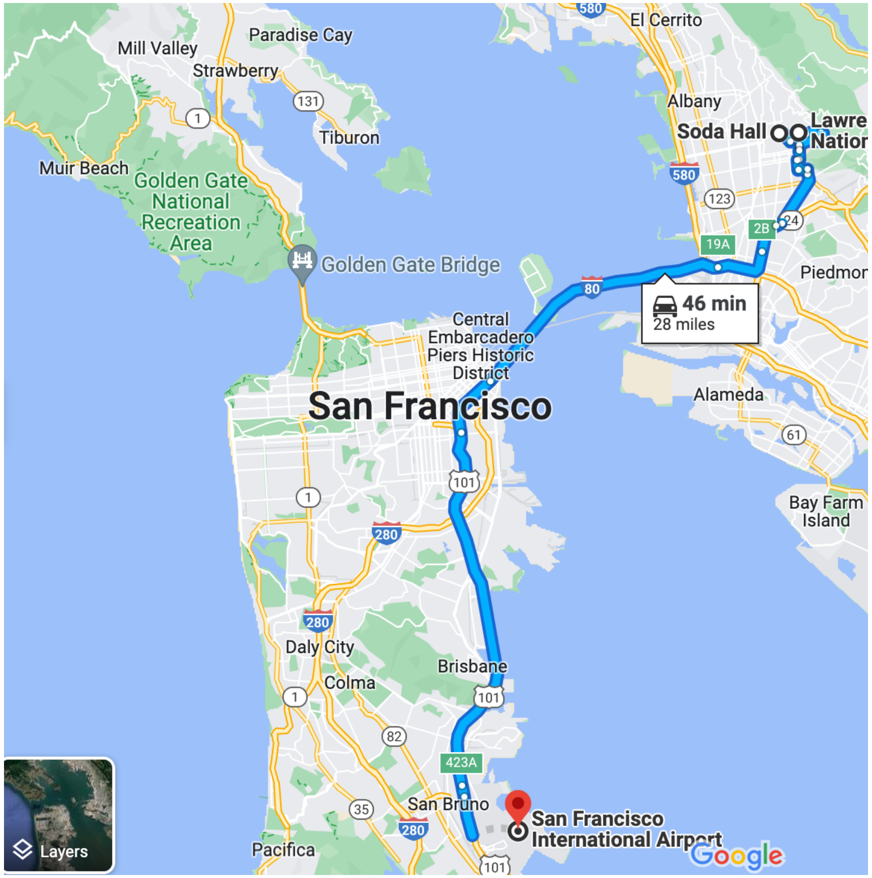
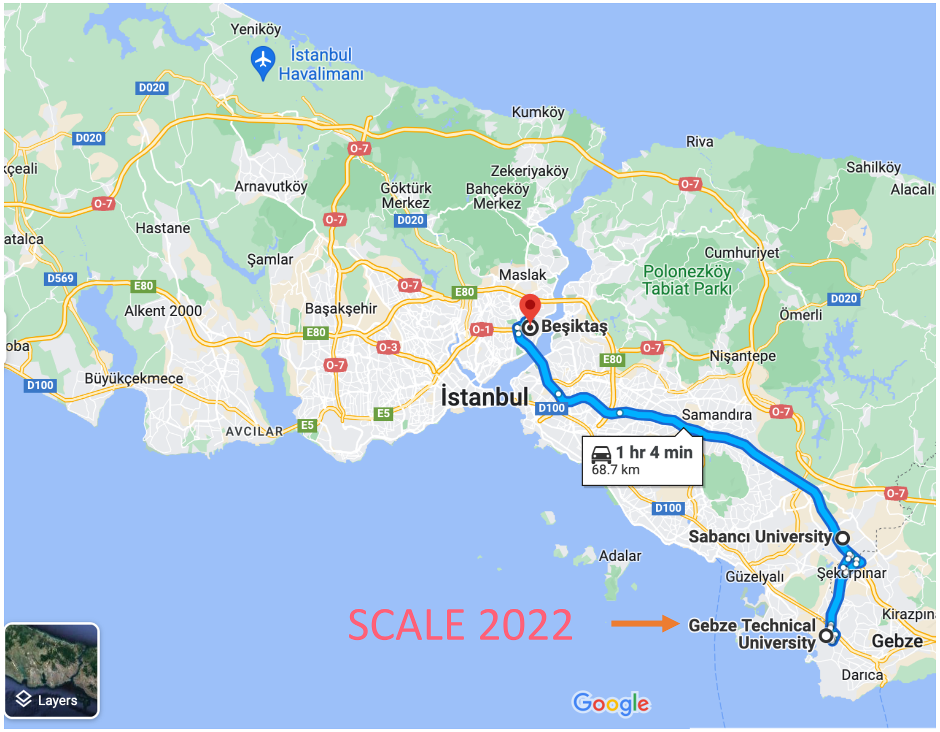
Sparse matrices powering three pillars of science: simulation, data, and learning

Aydın Buluç
Lawrence Berkeley National Laboratory & UC Berkeley

Tutorial at ISSAC, Lille, France
July 4, 2022

Some background

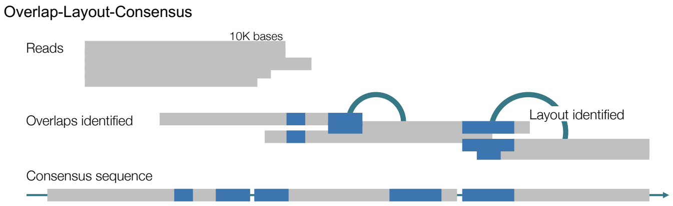




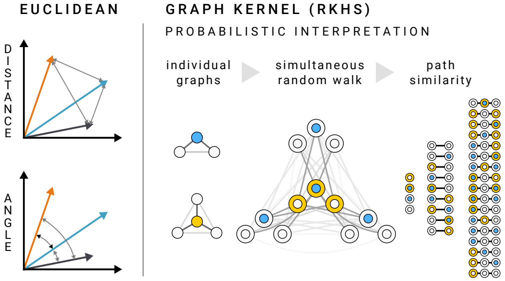
SCALE 2022 → Gebze Technical University

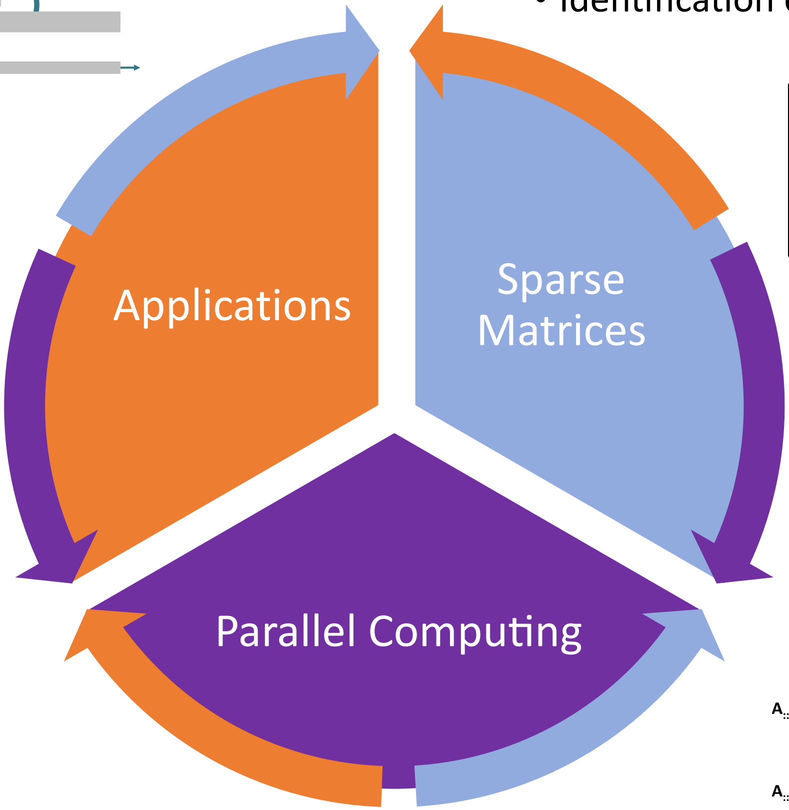
PASSION Lab Research Agenda

http://passion.lbl.gov



- Genomics
- Graph analysis
- Proteomics
- Machine learning



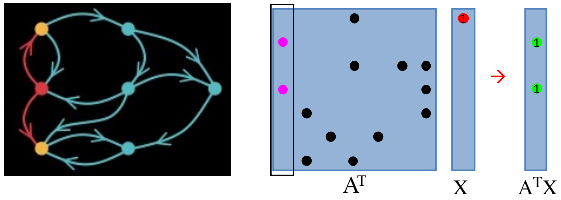


Applications

Sparse Matrices

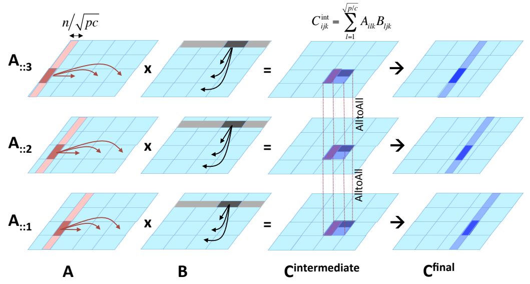
Parallel Computing

- New sparse data structures and algorithms
- Identification of computational primitives



GraphBLAS: graphs in the language of linear algebra
http://graphblas.org

Communication-avoiding algorithms for sparse matrices



- Parallel data structures
- Parallel programming
- Communication bounds

PASSION Lab People

Aydın Buluç (Principal Investigator)

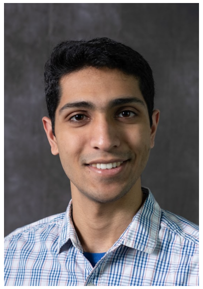
- Staff Scientist, AMCRD, Lawrence Berkeley National Laboratory
- Adjunct Assistant Professor, EECS Department (CS division), UC Berkeley

UC Berkeley PhD Students (co-advised)

Ben Brock

Giulia Guidi

Alok Tripathy



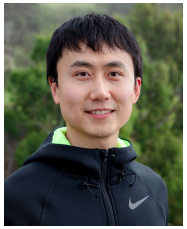
Vivek Bharadwaj

Undergraduate researchers:

- Richard Lettich
- Ujjaini Mukhopadhyay

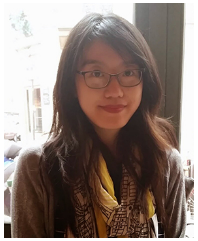
LBNL Research Scientists, Engineers, Postdocs, Visiting Fellows

Oguz Selvitopi



Yu-Hang Tang

Can Kizilkale



Helen Xu

Gabriel Raulet

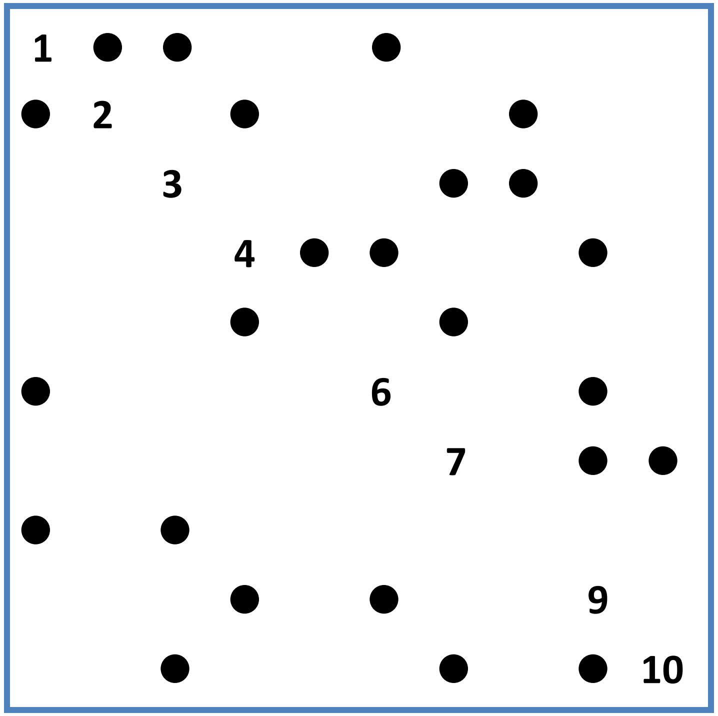
Koby Hayashi

Sparse Matrices

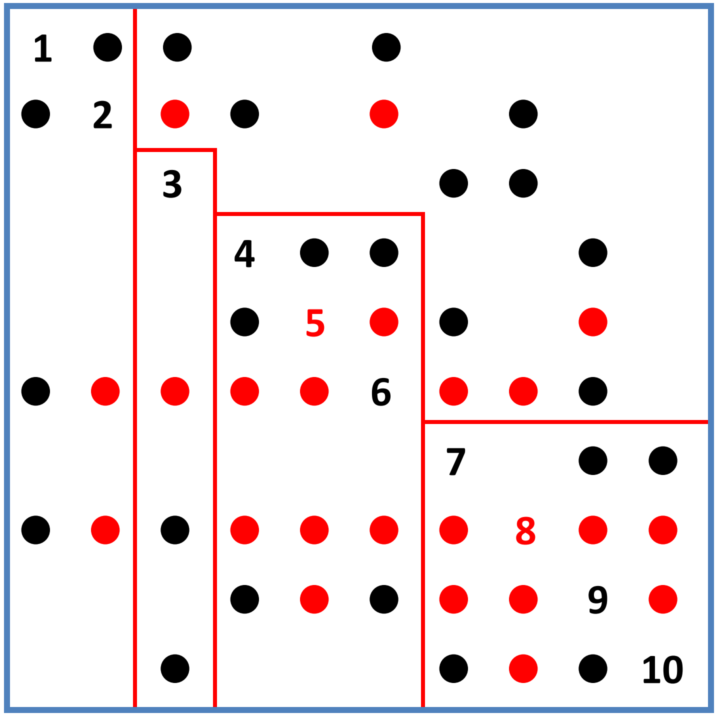
"I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were 'sparse' in the matrix, and that typically the triangular matrices associated with the forward and back solution provided by Gaussian elimination would remain sparse if pivot elements were chosen with care"

- Harry Markowitz, describing the 1950s work on portfolio theory that won the 1990 Nobel Prize for Economics

Sparse Matrices in Simulations



Original matrix A



Factors L+U

Original: Ax = b (hard to solve directly)

Factored: LUx = b (solvable by direct substitution)

High-level outline

- **Sparse matrices for graph algorithms**
- Sparse matrices for computational biology
- Sparse matrices for machine learning
- Parallel algorithms for sparse matrix primitives
- Available software

Sparse Matrices for Graphs

- Motivation
- Case studies:

A. **Graph traversals:** Breadth-first search
   - Motif: Sparse matrix times sparse vector (SpMSpV)

B. **Maximal Independent Sets:** Luby's algorithm
   - Motif: SpMSpV

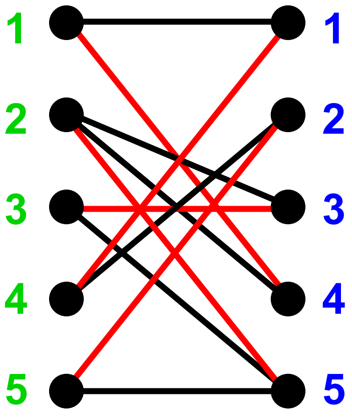
C. **Triangle Counting**
   - Motif: SpGEMM

D. **Betweenness Centrality:** Brandes' algorithm
   - Motif: SpMSpV or sparse matrix-matrix multiply (SpGEMM)

Large graphs in scientific computing

$$\begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & \bullet & & & \bullet & \\ 2 & & & \bullet & \bullet & \bullet \\ 3 & & & \bullet & & \bullet \\ 4 & \bullet & \bullet & & & \\ 5 & & \bullet & & & \bullet \\ \hline \end{array}$$

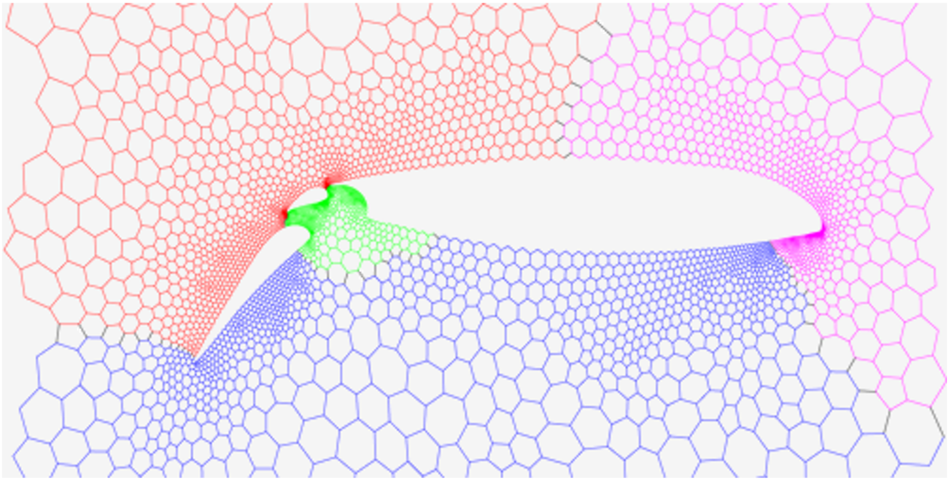
A



$$\begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ \hline 4 & {\color{red}\bullet} & \bullet & & & \\ 5 & & {\color{red}\bullet} & & & \bullet \\ 3 & & & {\color{red}\bullet} & & \bullet \\ 1 & \bullet & & & {\color{red}\bullet} & \\ 2 & & & \bullet & \bullet & {\color{red}\bullet} \\ \hline \end{array}$$

PA

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



Graph partitioning: *Dynamic load balancing* in parallel simulations

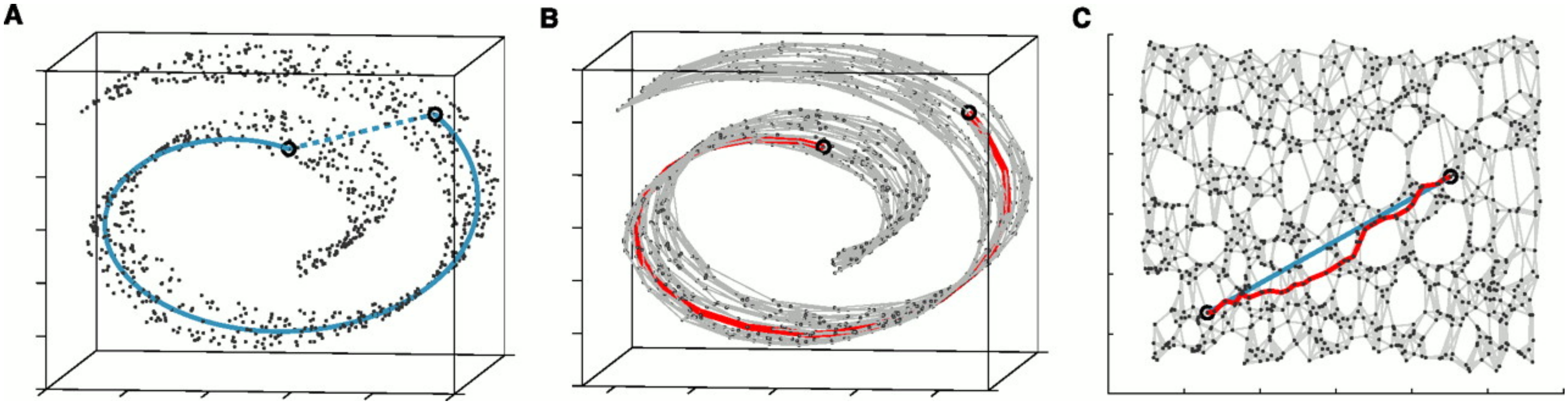
Picture (left) credit: Sanders and Schulz

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

Manifold Learning

Isomap (Nonlinear dimensionality reduction): Preserves the intrinsic geometry of the data by using the geodesic distances on manifold between all pairs of points

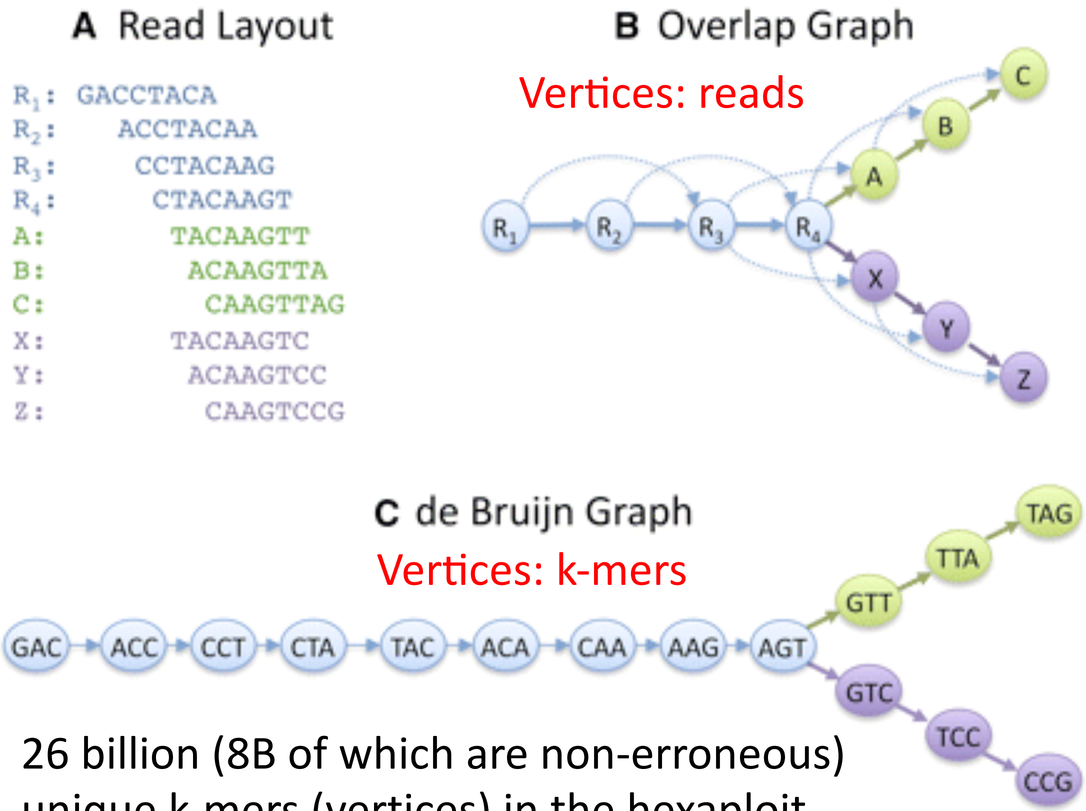
Tools used or desired:
- K-nearest neighbors
- *All pairs shortest paths (APSP)*
- Top-k eigenvalues



Tenenbaum, Joshua B., Vin De Silva, and John C. Langford. "A global geometric framework for nonlinear dimensionality reduction." Science 290.5500 (2000): 2319-2323.

Large graphs in Biology

Whole genome assembly



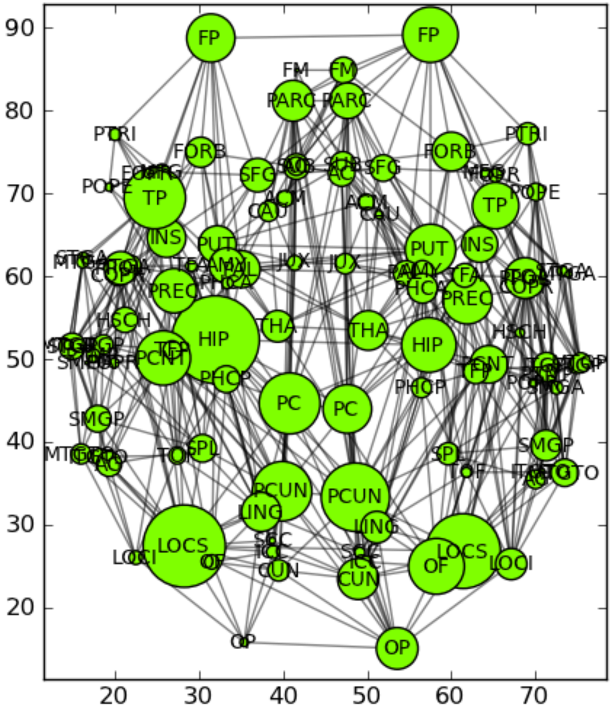
Vertices: reads

Vertices: k-mers

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

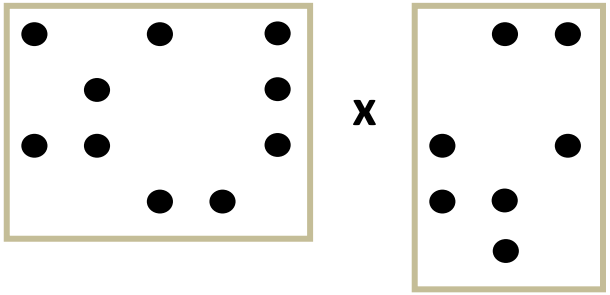
The case for sparse matrices

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

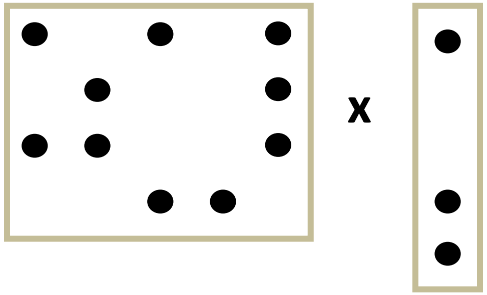
Traditional graph computations	Graphs in the language of linear algebra
Data driven, unpredictable communication.	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse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

Linear-algebraic primitives for graphs

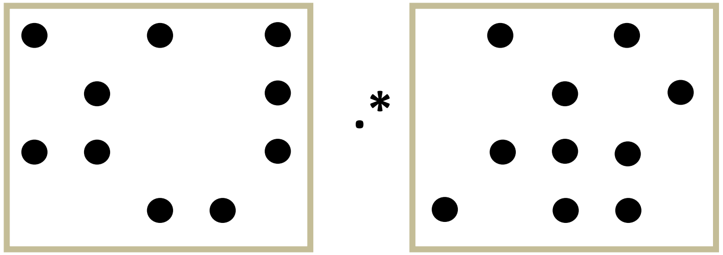
Sparse matrix X sparse matrix



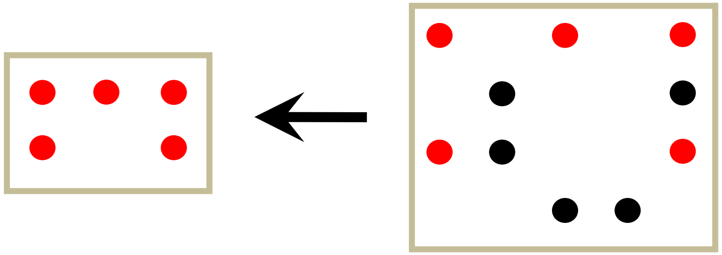
Sparse matrix X sparse vector



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

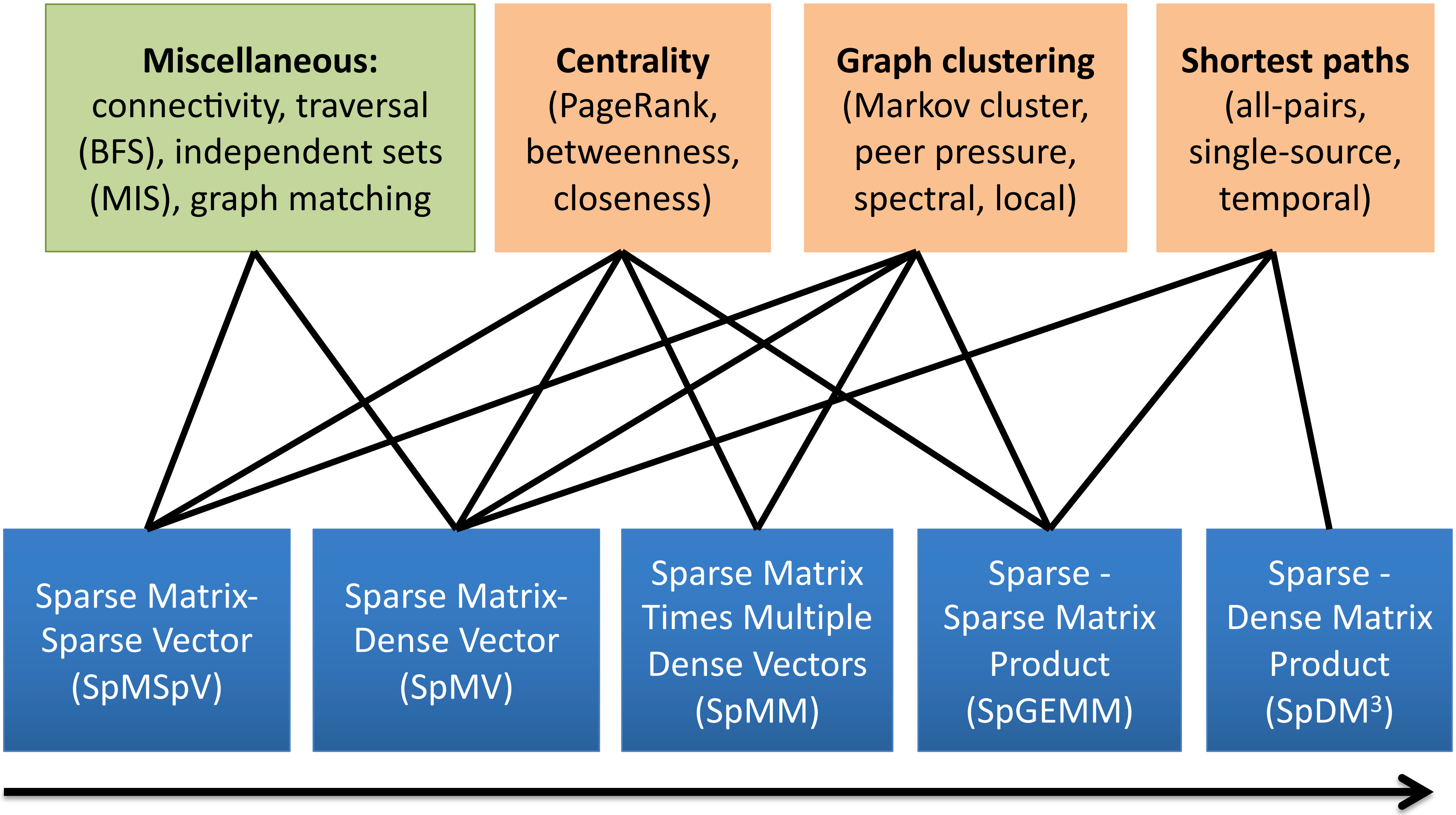
Examples of semirings in graph algorithms

Real field: **(R, +, x)**	Classical numerical linear algebra
Boolean algebra: **({0 1}, |, &)**	Graph connectivity
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**

Graph Algorithms on GraphBLAS

http://graphblas.org



Miscellaneous: connectivity, traversal (BFS), independent sets (MIS), graph matching

Centrality (PageRank, betweenness, closeness)

Graph clustering (Markov cluster, peer pressure, spectral, local)

Shortest paths (all-pairs, single-source, temporal)

Sparse Matrix-Sparse Vector (SpMSpV)

Sparse Matrix-Dense Vector (SpMV)

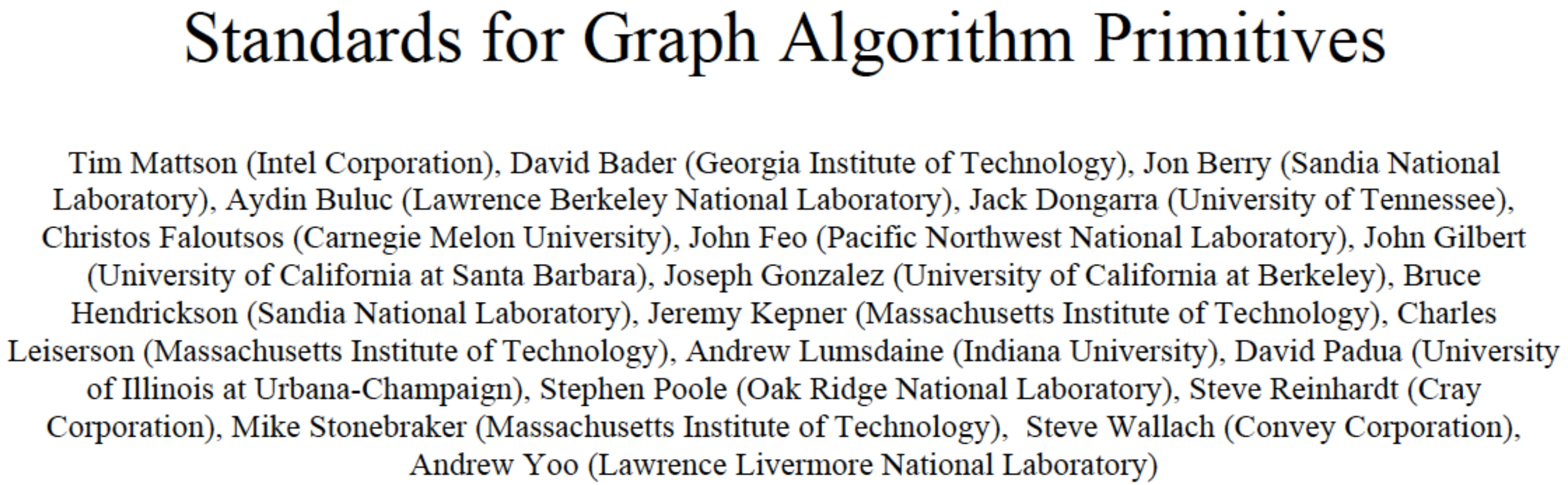
Sparse Matrix Times Multiple Dense Vectors (SpMM)

Sparse - Sparse Matrix Product (SpGEMM)

Sparse - Dense Matrix Product (SpDM^3)

GraphBLAS primitives in increasing arithmetic intensity

The GraphBLAS forum



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.

