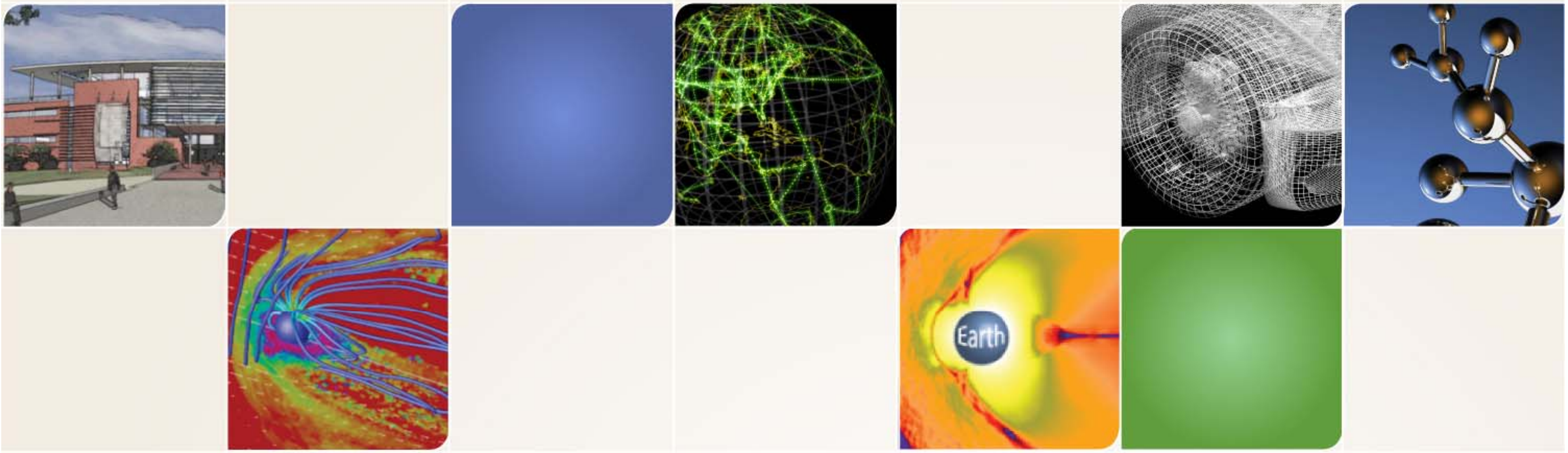


High-Performance Combinatorial Techniques for Analyzing Dynamic Interaction Networks

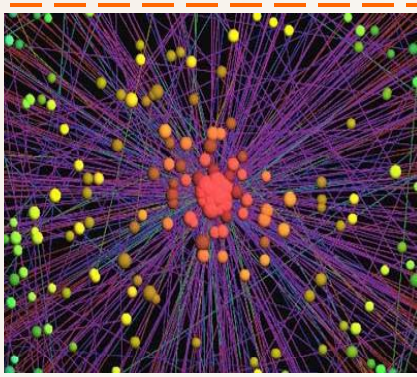
Kamesh Madduri David A. Bader

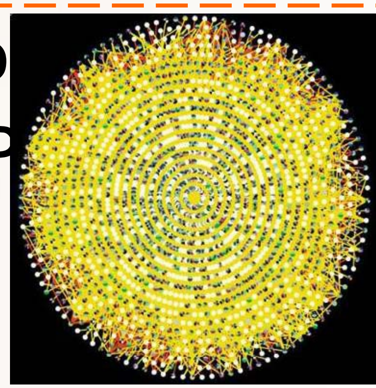
Georgia Tech College of Computing
Computational Science and Engineering

Acknowledgment of Support

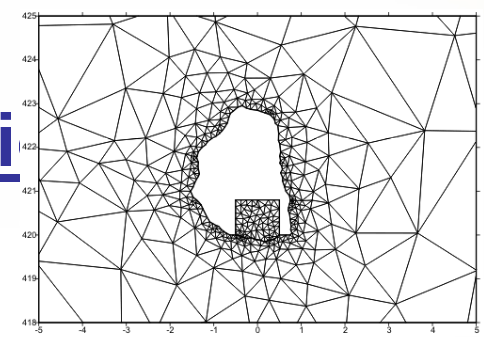
HPC for Large Graphs

- Emerging applications: Intelligence, health care, systems biology, Viral marketing …
- Graph abstractions at the core
- Social network analysis: **fundamentally different** graph topologies, and computations!
  - Graph traversal is one of the thirteen Berkeley *dwarf kernels*





Informatics: dynamic, high-dimensional data



Static networks, Euclidean topologies

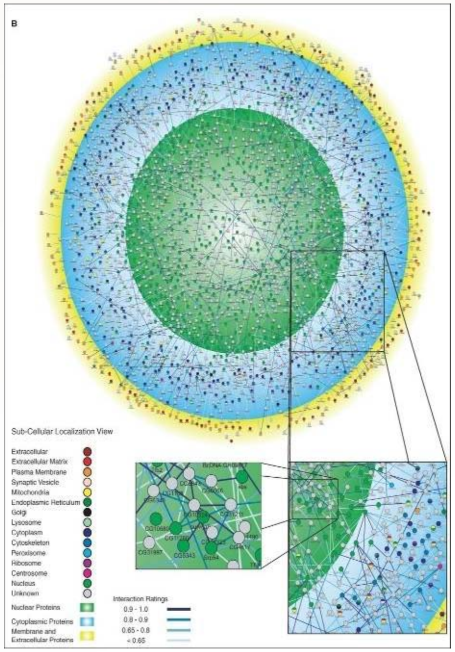
Image Sources: visualcomplexity.com (1,2), MapQuest (3)

