

Massive graph analysis on HPC systems

Kamesh Madduri

KMadduri@lbl.gov



Talk Outline

- Research challenges in massive graph analysis
 - “CS” problems: algorithms, performance, libraries/frameworks, productivity
- Impediments to high-performance complex network analysis on current systems
- Some results
 - Massive graph analysis on the Cray XMT
 - Graph analysis on cache-based multicore systems

Graph abstractions are pervasive

Sources of massive data: the Internet, Intelligence and surveillance applications, sensor networks, scientific applications, petascale simulations, experimental devices.

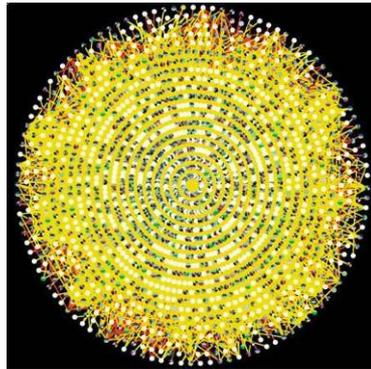
Cosmology

Application: Outlier detection.
Challenges: petascale datasets.
Graph problems: clustering, matching.



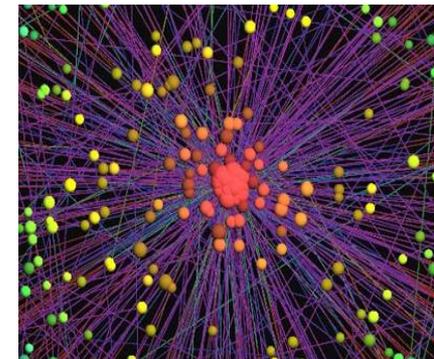
Bioinformatics

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



Social Informatics

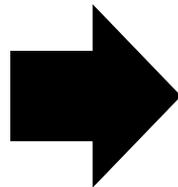
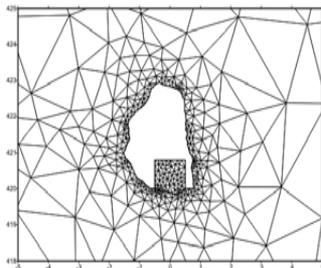
Application: Discover emergent communities, model spread of information.
Challenges: new analytics routines, uncertainty in data.
Graph problems: clustering, shortest paths, flows.



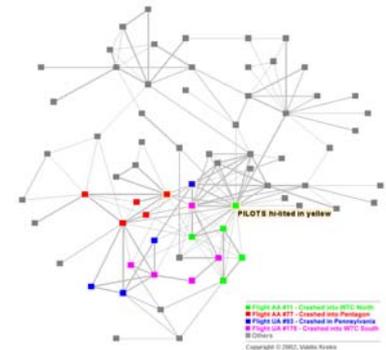
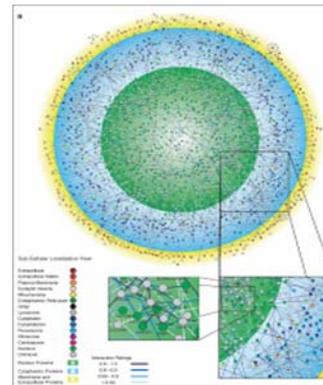
New challenges for analysis: data sizes, data heterogeneity, uncertainty, data quality, and dynamic/temporal nature.

Massive data analytics & Informatics

- Social, collaboration, citation, biological, epidemiological, web graphs, ...
- Informatics networks are **fundamentally different** from graph topologies and computations in scientific computing!



Static networks,
Euclidean topologies



Informatics: dynamic, high-
dimensional, heterogeneous
data

What we'd like to understand about these large networks (Math/CS problems):

- 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, ...

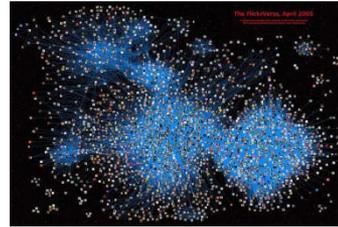
The CS/HPC research problems

- How do we **solve massive graph problems** (statistical analysis, dynamics, community detection, learning) with **current** algorithms/systems/languages/ programming models/libraries?
- How do we get analytics routines to scale for massive datasets?
 - We need new algorithms that exploit graph topology & modern computer architectures
- What are the **right abstractions** for designing portable, high-performance graph algorithms?
 - PRAM-like, BSP-like, MapReduce, matrices/tensors?
- **Application** → **Architecture** mapping for current systems?
 - Multicore servers, commodity clusters, supercomputers, clouds
- What can we do with **accelerators**?
 - GPUs, Cell processor, Netezza and Azul data warehousing appliances
- What are the **languages & programming models** we should be using for high productivity/performance?
 - C+threads, Cilk++, MPI, PGAS languages, Pig/Hive/Sawzall, ...

Software: what the community wants ...

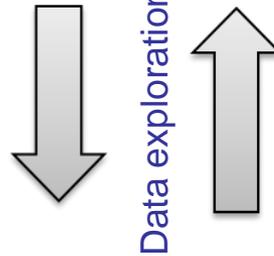


domain expert



massive data

Analysis routines that run everywhere, with **high performance**



Home computers



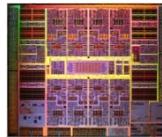
Commodity clusters



Accelerators



Multicore Servers



Massively multithreaded Systems (Cray XMT)



Petascale computers



Graph Software: current status

Home computers



Plethora of solutions, motivated by social network analysis and computational biology research problems. Cannot handle **massive data**.

Representative software: **igraph**, Cytoscape

Commodity clusters



Implementations of **Bulk-synchronous** algorithms; **MapReduce-based** approaches. Performance a concern. Likely not generic enough to process queries on dynamic networks.

Boost Graph library, CGM-lib

Accelerators

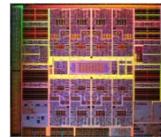


Petascale computers



Impressive performance on synthetic network instances/simple problems. Applicability to complex informatics problems unclear. e.g., recent BFS performance studies

Multicore Servers



SNAP: Tuned, heterogeneous implementations to solve complex graph analysis problems.

C + threads

Can process **networks with billions of vertices and edges**, on high-end multicore servers.

Fastest cache-based multicore implementations of several algorithms.

Massively multithreaded Systems (Cray XMT)



MTGL: Multithreaded graph library based on the “**visitor**” design pattern.

C++ with XMT pragmas

Can also run on **multicore** systems.

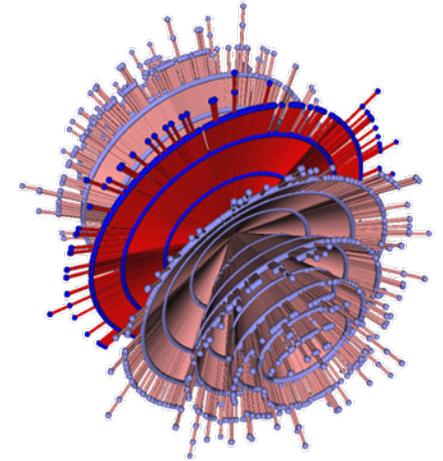
Graph-theoretic problems and algorithms are diverse ...