“If you want to go fast, go alone. If you want to go far, go together.” -- unknown

https://graphblas.github.io/

GraphBLAS C API Specification

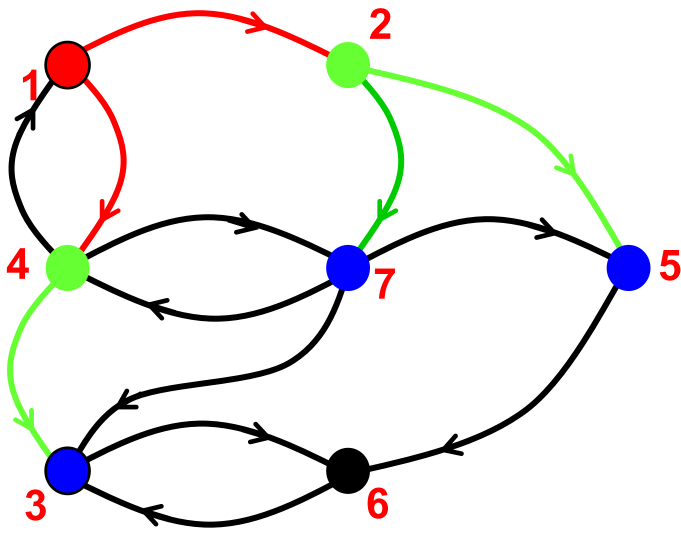
- **Goal:** A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an actual Application Programming Interface (API) that
  1. is faithful to the mathematics as much as possible, and
  2. enables efficient implementations on modern hardware.
- **Impact:** All graph and machine learning algorithms that can be expressed in the language of linear algebra
- **Innovation:** Function signatures (e.g. mxm, vxm, assign, extract), parallelism constructs (blocking v. non-blocking), fundamental objects (masks, matrices, vectors, descriptors), a hierarchy of algebras (functions, monoids, and semiring)

```
GrB_info GrB_mxm(GrB_Matrix                 *C,       // destination
                 const GrB_Matrix            Mask,
                 const GrB_BinaryOp          accum,
                 const GrB_Semiring          op,
                 const GrB_Matrix            A,
                 const GrB_Matrix            B
              [, const Descriptor            desc]);
```

$$C(\neg M) \oplus= A^T \oplus . \otimes B^T$$

A.Buluç, T. Mattson, S. McMillan, J. Moreira, C. Yang. “The GraphBLAS C API Specification”, version 1.3.0

Pattern 1: Sparse matrix times sparse vector (SpMSpV)



Single-source traversal:

BFS, connected components, matching, ordering, etc.

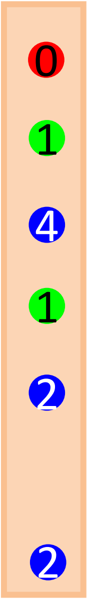
GrB_mxv(y, p, <semiring>, A, x, <desc>)

A: sparse adjacency matrix

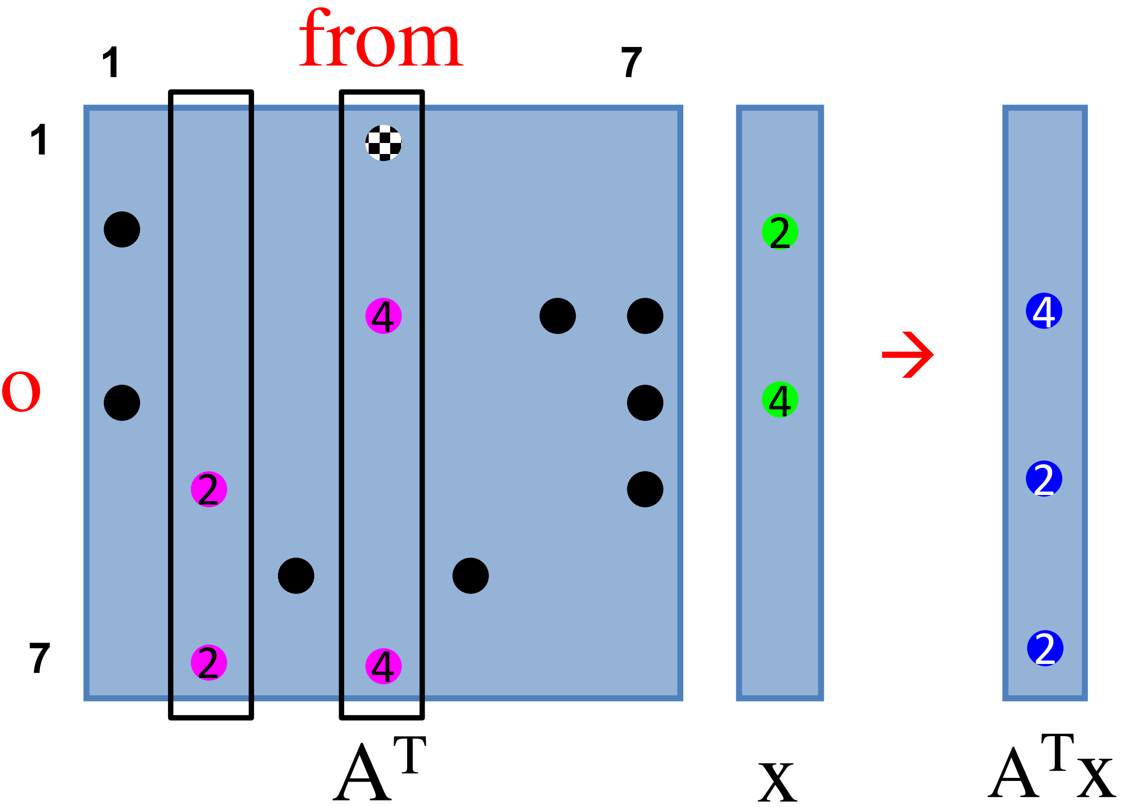
x: sparse input vector (previous frontier)

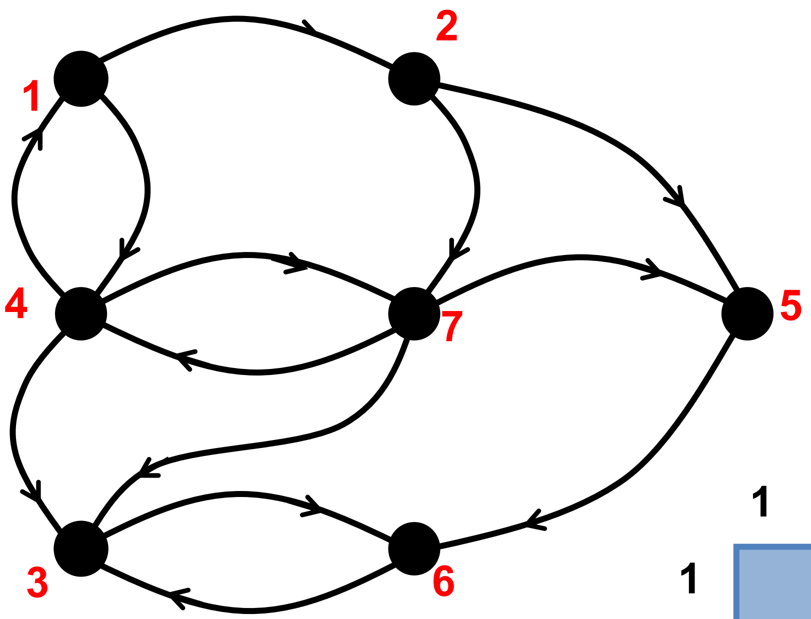
p: mask (already discovered vertices)

parents (p):

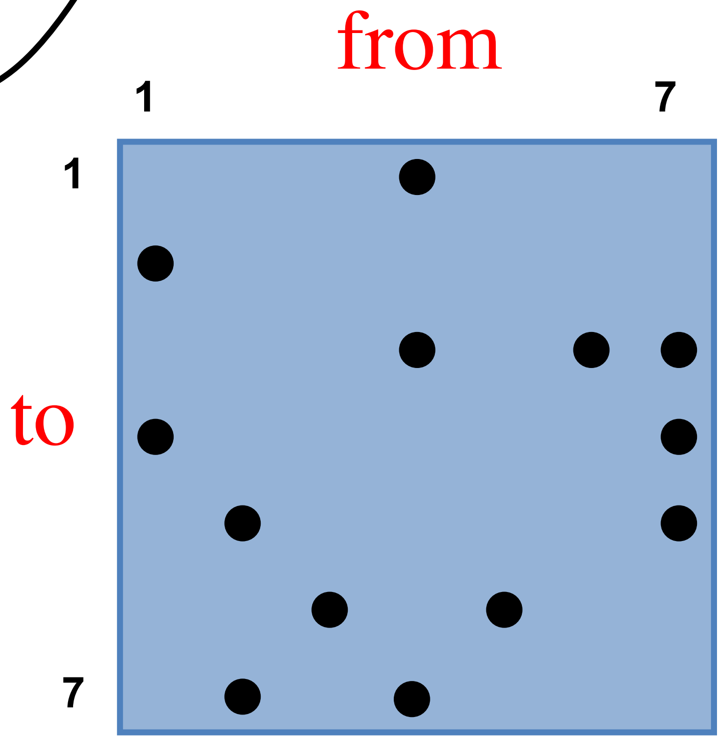


to

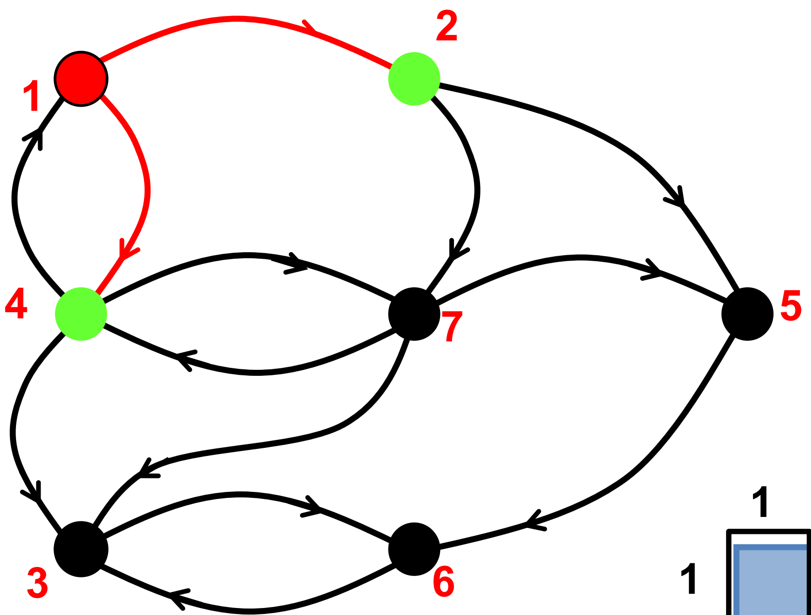




Breadth-first search in the language of matrices



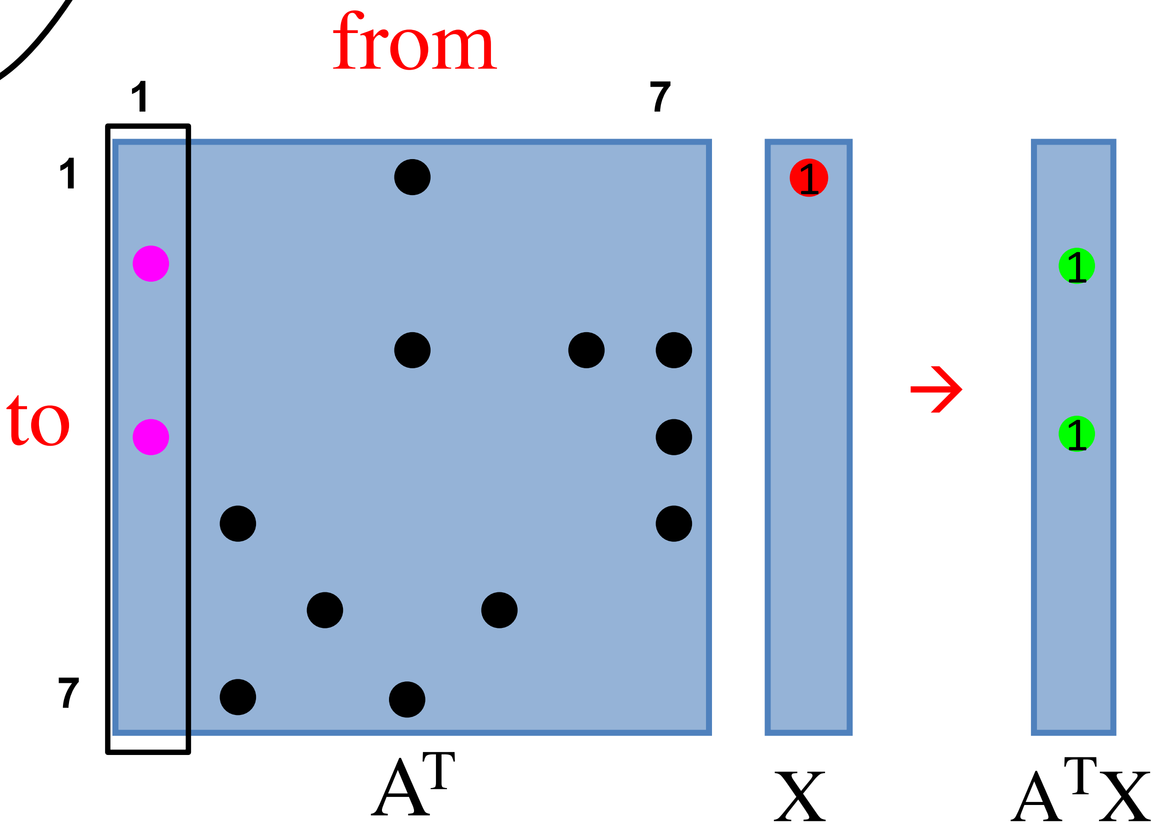
A^T

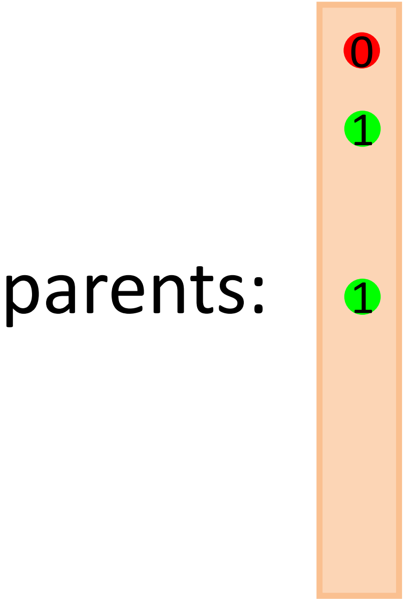


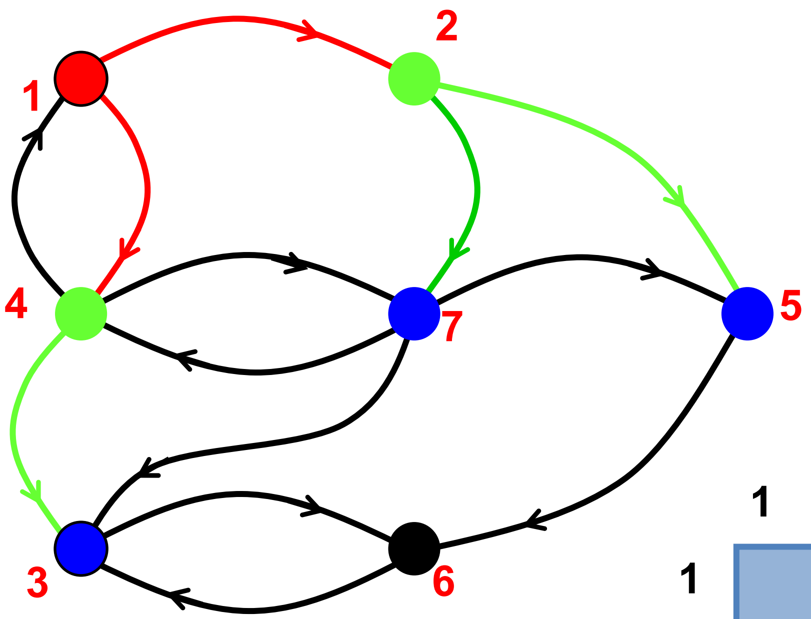
Particular semiring operations:

Multiply: select2nd

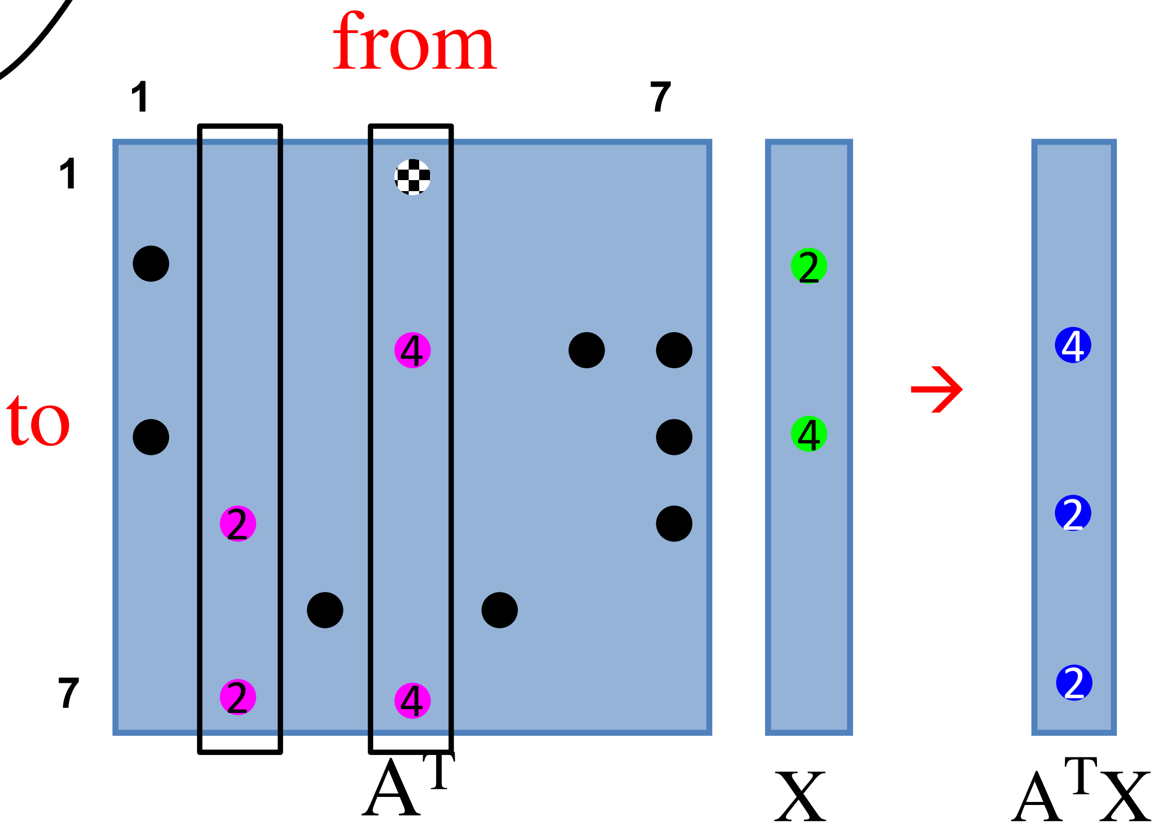
Add: minimum



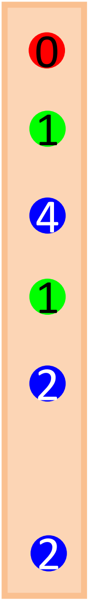




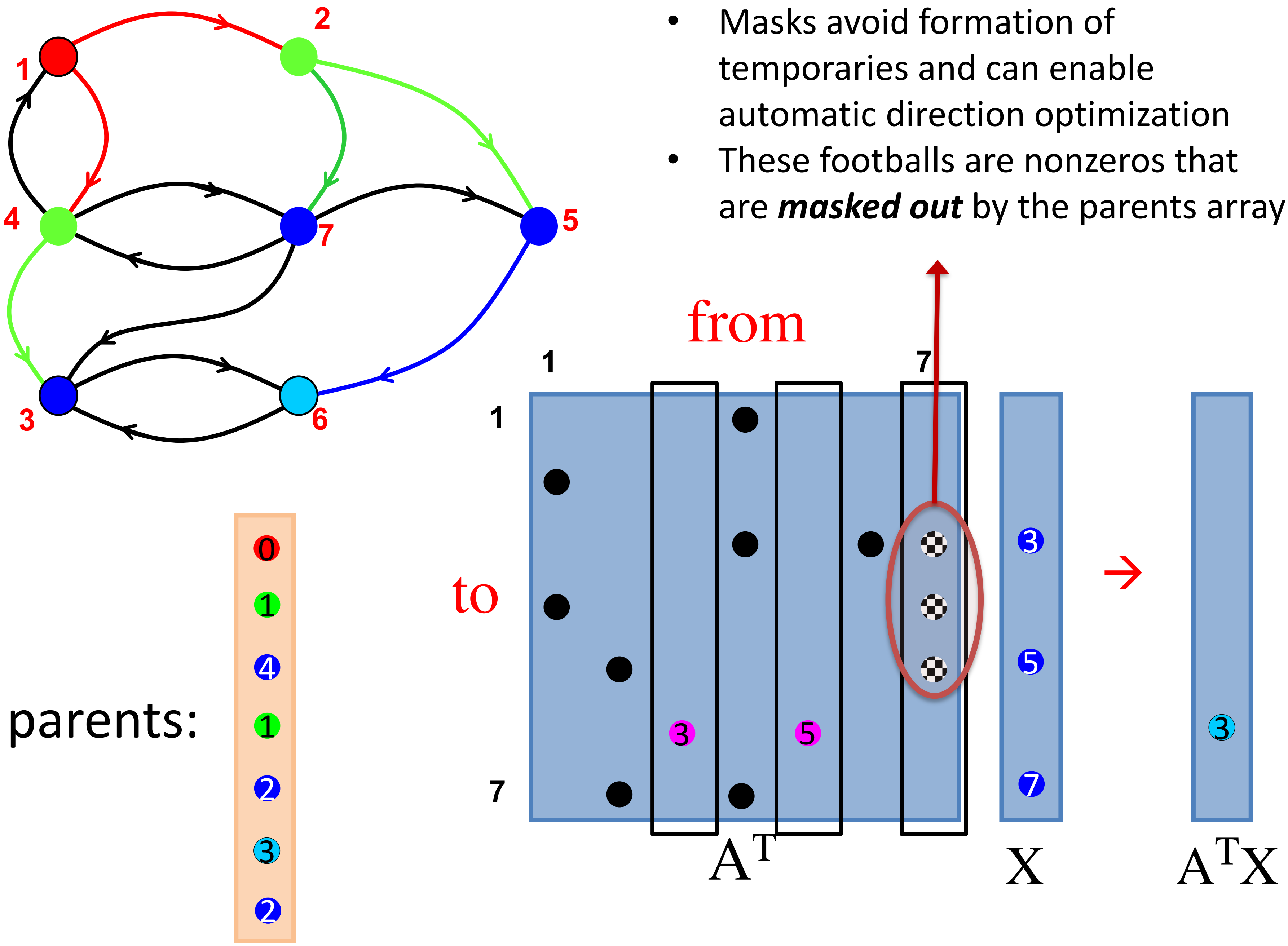
Select vertex with <u>minimum</u> label as parent

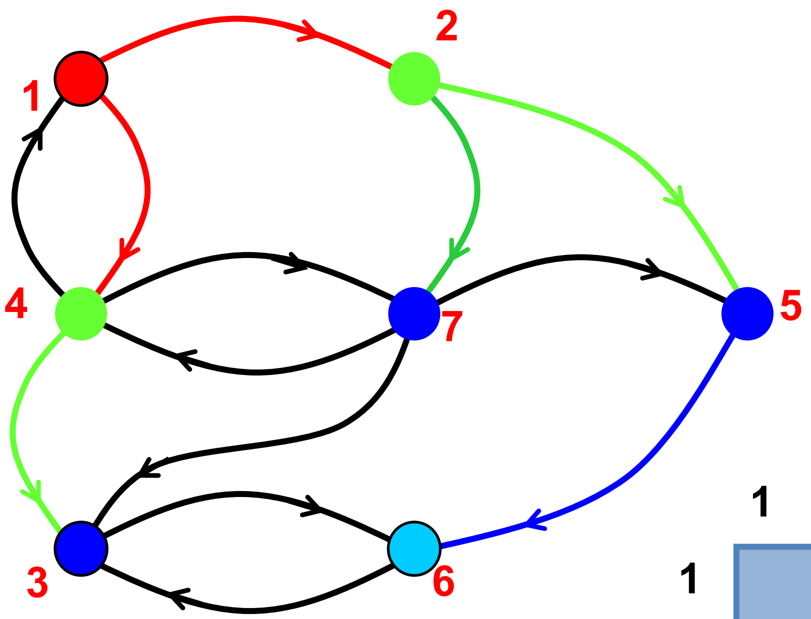


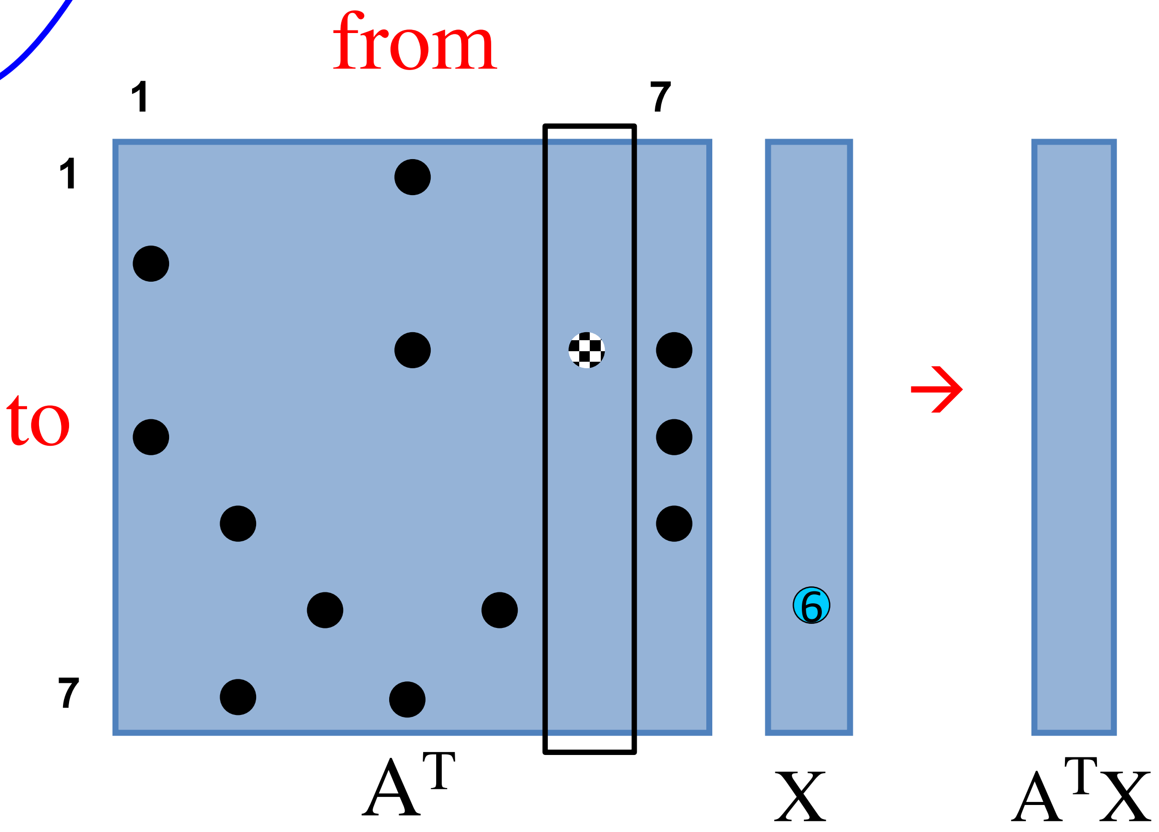
parents:



- Masks avoid formation of temporaries and can enable automatic direction optimization
- These footballs are nonzeros that are ***masked out*** by the parents array



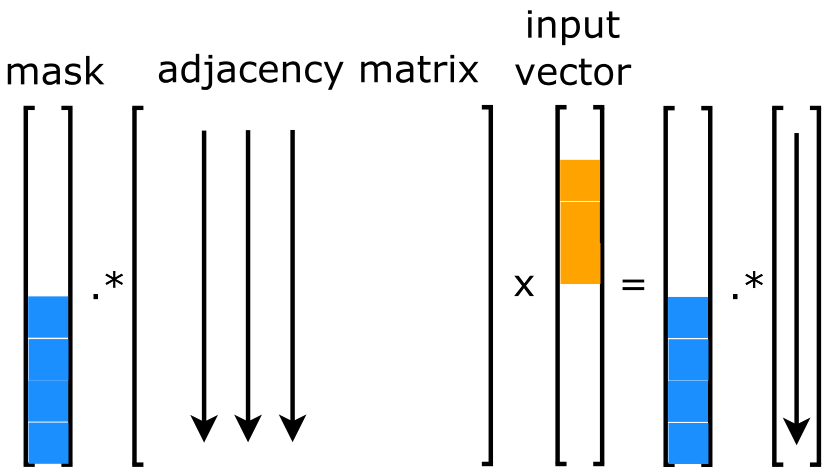

2
1
4
7
5
3
6


from
1
7
1
to
7
6
A^T
X
A^TX

Output sparsity via masks

- The actual operation is $x = A^T x \mathbin{.*} p$

  p is the parents array and .* is elementwise multiplication
- At first, our vision was limited: we only thought about eliminating temporaries in GrB_mxv
- But it was important enough to motivate the inclusion of masks into the GraphBLAS spec, though in limited form



Column-based matvec w/ mask

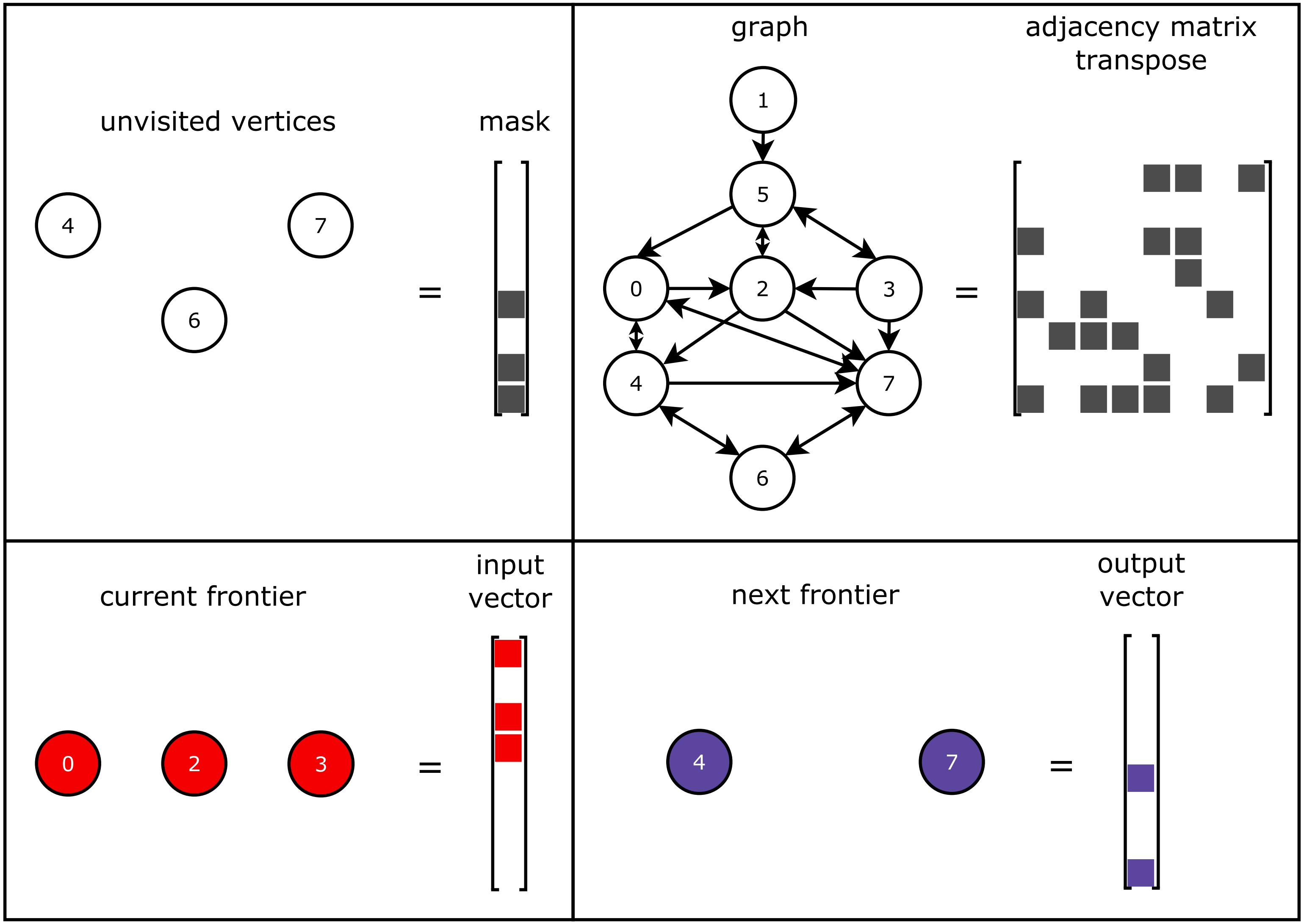
Idea was to run the same column-based algorithm, but checking against a mask before writing to output

Push-pull ≡ column-row matvec

This is a story on how languages (and in this case APIs) change our thinking and drive our creative process

- Carl Yang and I pondered quite a bit on whether it was possible to implement direction optimization in the language of matrices *
- Push-pull (also known as direction optimization) was just about running a row- vs. column-based matvec
- But it wouldn’t be competitive it its pure form because you were pulling from every vertex, not just unexplored ones.
- A year or so later, GraphBLAS had “masks”
- Now it was totally obvious how to make push-pull competitive in GraphBLAS

Enter "masks"



unvisited vertices

mask

graph

adjacency matrix transpose

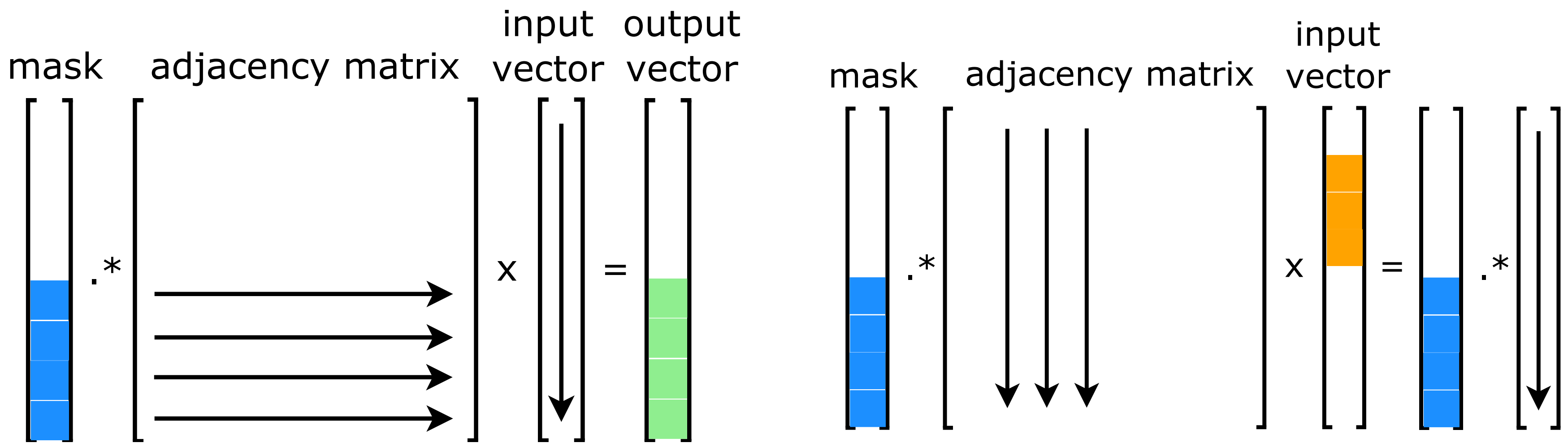
current frontier

input vector

next frontier

output vector

Masks make “pull” implementable competitively in GraphBLAS



Row-based matvec w/ mask

Column-based matvec w/ mask

- **Pull** is better for sufficiently sparse masks; **push** otherwise
- **Claim**: “direction optimization” would have been discovered automatically by the GraphBLAS runtime if we designed the interface back half a decade ago.

