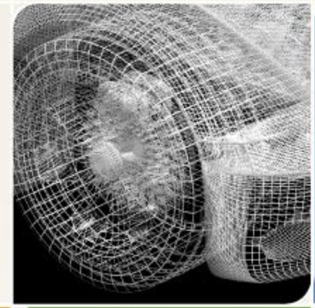
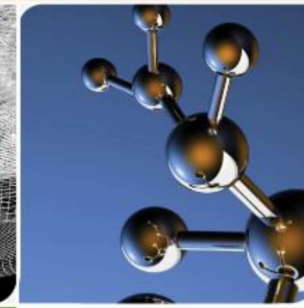
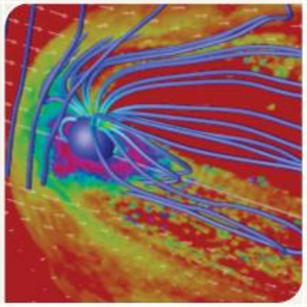
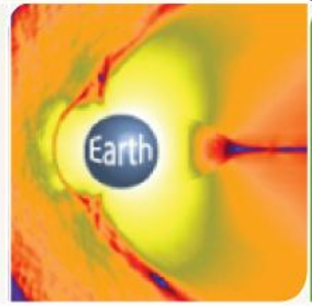








High-Performance Combinatorial Techniques for Analyzing Massive Dynamic Interaction Networks

David A. Bader and Kamesh Madduri

Georgia Tech College of Computing
Computational Science and Engineering

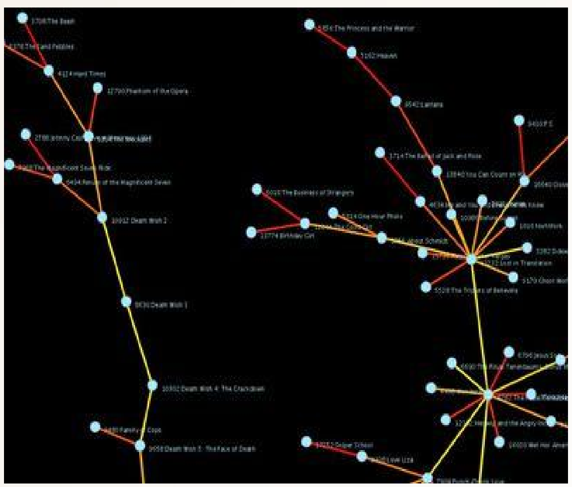
Talk Overview

- A graph-theoretic representation of temporal interaction networks
- Analysis kernels
- Algorithms for dynamic interaction networks
- SNAP: An open-source, parallel library for exploratory analysis of interaction networks

Motivation

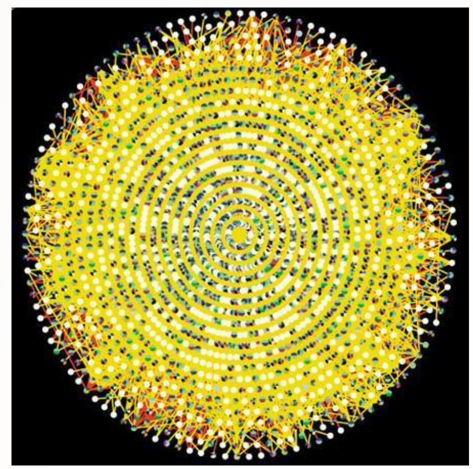
- Graph abstractions and algorithms are extensively used to analyze massive interaction data sets

Social networks



Community identification, Security and surveillance, viral marketing.

Computational Biology



Systems biology, disease modeling, behavioral ecology.

