Hybrid Parallel Programming for Massive Graph Analysis

Kamesh Madduri

KMadduri@lbl.gov

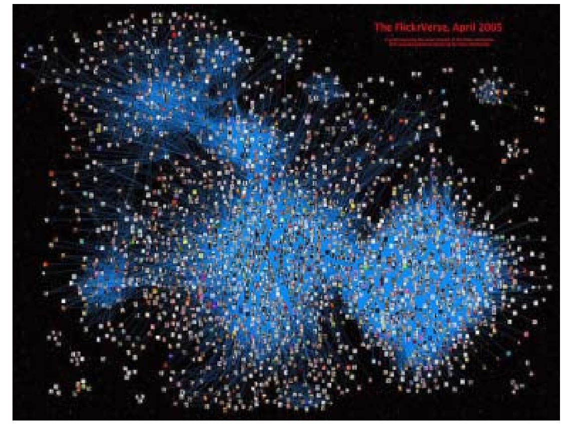
Computational Research Division

Lawrence Berkeley National Laboratory

SIAM Annual Meeting 2010

July 12, 2010

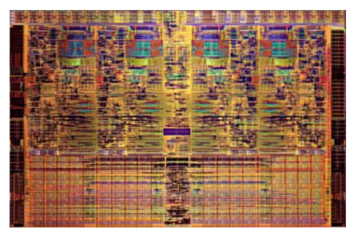
Hybrid Parallel Programming

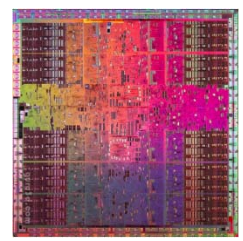


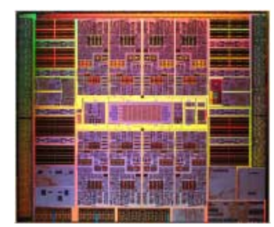
Large-scale graph analysis utilizing

- Clusters of x86 multicore processors
  - MPI + OpenMP/UPC

- CPU+GPU
  - MPI + OpenCL







- FPGAs, accelerators
  - Host code + accelerator code

Why hybrid programming?

- Traditional sources of performance improvement are flat-lining
- We need new algorithms that exploit large on-chip memory, shared caches, and high DRAM bandwidth

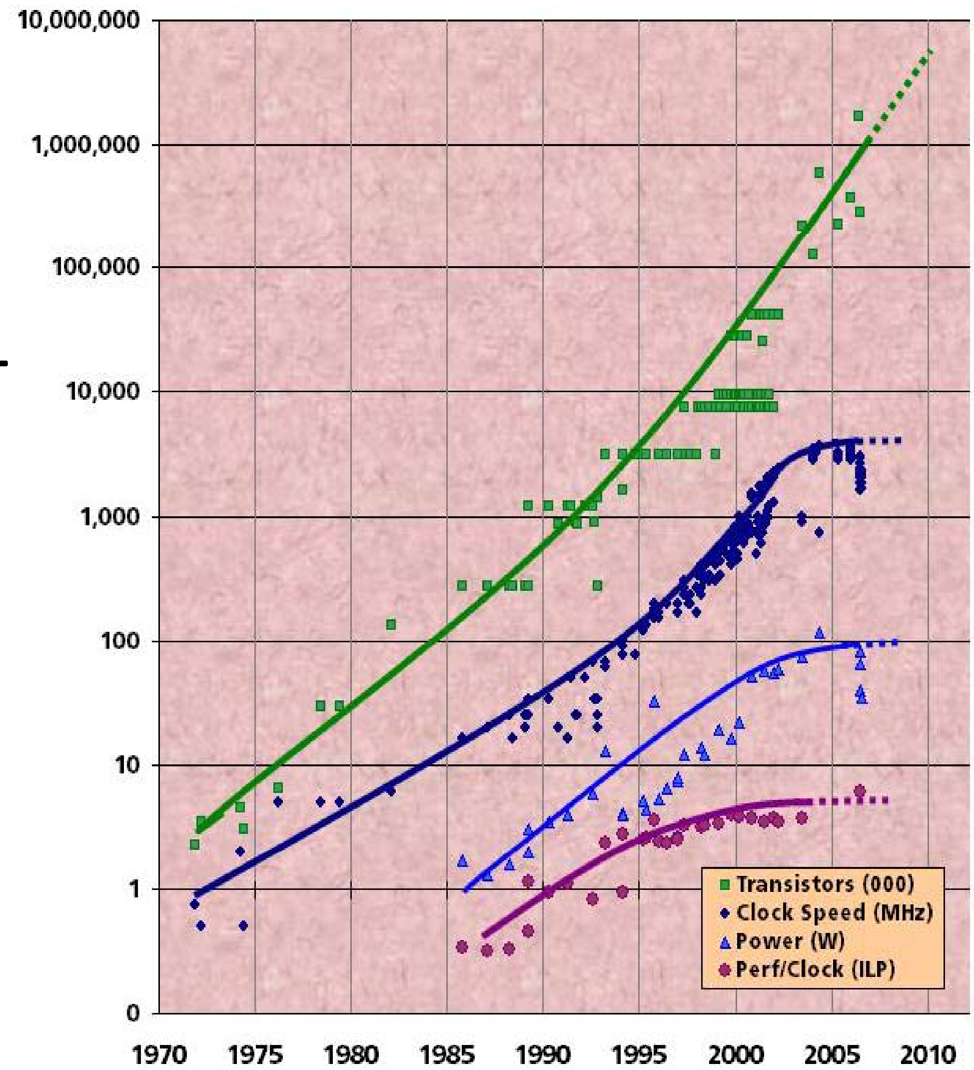
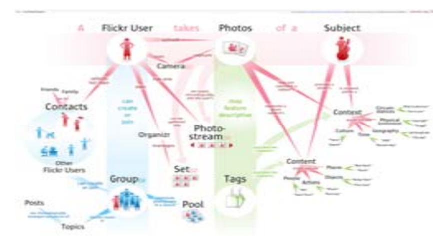


Image source: Herb Sutter, “The Free Lunch is Over”, Dr. Dobb’s Journal, 2009.