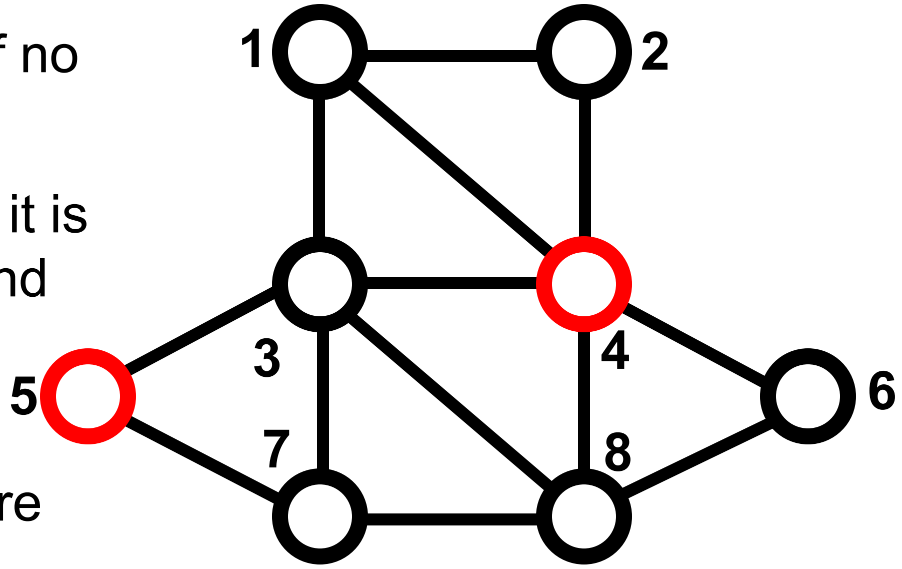
Yang, C., Buluc, A. and Owens, J.D., Implementing Push-Pull Efficiently in GraphBLAS. *ICPP’18*

Breadth-First Search in GraphBLAS

```
GrB_Vector q;                                  // vertices visited in each level
GrB_Vector_new(&q,GrB_BOOL,n);                 // Vector<bool> q(n) = false
GrB_Vector_setElement(q,(bool)true,s);         // q[s] = true, false everywhere else

GrB_Monoid Lor;                                // Logical-or monoid
GrB_Monoid_new(&Lor,GrB_LOR,false);

GrB_Semiring Boolean;                          // Boolean semiring
GrB_Semiring_new(&Boolean,Lor,GrB_LAND);

GrB_Descriptor desc;                           // Descriptor for vxm
GrB_Descriptor_new(&desc);
GrB_Descriptor_set(desc,GrB_MASK,GrB_SCMP);    // invert the mask
GrB_Descriptor_set(desc,GrB_OUTP,GrB_REPLACE); // clear the output before assignment

GrB_UnaryOp apply_level;
GrB_UnaryOp_new(&apply_level,return_level,GrB_INT32,GrB_BOOL);

/*
 * BFS traversal and label the vertices.
 */
level = 0;
GrB_Index nvals;
do {
  ++level;                                     // next level (start with 1)
  GrB_apply(*v,GrB_NULL,GrB_PLUS_INT32,apply_level,q,GrB_NULL);   // v[q] = level
  GrB_vxm(q,*v,GrB_NULL,Boolean,q,A,desc);     // q[!v] = q ||.&& A ; finds all the
                                               // unvisited successors from current q
  GrB_Vector_nvals(&nvals, q);
} while (nvals);                               // if there is no successor in q, we are done.
```

Maximal Independent Set

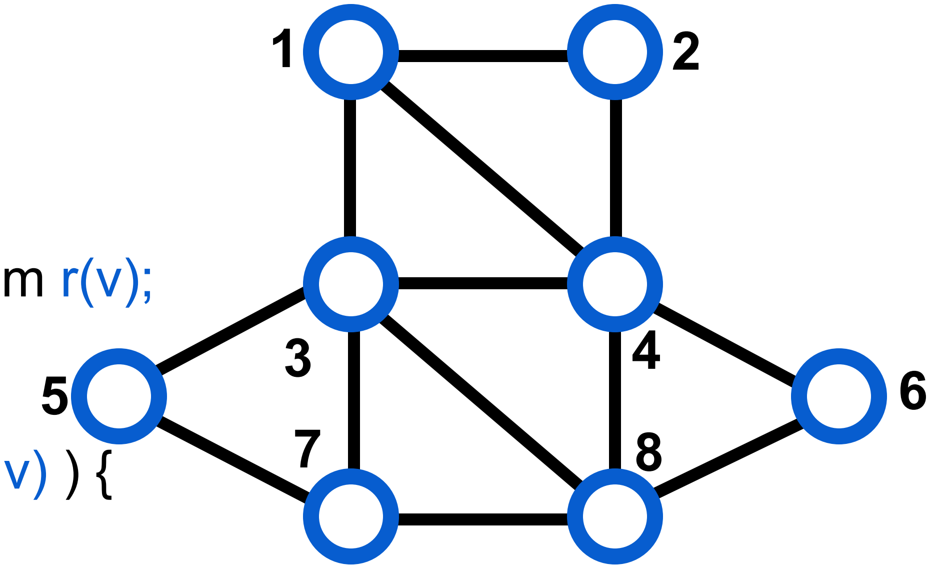
- Graph with vertices V = {1,2,…,n}
- A set S of vertices is independent if no two vertices in S are neighbors.
- An independent set S is maximal if it is impossible to add another vertex and stay independent
- An independent set S is maximum if no other independent set has more vertices
- Finding a *maximum* independent set is intractably difficult (NP-hard)
- Finding a *maximal* independent set is easy, at least on one processor.



The set of red vertices
S = {4, 5} is *independent*
and is *maximal*
but not *maximum*

Parallel, Randomized MIS Algorithm

```
S = empty set;  C = V;
while  C  is not empty {
    label each v in C with a random r(v);
    for all v in C in parallel {
        if r(v) < min( r(neighbors of v) ) {
            move v from C to S;
            remove neighbors of v from C;
        }
    }
}
```



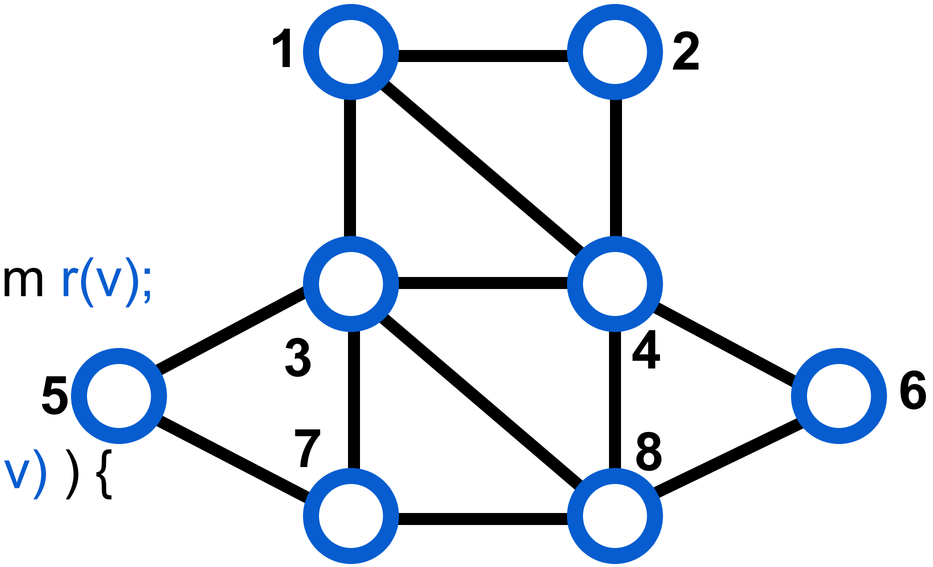
S = { }

C = { 1, 2, 3, 4, 5, 6, 7, 8 }

M. Luby. "A Simple Parallel Algorithm for the Maximal Independent Set Problem". *SIAM Journal on Computing* **15** (4): 1036–1053, 1986

Parallel, Randomized MIS Algorithm

```
S = empty set;  C = V;
while  C  is not empty {
    label each v in C with a random r(v);
    for all v in C in parallel {
        if r(v) < min( r(neighbors of v) ) {
            move v from C to S;
            remove neighbors of v from C;
        }
    }
}
```

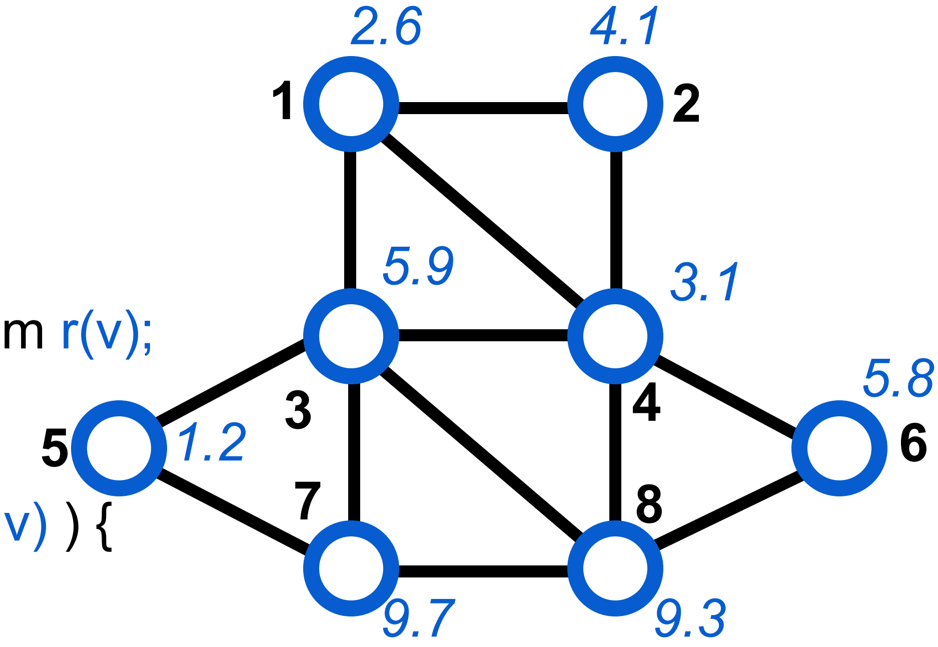


S = { }

C = { 1, 2, 3, 4, 5, 6, 7, 8 }

Parallel, Randomized MIS Algorithm

```
S = empty set;  C = V;
while  C  is not empty {
    label each v in C with a random r(v);
    for all v in C in parallel {
        if r(v) < min( r(neighbors of v) ) {
            move v from C to S;
            remove neighbors of v from C;
        }
    }
}
```

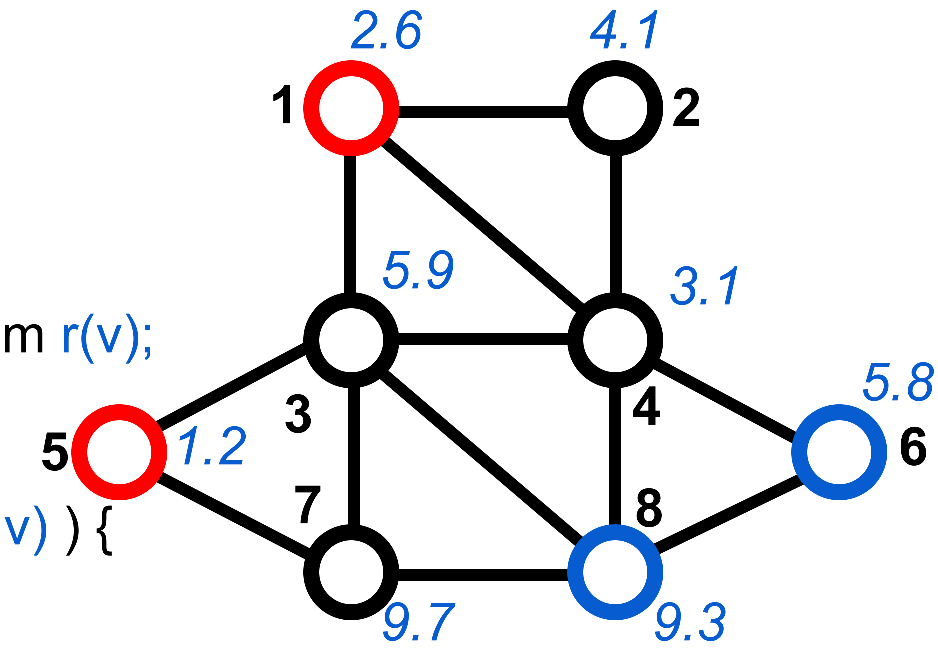


S = { }

C = { 1, 2, 3, 4, 5, 6, 7, 8 }

Parallel, Randomized MIS Algorithm

```
S = empty set;  C = V;
while  C  is not empty {
    label each v in C with a random r(v);
    for all v in C in parallel {
        if r(v) < min( r(neighbors of v) ) {
            move v from C to S;
            remove neighbors of v from C;
        }
    }
}
```

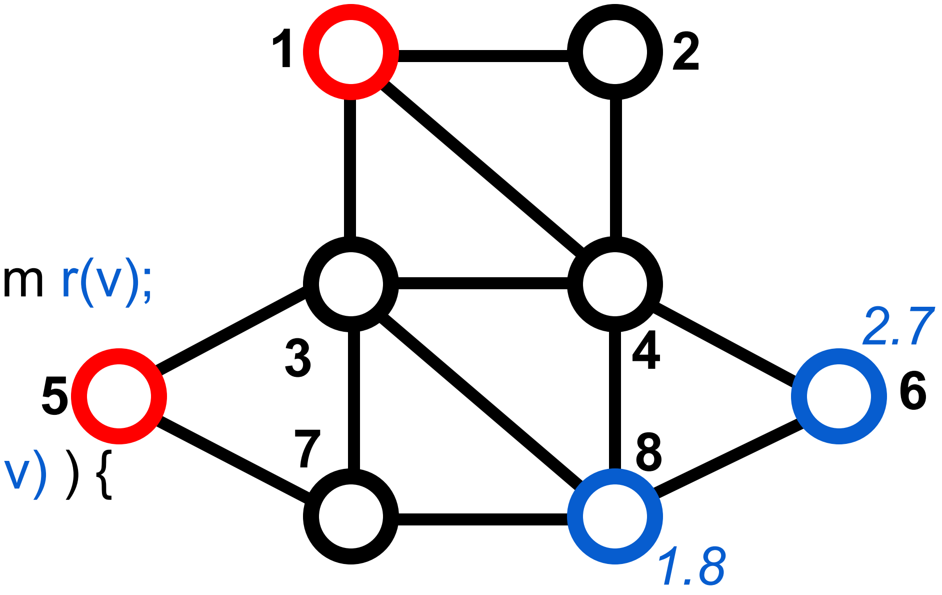


S = { 1, 5 }

C = { 6, 8 }

Parallel, Randomized MIS Algorithm

```
S = empty set;  C = V;
while  C  is not empty {
    label each v in C with a random r(v);
    for all v in C in parallel {
        if r(v) < min( r(neighbors of v) ) {
            move v from C to S;
            remove neighbors of v from C;
        }
    }
}
```

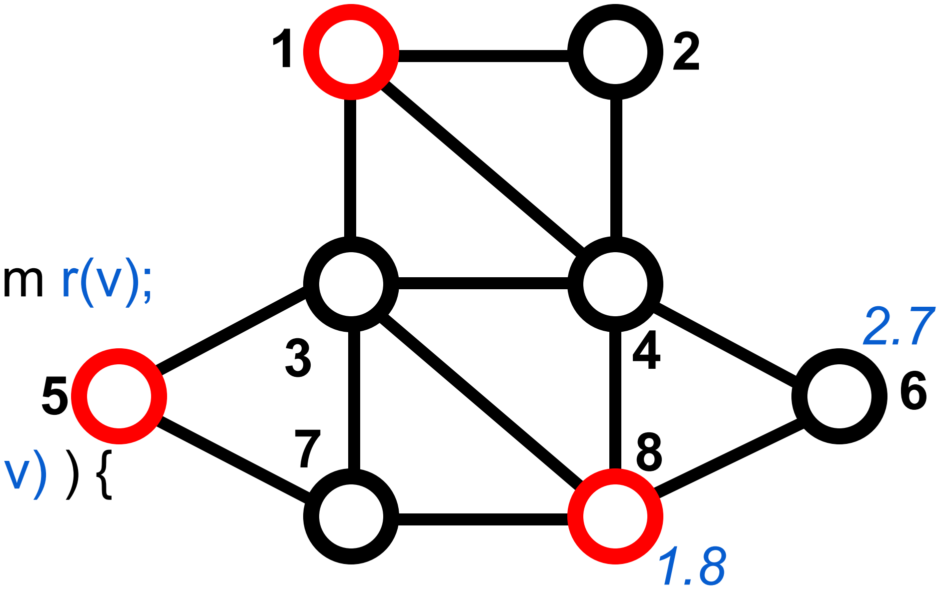


S = { 1, 5 }

C = { 6, 8 }

Parallel, Randomized MIS Algorithm

```
S = empty set;  C = V;
while  C  is not empty {
    label each v in C with a random r(v);
    for all v in C in parallel {
        if r(v) < min( r(neighbors of v) ) {
            move v from C to S;
            remove neighbors of v from C;
        }
    }
}
```

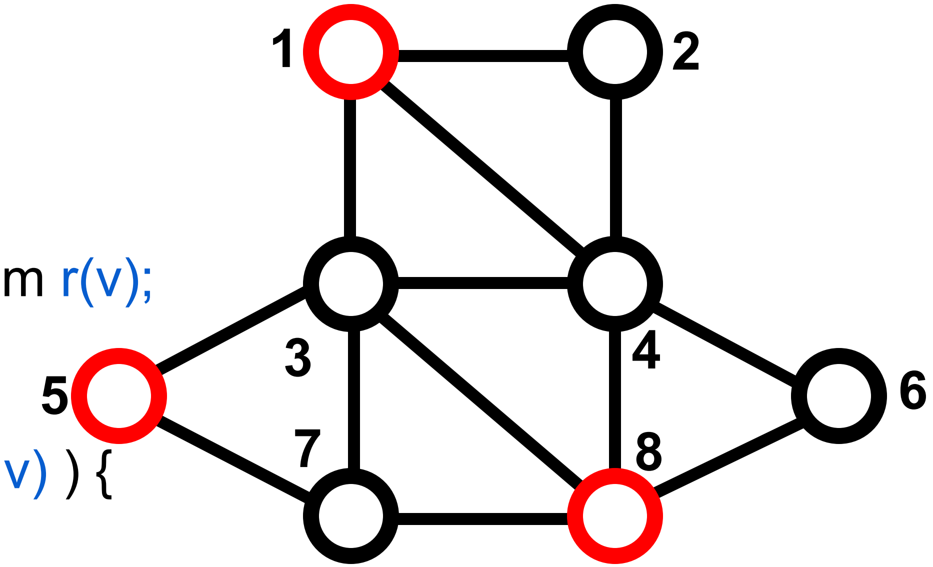


S = { 1, 5, 8 }

C = { }

Parallel, Randomized MIS Algorithm

```
S = empty set;  C = V;
while  C  is not empty {
    label each v in C with a random r(v);
    for all v in C in parallel {
        if r(v) < min( r(neighbors of v) ) {
            move v from C to S;
            remove neighbors of v from C;
        }
    }
}
```



Theorem: This algorithm "very probably" finishes within O(log n) rounds.

work **~ O(n log n), but** ***span*** **~O(log n)**

A Variant of Luby's Algorithm in GraphBLAS

```
// Iterate while there are candidates to check.
GrB_Index nvals;
GrB_Vector_nvals(&nvals, candidates);
while (nvals > 0) {
  // compute a random probability scaled by inverse of degree
  GrB_apply(prob,candidates,GrB_NULL,set_random,degrees,r_desc);

  // compute the max probability of all neighbors
  GrB_mxv(neighbor_max,candidates,GrB_NULL,maxSelect2nd,A,prob,r_desc);

  // select vertex if its probability is larger than all its active neighbors,
  // and apply a ”masked no-op” to remove stored falses
  GrB_eWiseAdd(new_members,GrB_NULL,GrB_NULL,GrB_GT_FP64,prob,neighbor_max,GrB_NULL);
  GrB_apply(new_members,new_members,GrB_NULL,GrB_IDENTITY_BOOL,new_members,r_desc);

  // add new members to independent set.
  GrB_eWiseAdd(*iset,GrB_NULL,GrB_NULL,GrB_LOR,*iset,new_members,GrB_NULL);

  // remove new members from set of candidates c = c & !new
  GrB_eWiseMult(candidates,new_members,GrB_NULL,
                GrB_LAND,candidates,candidates,sr_desc);

  GrB_Vector_nvals(&nvals, candidates);
  if (nvals == 0) { break; }                  // early exit condition

  // Neighbors of new members can also be removed from candidates
  GrB_mxv(new_neighbors,candidates,GrB_NULL,Boolean,A,new_members,GrB_NULL);
  GrB_eWiseMult(candidates,new_neighbors,GrB_NULL,
                GrB_LAND,candidates,candidates,sr_desc);

  GrB_Vector_nvals(&nvals, candidates);
}
```

http://graphblas.org

Pattern 2: Sparse matrix times sparse matrix (SpGEMM)

Multi-source traversal:

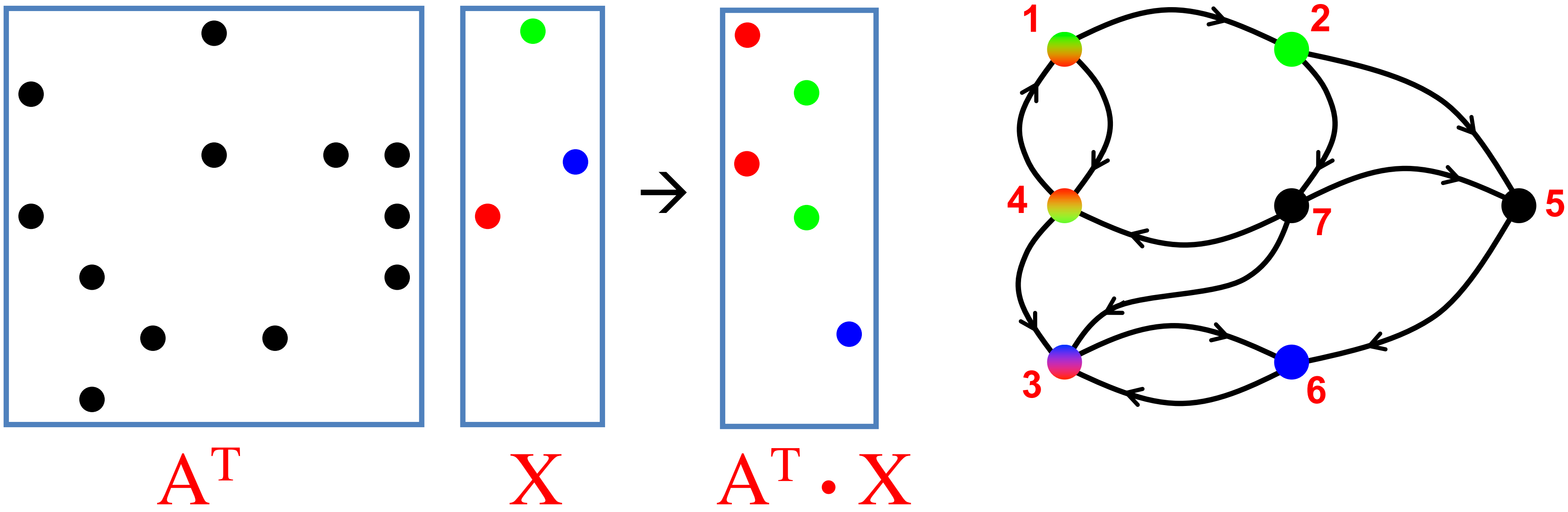
Ex: multi-source BFS, betweenness centrality, triangle counting*, Markov clustering*

GrB_mxm(Y, P, <semiring>, A, X, <desc>)

A: sparse adjacency matrix

X: sparse input matrix (previous frontier), n-by-b where b is the #sources

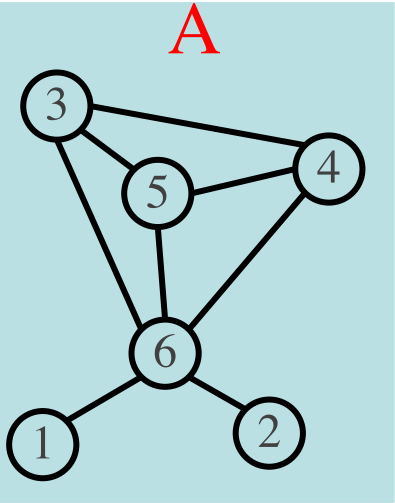
P: mask (already discovered vertices), multi-vector version of p from previous slide



Pattern 2: Sparse matrix times sparse matrix (SpGEMM)

Triangle counting is also multi-source(in fact, all sources) traversal:
It just stops after one traversal iteration only, discovering all wedges

`GrB_mxm(C, A, <semiring>, L, U, <desc>)`

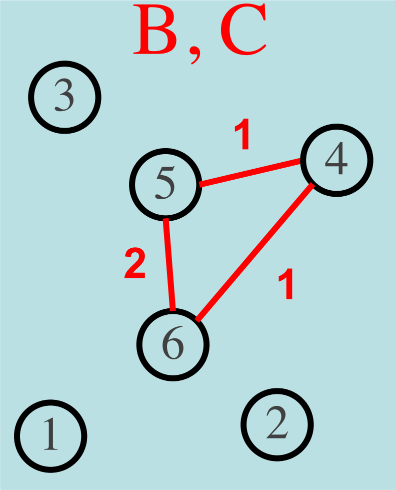


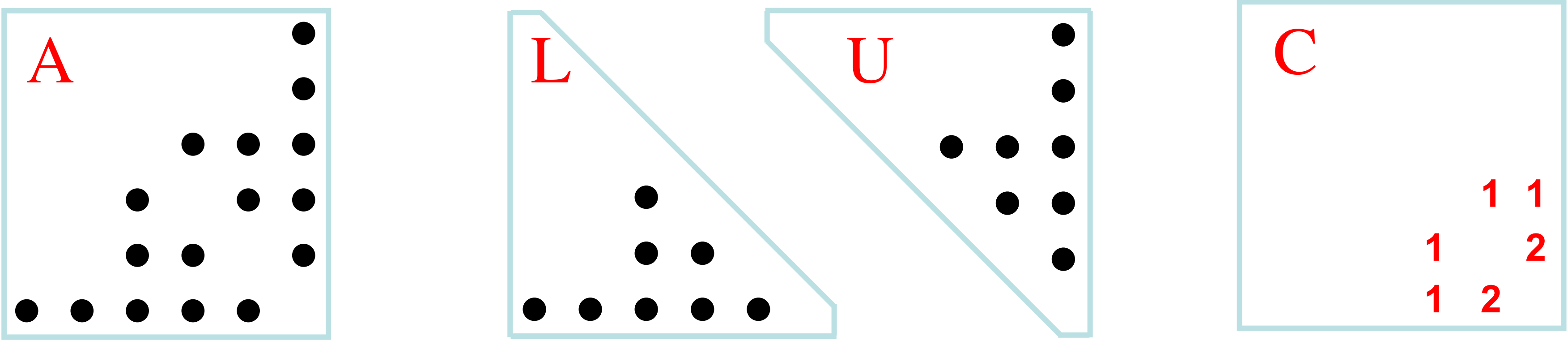
A = L + U (hi->lo + lo->hi)

L × U = B (wedge, low hinge)

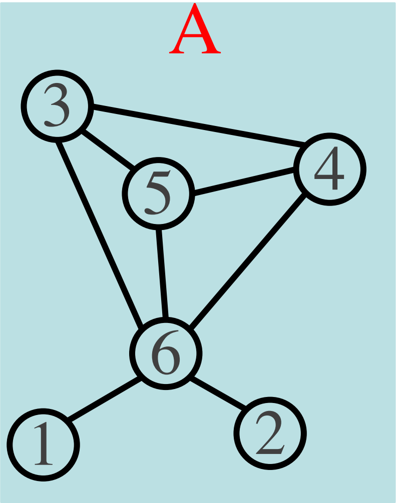
A ^ B = C (closed wedge)

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





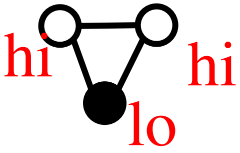
Counting triangles



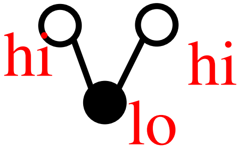
Clustering coefficient:

- 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

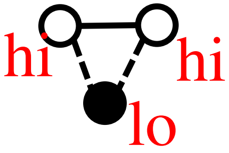
Cohen’s algorithm to count triangles:



- Count triangles by lowest-degree vertex.



- Enumerate “low-hinged” wedges.



- Keep wedges that close.