Images source: visualcomplexity.com

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

Parallel computing

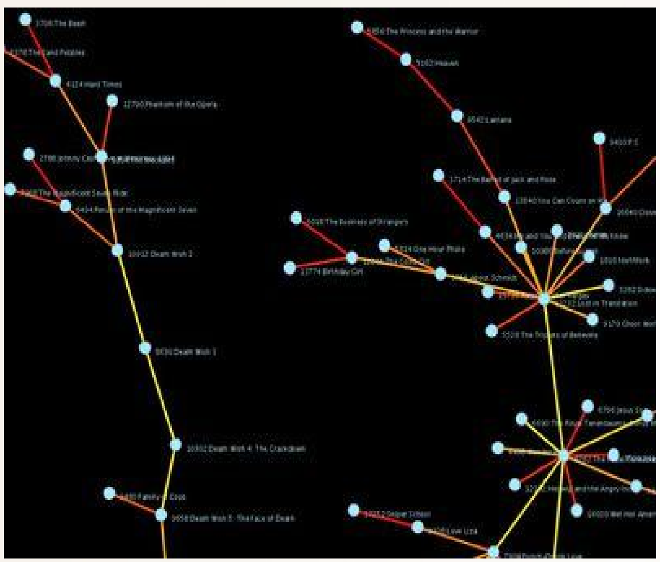
Large-scale Databases

Enabling technologies

Georgia Tech | College of Computing | Computational Science and Engineering

Our contributions

- An efficient computational framework for analyzing dynamic interaction networks
- Parallel graph analysis algorithms optimized for shared memory multi-core, SMP and multithreaded systems



Interaction data

Compact graph representation

Optimized graph kernels

Network Analysis Techniques

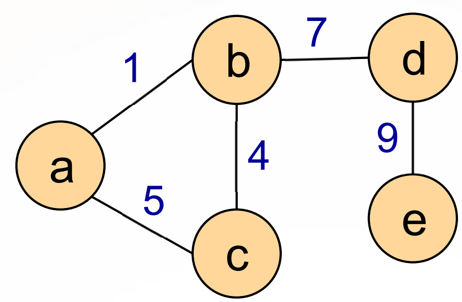
Previous Work

- We have designed fast parallel algorithms and efficient implementations for several graph theory kernels
  - List ranking, Connected Components, Spanning tree, MST, Graph traversal, Shortest paths
  - Algorithms: Centrality analysis, community identification
  - Applications: Protein-interaction network, social network analysis
- How do we adapt and extend these techniques to dynamic interaction networks?

Graph Representation

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

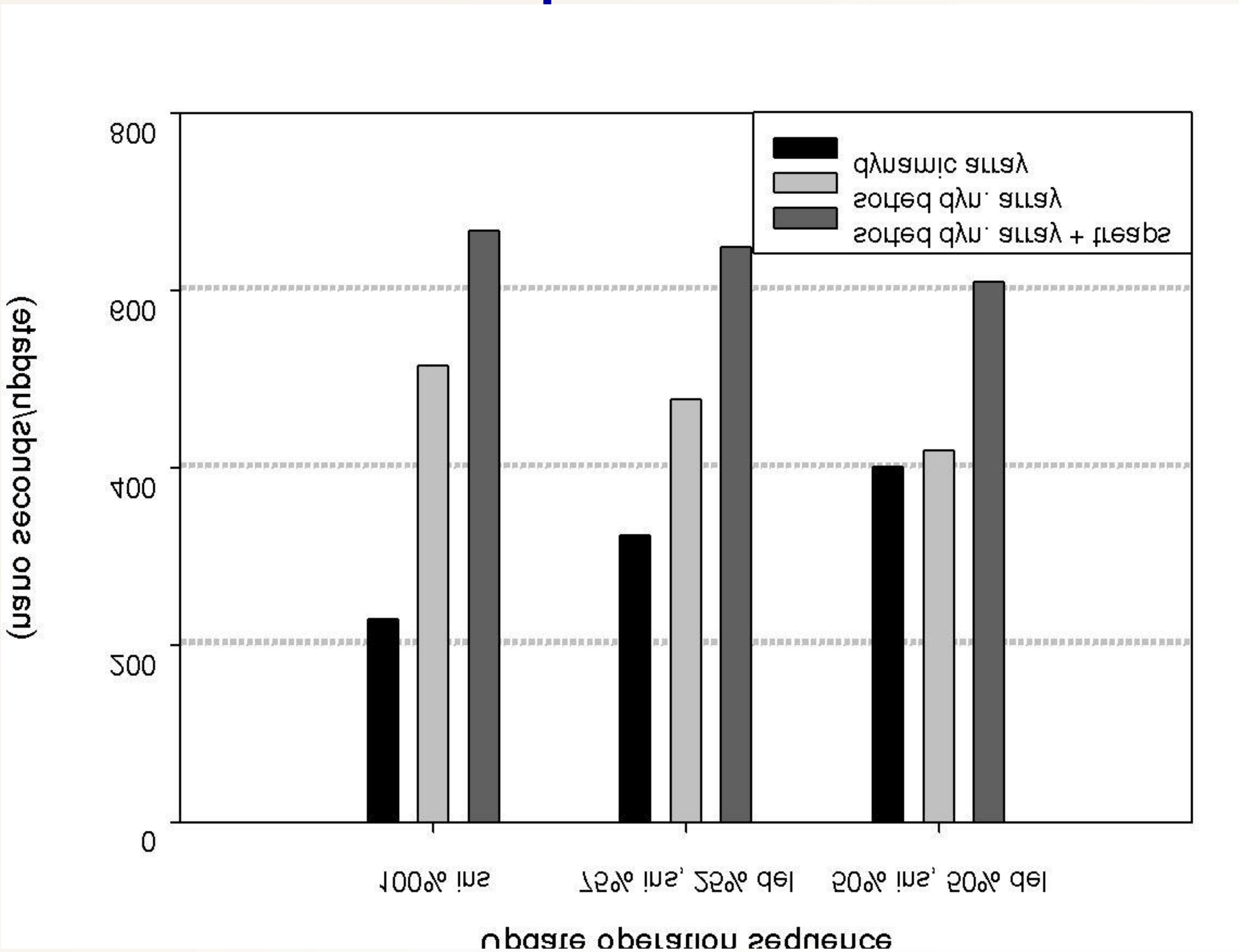
Interaction	Time-step
a b	1
b c	4
a c	5
b d	7
d e	9