Information Networks

- Massive, evolving, data-rich

Online social networks

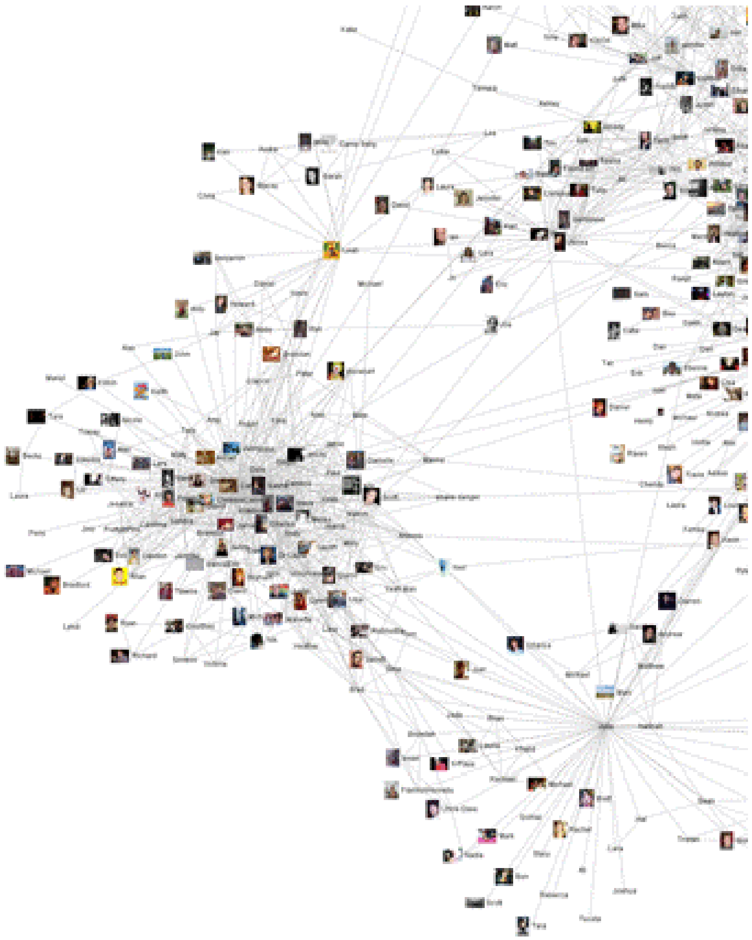
Systems Biology



Images source:
visualcomplexity.com

Georgia Tech College of Computing
Computational Science and Engineering

SNAP



Exploratory Network Analysis

SNAP parallel framework

Advanced Graph Analysis Queries
partitioning, subgraph isomorphism ...

Graph metrics and **Preprocessing routines**

Graph kernels
BFS, MST, connected components ...

Graph representation
formats, data structures

Georgia Tech College of Computing
Computational Science and Engineering

Dynamic Interaction Networks

- How do we adapt SNAP to dynamic interaction networks?
  - New data structures
  - Kernels
  - Algorithms

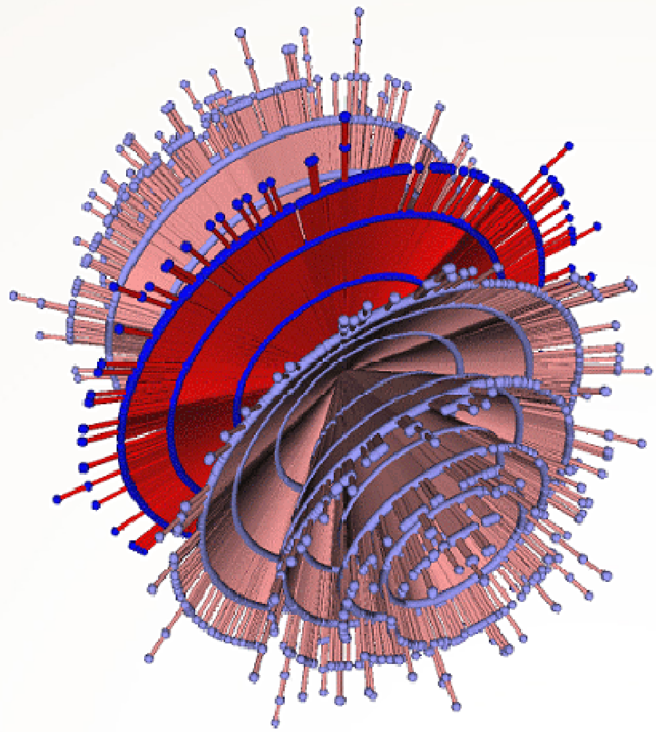


Image Source: Seokhee Hong

Dynamic Interaction Networks

- Analysis of dynamic interaction networks poses new computational challenges

Novel approaches

Classical graph algorithms

Data stream algorithms

Spectral techniques

Complex Network Analysis & Empirical studies

Applications involving Dynamic Interaction Networks

Dynamic graph algorithms

Realistic modeling

Many-core

Stream computing

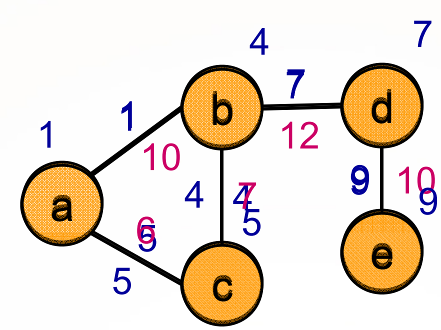
Affordable exascale data storage

Enabling technologies

Graph Representation

- Augment static graph representation with explicit time-ordering on vertices and edges [KKK02]
- Temporal graph *G(V, E, λ)*, with each edge having a time label λ(e), a non-negative integer value
- The time label is application-dependent
- Can define multiple time labels on vertices and edges

