Large-scale Graph Analysis

Kamesh Madduri

Computational Research Division

Lawrence Berkeley National Laboratory

KMadduri@lbl.gov madduri.org

Discovery 2015: HPC and Cloud Computing Workshop

June 17, 2011

Talk Outline

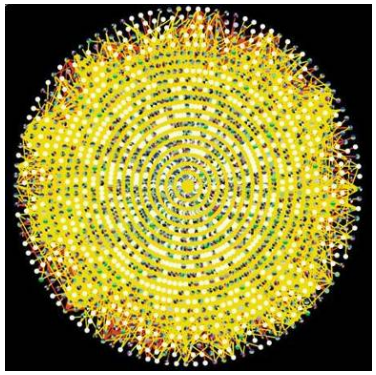
- Large-scale graph analytics: Introduction and motivating examples
- Overview of parallel graph analysis algorithms and software
- Application case studies
  - Community Identification in social networks
  - RDF data analysis using compressed bitmap indexes

Large-scale data analysis

- Graph abstractions are very useful to analyze complex data sets.
- Sources of data: petascale simulations, experimental devices, the Internet, sensor networks
- Challenges: data size, heterogeneity, uncertainty, data quality

Astrophysics: massive datasets, temporal variations

Bioinformatics: data quality, heterogeneity



Social Informatics: new analytics challenges, data uncertainty

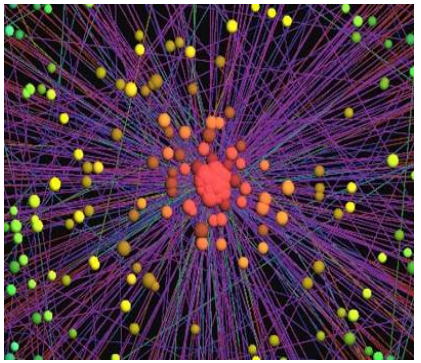


Image sources: (1) http://physics.nmt.edu/images/astro/hst_starfield.jpg (2,3) www.visualComplexity.com

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

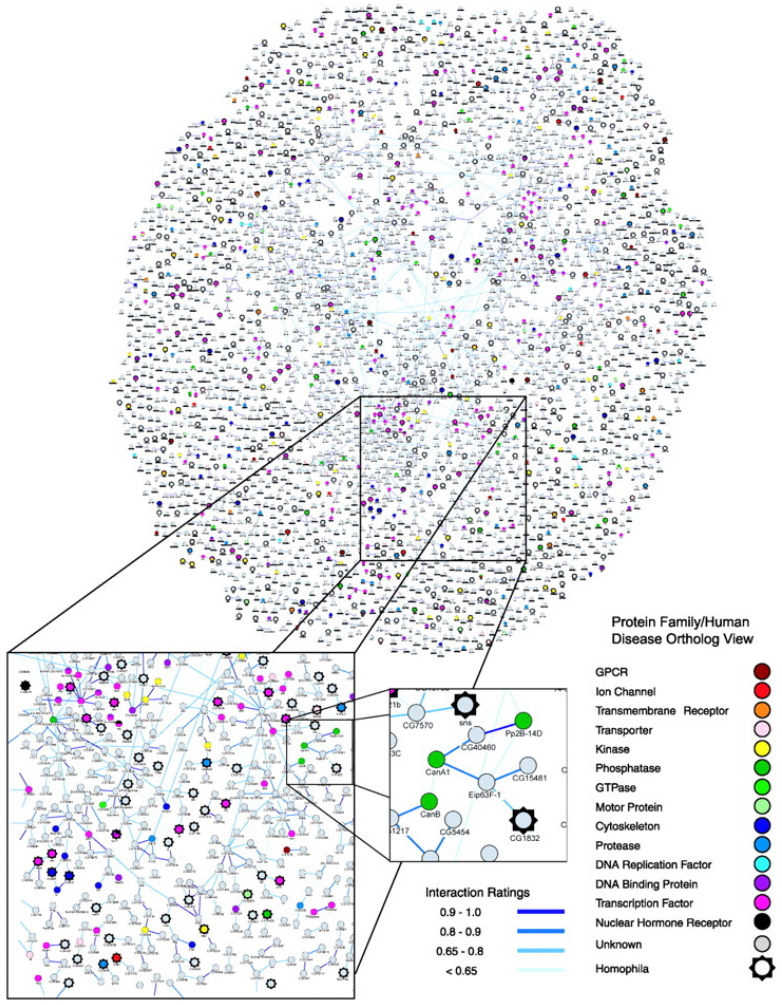


Image Source: Giot et al., "A Protein Interaction Map of *Drosophila melanogaster*", *Science 302*, 1722-1736, 2003.

Graph –theoretic problems in social networks

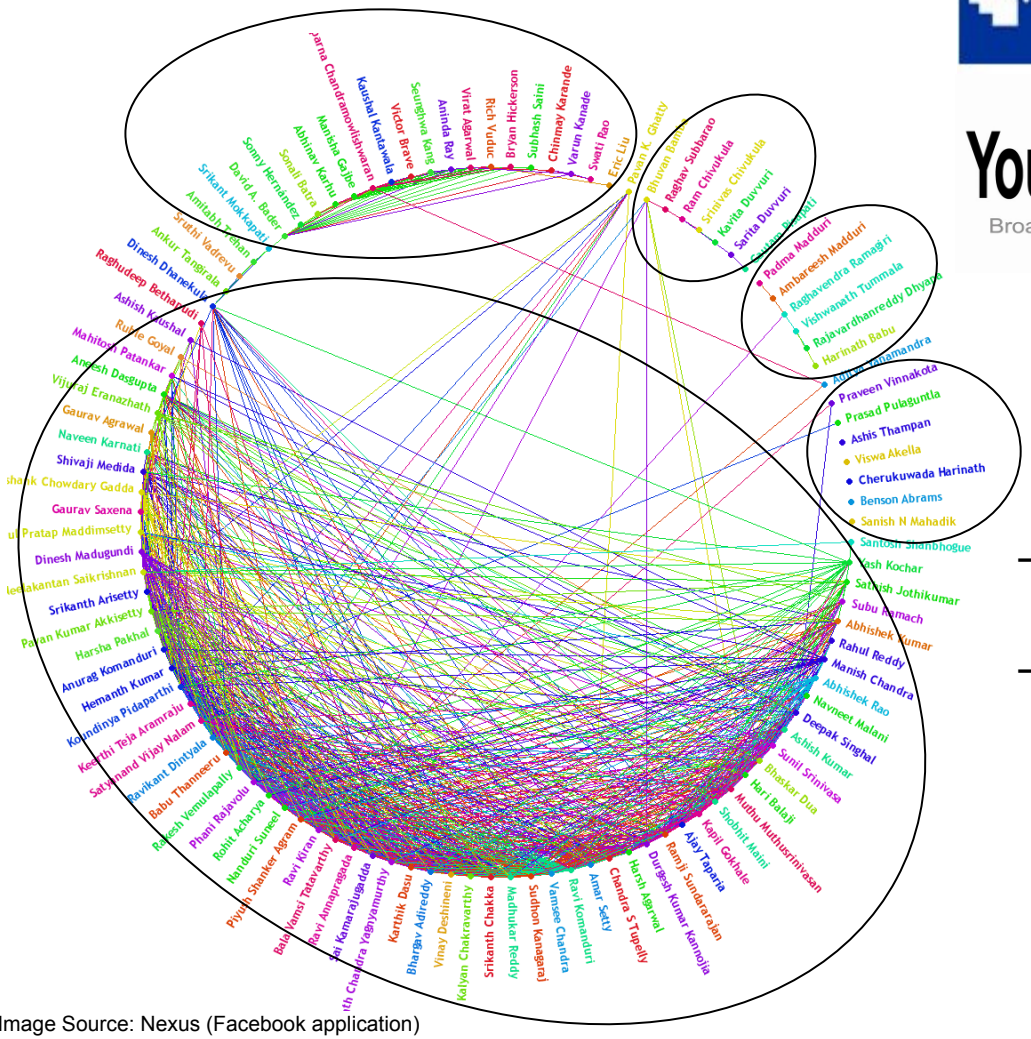


Image Source: Nexus (Facebook application)

myspace.com a place for friends

facebook

YouTube Broadcast Yourself

flickr BETA

LinkedIn

xanga.com THE WEBLOG COMMUNITY

- Targeted advertising: clustering and centrality
- Studying the spread of information

Network Analysis for Intelligence and Surveillance

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

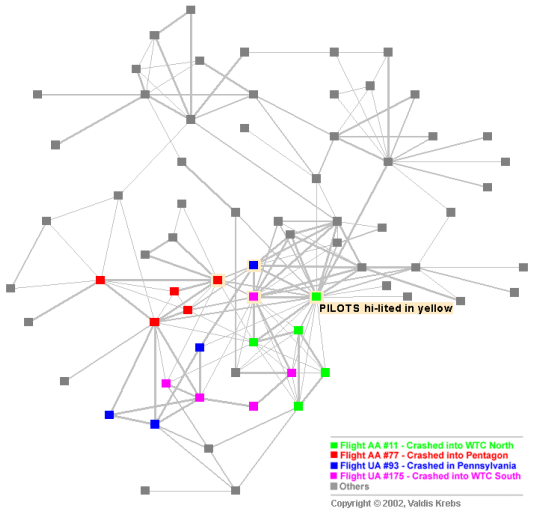


Image Source: http://www.orgnet.com/hijackers.html

- A global view of entities is often more insightful
- Detect anomalous activities by exact/approximate subgraph isomorphism.

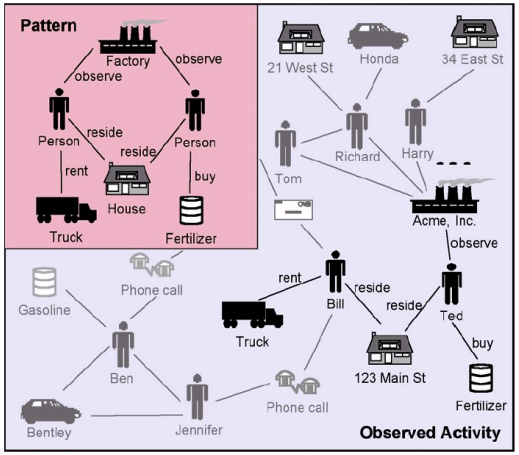
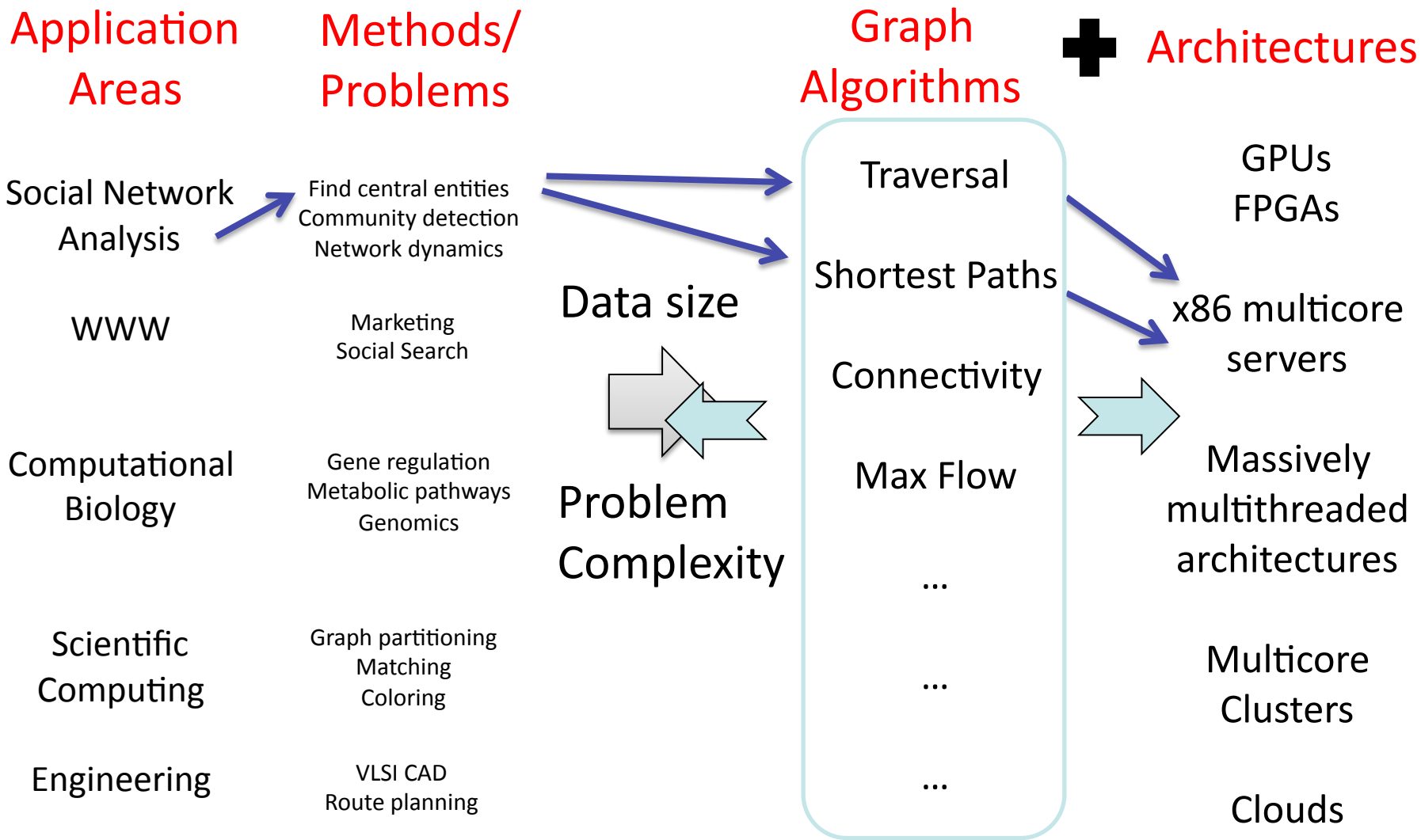


Image Source: T. Coffman, S. Greenblatt, S. Marcus, Graph-based technologies for intelligence analysis, CACM, 47 (3, March 2004): pp 45-47

Research in Parallel Graph Algorithms



Characterizing Graph-theoretic computations

Input data

Problem: Find *

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

Graph kernel

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

…..