Triangle Counting in GraphBLAS

```
/*
 * Given, L, the lower triangular portion of n x n adjacency matrix A (of and
 * undirected graph), computes the number of triangles in the graph.
 */
uint64_t triangle_count(GrB_Matrix L)                // L: NxN, lower-triangular, bool
{
  GrB_Index n;
  GrB_Matrix_nrows(&n, L);                           // n = # of vertices

  GrB_Matrix C;
  GrB_Matrix_new(&C, GrB_UINT64, n, n);

  GrB_Monoid UInt64Plus;                             // integer plus monoid
  GrB_Monoid_new(&UInt64Plus,GrB_PLUS_UINT64,0ul);

  GrB_Semiring UInt64Arithmetic;                     // integer arithmetic semiring
  GrB_Semiring_new(&UInt64Arithmetic,UInt64Plus,GrB_TIMES_UINT64);

  GrB_Descriptor desc_tb;                            // Descriptor for mxm
  GrB_Descriptor_new(&desc_tb);
  GrB_Descriptor_set(desc_tb,GrB_INP1,GrB_TRAN); // transpose the second matrix

  GrB_mxm(C, L, GrB_NULL, UInt64Arithmetic, L, L, desc_tb); // C<L> = L *.+ L'

  uint64_t count;
  GrB_reduce(&count, GrB_NULL, UInt64Plus, C, GrB_NULL);    // 1-norm of C

  GrB_free(&C);                       // C matrix no longer needed
  GrB_free(&UInt64Arithmetic);        // Semiring no longer needed
  GrB_free(&UInt64Plus);              // Monoid no longer needed
  GrB_free(&desc_tb);                 // descriptor no longer needed

  return count;
}
```

http://graphblas.org

High-level outline

- Sparse matrices for graph algorithms
- **Sparse matrices for computational biology**
- Sparse matrices for machine learning
- Parallel algorithms for sparse matrix primitives
- Available software

10 minutes break

- Coffee
- Bathroom
- Mingling

Genome assembly pipeline

Input

Read ○

Overlap Detection

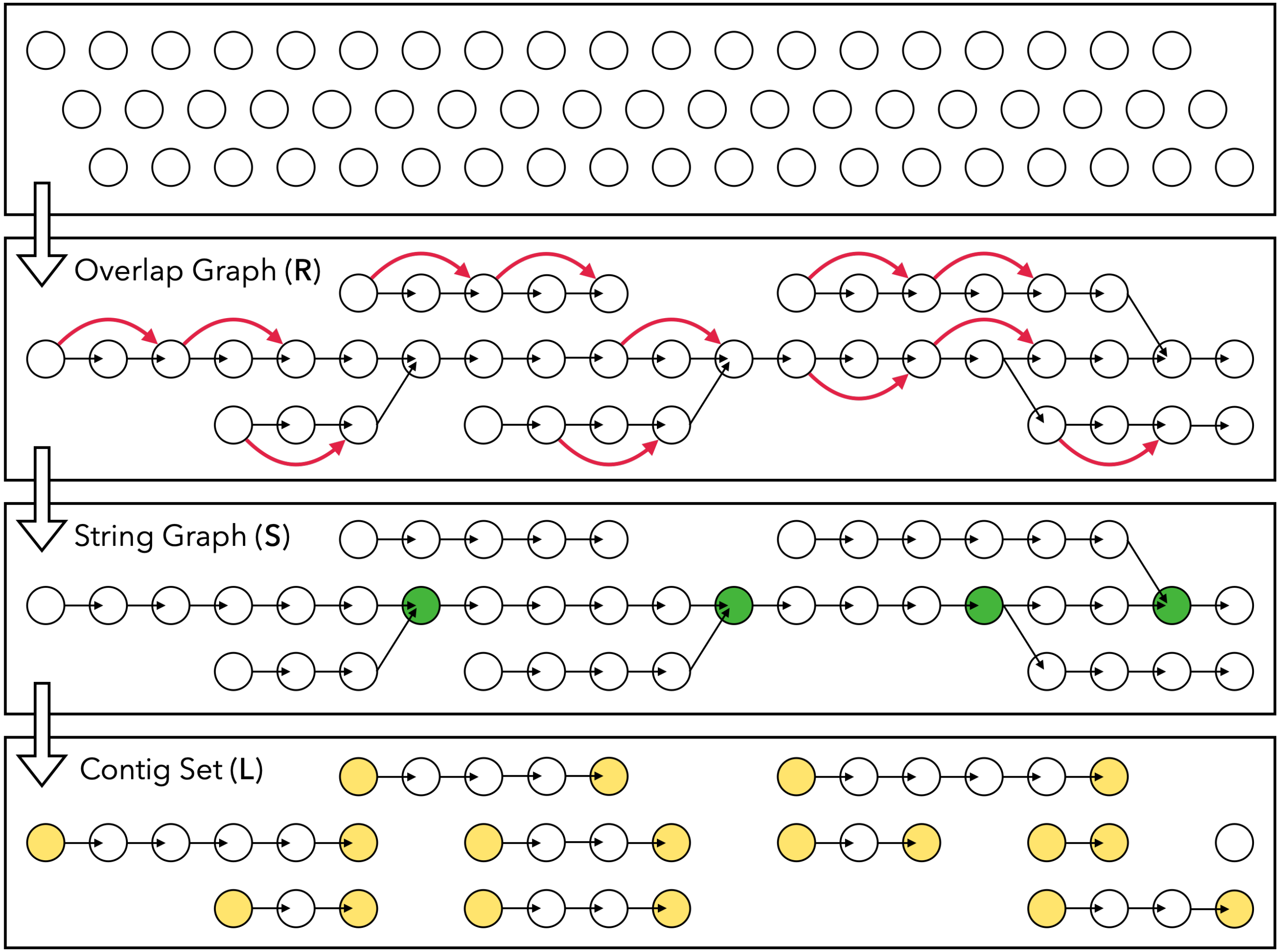
Transitive Edge →

Transitive Reduction

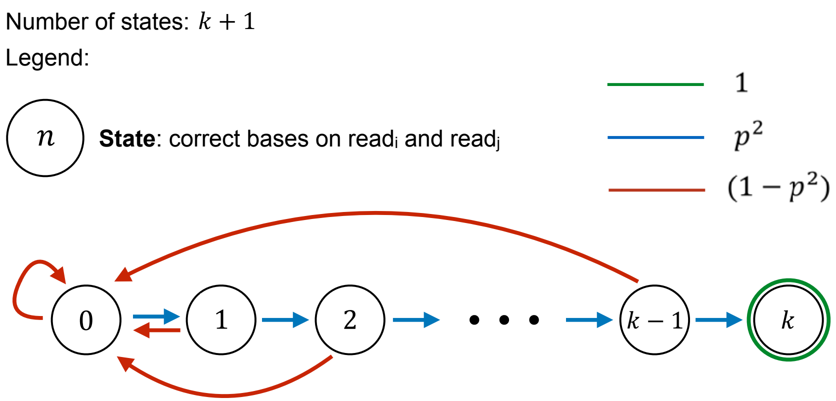
Branch Vertex 🟢

Contig Generation

Root Vertex 🟡



BELLA: Berkeley Long-read to Long-read Aligner and Overlapper

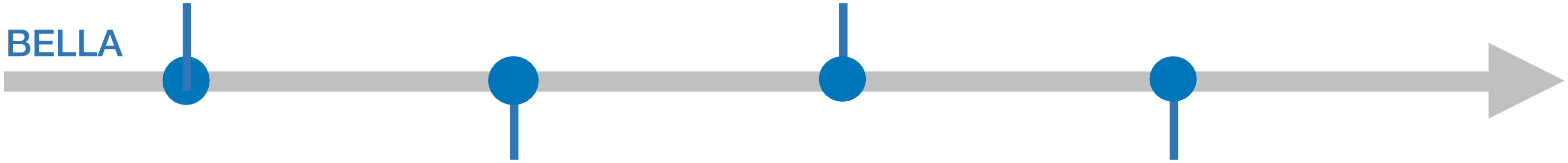


Number of states: $k+1$

Legend:

State: correct bases on read_i and read_j

- How to choose the right set of k-mers, otherwise there are too many of them?
- How to use alignment score to tell true alignments from false positives?



BELLA

Feasibility of a *k*-mer seed based approach

Novel procedure for pruning *k*-mers

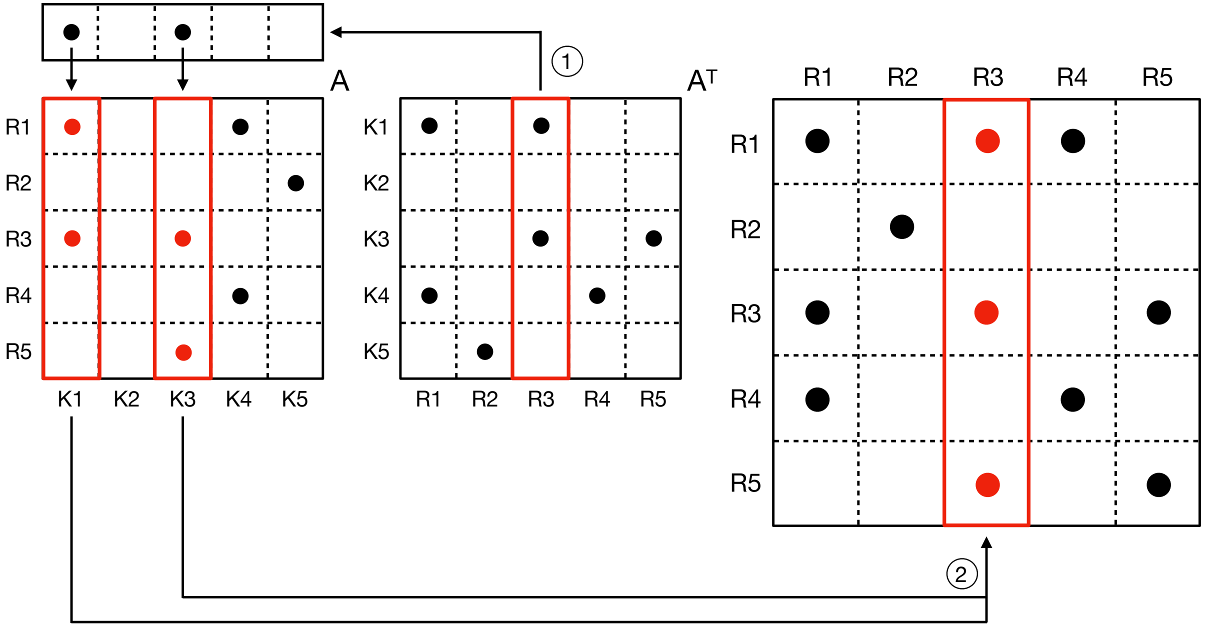
Overlap detection via sparse matrix multiplication

Seed-and-extend pairwise alignment

Guidi G, Ellis M, Rokhsar D, Yelick K, Buluç A. BELLA: Berkeley Efficient Long-Read to Long-Read Aligner and Overlapper. SIAM Conference on Applied and Computational Discrete Algorithms (ACDA), 2021

SpGEMM use case #1: read overlapping

- **Overlapping** is the most computationally expensive step in the overwhelming majority of long read assemblers.
- Imagine each read is a sample, its k-mer profile is its feature set
- Create a reads-by-kmers (sparse) matrix



$AA^T(i,j)$ = # shared k-mers between reads i and j, plus their positions in the reads

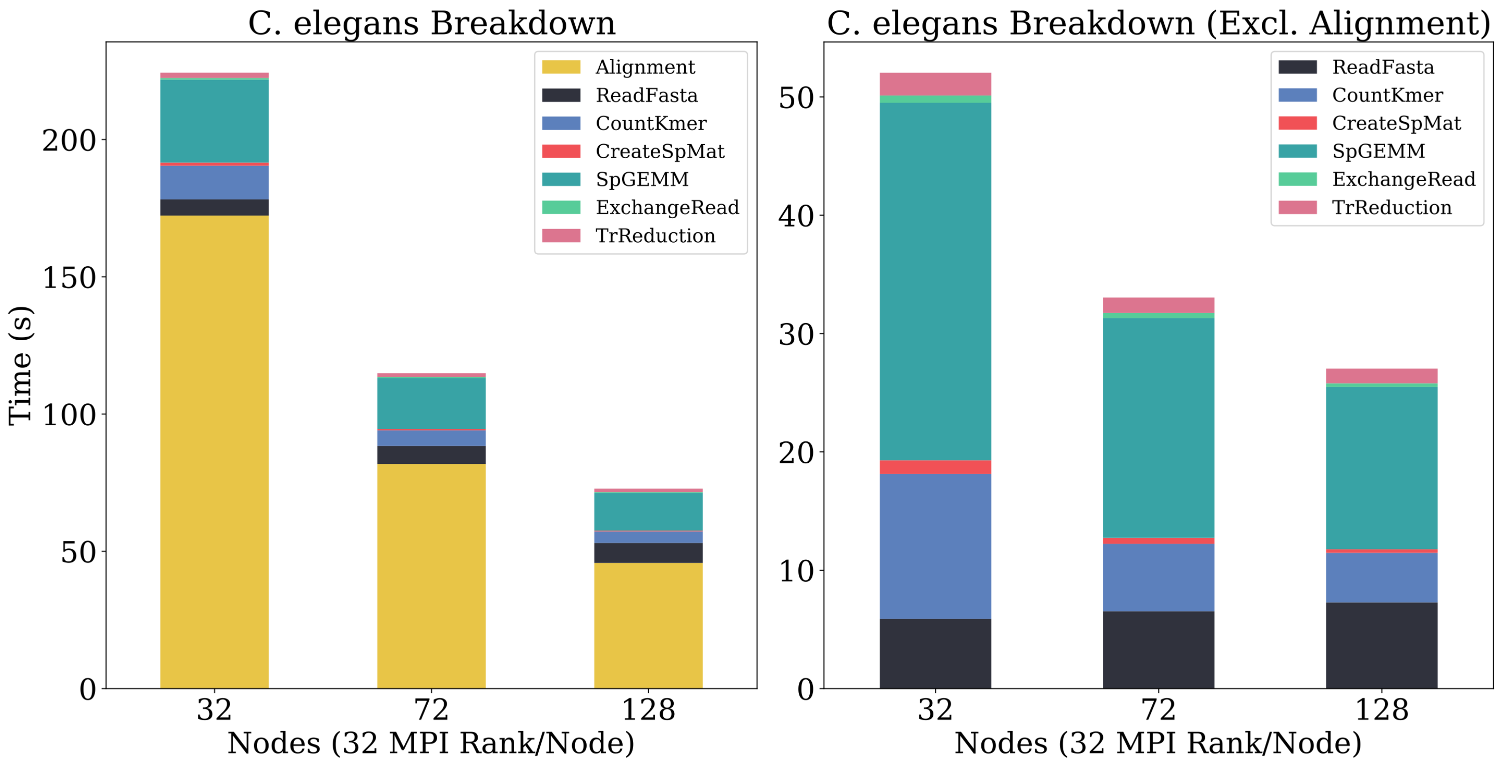
Use any fast SpGEMM algorithm, as long as it runs on ***arbitrary semirings***

diBELLA.2D performance results

diBELLA.2D: distributed-memory version of BELLA on 2D process grid

Performs *overlap detection* plus *transitive reduction (overlap to string graph)*

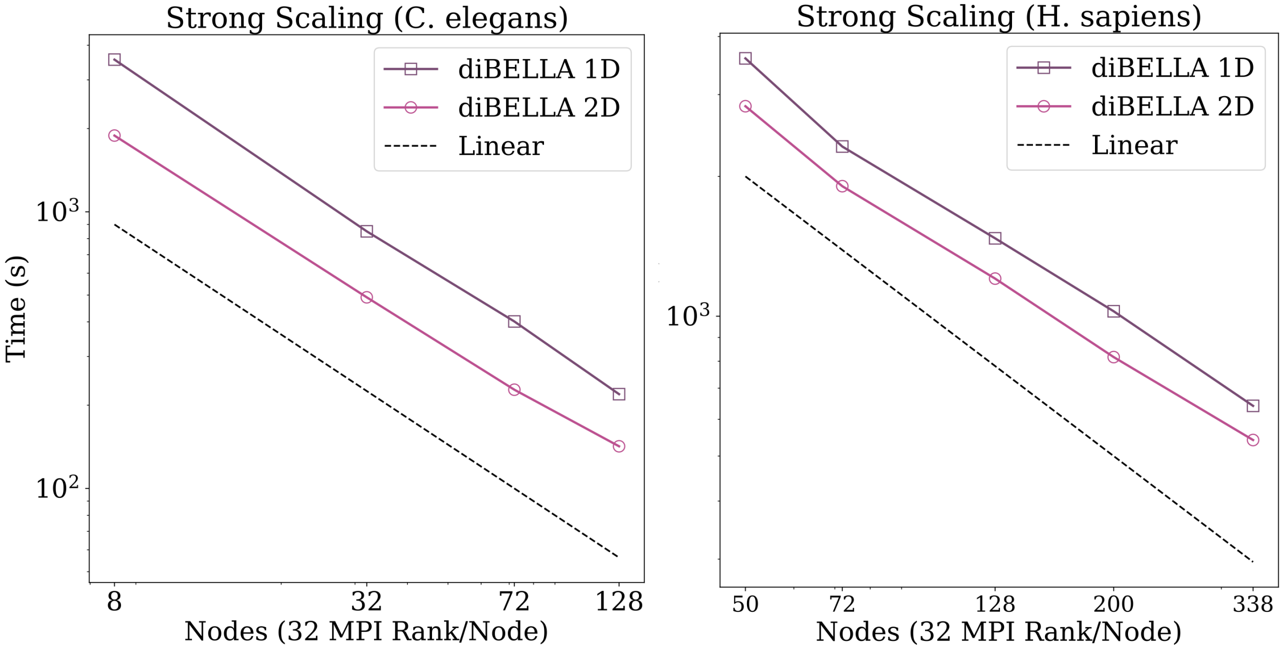
https://github.com/PASSIONLab/diBELLA.2D



Giulia Guidi, Oguz Selvitopi, Marquita Ellis, Leonid Oliker, Katherine Yelick, Aydin Buluç. Parallel String Graph Construction and Transitive Reduction for De Novo Genome Assembly. *IPDPS* 2021

Is the sparse matrix approach better?

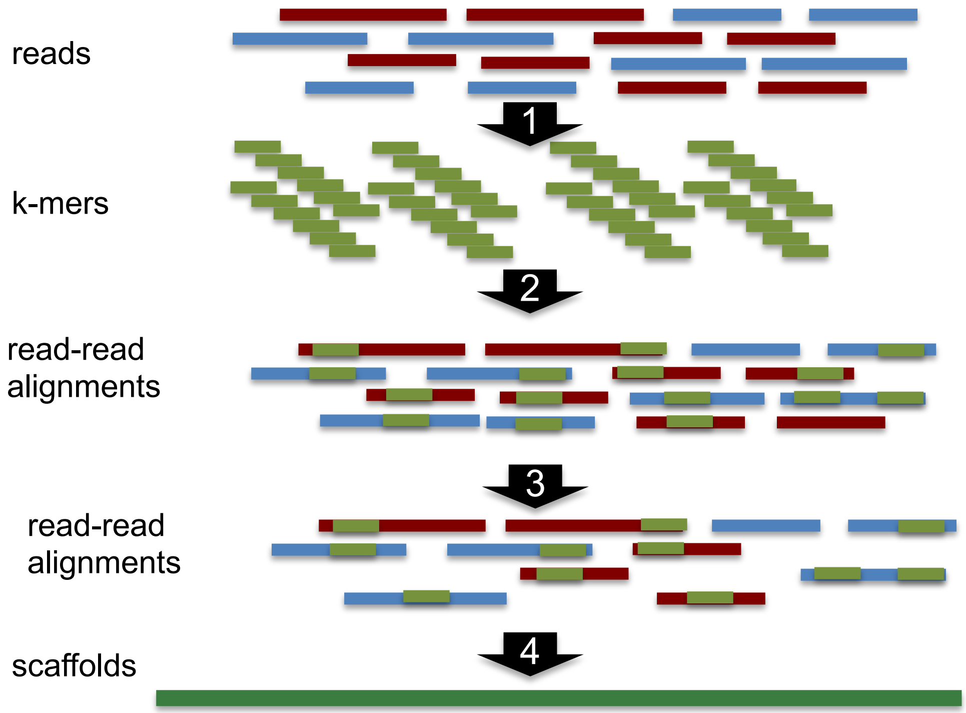
- Comparing the sparse matrix abstraction (diBELLA 2D [2], **weeks** of effort) with a direct implementation (diBELLA 1D [1], **years** of effort). Both use MPI
- Sparse matrices reduce communication via 2D sparse SpGEMM



[1] Marquita Ellis, Giulia Guidi, Aydin Buluç, Leonid Oliker, and Katherine Yelick. "diBELLA: Distributed long read to long read alignment." ICPP 2019

[2] Giulia Guidi, Oguz Selvitopi, Marquita Ellis, Leonid Oliker, Katherine Yelick, Aydin Buluç. Parallel String Graph Construction and Transitive Reduction for De Novo Genome Assembly. *IPDPS* 2021

Sparse matrix approach for assembly with long reads



1) K-mer Analysis
K-mer histogram

2) Sparse matrix building
A: reads-by-kmers

3) Overlapping via SpGEMM C $= AA^T$: reads-by-reads

4) X-drop alignments
M = filter(C, alignment_score)

5) Transitive Reduction
$M^{i+1} = \text{Prune}(M^i M^i \odot M^i)$

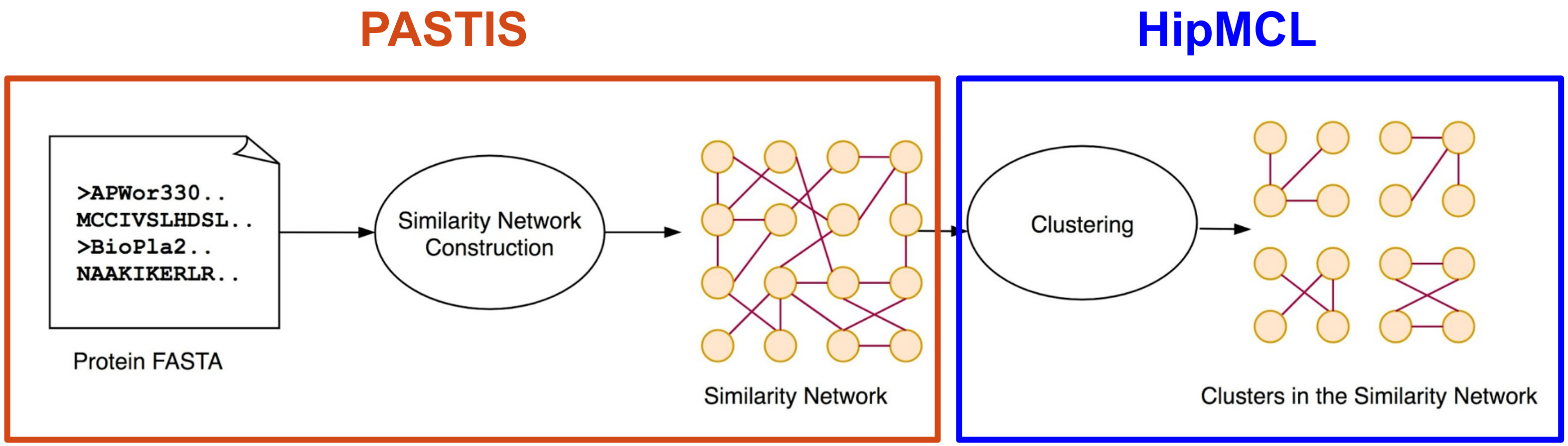
6) Contig generation [3]

- ○ Remove forks
- ○ Find connected components (CCs)
- ○ Local traversal of CCs

[3] Giulia Guidi, Gabriel Raulet, Daniel Rokhsar, Leonid Oliker, Katherine Yelick, and Aydın Buluç. Distributed-memory parallel contig generation for de novo long-read genome assembly. In ICPP, 2022

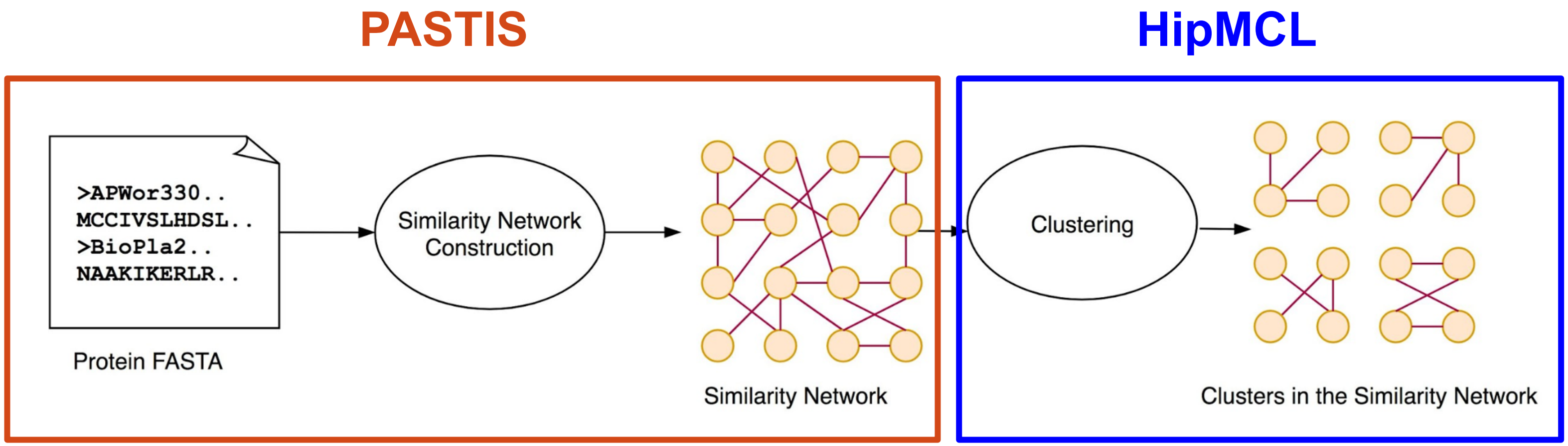
Protein Family Identification

- Problem: Given a large collection of proteins, identify groups of proteins that are homologous (i.e. descended from a common ancestor).
- Homologous proteins often have the same function (think of different variants of hemoglobin in many species)
- Often, only sequences (and not structure) of the proteins are available, so we infer homology via sequence similarity



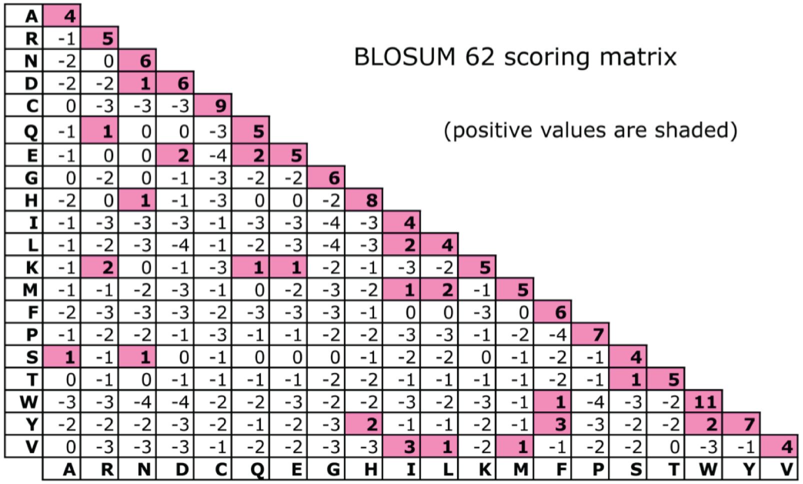
Protein Family Identification

- Many one-step approaches are possible that trade accuracy for lower memory consumption and faster execution (e.g., CD-HIT).
- The approach that seems to lead to highest accuracy:
  - Construct a similarity network over protein sequences using many-to-many sequence search (PASTIS)
  - Cluster this network to discover possible protein families (HipMCL)



SpGEMM use case #2: many-to-many protein alignment

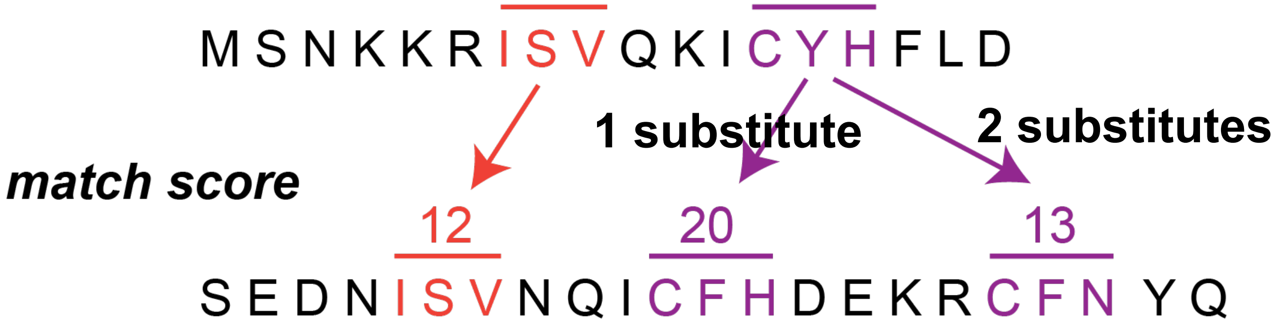
- Idea similar to BELLA, but removing the exact match restriction
- For homology detection, need to catch weaker signal (~30% ANI)
- K-mers with substitutes may be more valuable than exact matches!



BLOSUM 62 scoring matrix

(positive values are shaded)

A	4																			
R	-1	5																		
N	-2	0	6																	
D	-2	-2	1	6																
C	0	-3	-3	-3	9															
Q	-1	1	0	0	-3	5														
E	-1	0	0	2	-4	2	5													
G	0	-2	0	-1	-3	-2	-2	6												
H	-2	0	1	-1	-3	0	0	-2	8											
I	-1	-3	-3	-3	-1	-3	-3	-4	-3	4										
L	-1	-2	-3	-4	-1	-2	-3	-4	-3	2	4									
K	-1	2	0	-1	-3	1	1	-2	-1	-3	-2	5								
M	-1	-1	-2	-3	-1	0	-2	-3	-2	1	2	-1	5							
F	-2	-3	-3	-3	-2	-3	-3	-3	-1	0	0	-3	0	6						
P	-1	-2	-2	-1	-3	-1	-1	-2	-2	-3	-3	-1	-2	-4	7					
S	1	-1	1	0	-1	0	0	0	-1	-2	-2	0	-1	-2	-1	4				
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	1	5			
W	-3	-3	-4	-4	-2	-2	-3	-2	-2	-3	-2	-3	-1	1	-4	-3	-2	11		
Y	-2	-2	-2	-3	-2	-1	-2	-3	2	-1	-1	-2	-1	3	-3	-2	-2	2	7	
V	0	-3	-3	-3	-1	-2	-2	-3	-3	3	1	-2	1	-1	-2	-2	0	-3	-1	4
	A	**R**	**N**	**D**	**C**	**Q**	**E**	**G**	**H**	**I**	**L**	**K**	**M**	**F**	**P**	**S**	**T**	**W**	**Y**	**V**



M S N K K R I S V Q K I C Y H F L D

1 substitute

2 substitutes

match score

12

20

13

S E D N I S V N Q I C F H D E K R C F N Y Q

SpGEMM for many-to-many protein alignment

PASTIS (https://github.com/PASSIONLab/PASTIS) does distributed many-to-many protein sequence similarity search using sparse matrices

Introduce new sparse matrix **S**

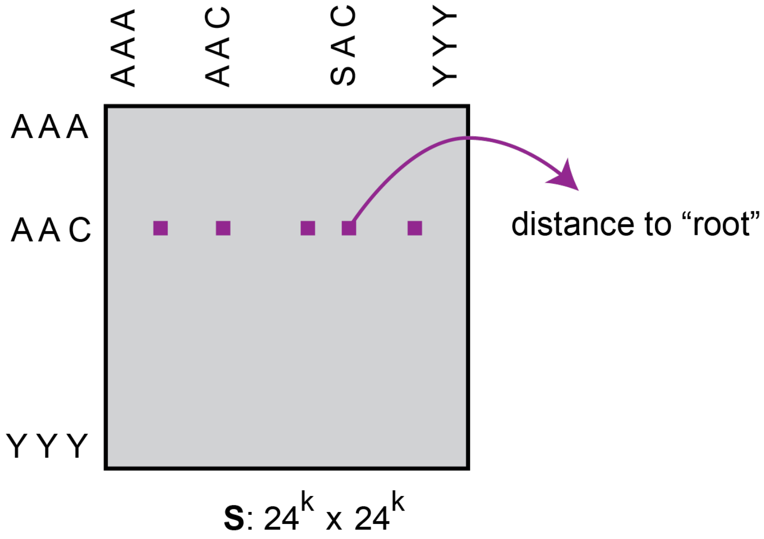
Contains substitution information

Each entry has **substitution cost**

Exact k-mers → C=AA^T

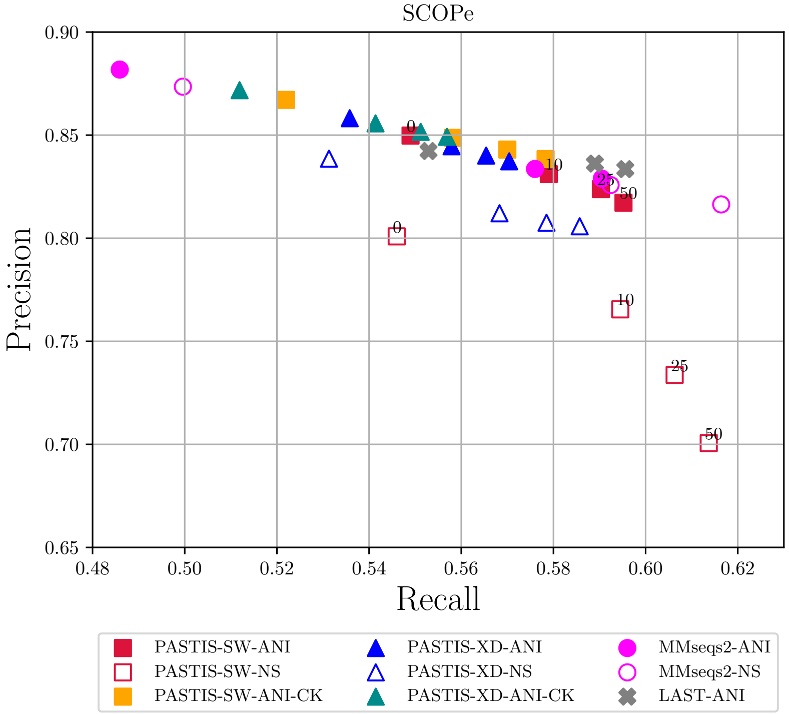
Substitute k-mers → C=ASA^T

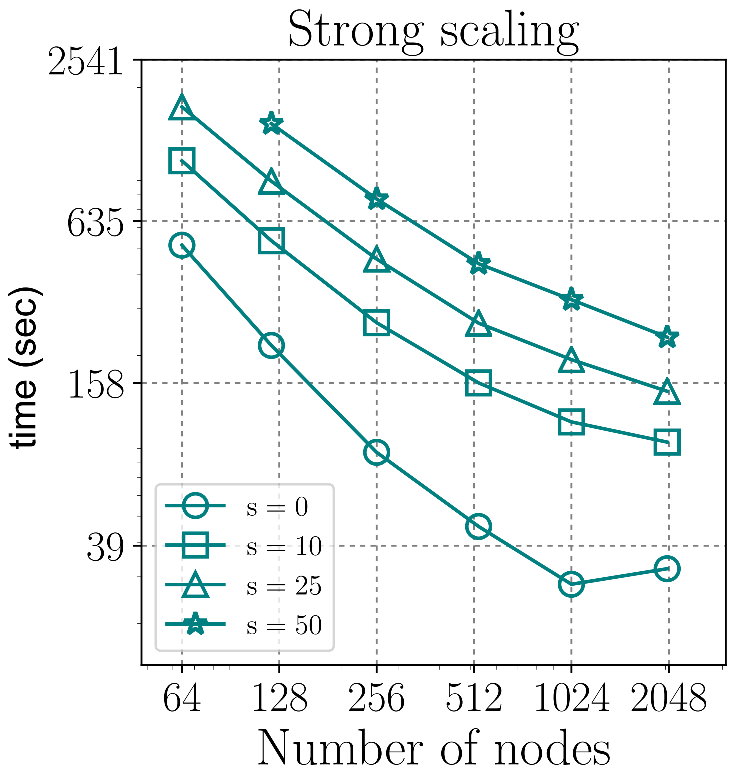
New semiring



Oguz Selvitopi, Saliya Ekanayake, Giulia Guidi, Georgios Pavlopoulos, Ariful Azad, and Aydın Buluç. Distributed Many-to-Many Protein Sequence Alignment Using Sparse Matrices. SC'20.

