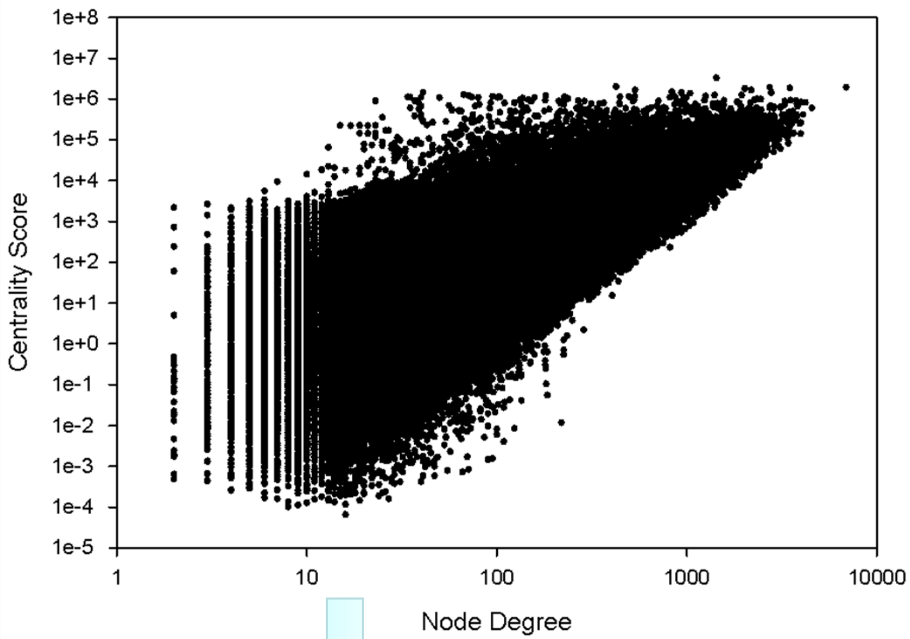
- SSCA#2 network, SCALE 24 (16.77 million vertices and 134.21 million edges.)

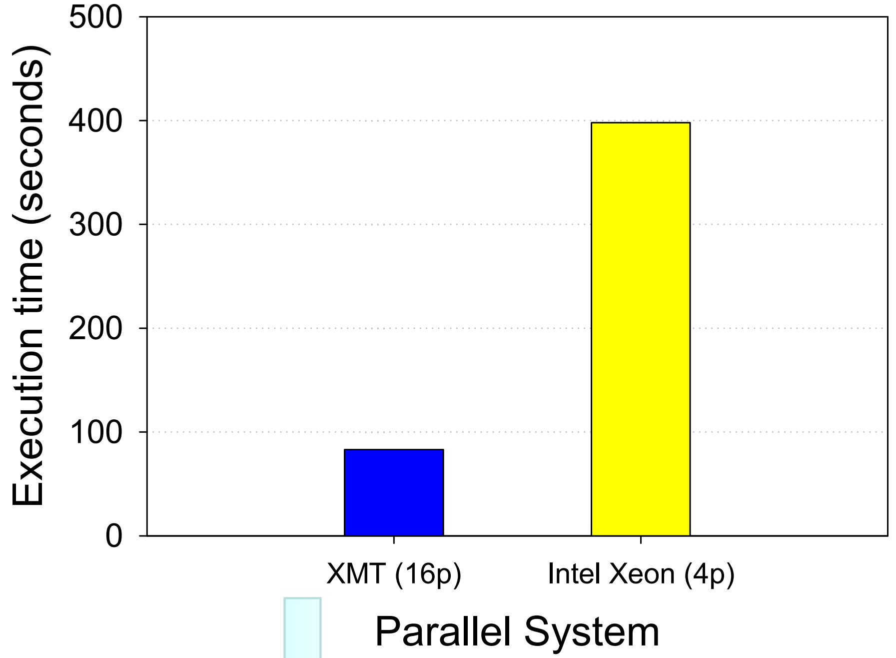


Speedup of 2.3 over previous approach.

Appoximate BC on the IMDb network

- Undirected network of 1.54 million vertices and 78 million edges, 256 randomly selected source vertices.



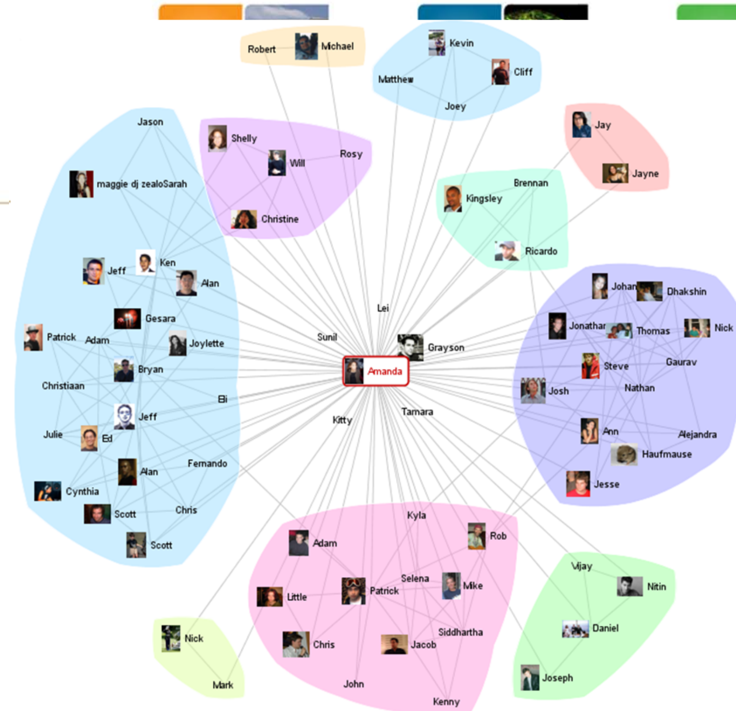


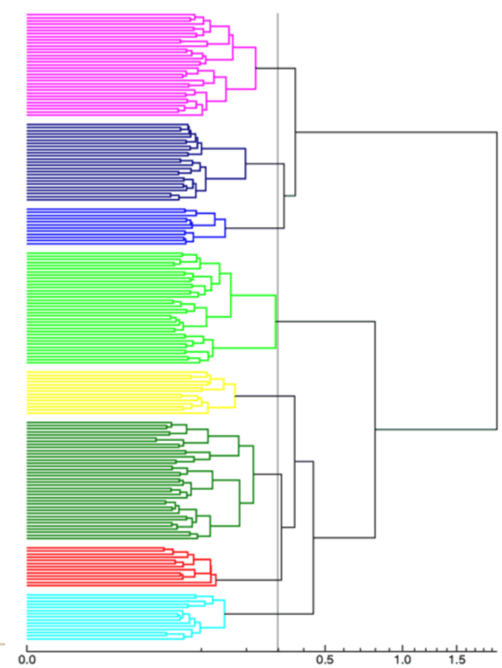
Low-degree vertices exhibit high variance in centrality scores.

16-proc Cray XMT is 4.75x faster than a 2.8 GHz quad-core Intel Xeon system.

Community Identification

- Implicit communities in large-scale networks are of interest in many cases.
  - WWW
  - Social networks
  - Biological networks
- Formulated as a graph clustering problem.
  - Informally, identify/extract "dense" sub-graphs.
- Several different objective functions exist.
  - Metrics based on intra-cluster vs. inter-cluster edges, community sizes, number of communities, overlap …
- Highly studied research problem
  - 100s of papers yearly in CS, Social Sciences, Physics, Comp. Biology, Applied Math journals and conferences.





Related Work: Partitioning Algorithms from Scientific Computing

- Theoretical and empirical evidence: existing techniques perform poorly on small-world networks

- [Mihail, Papadimitriou ’02] Spectral properties of power-law graphs are skewed in favor of high-degree vertices
- [Lang ’04] On using spectral techniques, “Cut quality varies inversely with cut balance” in social graphs: Yahoo! IM graph, DBLP collaborations
- [Abou-Rjeili, Karypis ’06] Multilevel partitioning heuristics give large edge-cut for small-world networks, new coarsening schemes necessary

Modularity: A popular optimization metric

- Measure based on *optimizing inter-cluster density over intra-cluster sparsity.*
- For a weighted, directed network with vertices partitioned into non-overlapping clusters, modularity is defined as

$$Q = \frac{1}{2w} \sum_{i \in V} \sum_{j \in V} \left(w_{ij} - \frac{w_i^{out} w_j^{in}}{2w} \right) \delta(C_i, C_j)$$

$$w_i^{out} = \sum_j w_{ij}, \; w_j^{in} = \sum_i w_{ij}, \; 2w = \sum_i \sum_j w_{ij}$$

$$\delta(C_i, C_j) = 1 \text{ if } C_i = C_j, \quad 0 \text{ otherwise.}$$

- If a particular clustering has no more intra-cluster edges than would be expected by random chance, Q=0. Values greater than 0.3 typically indicate community structure.
- Maximizing modularity is NP-complete.

Modularity

For an unweighted and undirected network, modularity is given by

$$Q = \frac{1}{2m} \sum_{i \in V} \sum_{j \in V} \left(e_{ij} - \frac{d_i d_j}{2m} \right) \delta(C_i, C_j)$$

$$e_{ij} = 1 \text{ if } \langle i, j \rangle \in E$$

$$\delta(C_i, C_j) = 1 \text{ if } C_i = C_j, \quad 0 \text{ otherwise.}$$

and in terms of clusters/modules, it is equivalently

$$Q = \sum_s \left(\frac{m_s}{m} - \left(\frac{\sum_{C_v = s} d_v}{2m} \right)^2 \right)$$

Resolution limit: Modules will not be found, optimizing modularity, if

$$m_s < \sqrt{m/2} - 1$$

Our Contributions

- New parallel algorithms for modularity-optimizing community identification.
  - Divisive: edge betweenness-based, spectral
  - Agglomerative
  - Hybrid, multi-level
- Several algorithmic optimizations for small-world networks.
- Analysis of large-scale complex networks constructed from real data.
- Note: No single “right” community detection algorithm exists. Community structure analysis should be user-driven and application-specific, combining various fast algorithms.