- Some algorithms have **excellent cache locality**
 - “dense” graphs, Bellman-Ford algorithm for APSP
- Some graph algorithms have a **bounded memory footprint**, or very good spatial locality
 - rich literature on streaming graph algorithms
- **Critical** to consider **graph size and topology** in application to architecture mapping
 - Can achieve high performance on GPUs if the graph + data structures fit in device memory.
 - Reasonably good performance on distributed memory clusters if the graph has low conductance (can be partitioned w/ low edge cut).
- We'll limit our discussion to the harder problems: poor locality, difficult to partition, dynamic, heterogeneous data, ...

Informatics → “Small-world” complex networks

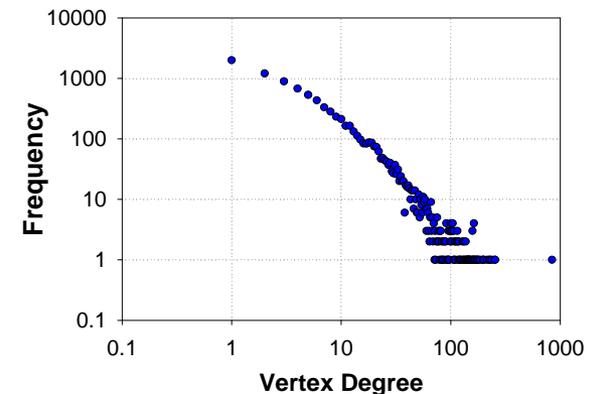
- Low graph diameter.
- Skewed (“power law”) degree distribution of the number of neighbors.
- Sparse: $m = O(n)$.
- Vertices, edges have multiple attributes.

“Six degrees of separation”



“Power law” degree distribution

Human Protein Interaction Network
(18669 proteins, 43568 interactions)

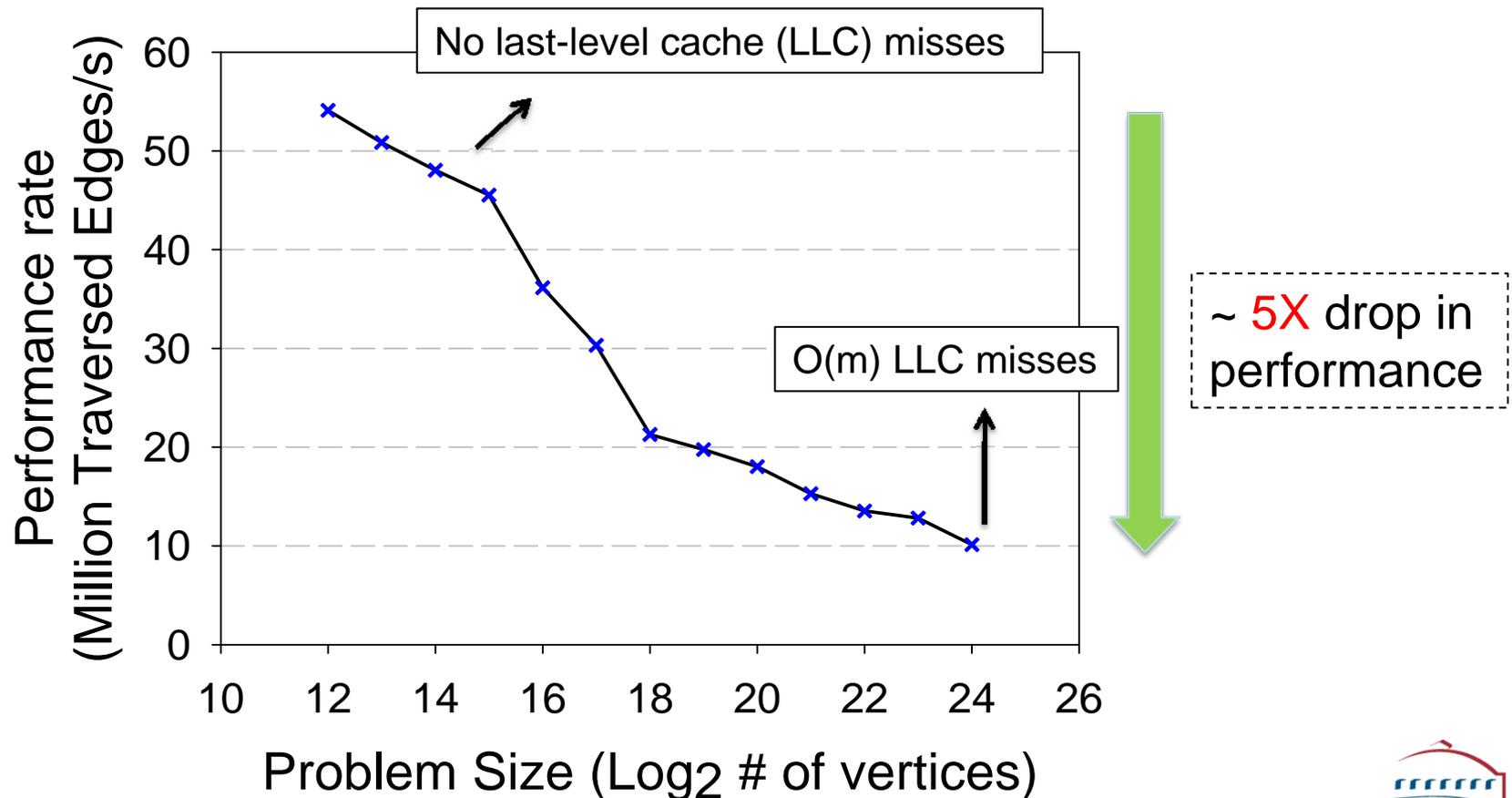


The problems: #1. The locality challenge

“Large memory footprint, low spatial and temporal locality impede performance”

Serial Performance of “approximate betweenness centrality” on a 2.67 GHz Intel Xeon 5560 (12 GB RAM, 8MB L3 cache)

Input: Synthetic R-MAT graphs (# of edges $m = 8n$)



The problems: #2. The parallel scaling challenge

“Classical parallel graph algorithms perform poorly on current parallel systems”

- Parallelization strategies at loggerheads with techniques for enhancing memory locality
- Classical “work-efficient” graph algorithms may not fully exploit new architectural features
 - Increasing complexity of memory hierarchy (x86), DMA support (Cell), wide SIMD, floating point-centric cores (GPUs).
- Tuning implementation to minimize parallel overhead is non-trivial
 - Shared memory: minimizing overhead of locks, barriers.
 - Distributed memory: bounding message buffer sizes, bundling messages, overlapping communication w/ computation.