PASTIS performance and accuracy

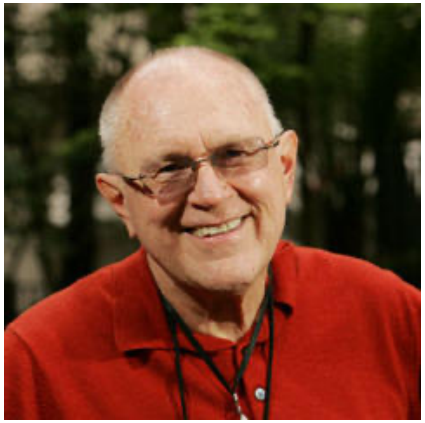


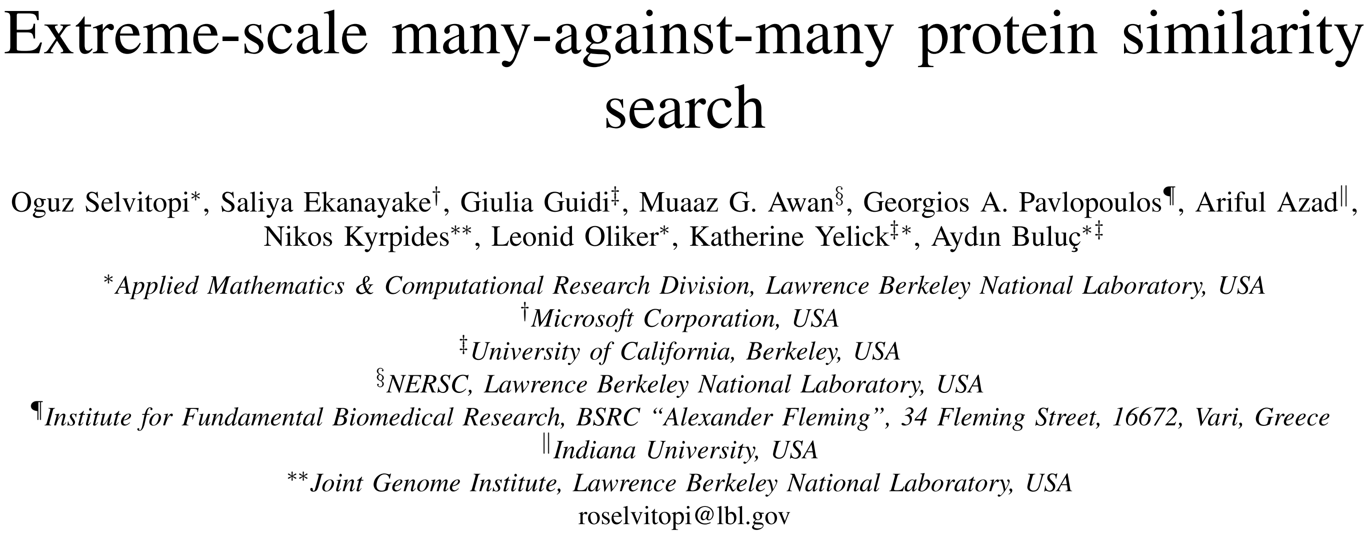


- *Protein similarity search* is the first and most time-consuming step in discovering protein families (proteins evolved from a common ancestor and who likely have the same function)
- *Protein family identification* is a key step in protein function discovery and taxonomic assignment of newly sequenced organisms

PASTIS in 2022

Hot off the press: Finalist for the 2022 ACM Gordon Bell Prize
https://en.wikipedia.org/wiki/Gordon_Bell_Prize





Extreme-scale many-against-many protein similarity search

Oguz Selvitopi[*], Saliya Ekanayake[†], Giulia Guidi[‡], Muaaz G. Awan[§], Georgios A. Pavlopoulos[¶], Ariful Azad[‖],
Nikos Kyrpides[**], Leonid Oliker[*], Katherine Yelick[‡*], Aydın Buluç[*‡]

[*]*Applied Mathematics & Computational Research Division, Lawrence Berkeley National Laboratory, USA*
[†]*Microsoft Corporation, USA*
[‡]*University of California, Berkeley, USA*
[§]*NERSC, Lawrence Berkeley National Laboratory, USA*
[¶]*Institute for Fundamental Biomedical Research, BSRC "Alexander Fleming", 34 Fleming Street, 16672, Vari, Greece*
[‖]*Indiana University, USA*
[**]*Joint Genome Institute, Lawrence Berkeley National Laboratory, USA*
roselvitopi@lbl.gov

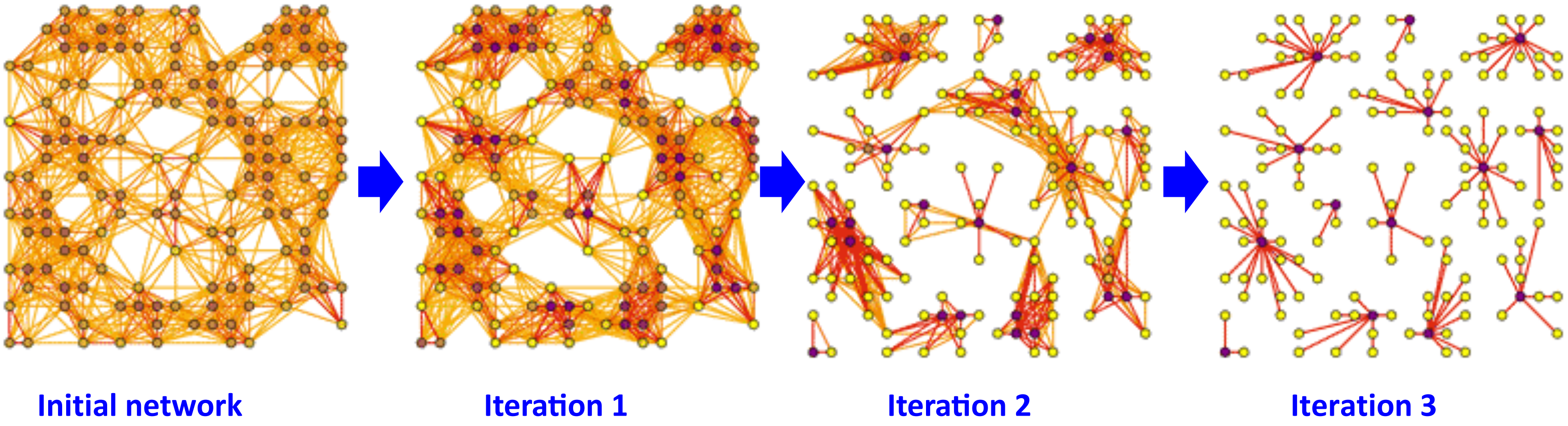
Abstract-- … We unleash the power of over 12,000 GPUs to perform all-vs-all protein similarity search on one of the largest publicly available datasets with 313 million proteins, in less than 4 hours, cutting the time-to-solution for many use cases from weeks. The variability of protein sequence lengths, as well as the sparsity of the space of pairwise comparisons, make this a challenging problem in distributed memory. …

SpGEMM use case #3: Markov Clustering

Markov clustering is also multi-source (in fact, all sources) traversal:
It alternates between SpGEMM and element-wise or column-wise pruning

GrB_mxm(C, GrB_NULL, <semiring>, A, A, <desc>)

A: sparse normalized adjacency matrix
C: denser (but still sparse) pre-pruned matrix for next iteration

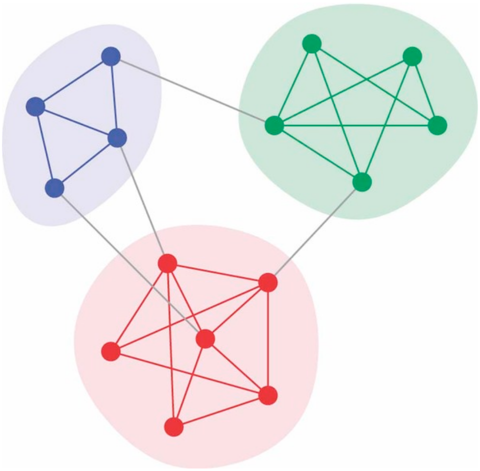


At each iteration:

Step 1 (Expansion): Squaring the matrix while pruning (a) small entries, (b) denser columns
Naïve implementation: sparse matrix-matrix product (SpGEMM), followed by column-wise top-K selection and column-wise pruning
Step 2 (Inflation) : taking powers entry-wise

The Markov Cluster Algorithm (MCL)

Widely popular and successful algorithm for discovering clusters (e.g. protein families) in protein interaction and protein sequence similarity networks



The number of **edges or higher-length paths** between two arbitrary nodes in a cluster is greater than the number of paths between nodes from different clusters

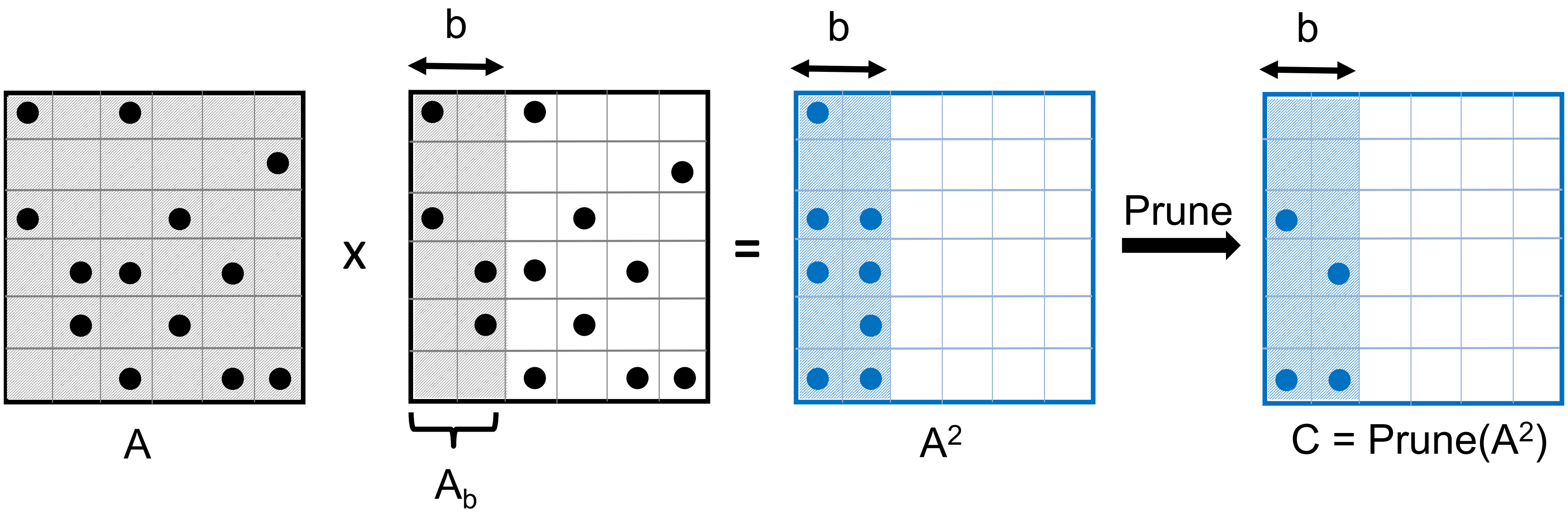
↓

Random walks on the graph will frequently remains within a cluster

↓

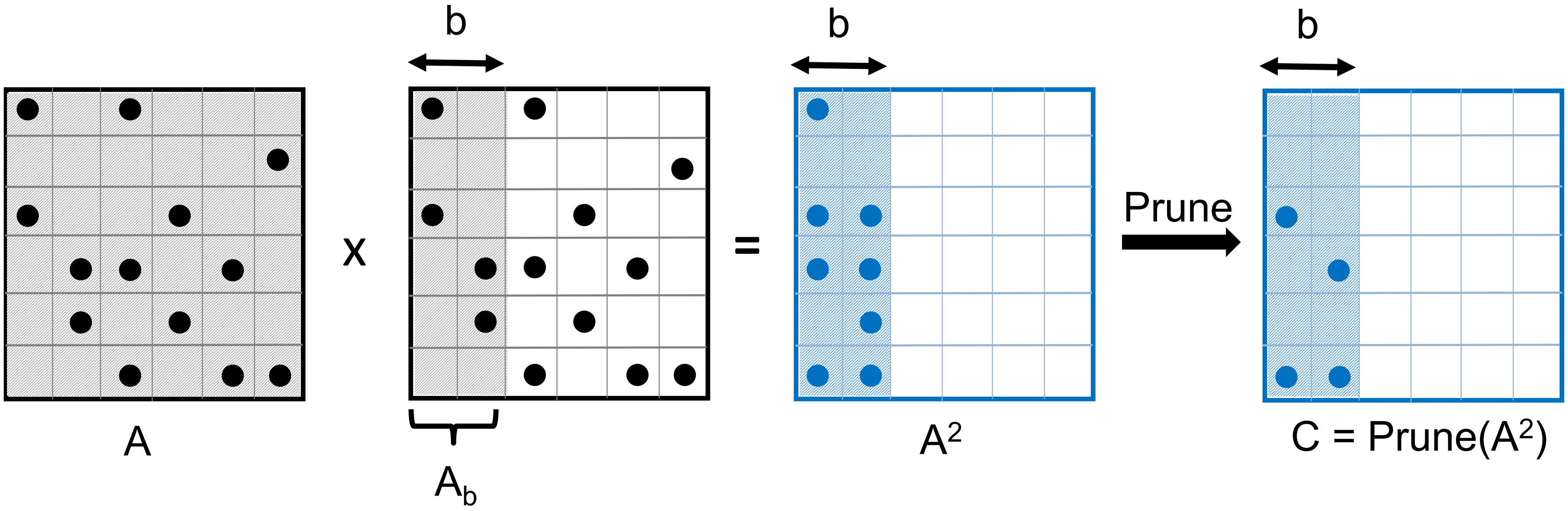
The algorithm **computes the probability** of random walks through the graph and **removes lower probability terms** to form clusters

A combined expansion and pruning step



- ❑ b: number of columns in the output constructed at once
  - – Smaller b: less parallelism, memory efficient (b=1 is equivalent to sparse matrix-sparse vector multiplication used in MCL)
  - – Larger b: more parallelism, memory intensive

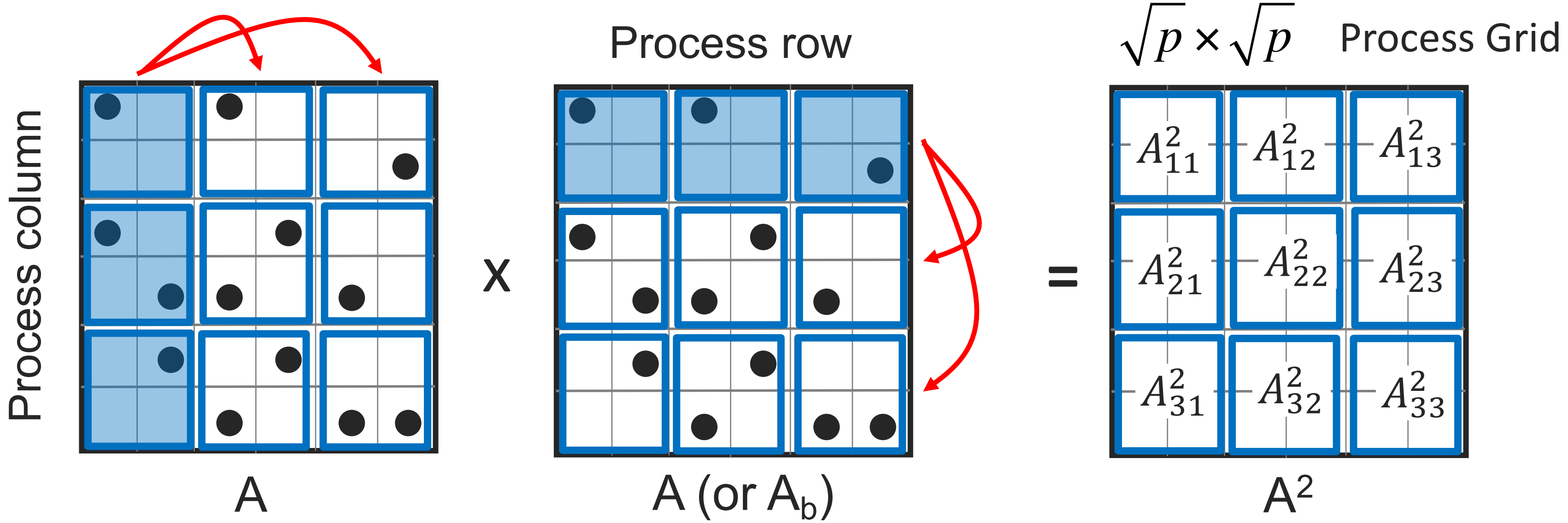
A combined expansion and pruning step



- b: number of columns in the output constructed at once
  - HipMCL selects b dynamically as permitted by the available memory
  - The algorithm works in h=N/b phases where N is the number of columns (vertices in the network) in the matrix

HipMCL: High-performance MCL

- HipMCL uses the most popular variant of Sparse SUMMA
- Both input matrices are broadcasted in stages and owners of output submatrices perform local sparse matrix multiplications
- When the number of phases increase (b decreases), A is re-broadcasted for each phase, increasing communication



A. Azad, G. Pavlopoulos, C. Ouzounis, N. Kyrpides, A. Buluç; HipMCL: a high-performance parallel implementation of the Markov clustering algorithm for large-scale networks, *Nucleic Acids Research, 2018*

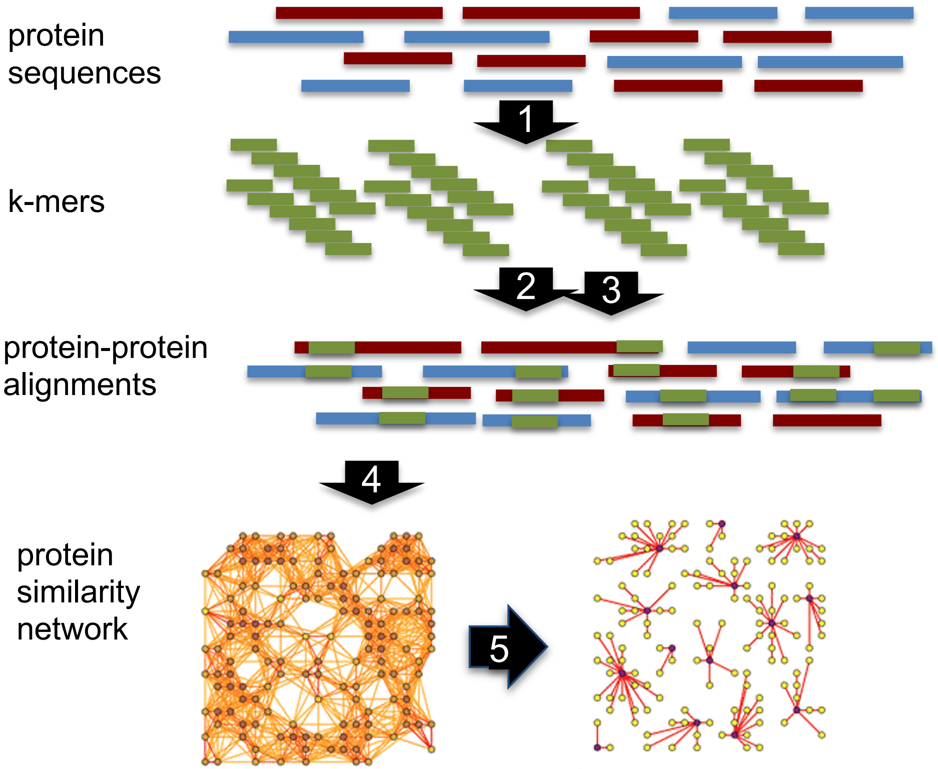
HipMCL on large networks

Data	Proteins	Edges	#Clusters	HipMCL time	platform
Isolate-1	47M	7 B	1.6M	1 hr	1024 nodes Edison
Isolate-2	69M	12 B	3.4M	1.66 hr	1024 nodes Edison
Isolate-3	70M	68 B	2.9M	2.41 hr	2048 nodes Cori KNL
MetaClust50	282M	37B	41.5M	3.23 hr	2048 nodes Cori KNL

MCL can not cluster these networks

Recap: Protein family identification using sparse matrices

PASTIS + HipMCL approach for protein family identification



1) K-mer Analysis
K-mer histogram

2) Sparse matrix building
A: proteins-by-kmers

3) Overlapping via SpGEMM C $= AA^T$ (or ASA^T)

4) Pairwise alignments
M = filter(C, alignment_score)

5) MCL iteration via SpGEMM
$M^{i+1} = \text{Prune}(M^i M^i)$

SpGEMM: Sparse matrix times sparse matrix

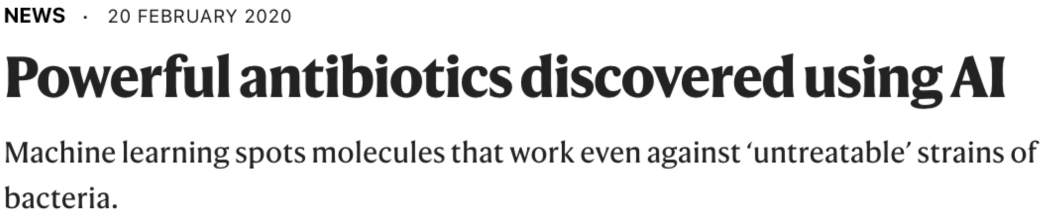
High-level outline

- Sparse matrices for graph algorithms
- Sparse matrices for computational biology
- **Sparse matrices for machine learning**
- Parallel algorithms for sparse matrix primitives
- Available software

Motivation for Graph Neural Networks

"GNNs are among the most general class of deep learning architectures currently in existence, […] and most other deep learning architectures can be understood as a special case of the GNN with additional geometric structure"

Bronstein, Michael M., et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges." (2021)

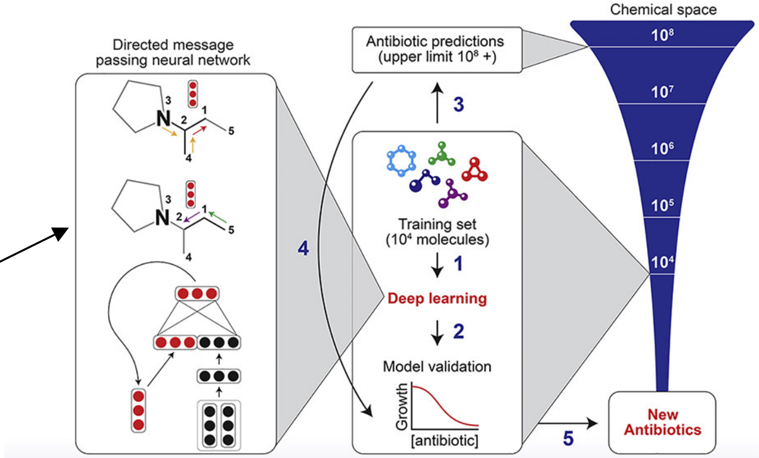


NEWS · 20 FEBRUARY 2020

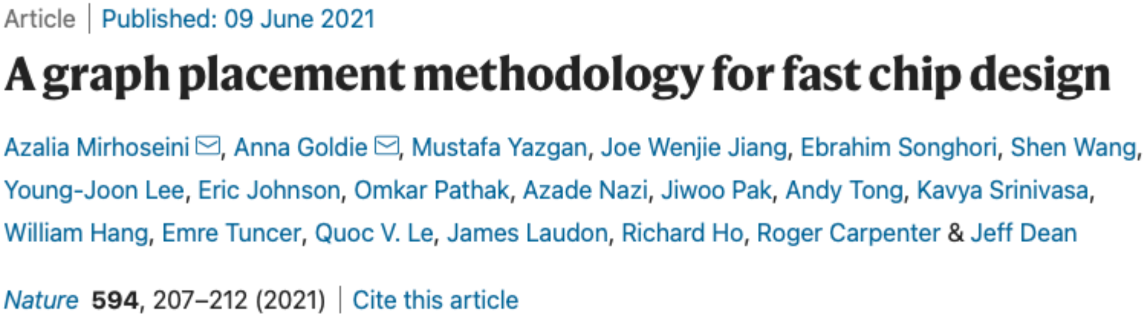
Powerful antibiotics discovered using AI

Machine learning spots molecules that work even against 'untreatable' strains of bacteria.

This is a graph neural network



Directed message passing neural network
Antibiotic predictions (upper limit 10^8 +)
Chemical space
10^8
10^7
10^6
10^5
10^4
3
4
Training set (10^4 molecules)
1
Deep learning
2
Model validation
Growth
[antibiotic]
5
New Antibiotics



Article | Published: 09 June 2021

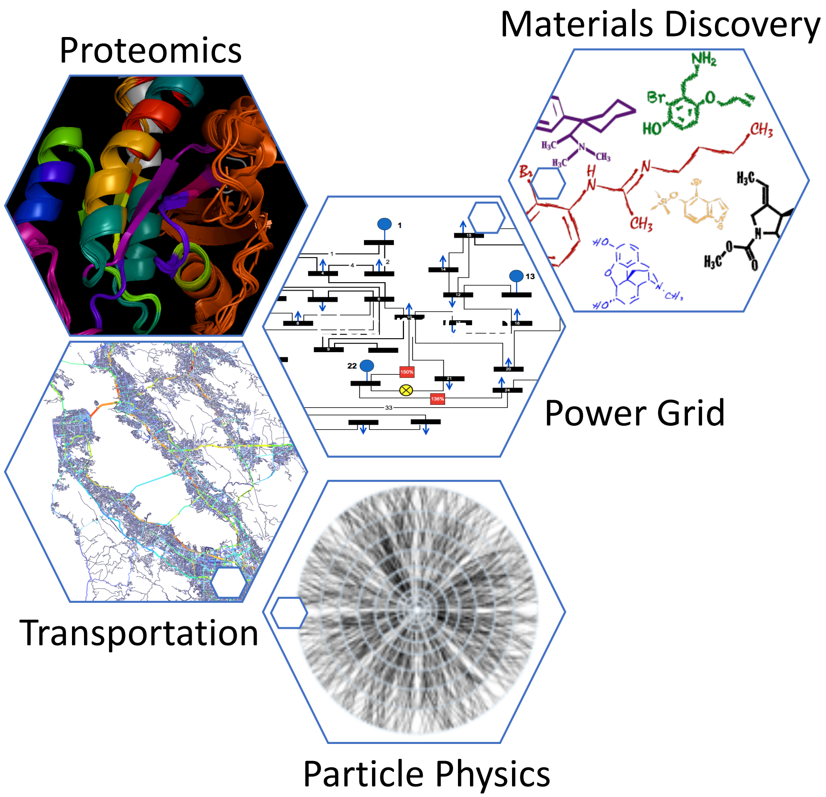
A graph placement methodology for fast chip design

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter & Jeff Dean

Nature **594**, 207–212 (2021) | Cite this article

… we pose chip floorplanning as a reinforcement learning problem, and develop an **edge-based graph convolutional neural network** architecture…

Graph Neural Networks (GNNs)



GNNs are finding success in many challenging scientific problems that involve interconnected data.

- Graph classification
- Edge classification
- **Node classification**

GNNs are computationally intensive to train. Distributed training need to scale to large GPU/node counts despite challenging sparsity.

What can I do with a GNN?

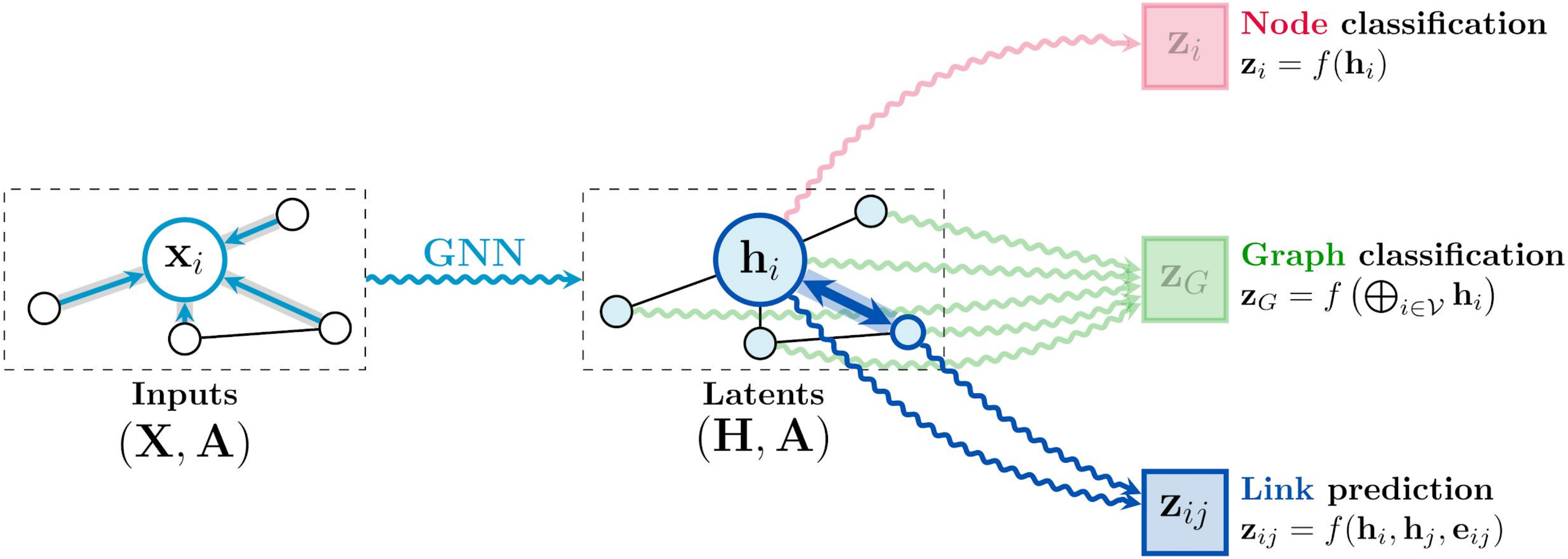
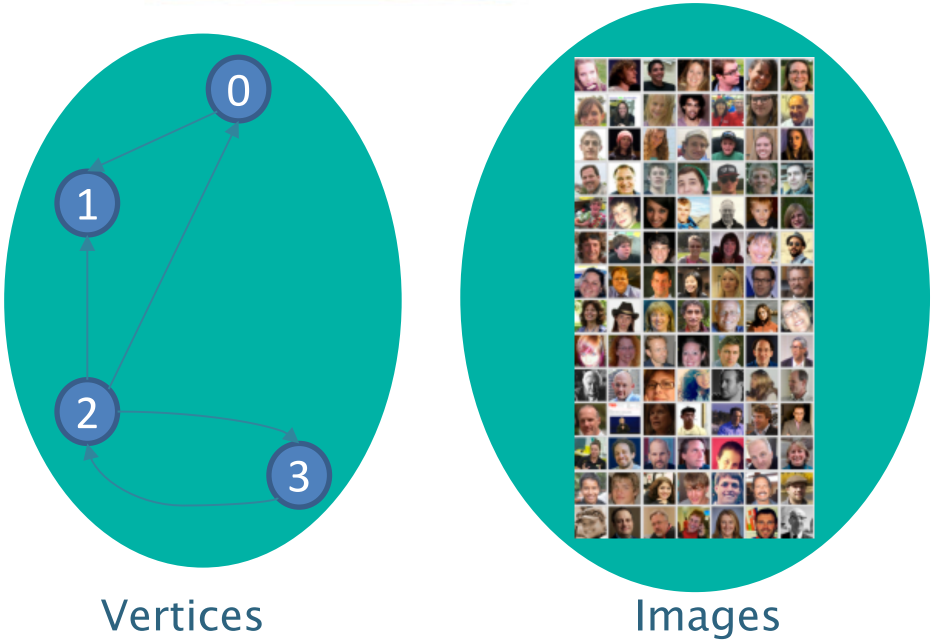

$\mathbf{x}_i$
GNN
$\mathbf{h}_i$
Inputs
$(\mathbf{X}, \mathbf{A})$
Latents
$(\mathbf{H}, \mathbf{A})$
$\mathbf{z}_i$
Node classification
$\mathbf{z}_i = f(\mathbf{h}_i)$
$\mathbf{z}_G$
Graph classification
$\mathbf{z}_G = f\left(\bigoplus_{i \in \mathcal{V}} \mathbf{h}_i\right)$
$\mathbf{z}_{ij}$
Link prediction
$\mathbf{z}_{ij} = f(\mathbf{h}_i, \mathbf{h}_j, \mathbf{e}_{ij})$

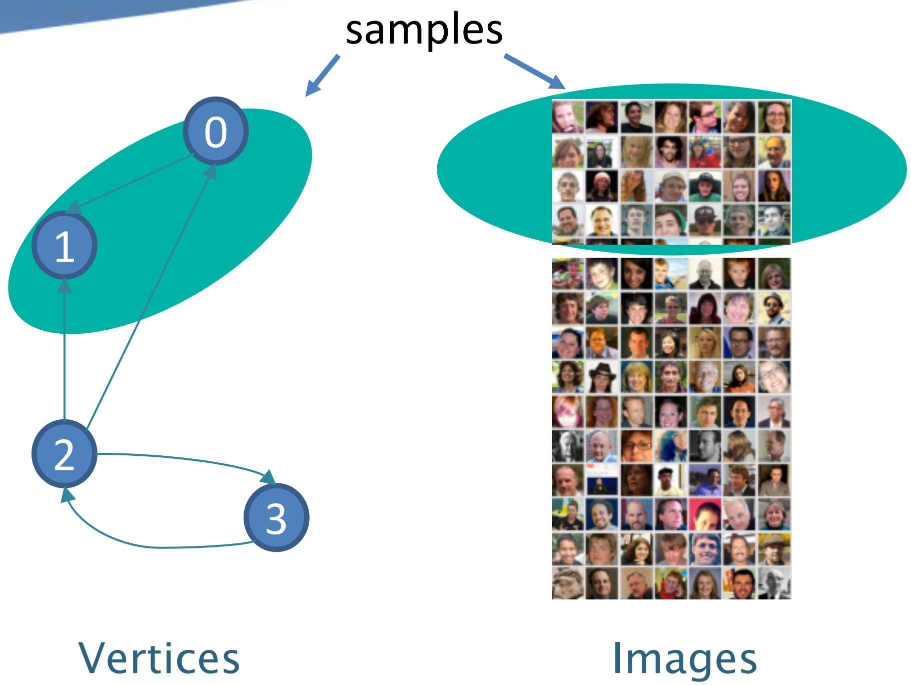
Figure source: Petar Veličković

Full-graph vs. mini-batch SGD



Full-graph training:

- Train on **entire** training set
- Slower convergence per epoch
- Faster training per epoch
- **Focus of this work**



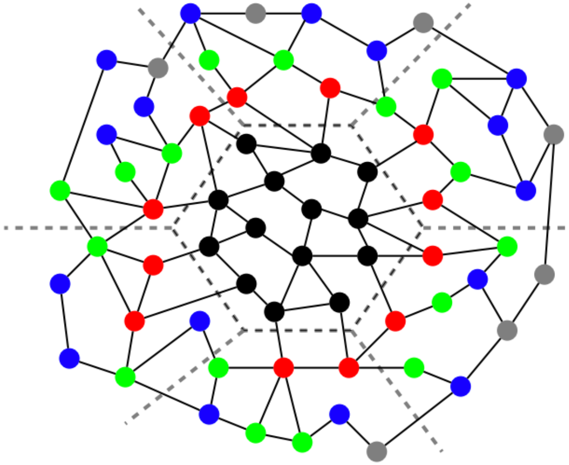
Mini-batch SGD:

- Train on multiple **samples** from training set
- Faster convergence per epoch
- Slower training per epoch
- Requires graph sampling, which effects accuracy and performance

Full-graph vs. mini-batch SGD

sample

No dependencies



Layered dependencies

- Vertices (unlike images) are dependent on each other
- L-layer GNN uses L-hop neighbors for vertices in batch
- Even for small L, must store ~whole graph for any minibatch for power-law graphs
- How to subsample from aggregated L-hop neighborhood and keep accuracy?
- CAGNET (Communication-Avoiding Graph Neural nETworks) full gradient descent to avoid such issues: https://github.com/PASSIONLab/CAGNET/

Graph convolution illustrated

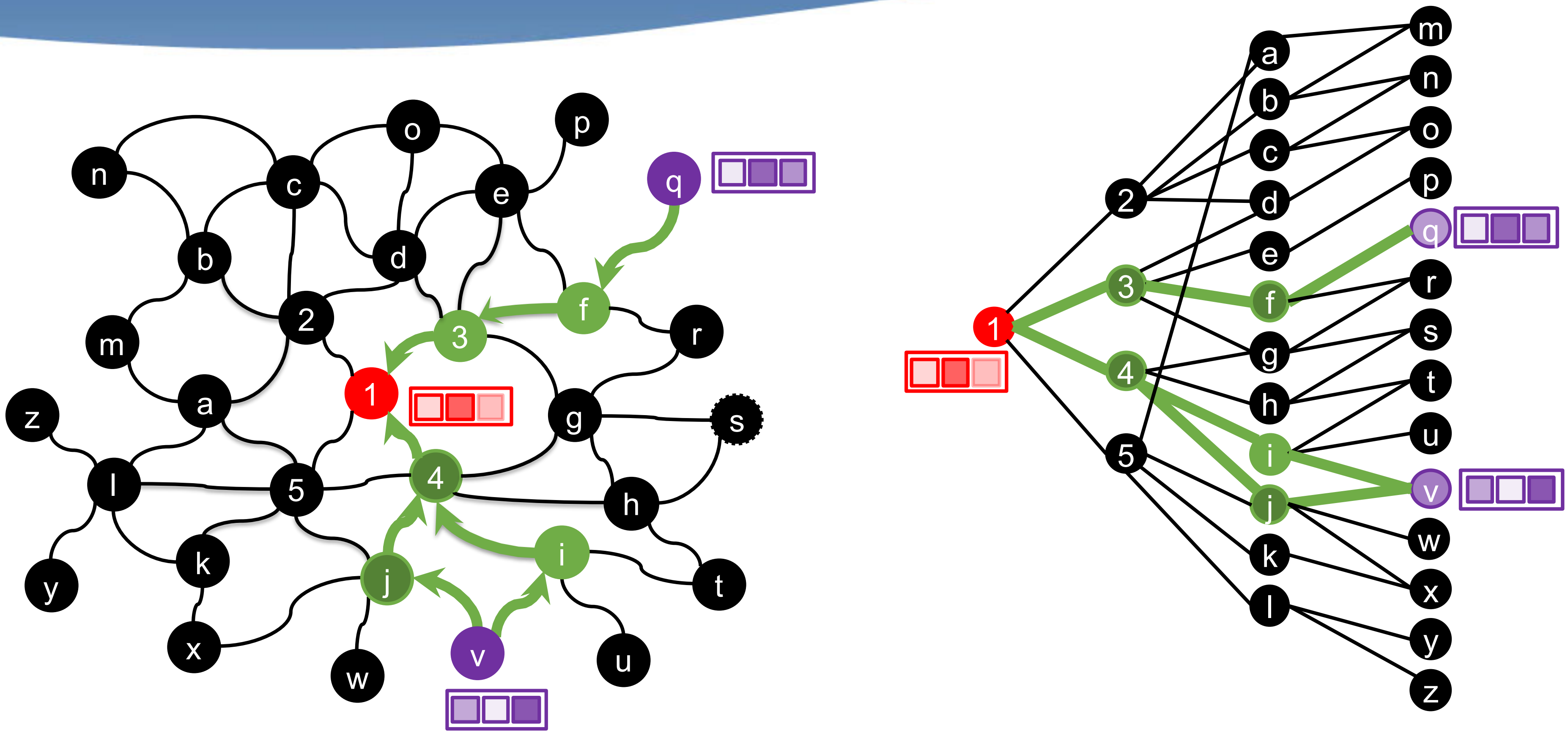
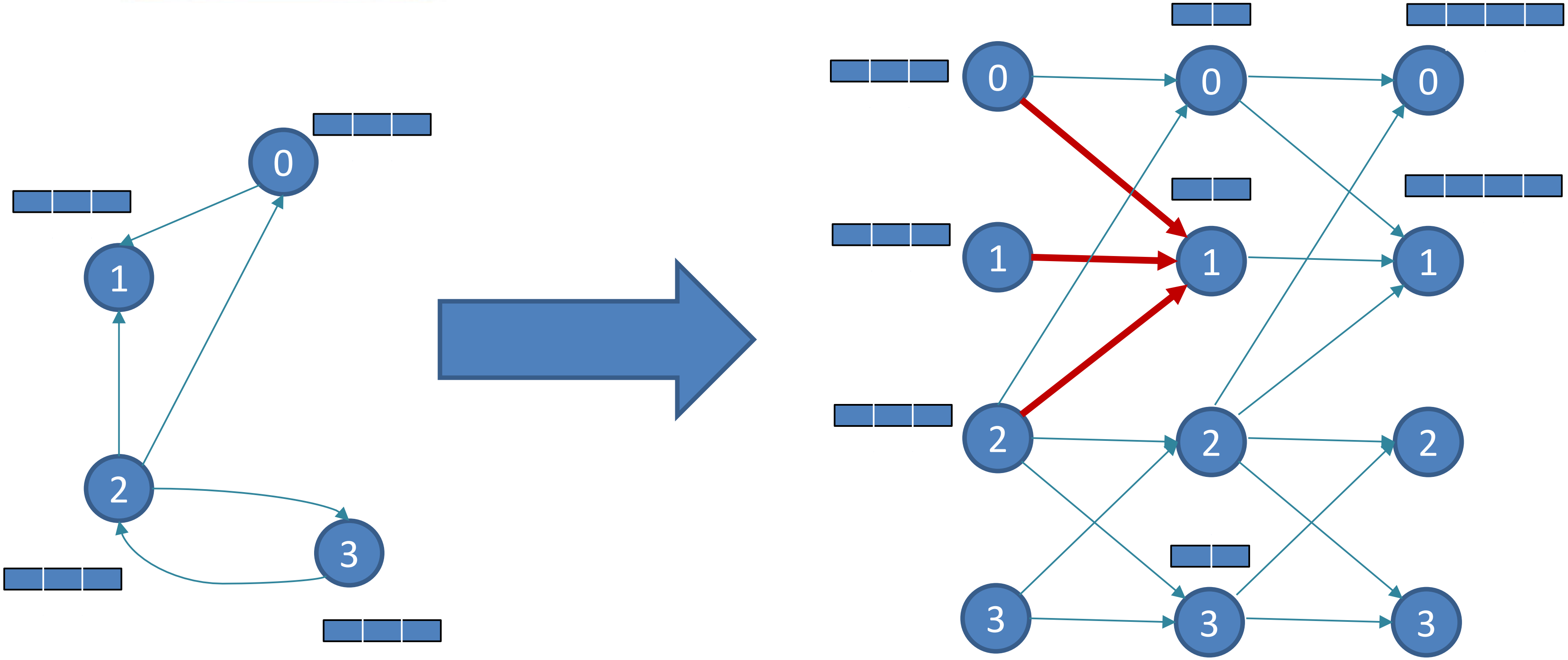


Illustration of the information flow in a Graph Neural Network (GNN). On the left is the graph in its natural form. The features (the shaded boxes) of vertices v and q are aggregated at vertex 1 through intermediate (green) vertices and edges. Features of other nodes are not shown but are also propagated. During training, the error is backpropagated in the opposite direction in the neural network, where each layer of the neural network propagates one hop of information.

Graph convolution illustrated

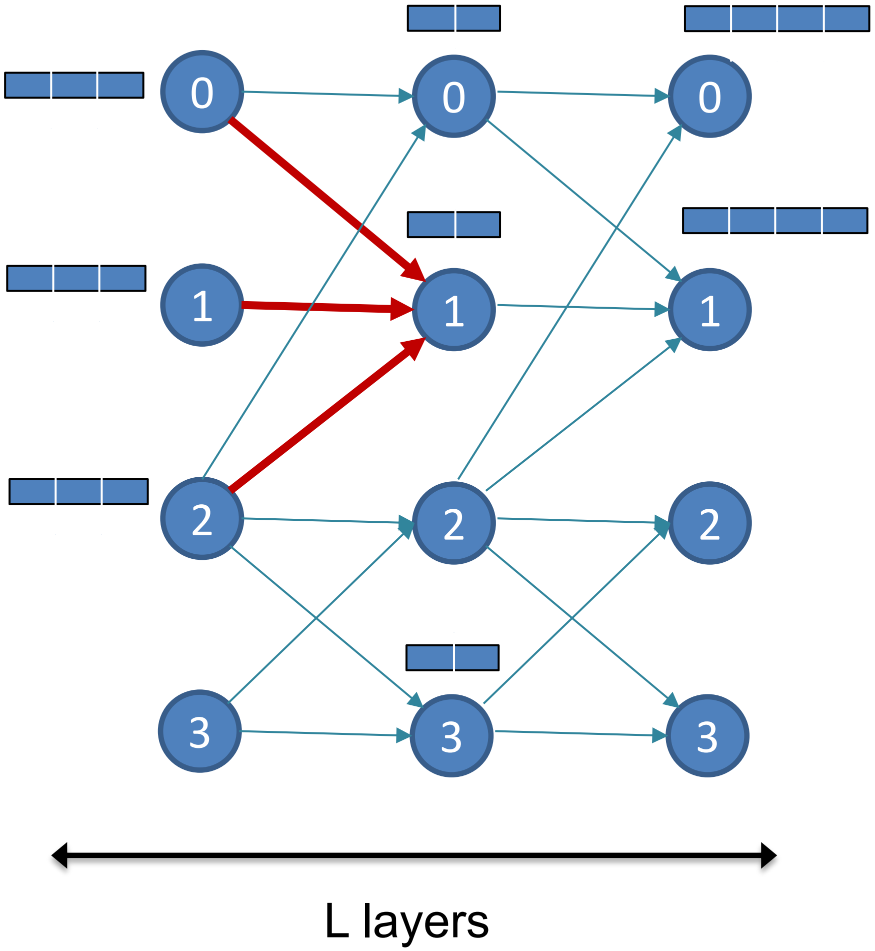


Input Graph

GNN of Input Graph

- Recall that a CNN can have different *channel* dimension at each layer.
- GNNs also have different embedding dimension at each layer

Memory cost of full-batch GCN training



Storage= $\sum_{i=1}^{L} nf^{i}$

$\approx O(nLf)$

Where $f = \frac{\sum_{i=1}^{L} f^{i}}{L}$

Say n = 100M, L = 4, f = 256, we are looking at 100B words, or 800GB

GNN Training

- Each node is initialized with a feature vector
  - H^0 has initial feature vector per node $(n\ x\ f)$
- Each node aggregates vectors of its neighbors, applies a weight
- Each layer computes gradients

```
for i = 1 … E
    for l = 1 … L
        $Z^l = A^T * H^{l-1} * W^l$
        $H^l = \sigma(Z^l)$
    ...
    for l = L-1 … 1
        $G^l = A * G^{l+1} * (W^{l+1})^T \odot \sigma'(Z^l)$
       $dH/dW = (H^{l-1})^T * A * G^l$
```

$A \in n\ x\ n$

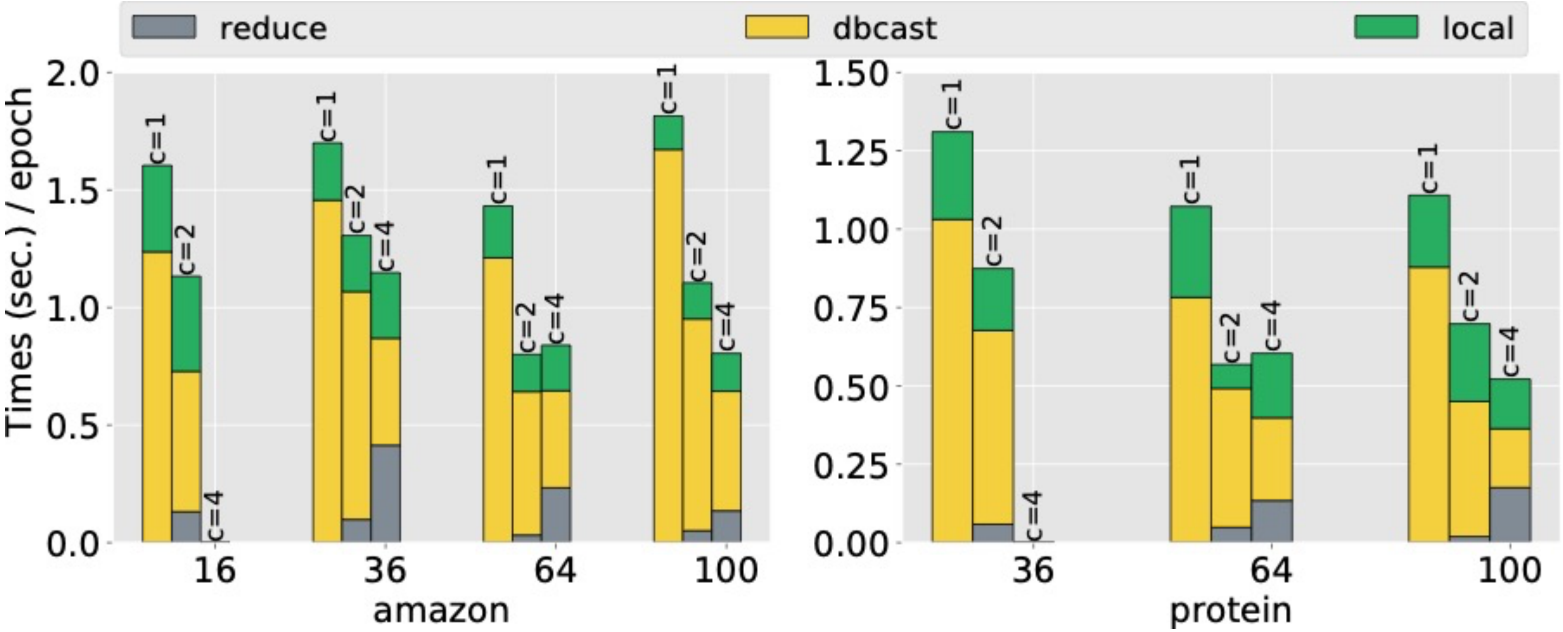
$H^l \in n\ x\ f^l$

$G^l \in n\ x\ f^l$

$W^l \in f^{l-1}\ x\ f^l$

- A is sparse and f << n, so the main workhorse is SpMM (sparse matrix times tall-skinny dense matrix)

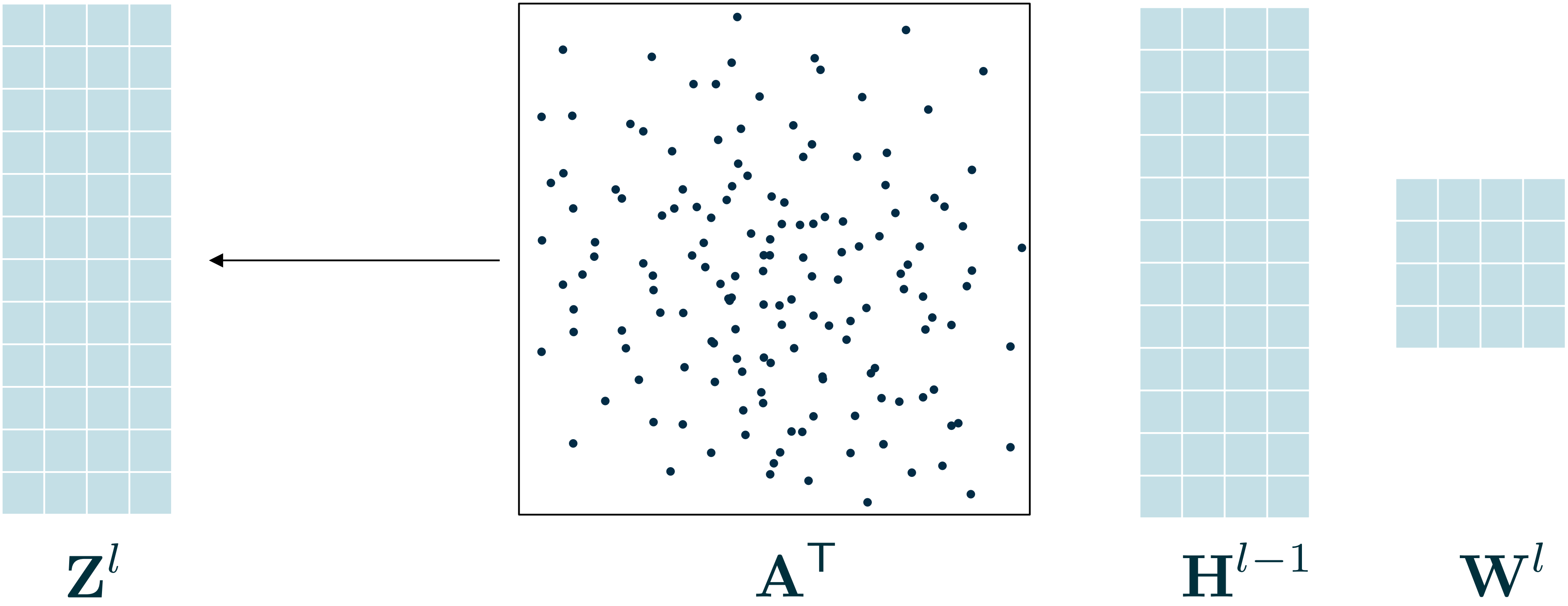
Communication avoidance (CA) In GNN Training



- Scales with both P (GPUs – x axis) and c (replication layers in CA algorithms)
- This is 1 GPU/node on Summit (all GPUs per node results in paper)
- Expect to scale with all GPUs / node with future architectures (e.g. Perlmutter)
- More results (2D and 3D algorithm) and 6 GPUs/node in the paper

Alok Tripathy, Katherine Yelick, Aydın Buluç. Reducing Communication in Graph Neural Network Training. SC'20

Bottleneck of full-graph GCN training



Cost(SpMM) >>> Cost(DGEMM)

(mostly because W is so small)

Pattern 3: Sparse matrix times tall-skinny dense matrix (SpMM)

Feature aggregation from neighbors:

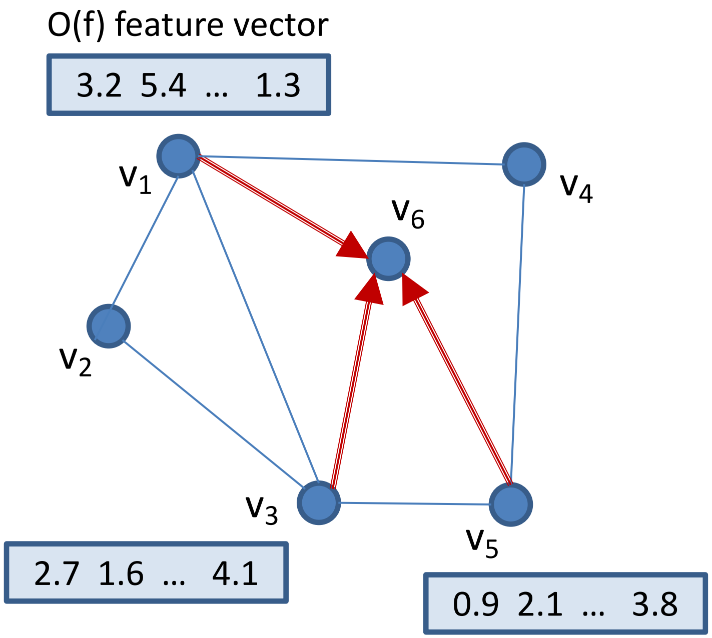
Used in Graph neural networks, graph embedding, etc.

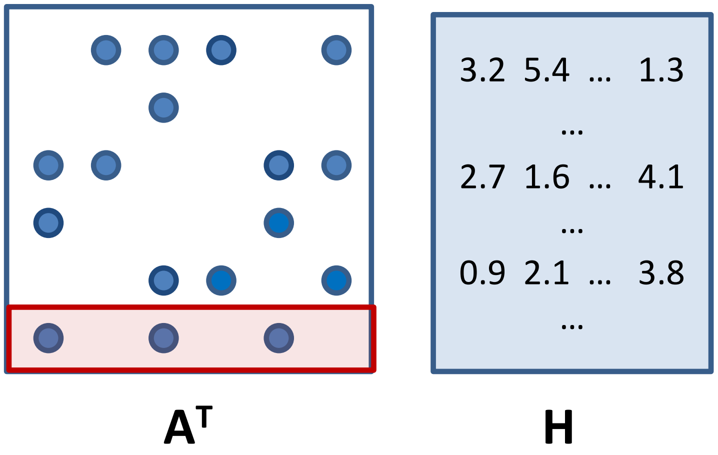
GrB_mxm(W, GrB_NULL, <semiring>, A, H, <desc>)

A: sparse adjacency matrix, n-by-n

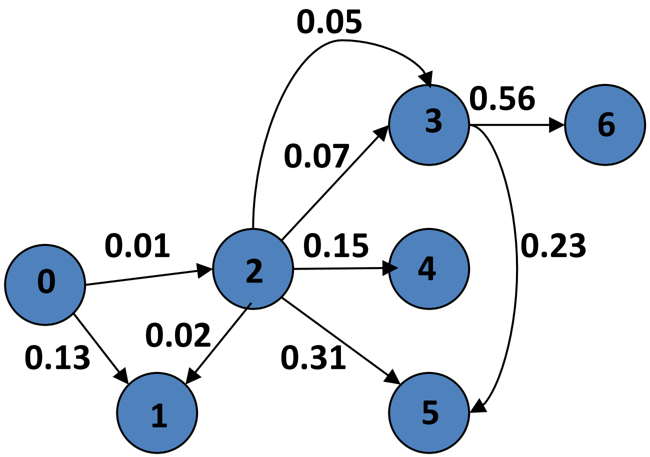
H: input dense matrix, n-by-f where f << n is the feature dimension

W: output dense matrix, new features





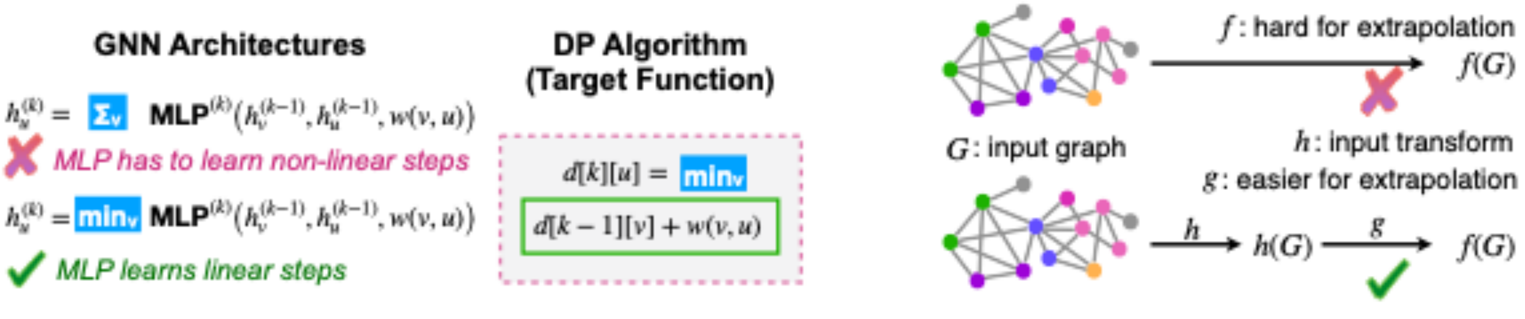
Expressiveness of GNNs: semirings for algorithmic alignment



- Consider shortest paths on a graph, a problem with lots of applications
- Most algorithms are based on applying the following relaxation in an intelligent way

function Relax (e(u,v))

d(v) = min { d(v), d(u) + w(u, v) }


GNN Architectures
$h_u^{(k)} = \Sigma_v \ \mathrm{MLP}^{(k)}(h_v^{(k-1)}, h_u^{(k-1)}, w(v,u))$
✗ MLP has to learn non-linear steps
$h_u^{(k)} = \min_v \ \mathrm{MLP}^{(k)}(h_v^{(k-1)}, h_u^{(k-1)}, w(v,u))$
✓ MLP learns linear steps
DP Algorithm
(Target Function)
$d[k][u] = \min_v$
$d[k-1][v] + w(v,u)$
f: hard for extrapolation
$f(G)$
G: input graph
h: input transform
g: easier for extrapolation
h
$h(G)$
g
$f(G)$

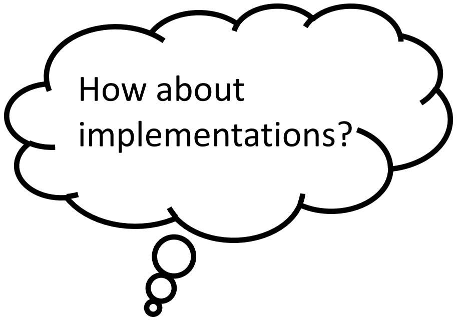
(a) Network architecture

(b) Input representation

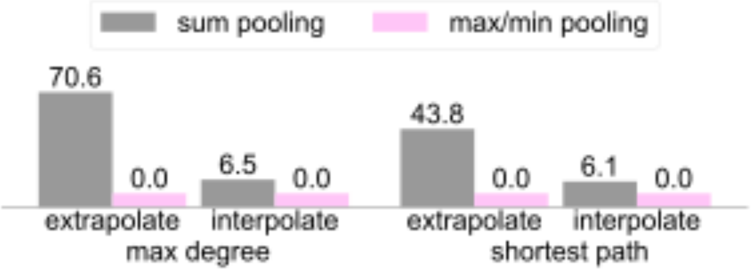
Xu K, et al. How neural networks extrapolate: From feedforward to graph neural networks. ICLR’21

Expressiveness of GNNs: semirings for algorithmic alignment

- Virtually all sparse matrix codes only support floating-point arithmetic
- Hence almost all GRL libraries only support neighborhood aggregation and pooling operations that can be represented as floating-point arithmetic



How about implementations?



Bars show mean average percentage error (MAPE), figure from Xu et al. (2021)

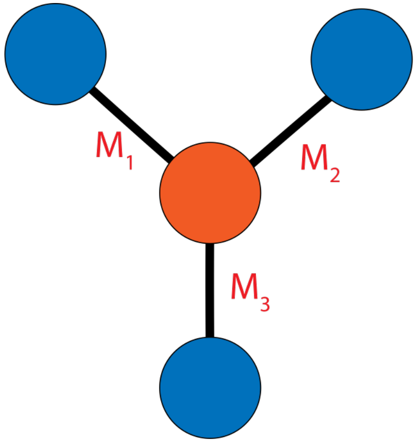
GraphBLAS *API* naturally supports any user-defined function:

GraphBLAS semiring: S = ⟨D1, D2, D3, ⊕, ⊗, 0[, 1]⟩ is defined by three domains D1, D2 and D3, an additive operation ⊕ : D3 × D3 → D3, a multiplicative operation ⊗ : D1×D2 →D3, an element 0∈D3

NEW! NVIDIA's cuSPARSE now has custom operators for SpMM: cusparseSpMMOp()

[More Sparse] Kernels in Machine Learning

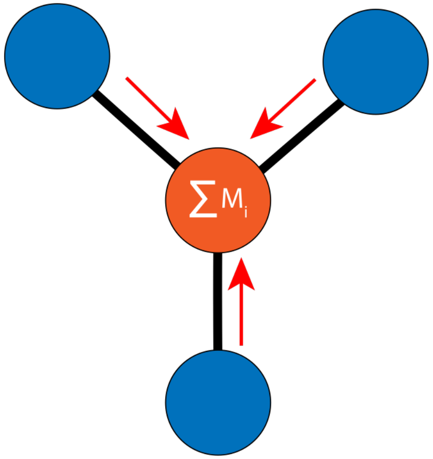
- Sampled Dense-Dense Matrix Multiplication (SDDMM) and Sparse-times-Dense Matrix Multiplication (SpMM) appear in a variety of applications:
  - Graph Neural Networks with Self-Attention
  - Collaborative Filtering with Alternating Least Squares
  - Document Clustering by Wordmover’s Distance



Message Generation

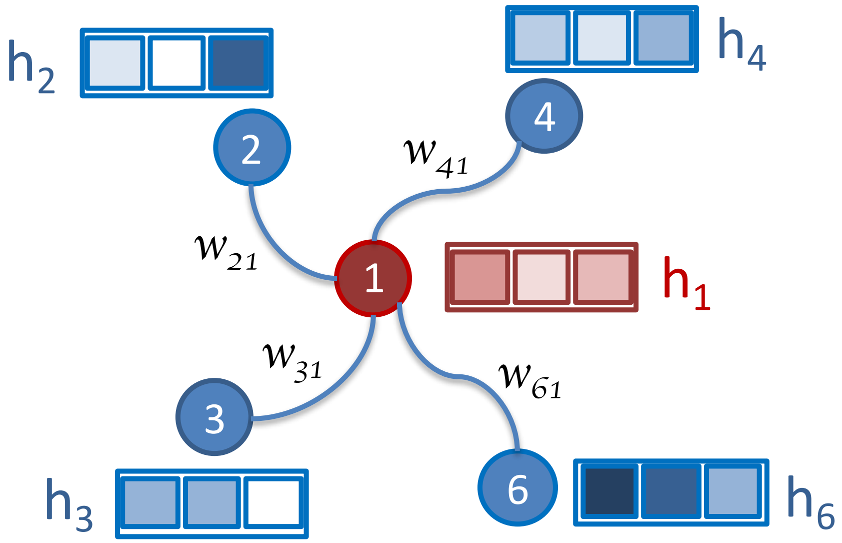
- Both kernels involve a single sparse matrix and two (typically tall-skinny) dense matrices. Typically, applications use both operations in sequence.

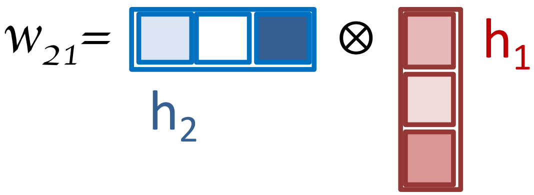
- When the sparse matrix is the adjacency matrix of a graph, we interpret the kernels as follows:
  - SDDMM generates a message on each edge
  - SpMM aggregates messages from incident edges



Message Aggregation

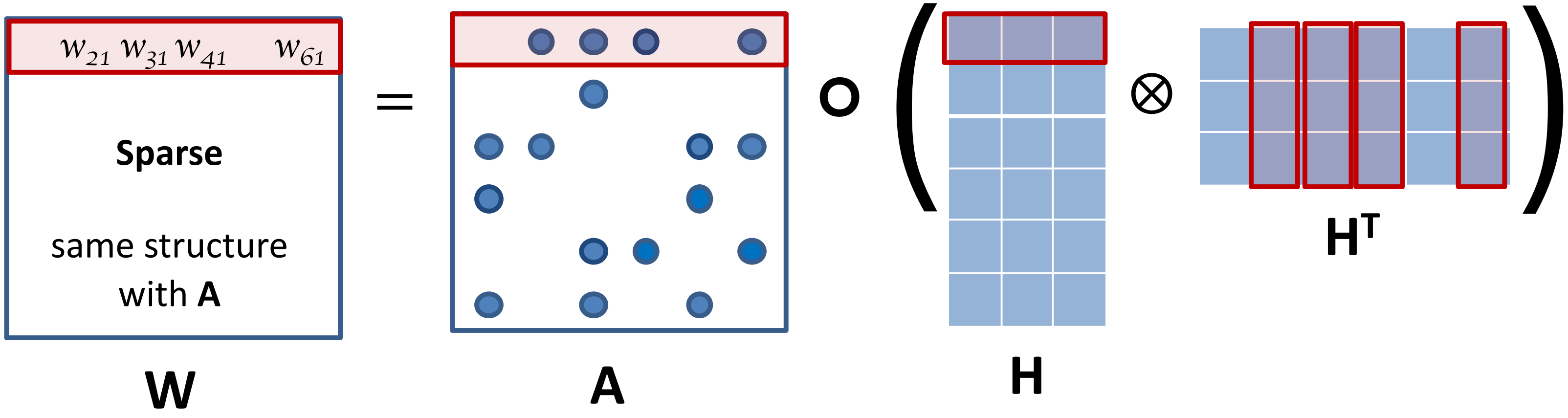
Graph attention: making edge weights learnable





SDDMM: Sampled dense-dense matrix multiplication

GrB_mxm(W, A, H, H, …);



Sparse

same structure with **A**

SpMM and SDDMM algorithmic duality

SDDMM and SpMM have **identical data access patterns**.
Consider serial algorithms for both kernels:

$\mathrm{R} \coloneqq \mathrm{SDDMM}(S, A, B)$

```
for $(i, j) \in S$
    $R_{ij} \coloneqq S_{ij}(A_{i:} \cdot B_{j:}^T)$
```

$\mathrm{A} \coloneqq \mathrm{SpMMA}(S, B)$

```
for $(i, j) \in S$
    $A_{i:} \mathrel{+}= S_{ij} B_{j:}$
```

Every nonzero (i, j) requires an interaction between row i of A and row j of B.

As a result:

Every distributed algorithm for SpMM can be converted to an algorithm for SDDMM with identical communication characteristics, and vice-versa.

