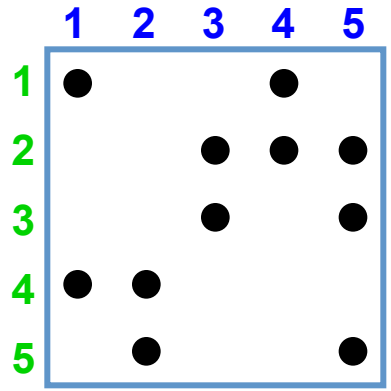
Faster parallel GraphBLAS kernels and new graph algorithms in matrix algebra

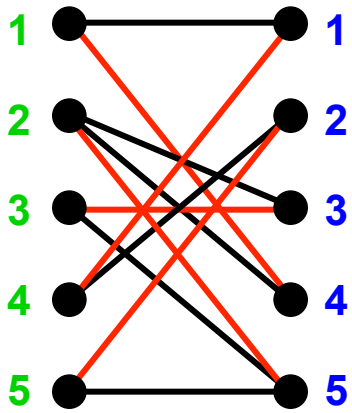
Aydın Buluç
Computational Research Division
Berkeley Lab (LBNL)

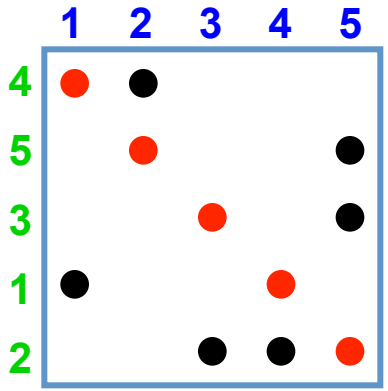
EECS, University of California, Berkeley
October 14, 2016

Large Graphs in Scientific Discoveries



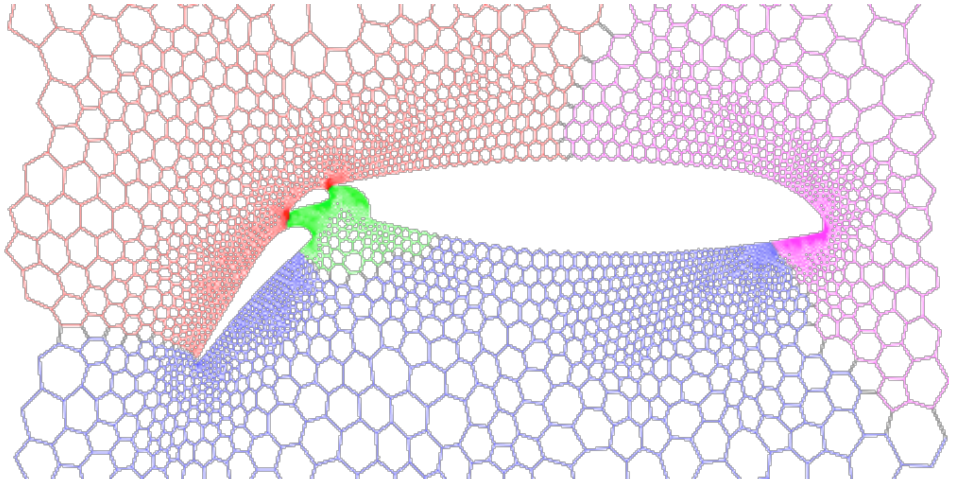
A





PA

Matching in bipartite graphs: Permuting to heavy diagonal or block triangular form

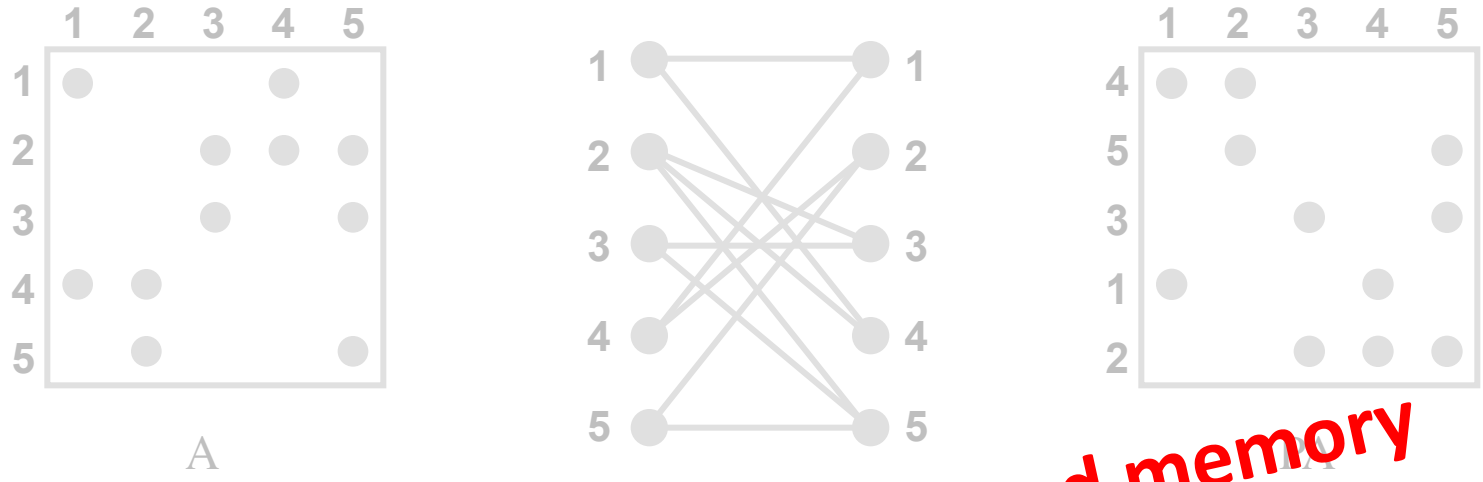


Graph partitioning: *Dynamic load balancing* in parallel simulations

Picture (left) credit: Sanders and Schulz

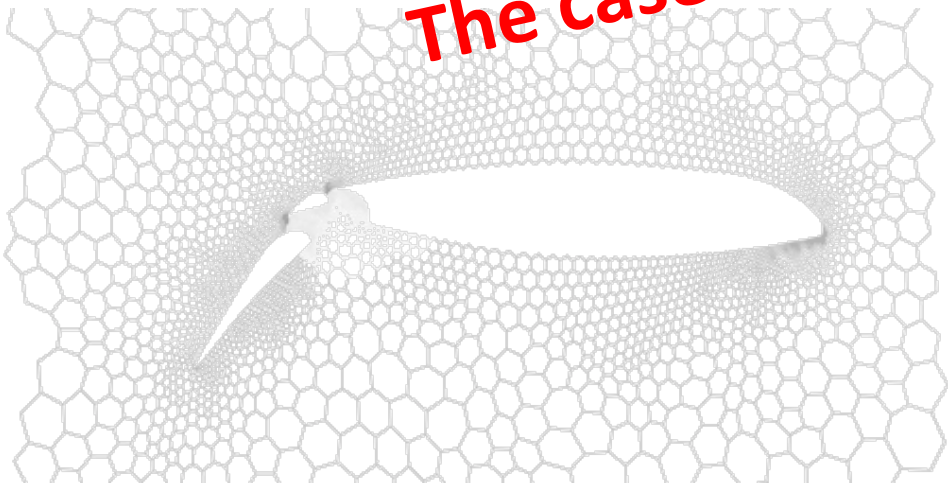
Problem size: as big as the sparse linear system to be solved or the simulation to be performed

Large Graphs in Scientific Discoveries



Matching in bipartite graphs: Permuting to heavy diagonal or block triangular form

The case for distributed memory



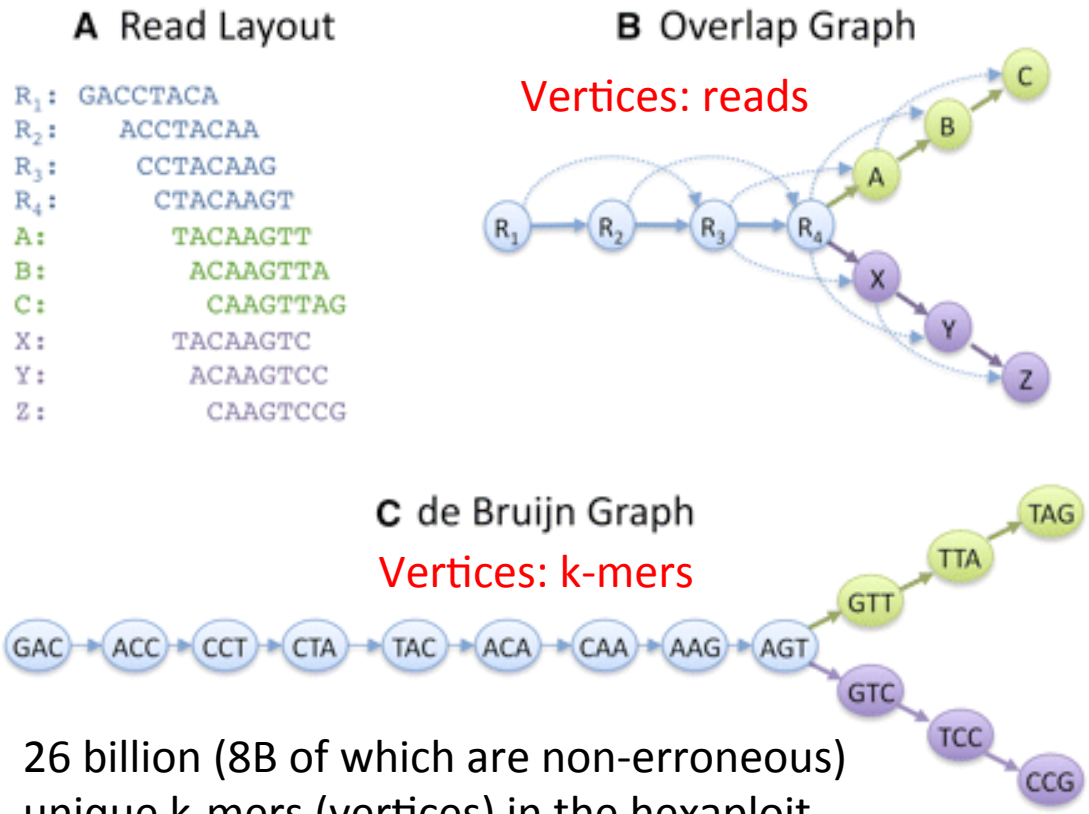
Graph partitioning: *Dynamic load balancing* in parallel simulations

Picture (left) credit: Sanders and Schulz

Problem size: as big as the sparse linear system to be solved or the simulation to be performed

Large Graphs in Scientific Discoveries

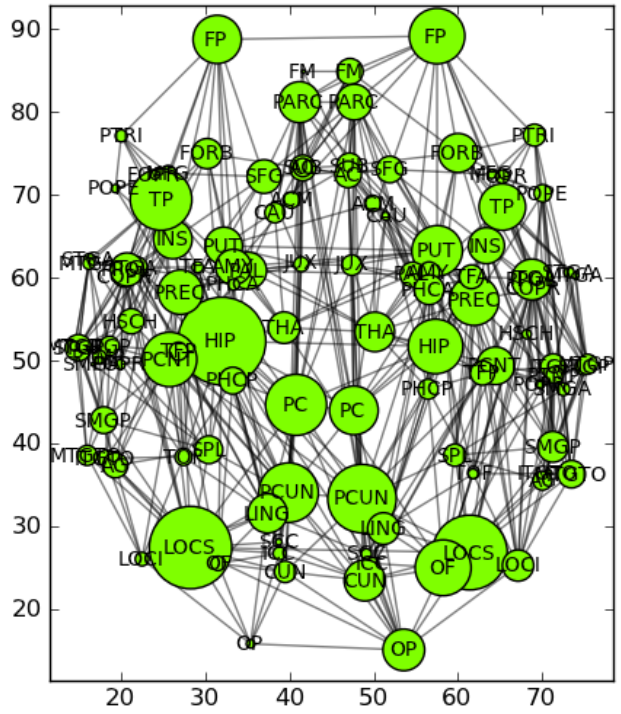
Whole genome assembly



26 billion (8B of which are non-erroneous) unique k-mers (vertices) in the hexaploit wheat genome W7984 for k=51

Schatz et al. (2010) Perspective: Assembly of Large Genomes w/2nd-Gen Seq. Genome Res. (figure reference)

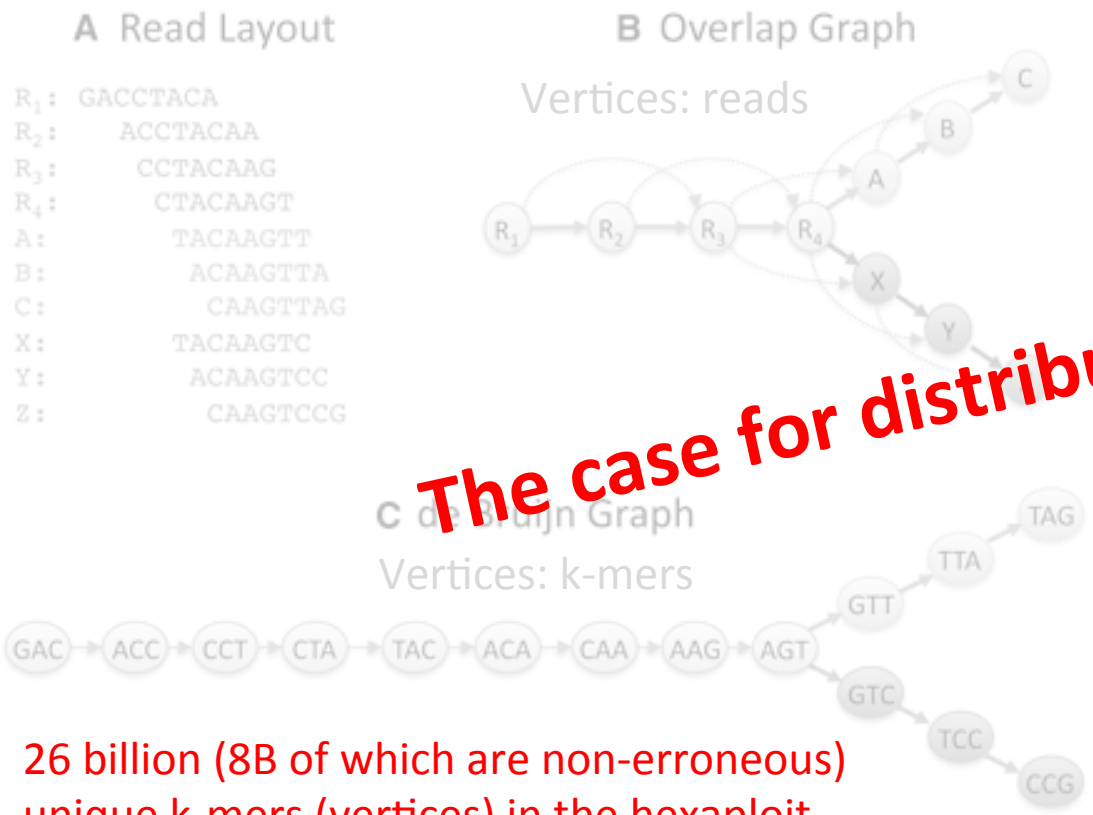
Graph Theoretical analysis of Brain Connectivity



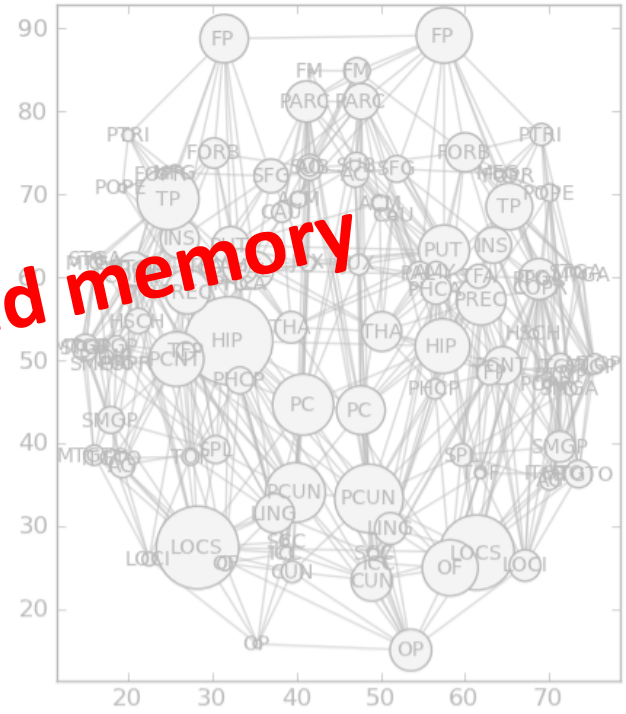
Potentially millions of neurons and billions of edges with developing technologies