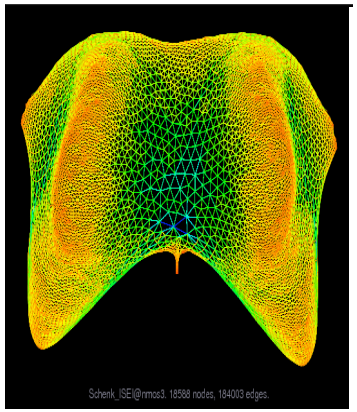
Factors that influence choice of algorithm

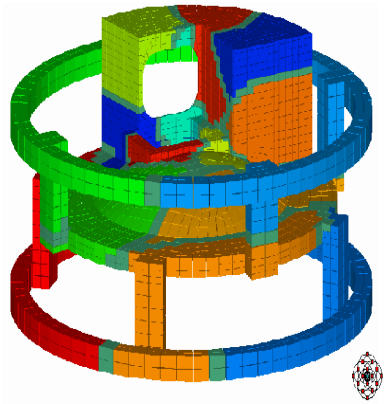
- graph sparsity (m/n ratio)
- static/dynamic nature
- weighted/unweighted, weight distribution
- vertex degree distribution
- directed/undirected
- simple/multi/hyper graph
- problem size
- granularity of computation at nodes/edges
- domain-specific characteristics

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

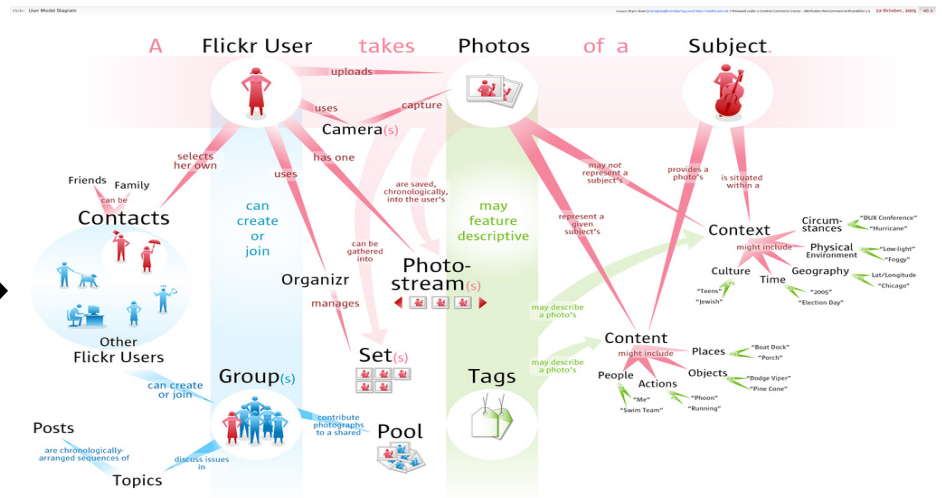
Massive data analytics & Informatics

- Information networks: social, collaborative, citation, biological, epidemiological networks, web crawls, …
- These are fundamentally different from graph topologies and computations in scientific computing!





static networks, 2D/3D topologies



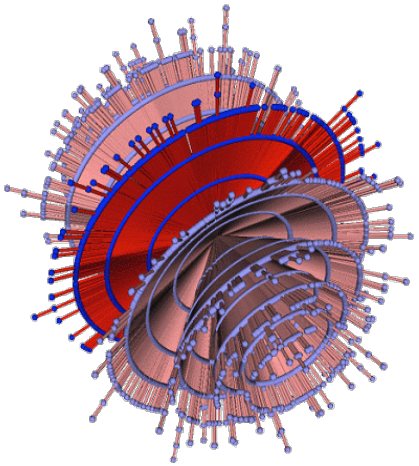
Informatics: dynamic, high-dimensional, heterogeneous data

Image sources: www.visualComplexity.com; Yifan Hu, "A gallery of large graphs"

Informatics → "Small-world" complex networks

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

"Six degrees of separation"



"Power law" degree distribution

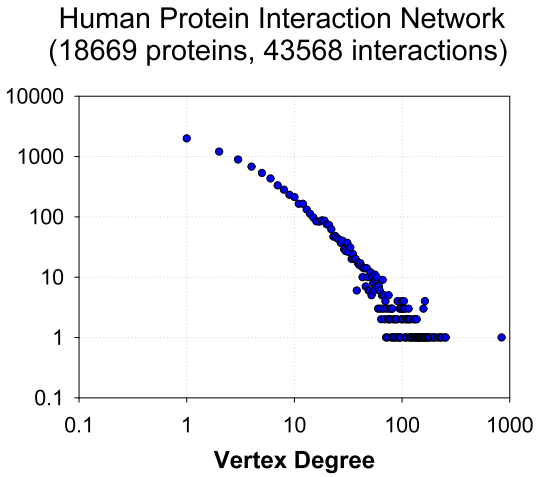
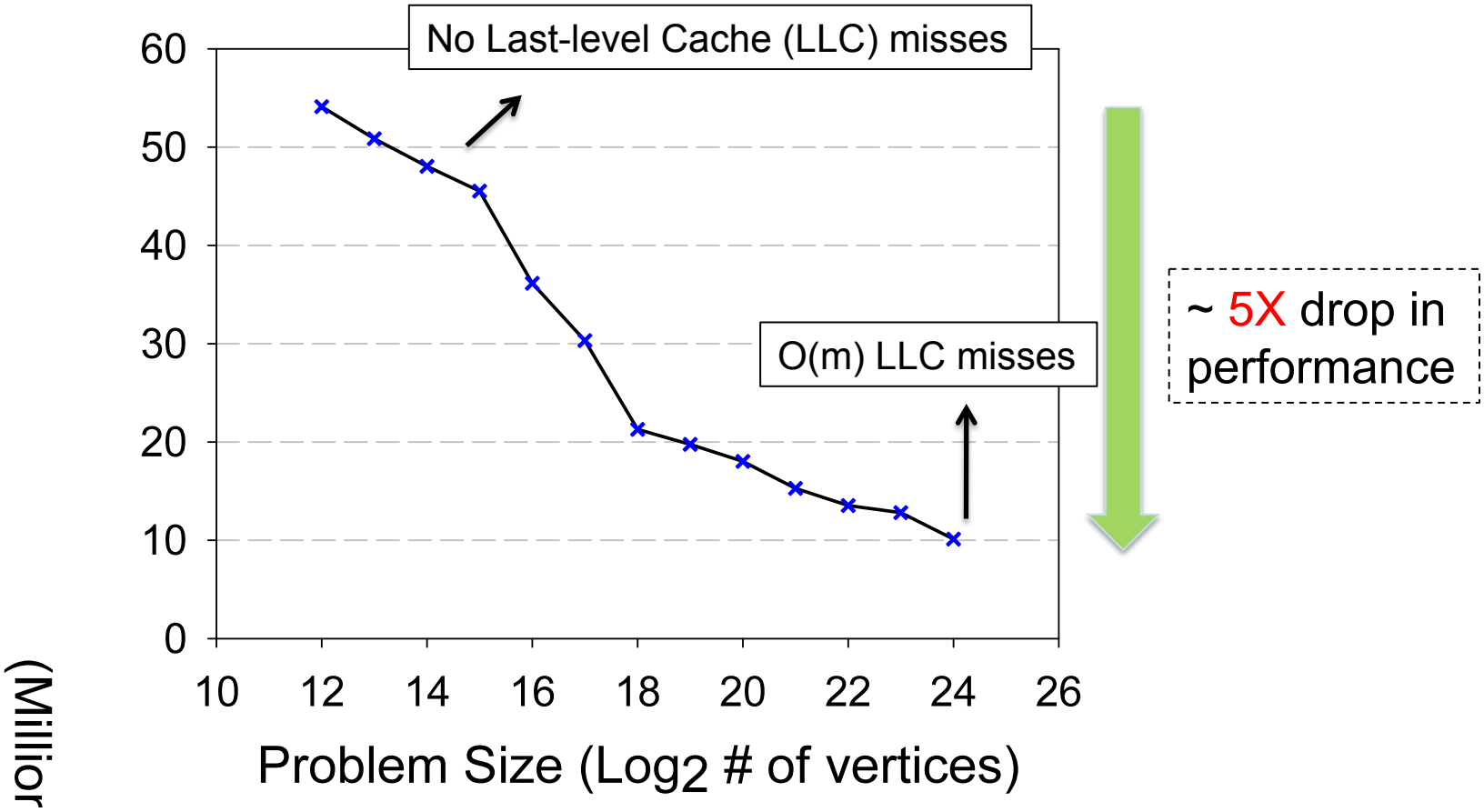


Image source: Seokhee Hong

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 parallel scaling challenge: "Classical parallel graph algorithms perform poorly on current parallel systems"

- Graph topology assumptions in classical algorithms do not match real-world datasets
- 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, processor heterogeneity, wide SIMD.
- 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.

SNAP: Small-world Network Analysis and Partitioning

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

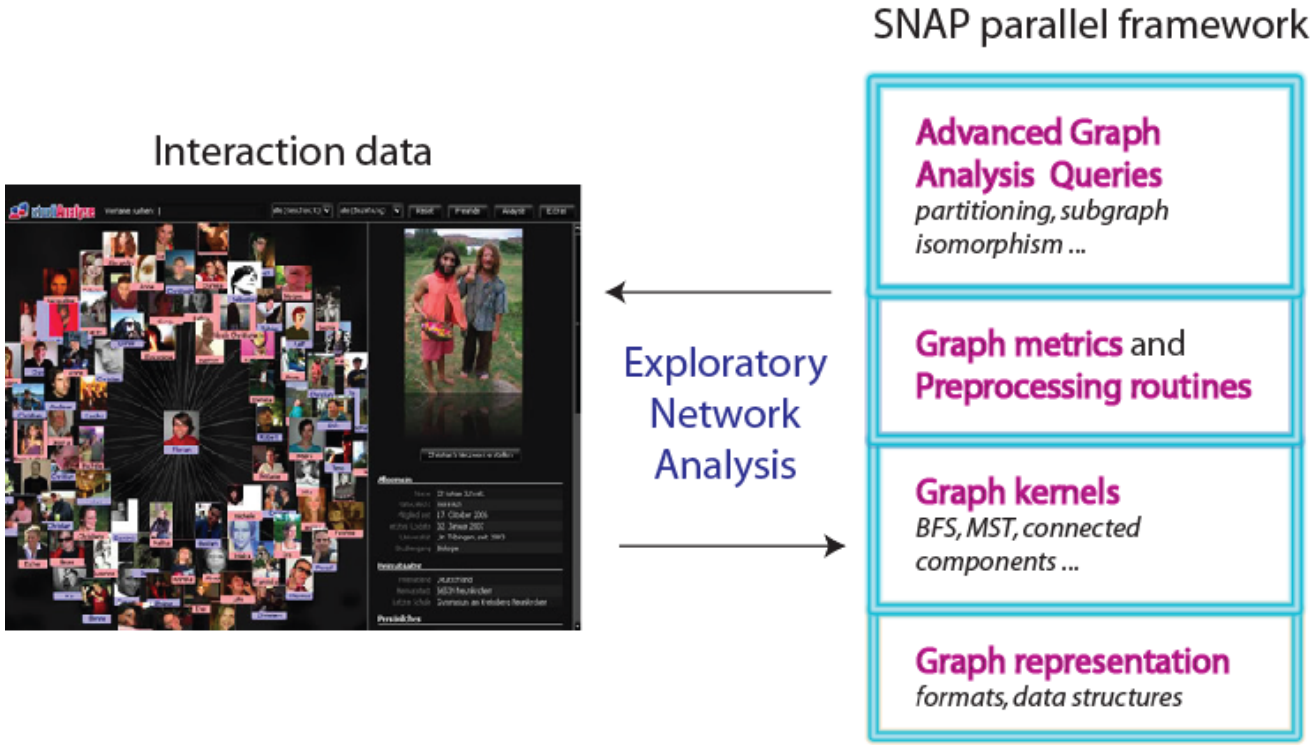


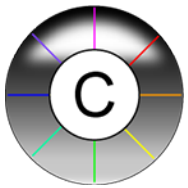
Image Source: visualcomplexity.com

SNAP Optimizations for real-world graphs

- Preprocessing kernels (connected components, biconnected components, sparsification) significantly reduce computation time.
  - ex. 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

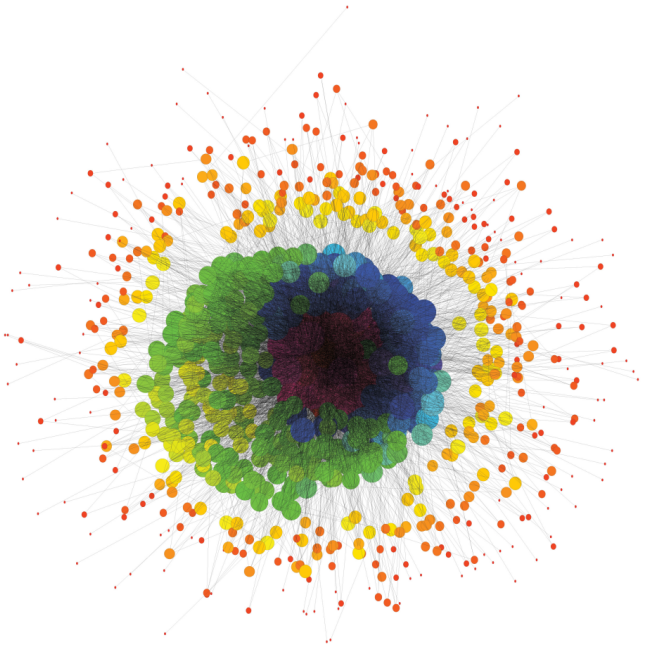
Libraries and Frameworks for graph analysis

- Boost Graph Library
  - C++, graph interface and components are generic
- JUNG
  - Java-based, graph algorithms + visualization engine
- igraph
  - C library with R and Python interfaces
- MultiThreaded Graph Library (MTGL)
  - Boost Graph library-like, for multithreaded architectures
- …
- Network Workbench
  - GUI for analysis, workflow
- Cytoscape
  - Biological network analysis, user-contributed plug-ins,
- …



Graph 500 "Search" Benchmark (graph500.org)

- BFS (from a single vertex) on a static, undirected R-MAT network with average vertex degree 16.
- Evaluation criteria: largest problem size that can be solved on a system, minimum execution time.
- Reference MPI, shared memory implementations provided.



- NERSC Franklin system is ranked #2 on Nov 2010 list.
  - BFS using 500 nodes of Franklin
- Graph 500 June 2011 list submissions
  - NERSC Hopper, 25 GTEPS, SCALE 37, 1800 nodes
  - NERSC Franklin, 16 GTEPS, SCALE 36, 4000 nodes

A. Buluc, K. Madduri, Proc. SC 2011

Talk Outline

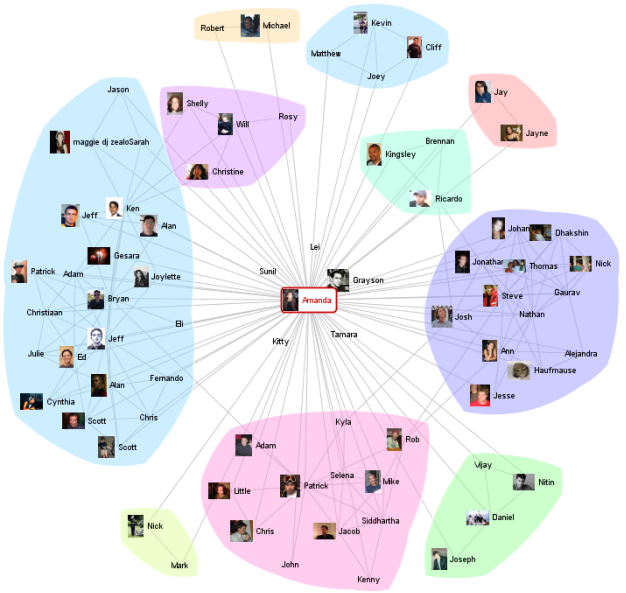
- Large-scale graph analytics: Introduction and motivating examples

- Designing parallel graph analysis algorithms and software

- Application case studies
  - Community Identification in social networks
  - RDF data analysis using compressed bitmap indexes

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.