V. Bharadwaj, A. Buluc, J. Demmel, "Distributed Memory Sparse Kernels for Machine Learning," 2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2022

Creating a parallel SDDMM algorithm from an SpMM algorithm

Consider any distributed algorithm for SpMMA that performs no replication. For all indices $k \in [1, r]$, the algorithm must (at some point)

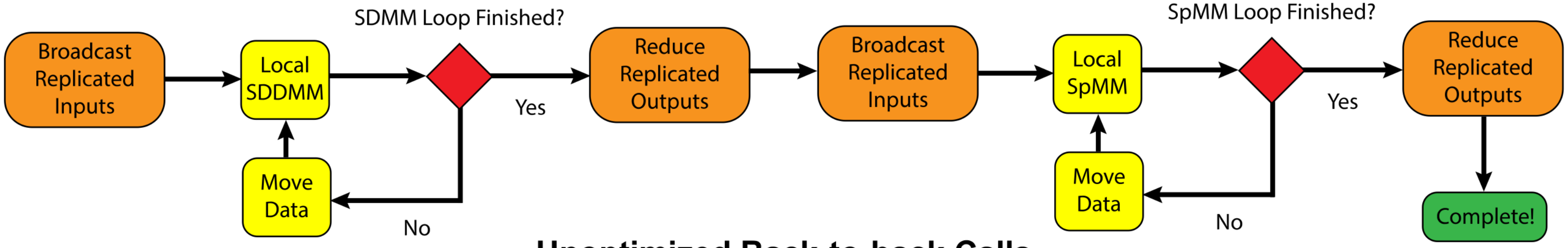
- Co-locate S_{ij}, A_{ik}, B_{jk} on a single processor
- Perform the update $A_{ik} \mathrel{+}= S_{ij} B_{jk}$

Transform this algorithm as follows:

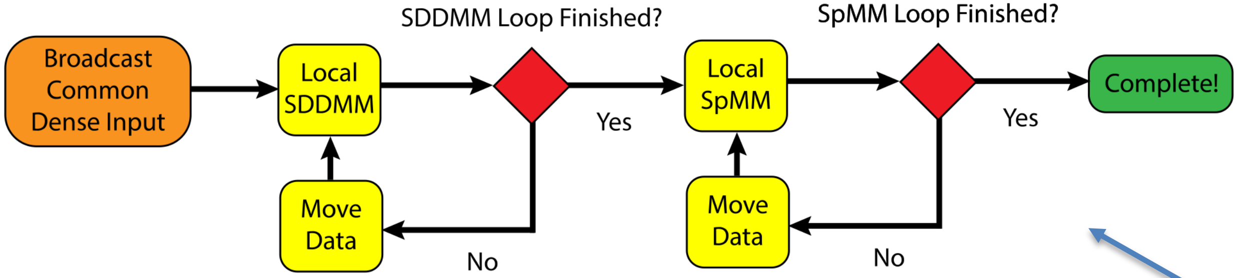
1. Change the input sparse matrix S to an output that is initialized to 0.
2. Change A from an output to an input.
3. Have each processor execute the local update: $S_{ij} \mathrel{+}= A_{ik} B_{jk}$

The resulting algorithm performs SDDMM (up to multiplication with the values initially in S) with communication characteristics and data layout identical to the original.

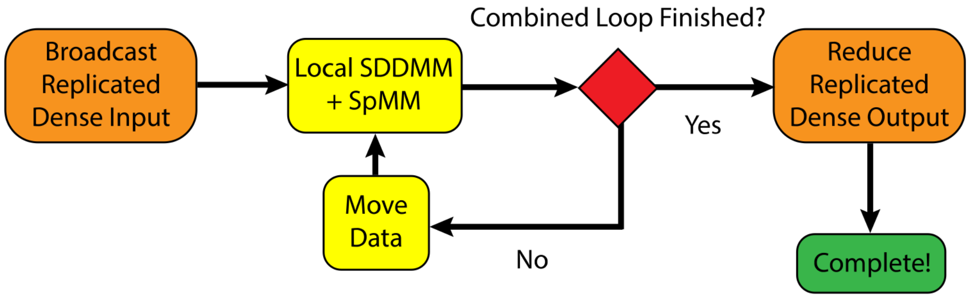
Communication Eliding Strategies for FusedMM: SDDMM+SpMM


Broadcast Replicated Inputs
Local SDDMM
SDMM Loop Finished?
Yes
No
Move Data
Reduce Replicated Outputs
Broadcast Replicated Inputs
Local SpMM
SpMM Loop Finished?
Yes
No
Move Data
Reduce Replicated Outputs
Complete!

Unoptimized Back-to-back Calls


Broadcast Common Dense Input
Local SDDMM
SDDMM Loop Finished?
Yes
No
Move Data
Local SpMM
SpMM Loop Finished?
Yes
No
Move Data
Complete!

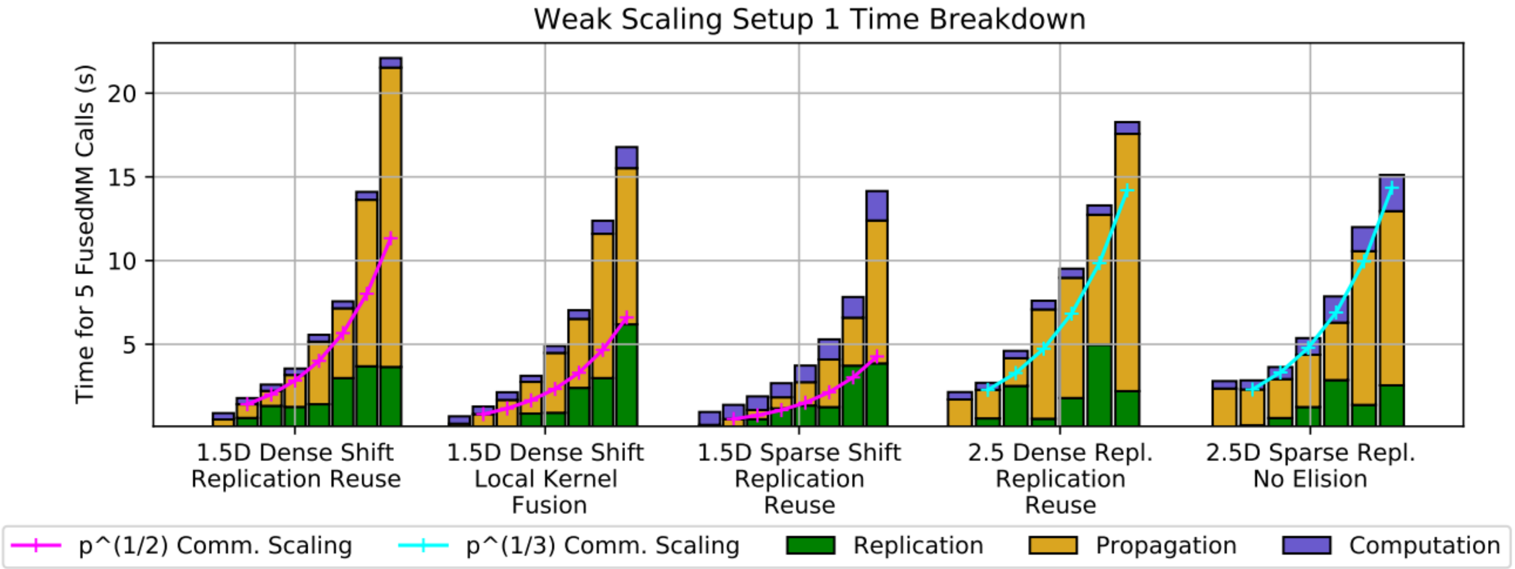
Replication Reuse


Broadcast Replicated Dense Input
Local SDDMM + SpMM
Combined Loop Finished?
Yes
No
Move Data
Reduce Replicated Dense Output
Complete!

Local Kernel Fusion

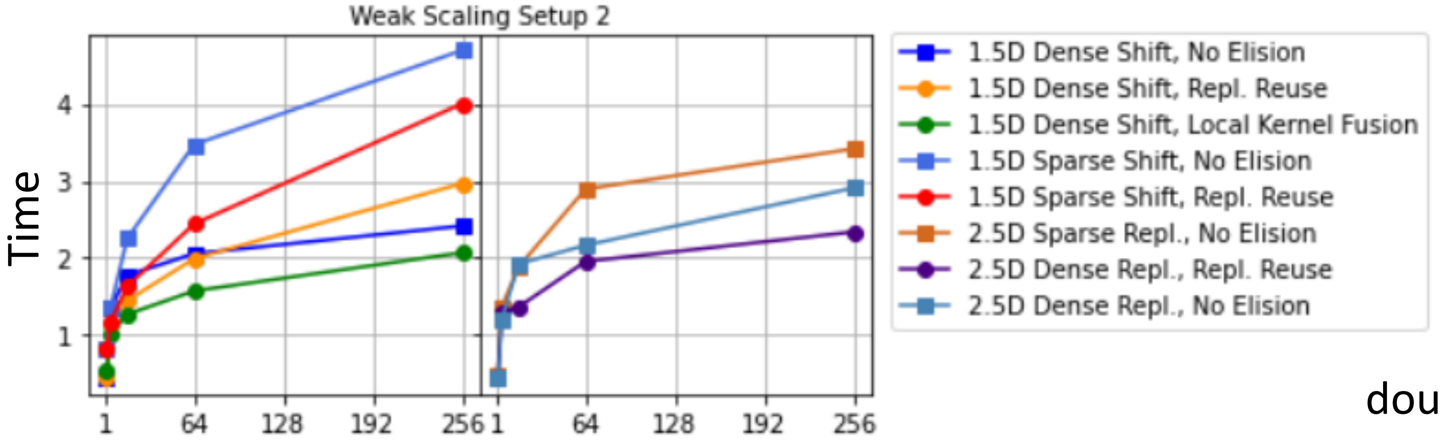
Mutually exclusive optimizations

Distributed FusedMM performance



$$\phi = \frac{\text{nnz}(S)}{nr}$$

remains constant



$$\phi = \frac{\text{nnz}(S)}{nr}$$

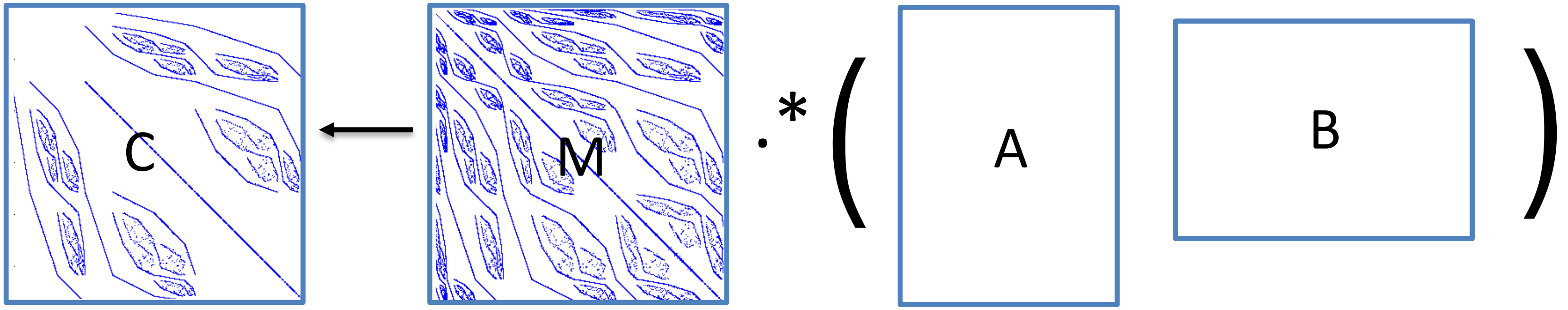
doubles at each process count quadrupling

High-level outline

- Sparse matrices for graph algorithms
- Sparse matrices for computational biology
- Sparse matrices for machine learning
- **Parallel algorithms for sparse matrix primitives**
- Available software

Sparse matrix-matrix multiplication

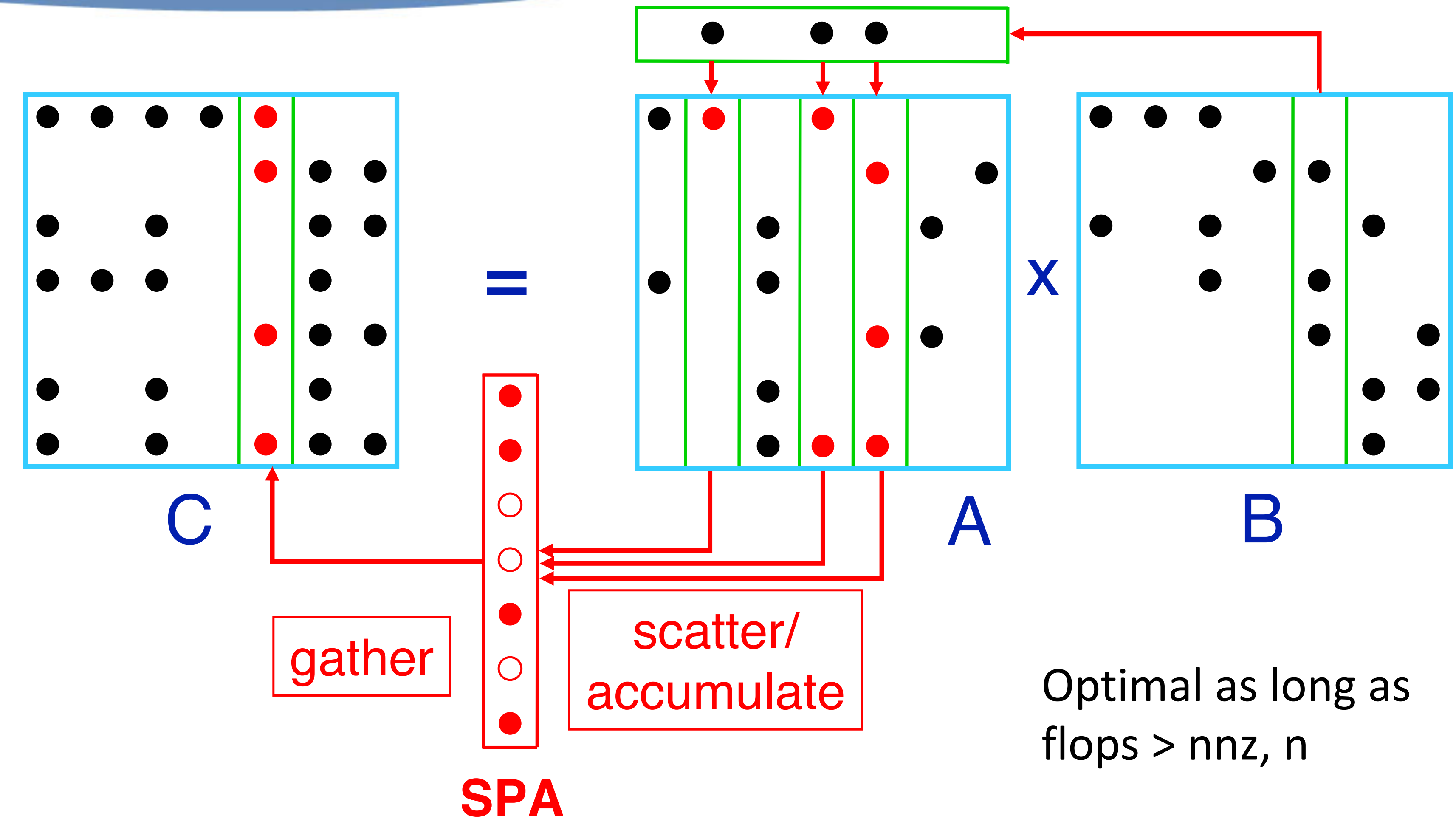
$$C(\neg M) \oplus= A^{T} \oplus.\otimes B^{T}$$



M: the output mask (also called a sampling matrix), **always sparse if present**
A, B: input matrices, at least one is sparse unless the mask is present
C: output matrix

SpGEMM: A, B are sparse, C can be sparse or dense (depending on shape)
Masked-SpGEMM: Same as SpGEMM, with mask (M) present
SpMM: A sparse, B and C dense (tall skinny), often no mask (M)
SDDMM: A, B are dense, M present, C sparse
SpMV: degenerate case of SpMM with B and C having 1 column
SpMSpV: degenerate case of SpGEMM with B, C, (possibly M) having 1 column

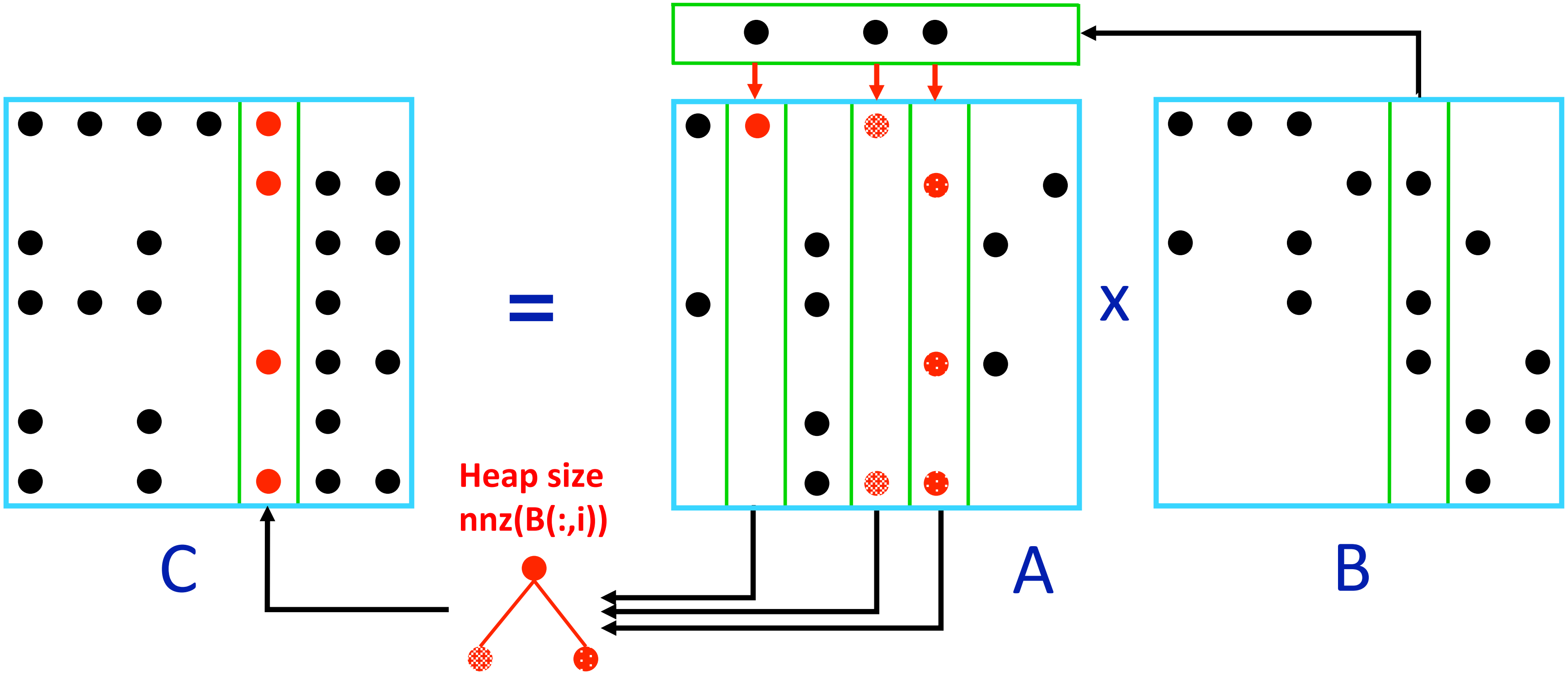
Basic serial SpGEMM (Gustavson, 1978)



Optimal as long as flops > nnz, n

- Implemented in Matlab & other popular software
- Not directly applicable to multithreading: SPA falls out of cache and takes up too much space in aggregate

More parallelizable SpGEMM (Azad et al., 2016)



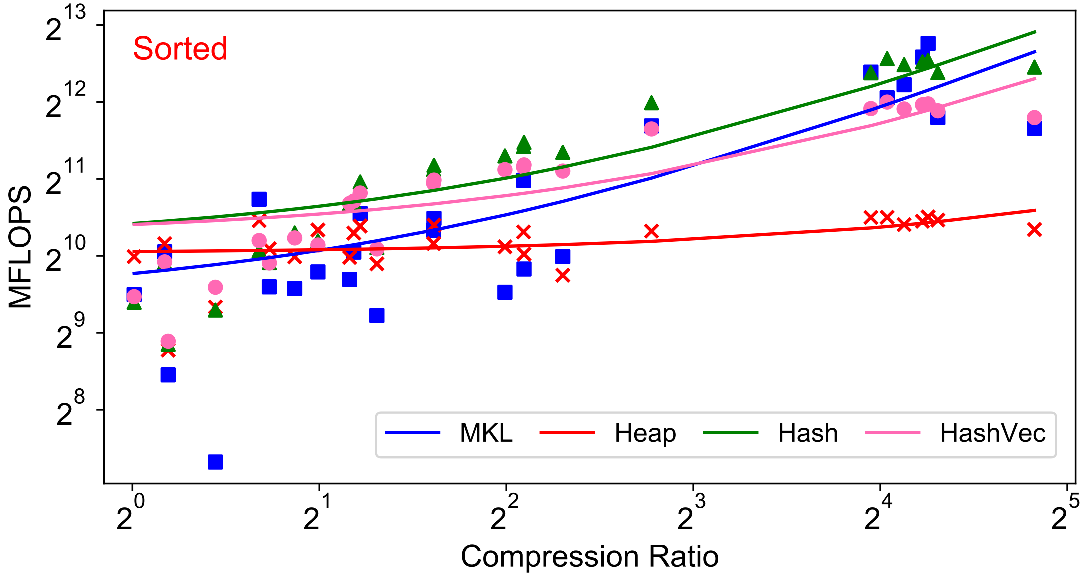
- Implemented in CombBLAS and SparseSuite:GraphBLAS
- Memory efficient and suitable for multithreading
- Not great for high compression ratio cases (more later)

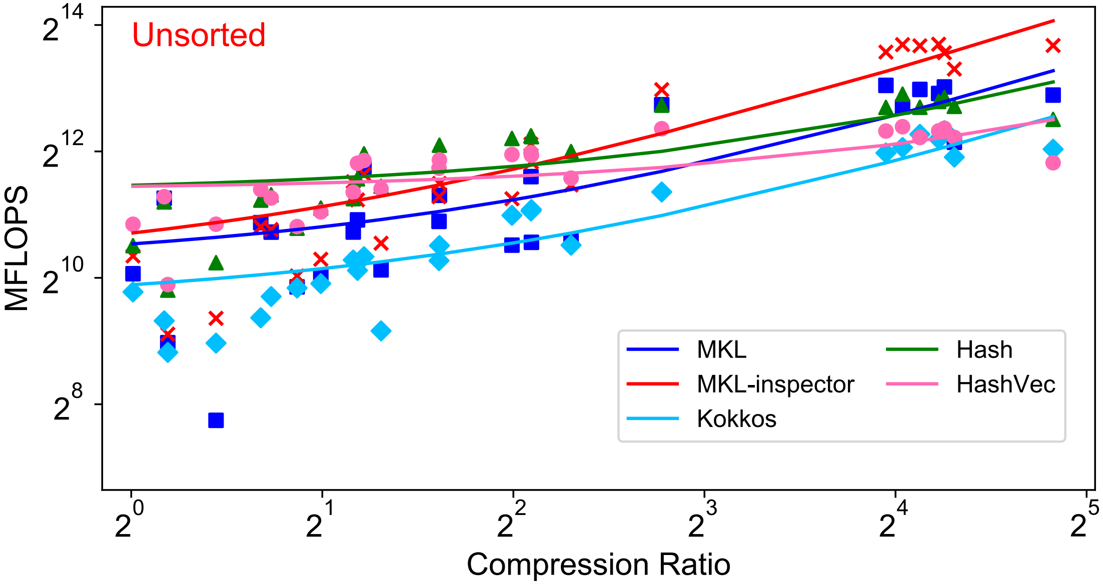
New shared-memory SpGEMM kernels

- Optimizing algorithms for Intel architectures
- Heap [Azad, 2016]
  - Priority queue indexed by column indices
  - **Requires logarithmic time to extract elements**
  - **Space efficient**: O(nnz(a_{i*}))
    - Better cache utilization
- Hash [Nagasaka, 2016]
  - Uses hash table for accumulator, based on GPU work
    - **Low memory usage and high performance**
  - Each thread once allocates the hash table and reuses it
  - **Extended to HashVector to exploit wide vector register**

Fast shared-memory SpGEMM kernels

- Compression ratio (CR): flops/nnz(C)
- Combinatorial BLAS and HipMCL used to use heap
- Stable performance but significant gap in high CR
- HipMCL inputs have high CR





- We integrated hash algorithms to CombBLAS and HipMCL

Yusuke Nagasaka, Satoshi Matsuoka, Ariful Azad, and Aydin Buluc. Performance optimization, modeling and analysis of sparse matrix-matrix products on multi-core and many-core processors. Parallel Computing, 90:102545, 2019.

SpGEMM on GPUs: many libraries

- **bhsparse** [1]
  - Hybrid method for result matrix pre-allocation
    - 3 strategies (heap-based,
  - Parallel insert operations via fast merging
  - Heuristic-based load balancing (bins)

- **rmerge2** [2]
  - Iterative row-merging
  - Aggregate duplicate column indices via warp shuffles (merge $W = 32$ rows)
  - Requires no shared memory but many registers
  - Grouping into cases for load balancing

- **nsparse** [3]
  - Linear probing shared-memory hash table
  - Row grouping based on number of nonzero elements or intermediate products (load balancing)
  - Warp shuffle and shared memory for accumulations
  - Concurrent kernel execution via streams

- Performance might differ depending on
  - Compression rate
  - Matrix structure
  - GPU microarchitecture

This list is circa 2018, today we have more codes such as **AC-SpGEMM**

We are doing great compared to 10+ years ago when the SpGEMM primitive wasn't popular.

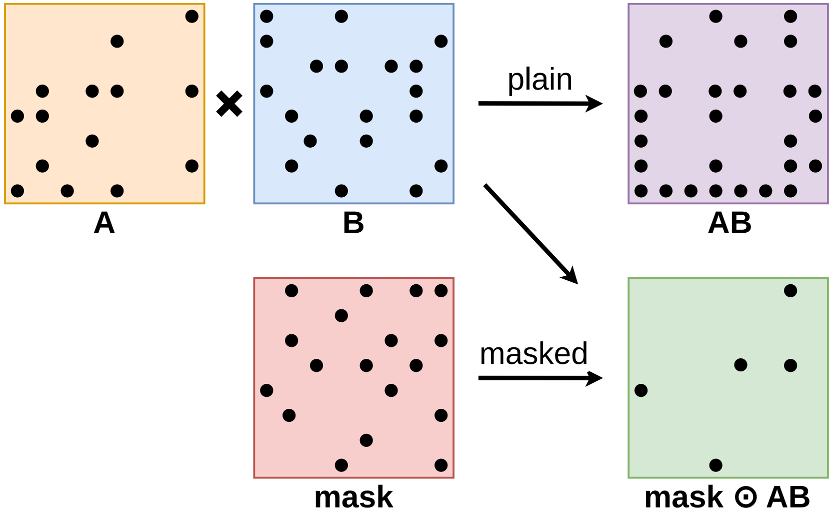
[1] Liu, Weifeng, and Brian Vinter. "An efficient GPU general sparse matrix-matrix multiplication for irregular data." In Parallel and Distributed Processing Symposium, 2014 IEEE 28th International, pp. 370-381. IEEE, 2014.

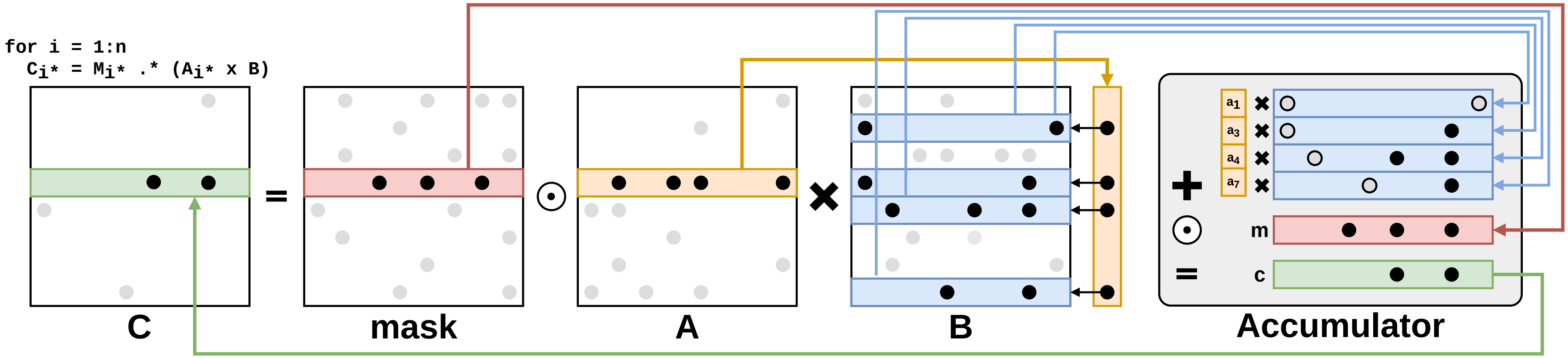
[2] Gremse, Felix, Kerstin Küpper, and Uwe Naumann. "Memory-Efficient Sparse Matrix-Matrix Multiplication by Row Merging on Many-Core Architectures." SIAM Journal on Scientific Computing 40, no. 4 (2018): C429-C449.

[3] Nagasaka, Yusuke, Akira Nukada, and Satoshi Matsuoka. "High-Performance and Memory-Saving Sparse General Matrix-Matrix Multiplication for NVIDIA Pascal GPU." In 2017 46th International Conference on Parallel Processing (ICPP), pp. 101-110. IEEE, 2017.

New algorithms for Masked SpGEMM

Main Idea: When certain output entries of SpGEMM are not needed (masked out), it is wasteful to materialize/compute the product first and then to mask out entries





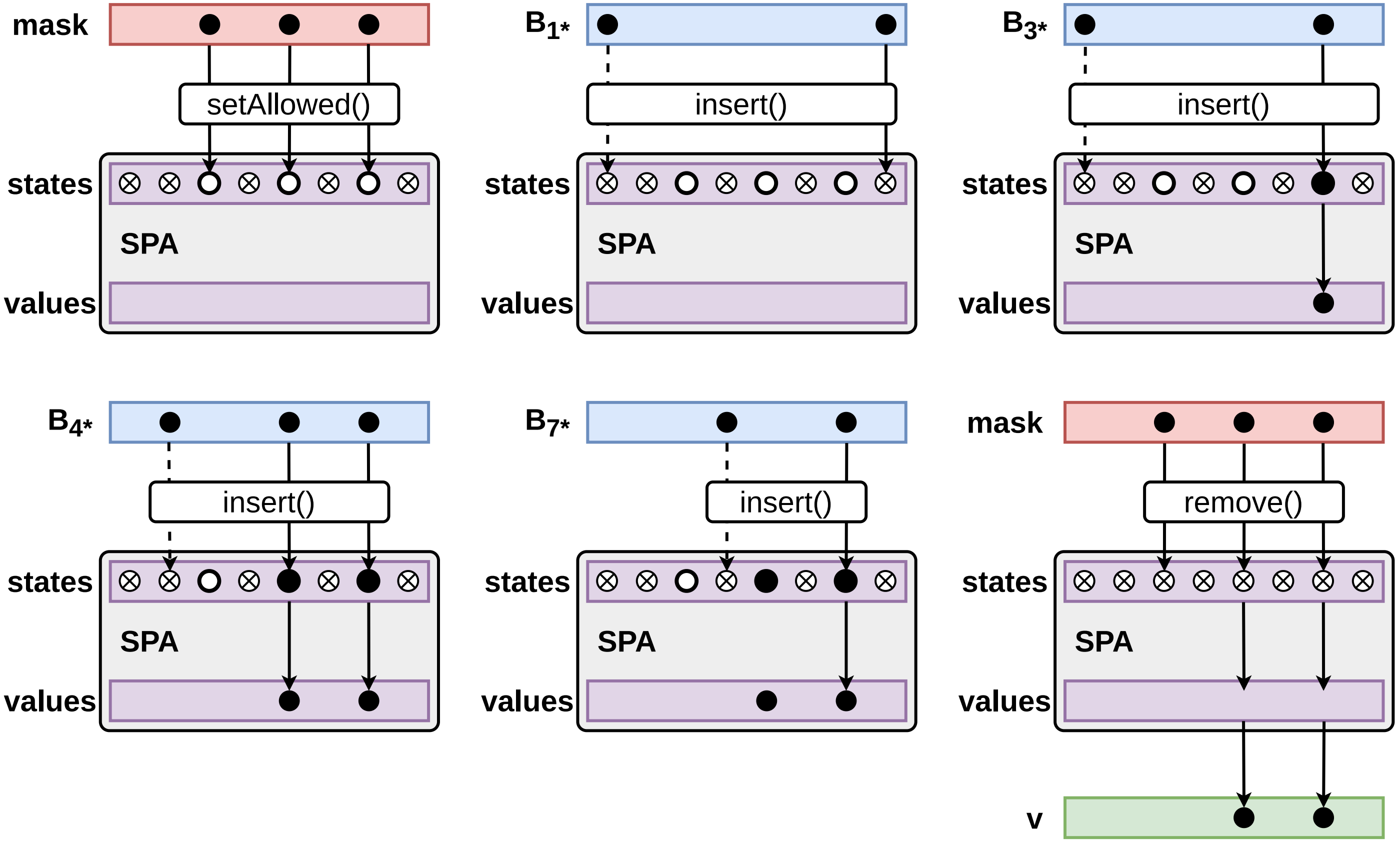
```
for i = 1:n
  Ci* = Mi* .* (Ai* x B)
```

- Row-wise Masked SpGEMM using an accumulator to compute output row C_{i*}.
- The rows corresponding to the column indices of entries in row A_{i*} are merged and filtered through the respective mask entries to compute C_{i*}.
- This merging and filtering process can be performed in a number of ways.