Large Graphs in Scientific Discoveries

Whole genome assembly



Graph Theoretical analysis of Brain Connectivity



The case for distributed memory

26 billion (8B of which are non-erroneous) unique k-mers (vertices) in the hexaploit wheat genome W7984 for k=51

Potentially millions of neurons and billions of edges with developing technologies

Schatz et al. (2010) Perspective: Assembly of Large Genomes w/2nd-Gen Seq. Genome Res. (figure reference)

Outline

- **GraphBLAS:** standard building blocks for graph algorithms in the language of linear algebra
- **SpGEMM**: Computing the sparse matrix-matrix multiplication in parallel
- **Triangle counting/enumeration** in matrix algebra
- **Bipartite graph matching** in parallel
- **Other contributions** and **future work**

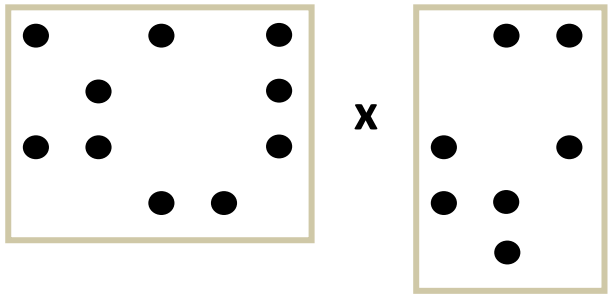
The case for sparse matrices

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

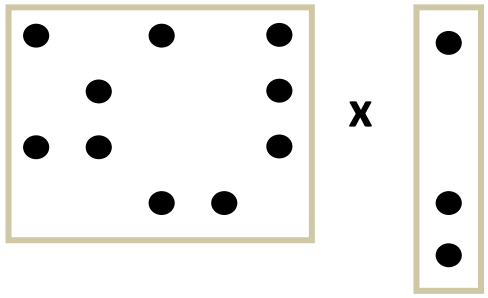
Traditional graph computations	Graphs in the language of linear algebra
Data driven, unpredictable communication.	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse grained parallelism, bandwidth limited

Linear-algebraic primitives for graphs

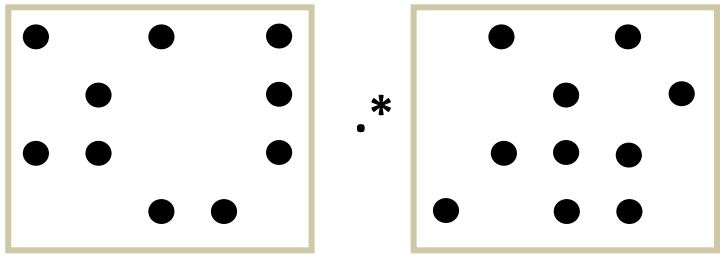
Sparse matrix X sparse matrix



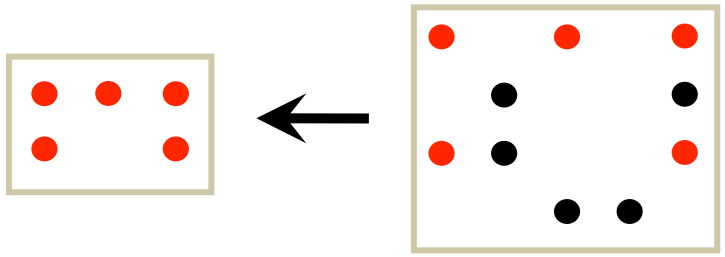
Sparse matrix X sparse vector



Element-wise operations



Sparse matrix indexing



Is **think-like-a-vertex** really more productive?

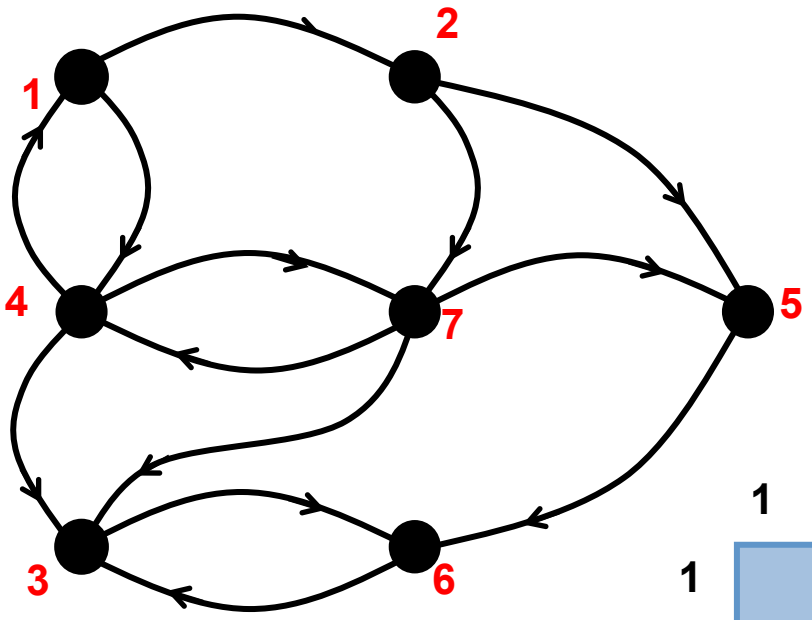
"Our mission is to build up a linear algebra sense to the extent that vector-level thinking becomes as natural as scalar-level thinking."

- Charles Van Loan

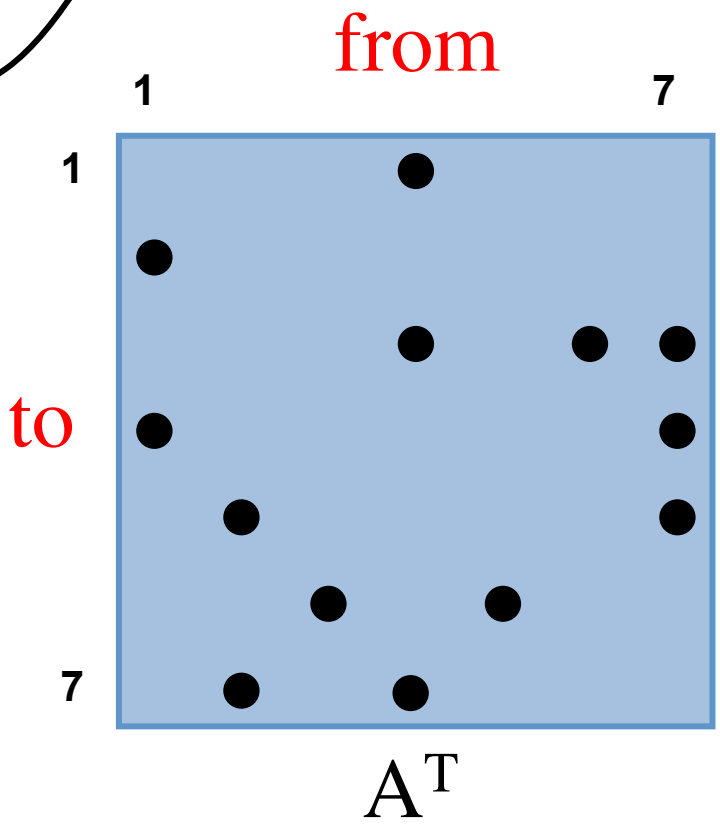
Examples of semirings in graph algorithms

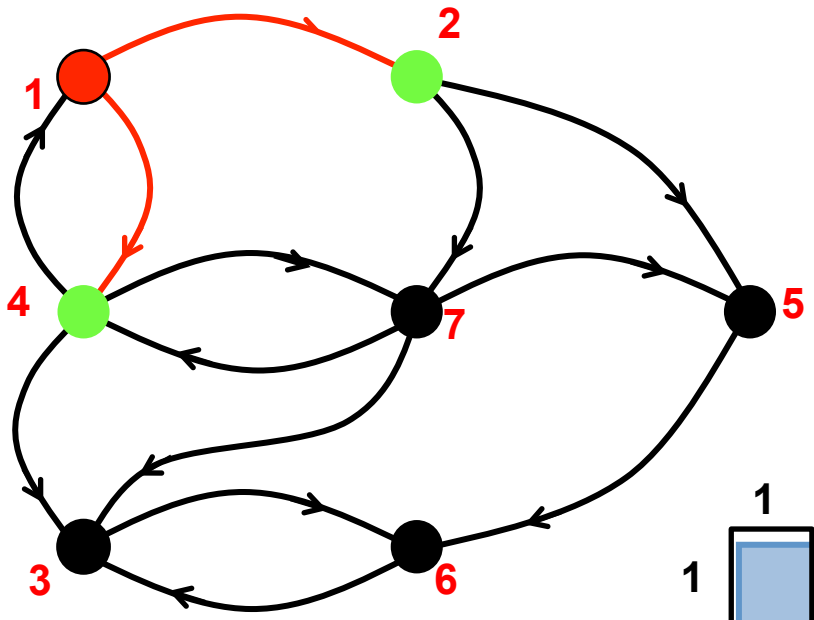
Real field: **(R, +, x)**	Classical numerical linear algebra
Boolean algebra: **({0 1}, |, &)**	Graph traversal
Tropical semiring: **(R U {∞}, min, +)**	Shortest paths
(S, select, select)	Select subgraph, or contract nodes to form quotient graph
(edge/vertex attributes, vertex data aggregation, edge data processing)	Schema for user-specified computation at vertices and edges
(R, max, +)	Graph matching &network alignment
(R, min, times)	Maximal independent set

- **Shortened semiring notation: (Set, Add, Multiply)**. Both identities omitted.
- **Add:** Traverses edges, **Multiply:** Combines edges/paths at a vertex
- Neither add nor multiply needs to have an inverse.
- Both **add** and **multiply** are **associative**, **multiply distributes** over **add**



Breadth-first search in the language of matrices

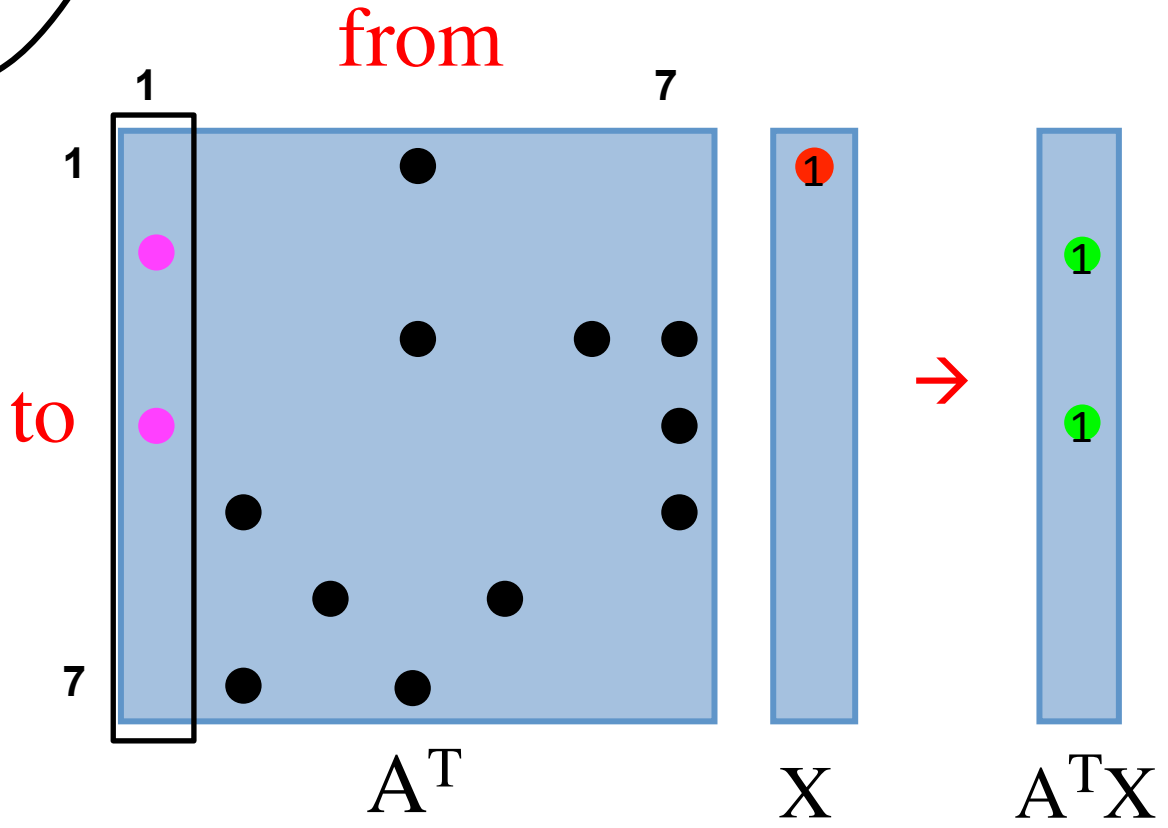




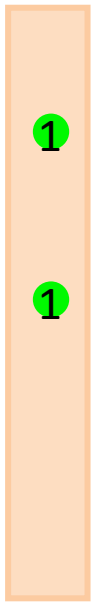
Particular semiring operations:

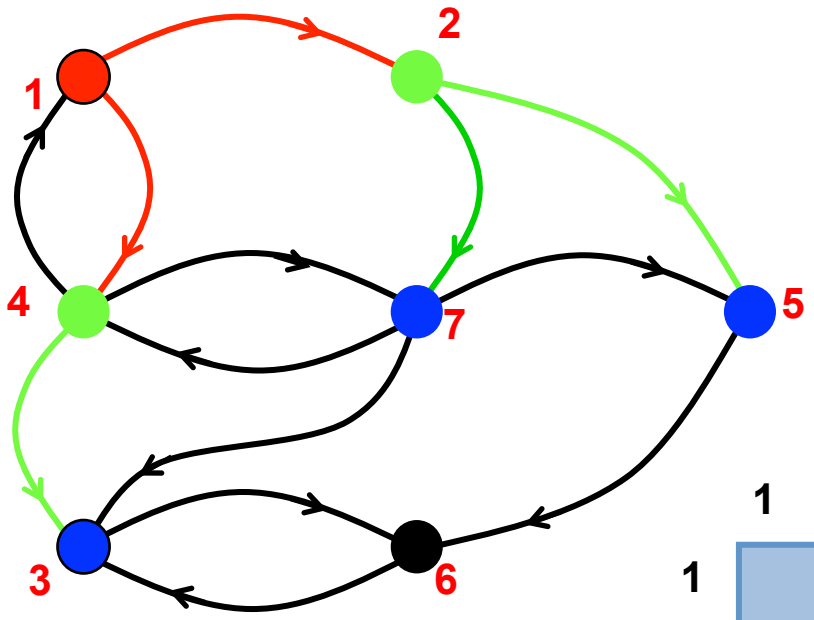
Multiply: select

Add: minimum



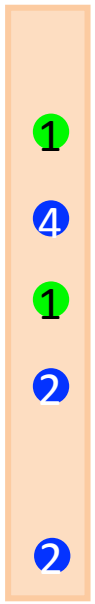
parents:





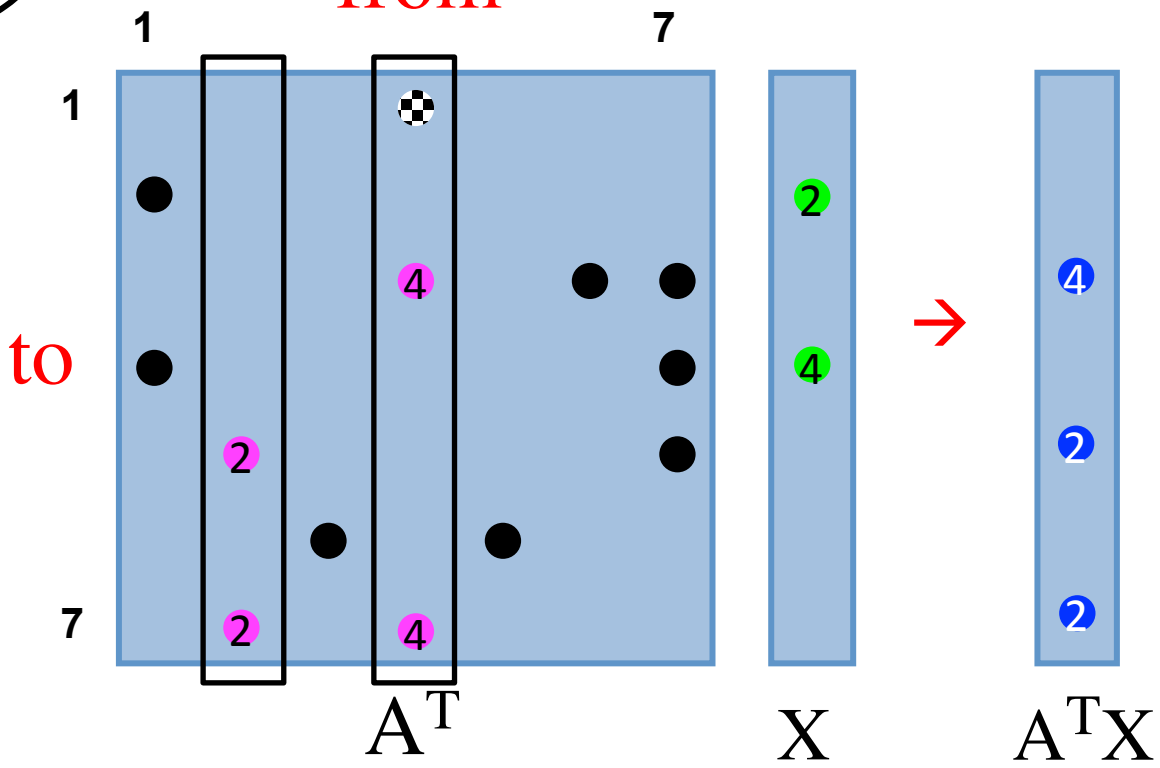
Select vertex with
minimum label as parent

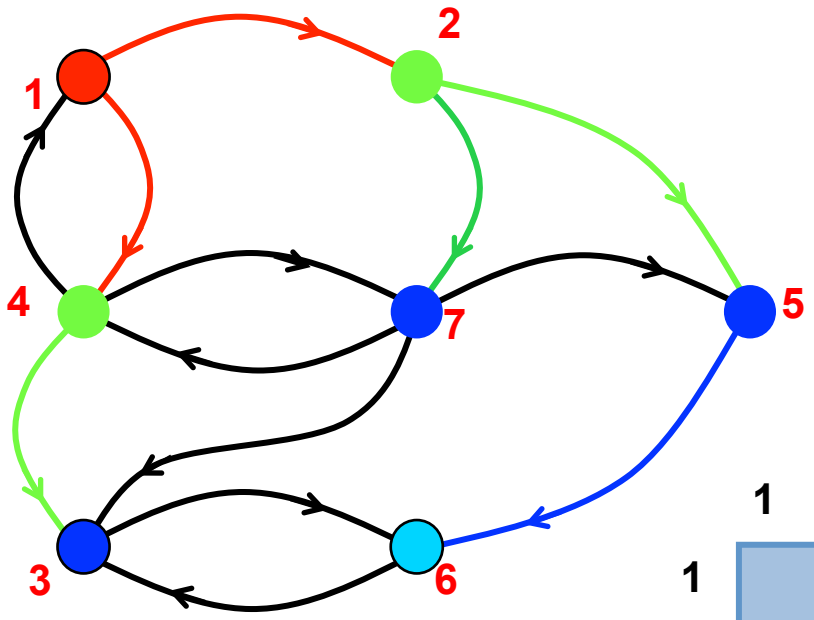
parents:



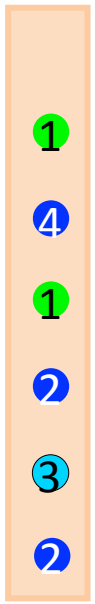
from

to



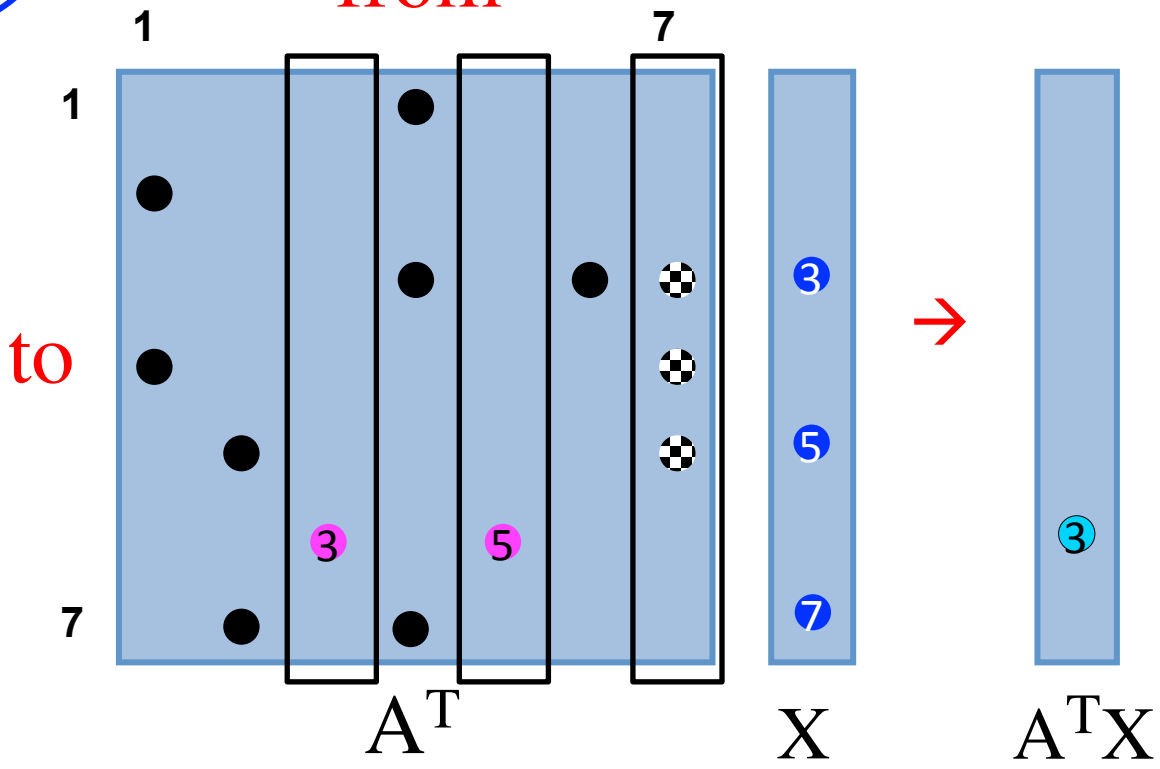


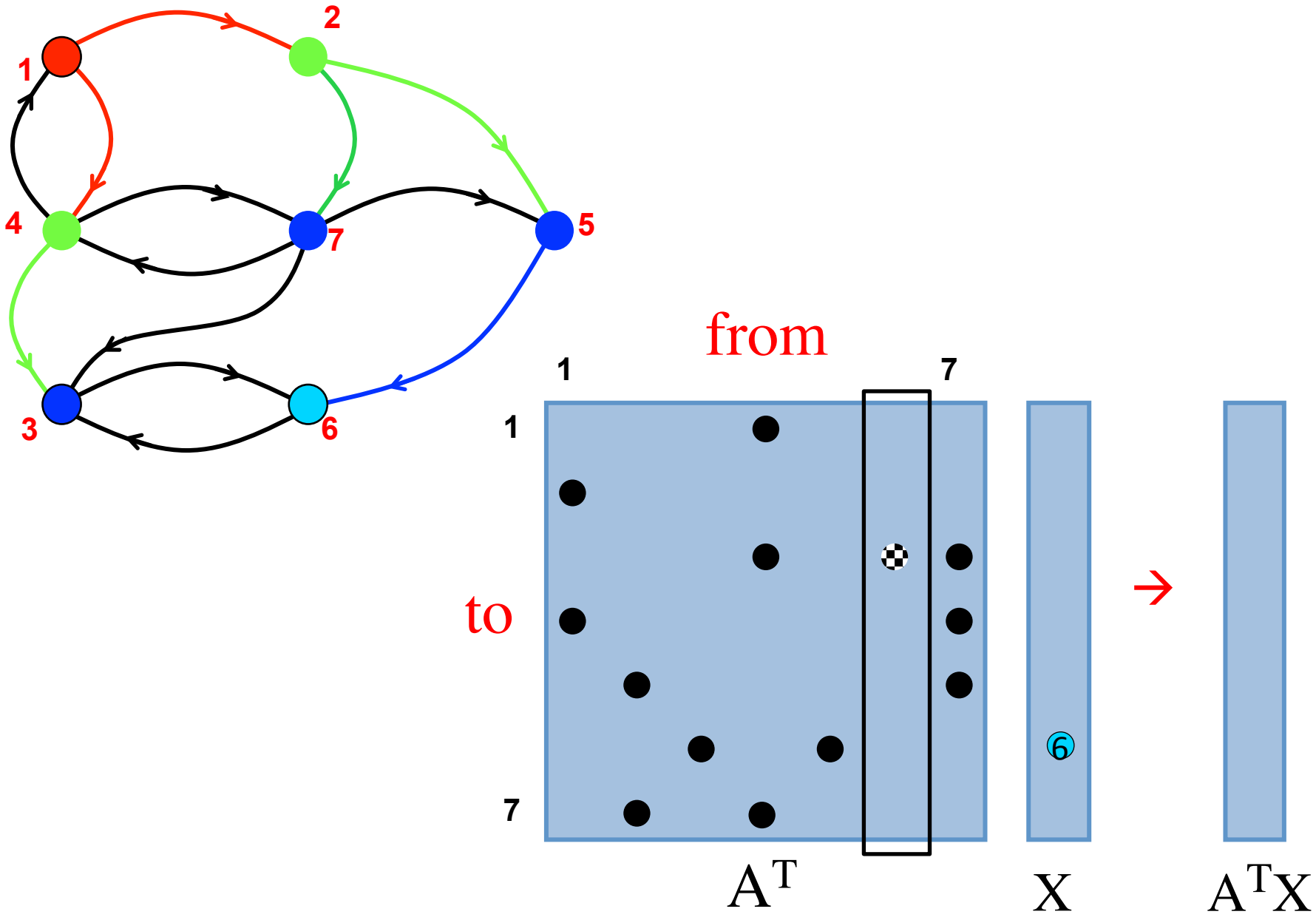
parents:



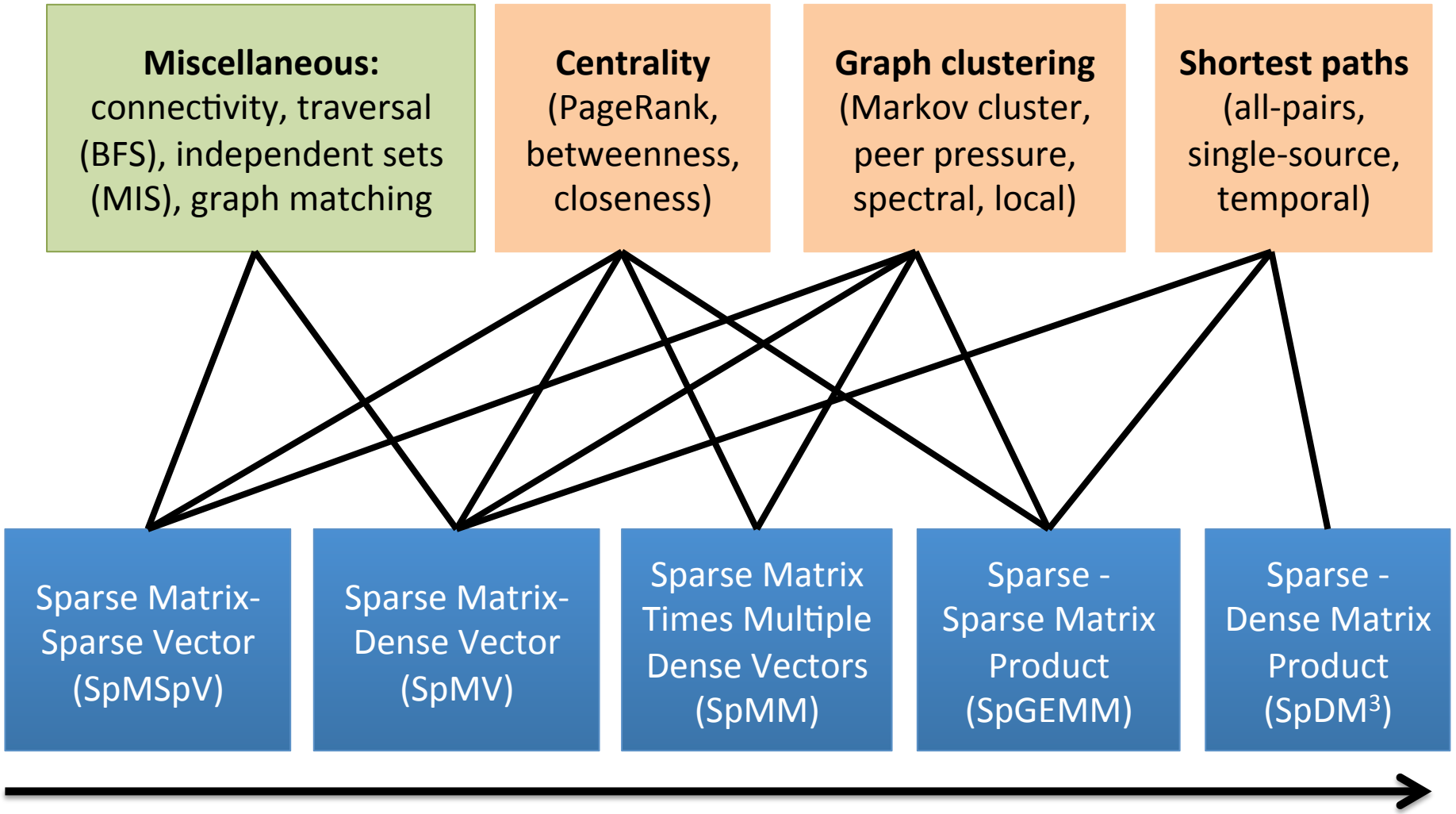
from

to




1
2
3
4
5
6
7
from
to
1
7
1
7
6
A^T
X
A^TX

Graph Algorithms on GraphBLAS



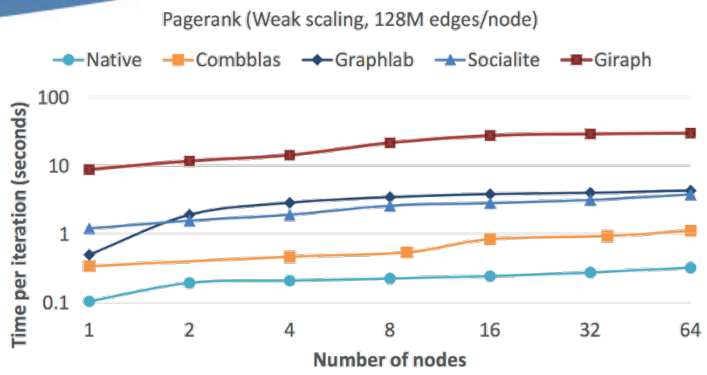
Some Graph BLAS basic functions

Function (CombBLAS equiv)	Parameters	Returns	Matlab notation
MxM (SpGEMM)	- sparse matrices **A** and **B** - optional unary functs	sparse matrix	**C** = **A** * **B**
MxV (SpM{Sp}V)	- sparse matrix **A** - sparse/dense vector **x**	sparse/dense vector	**y** = **A** * **x**
ewisemult, add, … (SpEWiseX)	- sparse matrices or vectors - binary funct, optional unarys	in place or sparse matrix/vector	**C** = **A** .* **B** **C = A + B**
reduce (Reduce)	- sparse matrix **A** and funct	dense vector	**y** = sum(**A**, op)
extract (SpRef)	- sparse matrix **A** - index vectors **p** and **q**	sparse matrix	**B** = **A**(**p**, **q**)
assign (SpAsgn)	- sparse matrices **A** and **B** - index vectors **p** and **q**	none	**A**(**p**, **q**) = **B**
buildMatrix (Sparse)	- list of edges/triples **(i, j, v)**	sparse matrix	**A = sparse(i, j, v, m, n)**
extractTuples (Find)	- sparse matrix **A**	edge list	**[i, j, v] = find(A)**

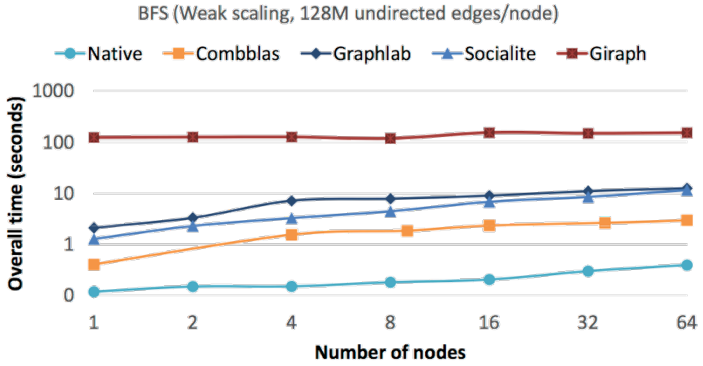
Performance of Linear Algebraic Graph Algorithms

Combinatorial BLAS fastest among all tested graph processing frameworks on 3 out of 4 benchmarks in an independent study by Intel.