Interaction	Time-interval
a b	1-10
b c	4-7
a c	5-6
b d	7-12
d e	9-10



Georgia Tech College of Computing
Computational Science and Engineering

Graph Representation: adjacency data structures

- Static representation: adjacency arrays
  - Space-efficient, cache-friendly
- In dynamic networks, we need to primarily support edge and vertex membership queries, insertions, and deletions
  - Should be space-efficient, with low synchronization overhead
- We experiment with various representations
  - Resizable adjacency arrays
  - Adj. arrays, sorted by vertex identifiers
  - Adj. arrays for low-degree vertices, treaps for high-degree vertices (for sparse graphs with power-law degree distributions)
  - Memory requirements: ~ (4n+m)w bytes, w: memory-word size
- We can choose appropriate representation based on the insertion/deletion ratio, and graph structural update rate.

Georgia Tech College of Computing
Computational Science and Engineering

Processing Structural Updates

- Insertion of an edge
  – Update adjacency list of corresponding vertex
- Deletion of an edge
  – Delete from adjacency list
  – Time label
- Insertion of a vertex
  – Time label
- Deletion of a vertex
  – Time label
- Batched updates
  – Sort by vertex and edge identifiers

Multicore and SMP Servers

IBM p5 570

- 16-way Power5 SMP
- 1.9 GHz processor
- 256 GB physical memory
- 32KB L1D, 2MB L2, 32MB L3
- 8-way superscalar
- SMT on each core

Sun Fire T2000 (First gen. Niagara)

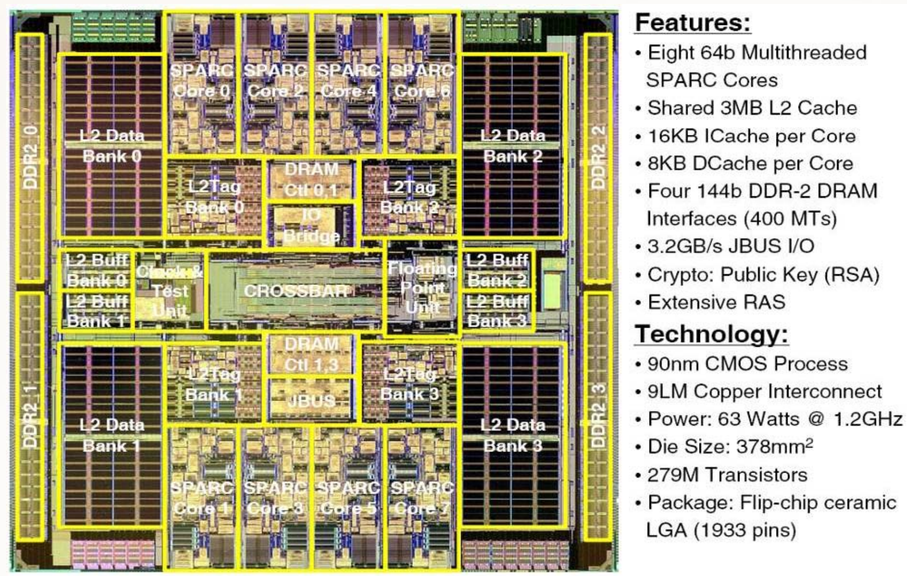
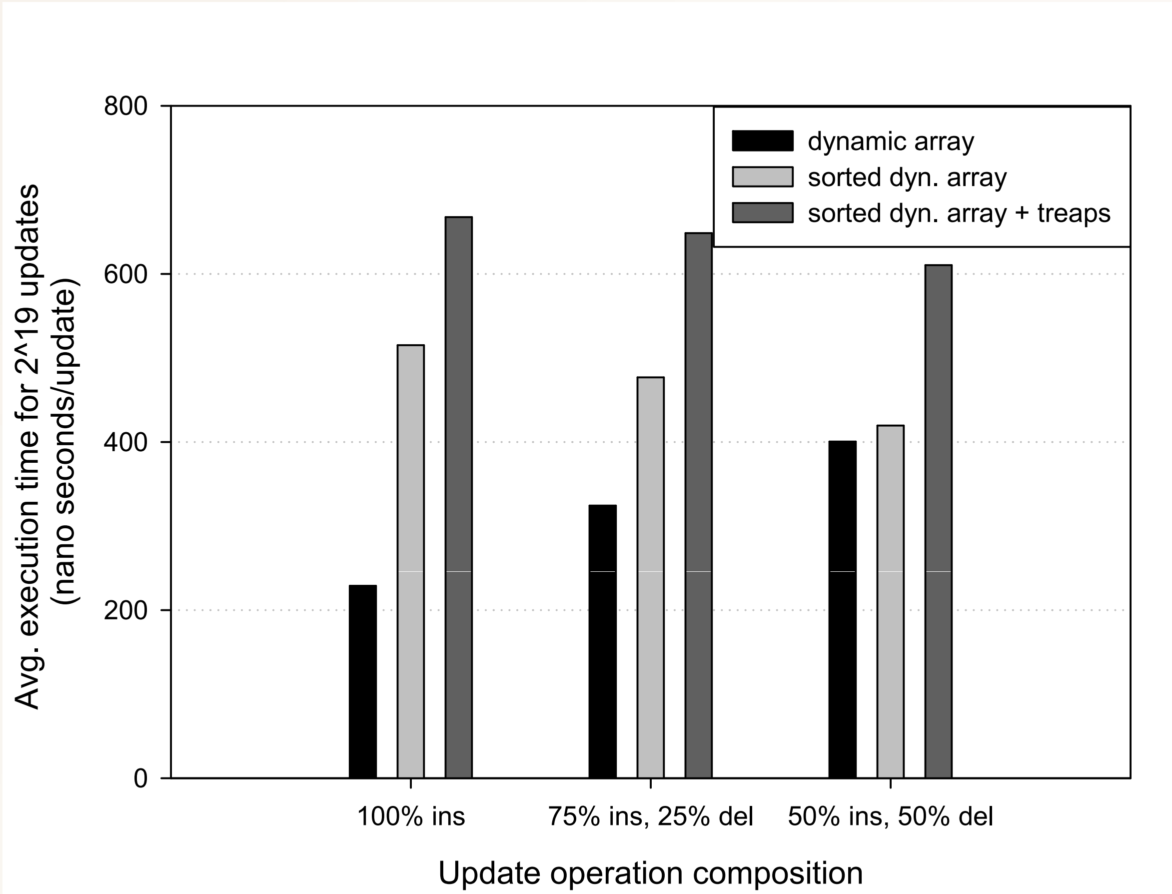