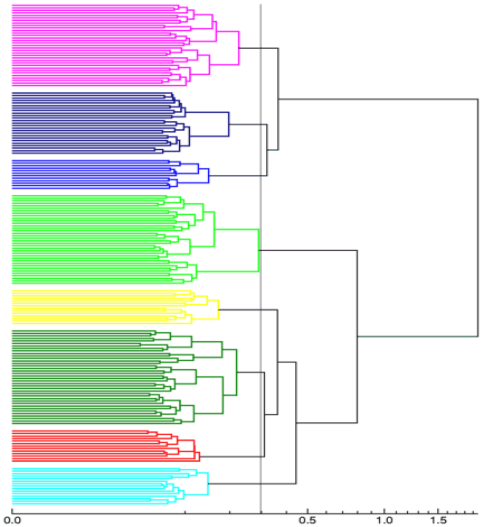


Image Source: visualcomplexity.com

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

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

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

$$\delta(C_i, C_j) = 1 \text{ if } C_i = C_j,$$

$$0 \text{ otherwise.}$$

- 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

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

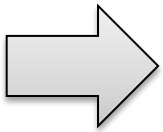
$$e_{ij} = 1 \text{ if } \langle i, j\rangle \in E$$

$$\delta(C_i, C_j) = 1 \text{ if } C_i = C_j,$$

$$0 \text{ otherwise.}$$

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.

Bader and Madduri, “SNAP”, IPDPS 2008.

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.

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

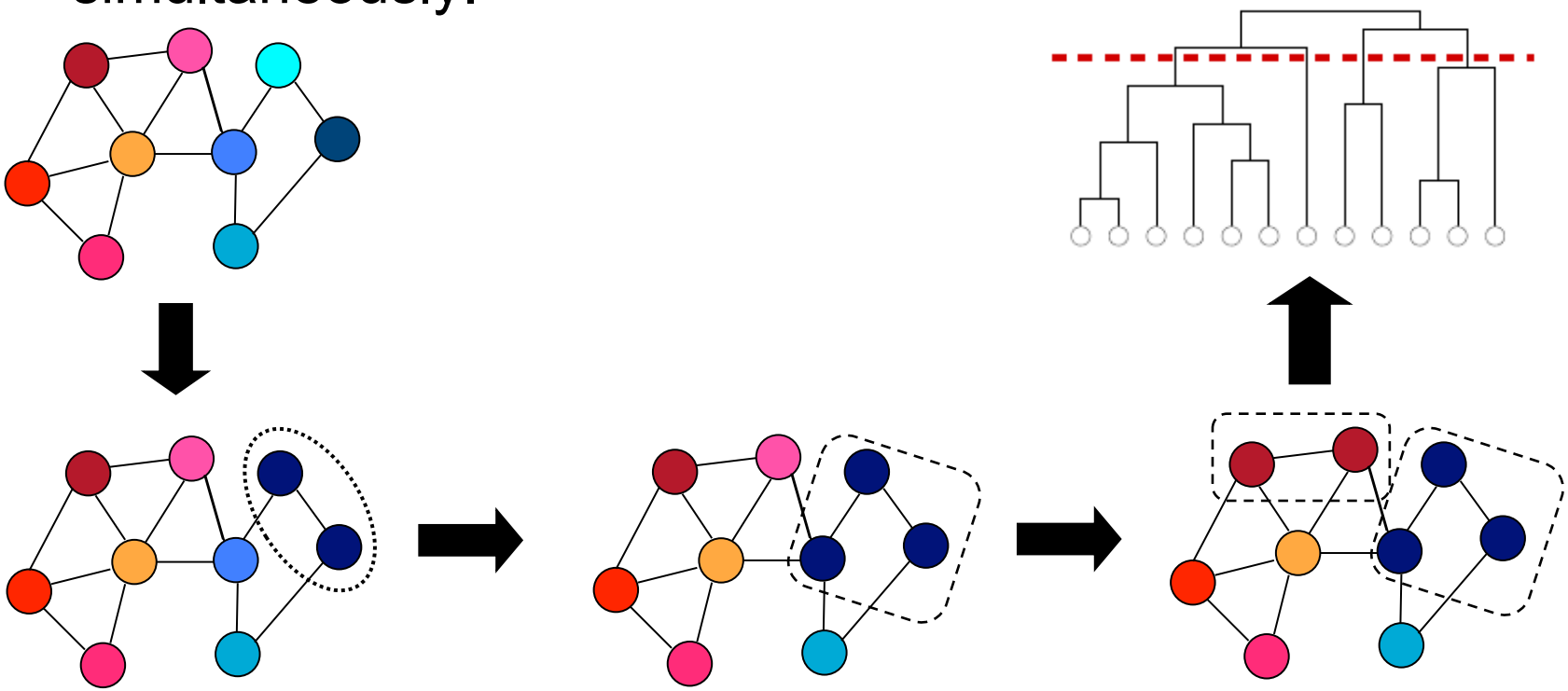
    - Centrality computation is the compute-intensive step, parallelize it.
  - Spectral: apply recursive spectral bisection on the "modularity matrix" B, whose elements are defined as $B_{ij} = A_{ij} - d_i d_j / 2m$. Modularity can be expressed in terms of B as:

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

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

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.

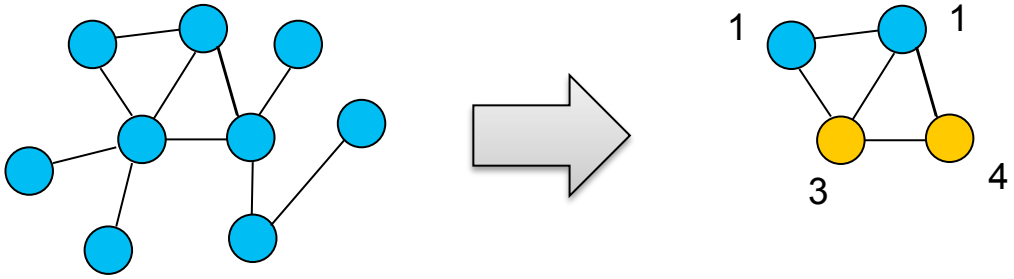


Other Community Identification Approaches

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

Engineering a hybrid parallel community identification algorithm

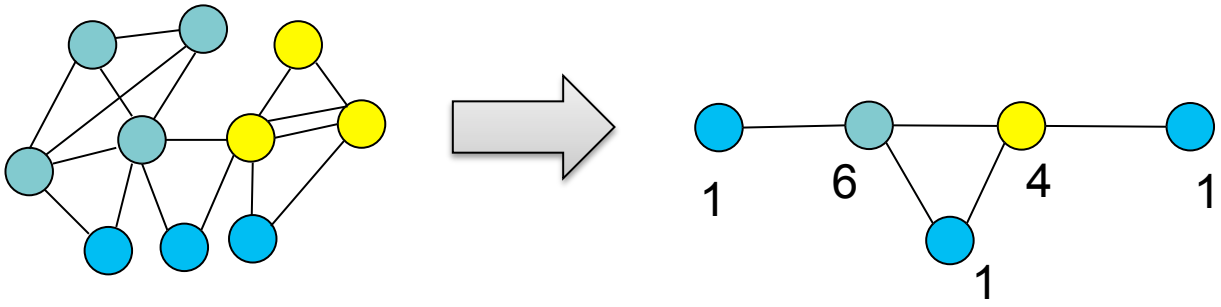
- How would a memory-efficient, near linear-work greedy approach perform on real data?
- Helpful preprocessing steps
  - 2-Core reduction of the graph
    - High-percentage of degree-1 vertices in networks with exponential and power-law degree distributions.



  - 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.

Real-world data

Assembled a collection for algorithm performance analysis, from some of the largest publicly-available network data.

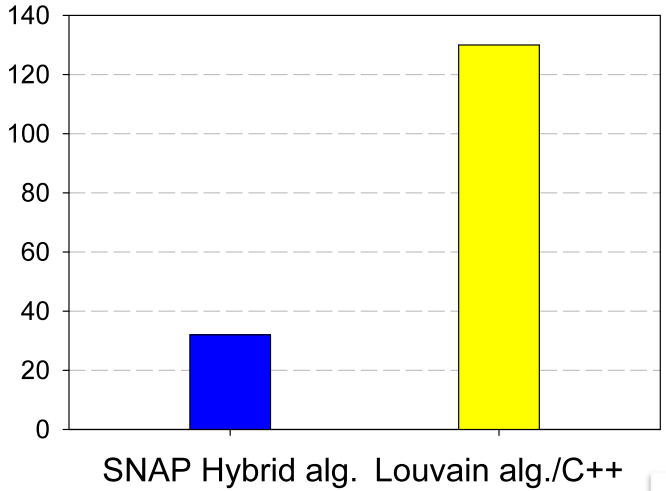
Name	# vertices	# edges	Type
Amazon-2003	473.30 K	3.50 M	co-purchaser
eu-2005	862.00 K	19.23 M	www
Flickr	1.86 M	22.60 M	social
wiki-Talk	2.40 M	5.02 M	collab
orkut	3.07 M	223.00 M	social
cit-Patents	3.77 M	16.50 M	cite
Livejournal	5.28 M	77.40 M	social
uk-2002	18.50 M	198.10 M	www
USA-road	23.90 M	29.00 M	Transp.
webbase-2001	118.14 M	1.02 B	www

SNAP vs. Other Implementations (serial performance)

Webbase-2001
(n=118M, m=1.02B)

Largest network analyzed in prior papers.

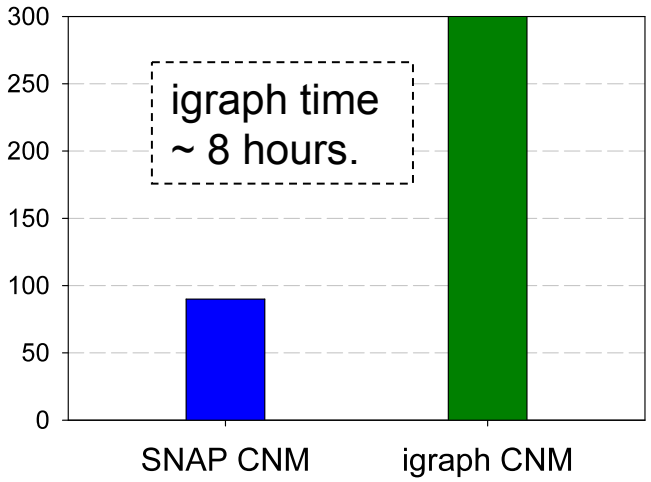
SNAP **4x** faster! Requires 12 GB memory vs. 30 GB for Louvain algorithm.



Amazon-2003
(n=473K, m=3.5M)

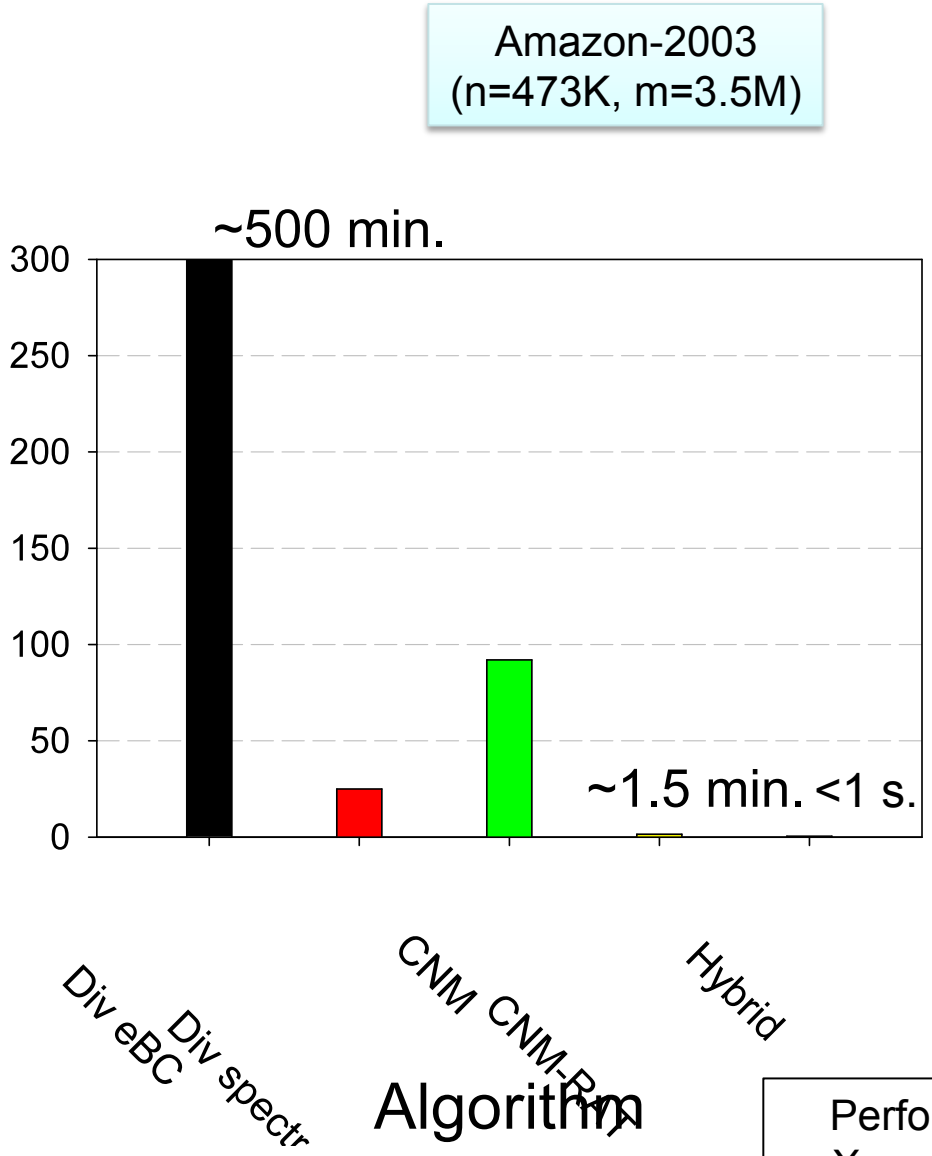
igraph: C library for network analysis.

