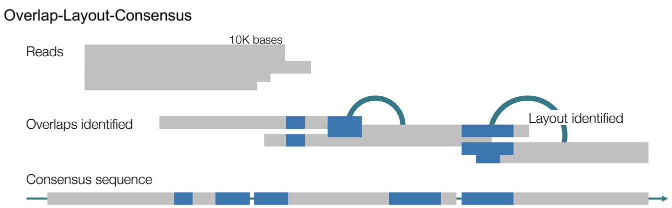
Distributed Sparse Matrices in Graph Algorithms and Graph Learning

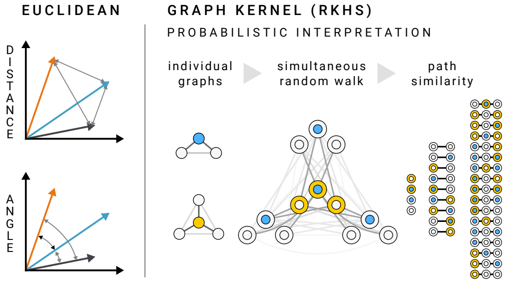
Aydın Buluç
Lawrence Berkeley National Laboratory & UC Berkeley
Keynote at GTA[3] Workshop at IEEE BigData
December 17, 2022

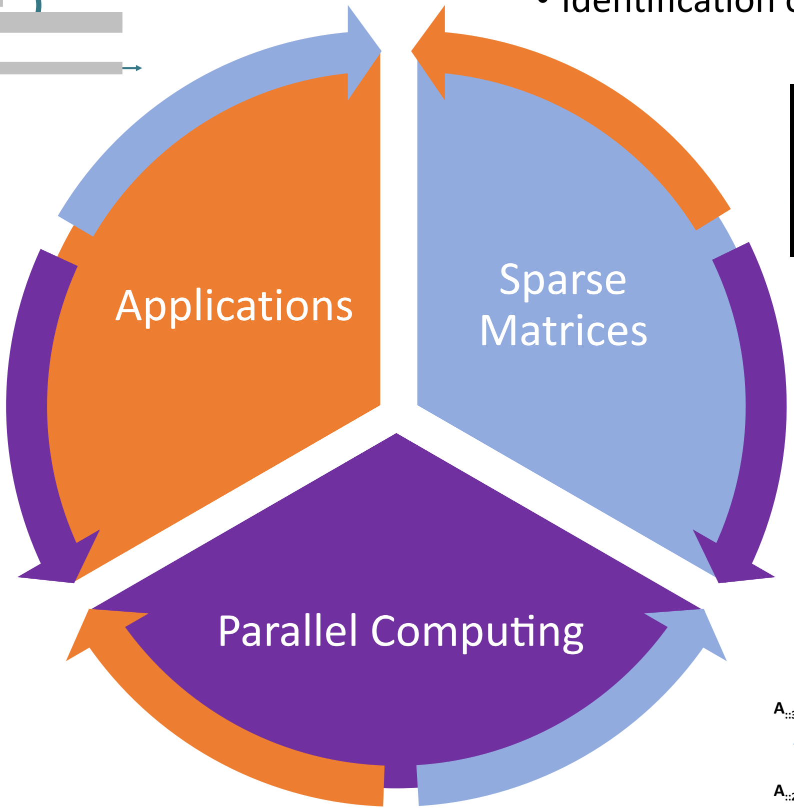
PASSION Lab Research Agenda

http://passion.lbl.gov

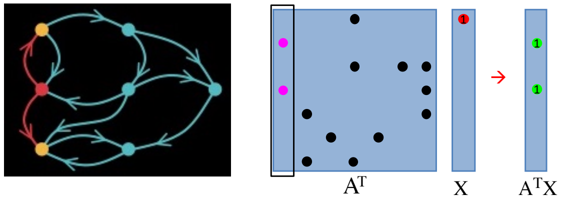


- Genomics
- Graph analysis
- Proteomics
- Graph learning



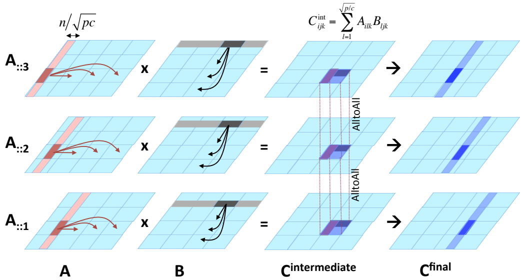


- New sparse data structures and algorithms
- Identification of computational primitives