Talk Outline

- Research challenges in massive graph analysis
 - “CS” problems: algorithms, performance, libraries/frameworks, productivity
- Impediments to high-performance complex network analysis on current systems
- Results
 - Case study: graph traversal-based algorithms
 - Massive graph analysis on the Cray XMT
 - Graph analysis on cache-based multicore systems

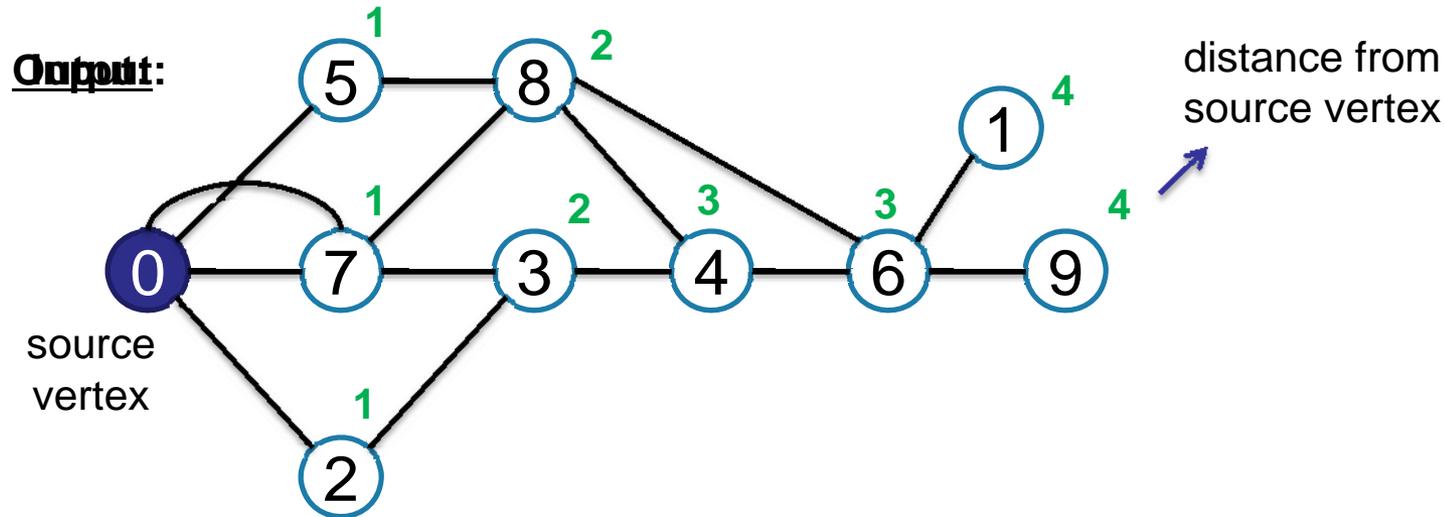
Prior research @ Georgia Tech

- Parallel graph **algorithms** and **efficient implementations** for massive complex network analysis
 - Graph traversal, st-connectivity [[ICPP06a](#)]
 - Shortest paths [[ALENEX07](#), [MTAAP07](#)]
 - Centrality metrics in network analysis [[ICPP06b](#), [WAW07](#)]
 - Community Identification [[IPDPS08](#)]
- Open-source graph **software**: [graphanalysis.org](#), **SNAP**, GTgraph, DIMACS-ch9-shortest_paths
- HPCS SSCA Graph Analysis **benchmark** [[HiPC06](#)]
- **Applications** to systems biology: lethality in the human protein interaction network [[ParCo09](#)]

Current research @ LBNL

- Data structures, algorithms for processing connectivity queries in massive **dynamic** networks [[IPDPS09](#)]
- A lock-free parallel betweenness centrality algorithm for the Cray XMT [[MTAAP09](#)]
- Performance tuning graph traversal on multicore [[SIAM AN09](#)]
- Sequential algorithms: single-source shortest paths with few edge weights [[JODA09](#)], new priority queue for single-source shortest paths [[COCOA09](#)], negative cycle detection [[FAW09](#)]
- Parallel algorithms for
 - **Community detection**: overlapping communities, spectral techniques
 - **Frequent subgraph mining** in temporal networks
- Scaling the SSCA#2 graph analysis benchmark on distributed memory clusters (with UPC)

Graph traversal (BFS) problem definition

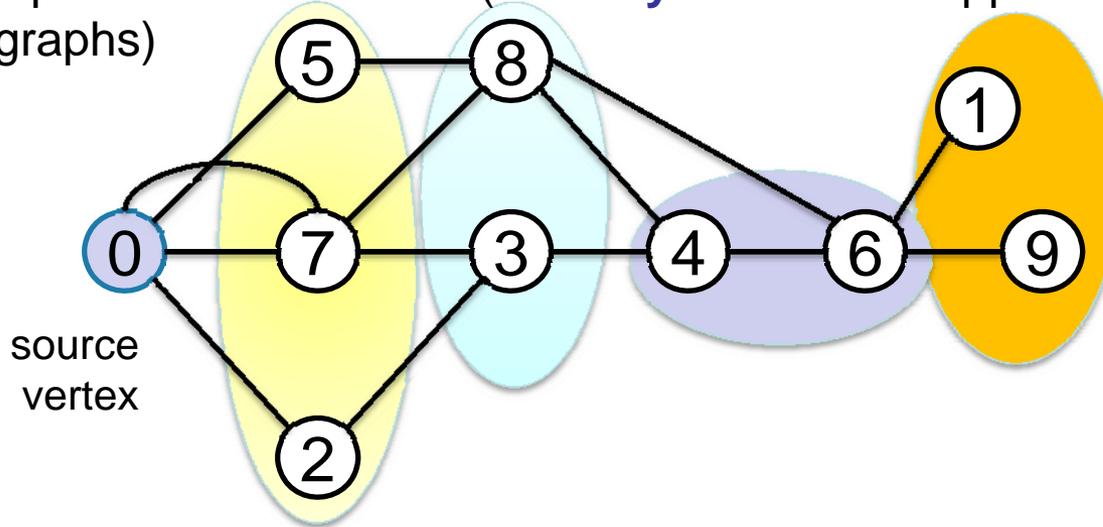


Memory requirements (# of machine words):

- Sparse graph representation: $m+n$
- Stack of visited vertices: n
- Distance array: n

Parallel BFS Strategies

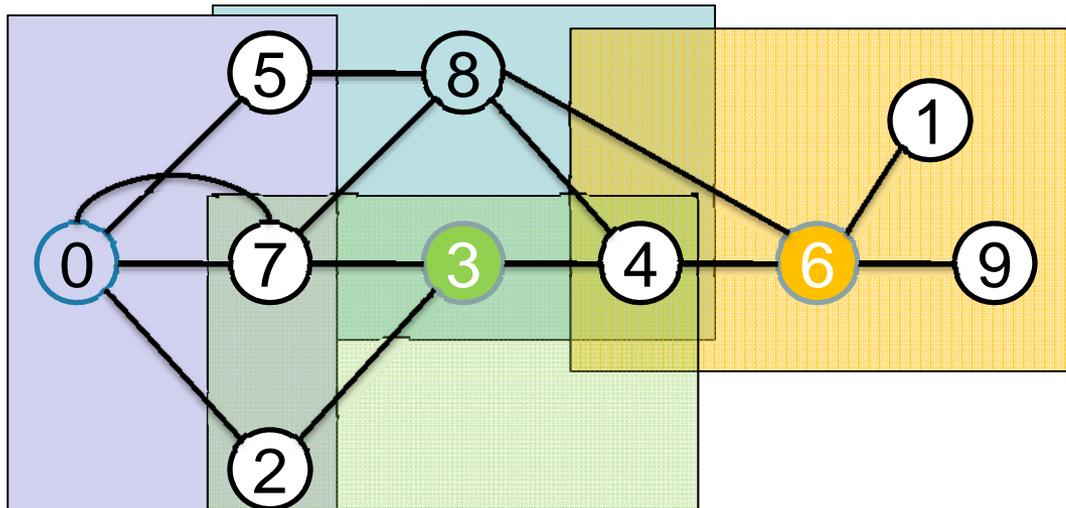
1. Expand current frontier (**level-synchronous** approach, suited for **low diameter** graphs)



source
vertex

- $O(D)$ parallel steps
- Adjacencies of all vertices in current frontier are visited in parallel

2. Stitch multiple concurrent traversals (Ullman-Yannakakis approach, suited for **high-diameter** graphs)



source
vertex

- path-limited searches from “super vertices”
- APSP between “super vertices”

Hiding memory latency

- Ideally, we want to do 1 (or more) memory op/clock cycle/processor.
- However, we have to do with
 - Caches
 - Reduce latency by storing data close to the processor
 - Vectors
 - Amortize latency by fetching N words at a time
 - Parallelism
 - Hide latency by switching tasks
 - **Little's law**: concurrency = bandwidth * latency

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!