This talk: Two case studies

- MPI + OpenMP on shared-memory multicore processor clusters
  - Graph analytics on online social network crawls, synthetic "power-law" random graphs
  - Traversal and simplification of a DNA fragment assembly string graph arising in a de novo short-read genome assembly algorithm

Characterizing Large-scale graph-theoretic computations



Random/Global memory accesses

Enumerate all contacts of K within X hops

Find all events in the past six months similar to event "Y"

Locality Characteristics

Enumerate all friends of K

List today's top trending events

Streaming data/ Local computation

Computational Complexity

O(N)

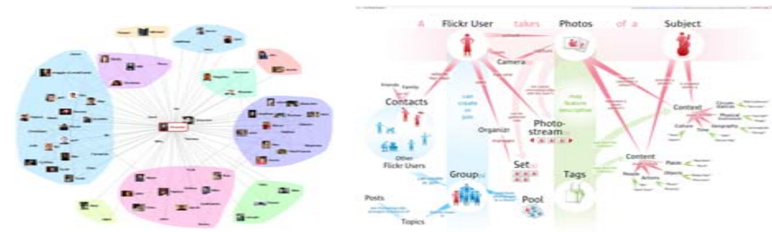
O(N log N)

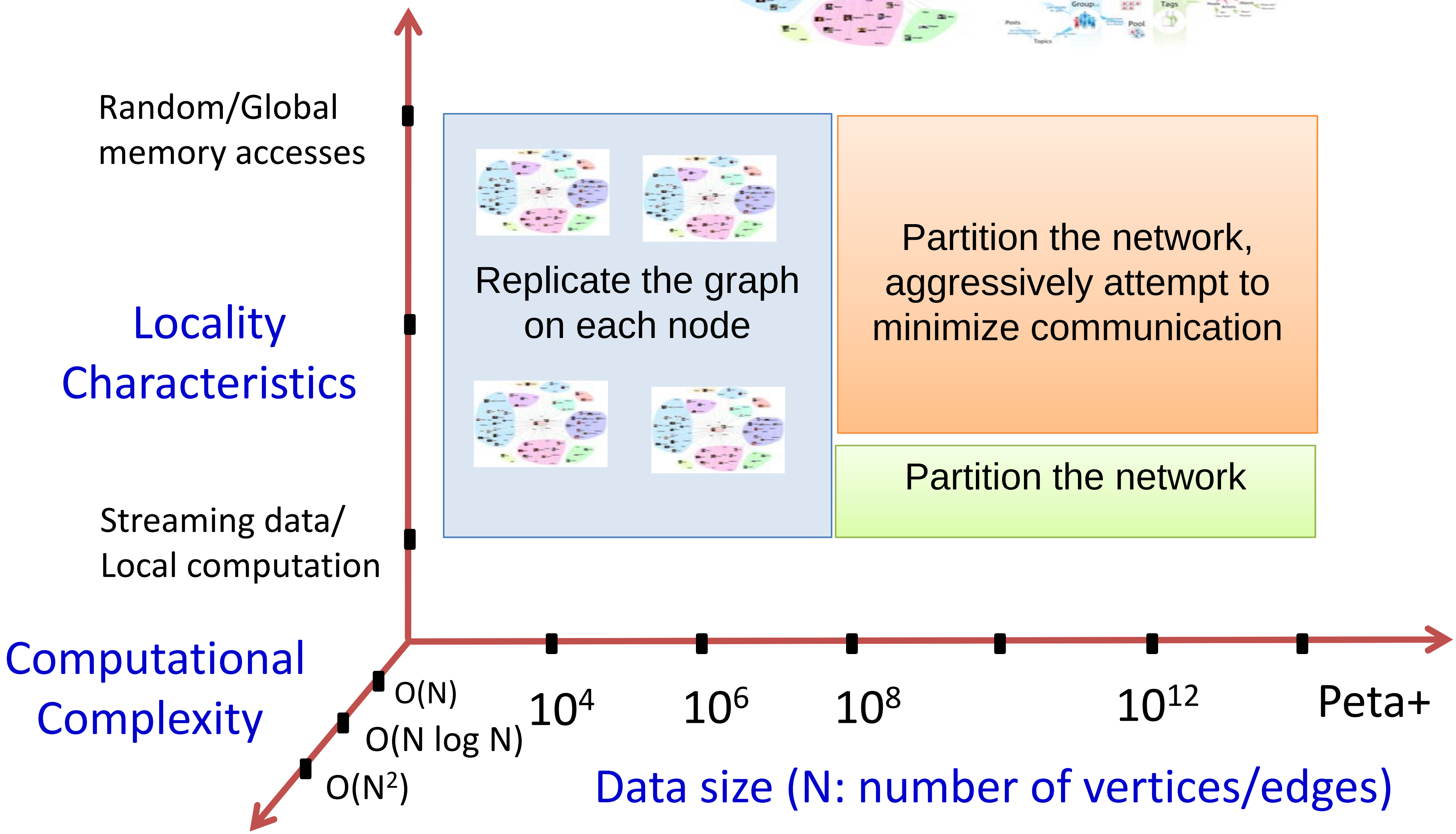
O(N^2)

10^4 10^6 10^8 10^{12} Peta+

Data size (N: number of vertices/edges)