Graph Representation: 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
- Batched update operations, set operations on treaps
- We can choose appropriate representation based on the insertion/deletion ratio, and graph structural update rate.

Dynamic network updates: Performance



Initial graph: synthetic scale-free network of 2^{20} vertices and 2^{22} edges; 2^{20} edge updates

Parallel performance results on Sun UltraSparc T2000 (16 threads)

Data structures

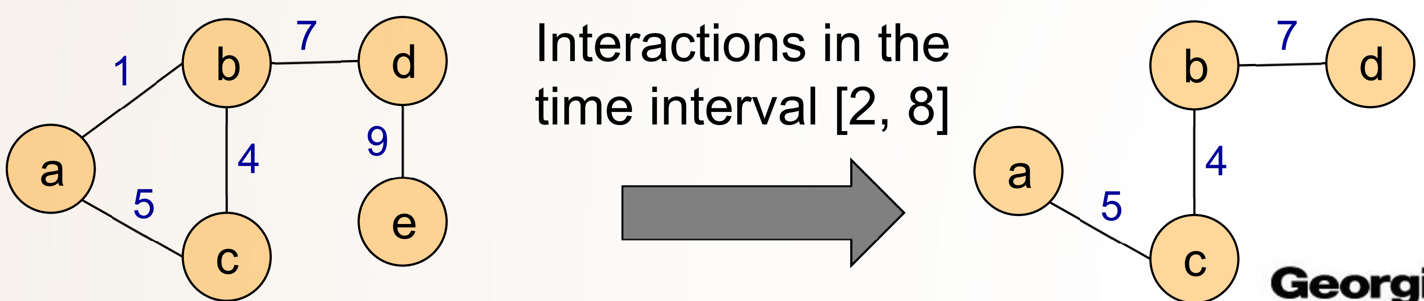
- Compressed representations: eg. web-graph
  - Vertex reordering, compact interval representations, compression of similar adjacency lists
- Processing dynamic insertions and deletions
  - Dynamic tree problem for connectivity
  - Self-adjusting data structures: ST (link-cut) trees, top trees, RC-trees …
  - 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 kernels, using the temporal interaction network representation; this enables efficient implementation of high-level algorithms
- We have preliminary results for the following kernels
  - Induced subgraphs
  - Connectivity, spanning forest
  - BFS
  - Single-source shortest paths

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, Induced subgraphs reduces to dynamic edge insertion and deletion problem, O(m+n) work