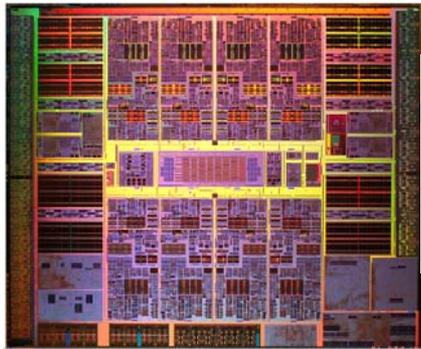
Large-scale Graph Traversal [DIMACS Challenge '07]

Problem	Graph	Result	Comments
Multithreaded BFS	Random graph, 256M vertices, 1B edges	2.3 sec (40p) 73.9 sec (1p) MTA-2	Processes all low-diameter graph families
External Memory BFS	Random graph, 256M vertices, 1B edges	8.9 hrs (3.2 GHz Xeon)	State-of-the-art external memory BFS
Multithreaded SSSP	Random graph, 256M vertices, 1B edges	11.96 sec (40p) MTA-2	Works well for all low-diameter graph families
Parallel Dijkstra	Random graph, 240M vertices, 1.2B edges	180 sec , 96p 2.0GHz cluster	Best known distributed-memory SSSP implementation for large-scale graphs

Cray MTA-2 is the predecessor to the XMT.

Shared memory multicore/SMP servers

Sun “Niagara” Multicore Servers



Cache-based multicore servers with chip multithreading

Sun Fire T2000: UltraSparc T1 (*Niagara1*)

1 socket x 8 cores x 4 threads per core; **16 GB** RAM
3 MB shared L2 cache; 900 MHz processor

Sun Fire T5120: UltraSparc T2 (*Niagara2*)

1 socket x 8 cores x 8 threads per core; **32 GB** RAM
4 MB shared L2 cache; 1167 MHz processor

IBM Power 570 SMP Server

- High-bandwidth shared memory multiprocessor system.
- 16 Power5 1.9 GHz processors
- **256 GB** physical memory
- 32KB L1 data cache, 2MB L2, 32MB L3 per processor.
- 8-way superscalar, 2-way SMT on each core

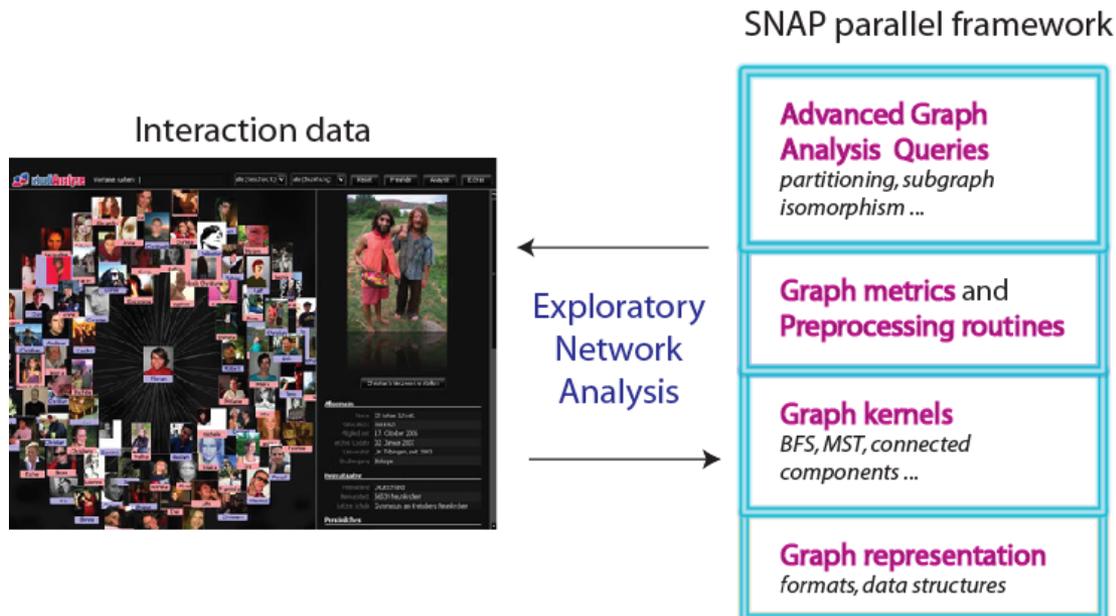


Test Networks

- Synthetic networks generated using the [R-MAT \[CZF04\]](#) graph model, based on matrix Kronecker products.
- Representative of several real-world graph families.
- Challenging instances for parallelization due to unbalanced degree distribution.

snap-graph: Small-world Network Analysis and Partitioning

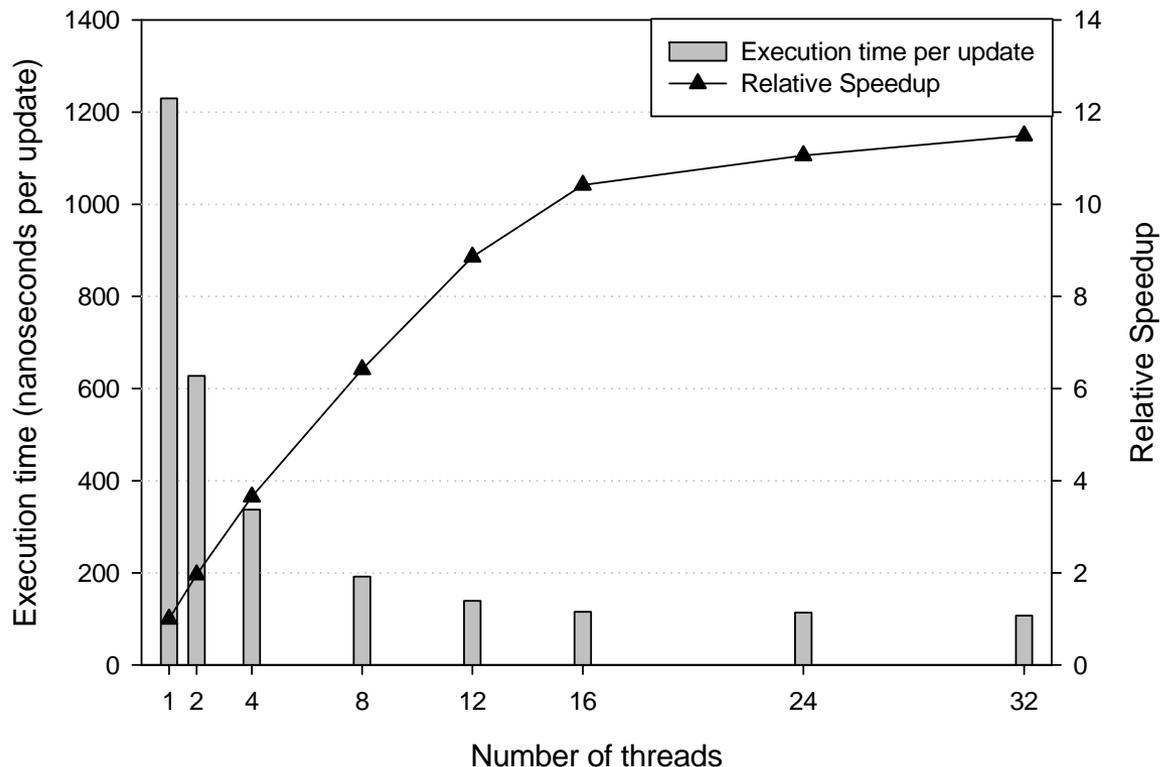
- **SNAP**: Parallel framework for small-world complex graph analysis
- **10-100x faster** than existing approaches.
 - Parallelism, algorithm engineering, exploiting graph topology.
- Can process graphs with **billions** of vertices and edges.
- Open-source: snap-graph.sourceforge.net