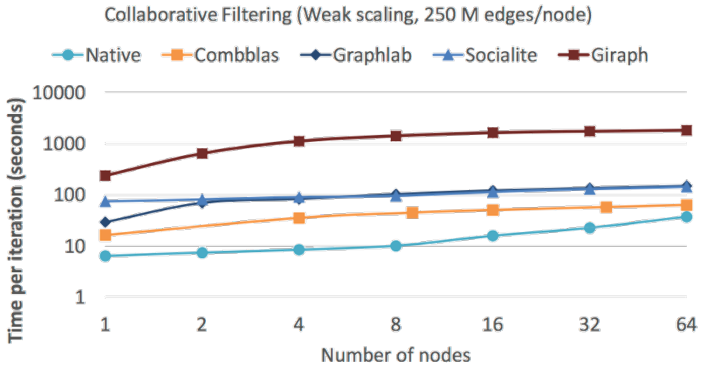
The linear algebra abstraction enables high performance, within 4X of native performance for PageRank and Collaborative filtering.



(a) PageRank



(b) Breadth-First Search



(c) Collaborative Filtering

Satish, Nadathur, et al. "Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets", in SIGMOD'14

The Graph BLAS effort

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

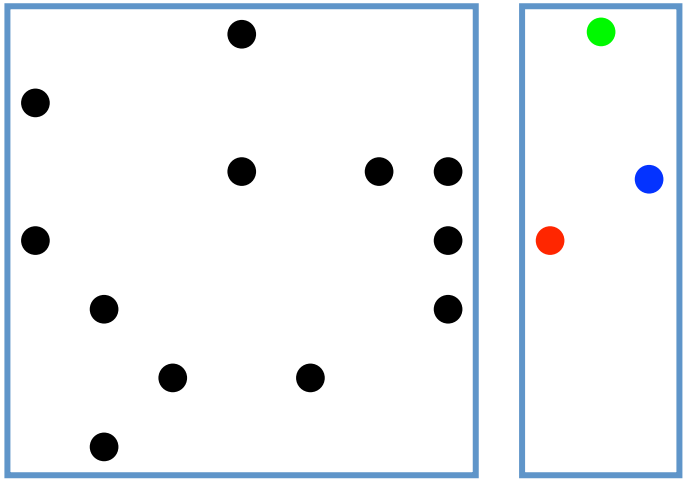
Abstract-- It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitive building blocks. This paper is a position paper defining the problem and announcing our intention to launch an open effort to define this standard.

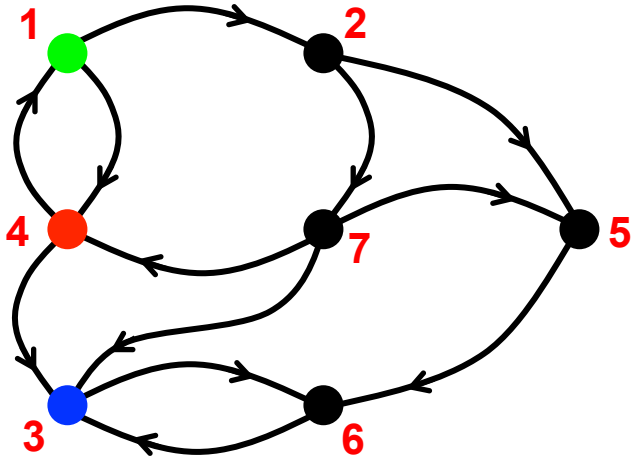
- The GraphBLAS Forum: http://graphblas.org
- IEEE Workshop on Graph Algorithms Building Blocks (at IPDPS): http://www.graphanalysis.org/workshop2017.html

Outline

- **GraphBLAS:** standard building blocks for graph algorithms in the language of linear algebra
- **SpGEMM**: Computing the sparse matrix-matrix multiplication in parallel
- **Triangle counting/enumeration** in matrix algebra
- **Bipartite graph matching** in parallel
- Other contributions and future work

Multiple-source breadth-first search

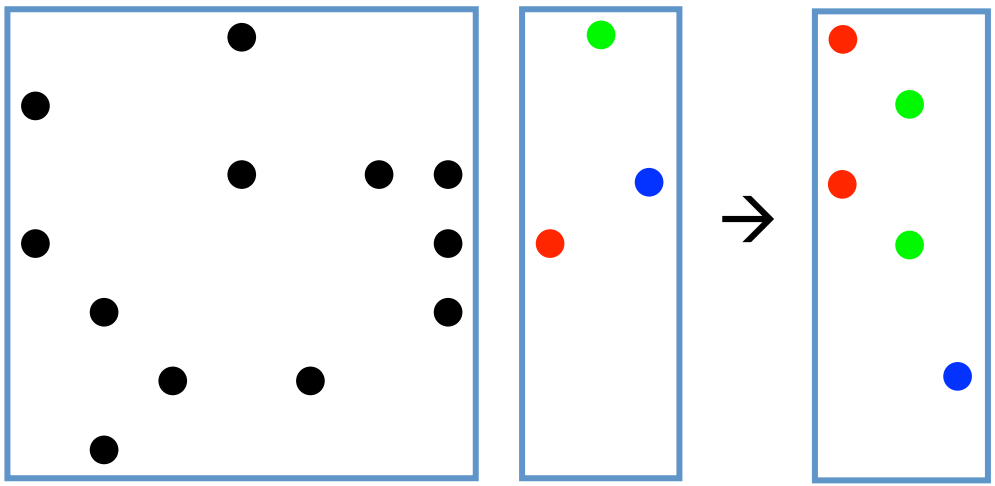


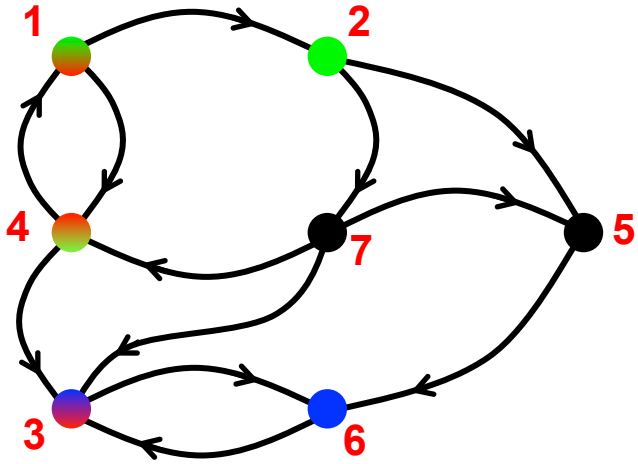


- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
- Highly-parallel implementation for Betweenness Centrality*

*: A measure of influence in graphs, based on shortest paths

Multiple-source breadth-first search





- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
- Highly-parallel implementation for Betweenness Centrality*

*: A measure of influence in graphs, based on shortest paths

Sparse Matrix-Matrix Multiplication

Why sparse matrix-matrix multiplication?

- Used for algebraic multigrid, graph clustering, betweenness centrality, graph contraction, subgraph extraction, cycle detection, quantum chemistry, high-dimensional similarity search, …

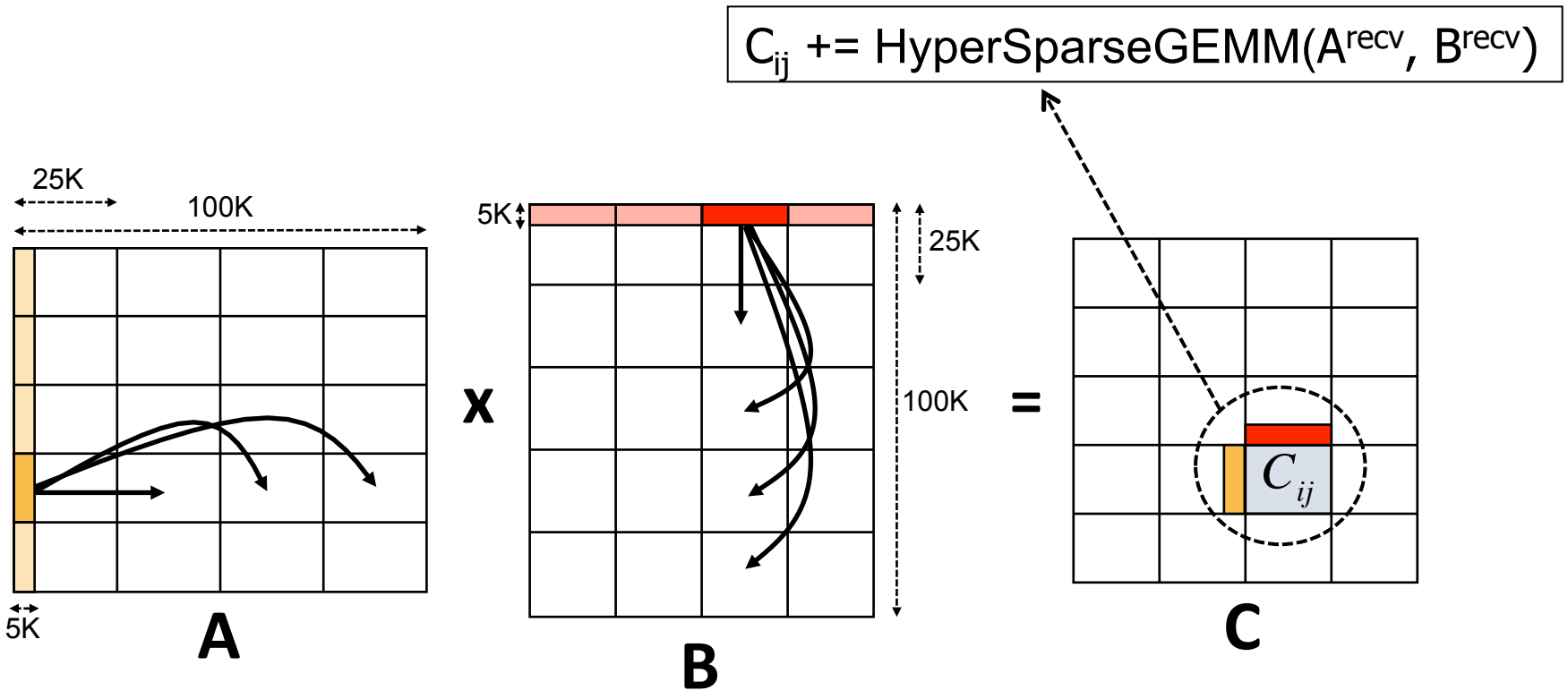
How do dense and sparse GEMM compare?

Dense:	**Sparse:**
Lower bounds match algorithms.	Significant gap
Allows extensive data reuse	Inherent poor reuse?

What do we obtain here?

- Improved (higher) lower bound
- New optimal algorithms
- **Only for random matrices**
  - Erdős-Rényi(n,d) graphs aka $G(n, p=d/n)$

2D Algorithm: Sparse SUMMA

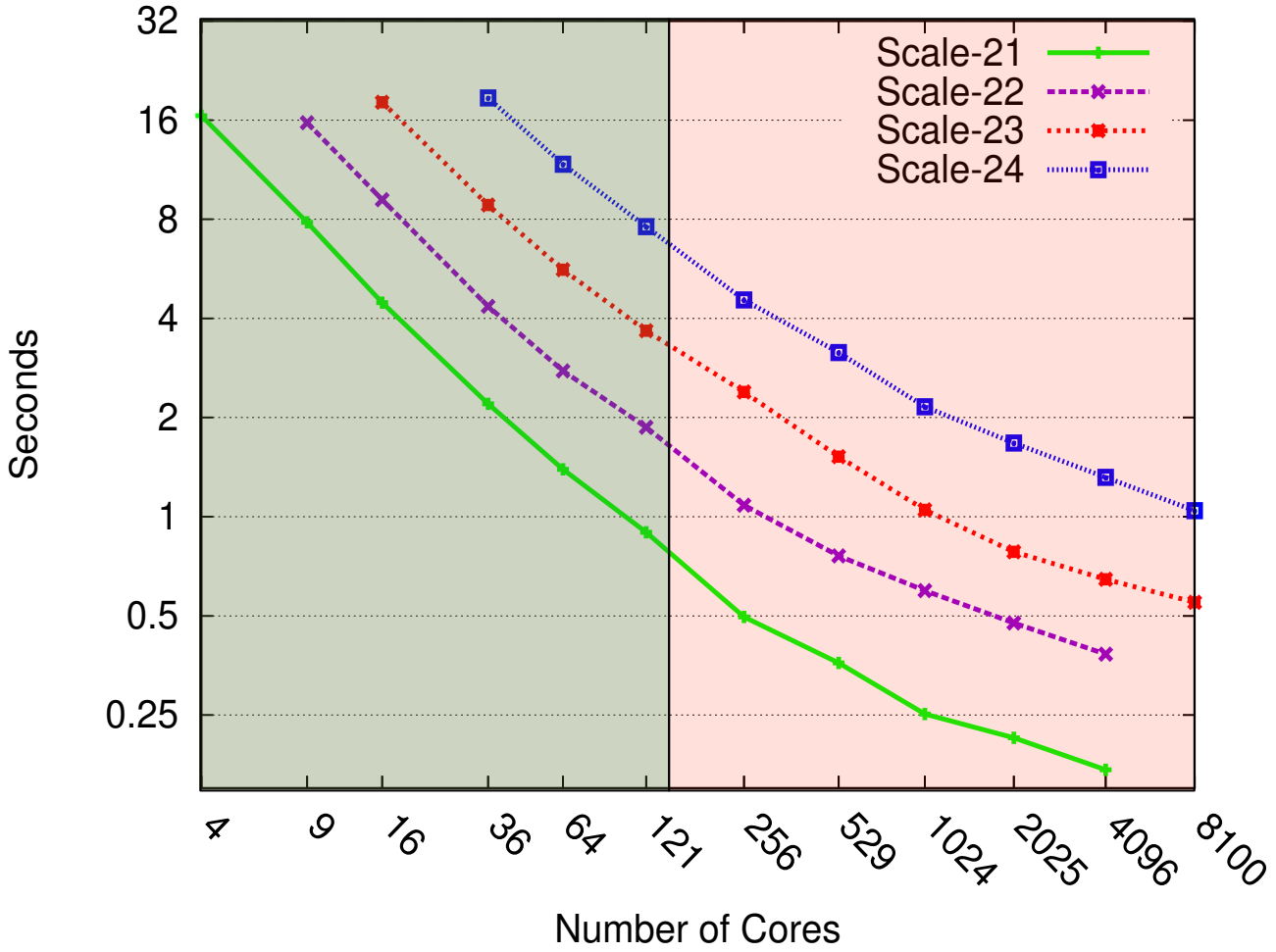


2D algorithm: Sparse SUMMA (based on dense SUMMA)

General implementation that handles rectangular matrices

2D Sparse SUMMA on square inputs

Almost linear scaling until bandwidth costs starts to dominate



Scaling proportional to √p afterwards

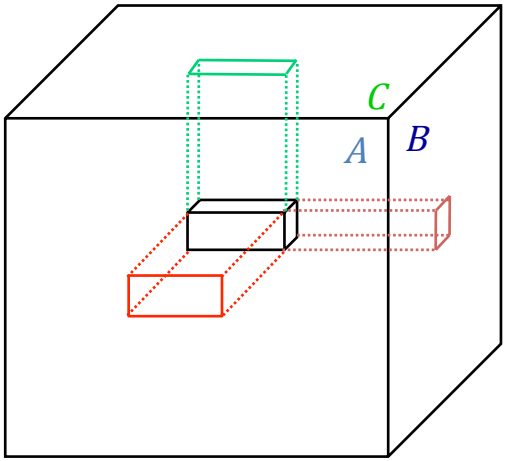
NERSC/Franklin
Cray XT4

R-MAT, edgefactor: 8
a=0.6, b=c=d=0.4/3

Aydin Buluç and John R. Gilbert. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal of Scientific Computing (SISC), 2012*.