Image Sources: ibm.com and sun.com

Georgia Tech College of Computing
Computational Science and Engineering

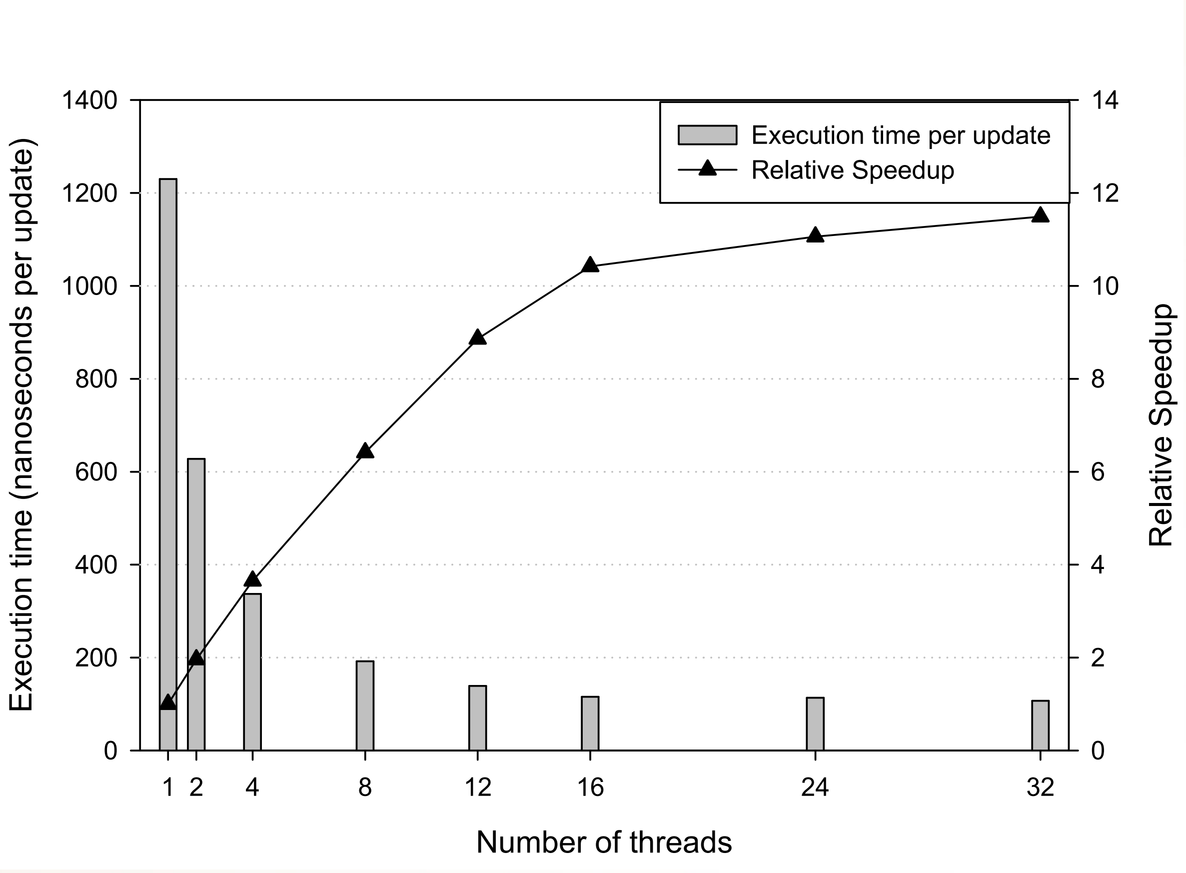
Dynamic network updates: Performance



Graph: 1M vertices and 4M edges,
System: 3.2 GHz Xeon

Georgia Tech College of Computing
Computational Science and Engineering

Structural Updates: Parallel Performance



Graph: 25M vertices and 200M edges,
System: Sun Fire T2000

Georgia Tech College of Computing
Computational Science and Engineering

Alternate data representations

- Compressed representations: eg. web-graph
  - Vertex reordering, compact interval representations, compression of similar adjacency lists
- Processing dynamic insertions and deletions
  - Dynamic tree problem for connectivity
  - Self-adjusting data structures: ST (link-cut) trees, top trees, RC-trees …
  - ST-trees are simple to implement, perform well for low-diameter graphs [Tarjan & Werneck, WEA07]
  - Supporting concurrent insertions and deletions?