SNAP requires 0.5 GB memory vs. 12 GB+ for igraph CNM implementation!



Results on a Intel Xeon 5560 ("Nehalem") system
- 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.

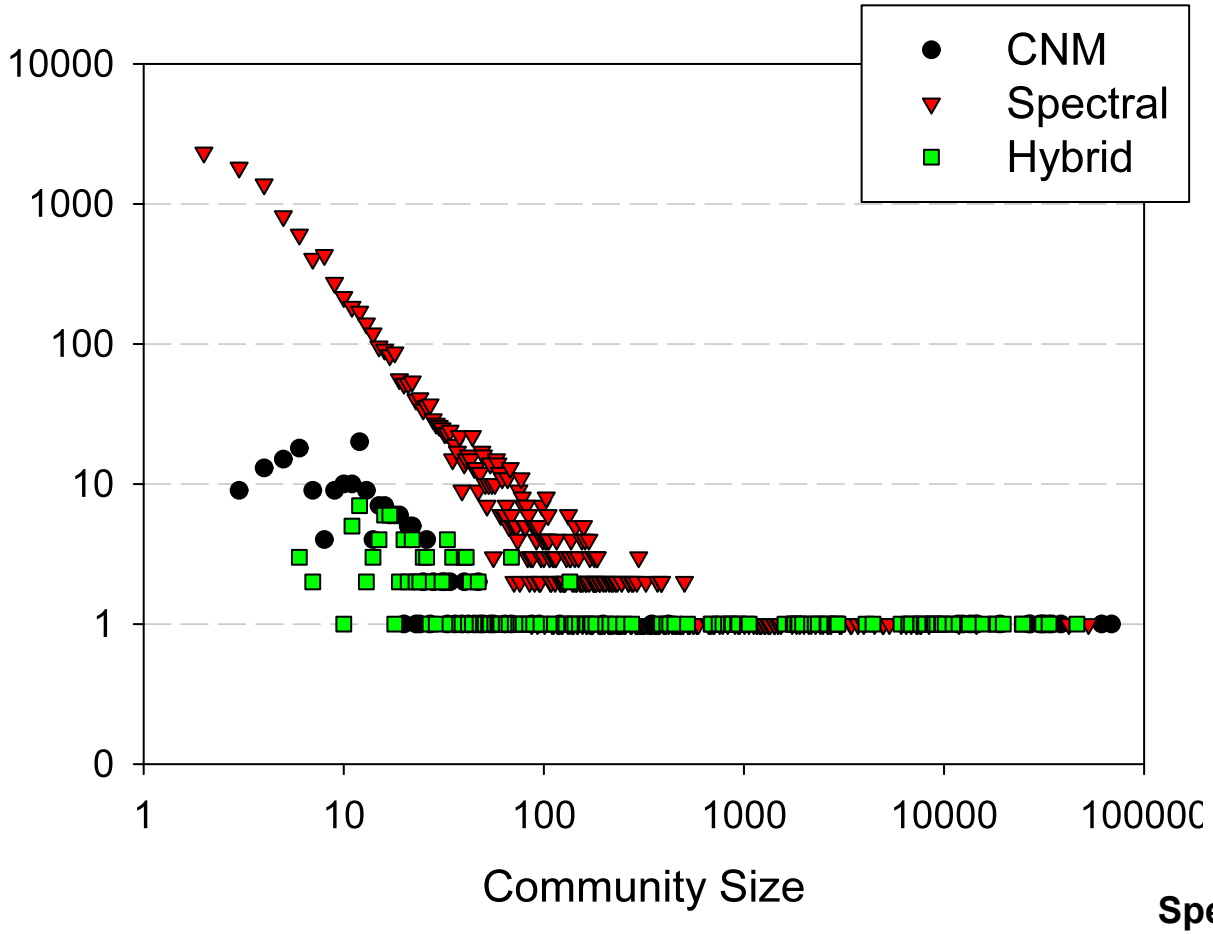
SNAP Algorithms: Comparative Performance



- Smallest network in the test suite.
- Divisive edgeBC and basic agglomerative clustering algorithm (CNM) highly compute-intensive.
- CNM-RAT (comm. sizes factored in) significantly faster than CNM.

Performance results on a Intel Xeon 5560 (Nehalem) system.

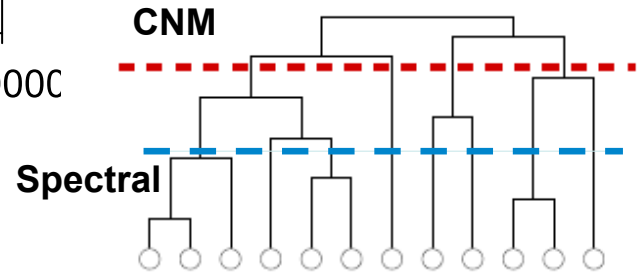
Communities: Sizes and Cardinality



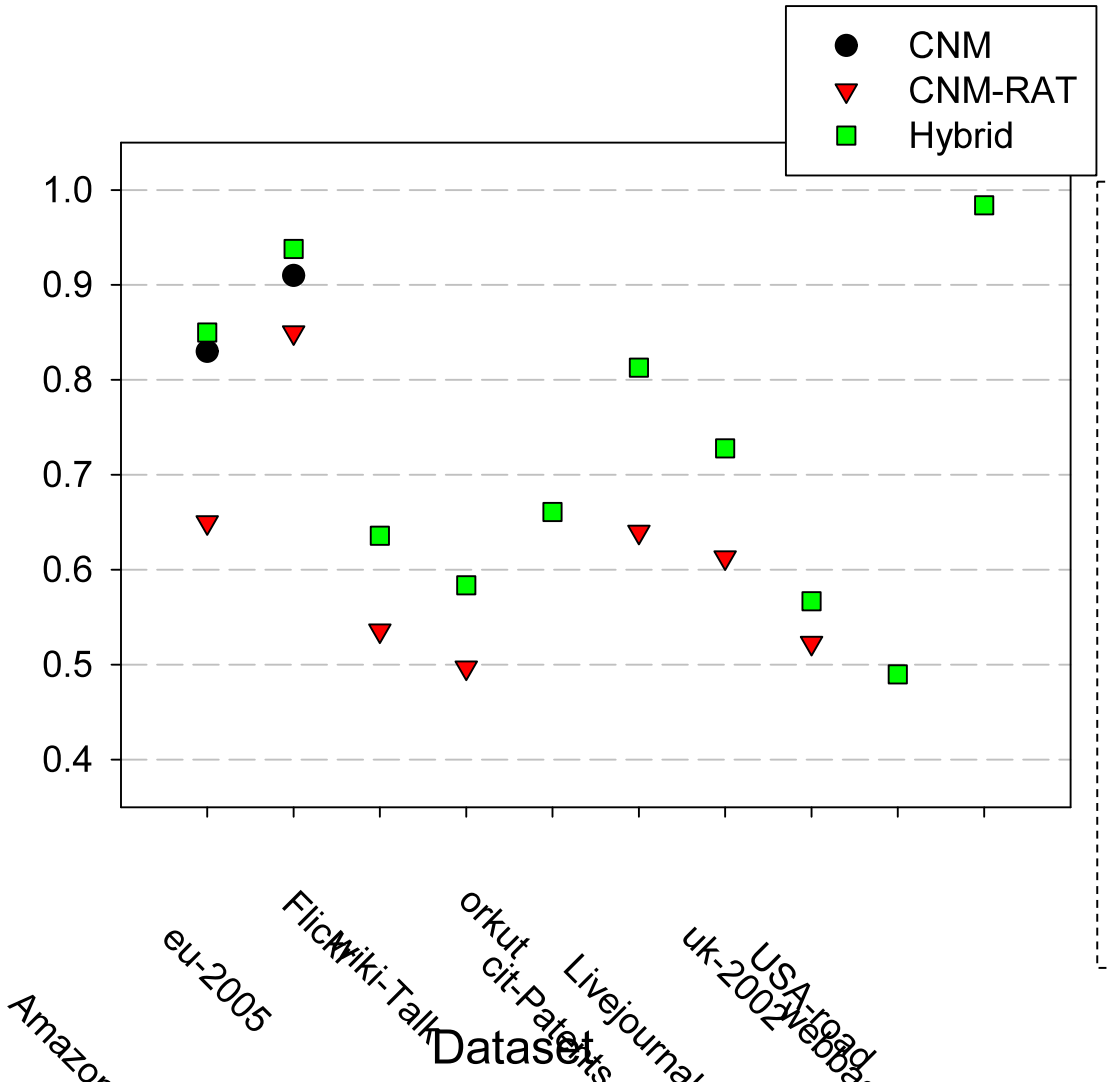
Amazon-2003
(n=473K, m=3.5M)

# of communities	
CNM	240
Spectral	10K
Hybrid	196

Spectral alg. fails to resolve communities beyond one level of the agglomerative clustering dendrogram!



Communities: Modularity

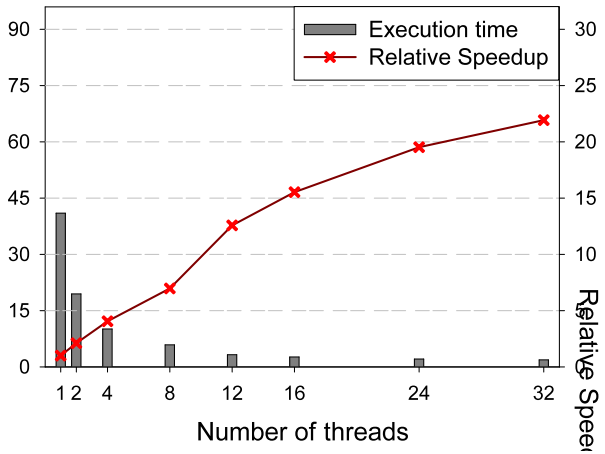


- Hybrid approach performs surprisingly well.
- # of communities from CNM-RAT and Hybrid roughly the same.
- CNM-RAT suffers in modularity quality.
- CNM did not finish (> 6hrs) for most networks.

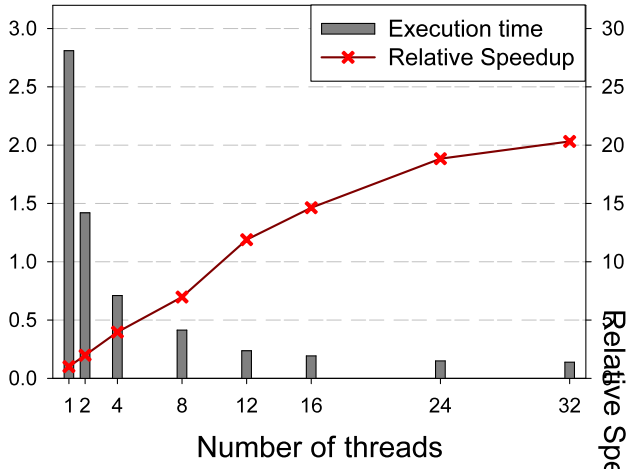
Dynamic graph computations

- Analysis of dynamic graphs becoming increasingly important: detect trends (WWW), allegiance switching, emerging communities (social networks).
- Goal: Avoid computing from scratch.
- Consider path-based problems:
  - Are there paths connecting *s* and *t* between time T_1 and T_2?
  - Does the path between *s* and *t* shorten drastically?
  - Is a vertex suddenly very central?
- Contribution: New **space-efficient data structures**, fast algorithms

Construction time



Query time (1 million queries)



Link-cut tree for answering connectivity queries. Performance results on Sun Fire T5140. 10 million vertices, 80 million edges

Madduri and Bader, IPDPS'09.

Talk Outline

- Large-scale graph analytics: Introduction and motivating examples
- Designing parallel graph analysis algorithms and software
- Application case studies
  - Community Identification in social networks
  - RDF data analysis using compressed bitmap indexes

Semantic data analysis

- The RDF (Resource Description Framework) data model is a popular abstraction for linked data repositories
  - *triple* form [<subject> <predicate> <object>]

  - Data sets with a few billion triples quite common

- Emergence of “triple stores”, custom databases for storage and retrieval of RDF data
  - Jena, Virtuoso, Sesame

FastBit-RDFJoin

- We use the compressed bitmap indexing software FastBit to index RDF data
- Search queries on RDF data can be accelerated with use of compressed bitmap indexes

- Our Contributions:
  - Parallel bitmap index construction (we store the sparse graphs corresponding to each unique predicate)
  - New query-answering approach: pattern matching queries on RDF data are modified to use bitmap indexes.
- Speedup insight: SPARQL queries can be expressed as fast and I/O optimal bit vector operations.

K. Madduri, K. Wu, Proc. SSDBM 2011.

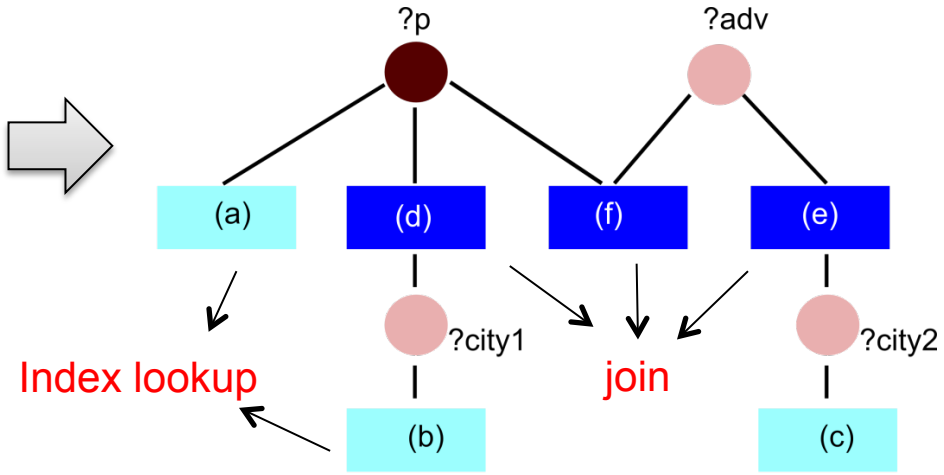
Answering a SPARQL Query with Bitmap Indexes

Search Query: list of all scientists born in a city in USA, who have/had a Doctoral advisor born in Chinese city.

SPARQL Query

```
Select ?p where {
   ?p      <type>              'scientist' .
   ?city1 <locatedIn>          'USA' .
   ?city2 <locatedIn>          'China' .
   ?p      <bornInLocation> ?city1 .
   ?adv   <bornInLocation> ?city2 .
   ?p      <hasDoctoralAdvisor> ?adv .
```

Query Graph



The ordering of bit vector operations determines query work performed.