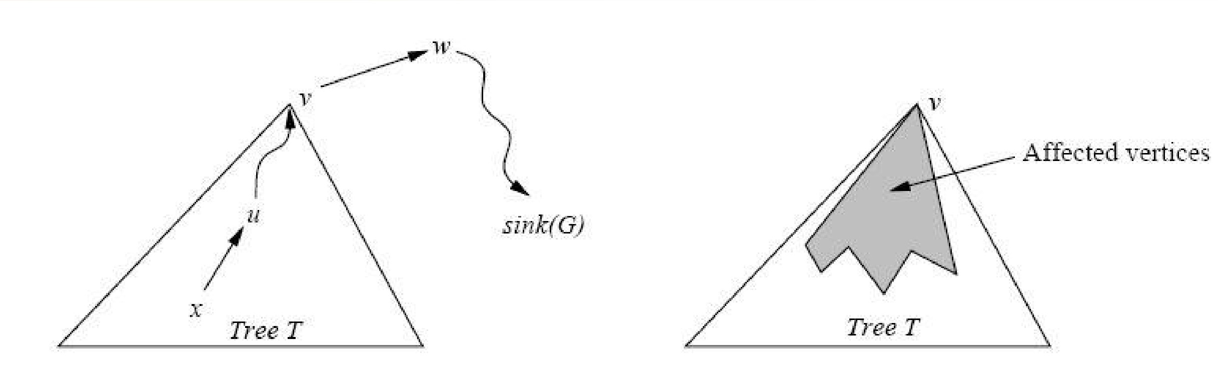


Graph Traversal (BFS)

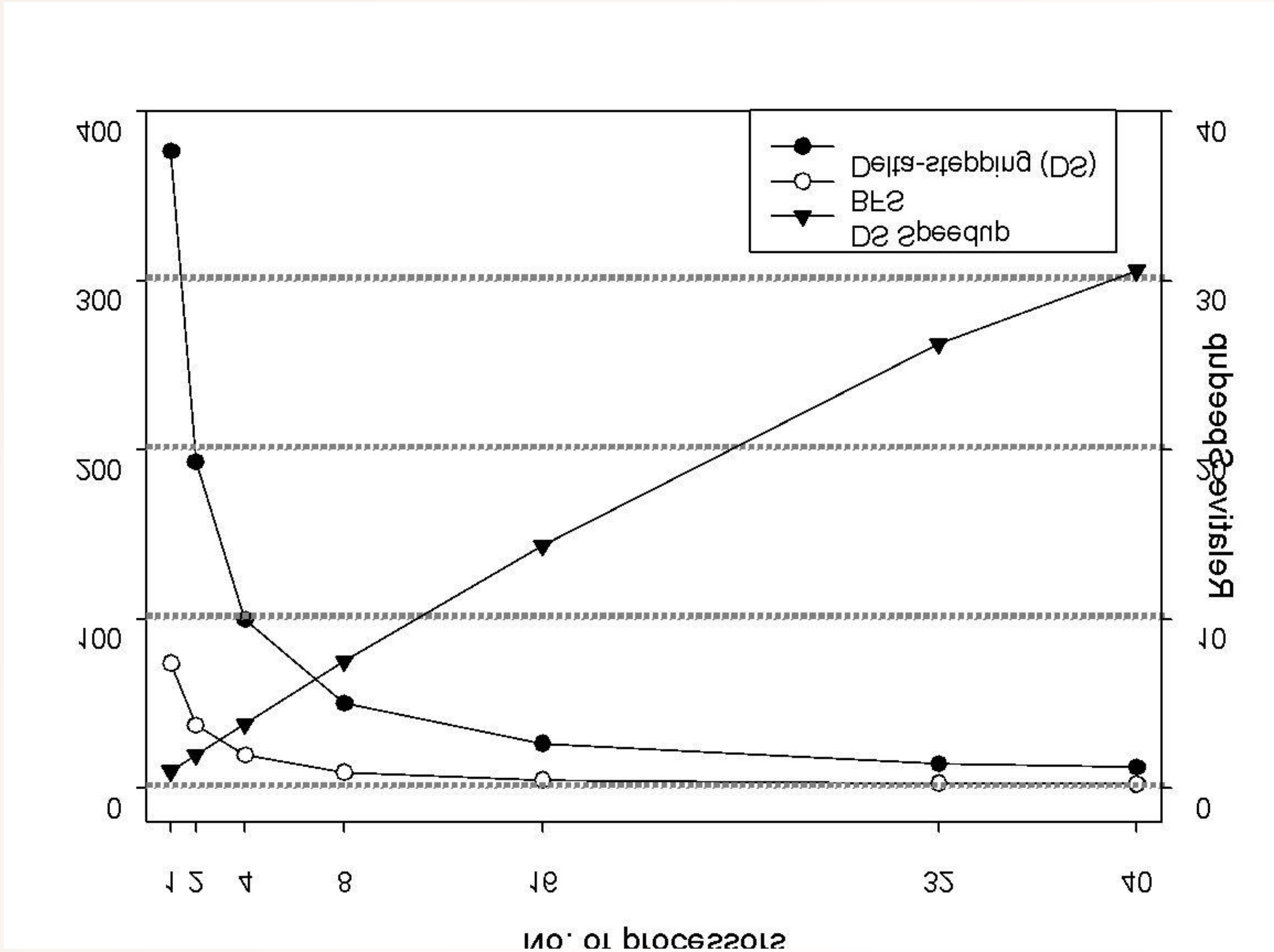
- Level-synchronous graph traversal for low-diameter graphs, each edge in the graph visited only once.
- Fast, efficient implementations on shared memory systems
- Dynamic networks
  - Filter vertices and edges according to time-stamp information, recompute BFS from scratch
  - Dynamic graph algorithms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