Divisive Clustering, Parallelization

- Top-down approach: Start with entire network as one community, recursively split the graph to yield smaller modules.
- Two popular methods:
  - Edge-betweenness based: iteratively remove high-centrality edges.

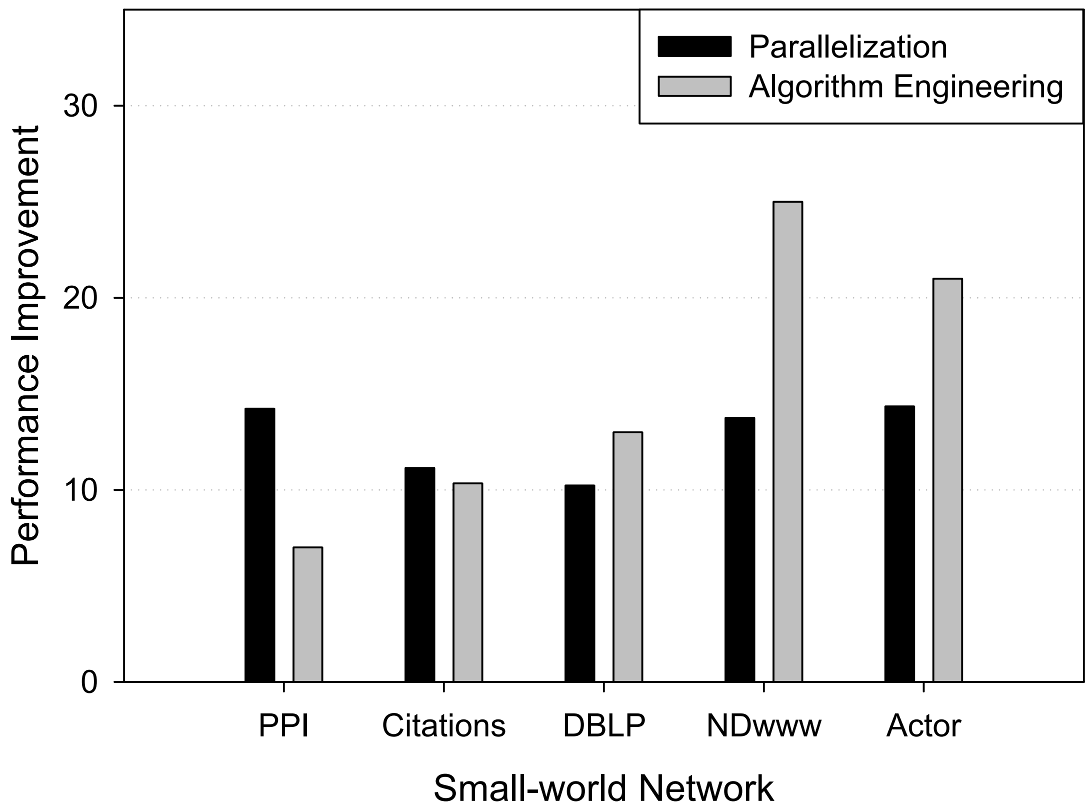
$$\boxed{BC(e) = \sum_{s,t \in V} \frac{\sigma_{st}(e)}{\sigma_{st}}}$$

    - Centrality computation is the compute-intensive step, parallelize it.
  - Spectral: apply recursive spectral bisection on the "modularity matrix" B, whose elements are defined as $B_{ij} = A_{ij} - d_i d_j / 2m$. Modularity can be expressed in terms of B as:

$$\boxed{Q = \frac{1}{4m} s^T B s}$$

    - Parallelize the eigenvalue computation step (dominated by sparse matrix-vector products).

Faster Community Identification Algorithms: Performance Improvement over the Girvan-Newman (ref) approach

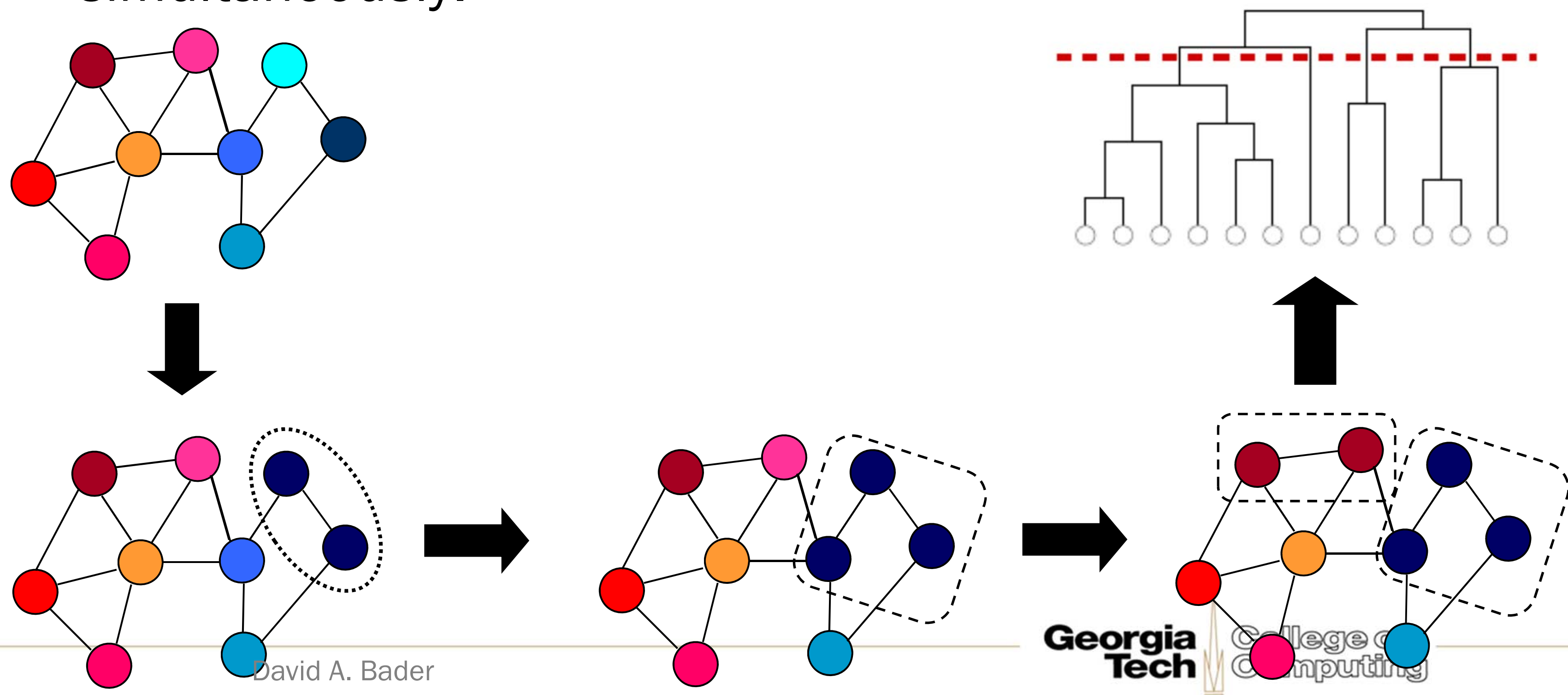


- Speedup from Algorithm Engineering (approximate BC) and parallelization (Sun Fire T2000) are **multiplicative**!
- **100-300X** overall performance improvement over Girvan-Newman approach

Graphs: Real-world networks (order of Millions), System: Sun Fire T2000

Agglomerative Clustering, Parallelization

- Bottom-up approach: Start with *n* singleton communities, iteratively merge pairs to form larger communities.
  - What measure to minimize/maximize? modularity
  - How do we order merges? priority queue
- Parallelization: perform multiple “independent” merges simultaneously.

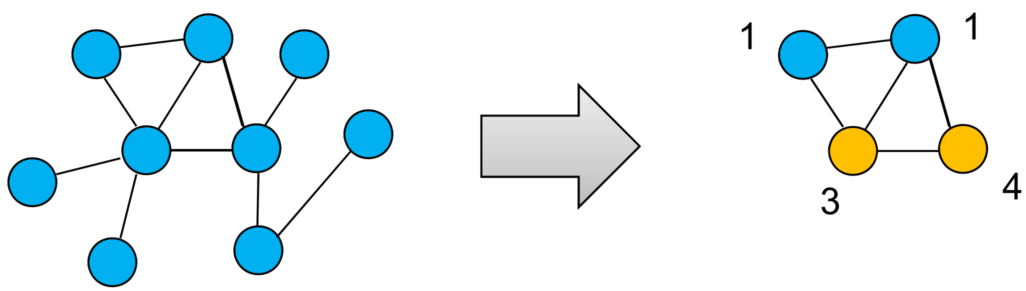


Other Community Identification Approaches

- Simulated annealing
- Extremal optimization
- Linear programming
- Statistical inference
- Spin models, random walks
- Clique percolation
- …

Engineering a hybrid parallel community identification algorithm

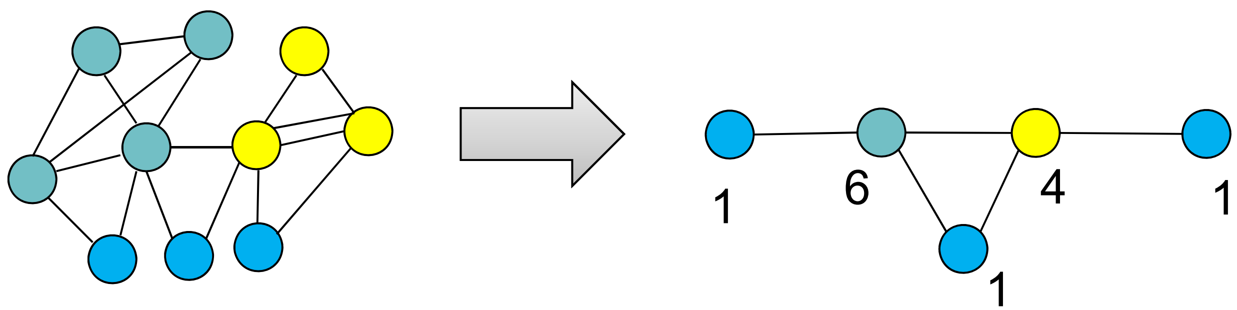
- How would a memory-efficient, near linear-work greedy approach perform on real data?
- Helpful preprocessing steps
  - 2-Core reduction of the graph
    - High-percentage of degree-1 vertices in networks with exponential and power-law degree distributions.