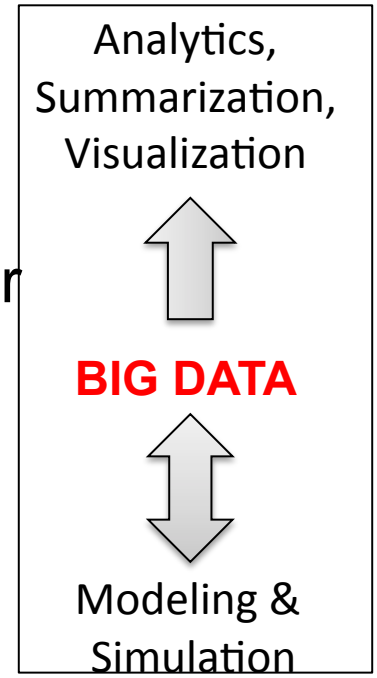
Performance results: LUBM benchmark

- LUBM SPARQL test query evaluation time in milliseconds, performance on a 2.67 GHz Intel Xeon processor.

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
LUBM-1M, Cold caches							
FastBit	0.078	16.2	0.098	0.150	0.118	2.85	0.12
RDF-3X	15.5	32.3	7.3	1.27	1.14	92.4	1.25
LUBM-1M, Warm caches							
FastBit	0.008	15.7	0.026	0.042	0.028	2.59	0.032
RDF-3X	0.385	10.2	0.385	0.553	1.11	76.1	0.89
Speedup	48.1×	0.65×	14.8×	13.16×	39.6×	29.4×	27.8×
LUBM-50M, Cold caches							
FastBit	0.30	1320	1.26	0.65	0.34	139	0.643
RDF-3X	0.43	572	2.9	0.75	2.1	4150	4.62
LUBM-50M, Warm caches							
FastBit	0.167	1311	0.92	0.40	0.19	135	0.46
RDF-3X	0.31	544	0.193	0.70	1.95	4021	1.52
Speedup	1.86×	0.42×	0.21×	1.75×	10.26×	29.8×	3.30×

Summary: Our Research Enables Complex Data-intensive Applications

- The SNAP (snap-graph.sf.net) framework offers novel parallel methods for social and information network analytics
  - Two orders of magnitude faster than competing "serial" software approaches
- We have designed the first parallel methods for several community detection formulations
- Ongoing research projects
  - Semantic data analytics using compressed bitmap indexes
  - Eulerian path-based de novo genome assembly
- Future research direction: Modeling network dynamics; persistent monitoring of dynamically changing properties



Collaborators, Acknowledgments

- Scientific Data Management & Future Technologies research groups, LBNL
- Prof. David A. Bader, Georgia Institute of Technology
- Jonathan Berry, Bruce Hendrickson (Sandia National Laboratories)
- John Feo, Daniel Chavarria (Pacific Northwest National Laboratories)
- PNNL CASS-MT Center and Cray Inc. for access to their XMT systems.
- Par Lab @ UC Berkeley for access to their Millennium cluster systems.
- Research supported in part by DOE Office of Science under contract number DE-AC02-05CH11231.

Thank you!

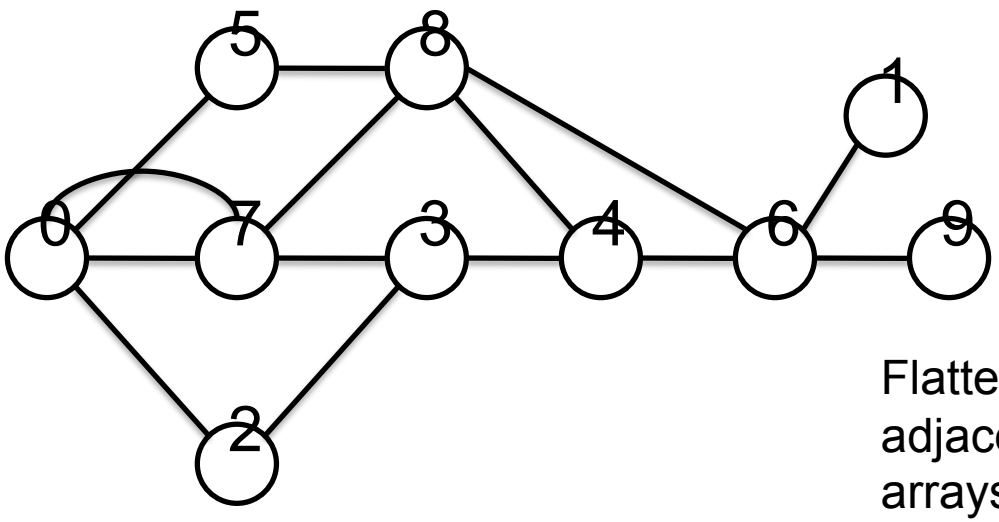
- Questions?

Backup Slides

Graph 500 and Parallel BFS

Graph representation

- Compressed Sparse Row-like



Flatten adjacency arrays

Vertex	Degree	Adjacencies			
0	4	2	5	7	7
1	1	6			
2	2	0	3		
3	3	2	4	7	
4	3	3	6	8	
5	2	0	8		
6	4	1	4	8	9
7	4	0	0	3	8
8	4	4	5	6	7
9	1	6			

Index into adjacency array

0	4	5	7	…	28

Size: n+1

Adjacencies

2	5	7	7	6	0	3	2	4	….	6	7	6

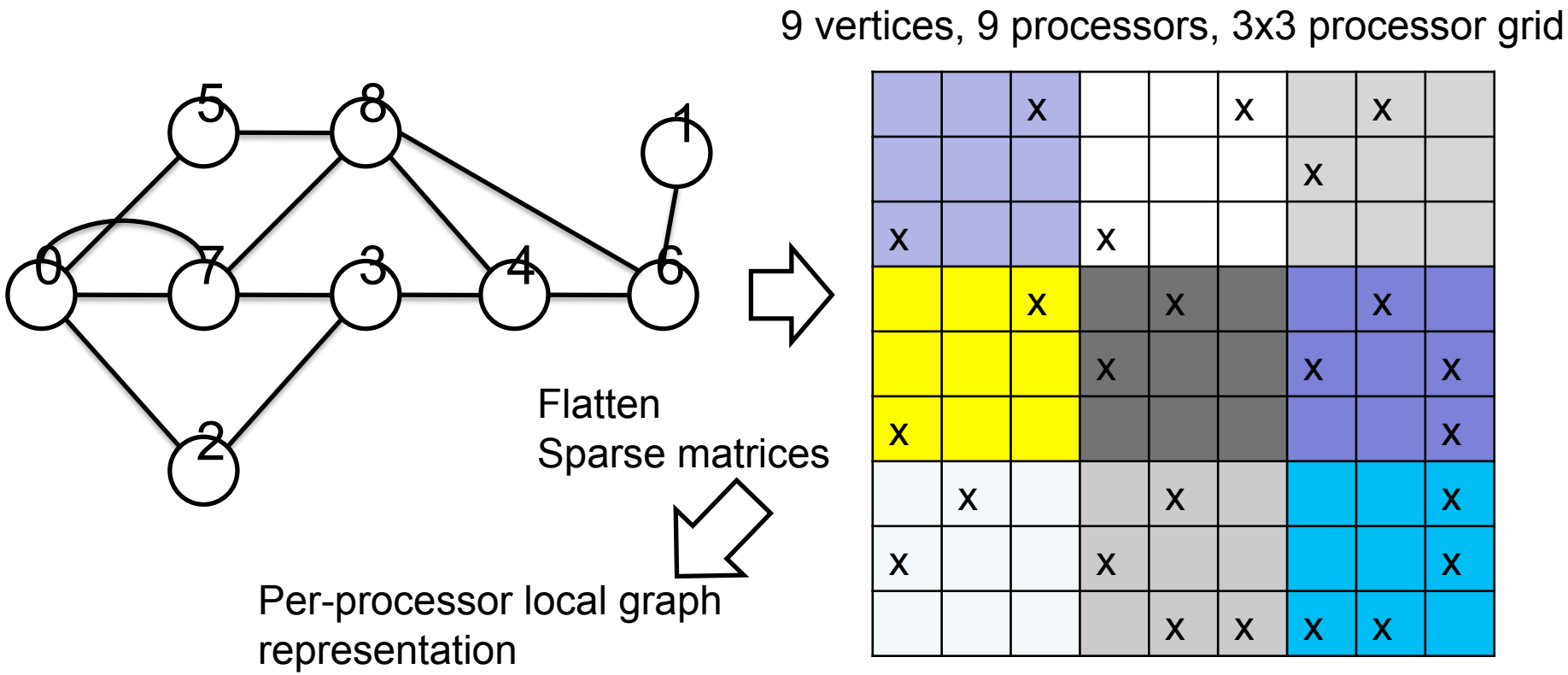
Size: 2*m

Distributed Graph representation

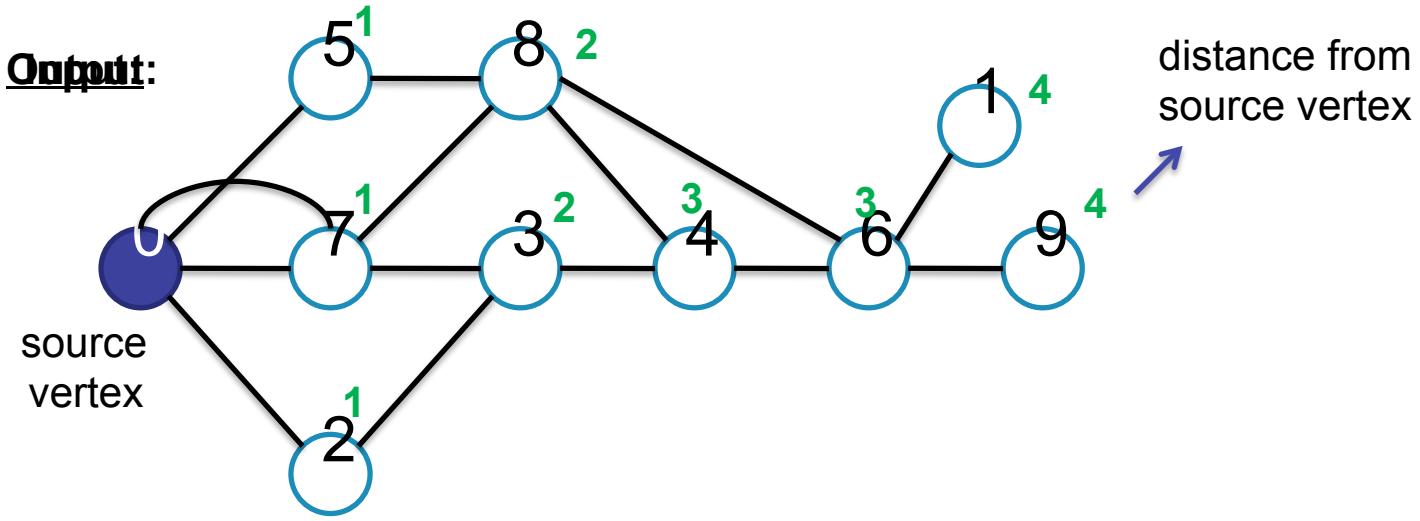
- Each processor stores the entire graph (“full replication”)
- Each processor stores n/p vertices and all adjacencies out of these vertices (“1D partitioning”)
- How to create these “p” vertex partitions?
  - Graph partitioning algorithms: recursively optimize for conductance (edge cut/size of smaller partition)
  - Randomly shuffling the vertex identifiers ensures that edge count/processor are roughly the same

2D graph partitioning

- Consider a logical 2D processor grid (p_r * p_c = p) and the matrix representation of the graph
- Assign each processor a sub-matrix (i.e, the edges within the sub-matrix)



Graph traversal (BFS) problem definition

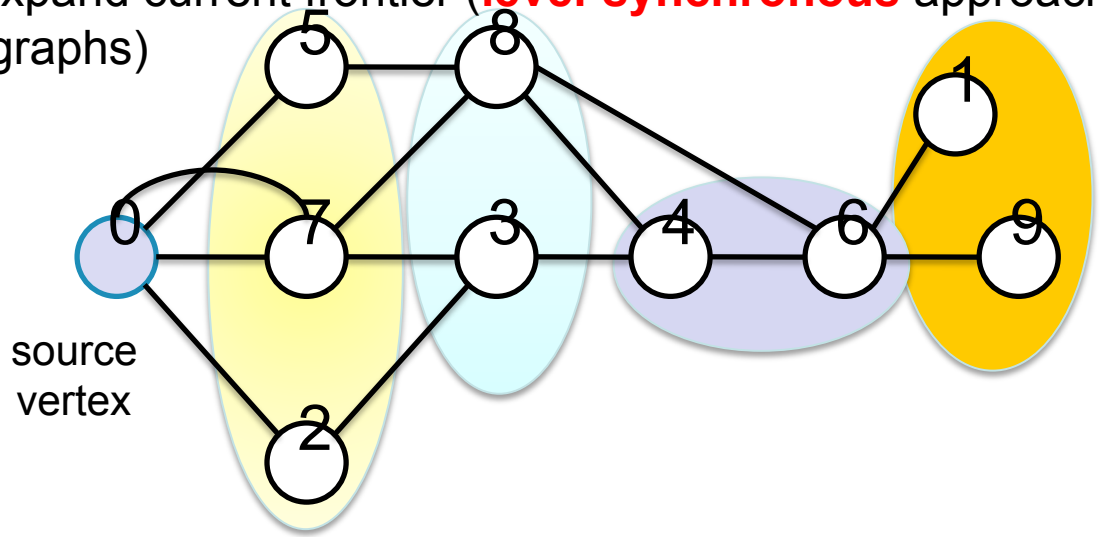


Memory requirements (# of machine words):