Parallelization Strategy

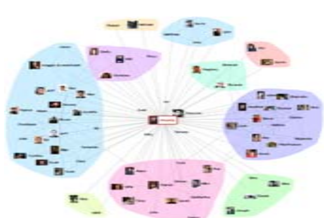


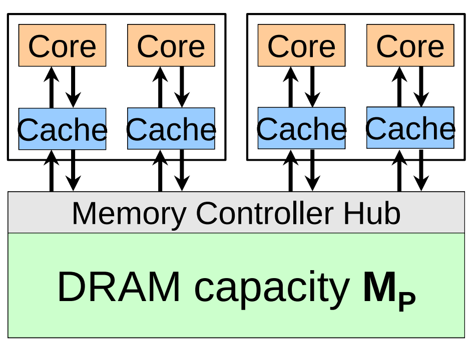


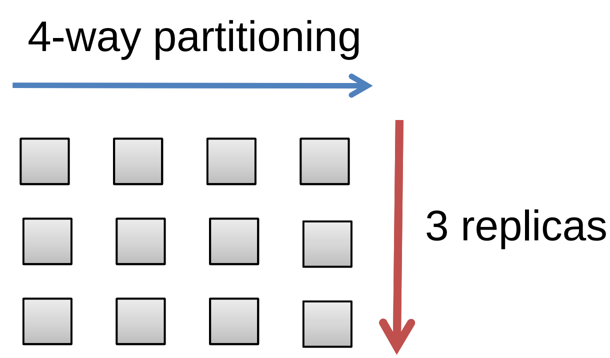
Minimizing Communication

- Irregular and memory-intensive graph problems: Intra- and Inter-node communication (+ I/O costs, memory latency) costs typically dominate local computational complexity
- Key to parallel performance: Enhance data locality, avoid superfluous inter-node communication
  - **Avoid a P-way partitioning** of the graph
  - Create PM_P/M_G replicas

Graph + data structures **M_G**







$P=12$, $M_G/M_P = 4$

Real-world data

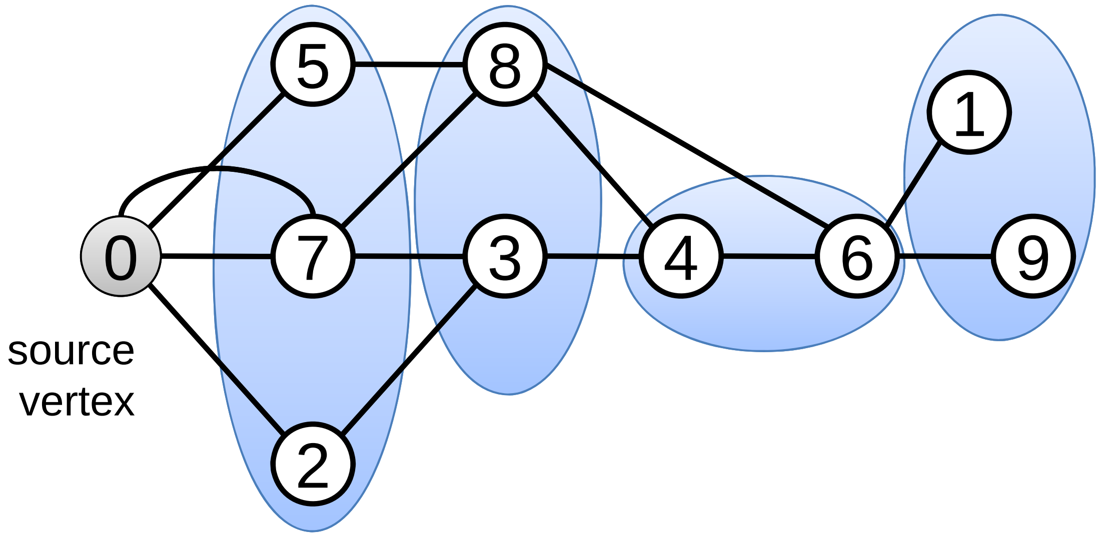
Assembled a collection of graphs for algorithm performance analysis, from some of the largest publicly-available network data sets.

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

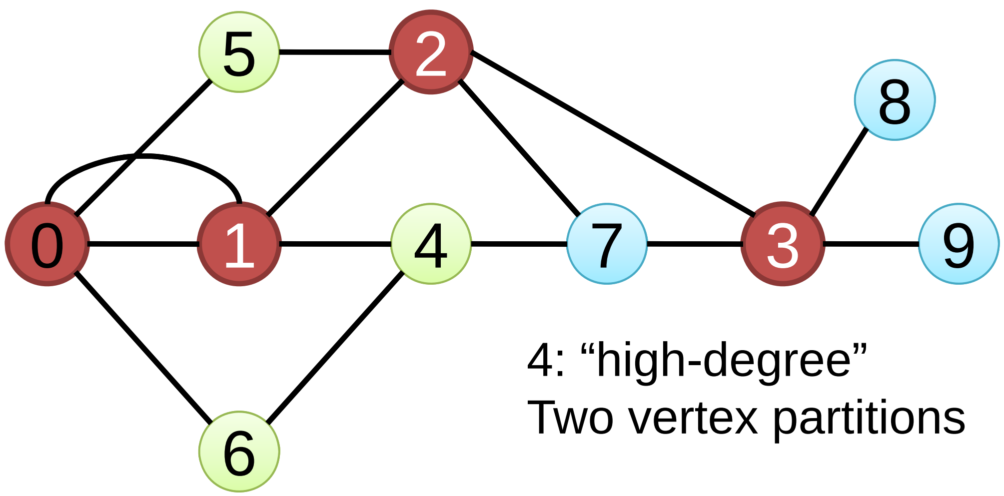
“2D” Graph Partitioning Strategy

- Tuned for graphs with unbalanced degree distributions and incremental updates
  - Sort vertices by degree
  - Form roughly M_G/M_P local communities around “high-degree” vertices & partition adjacencies
  - Reorder vertices by degree, assign contiguous chunks to each of the M_G/M_P nodes
  - Assign ownership of any remaining low-degree vertices to processes
- Comparison: 1D p-way partitioning, 1D p-way partitioning with vertices shuffled