Optimizations/heuristics for real-world graphs

- **Preprocessing kernels** (connected components, biconnected components, sparsification) significantly reduce computation time.
 - e.g. a high number of isolated and degree-1 vertices
 - store BFS/shortest path trees from high degree vertices and reuse them
 - Typically **3-5X** performance improvement
- **Exploit small-world network properties** (low graph diameter)
 - Load balancing in the **level-synchronous parallel BFS** algorithm
 - SNAP data structures are **optimized for unbalanced degree** distributions

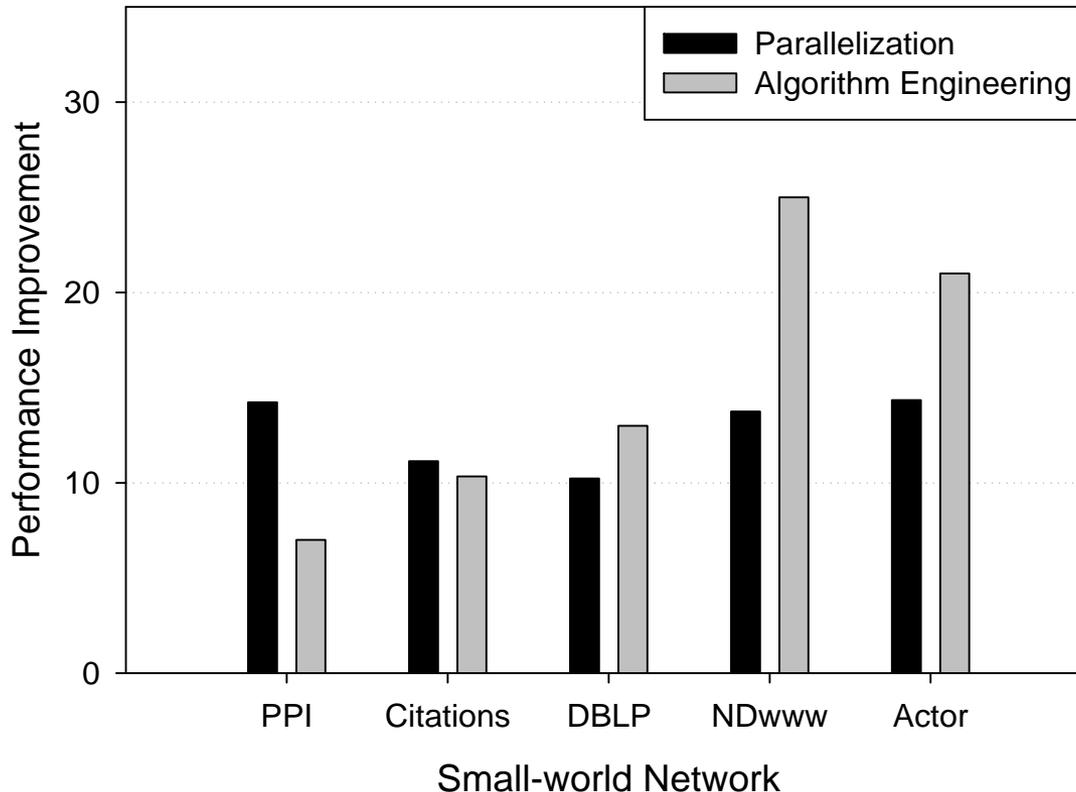
Compact graph representations for dynamic network analysis [IPDPS09]



- New graph representations for dynamically evolving small-world networks in SNAP.
- Support fast, parallel structural updates to low-diameter scale-free and small-world graphs.

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

Faster Community Identification Algorithms: Performance Improvement over the Girvan-Newman approach [IPDPS08]