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

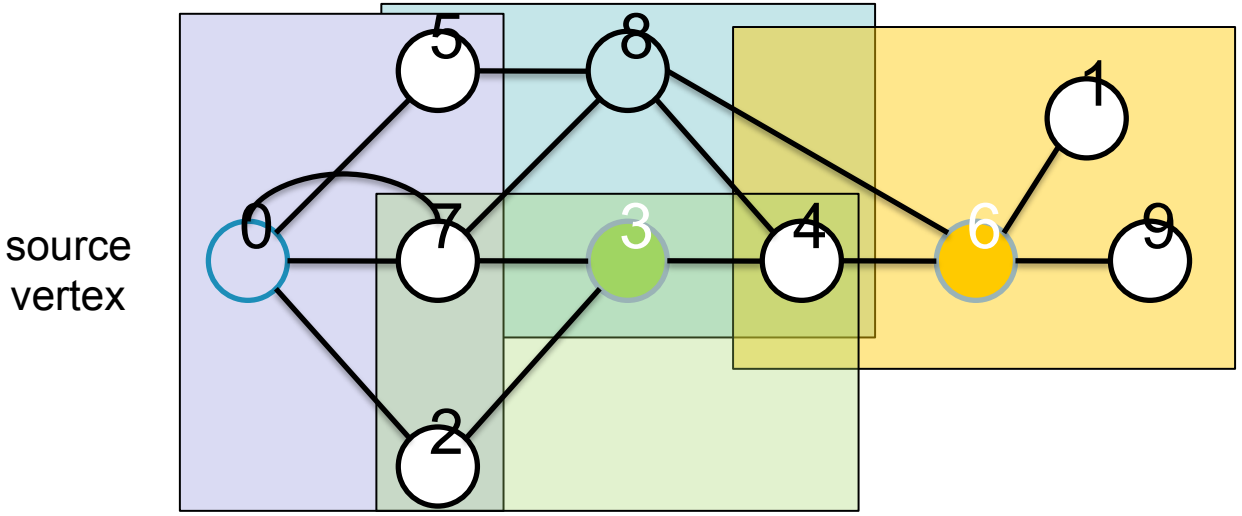
Parallel BFS Strategies

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



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

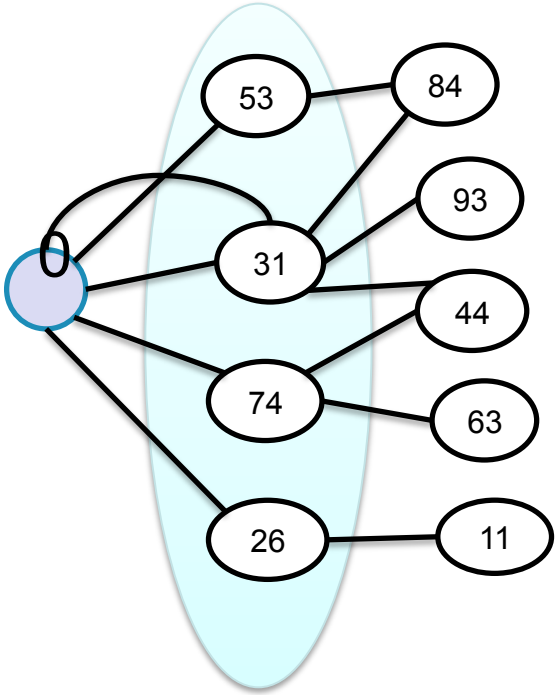
2. Stitch multiple concurrent traversals (Ullman-Yannakakis approach, suited for **high-diameter** graphs)



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

A deeper dive into the “level synchronous” strategy

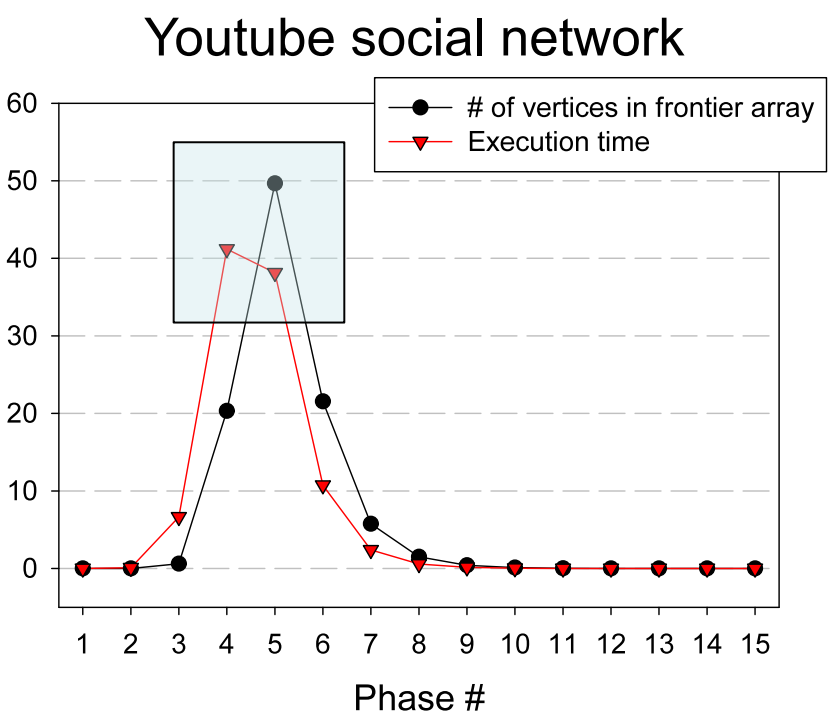
Locality (where are the random accesses originating from?)

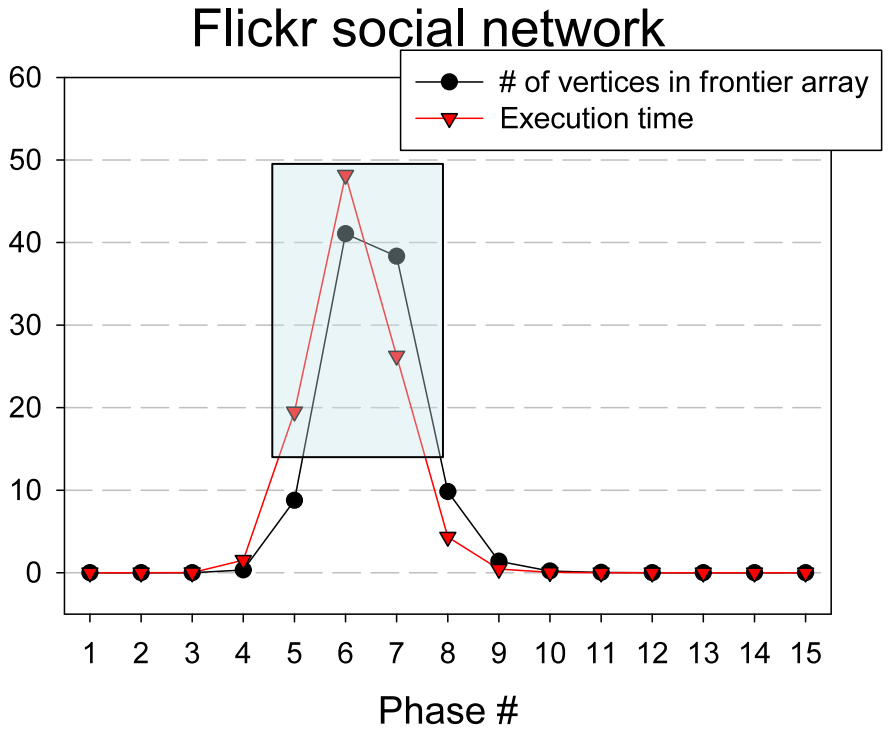


1. Ordering of vertices in the “current frontier” array, i.e., accesses to adjacency indexing array, cumulative accesses O(n).
2. Ordering of adjacency list of each vertex, cumulative O(m).
3. Sifting through adjacencies to check whether visited or not, cumulative accesses O(m).

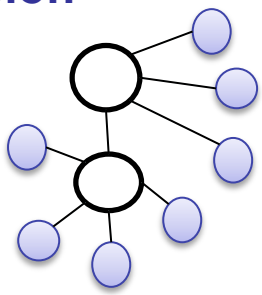
1. Access Pattern: idx array -- 53, 31, 74, 26
2,3. Access Pattern: d array -- 0, 84, 0, 84, 93, 44, 63, 0, 0, 11

Performance Observations

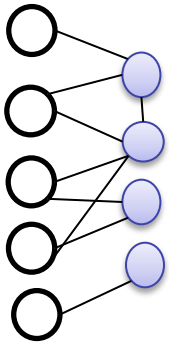




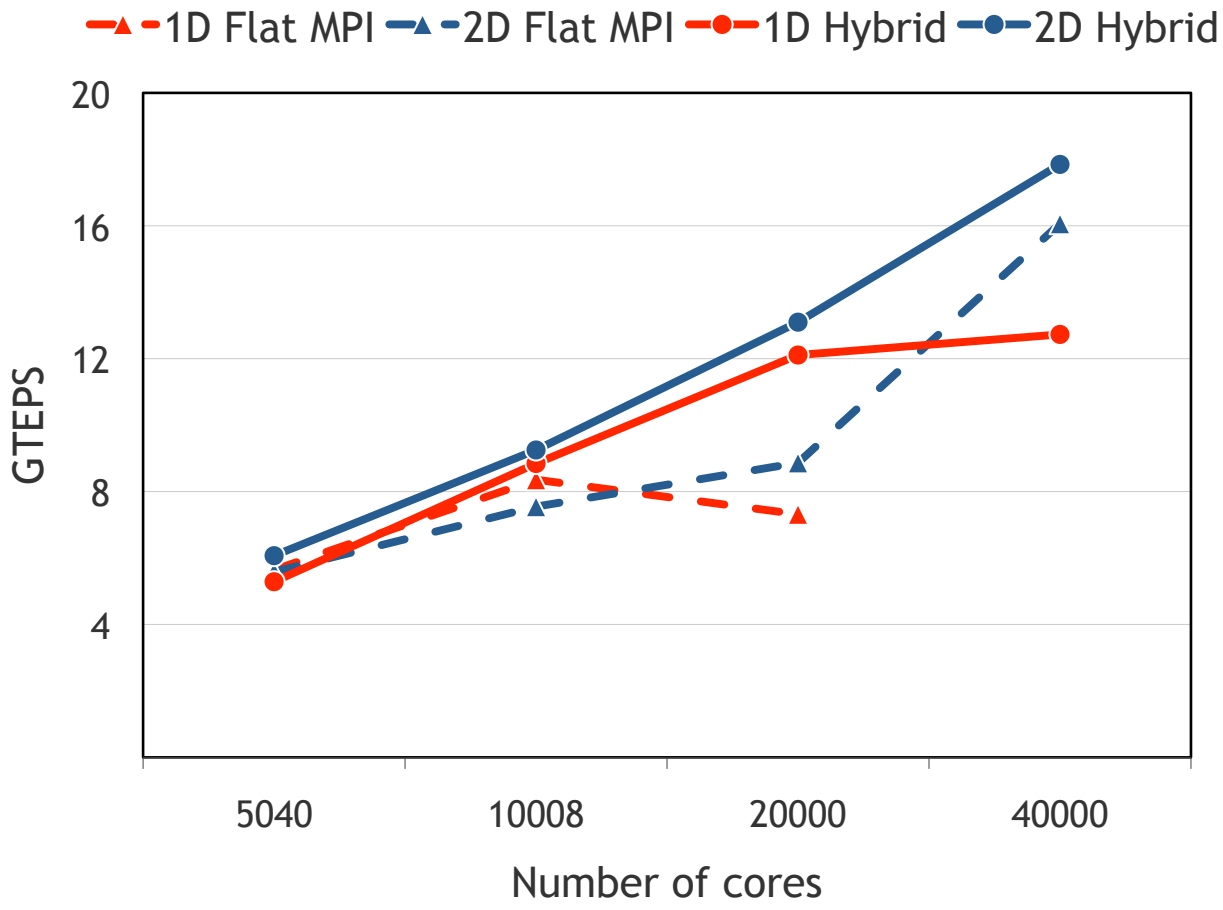
Graph expansion



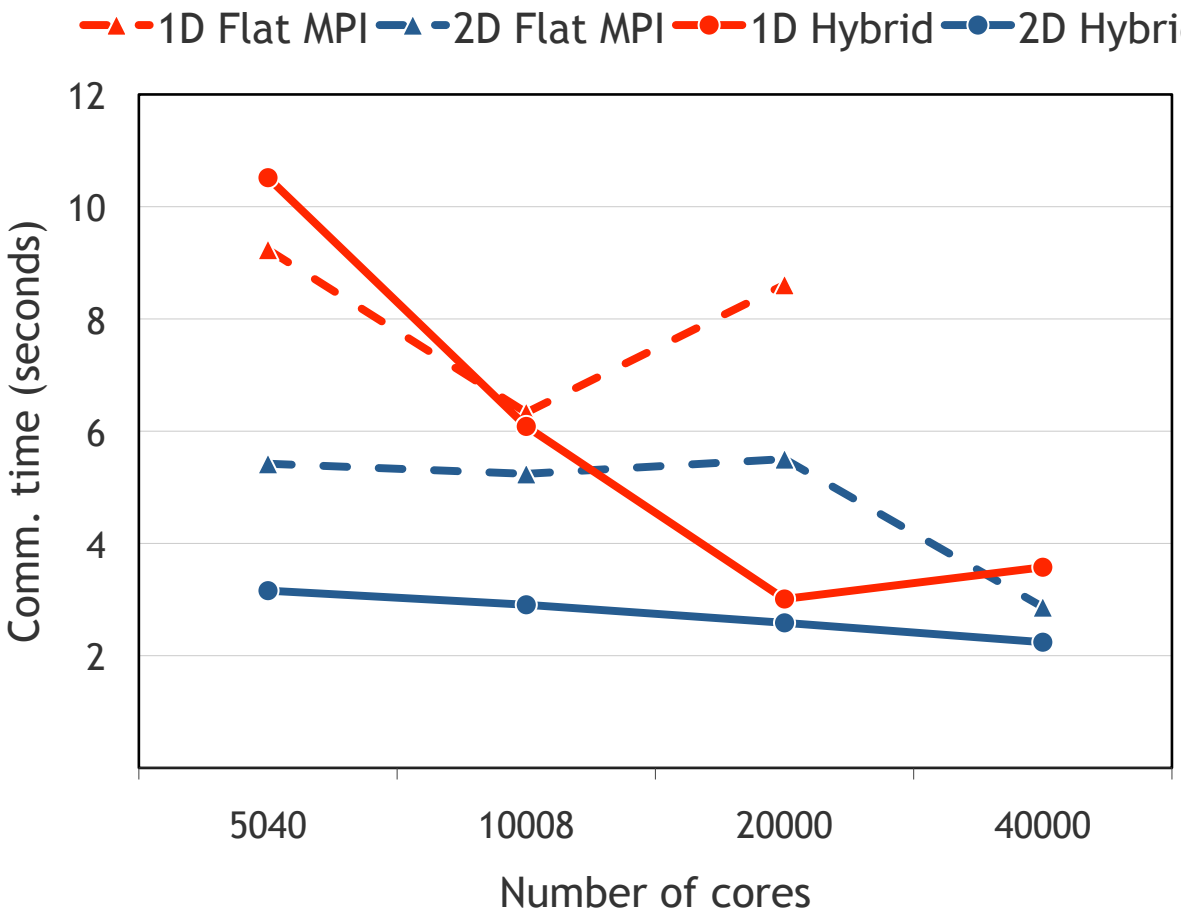
Edge filtering



Graph500 BFS: SCALE 32 performance on Hopper (Cray XE6, 24 cores per node)



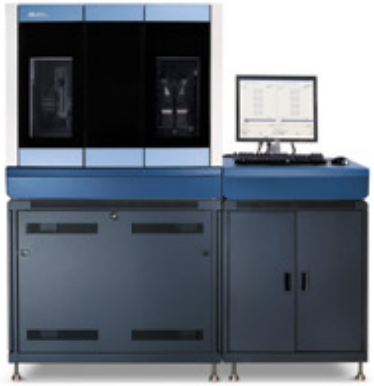
Graph500 BFS: SCALE 32 communication time on Hopper (lower is better)