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

Communication-avoiding algorithms for sparse matrices



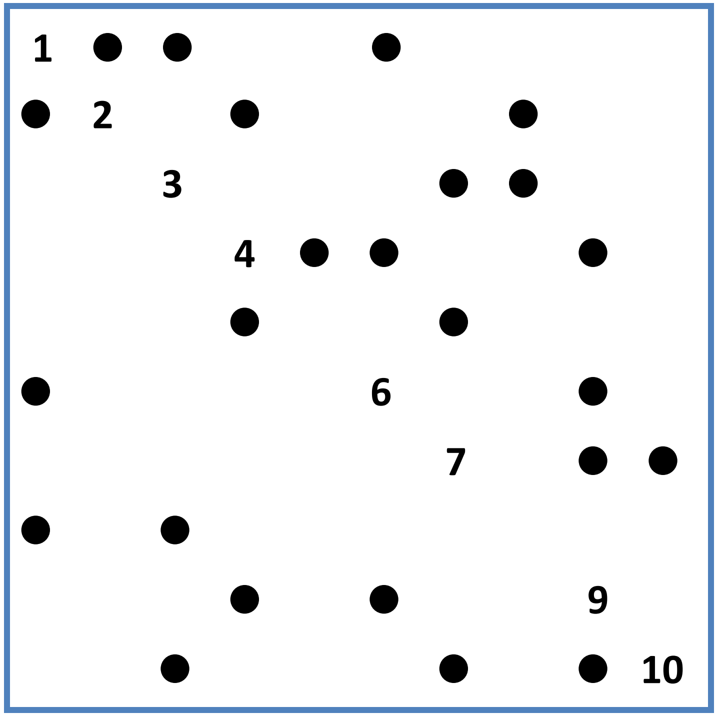
- Parallel data structures
- Parallel programming
- Communication bounds

Sparse Matrices

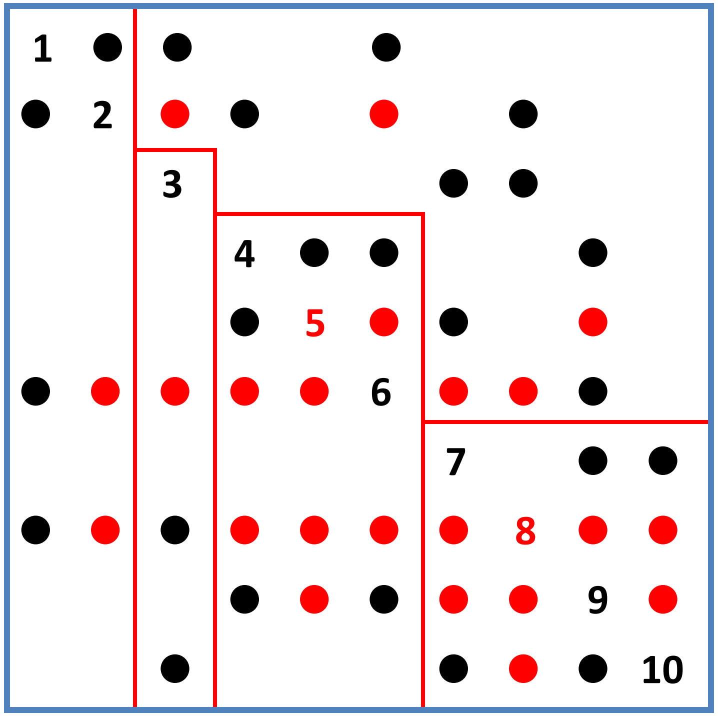
"I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were 'sparse' in the matrix, and that typically the triangular matrices associated with the forward and back solution provided by Gaussian elimination would remain sparse if pivot elements were chosen with care"

- 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 graph learning
- Parallel algorithms for sparse matrix primitives
- Available software