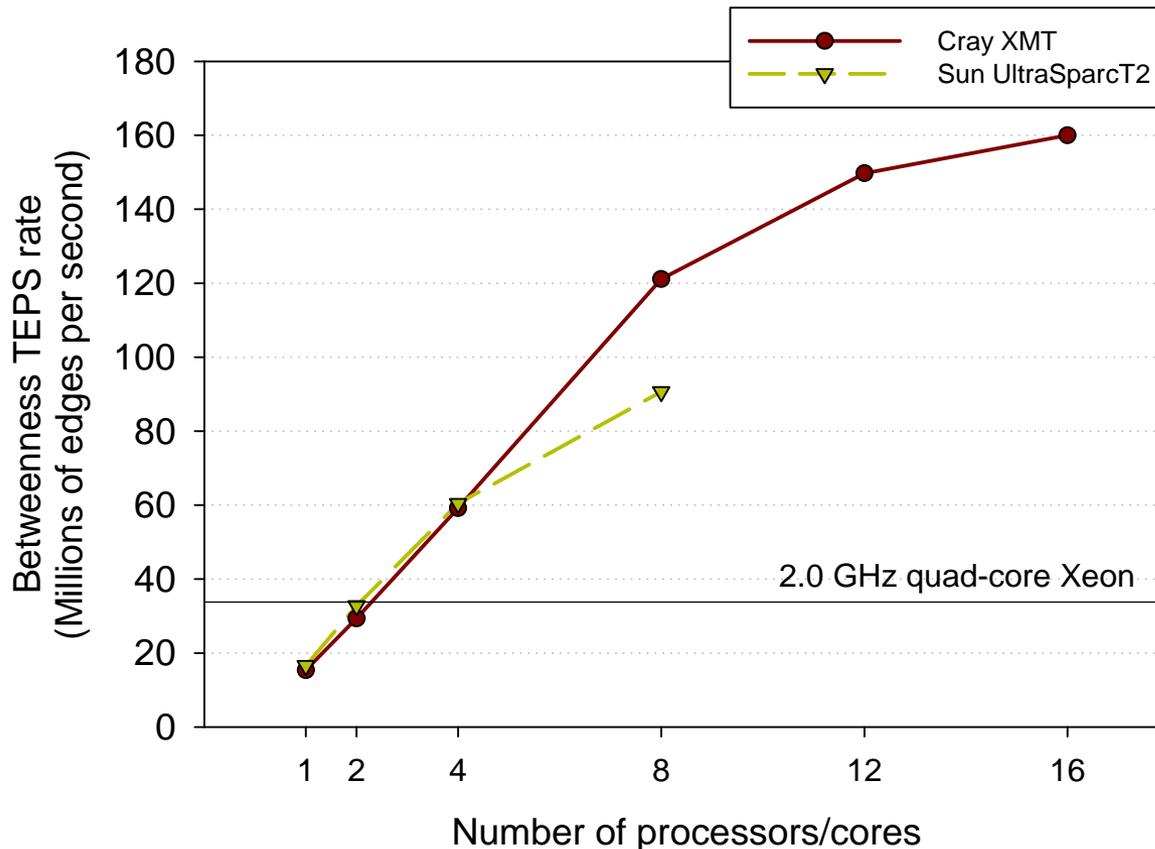


- Speedup from Algorithm Engineering (approximate BC) and parallelization (Sun Fire T2000) are **multiplicative!**
- **100-300X** overall performance improvement over Girvan-Newman approach

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

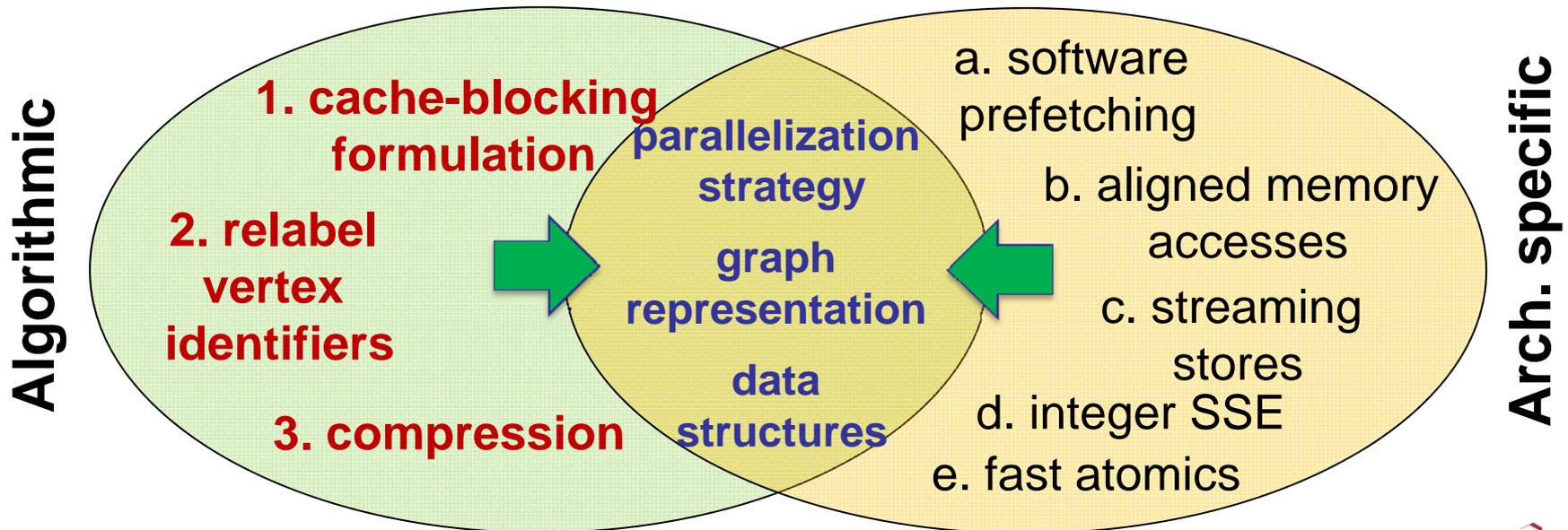
Cray XMT vs. Cache-based multicore

- Approximate “betweenness centrality” on SSCA#2 network, SCALE 24 (16.77 million vertices and 134.21 million edges.)



How about x86 cache-based multicore?

- Preliminary research on speeding up graph-traversal based algorithms on emerging **x86 cache-based multicore** systems.
- Case study: a closer look at parallel Breadth-First Search (BFS) on a single-socket Intel Nehalem system.
- Adapting several optimization strategies from LBL/UC Berkeley scientific computing multicore auto-tuning research:



Tuning BFS on x86 multicore: hairy!

1. **Software prefetching** on the Intel Core i7 (supports 32 loads and 20 stores in flight)
 - Speculative loads of **index array** and **adjacencies of frontier vertices** will reduce compulsory cache misses.
 - Hardware prefetcher doesn't help, disable it.
2. **Aligning adjacency lists** to optimize memory accesses
 - 16-byte aligned loads and stores are faster.
 - Alignment helps reduce cache misses due to fragmentation
 - 16-byte aligned **non-temporal stores** (during creation of new frontier) are fast.
3. **SIMD SSE integer intrinsics** to process “high-degree vertex” adjacencies.
4. Fast atomics (BFS is lock-free w/ low contention, and **CAS-based intrinsics** have very low overhead)
 - Pipelined atomics in the near future
5. **Hugepage** support (significant TLB miss reduction)
6. NUMA-aware memory allocation exploiting first-touch policy

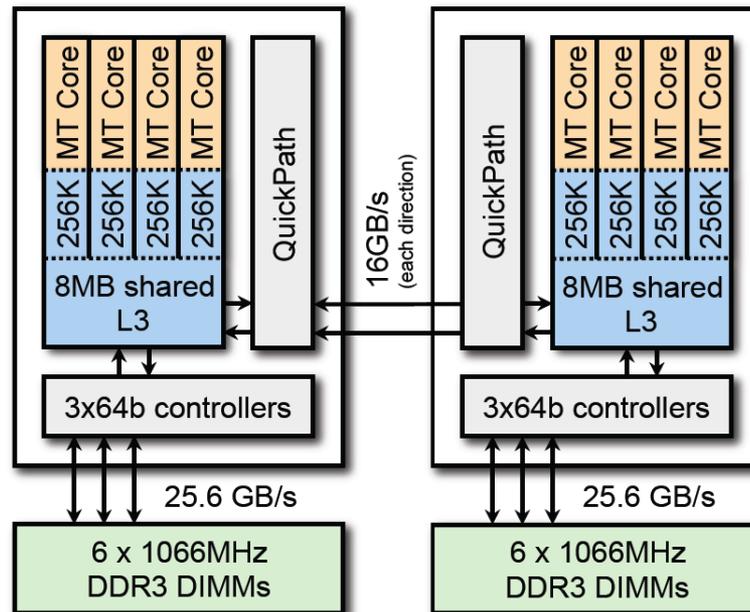
x86 Parallel BFS: Experimental Setup

Network	n	m	Max. out-degree	% of vertices w/ out-degree 0,1,2
Orkut	3.07M	223M	32K	5
LiveJournal	5.28M	77.4M	9K	40
Flickr	1.86M	22.6M	26K	73
Youtube	1.15M	4.94M	28K	76
R-MAT	8M-64M	8n	$n^{0.6}$	

Intel Xeon 5560 (Core i7, "Nehalem")

- 2 sockets x 4 cores x 2-way SMT
- 12 GB DRAM, 8 MB shared L3
- 51.2 GBytes/sec peak bandwidth
- 2.66 GHz proc.