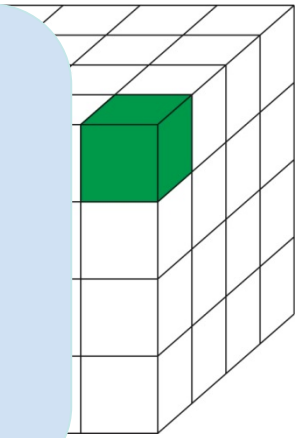
The computation cube and sparsity-independent algorithms

Matrix multiplication: $\forall(i,j) \in n \times n, \quad C(i,j) = \Sigma_k A(i,k)B(k,j),$



Sparsity independent algorithms:
assigning grid-points to processors is independent of sparsity structure.
- In particular: if C_{ij} is non-zero, who holds it?
- all standard algorithms are sparsity independent

Assumptions:
- Sparsity independent algorithms
- input (and output) are sparse:
- The algorithm is load balanced



1D algorithms 2D algorithms 3D algorithms

Algorithms attaining lower bounds

Previous Sparse Classical:

$$\Omega\left(\frac{\#FLOPs}{\left(\sqrt{M}\right)^3}\cdot\frac{M}{P}\right) = \Omega\left(\frac{d^2n}{P\sqrt{M}}\right)$$

✗ No algorithm attain this bound!

[Ballard, et al. SIMAX'11]

New Lower bound for Erdős-Rényi(n,d) :

$$\Omega\left(\min\left\{\frac{dn}{\sqrt{P}},\frac{d^2n}{P}\right\}\right)$$ ✓

[here] Expected
(Under some technical assumptions)

No previous algorithm attain these.

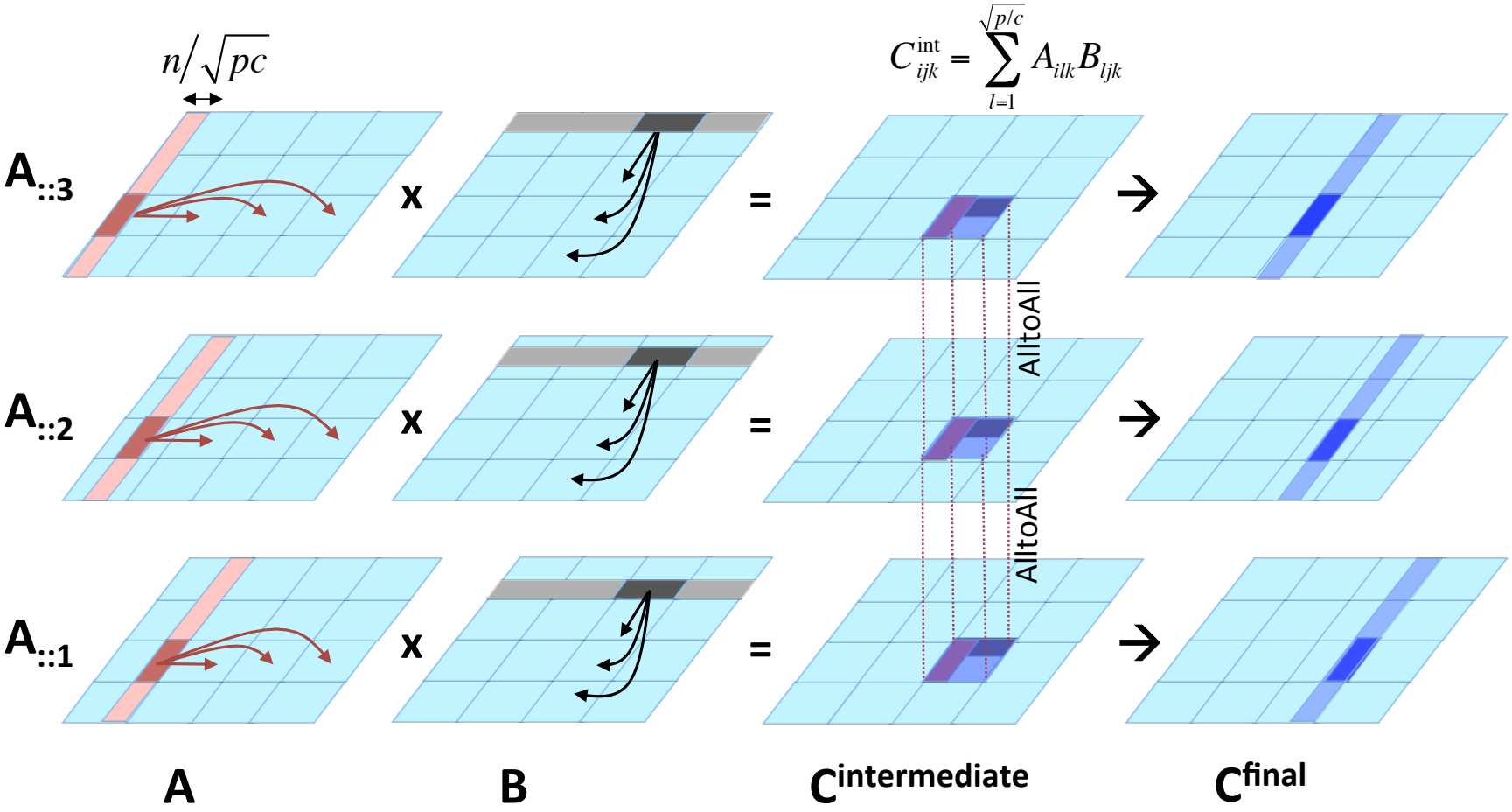
Two new algorithms achieving the bounds
(Up to a logarithmic factor)

i. Recursive 3D, based on
[Ballard, et al. SPAA'12]

ii. Iterative 3D, based on
[Solomonik & Demmel EuroPar'11]

Ballard, **B.**, Demmel, Grigori, Lipshitz, Schwartz, and Toledo. Communication optimal parallel multiplication of sparse random matrices. In SPAA 2013.

3D parallel SpGEMM in a nutshell

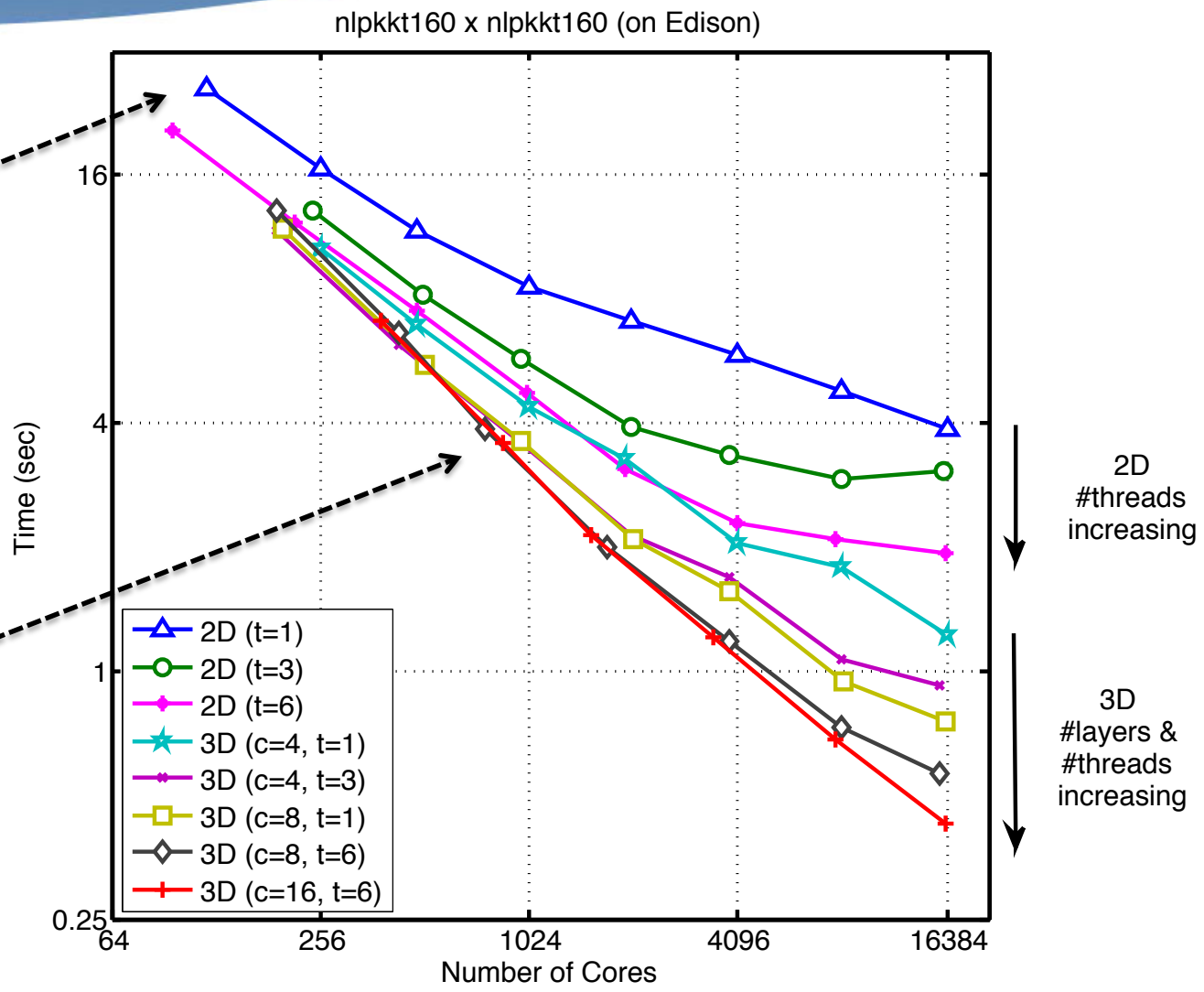


Ariful Azad, Grey Ballard, **B.**, James Demmel, Laura Grigori, Oded Schwartz, Sivan Toledo, Samuel Williams. Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication. SIAM Journal on Scientific Computing (SISC), to appear.

3D SpGEMM performance

2D (non-threaded) is the previous state-of-the art

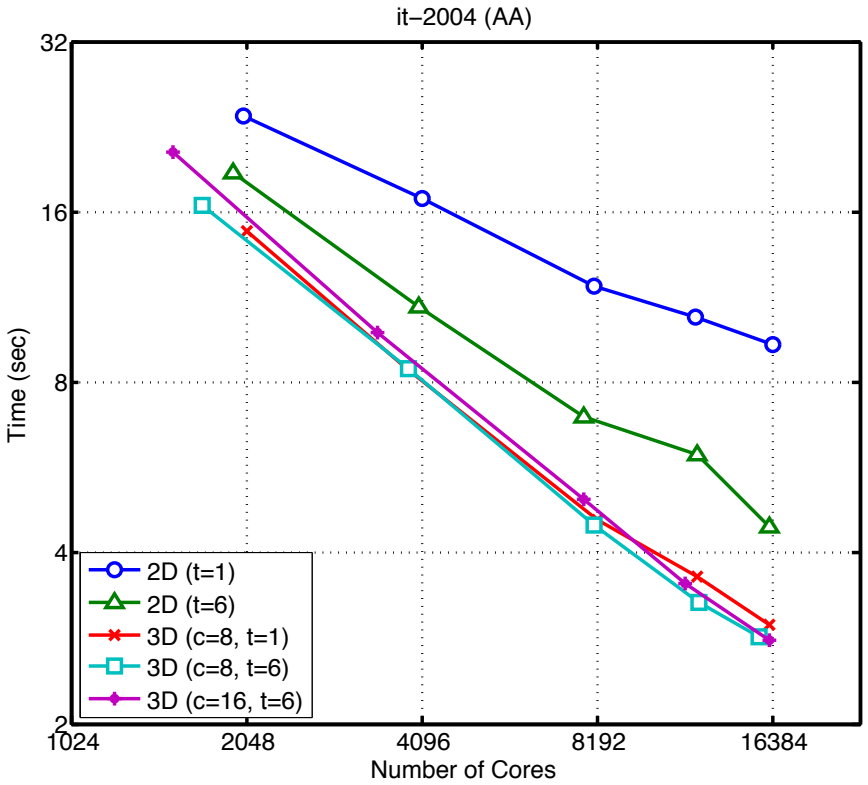
3D (threaded) – first presented here – beats it by 8X at large concurrencies

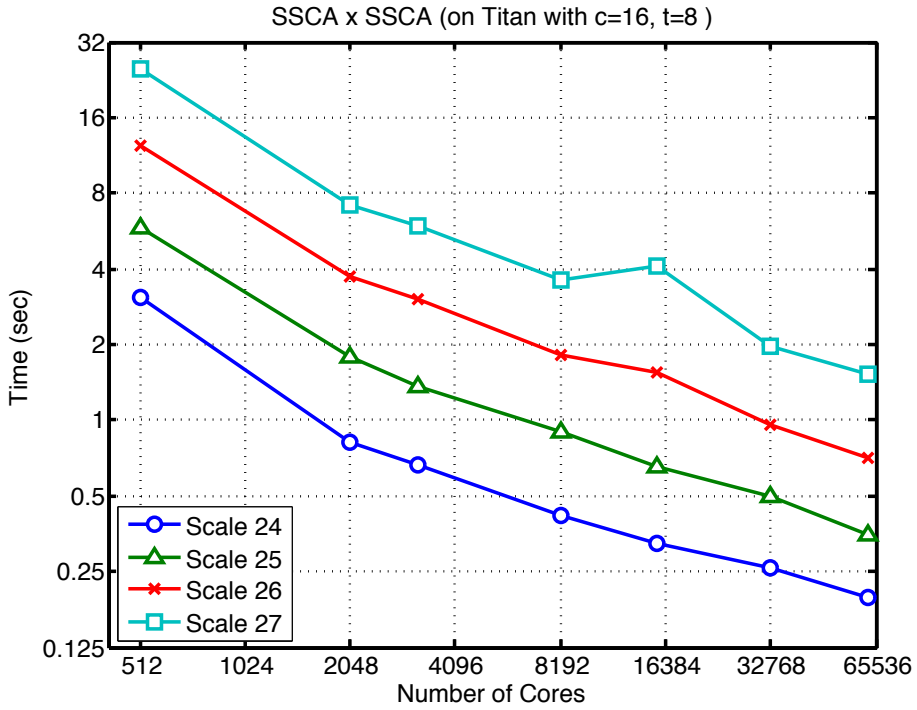


Strong scaling of different variants of 2D and 3D algorithms when squaring of nlpkkt160 matrix on Edison.

3D SpGEMM performance

- Matrix squaring (A*A), proxy for Markov clustering (MCL)





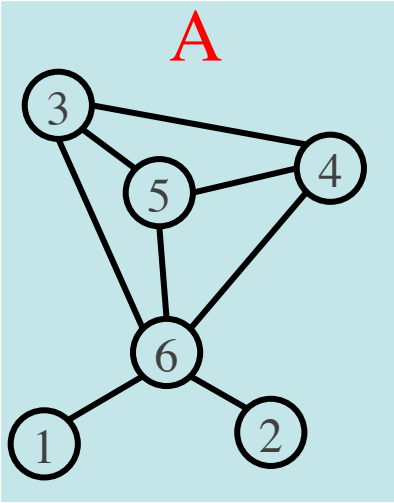
2004 web crawl of .it domain
41 million vertices, 1.1 billion edges

SSCA (R-MAT) matrices on Titan

Outline

- **GraphBLAS:** standard building blocks for graph algorithms in the language of linear algebra
- **SpGEMM**: Computing the sparse matrix-matrix multiplication in parallel
- **Triangle counting/enumeration** in matrix algebra
- **Bipartite graph matching** in parallel
- **Other contributions** and **future work**

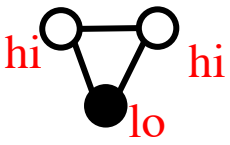
Counting triangles (clustering coefficient)



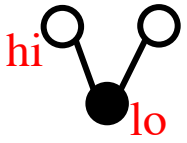
Clustering coefficient:

- Pr (wedge i-j-k makes a triangle with edge i-k)
- 3 * # triangles / # wedges
- 3 * **4** / **19** = **0.63** in example
- may want to compute for each vertex j

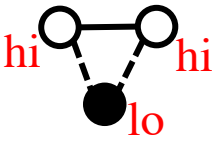
Cohen's algorithm to count triangles:



- Count triangles by lowest-degree vertex.