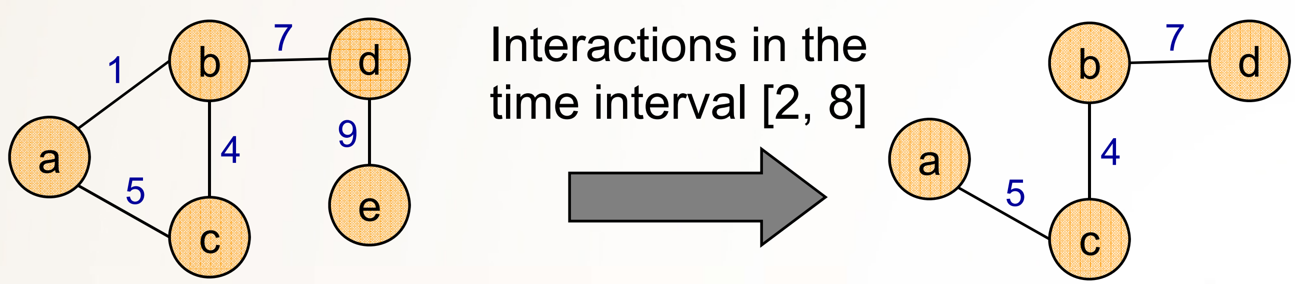
Graph kernels

- Fine-grained parallelization of fundamental building blocks, using the temporal interaction network representation
- Enables efficient implementation of high-level algorithms
- Parallel approaches for the following kernels [Bader, Madduri 08]
  - Induced subgraphs
  - Connectivity, spanning forest
  - BFS
  - Single-source shortest paths

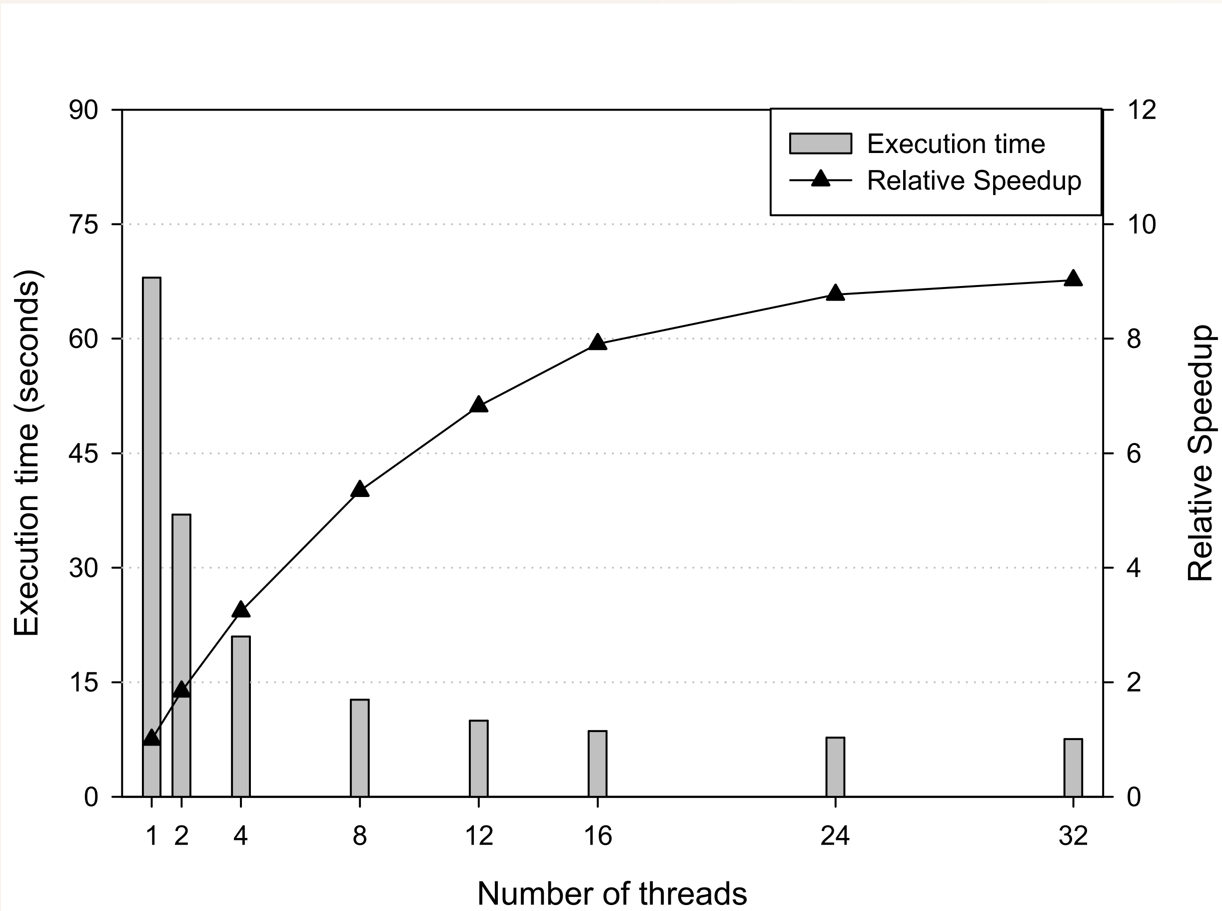
Georgia Tech College of Computing
Computational Science and Engineering