Masked Sparse Accumulator (MSA)

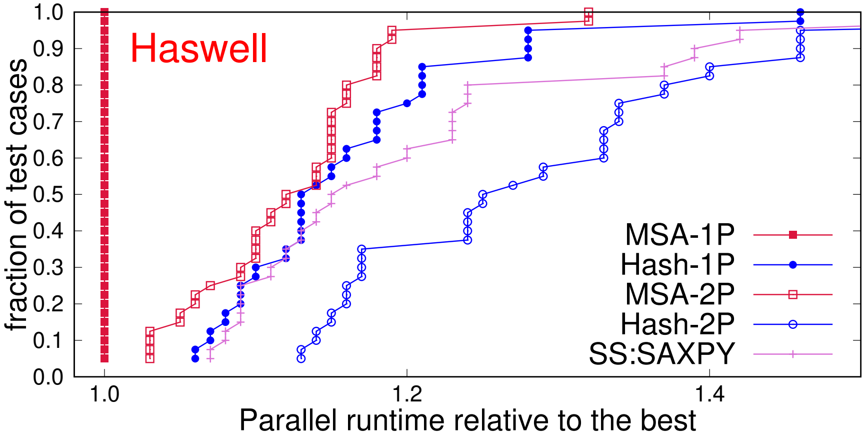
Execution of 1 row of SpGEMM with Masked Sparse Accumulator (MSA)

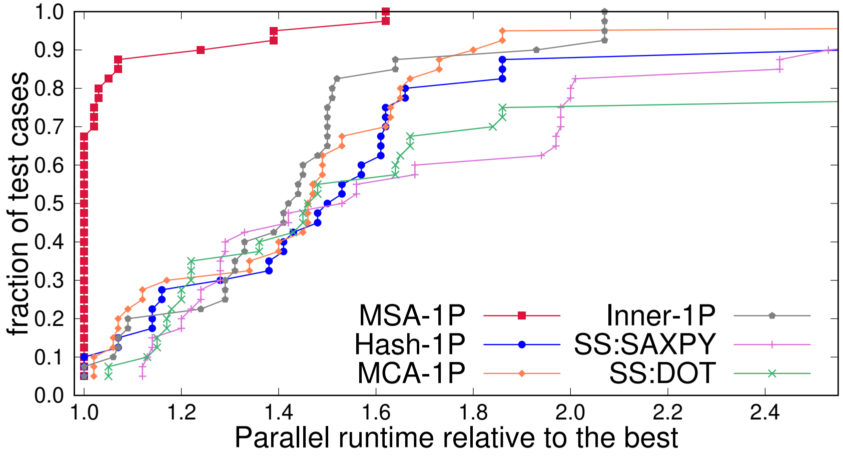
(a) initialize (b) MSA+=u_1 B_{1*} (c) MSA+=u_3 B_{3*} (d) MSA+=u_4 ×B_{4*} (e) MSA+=u_7 ×B_{7*} (f) output

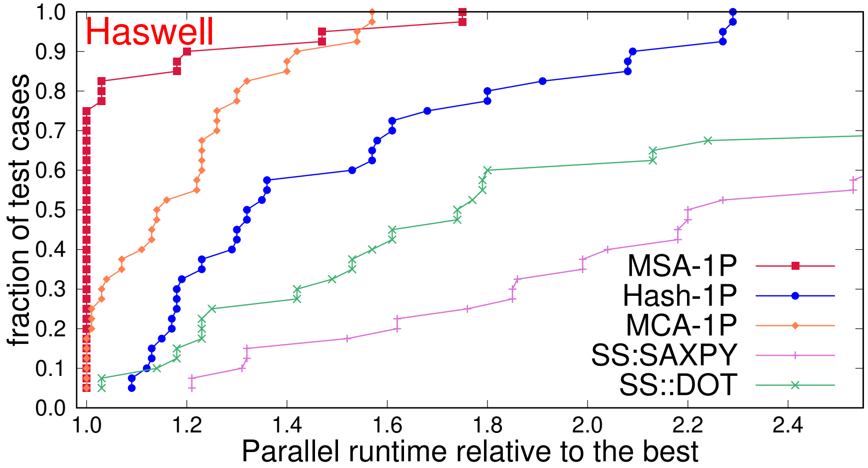


Srdjan Milaković, Oguz Selvitopi, Israt Nisa, Zoran Budimlić, and Aydın Buluç. Parallel algorithms for masked sparse matrix-matrix products. *arXiv preprint arXiv:2111.09947*, 2021 (Poster at PPOPP’22, to appear at ICPP’22)

Performance of Masked SpGEMM algorithms







Top (left): Betweenness Centrality

Top (right): k-truss

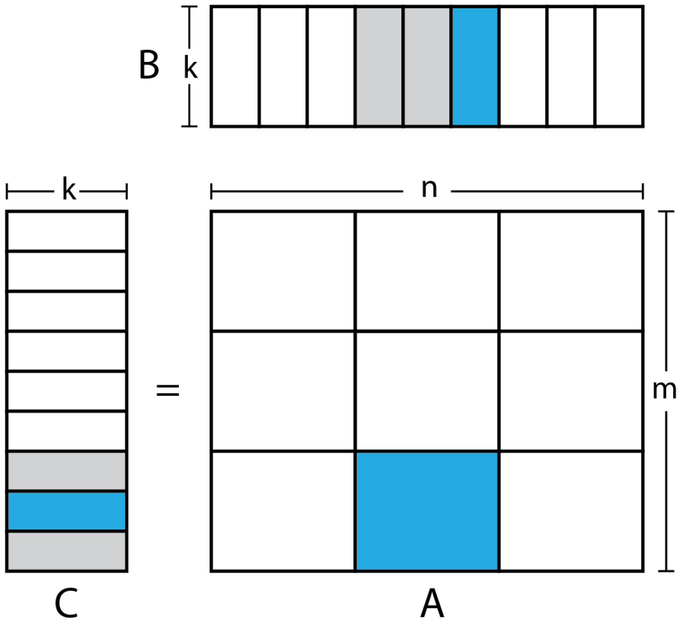
Bottom: Triangle counting

SS is the Suitesparse:GraphBLAS

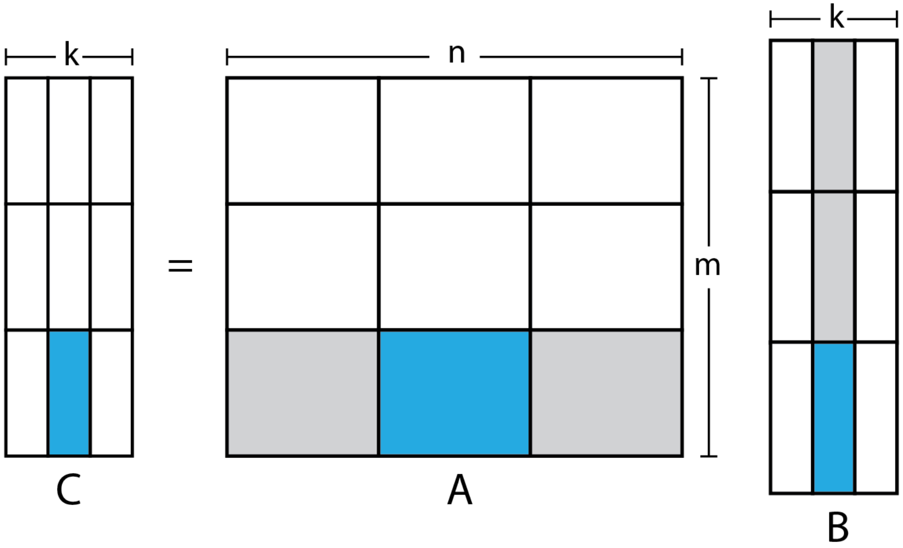
SS:DOT and Inner-1P do sparse dot products

Srdjan Milaković, Oguz Selvitopi, Israt Nisa, Zoran Budimlić, and Aydın Buluç. Parallel algorithms for masked sparse matrix-matrix products. *arXiv preprint arXiv:2111.09947*, 2021 (Poster at PPOPP'22, to appear at ICPP'22)

Distributed SpMM algorithms



A is sparse, **B** and **C** are dense



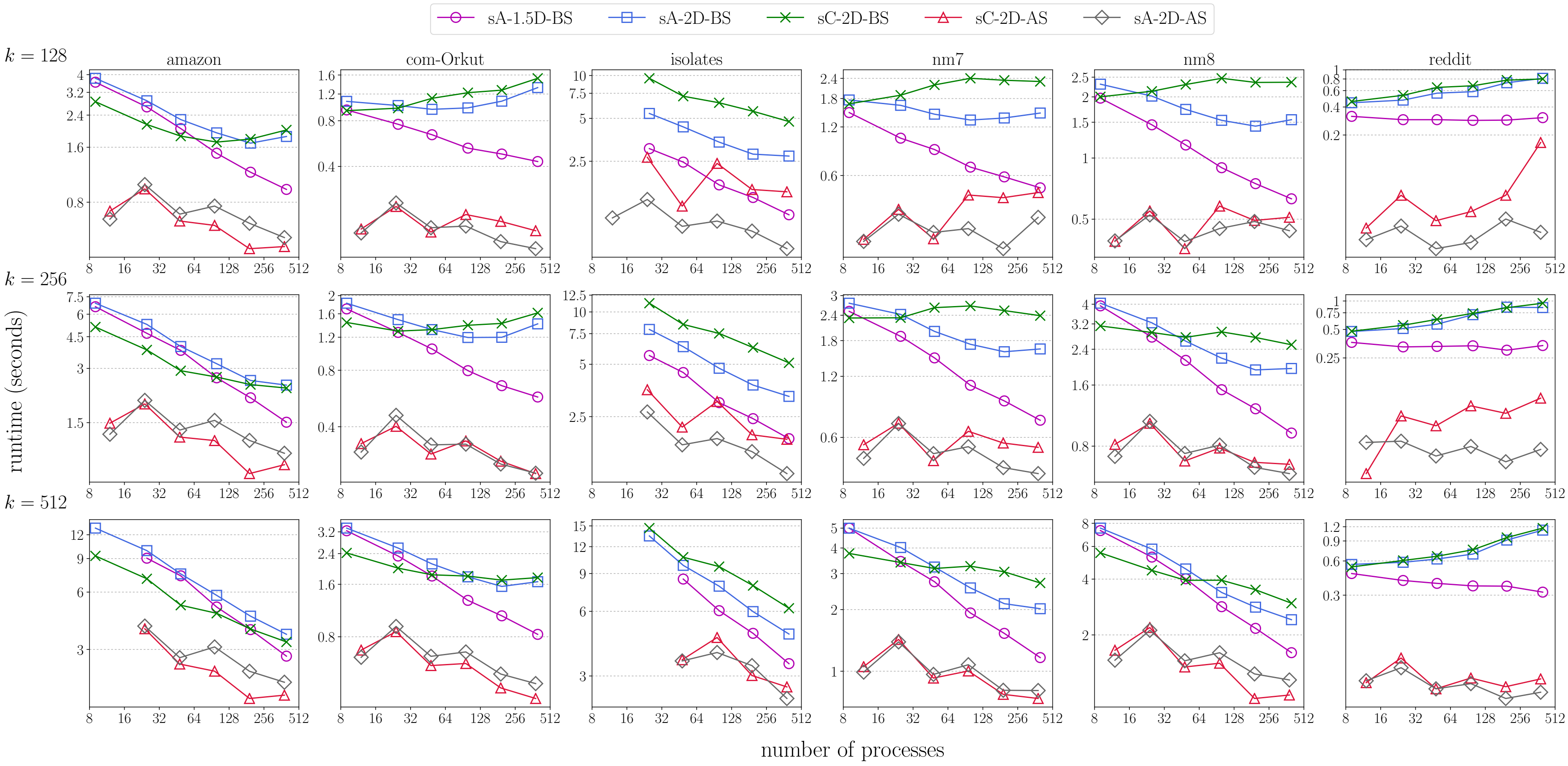
- Stationary A, 1.5D algorithm
- **A** is split on a p/c-by-c grid

- Stationary C, 2D algorithm
- Memory optimal

- 1D algorithm not shown, degeneration of sA-1.5D for the c=1 case
- Right before reduction, sA-1.5D uses c times more dense-matrix memory

Could we do SpMM differently?

BS: bulk-synchronous (MPI)
AS: asynchronous (RDMA)

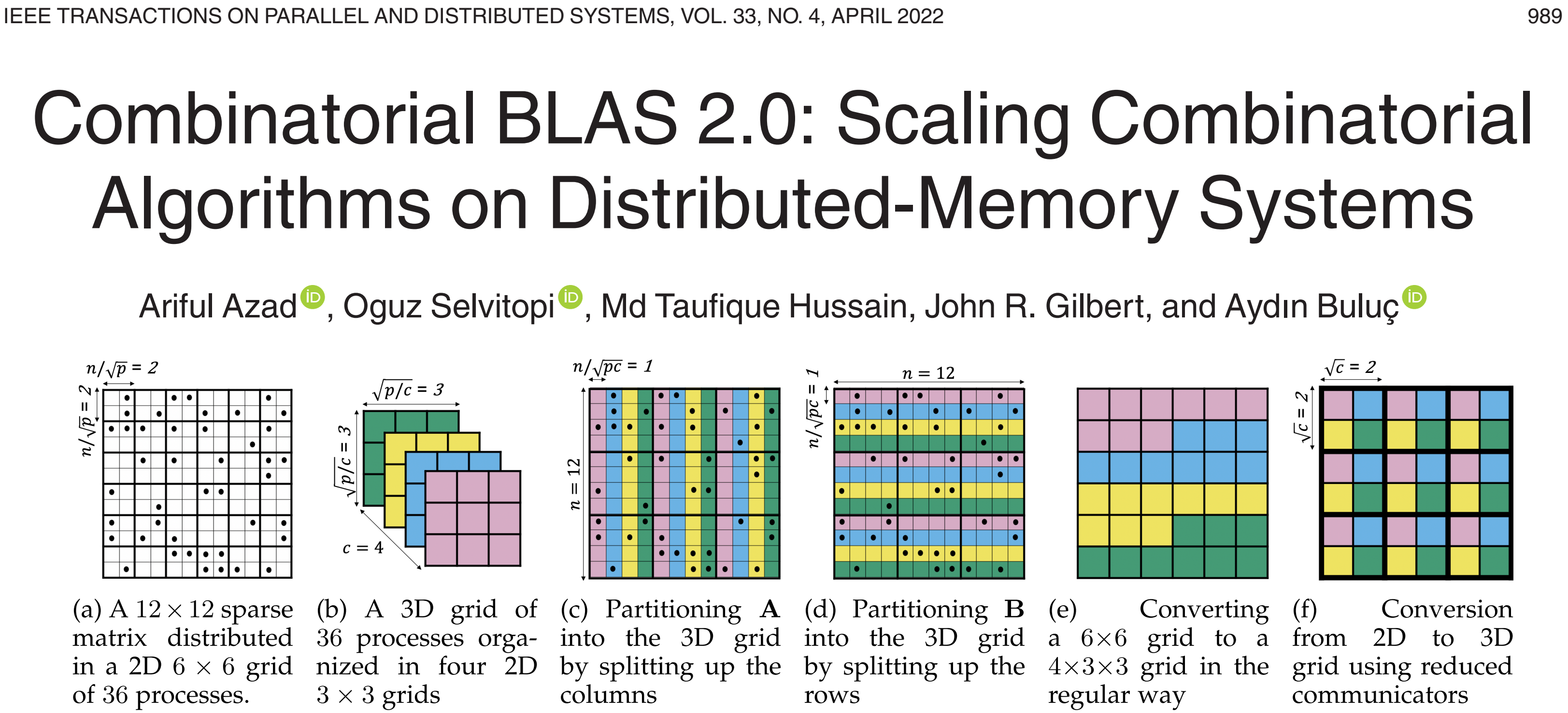


Oguz Selvitopi , Benjamin Brock, Israt Nisa, Alok Tripathy, Katherine Yelick, Aydın Buluç. Distributed-Memory Parallel Algorithms for Sparse Times Tall-Skinny-Dense Matrix Multiplication. ICS’21

High-level outline

- Sparse matrices for graph algorithms
- Sparse matrices for computational biology
- Sparse matrices for machine learning
- Parallel algorithms for sparse matrix primitives
- **Available software**

Combinatorial BLAS 2.0 innovations



IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 33, NO. 4, APRIL 2022 989

Combinatorial BLAS 2.0: Scaling Combinatorial Algorithms on Distributed-Memory Systems

Ariful Azad, Oguz Selvitopi, Md Taufique Hussain, John R. Gilbert, and Aydın Buluç

(a) A 12×12 sparse matrix distributed in a 2D 6×6 grid of 36 processes.

(b) A 3D grid of 36 processes organized in four 2D 3×3 grids

(c) Partitioning **A** into the 3D grid by splitting up the columns

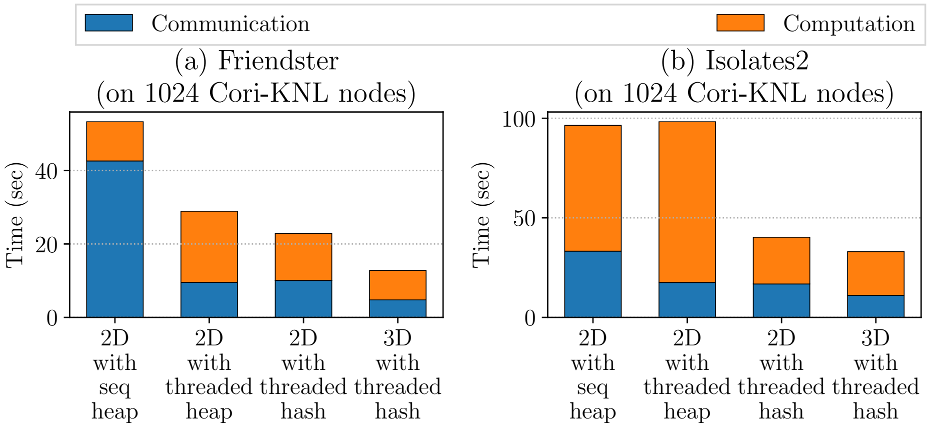
(d) Partitioning **B** into the 3D grid by splitting up the rows

(e) Converting a 6×6 grid to a $4\times3\times3$ grid in the regular way

(f) Conversion from 2D to 3D grid using reduced communicators

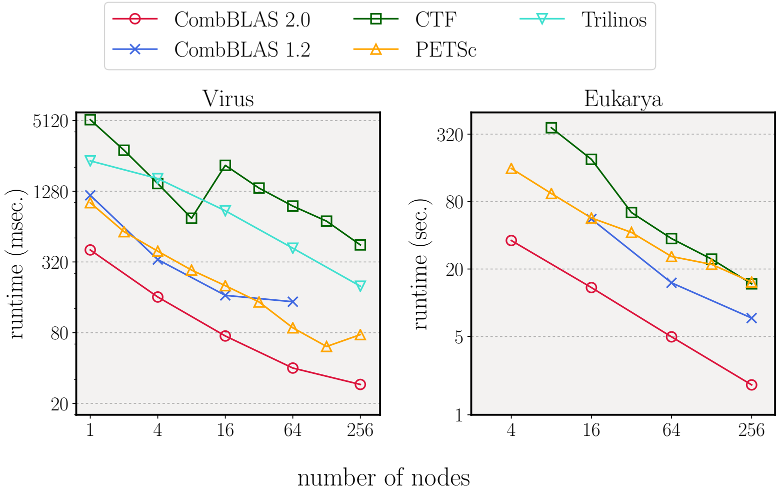
- communication avoiding algorithms,
- hierarchical parallelism via in-node multithreading,
- accelerator support via GPU kernels,
- generalized semiring support,
- implementations of key data structures and functions,
- scalable distributed I/O operations for human-readable files

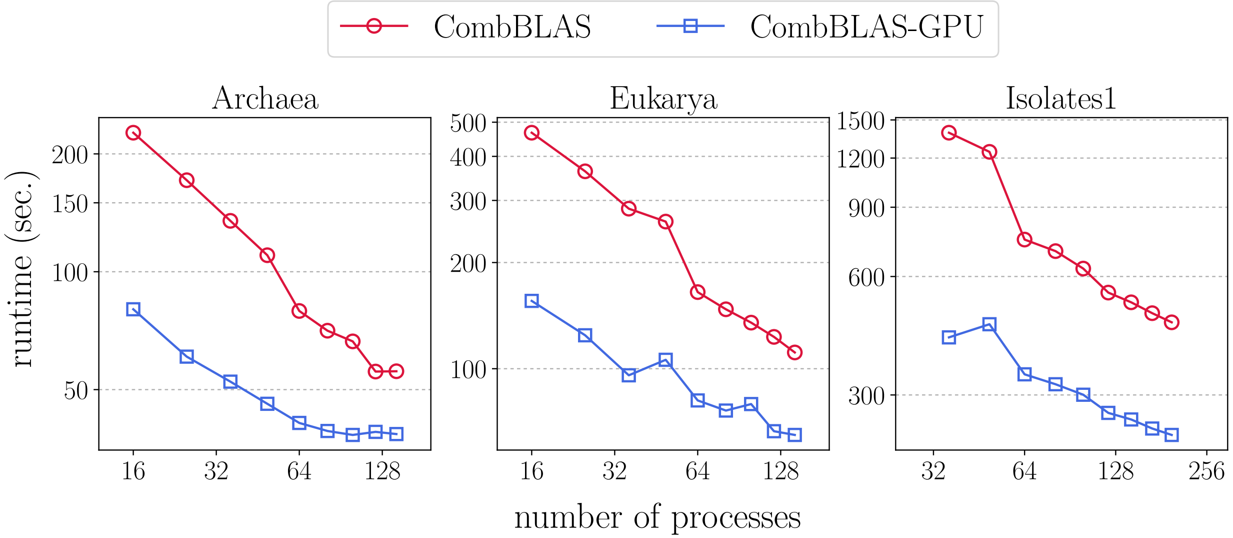
Combinatorial BLAS 2.0 performance



Distributed SpGEMM performance evolution

Parallel SpGEMM runtime of CombBLAS 1.0, 2.0, and other popular parallel sparse linear algebra libraries





Impact of GPU-enabled and disabled CombBLAS backends for HipMCL

GraphBLAST

- First “high-performance” GraphBLAS implementation on the GPU
- Optimized to take advantage of both input and output sparsity
- Automatic direction-optimization through the use of masks
- Competitive with fastest GPU (Gunrock) and CPU (Ligra) codes
- Outperforms multithreaded SuiteSparse::GraphBLAS

Design principles:

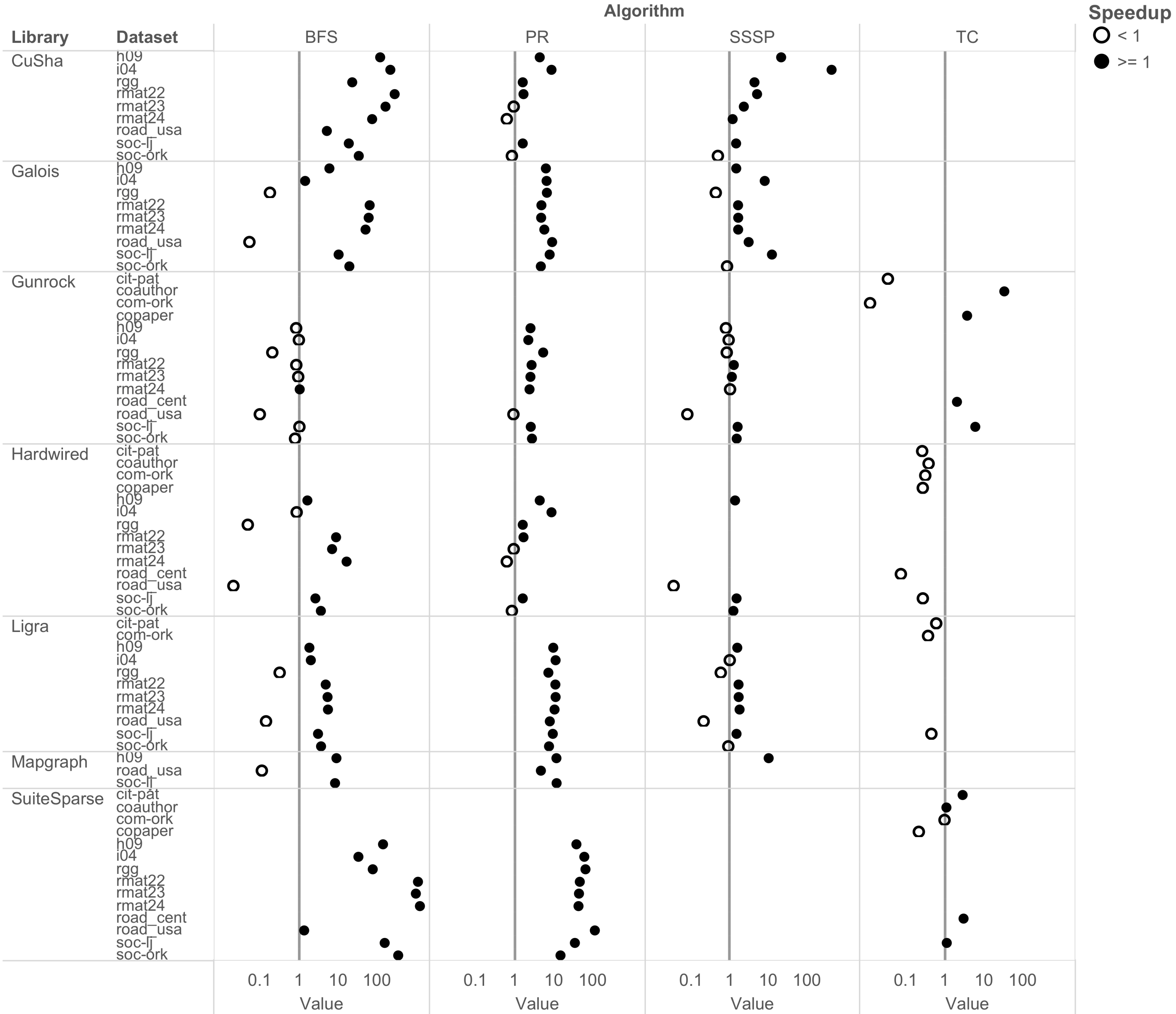
1. Exploit input sparsity => direction-optimization
2. Exploit output sparsity => masking
3. Proper load-balancing => key for GPU implementations

Extensively evaluated on (more implemented, google for github repo)

- Breadth-first-search (BFS)
- Single-source shortest-path (SSSP)
- PageRank (PR)
- Triangle counting (TC)

https://github.com/gunrock/graphblast

Yang, Buluc, Owens, “GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU”, ACM Transactions on Mathematical Software (TOMS), 2022


Algorithm
Library
Dataset
BFS
PR
SSSP
TC
Speedup
< 1
>= 1
CuSha
h09
i04
rgg
rmat22
rmat23
rmat24
road_usa
soc-lj
soc-ork
Galois
h09
i04
rgg
rmat22
rmat23
rmat24
road_usa
soc-lj
soc-ork
Gunrock
cit-pat
coauthor
com-ork
copaper
h09
i04
rgg
rmat22
rmat23
rmat24
road_cent
road_usa
soc-lj
soc-ork
Hardwired
cit-pat
coauthor
com-ork
copaper
h09
i04
rgg
rmat22
rmat23
rmat24
road_cent
road_usa
soc-lj
soc-ork
Ligra
cit-pat
com-ork
h09
i04
rgg
rmat22
rmat23
rmat24
road_usa
soc-lj
soc-ork
Mapgraph
h09
road_usa
soc-lj
SuiteSparse
cit-pat
coauthor
com-ork
copaper
h09
i04
rgg
rmat22
rmat23
rmat24
road_cent
road_usa
soc-lj
soc-ork
0.1
1
10
100
Value
0.1
1
10
100
Value
0.1
1
10
100
Value
0.1
1
10
100
Value

More GraphBLAS implementations

SuiteSparse library (Texas A&M): First fully conforming GraphBLAS release

http://faculty.cse.tamu.edu/davis/suitesparse.html

GraphBLAS C (IBM): the second fully conforming release

https://github.com/IBM/ibmgraphblas

GBTL: GraphBLAS Template Library (CMU/SEI/IU/PNNL): GraphBLAS C++ implementation

https://github.com/cmu-sei/gbtl

ALP/GraphBLAS (Huawei): GraphBLAS C++ implementation

https://gitee.com/CSL-ALP/graphblas.git

Python bindings:

- PyGB: A python wrapper around GBTL (UW/PNNL/CMU)
  - https://github.com/jessecoleman/gbtl-python-binding
- pygraphblas: A python wrapper around SuiteSparse GraphBLAS
  - https://github.com/michelp/pygraphblas
- grblas: Anaconda's python wrapper around SuiteSparse GraphBLAS
  - https://github.com/metagraph-dev/grblas

pggraphblas: A PostgreSQL wrapper around Suite Sparse GraphBLAS

https://github.com/michelp/pggraphblas

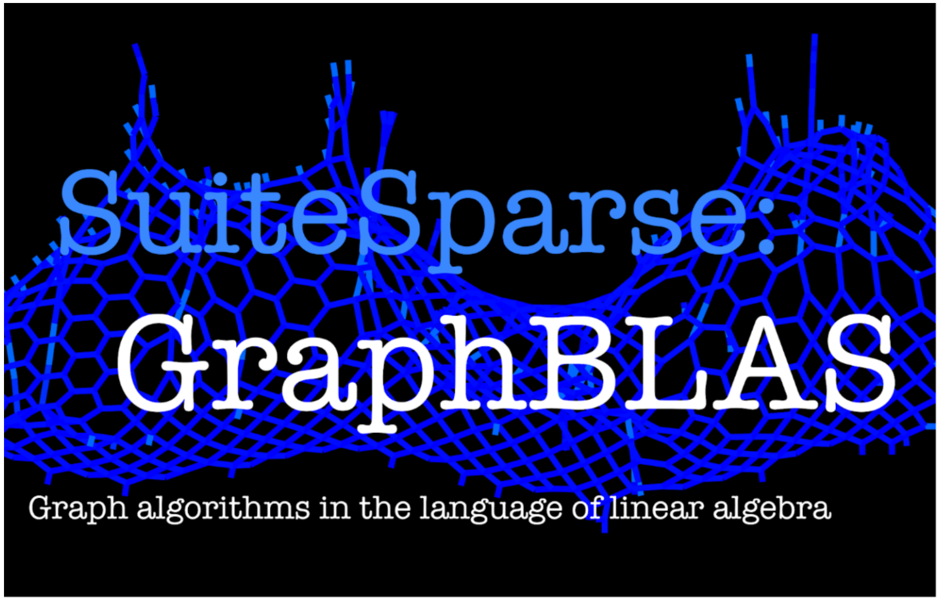
Julia wrapper around SuiteSparse

SuiteSparseGraphBLAS.jl

Matlab and Julia wrappers around SuiteSparse GraphBLAS

https://aldenmath.com

Suitesparse:GraphBLAS



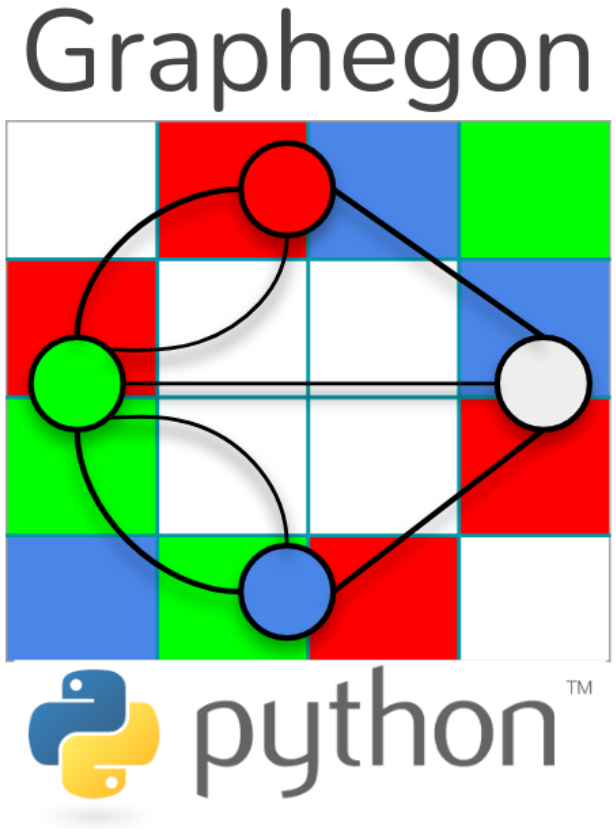
Tim Davis
Texas A&M
University

SuiteSparse:GraphBLAS : open-source GraphBLAS library with OpenMP (Apache 2.0)

- high performance, internal parallelism, allows for easy-to-code fast graph algorithms
- fully compliant with v1.3 C API
- MATLAB interface, many overloaded operators and functions (C(M)=A*B, etc)
- GxB extensions: ANY monoid, import/export, positional ops in semirings, scalars, float and double complex, select operation, PAIR operator, type query, subassign, ...
- matrix data structures: sparse (CSR/CSC) / hypersparse / bitmap / full
- https://people.engr.tamu.edu/davis/GraphBLAS.html

logo: mathematical art by T. D. http://www.notesartstudio.com/sincere.html

pygraphblas



- pygraphblas is developed by Graphegon.
- Open-source package using the SuiteSparse:GraphBLAS library.
- Specializing in GraphBLAS solutions using C, Python and PostgreSQL.
- https://github.com/Graphegon/pygraphblas

Pygraphblas Documentation at: https://graphegon.github.io/pygraphblas/pygraphblas/index.html

Conclusions

- Sparse matrix techniques underlie computations from disparate fields:
  a. Scientific computing
  b. Machine learning
  c. Graph analysis
  d. Bioinformatics
- GraphBLAS already seem to have the right abstraction with its flexible **masks** and **semirings** to be the default backend of many of these computations
- Extreme parallelism and data, and hence **the need for distributed memory parallelism** is here to stay and will get worse
- **Communication-avoiding algorithms, and novel data structures for sparse matrices** will be the key to overcome these adverse technological trends

Acknowledgments

Ariful Azad, Vivek Bharadwaj, Ben Brock, Zoran Budimlić, Tim Davis, James Demmel, Saliya Ekanayake, Marquita Ellis, John Gilbert, Giulia Guidi, Md Taufique Hussain, Jeremy Kepner, Nikos Krypides, Tim Mattson, Scott McMillan, Srđan Milaković, Jose Moreira, Israt Nisa, John Owens, Georgios Pavlopoulos, Doru Popovici, Gabriel Raulet, Dan Rokhsar, Oguz Selvitopi, Yu-Hang Tang, Alok Tripathy, Carl Yang, Kathy Yelick.

Our Research Team: http://passion.lbl.gov

Our work is funded by U.S. DEPARTMENT OF ENERGY | Office of Science