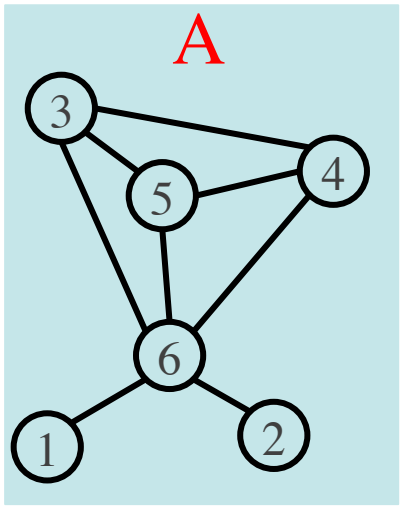


- Enumerate "low-hinged" wedges.



- Keep wedges that close.

Counting triangles (clustering coefficient)

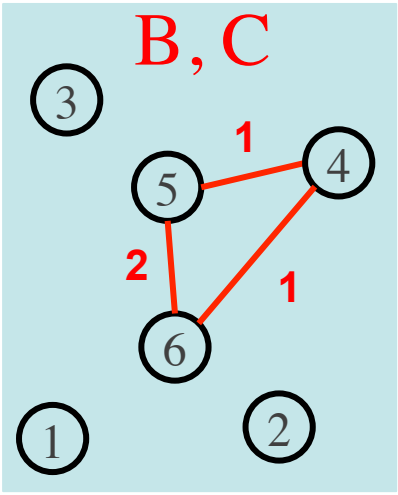


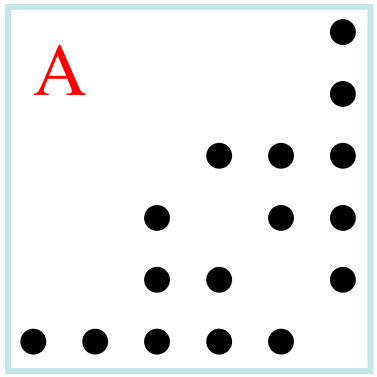
$A = L + U$ (hi->lo + lo->hi)

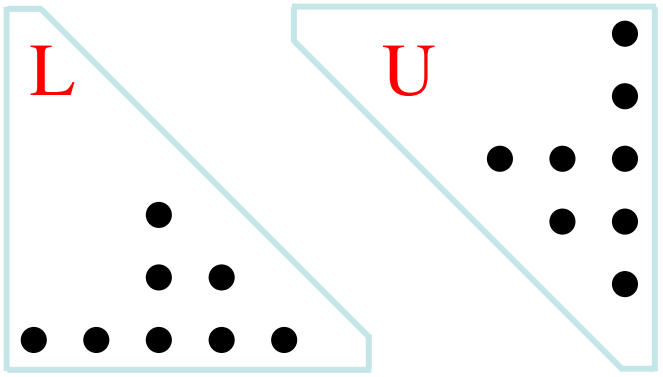
$L \times U = B$ (wedge, low hinge)

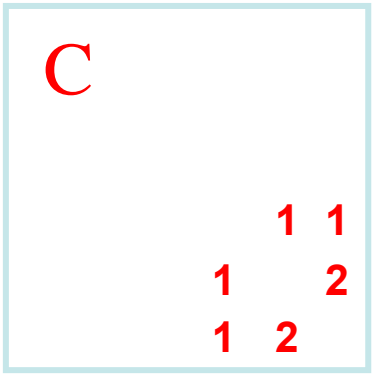
$A \wedge B = C$ (closed wedge)

$sum(C)/2 =$ **4 triangles**



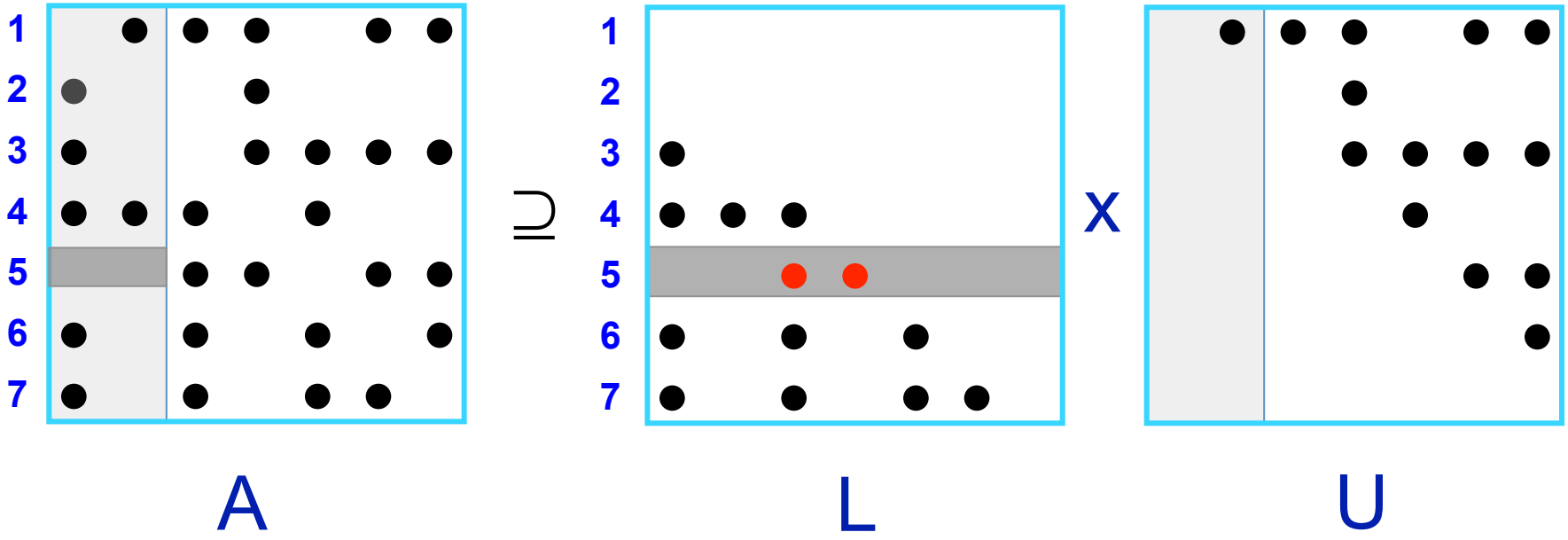






Masked sparse matrix-matrix multiplication (SpGEMM)

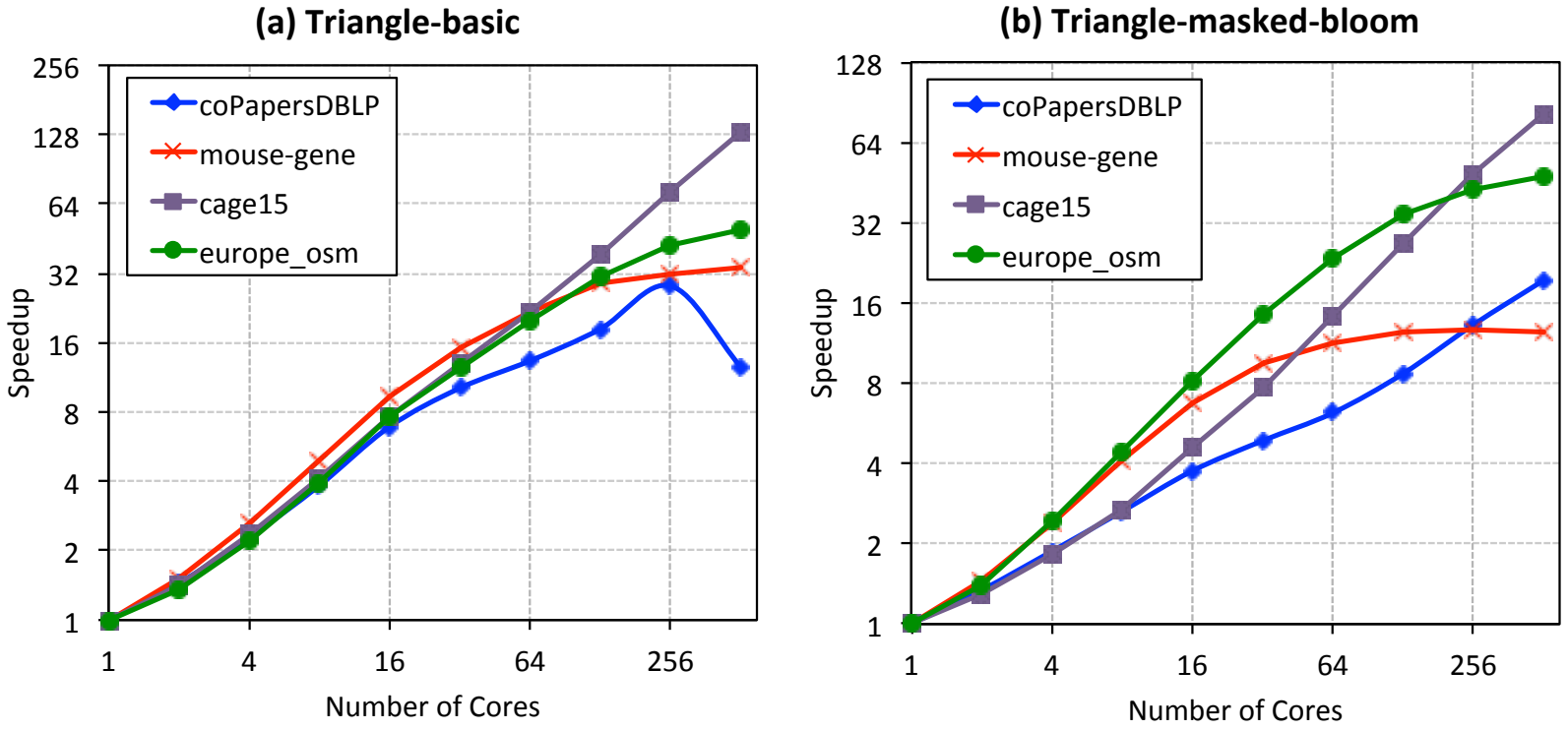
Special SpGEMM: The nonzero structure of the product is contained in the original adjacency matrix A



Avoid communicating nonzeros that fall outside this structure via C = MaskedSpGEMM (L, U, A), which is mathematically equivalent to performing B = L U followed by C = A ^ B.

Ariful Azad, **B.**, and John R Gilbert. "Parallel triangle counting and enumeration using matrix algebra". *Graph Algorithm Building Blocks (GABB), IPDPSW, 2015*

Distributed memory performance of triangle counting



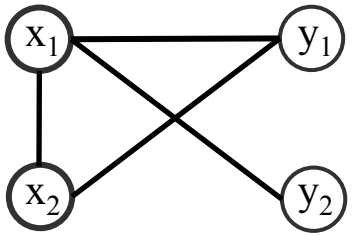
- Improved 1D algorithm (SPAA'13 by Ballard, Buluc, et al.), which had not been implemented and evaluated before.
- Scales reasonably well until 512 cores of NERSC/Edison. Further scaling requires 2D/3D algorithms (ongoing work)
- Bloom filters are used in (b) to signal the presence/absence of zeros in the output to avoid communication.

Outline

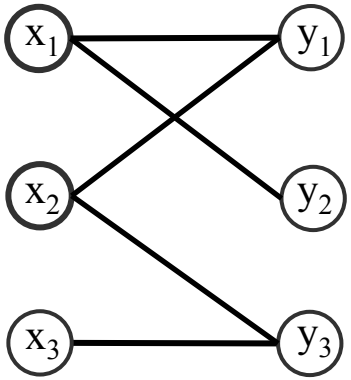
- **GraphBLAS:** standard building blocks for graph algorithms in the language of linear algebra
- **SpGEMM**: Computing the sparse matrix-matrix multiplication in parallel
- **Triangle counting/enumeration** in matrix algebra
- **Bipartite graph matching** in parallel
- **Other contributions** and **future work**

Graph vs. Bipartite Graph

- A graph consists of vertices and edges: G(V,E)
- A bipartite graph: vertices are grouped into two sets. No edge between vertices in a set.
- Notation: **n = #vertices, m= #edges**



A graph

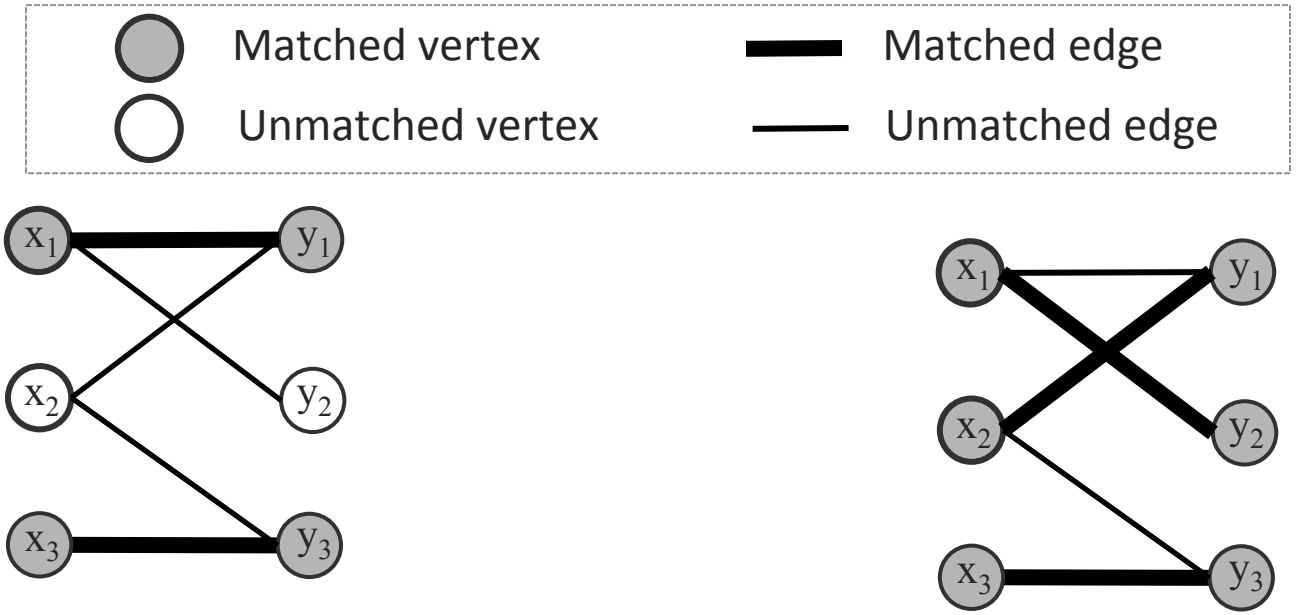


A bipartite graph

A Matching in a Graph

Matching: A subset M of edges with no common end vertices.

$|M|$ = **Cardinality** of the matching M

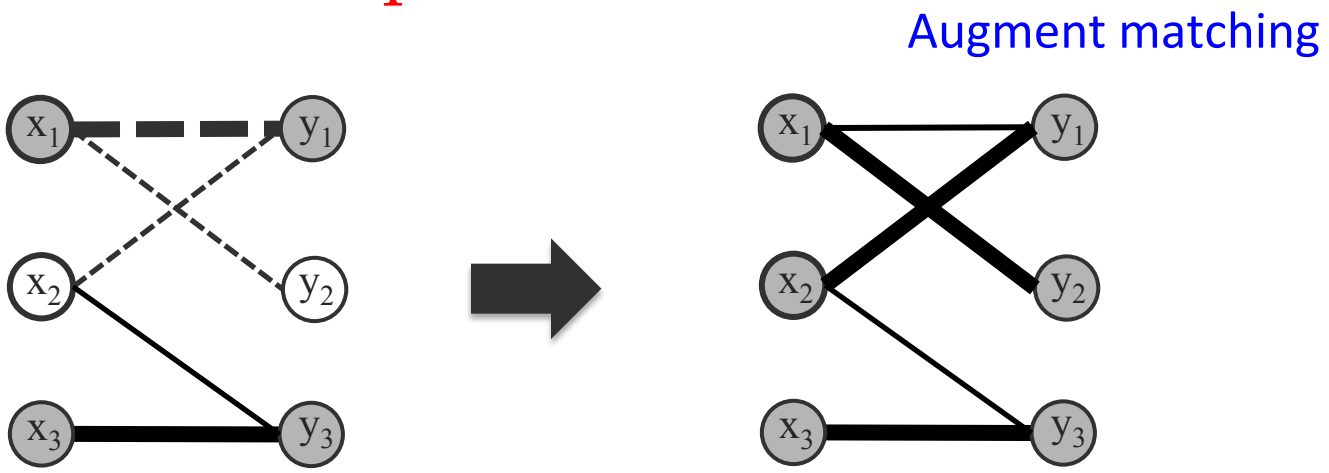


A Matching (**Maximal** cardinality)

Maximum Cardinality Matching

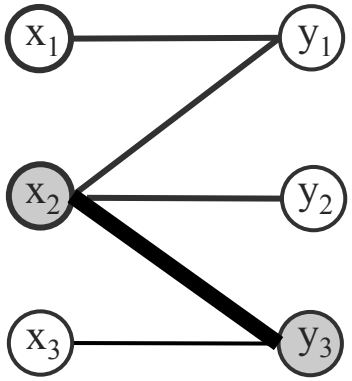
Maximum-cardinality matching

- Augmenting path: A path that alternates between matched and unmatched edges with unmatched end points.