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 algorithms, 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]



Parallel Performance: BFS and Shortest Paths



synthetic scale-free network of 2^{28} vertices and 2^{30} edges

Parallel performance results on The Cray MTA-2 (1-40 processors)

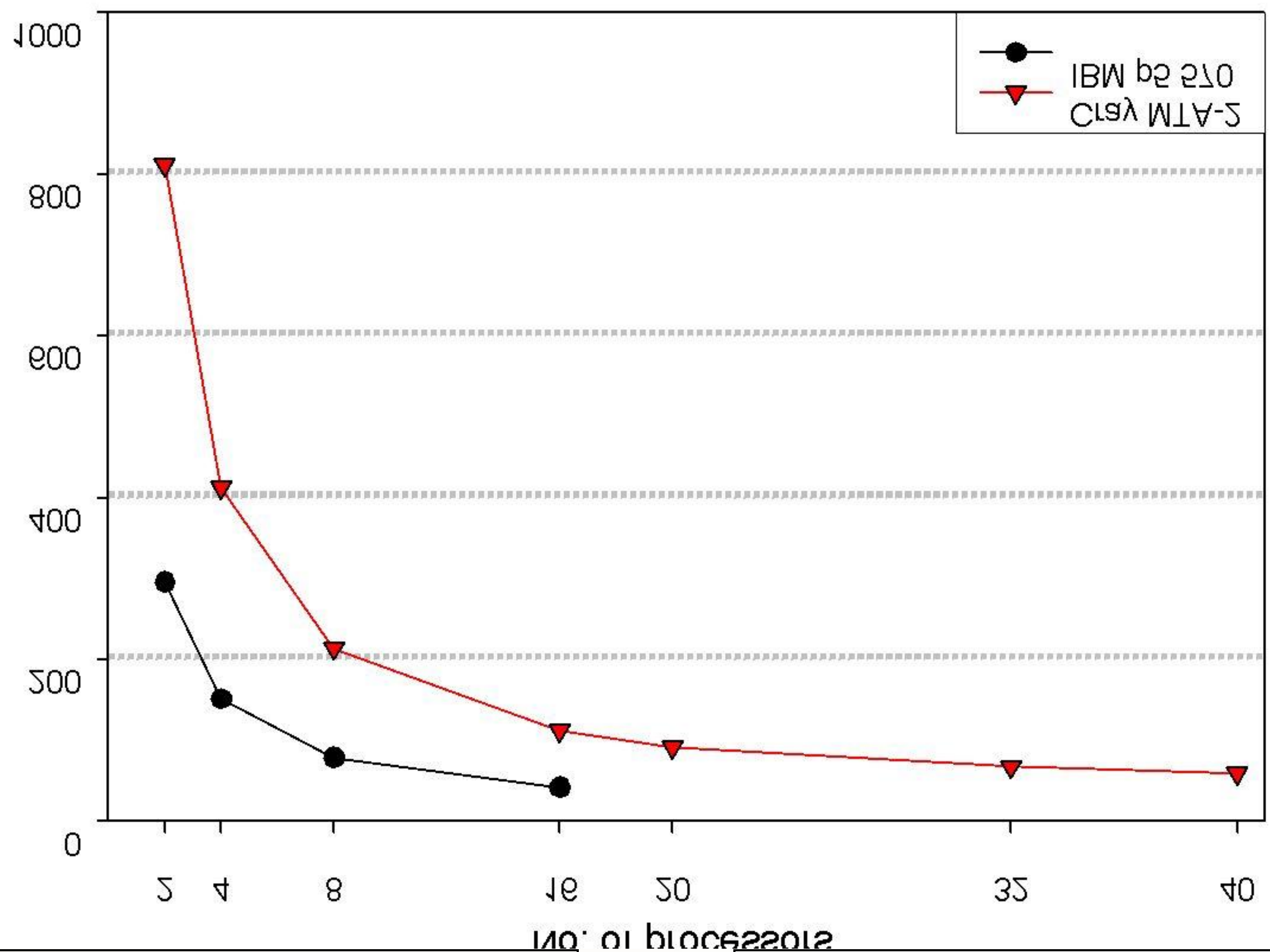
Connectivity

- 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
  - If not, we can use a bidirectional BFS algorithm that requires zero preprocessing time and no additional space

Centrality Analysis

- Determine important vertices and edges in a graph, based on topological characteristics such as degree distributions, paths, flows etc.
- Betweenness is a popular metric
- Extensively uses the BFS (unweighted) and shortest path (weighted) kernels
- Compute-intensive: O(mn) work for unweighted graphs
- We have designed parallel algorithms for exact and approximate betweenness of a given edge/vertex
- We can easily modify the static weighted-graph algorithm for betweenness centrality to consider temporal information

Parallel Performance: Betweenness Centrality



IMDB movie-actor interaction network: 392K vertices and 31.7M edges

Parallel performance results on the Cray MTA-2 and IBM p5 570

Graph partitioning, community detection

- We explore fast partitioning heuristics with modularity as the optimization metric
- Preprocessing kernels
  - connected, bi-connected components
  - network sparsification
- Betweenness centrality-based divisive partitioning techniques