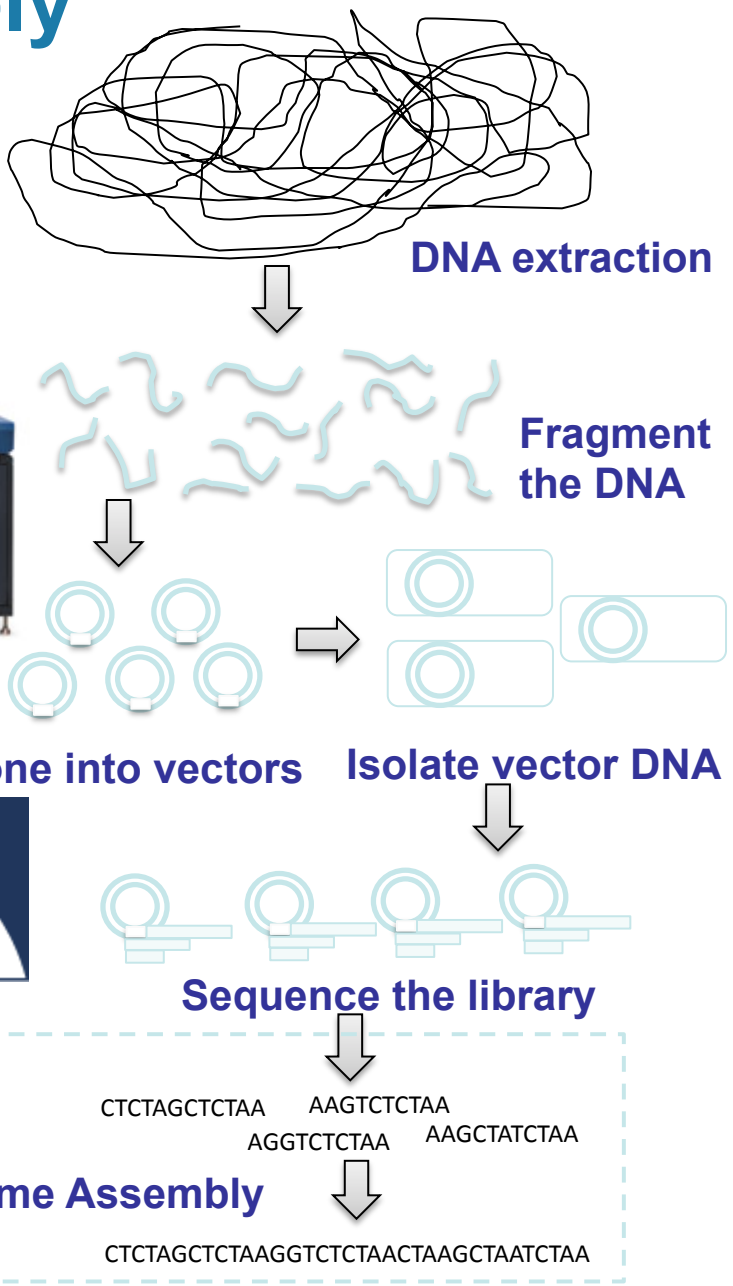


De novo Genome Assembly

De novo Genome Assembly

- Genome Assembly: “a big jigsaw puzzle”
- *De novo:* Latin expression meaning “from the beginning”
  - No prior reference organism
  - Computationally falls within the **NP-hard** class of problems





Eulerian path-based assembly strategies

- Break up the (short) reads into overlapping strings of length *k*.

k = 5

ACGTTATATATTCTA ⇨ ACGTT CGTTA GTTAT TTATA ….. TTCTA

CCATGATATATTCTA ⇨ CCATG CATGA ATGAT TGATA ….. TTCTA

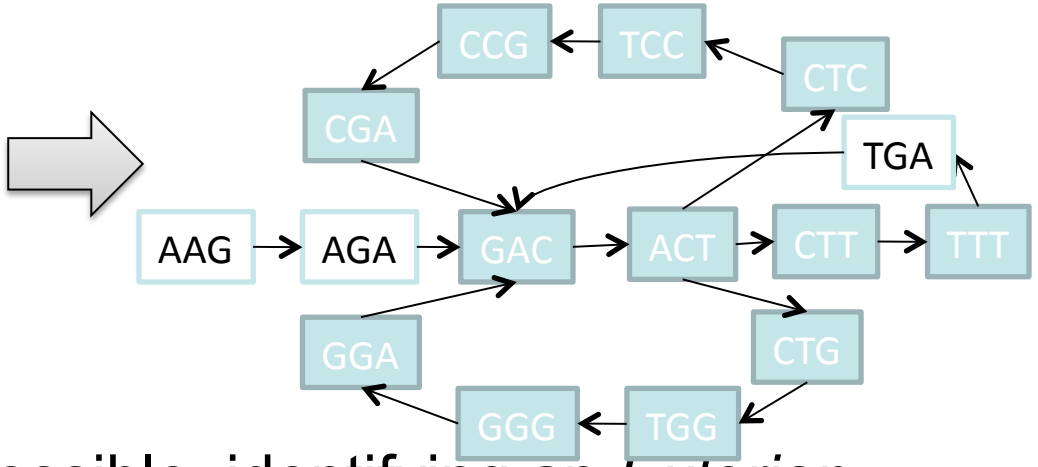
- Construct a de Bruijn graph (a directed graph representing overlap between strings)

de Bruijn graph-based Assembly

- Each (*k-1)*-mer represents a node in the graph
- Edge exists between node *a* to *b* iff there exists a *k*-mer such that its prefix is *a* and suffix is *b.*

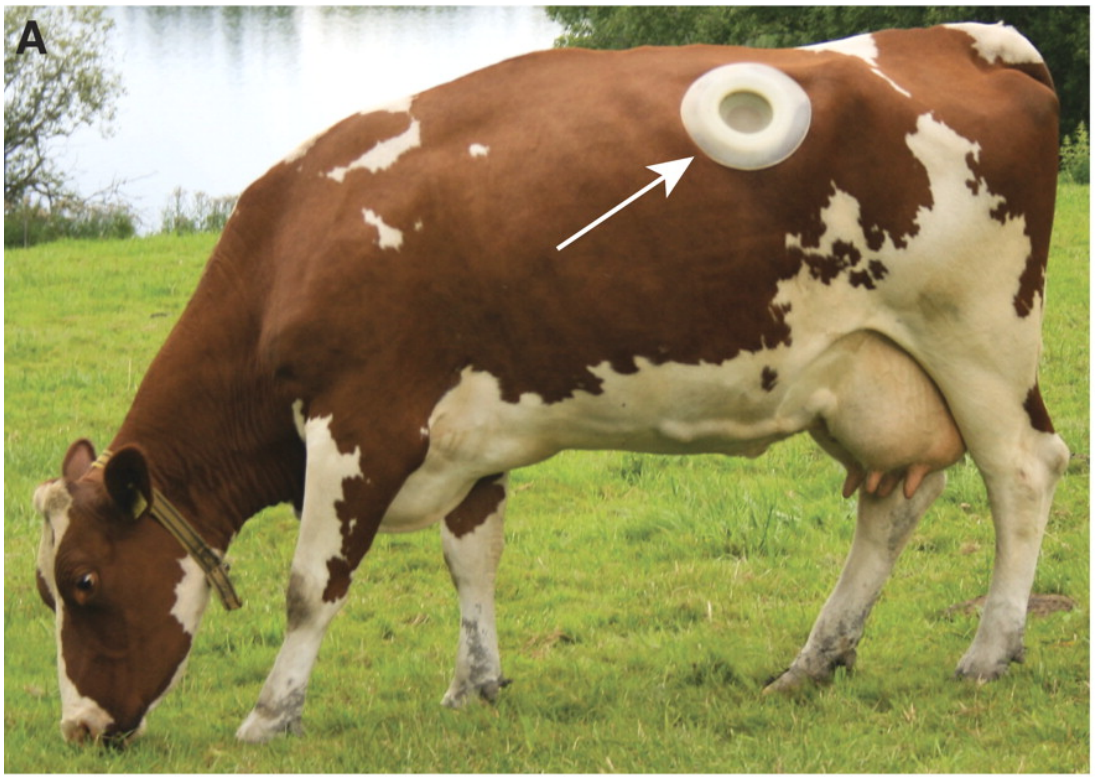
AAGACTCCGACTGGGACTTT

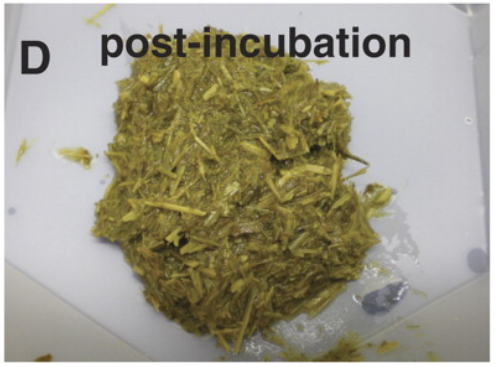
ACTCCGACTGGGACTTTGAC



- Traverse the graph (if possible, identifying an *Eulerian path*) to form contigs.
- However, correct assembly is just one of the many possible Eulerian paths.

Application: Identification of biomass-degrading Genes and Genomes from cow rumen





Goal: Identify microbial enzymes that aid in deconstruction of plant polysaccharides.
Cow rumen microbes known to be particularly effective In breaking down switchgrass.

Image Source: Hess et al., Science 331(6016), 463-467, 2011.

Metagenomes

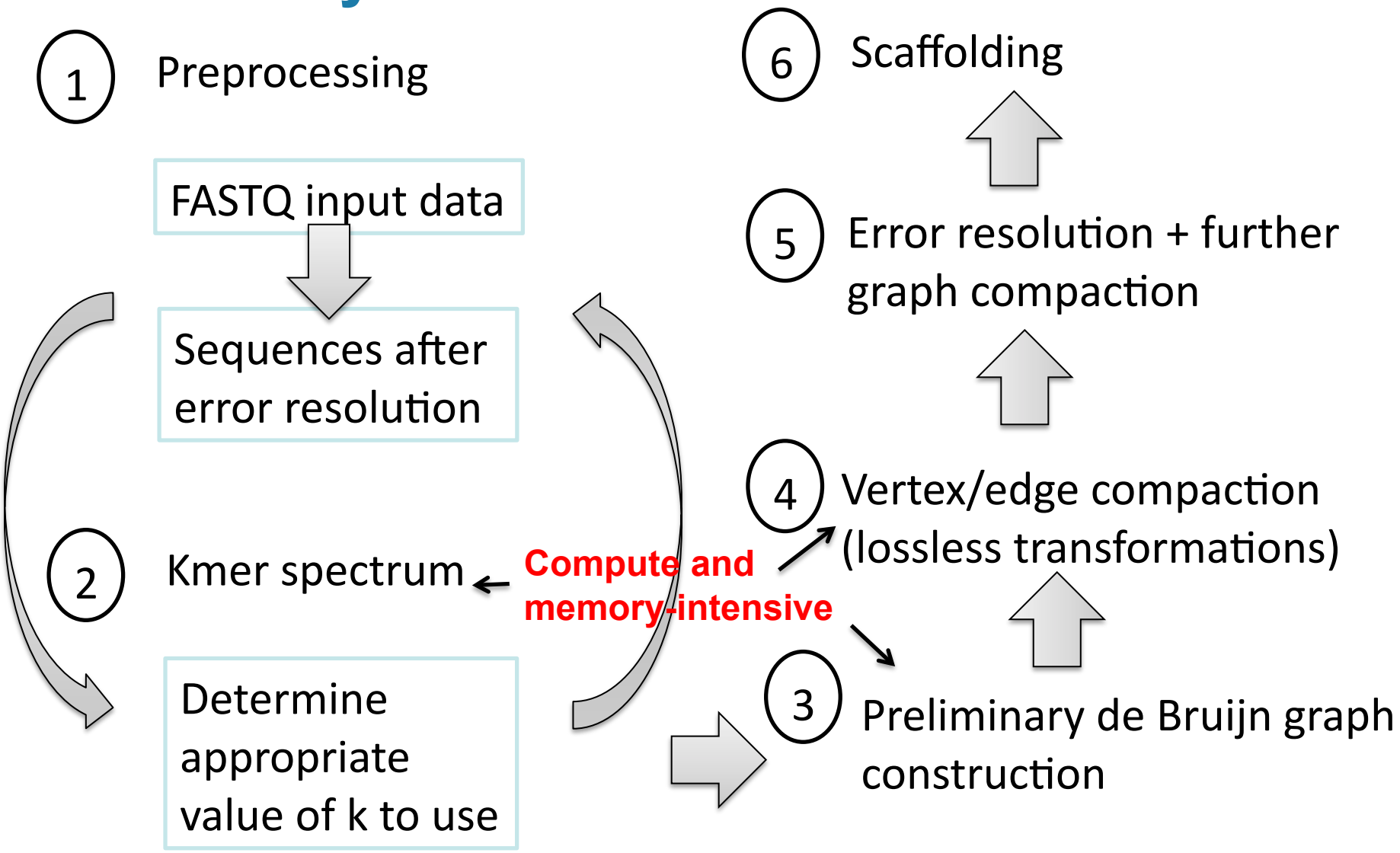
- Two major complications for de novo assembly:
  - Likely uneven representation of organisms within a sample
  - Likely polymorphisms between closely related members in an environment
- Assembly is difficult even if we have an estimate of organism representation in a sample
- If coverage is not known, Poisson likelihood estimates used by isolate genome assemblers break down.

Towards designing a metagenome assembler

- Given the challenges, what approach do we take?

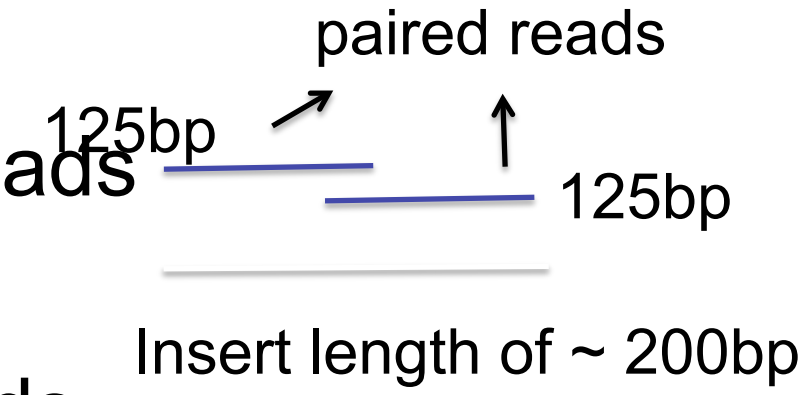
- We can still construct the de Bruijn graph
  - Try out various values of *k*, use base quality information
  - Require parallel computation for dealing with the large data sizes
- Understand data set characteristics to suggest algorithmic changes in current assemblers
  - Can we automate selection of *k*?
  - What is the genome coverage like?
  - Can we predict the approximate size of the metagenome?