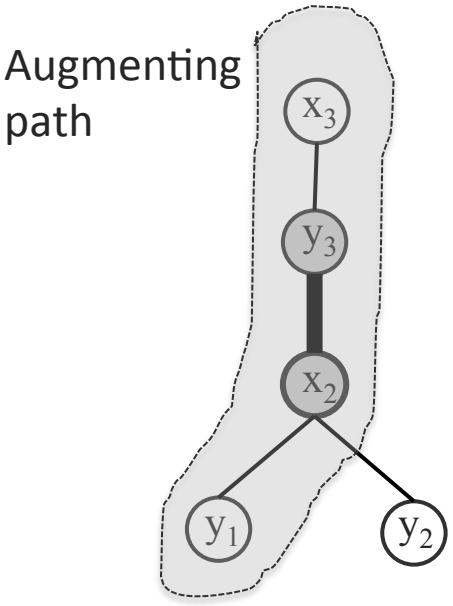


- Algorithm: **Search** for augmenting paths and flip edges across the paths to increase matching.

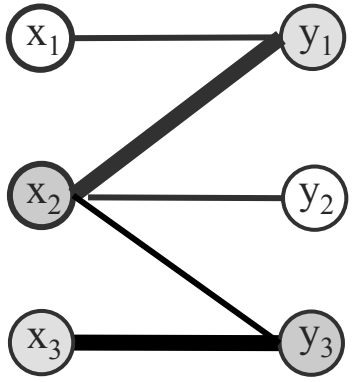
Single-source (SS) search for augmenting paths



1. Initial matching



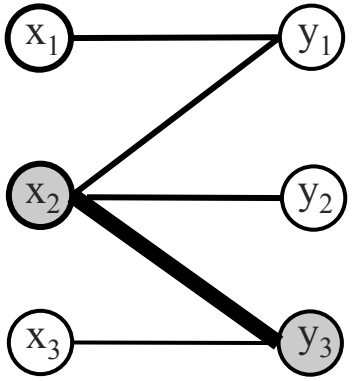
2. Search for augmenting path from x_3. stop when an unmatched vertex found. **Discard the whole tree when no augmenting path is found**



3. Increase matching by flipping edges in the augmenting path

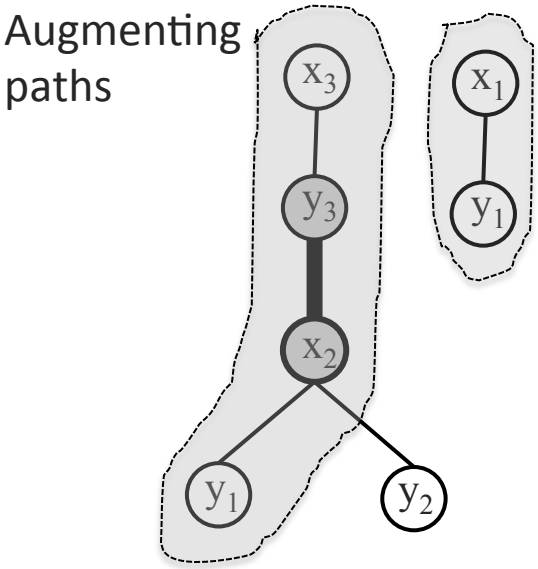
Repeat the process for other unmatched vertices

Multi-source (MS) search for augmenting paths

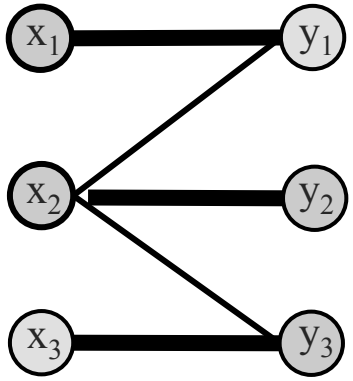


1. Initial matching

Search Forest



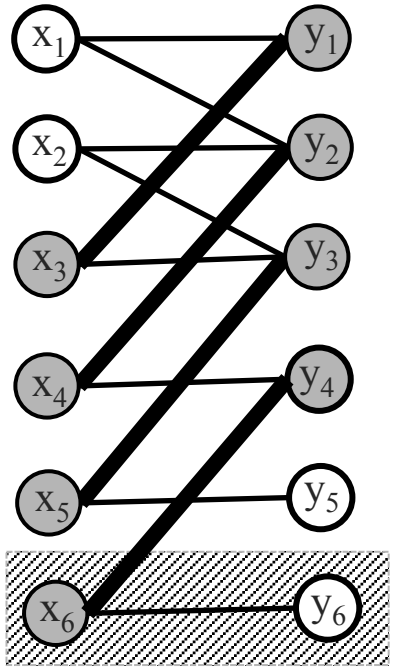
2. Search for **vertex-disjoint** augmenting paths from x_3 & x_1. Grow a tree until an unmatched vertex is found in it. **Can not discard a tree if no augmenting path is found.**



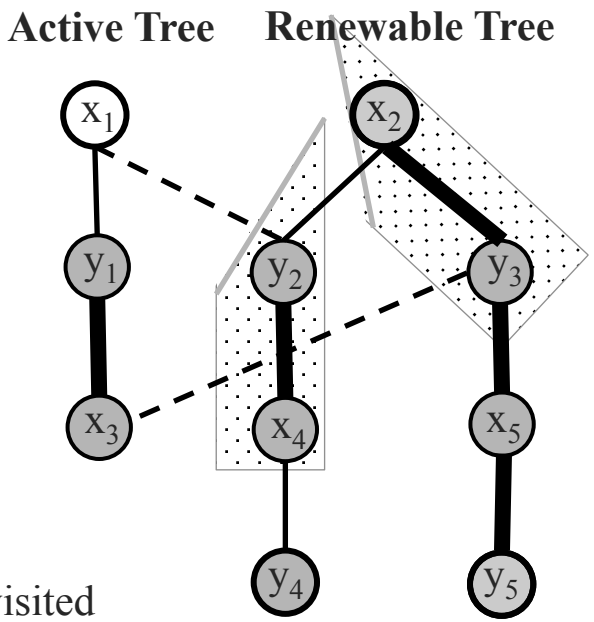
3. Increase matching by flipping edges in the augmenting paths

Repeat the process until no augmenting path is found

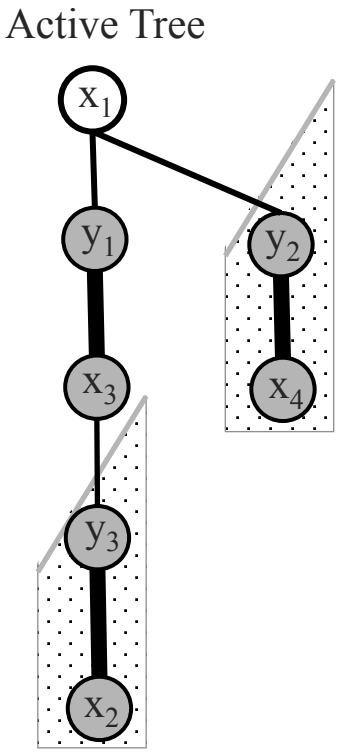
Tree Grafting



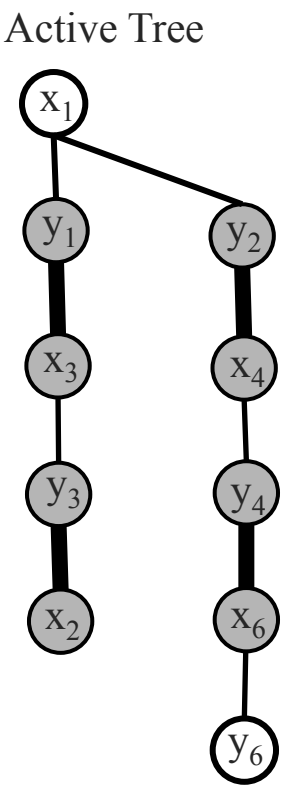
(a) A maximal matching in a Bipartite Graph