  - Filter very high-degree vertices ($d > d_H \approx \sqrt{n}$)
    - Ambiguity on what cluster they belong to.

Hybrid approaches: Parallelization

- Coarsen/sparsify graph
  - Local search at vertices to identify dense components, completely relax priority queue constraint => abundant parallelism.



  - Future work: Identify network-specific motifs (bipartite cliques).
- Run greedy agglomerative approach once graph is less than size threshold.

ALGORITHMS: K-BETWEENNESS CENTRALITY

k-Betweenness Centrality, BC_k

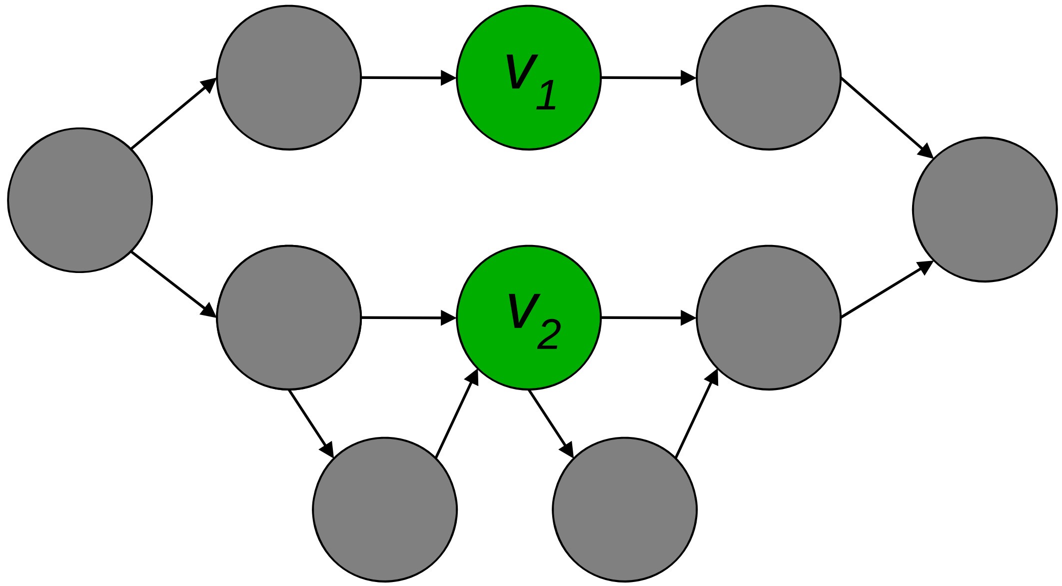
- A new twist on betweenness centrality:
  - Count **short** paths in addition to **shortest** paths
  - Captures wider connectivity information
- Applying BC_k to a real data set:
  - How do the BC indices behave with increasing *k*?
  - Which vertices have zero scores?
    - (Directed and undirected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

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*.

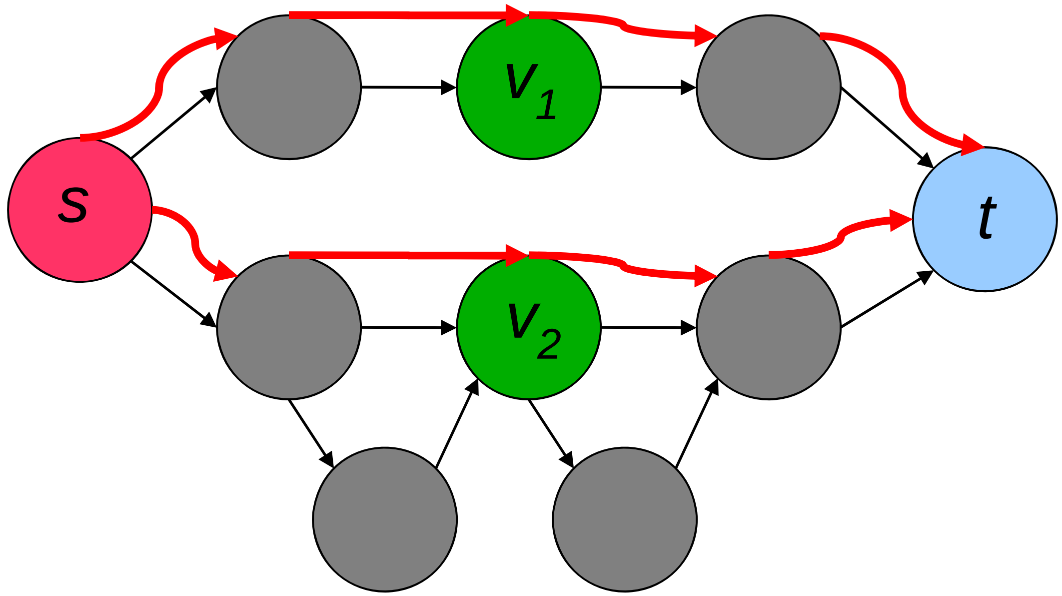
$$BC_k(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st_k}(v)}{\sigma_{st_k}}$$

Betweenness Centrality



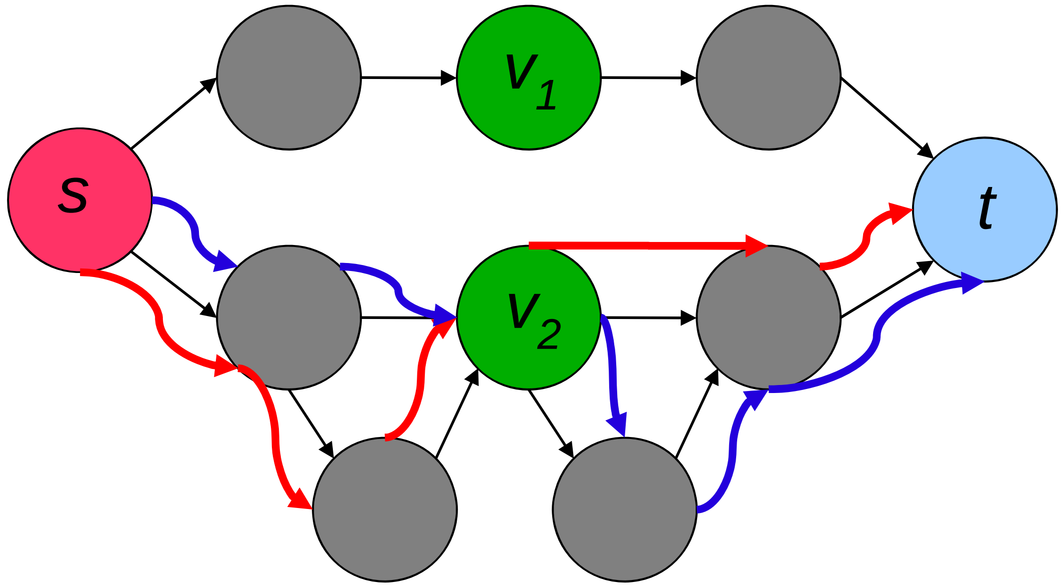
- How important are v_1 and v_2? Use betweenness centrality.
- The betweenness centrality of v_1, $BC(v_1)$:
  - Consider **shortest** paths between any two vertices $s, t \neq v_1$.
  - Sum over all such s, t: fraction of paths passing through v_1

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...

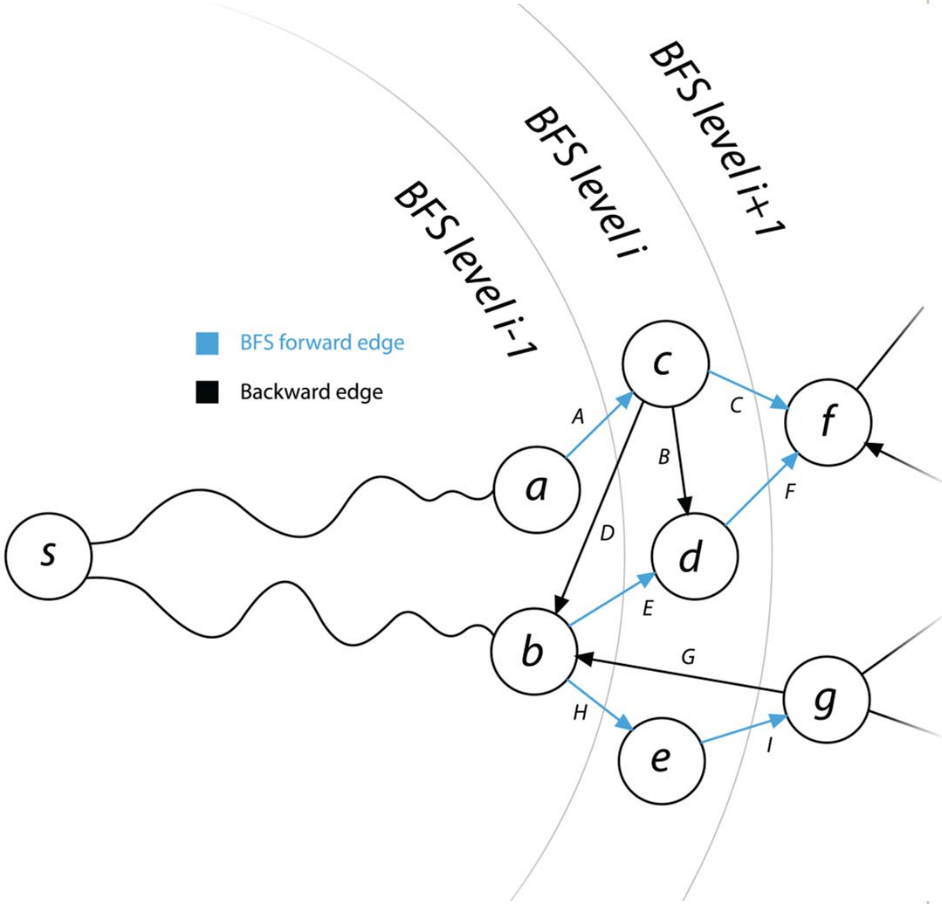
k-Betweenness Centrality: Shortest + k



- 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, $BC_0(v) = BC(v)$.