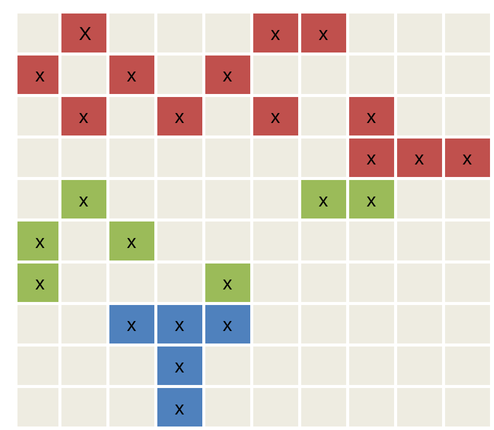
Parallel Breadth-First Search Implementation



- Expensive preprocessing partitioning + reordering step, currently untimed



4: “high-degree”
Two vertex partitions



Parallel BFS Implementation

- Concurrency in each phase limited by size of frontier array
- Local computation: inspecting adjacencies, creating a list of unvisited vertices
- Parallel communication step: All-to-all exchange of frontier vertices
  - Potentially P^2 exchanges
  - Partitioning, replication, and reordering significantly reduce number of messages

Single-node Multicore Optimizations

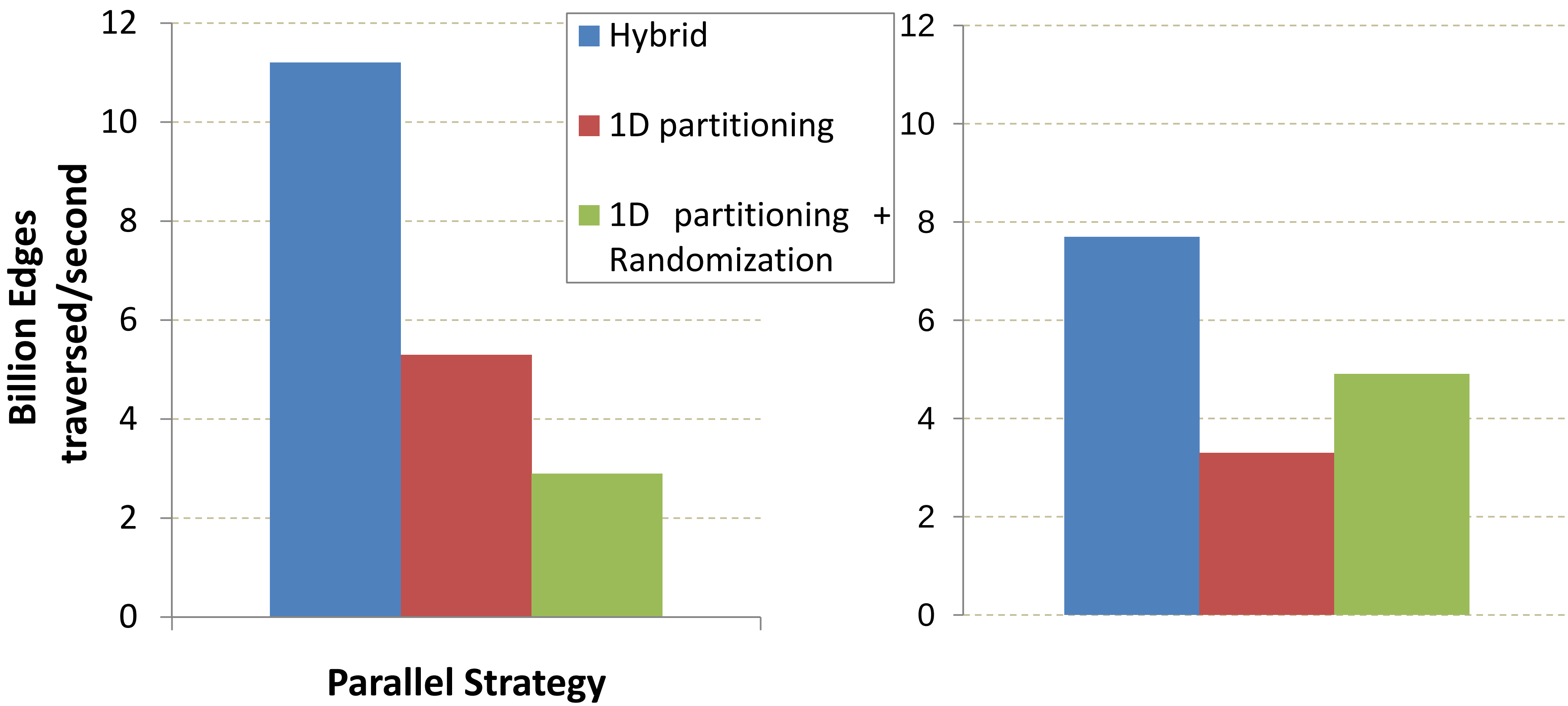
1. **Software prefetching** on Intel Nehalem (supports 32 loads and 20 stores in flight)
   - Speculative loads of **index array** and **adjacencies of frontier vertices** will reduce compulsory cache misses.
2. **Aligning adjacency lists** to optimize memory accesses
   - 16-byte aligned loads and stores 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)
5. **Hugepage** support (significant TLB miss reduction)
6. **NUMA-aware memory** allocation exploiting first-touch policy

Parallel Performance

- 32 nodes of NERSC's Carver system
  - dual-socket, quad-core Intel Nehalem 2.67 GHz processor node
  - 24 GB DDR3 1333 MHz memory per node, or roughly 768 TB aggregate memory