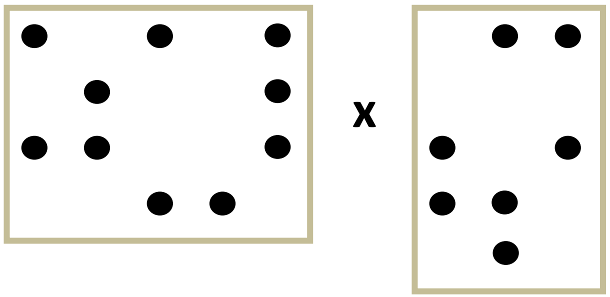
The case for sparse matrices

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

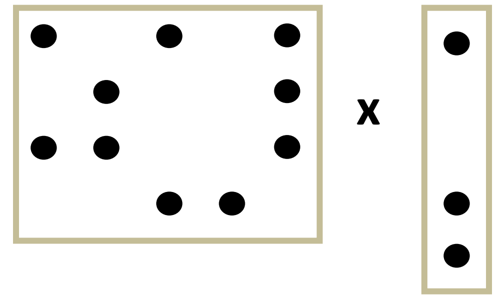
Traditional graph computations	**Graphs in the language of linear algebra**
Data driven, unpredictable communication.	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine grained data accesses, dominated by latency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