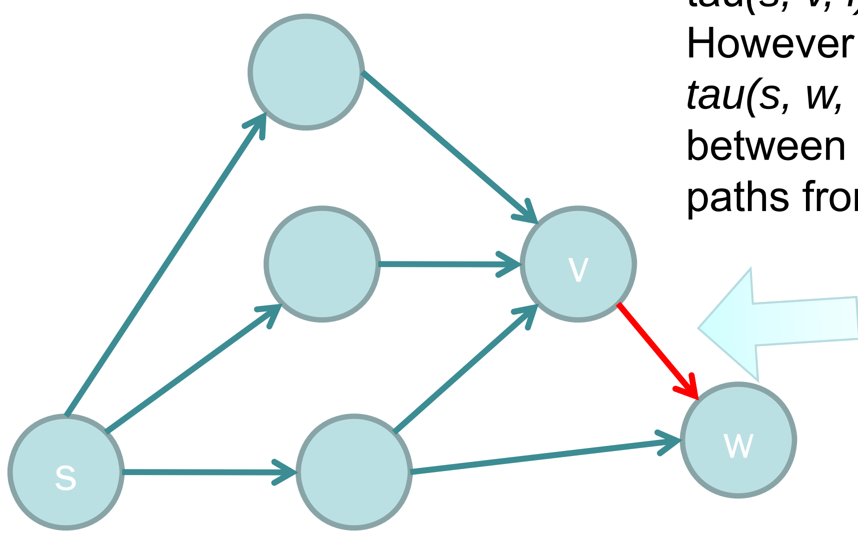
k-Betweenness: terms

- From any source vertex s, we can calculate the distance to any vertex $v = d(s, v)$
- Any *forward* edge $v \rightarrow w$ has $d(s,w) - d(s,v) = 1$
- Otherwise it is a *backward* edge. Any path originating at s with a backward edge cannot be a shortest path
- The degree of *inefficiency* of a backward edge $v \rightarrow w$ is given by $x(v, w) = d(s,v) - d(s,w) + 1$
- Any path from s to t of length $d(s, t) + k$ has edges with inefficiencies summing to k



k-Betweenness algorithm (1/4)

- Let *tau*(*s, t, i*) be the number of paths from *s* to *t* of distance *d*(*s, t*) + *i*
- Calculate *tau* (*s, t, i*) for all *t* in graph *G*, and $0 \le i \le k$. This can be done in *k+1* graph traversals



When an edge *v-w* is forward, any path counted in tau*(s, v, i)* gets propagated as a path in *tau(s, w, i)*. However if *x(v, w)* > 0, *tau(s, v, i)* gets propagated to *tau(s, w, i+x(v, w))*. Here, since 3 shortest paths exist between *s* and *v*, the red link contributes adds 3 *k*=1 paths from *s* to *w.*

tau(s, v, 0) = 3
x(v, w) = 1
contributes +3 to *tau(s, w, 1)*

k-Betweenness algorithm (2/4)

- Dependency calculation is complicated for *k*-Betweenness, requires $O(k^2)$ accumulations
- Consider a path from s to t with total inefficiency $x(s, t)=k$. Consider a vertex v on that path. Then $x(s, v) + x(v, t)=k$
- Define the *i,k-dependency* of s on v with *head-distribution* h = $delta(s, i, k, v, h)$ = the sum, over vertices t, of ratios y/z where z is the number of paths from s to t of length $\leq d(s, t) + k$ and y is the number of paths that:
  - go from s to t, through v
  - are of length $d(s, t) + i$
  - have $x(s, v) = h$ and $x(v, t) = i - h$

k-Betweenness algorithm (3/4)

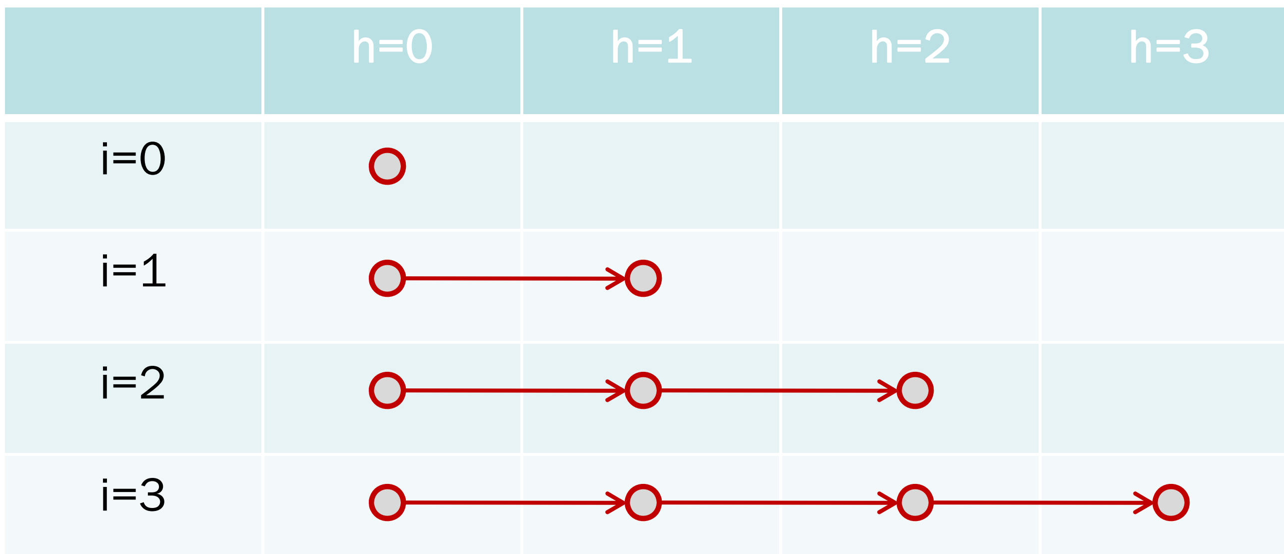
- Need to calculate *delta(s, i, k, v, h)* for all $v \neq s$ in graph G, $0 \leq i \leq k$, and $0 \leq h \leq i$
- A neighbor w of v is an i neighbor if $x(v, w) = i$. We can set up a recurrence by separating neighbors in this way
- (For N_2 an extra term appears to account for t=w)

$$\delta_{s_{i=2,h=0,k}}(v) = \sum_{t\neq s\neq w} \Big(\sum_{w\in N_0} \frac{\tau_{sv}}{\tau_{sw}} \cdot \frac{\tau_{st_{(0,2)}}(w)}{\sigma_{st_k}} + \sum_{w\in N_1} \frac{\tau_{sv}}{\tau_{sw_1}} \cdot \frac{\tau_{st_{(1,1)}}(w)}{\sigma_{st_k}} + \sum_{w\in N_2} \frac{\tau_{sv}}{\tau_{sw_2}} \cdot \frac{\tau_{st_{(2,0)}}(w)}{\sigma_{st_k}} \Big) + \sum_{w\in N_2} \frac{\tau_{sv}}{\sigma_{sw_k}}$$

$$= \sum_{w\in N_0} \frac{\tau_{sv}}{\tau_{sw}} \cdot \delta_{s_{2,0,k}}(w) + \sum_{w\in N_1} \frac{\tau_{sv}}{\tau_{sw_1}} \cdot \delta_{s_{2,1,k}}(w) + \sum_{w\in N_2} \Big(\frac{\tau_{sv}}{\tau_{sw_2}} \cdot \delta_{s_{2,2,k}}(w) + \frac{\tau_{sv}}{\sigma_{sw_k}} \Big)$$

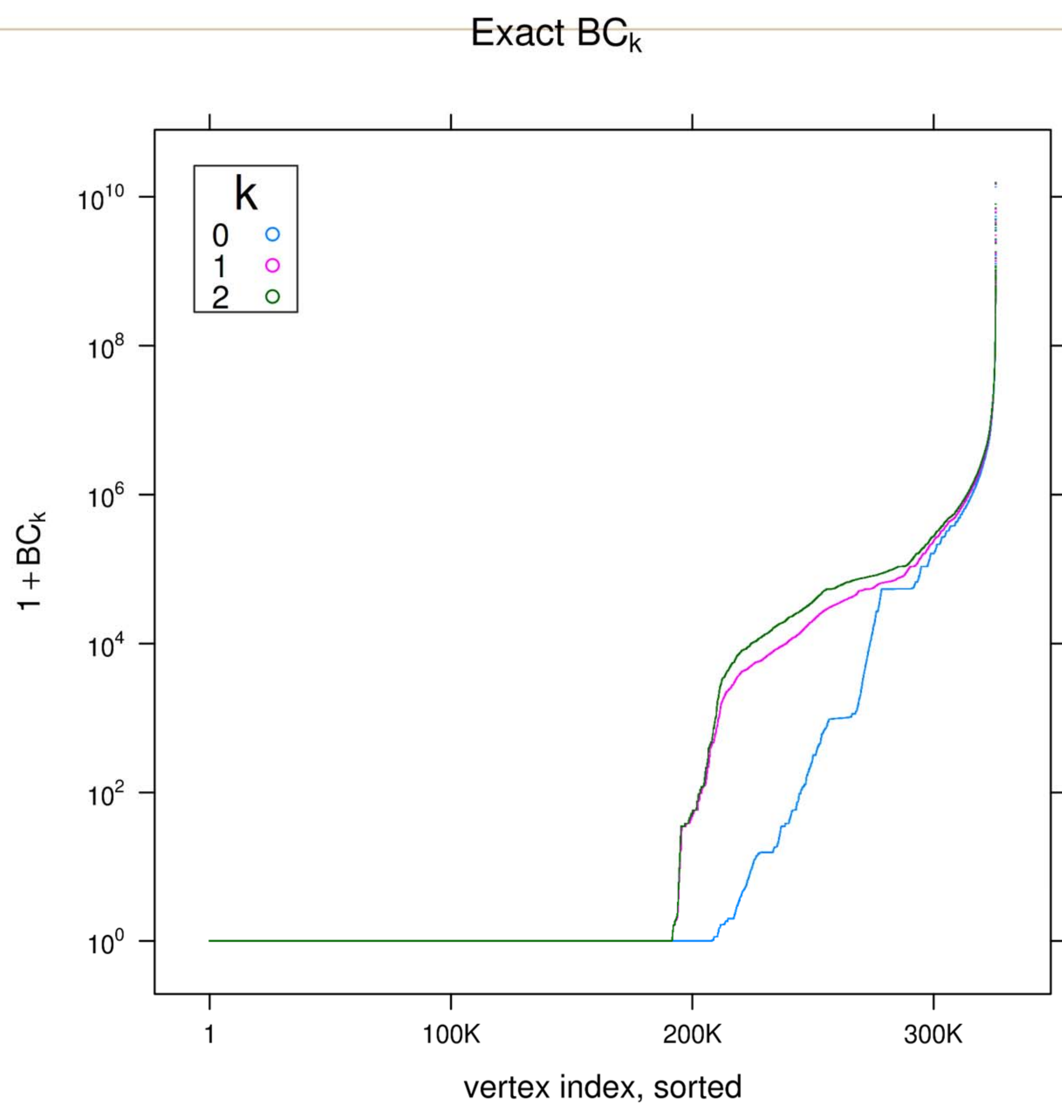
k-Betweenness algorithm (4/4)