Induced Subgraphs

- Utilizing temporal information, dynamic graph queries can be reformulated as problems on static networks
  - eg. Queries on entities up to a particular time instant, time interval etc.
- Induced subgraph kernel: facilitates this dynamic → static graph problem transformation
- Assumption: the system has sufficient physical memory to hold the entire graph, ~ (m+4n)w bytes
- Computationally, very similar to doing batched insertions and deletions, linear work



Induced Subgraphs: Parallel Performance



- We reduce execution time of linear-work kernels from minutes to seconds for massive small-world networks (billions of vertices and edges)

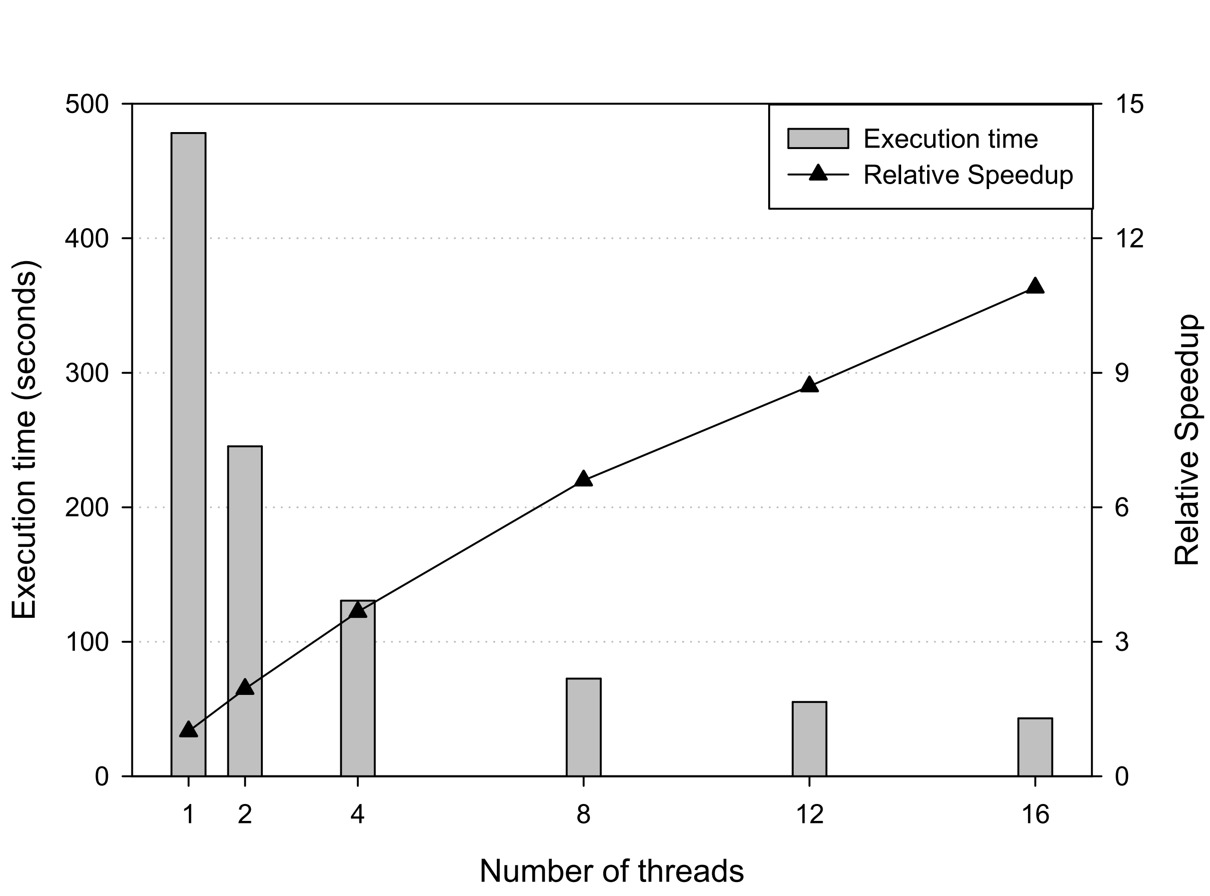
Graph: 500M vertices and 2B edges,
System: IBM p5 570 SMP

Graph Traversal (BFS)

- Level-synchronous graph traversal for low-diameter graphs, each edge in the graph visited only once.
- Fast, efficient implementations on shared memory systems
- Dynamic networks
  - Filter vertices and edges according to time-stamp information, recompute BFS from scratch
  - Dynamic graph algorithms for BFS: better amortized work bounds, space requirements are higher