Steps in the new de Bruijn graph-based assembly scheme



1. Preprocessing

- Process base quality information
- Mark ambiguous bases
- Try to merge paired reads



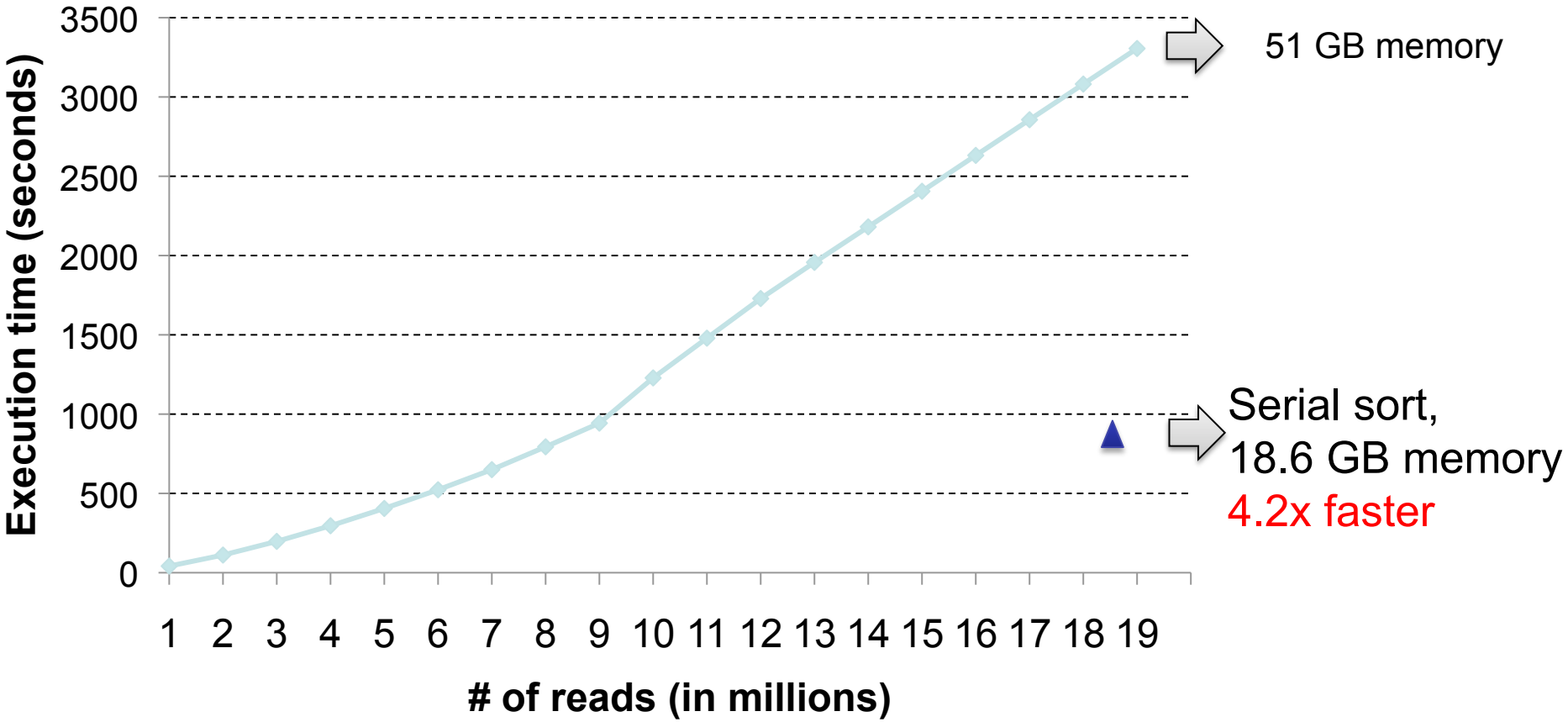
- Write back filtered reads
- Parallelization strategy: split input files into "P" parts; each node processes its file independently
  - Predominantly I/O bound

2. Kmer spectrum construction

- Need a dictionary to track occurrences of each kmer
- Velvet uses a splay tree to track unique kmers
  - Splaying expensive for large data sizes; maintaining an ordered set unnecessary when kmer updates are predominantly insert-only ("cow rumen" dataset)
- Alternative: Ingest all kmers, perform lexicographical sort
- Parallelization: enumerate kmers independently + one global sort to get kmer count

Finding unique kmers: hashing vs sorting

Splay tree update time for a data set of 19.5 million (125 bp) reads (k=61)

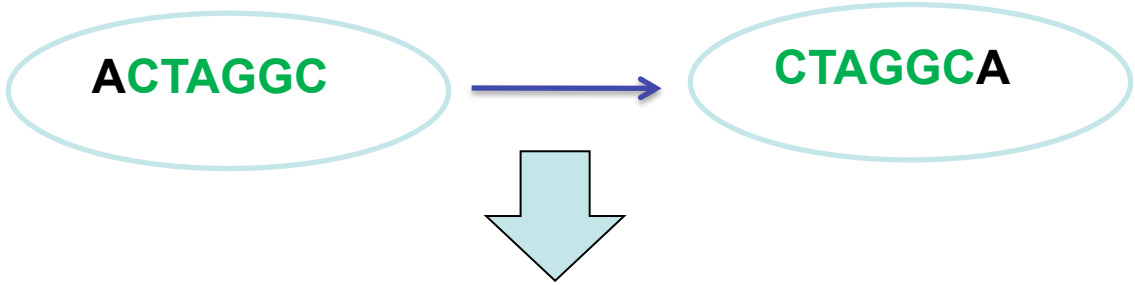


Serial performance results on a 512 GB system (2.6 GHz Opteron processor)

3. Graph construction

- Store edges only, represent vertices (kmers) implicitly.
- Distributed graph representation
- Sort by start vertex
- Edge storage format:

Read ‘x’:



Store edge (ACTAGGCA), orientation, originating read id (x), edge count

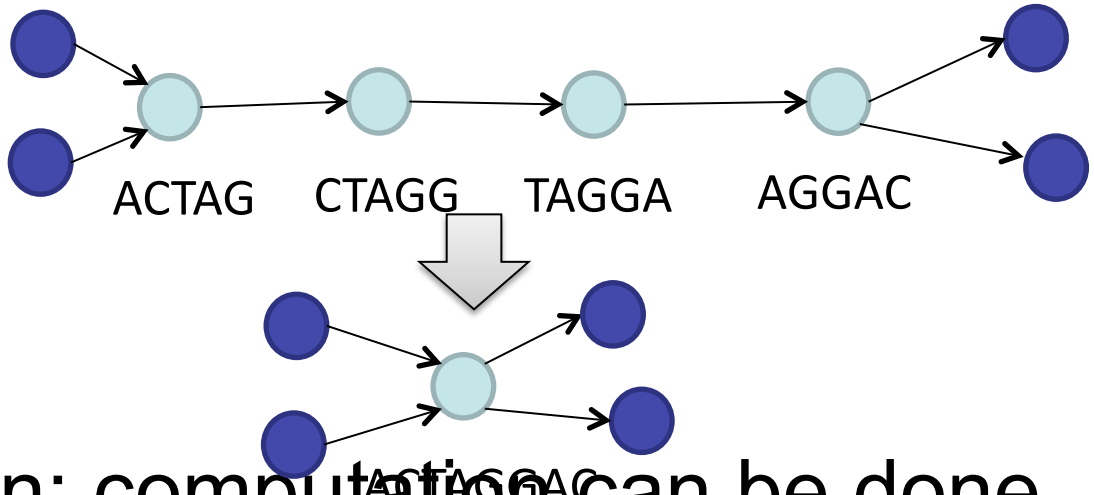
Use 2 bits per nucleotide

4. Vertex compaction

- High percentage of unique kmers

  ⇒ Try compacting kmers from same read first

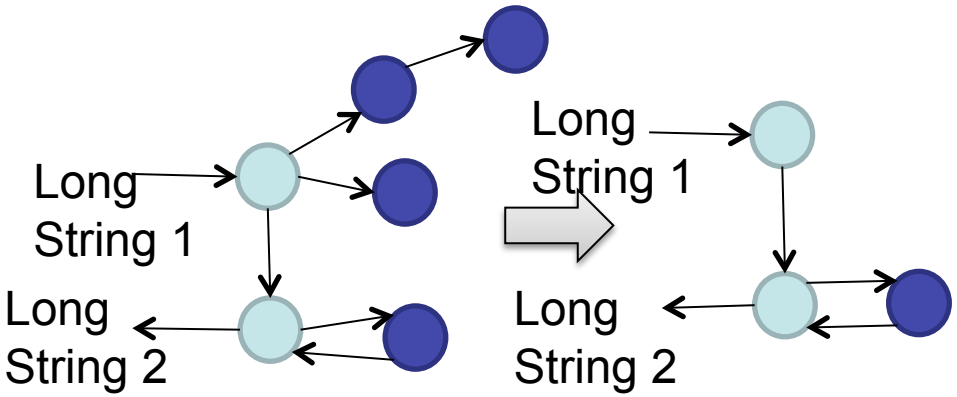
  – If kmer length is k, potentially k-times space reduction!



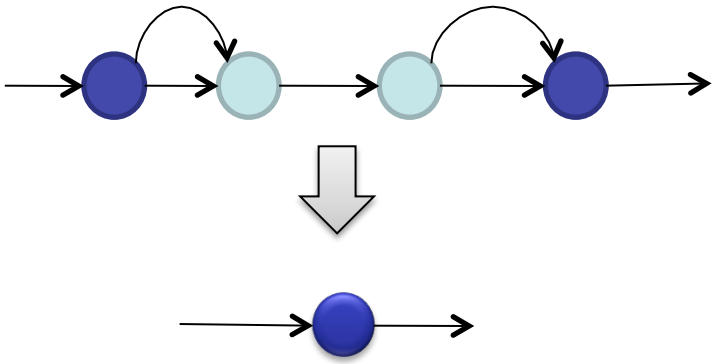
- Parallelization: computation can be done locally by sorting by read ID, traversing unit-cardinality kmers.

5. Redistributing reads for Velvetg execution

- Identify connected components in the string graph
- Error resolution and scaffolding can be concurrently performed on multiple independent components



Compress/remove whiskers

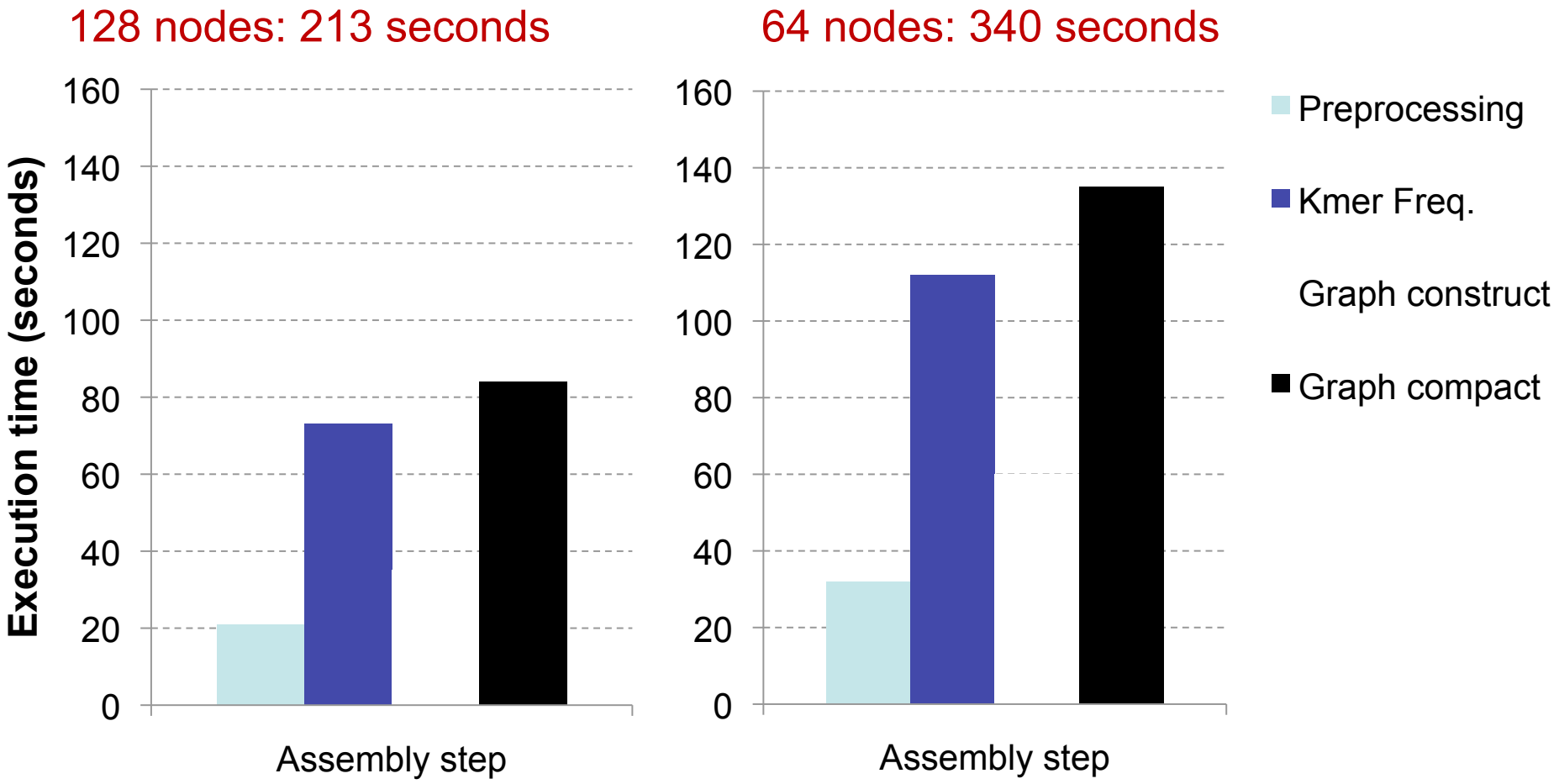


Identify and fix “low coverage” edges

Parallel Implementation Details

- Current data set (after preprocessing) requires 320 GB for in-memory graph construction
  - Experimented with 64 nodes (256-way parallelism) and 128 nodes (512-way) of NERSC Franklin (Cray XT4 system, 2.3 GHz quad-core Opteron processor)
- MPI across nodes + OpenMP within a node
- Local sort: multicore-parallel quicksort
- Global sort: sample sort

Parallel Performance



- Comparison: *Velveth* (up to graph construction) takes ~ 12 hours on the 512 GB Opteron system.

Observations

- Very conservative graph assembly
  - No filtering, getting exact global counts of kmers
- 20% of time spent in MPI communication (including global sort)
- 3.3x intra-node speedup for parallel sort (~7 GB array), execution time 32 seconds
- I/O (preprocessing and kmer freq.) not a bottleneck up to 128 nodes