- Dependency graph: each *i* is independent, only need *k+1* graph traversals for *(k+1)*(k+2)/2 accumulations*
- Then $BC_k(v)$ can be computed by summing over all *deltas* for a particular *v, k*.



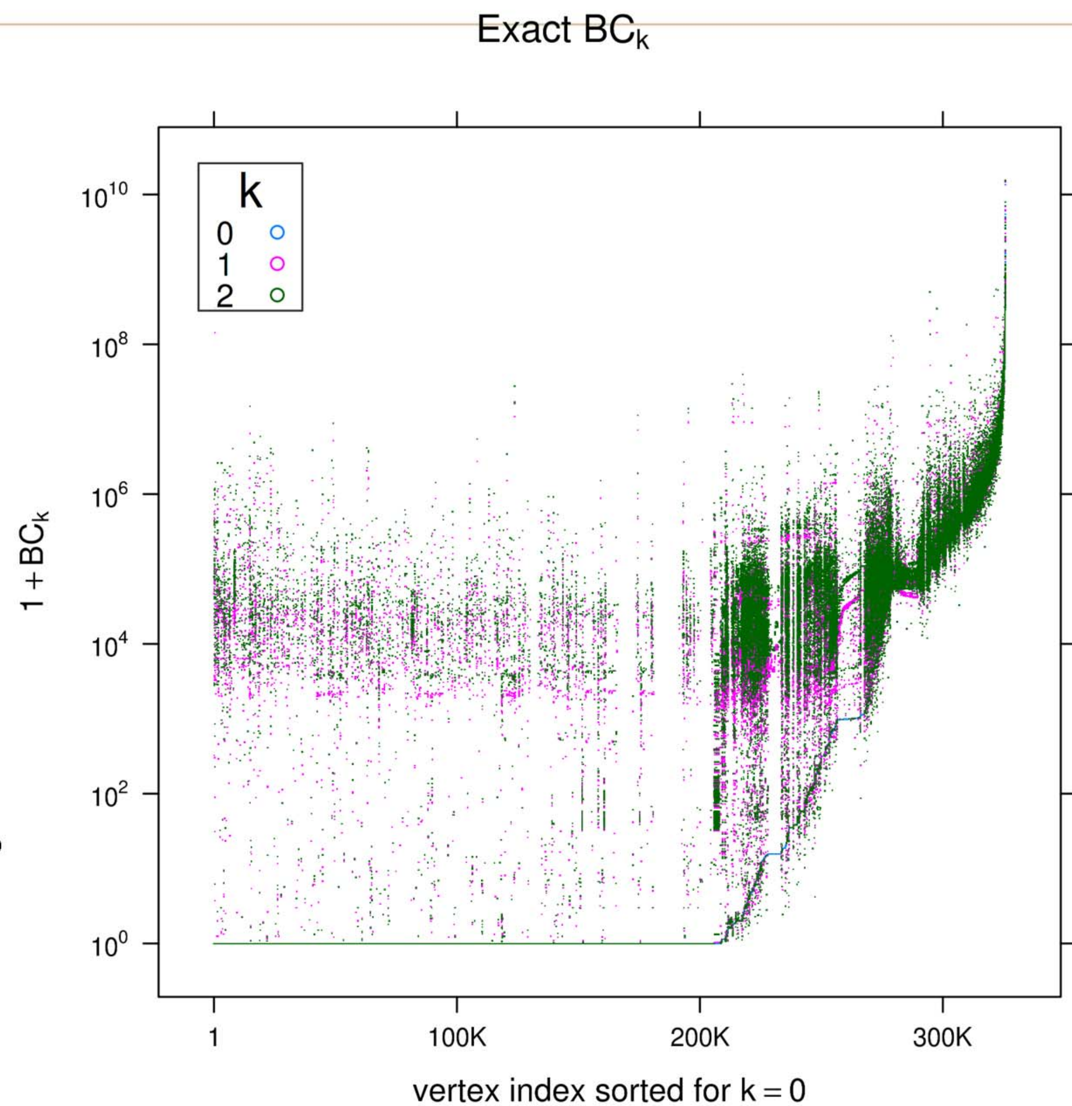
BC_k for k > 0: More Path Information

- Exact BC_k for k = 0, 1, 2
- On directed web graph
- 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



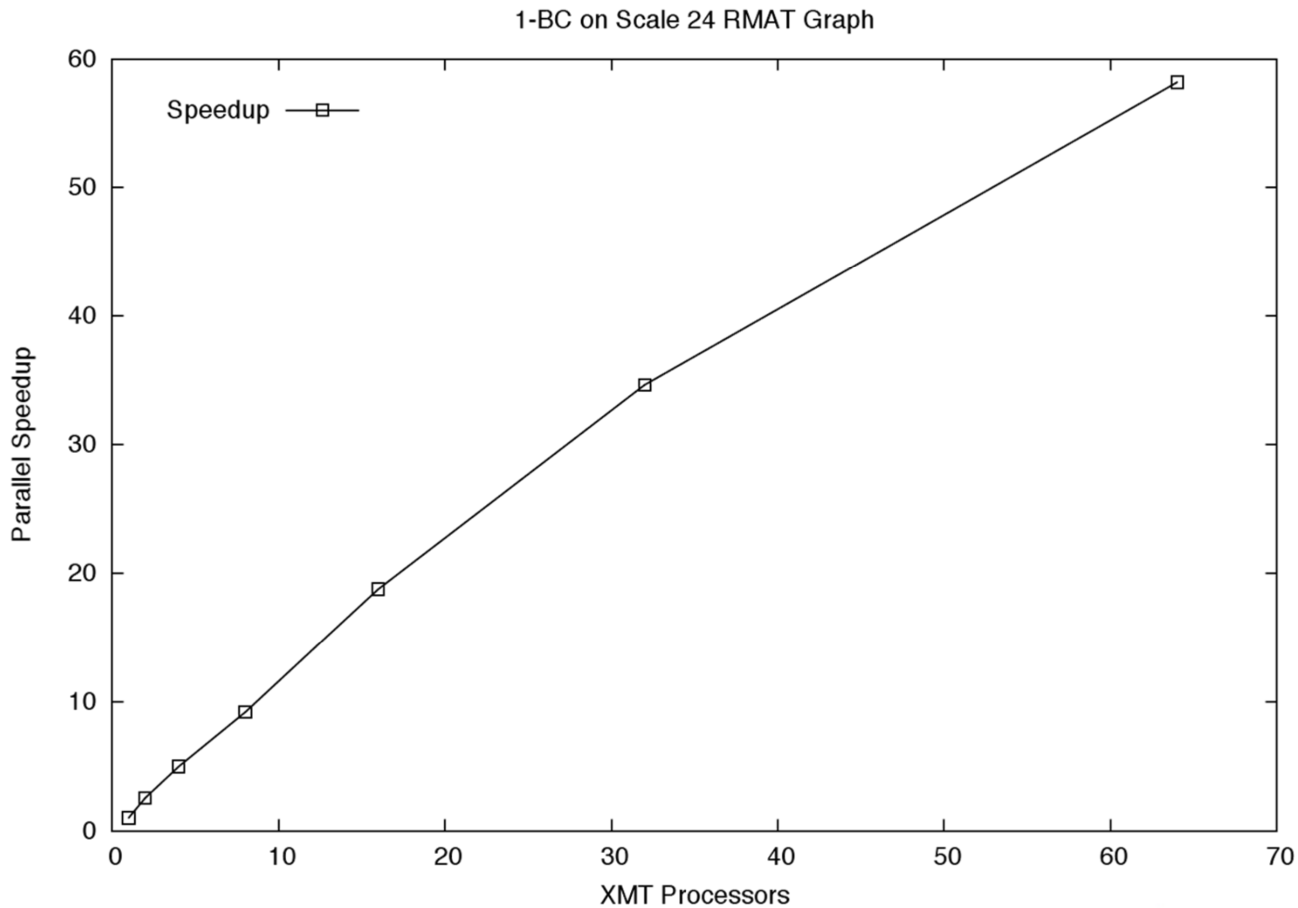
BC_k for k > 0: More Path Information

- Exact BC_k for k = 0, 1, 2
- On directed web graph
- 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!