Performance averaged over 10 different source vertices, 3 runs each.

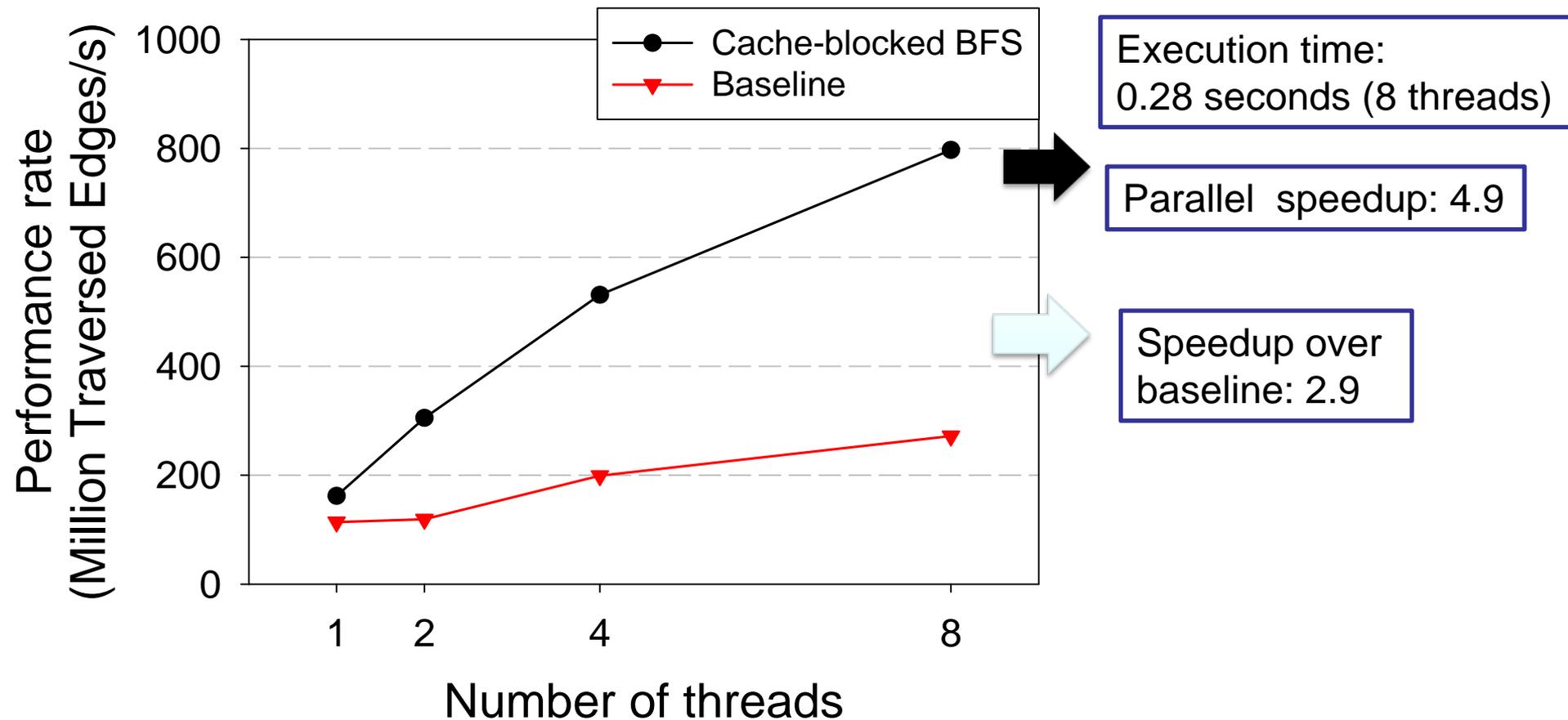


Impact of optimization strategies

Optimization	Generality	Impact*	Tuning required?
(Preproc.) Sort adjacency lists	High	--	No
(Preproc.) Permute vertex labels	Medium	--	Yes
Preproc. + binning frontier vertices + cache blocking	M	2.5x	Yes
Lock-free parallelization	M	2.0x	No
Low-degree vertex filtering	Low	1.3x	No
Software Prefetching	M	1.10x	Yes
Aligning adjacencies, streaming stores	M	1.15x	No
Fast atomic intrinsics	H	2.2x	No

* Optimization speedup (performance on 4 cores) w.r.t baseline parallel approach, on a synthetic R-MAT graph ($n=2^{23}, m=2^{26}$)

Parallel performance (Orkut graph)



Graph: 3.07 million vertices, 220 million edges
Single socket of Intel Xeon 5560 (Core i7)

How about using my petascale Blue Gene-P/XT5 cluster?

- All problems can be solved on large-scale distributed memory clusters; **most** can be solved **efficiently**. How efficiently can be solve graph problems?
- In theory and practice, **bleak projections** for graph **partitioning**:
 - Erdos, Graham, Szemerédi '75: (almost all) sparse graph classes lack good separators.
 - Lang '06, Leskovec et al. '08: spectral partitioning on power-law networks produces unbalanced cuts.
- In practice, poor performance reported for distributed memory implementations (in comparison to SMP/multicore servers/XMT with lot of shared memory)
 - Prior work on traversal-based algorithms for massive graphs (~ billions of entities) has led to observations such as **4 Cray MTA-2 processors = 32K BlueGene/L cores!**
- Similar challenges with Clouds/Roadrunner/GPU clusters
- Worth exploring: **Hierarchical UPC + threads with graph compression + replication** (instead of partitioning)

Summary

- Massive data and graph abstractions are everywhere
- Exciting Math/CS and CS/HPC research problems to study

Research Contributions

- The SNAP graph analysis framework for cache-based multicore systems.
- Multithreaded algorithms for graph analysis on the Cray XMT.
- Performance tuning graph algorithms on x86 multicore systems.

Collaborators

- SDM @ LBL, FTG @ LBL, BEBOP @ UC Berkeley
- David A. Bader (Georgia Tech)
- Jonathan Berry, Bruce Hendrickson (Sandia National Laboratories)
- John Feo, Daniel Chavarria (Pacific Northwest National Laboratories)
- Guojing Cong (IBM Research)
- K. Subramani (West Virginia University)