Georgia Tech College of Computing
Computational Science and Engineering

BFS: Parallel Performance



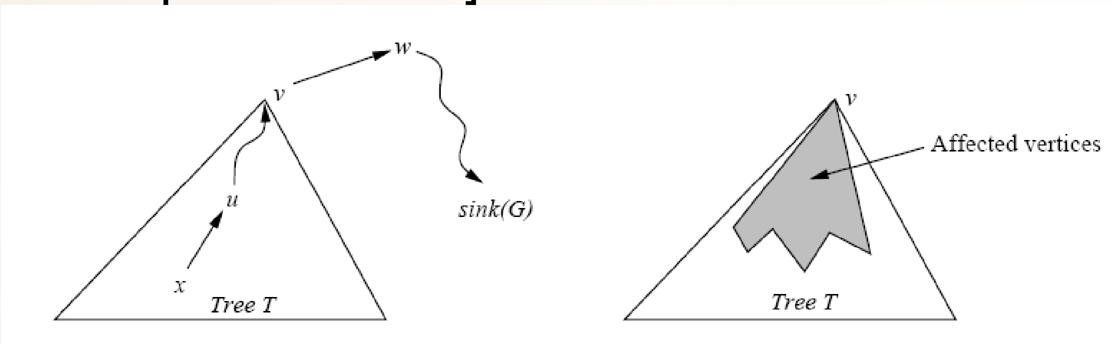
We reduce execution time of linear-work kernels from minutes to seconds for massive small-world networks (billions of vertices and edges)

Graph: 500M vertices and 4B edges,
System: IBM p5 570 SMP

Georgia Tech College of Computing
Computational Science and Engineering

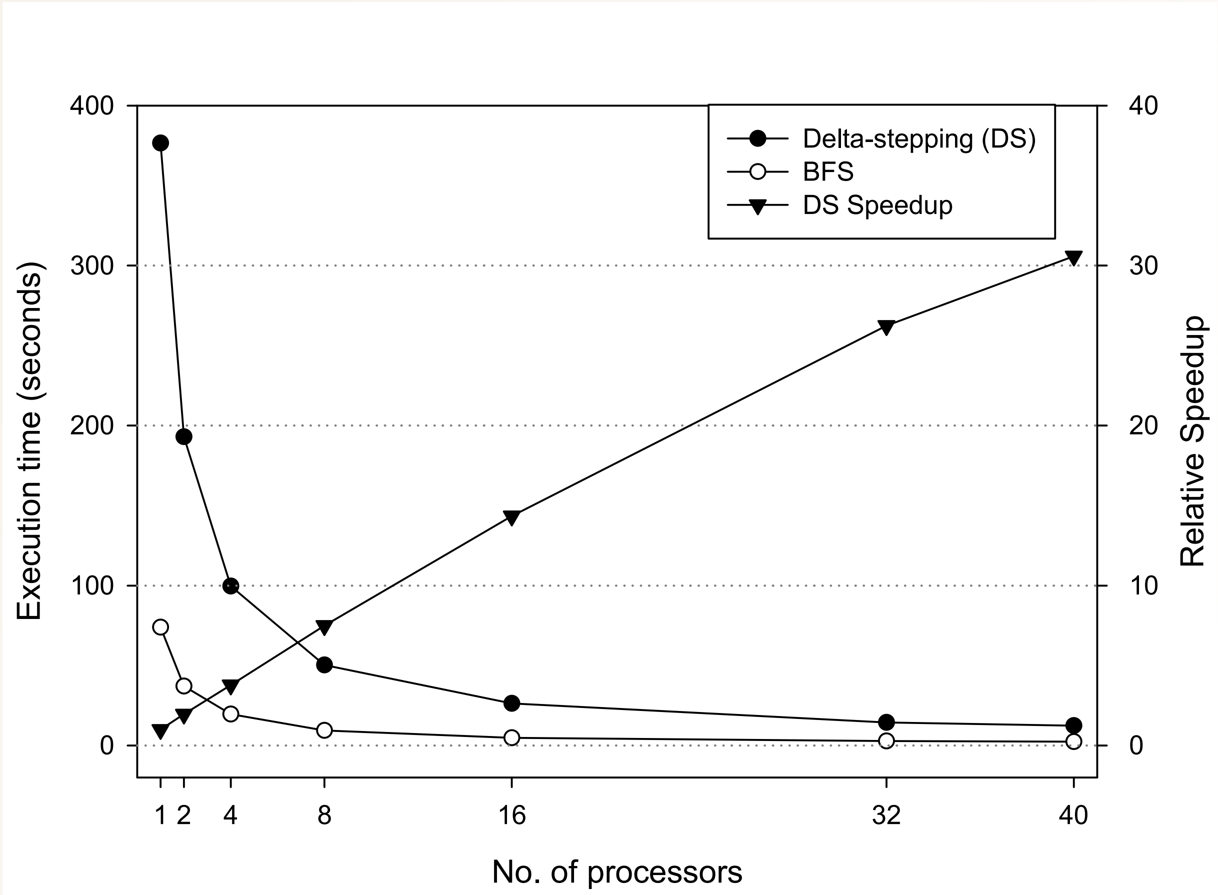
Shortest Paths

- SSSP for dynamic networks is more challenging
- We design a parallel formulation of the Ramalingam-Reps algorithm for arbitrary graphs, under edge deletions
- Affected region in the graph due to edge insertions and deletions
- Two phases in the algorithm:
  - Phase 1: compute the set of affected edges, similar to a topological ordering algorithm
  - Phase 2: update distance values, similar to a batched version of Dijkstra's algorithm [use prior Delta-stepping parallel implementation]