Scalability of k-Betweenness Centrality

- 52x speedup for k=1 on a 64p Cray XMT

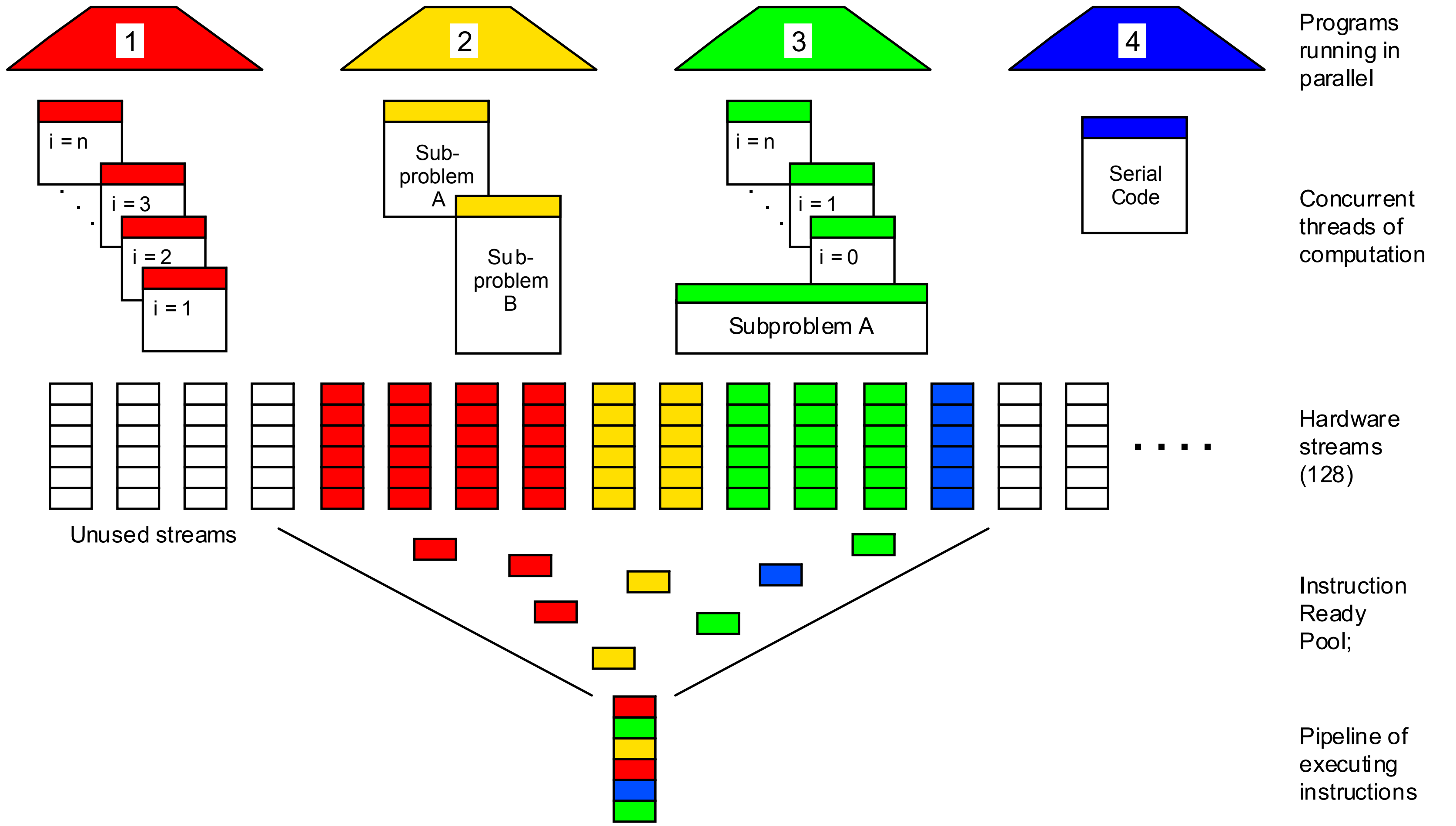


CRAY XMT ARCHITECTURE

Cray XMT Operation

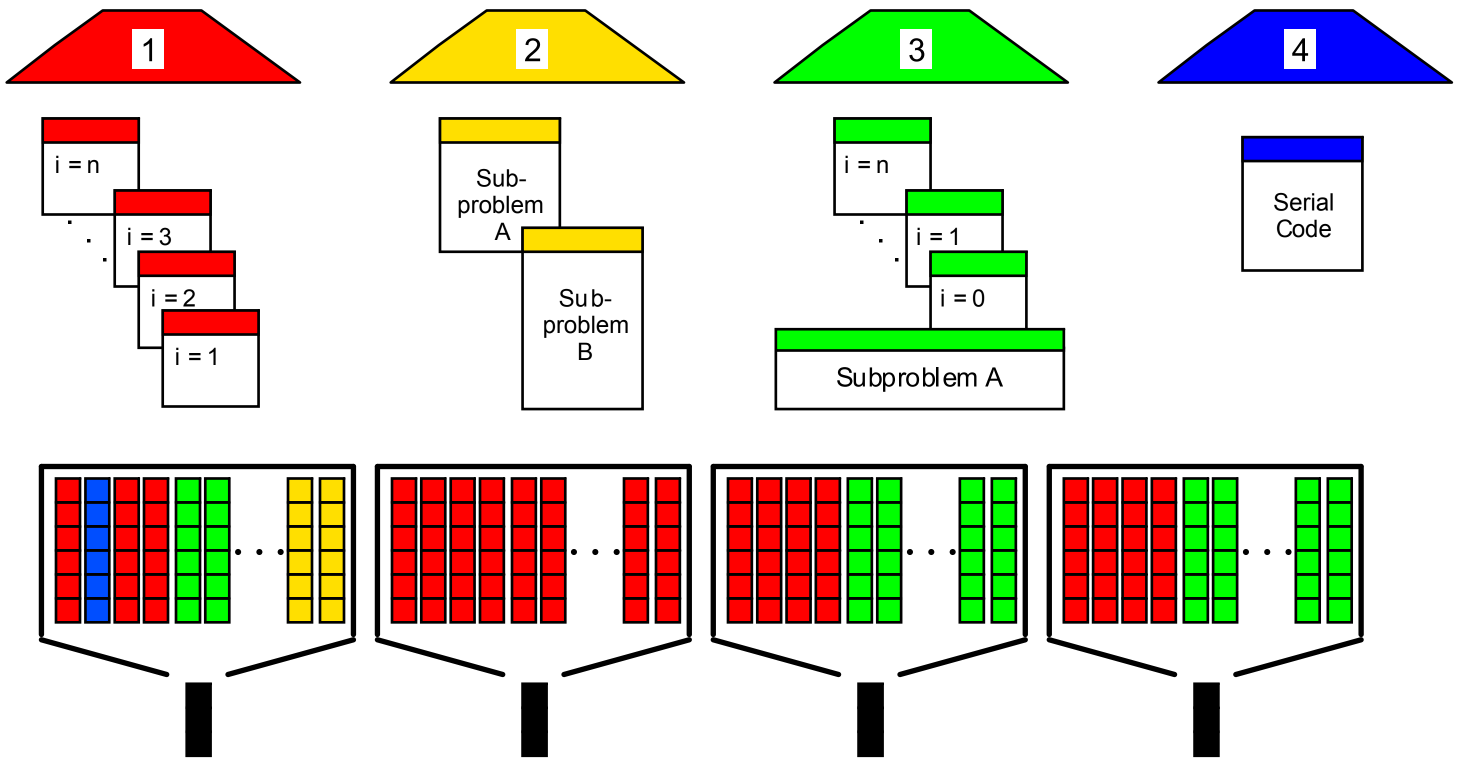
- Tolerates latency by extreme multithreading
  - Each processor supports 128 hardware threads
  - Context switch in a single tick
  - No cache or local memory
  - Context switch on memory request
  - Multiple outstanding loads
- Remote memory requests do not stall processors
  - Other streams work while the request gets fulfilled
- Light-weight, word-level synchronization
  - Minimizes access conflicts
- Hashed global shared memory
  - 64-byte granularity, minimizes hotspots
- High-productivity graph analysis!

XMT ThreadStorm Processor (logical view)



Slide Credit: Cray, Inc.

XMT ThreadStorm System (logical view)



Programs running in parallel

Concurrent threads of computation

Multithreaded across multiple processors

Slide Credit: Cray, Inc.