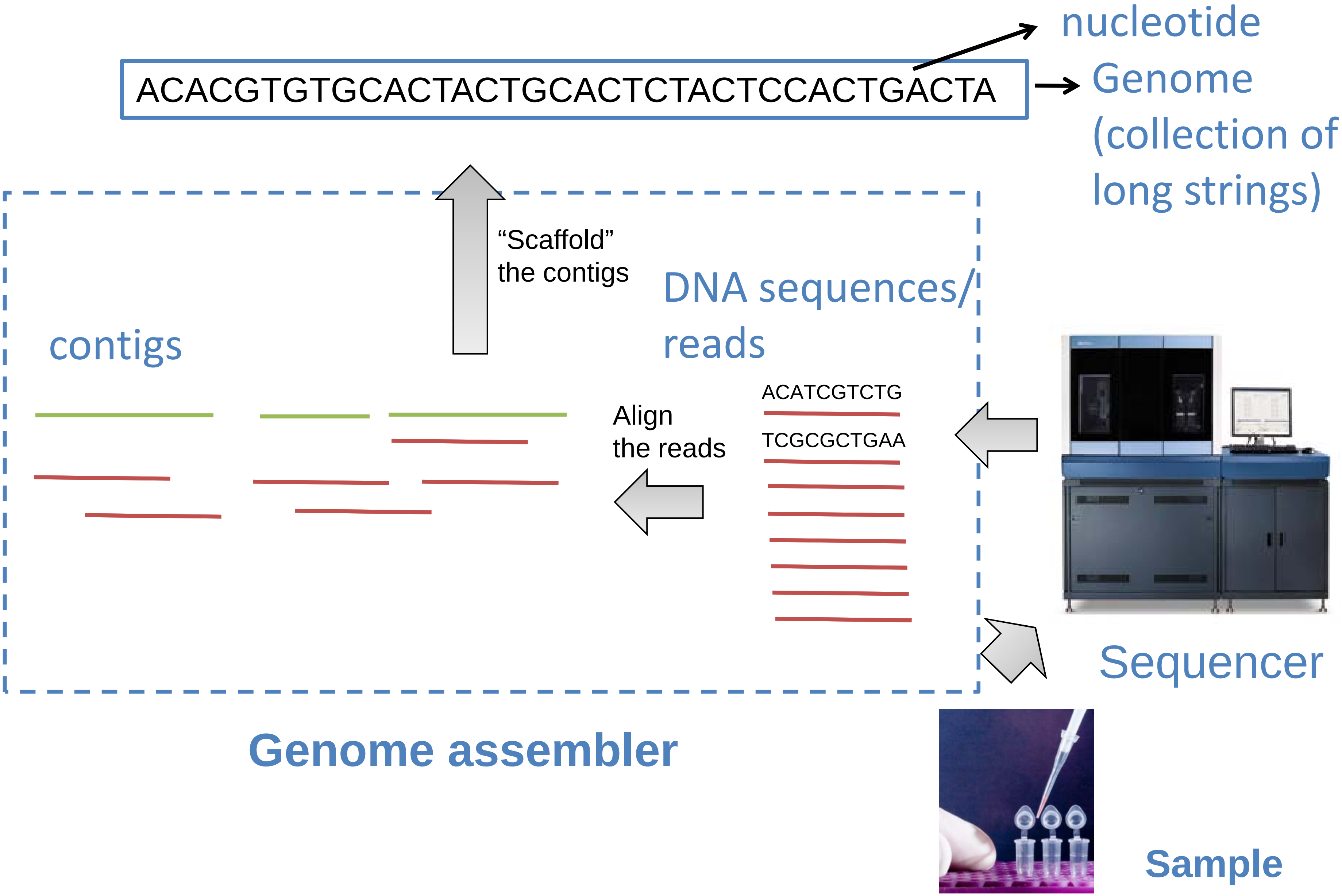
Orkut crawl: 3.07M vertices, 227M edges

Synthetic RMAT network: 2 billion vertices, 32 billion edges



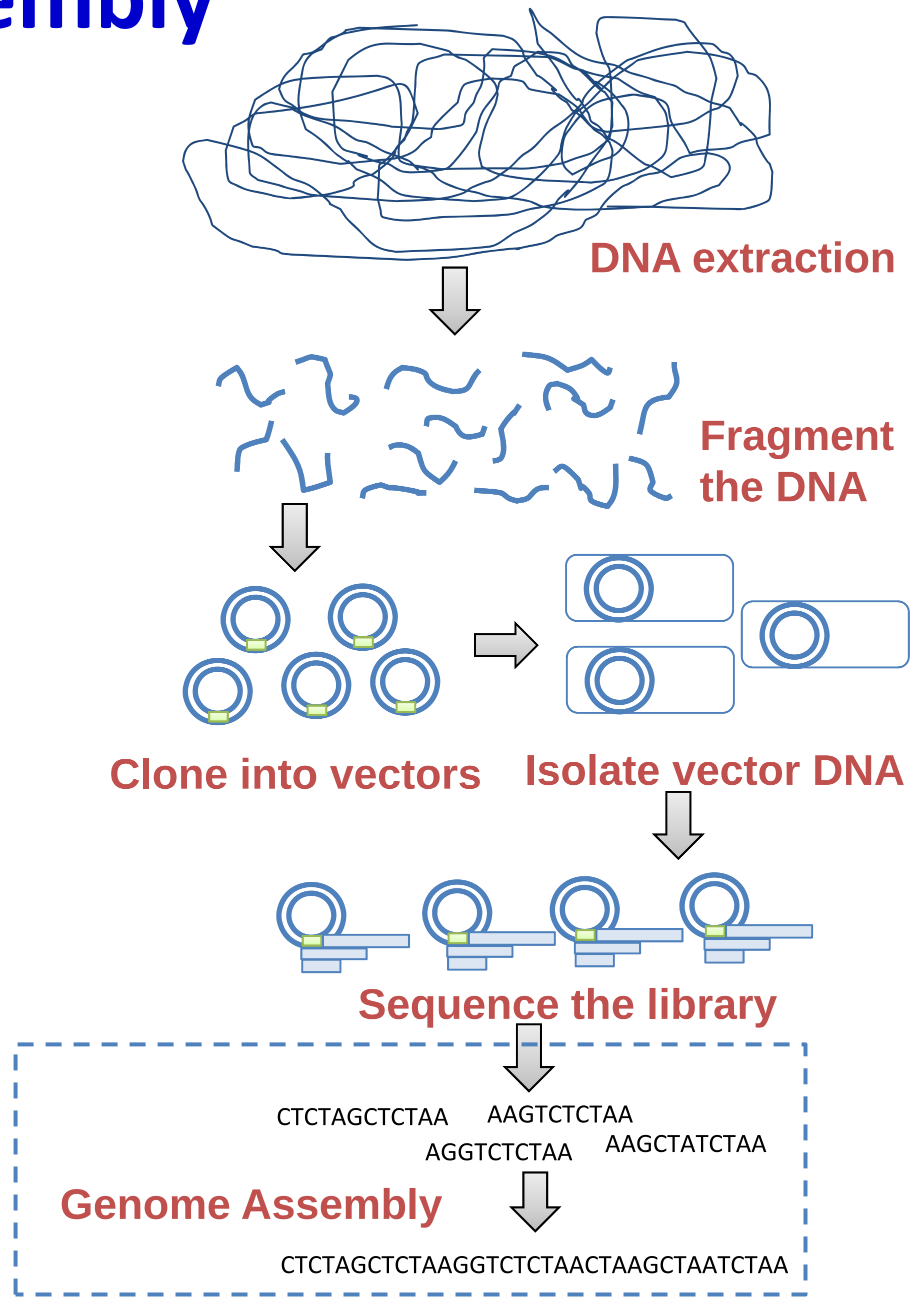
Single-node performance: 300-500 M traversed edges/second.

Genome Assembly Preliminaries