Georgia Tech College of Computing
Computational Science and Engineering

Parallel Performance: BFS and Shortest Paths



Graph: 256M vertices and 1B edges,
System: Cray MTA-2

Connectivity

- Parallel Connected components for static graphs: O(m+n) work, based on the Shiloach-Vishkin algorithm
- Extension to dynamic networks
  - Induced subgraphs, followed by the static connected components algorithm
- Connectivity queries can be answered by maintaining a spanning forest of the graph
- Dynamic connectivity is a well-studied problem
  - Poly-log update and query times require linear pre-processing time and space, and dynamic tree data structures
  - Dynamic approaches are useful only when the rate of queries and updates are high

Algorithms

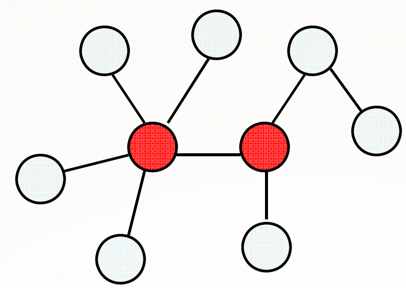
- Formulating Network Analysis metrics in a temporal setting are open problems
  - Betweenness Centrality
  - Community Identification

Georgia Tech College of Computing
Computational Science and Engineering

Betweenness Centrality (BC)

- Centrality metrics: Quantitative measures to capture the importance of a node/vertex/actor in a graph
  - Degree, Closeness, Stress, Betweenness
- Betweenness

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

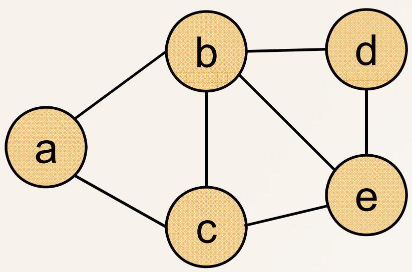


- σ_{st} -- No. of shortest paths between vertices *s* and *t*
- $\sigma_{st}(v)$ -- No. of shortest paths between vertices *s* and *t* passing through *v*

- Exact BC is compute-intensive

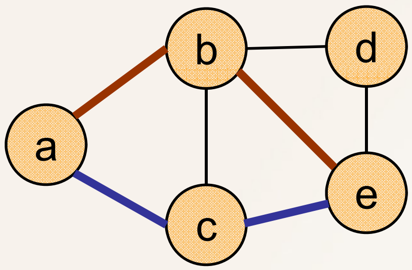
Georgia Tech College of Computing
Computational Science and Engineering

Temporal Path



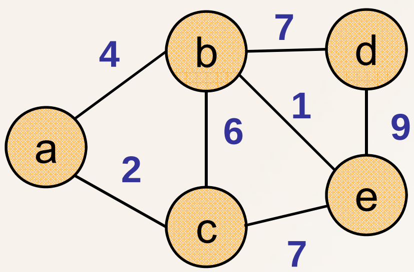
Georgia Tech College of Computing
Computational Science and Engineering

Temporal Path

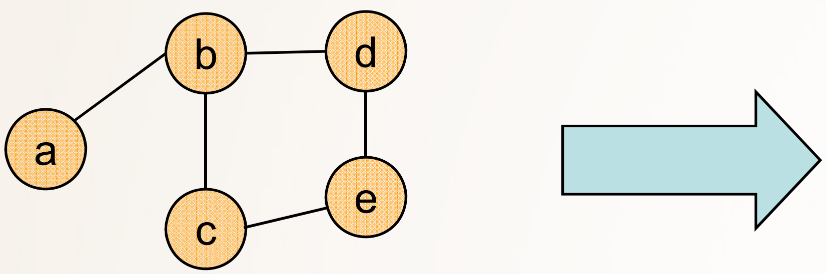


Two unweighted shortest paths between
a and e

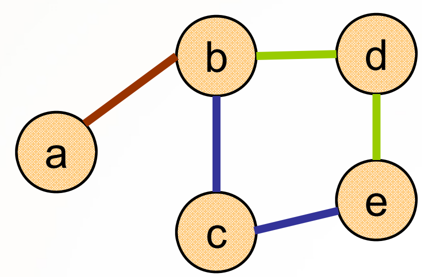
Temporal Path



Consider edges in the time interval 3-10



Two different shortest paths between a and e!



Community Identification

Algorithm 1: Temporal betweenness centrality-based divisive clustering algorithm

---

Input: $G(V, E)$, length function $l : E \rightarrow \mathbb{R}$, time-stamp $\lambda(e) \forall\ e \in E$.

Output: A partition $C = (C_1, ..., C_k)$ $(C_i \neq \phi$ and $C_i \cap C_j = \phi)$ of V that maximizes modularity; A dendrogram D representing the clustering steps.

1. Preprocessing step: Compute *Biconnected components*, identify articulation points and bridges.
2. $numIter \longleftarrow 0$;
3. **while** $numIter < m$ **do**
4.   Find edge e_m with the highest *approximate temporal betweenness centrality* score **in parallel**.
5.   Mark edge e_m as *deleted* in the graph G.
6.   Run connected components on G, update dendrogram and number of clusters **in parallel**.
7.   Compute modularity of the current partitioning **in parallel**.
8. $numInter \longleftarrow numIter + 1$;

   end

9. Inspect the dendrogram, set C to the clustering with the highest modularity score.

Conclusions

- We study data representations and parallel approaches for solving massive interaction network problems
- Applications: Community identification, centrality analysis