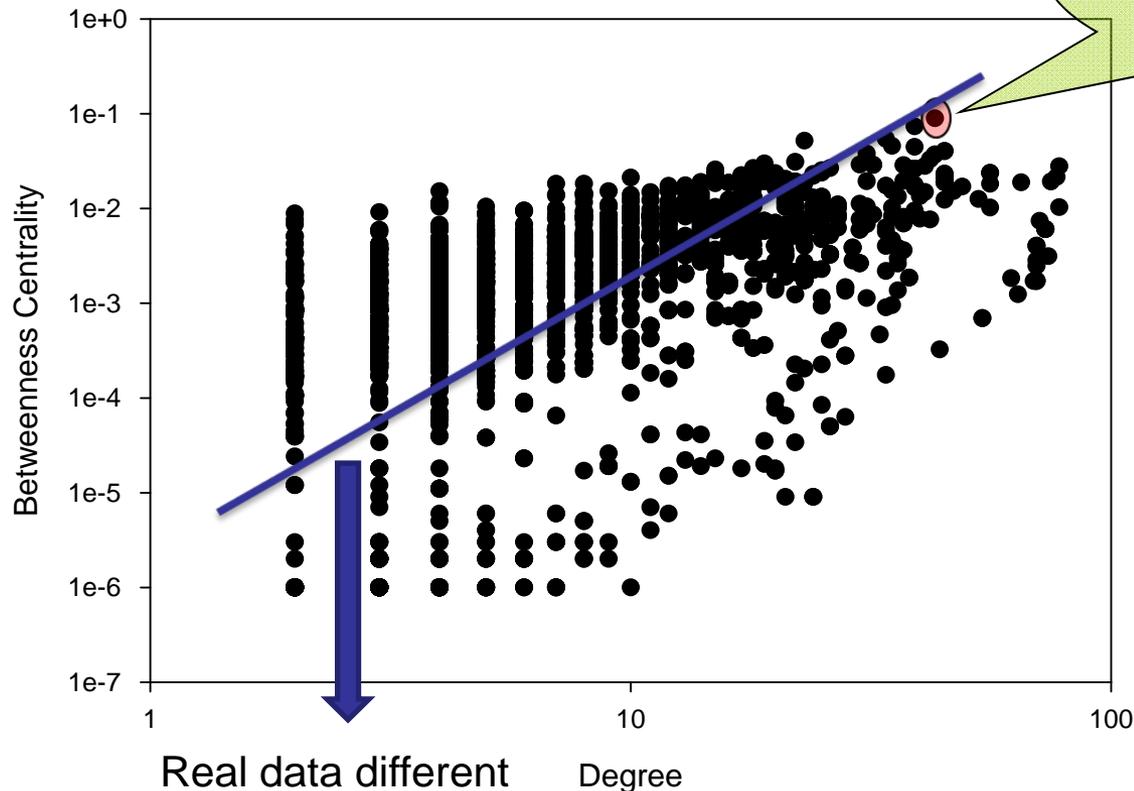
Acknowledgments

- PNNL CASS-MT Center for access to the Cray XMT.
- Cray Inc., for access to their guest XMT system.
- Par Lab @ UC Berkeley for access to the Millennium cluster systems.
- Research supported in part by DOE Office of Science under contract number DE-AC02-05CH11231.

Backup Slides

Centrality Analysis applied to Protein Interaction Networks (PINs) leads to interesting insight!

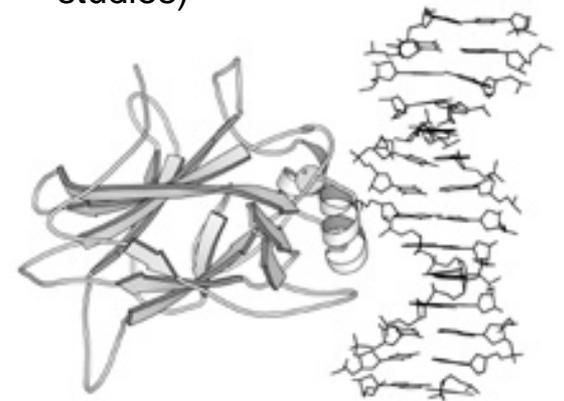
Human Genome core protein interactions
Degree vs. Betweenness Centrality



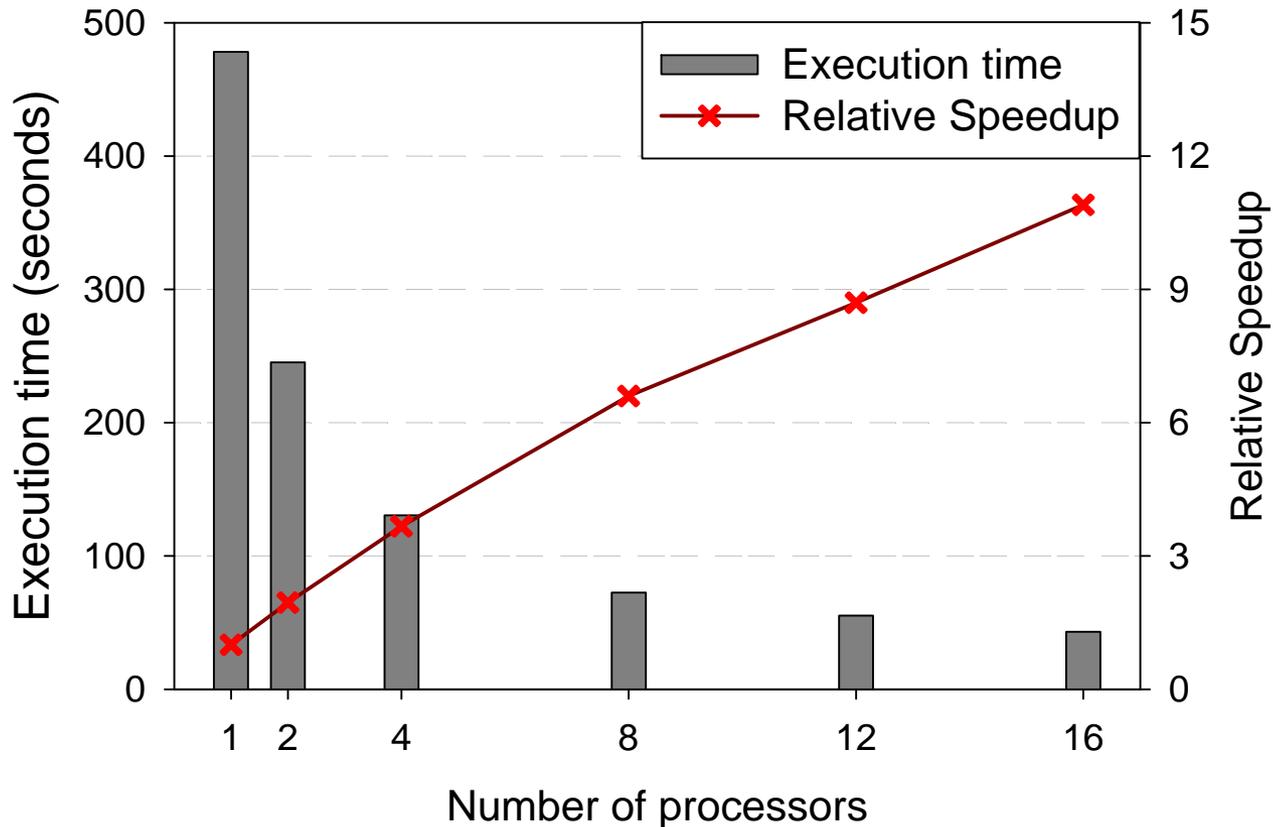
Real data different
from modeled trend!

43 interactions
Protein Ensembl ID
ENSG00000145332.2
Kelch-like protein 8

(commonly occurring protein in
breast cancer drug research
studies)



Parallel Performance on the IBM Power 570



Graph traversal
w/ edge updates

RMAT network with 500 million vertices,
4 billion edges.