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 strategies

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

k = 5

ACGTTATATATTCTA → ACGTT CGTTA GTTAT TTATA ..... TTCTA

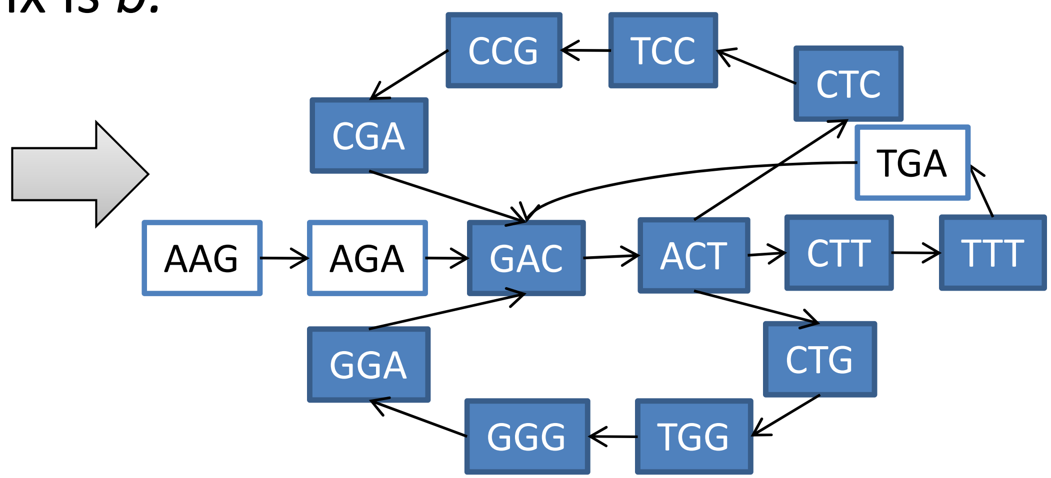
CCATGATATATTCTA → CCATG CATGA ATGAT TGATA ..... TTCTA