What is not important on XMT

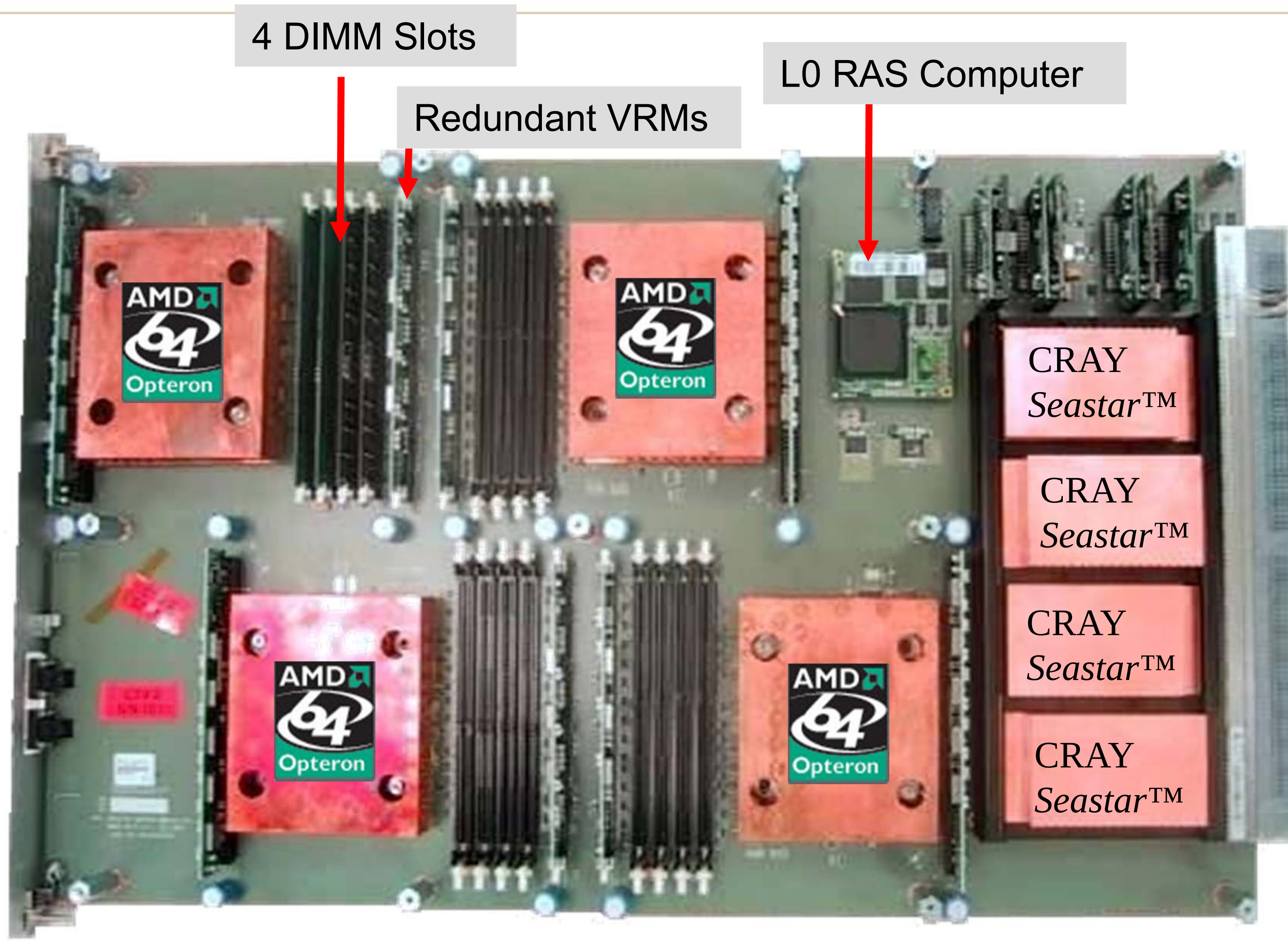
- Placing data near computation
- Modifying shared data
- Accessing data in order
- Using indirection or linked data-structures
- Partitioning program into independent, balanced computations
- Using adaptive or dynamic computations
- Minimizing synchronization operations

Red Storm

- Red Storm consists of over 10,000 AMD Opteron™ processors connected by an innovative high speed, high bandwidth 3D mesh interconnect designed by Cray (Seastar)
- Cray is responsible for the design, development, and delivery of the Red Storm system to support the Department of Energy's Nuclear stockpile stewardship program for advanced 3D modeling and simulation

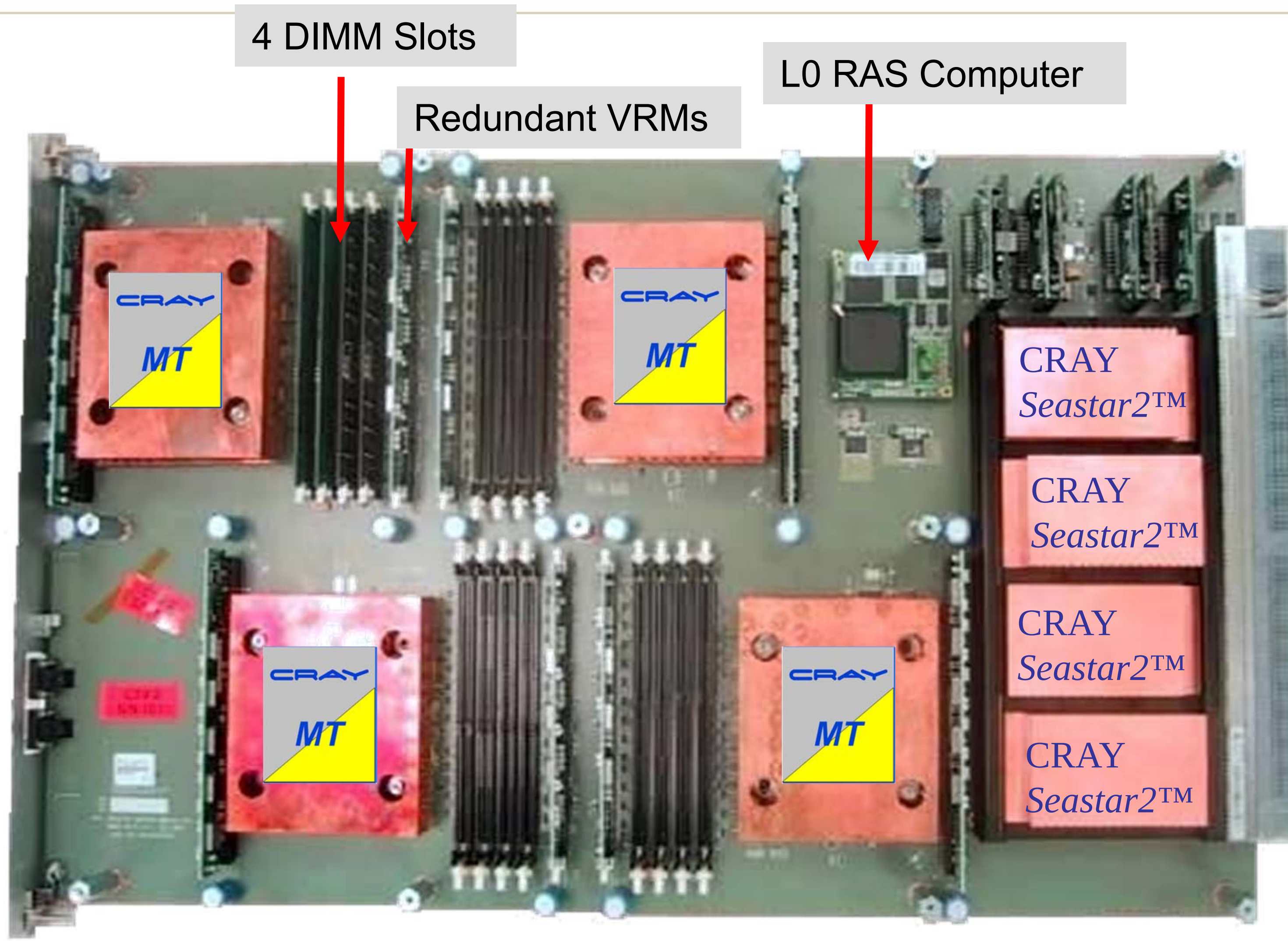
Slide Credit: Cray, Inc.

Red Storm Compute Board



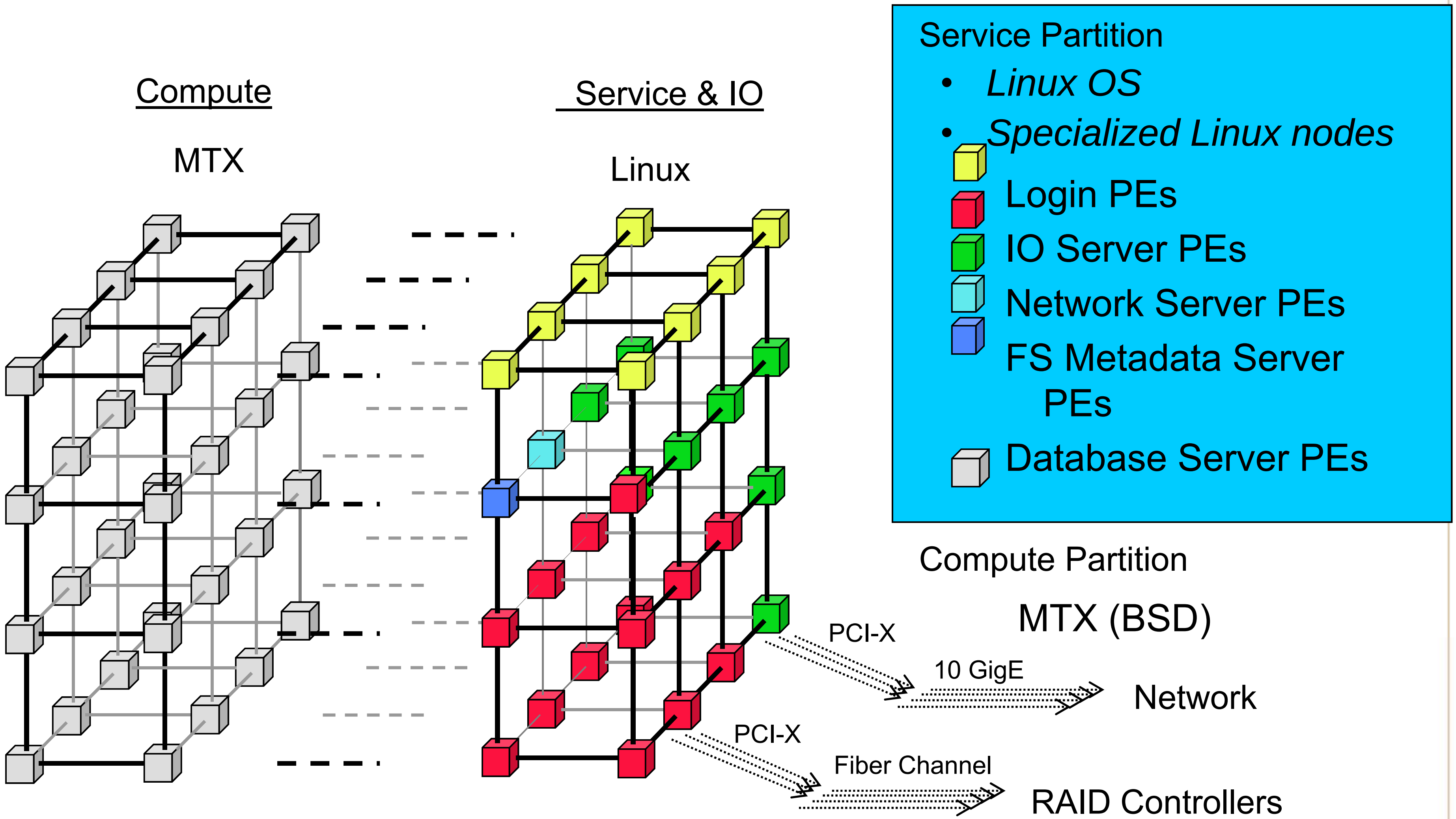
Slide Credit: Cray, Inc.