(b) Alternating BFS Forest
Augment in forest

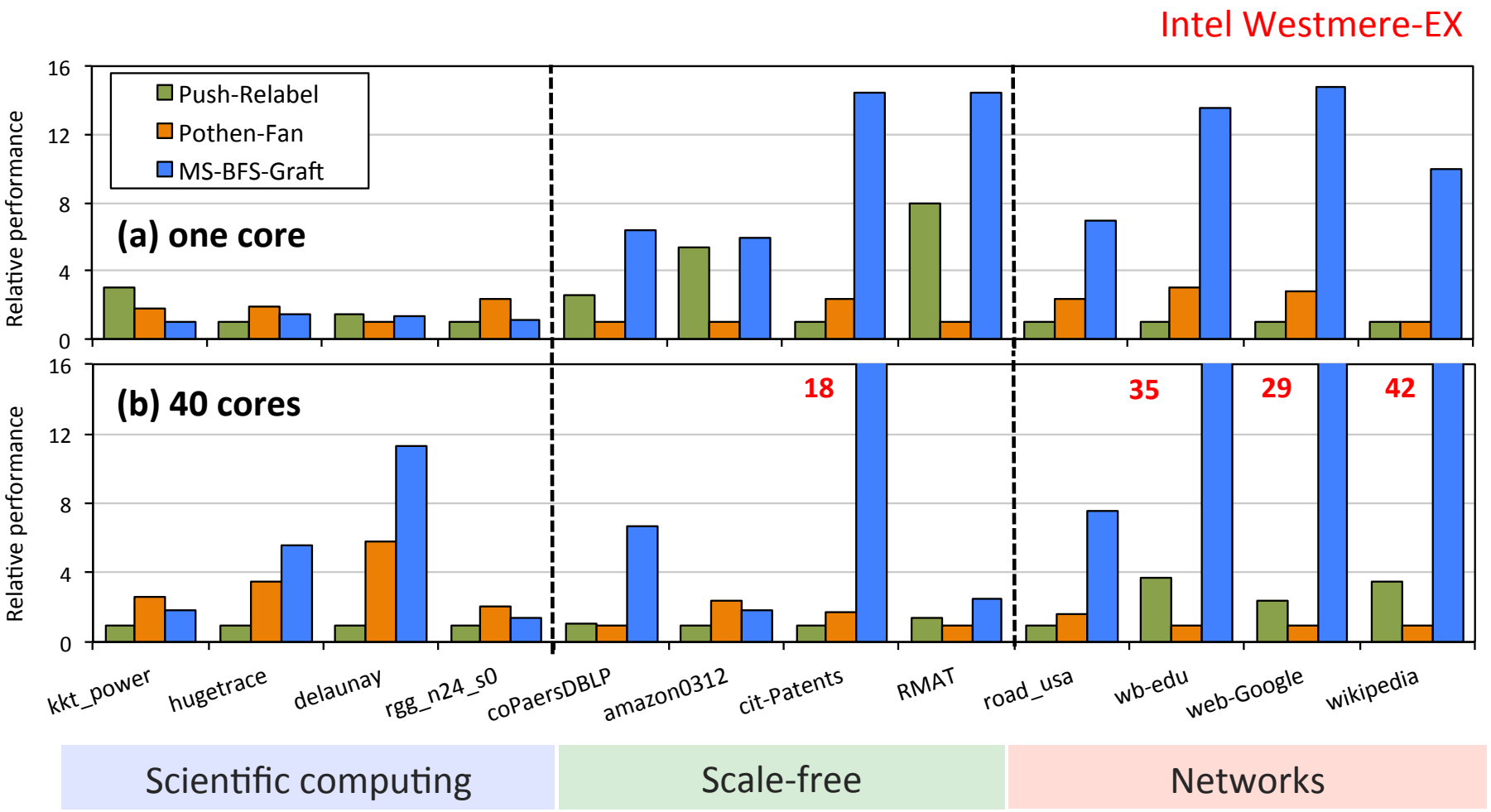


(c) Tree Grafting



(d) Continue BFS

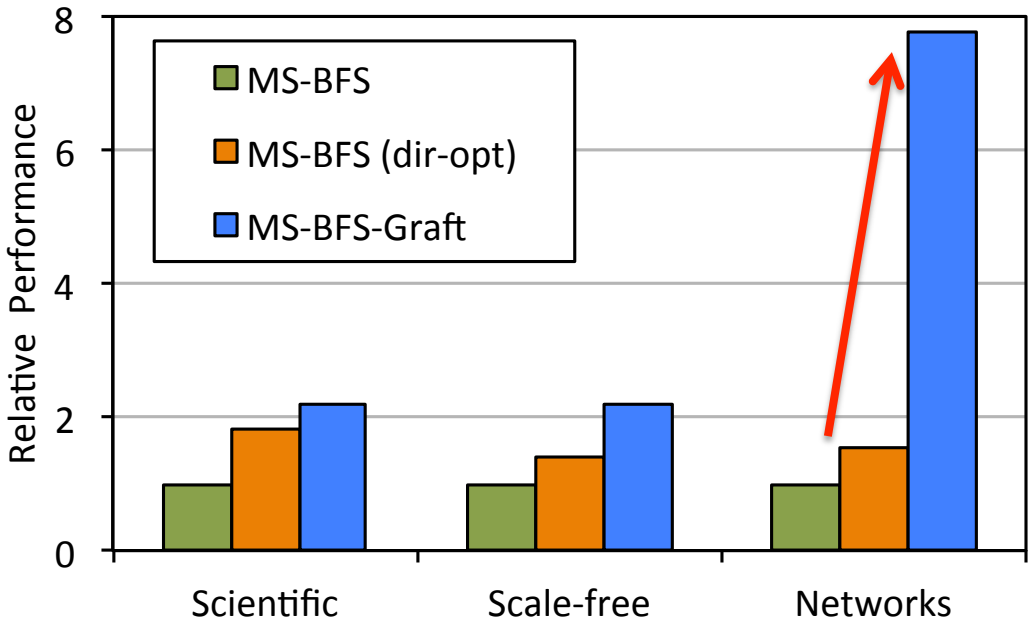
Comparison with state-of-the-art



Sources of performance

1. Direction optimized BFS (1.6x)
2. Tree grafting (3x)

On 40 cores
Intel Westmere-EX

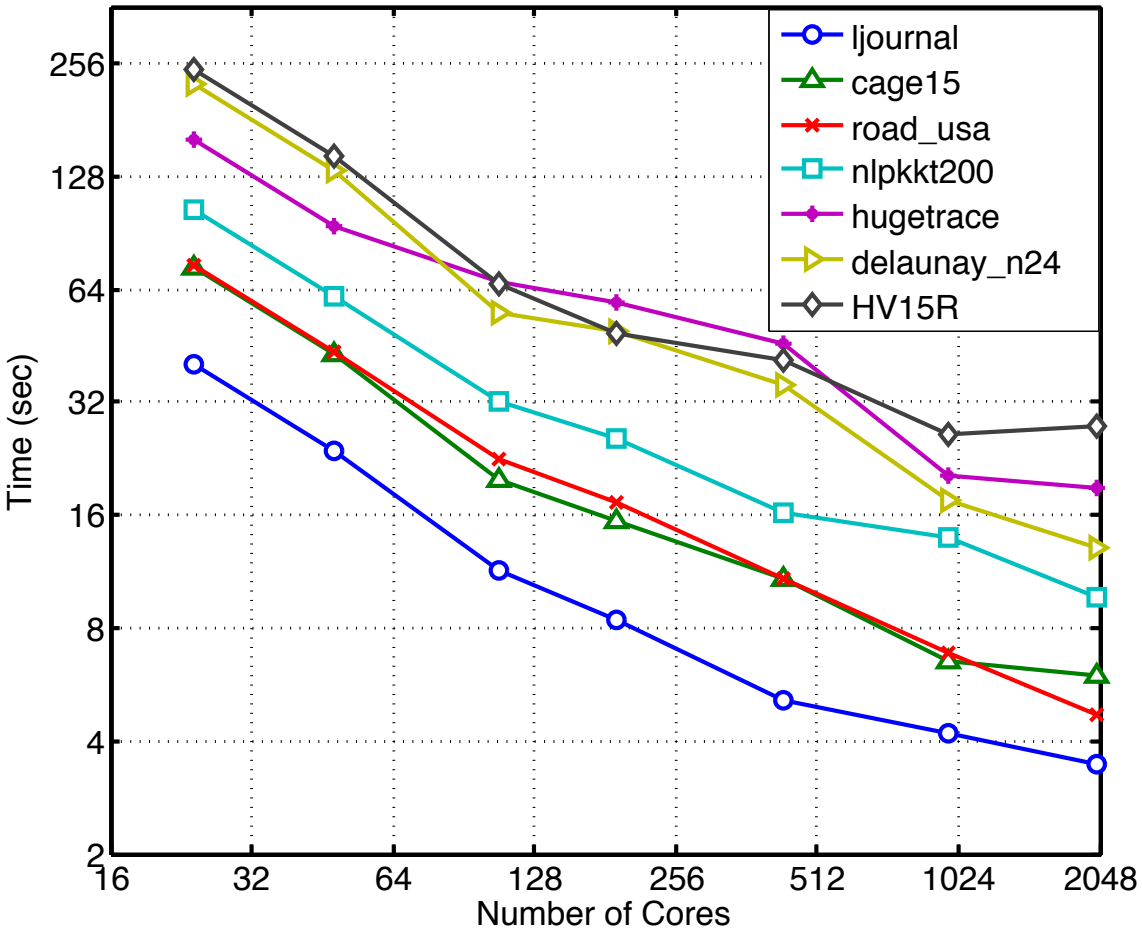


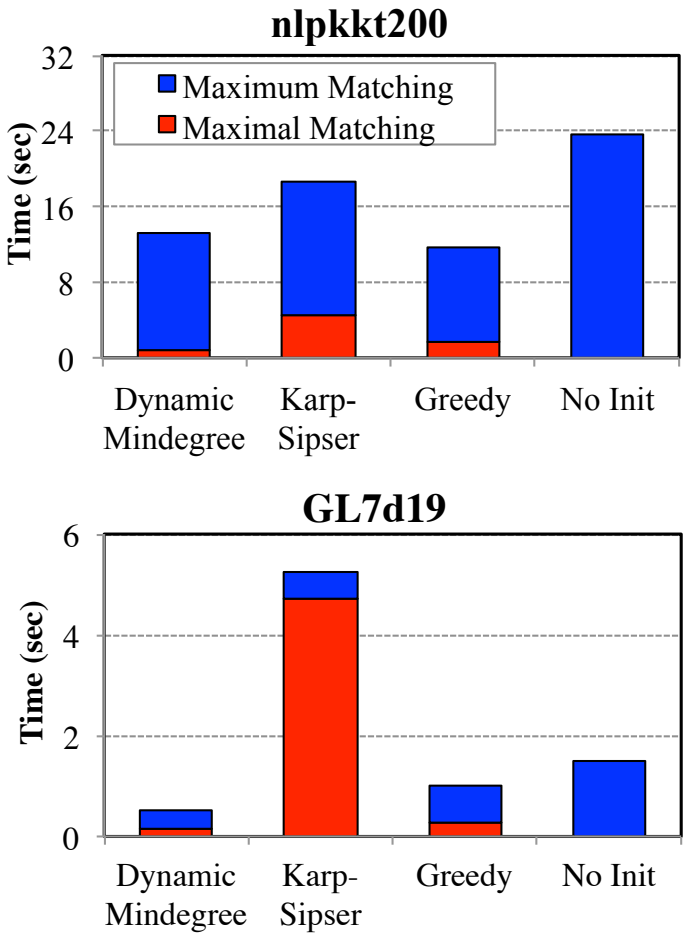
Highest performance improvement on graphs with low matching number

Ariful Azad, **B.**, Alex Pothen. A parallel tree grafting algorithm for maximum cardinality matching in bipartite graphs. In Proceedings of the IPDPS, 2015.

Ariful Azad, **B.**, Alex Pothen. Computing maximum cardinality matchings in parallel on bipartite graphs via tree-grafting. IEEE Transactions on Parallel and Distributed Systems (TPDS)), 2016.

Matching in distributed memory





Ariful Azad and Aydin Buluç. Distributed-memory algorithms for maximum cardinality matching in bipartite graphs. In Proceedings of the IPDPS, 2016

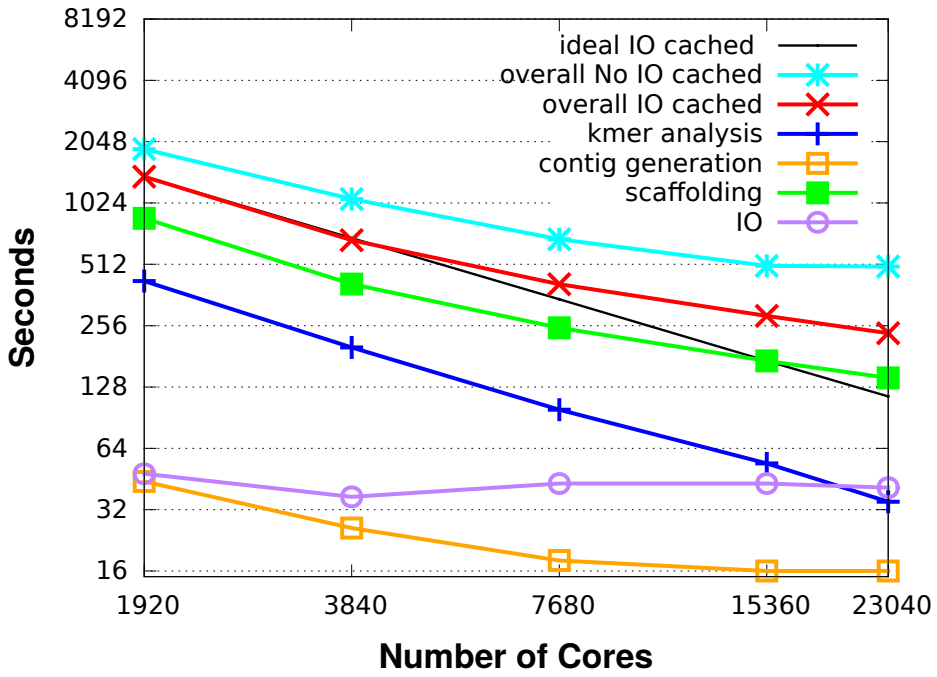
Outline

- **GraphBLAS:** standard building blocks for graph algorithms in the language of linear algebra
- **SpGEMM**: Computing the sparse matrix-matrix multiplication in parallel
- **Triangle counting/enumeration** in matrix algebra
- **Bipartite graph matching** in parallel
- **Other contributions** and **future work**

HipMer: An Extreme-Scale De Novo Genome Assembler

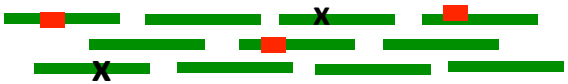
Meraculous assembler is used in production at the Joint Genome Institute

- Wheat assembly is a "grand challenge"
- Hardest part is contig generation (large in-memory *hash table* that represents graph)
- HipMer is an efficient parallelization of Meraculous that leverages the capabilities of Unified Parallel C (UPC)



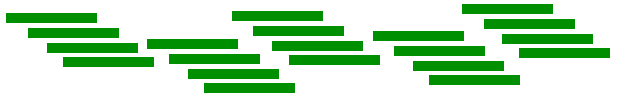
Meraculous Assembly Pipeline

reads



New fast & parallel I/O

k-mers



New k-mer analysis filters errors using probabilistic "Bloom Filter"

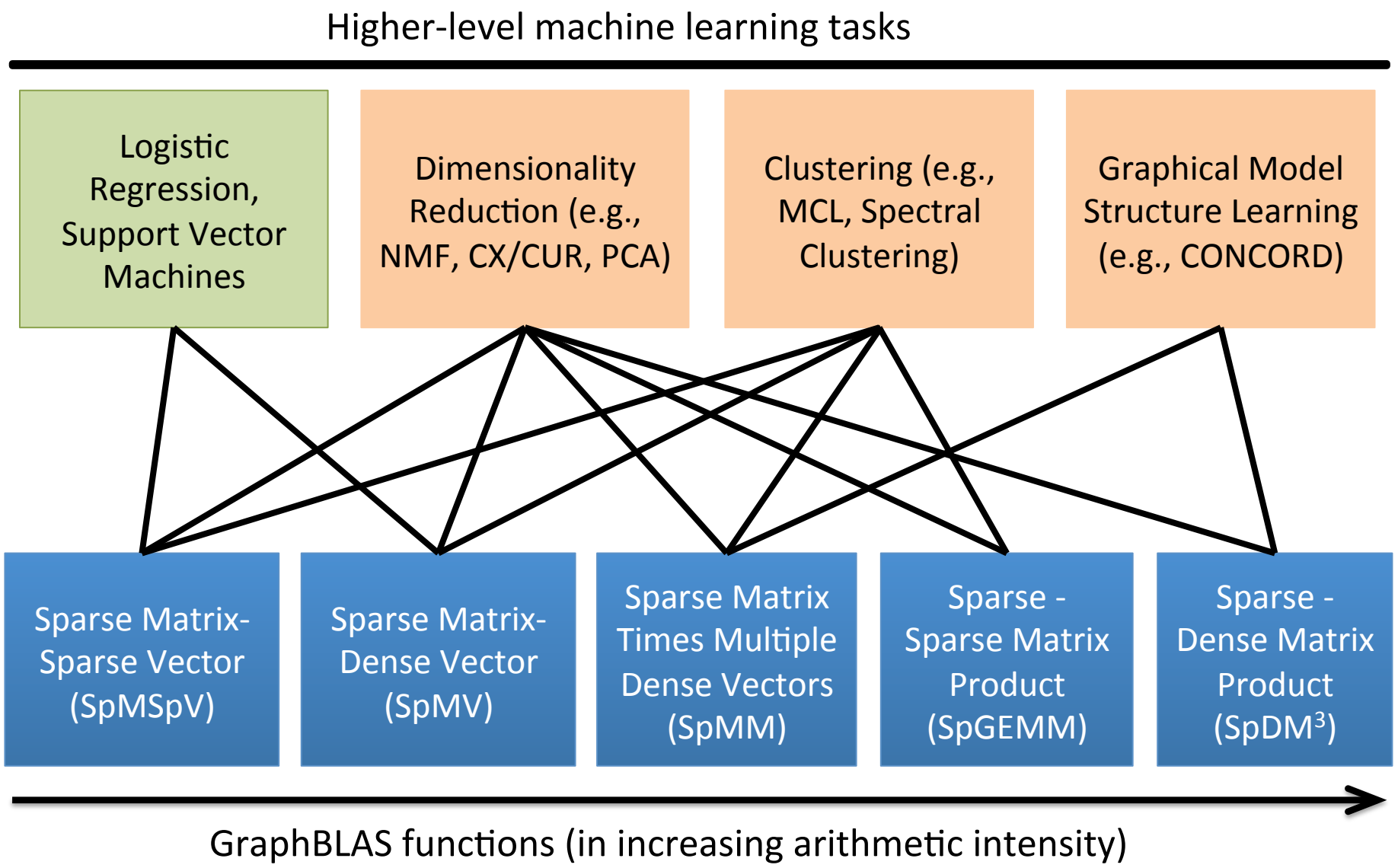
contigs

Graph algorithm (connected components) scales to 15K cores on NERSC's Edison

Scaffolding using Scalable Alignment

Performance improvement from ***days*** to ***minutes***

Future Work: Machine Learning



Future Work: Metagenomics

- Microbiomes: dynamic consortia of hundreds or thousands of microbial species and strains of varying abundance and diversity.
- Metagenomics: the application of high-throughput genome sequencing technologies to DNA extracted from microbiomes

Challenges:

- Metagenome assembly is computationally harder than single genome
- Protein clustering at this scale has never been done before

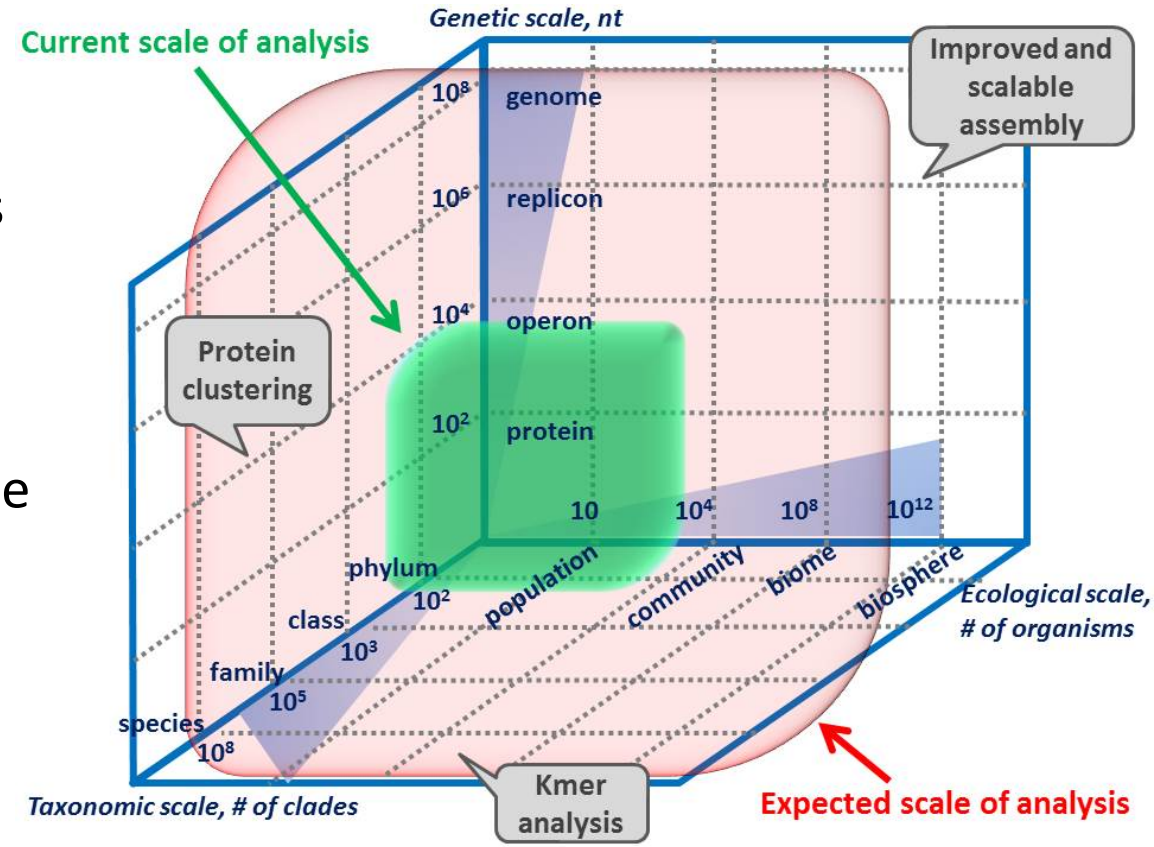


Figure credit: Natalia Ivanova (JGI)

Acknowledgments

Ariful Azad, David Bader, Grey Ballard, Scott Beamer, Jarrod Chapman, James Demmel, Rob Egan, Evangelos Georganas, John Gilbert, Laura Grigori, Steve Hofmeyr, Costin Iancu, Jeremy Kepner, Penporn Koanantakool, Benjamin Lipshitz, Tim Mattson, Scott McMillan, Henning Meyerhenke, Jose Moreira, Sang Oh, Lenny Oliker, John Owens, Alex Pothen, Dan Rokhsar, Oded Schwartz, Sivan Toledo, Sam Williams, Carl Yang, Kathy Yelick.

Work is funded by