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

de Bruijn graphs

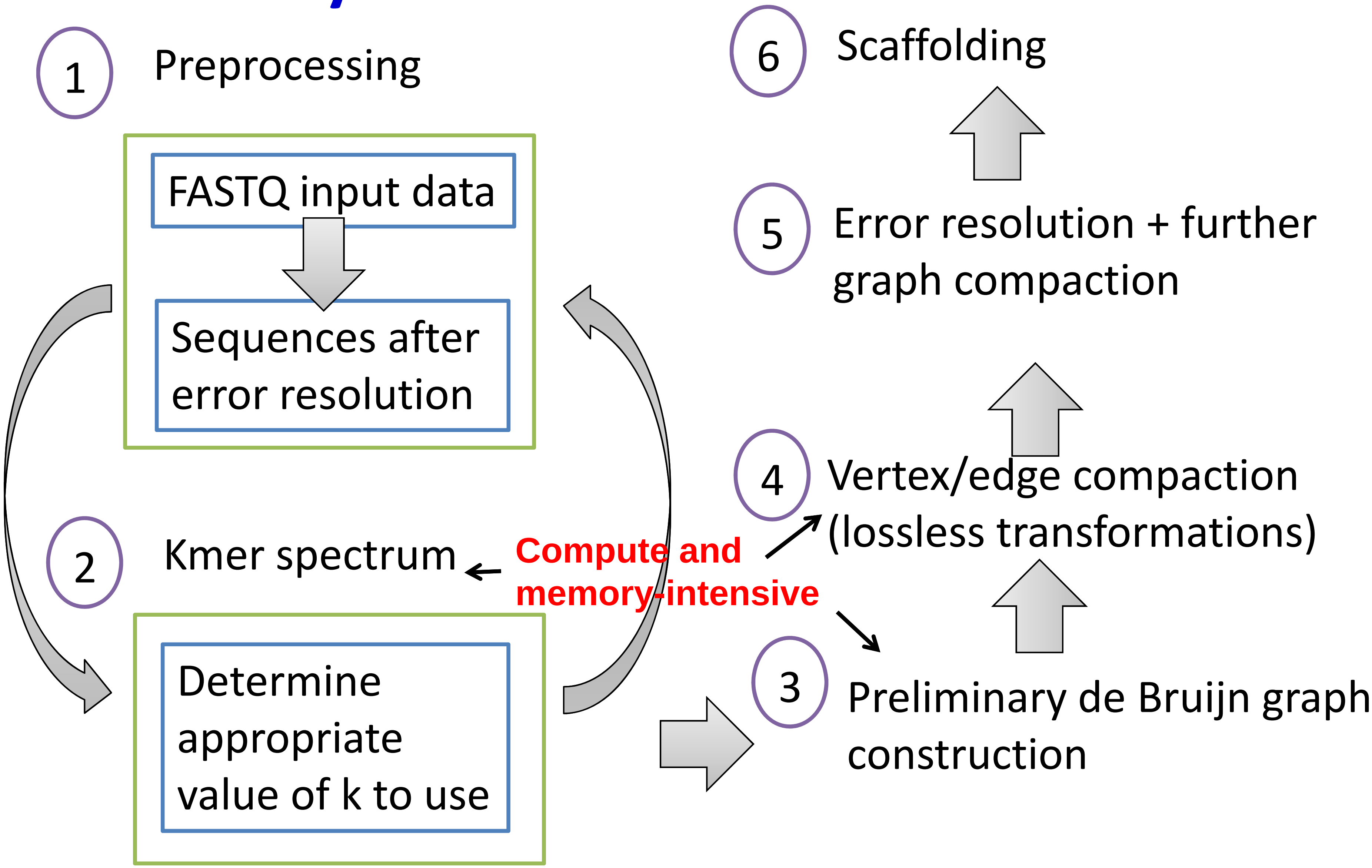
- Each *(k-1)*-mer represents a node in the graph
- Edge exists between node *a* to *b* iff there exists a *k*-mer such that its prefix is *a* and suffix is *b.*

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.

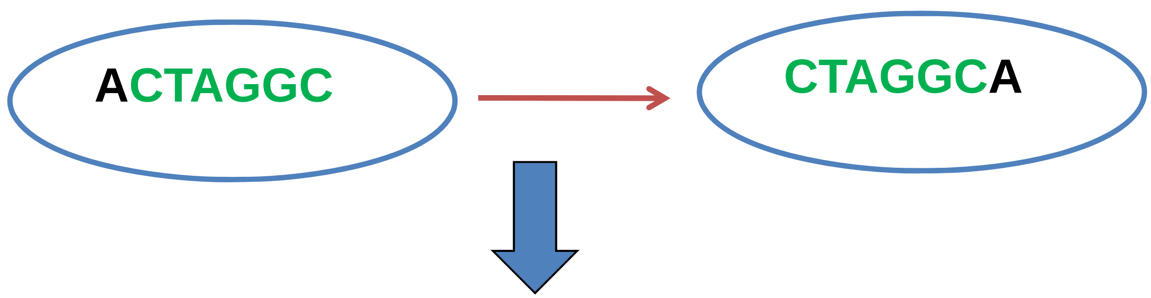
Steps in the de Bruijn graph-based assembly scheme



Graph construction

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

Read ‘x’:

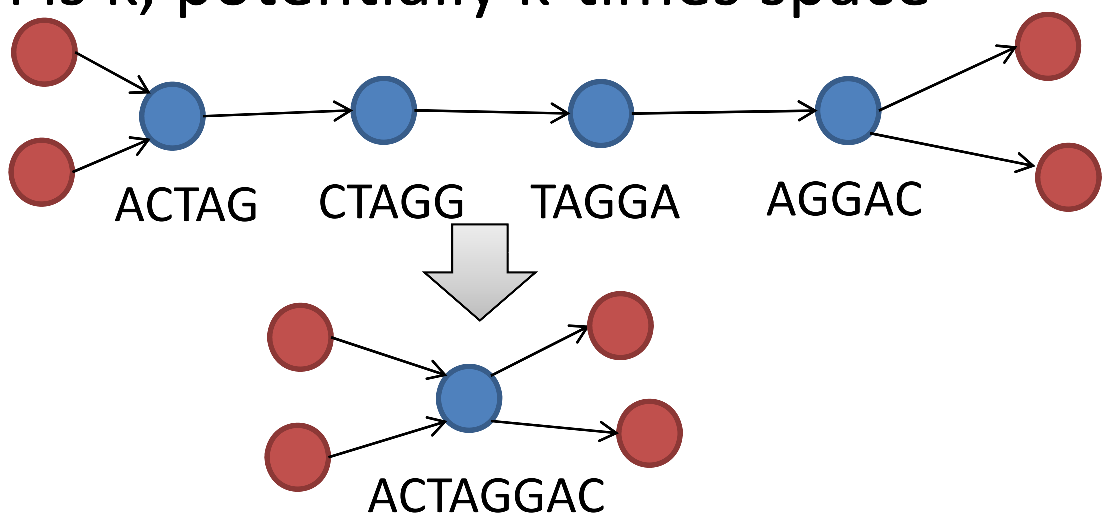


Store edge (ACTAGGCA), orientation,
edge direction, edge id (y), originating read id (x), edge count

2 bits per nucleotide

Vertex compaction

- High percentage of unique kmers
  - $\Rightarrow$ 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.