- Current work: adapting existing multi-level and spectral partitioning algorithms for the community identification problem in real-world graphs

Georgia Tech College of Computing
Computational Science and Engineering

Acknowledgment of Support

- National Science Foundation
  - **CSR**: A Framework for Optimizing Scientific Applications (06-14915)
  - **CAREER**: High-Performance Algorithms for Scientific Applications (06-11589; 00-93039)
  - **ITR**: Building the Tree of Life -- A National Resource for Phyloinformatics and Computational Phylogenetics (EF/BIO 03-31654)
  - **ITR/AP:** Reconstructing Complex Evolutionary Histories (01-21377)
  - **DEB** Comparative Chloroplast Genomics: Integrating Computational Methods, Molecular Evolution, and Phylogeny (01-20709)
  - **ITR/AP(DEB):** Computing Optimal Phylogenetic Trees under Genome Rearrangement Metrics (01-13095)
  - **DBI:** Acquisition of a High Performance Shared-Memory Computer for Computational Science and Engineering (04-20513).
- NASA Graduate Student Researcher Program (GSRP) Fellowship for Kamesh Madduri, NASA NP-2005-07-375-HQ
- IBM PERCS / DARPA High Productivity Computing Systems (HPCS)
  - DARPA Contract NBCH30390004
- IBM Shared University Research (SUR) Grant
- Sony-Toshiba-IBM (STI)
- Microsoft Research
- Sun Academic Excellence Grant

Conclusions

- We study data representations and parallel approaches for solving massive interaction network problems
- Applications: Community identification, centrality analysis
- We present SNAP, an open-source parallel library for large-scale network analysis
  - http://www.cc.gatech.edu/~kamesh/SNAP