XMT Compute Board



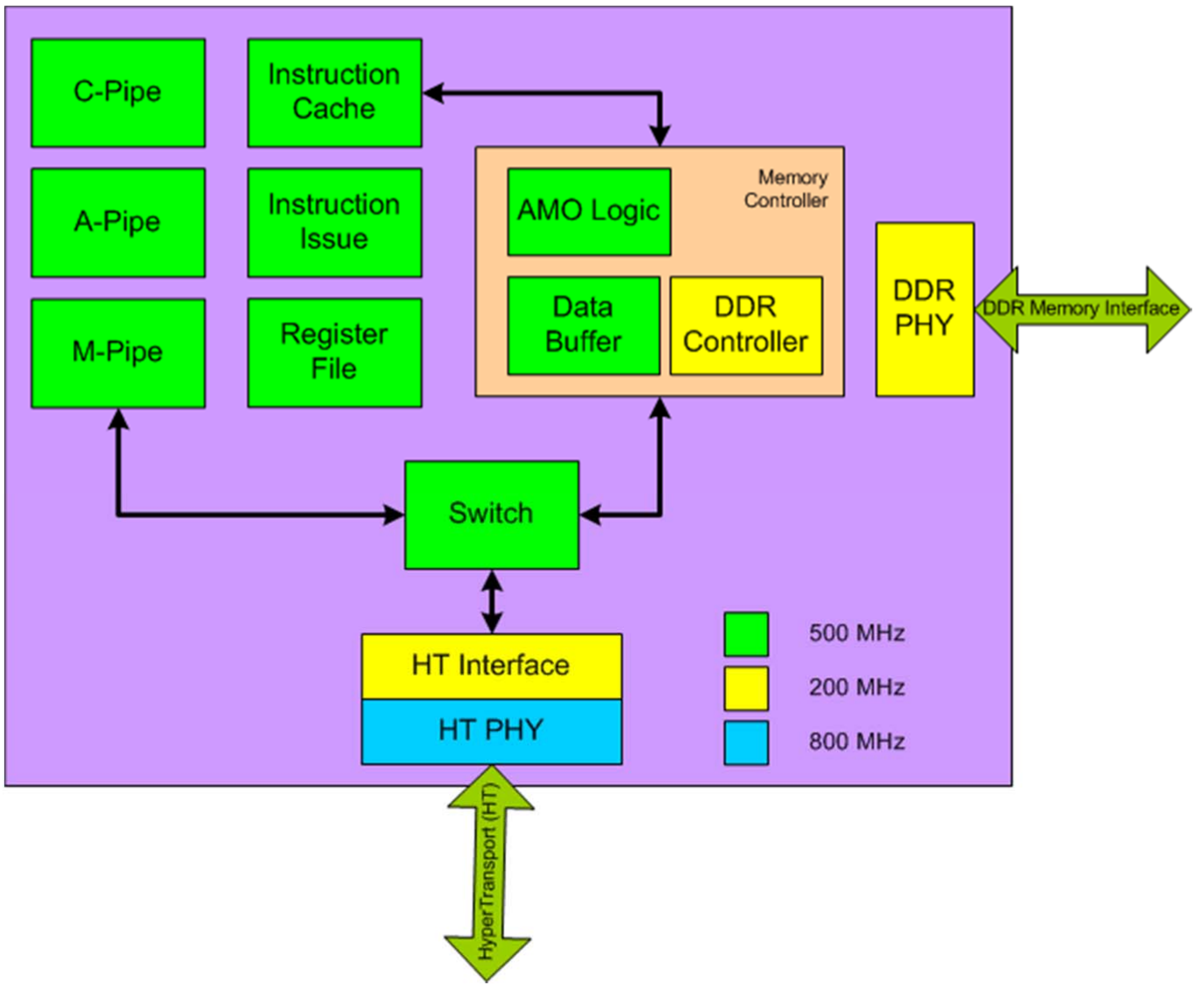
Slide Credit: Cray, Inc.

XMT system architecture



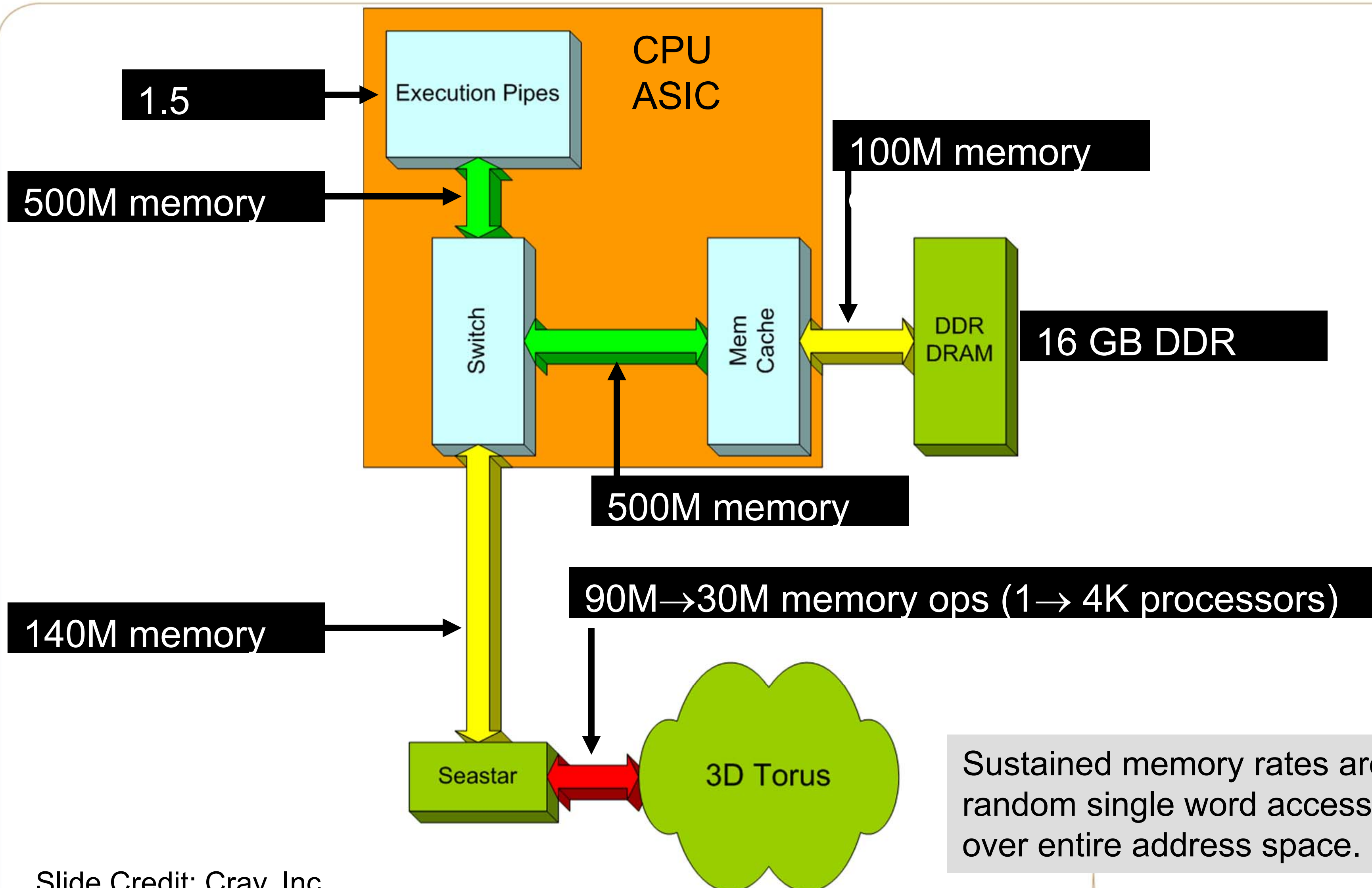
Slide Credit: Cray, Inc.

XMT ThreadStorm CPU



Slide Credit: Cray, Inc.

XMT Speeds and Feeds



Sustained memory rates are for random single word accesses over entire address space.

Slide Credit: Cray, Inc.

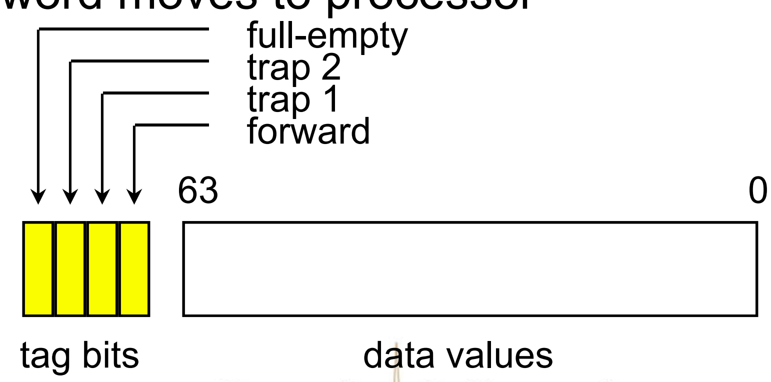
XMT memory

- Shared memory
  - Some memory can be reserved as local memory at boot time
  - Only compiler and runtime system have access to local memory

- Memory module cache
  - Decreases latency and increases bandwidth
  - No coherency issues

- 8 word data segments randomly distributed across the memory system
  - Eliminates stride sensitivity and hotspots
  - Makes programming for data locality impossible
  - Segment moves to cache, but only word moves to processor

- Full/empty bits on all data words



Slide Credit: Cray, Inc.