Summary of various steps and Analysis

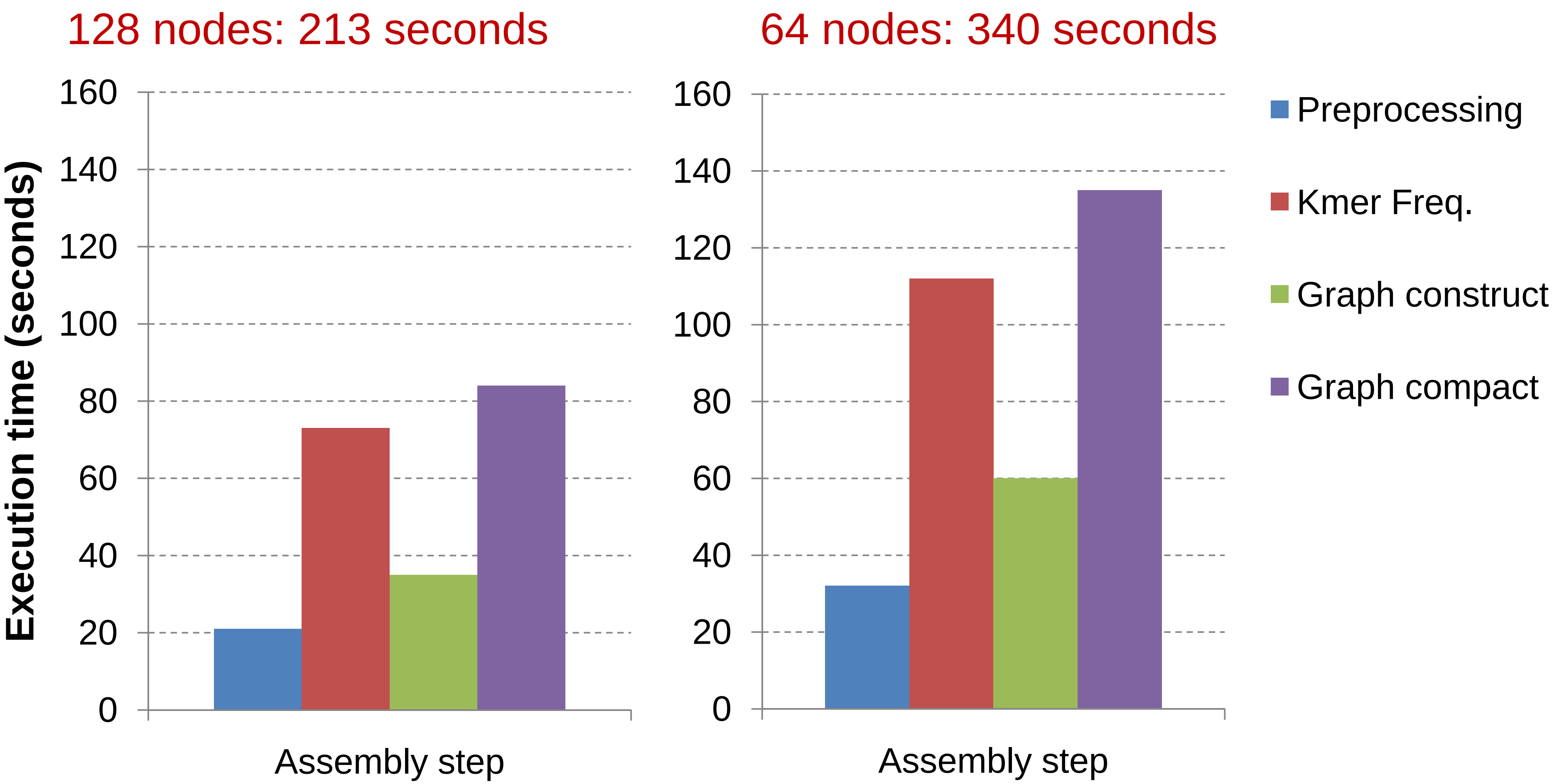
A metagenomic data set (140 million reads, 20G bp), k = 45.

Step	Memory footprint	Approach used	Parallelism & Computational kernels
1. Preprocessing	minimal	Streaming file read and write, kmer merging	"Pleasantly parallel", I/O-intensive
2. Kmer spectrum	~ 200 GB	3 local sorts, 1 AlltoAll communication steps.	Parallel sort, AlltoAllv
3. Graph construction	~ 320 GB	Two sorts	Fully local computation
4. Graph compaction	~ 60 GB	3+ local sorts, 2 AlltoAll communication steps + local graph traversal	AlltoAllv + local computation
5. Error detection	~ 35 GB	Connected components + AlltoAll	Intensive communication
6. Scaffolding	? GB	Euler tours over components	Mostly local computation

Parallel Implementation Details

- Data set under consideration requires 320 GB for in-memory processing
  - NERSC Franklin system [Cray XT4, 2.3 GHz quad-core Opterons, 8 GB memory per node]
  - Experimented with 64 nodes (256-way parallelism) and 128 nodes (512-way)
- MPI across nodes + OpenMP within a node
- Local sort: multicore-parallel radix sort
- Global sort: bin data in parallel + AlltoAll comm. + local sort + AlltoAll comm.

Parallel Performance



- Comparison: *Velvet* (open-source serial code) takes ~ 12 hours on a 500 GB machine.

Talk Summary

- Two examples of “hybrid” parallel programming for analyzing large-scale graphs
  - Up to 3x faster with hybrid approaches on 32 nodes
- Two different types of graphs, the strategies to achieve high performance differs
  - Social and information networks: low diameter, difficult to generate balanced partitions with low edge cuts
  - DNA fragment string graph: O(N) diameter, multiple connected components
- Single-node multicore optimizations + communication optimization (reducing data volume and number of messages in All-to-all exchange).

Acknowledgments

Thank you!

Questions?

KMadduri@lbl.gov