Linear-algebraic primitives for graphs

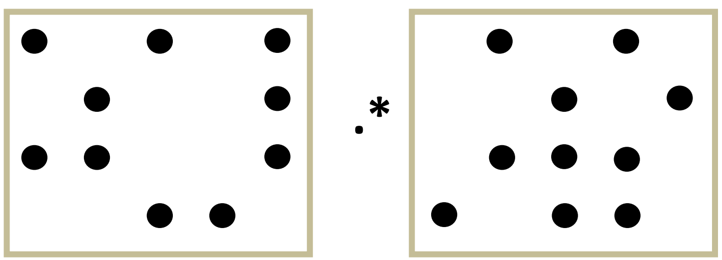
Sparse matrix X sparse matrix



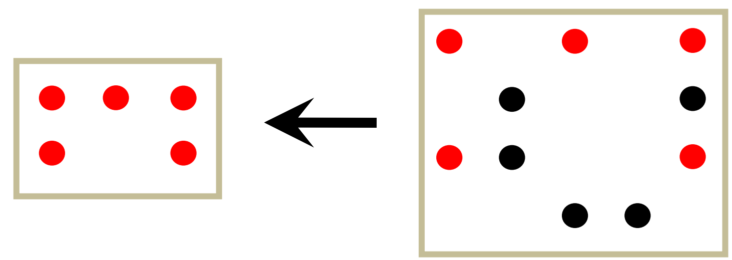
Sparse matrix X sparse vector



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

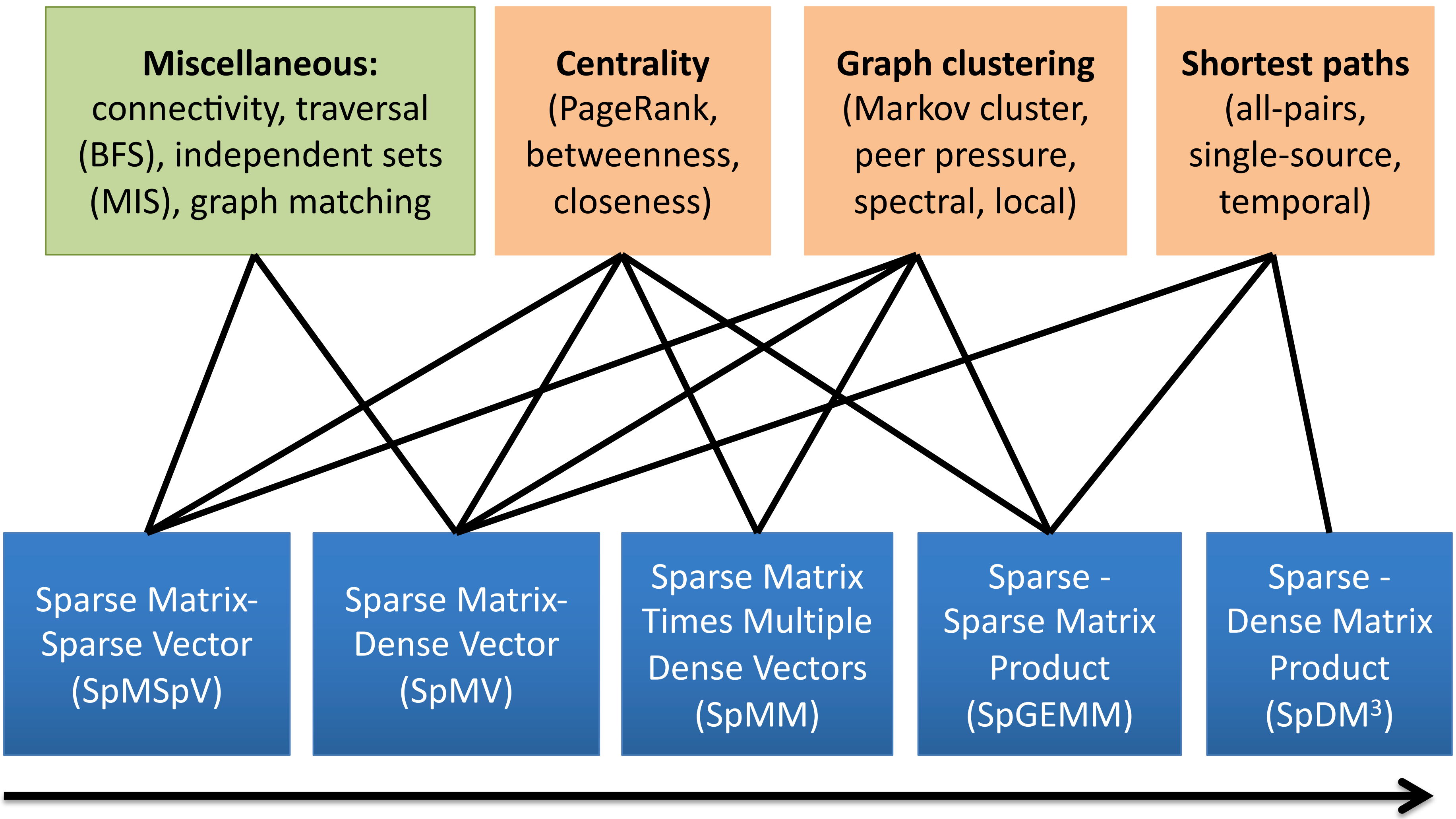
Examples of semirings in graph algorithms

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



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

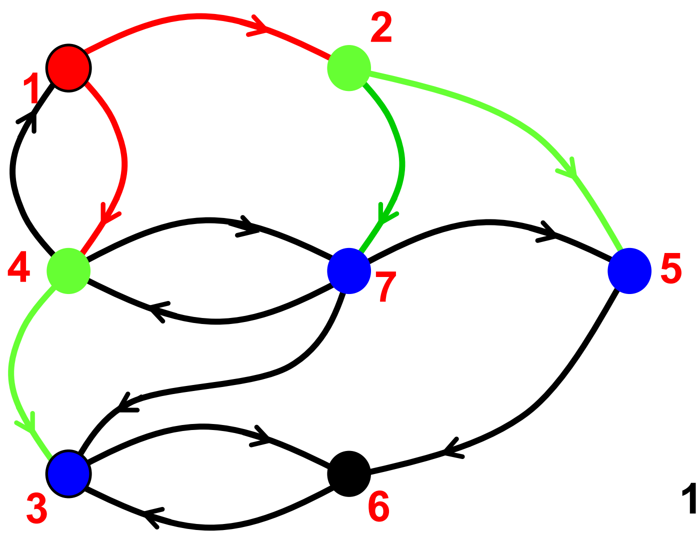
- **Goal:** A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an actual Application Programming Interface (API) that
  i. is faithful to the mathematics as much as possible, and
  ii. enables efficient implementations on modern hardware.
- **Impact:** All graph 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]);
```

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

B. Brock, A. Buluç, T. Mattson, S. McMillan, J. Moreira, “The GraphBLAS C API Specification”, version 2.0.0

Pattern 1: Sparse matrix times sparse vector (SpMSpV)



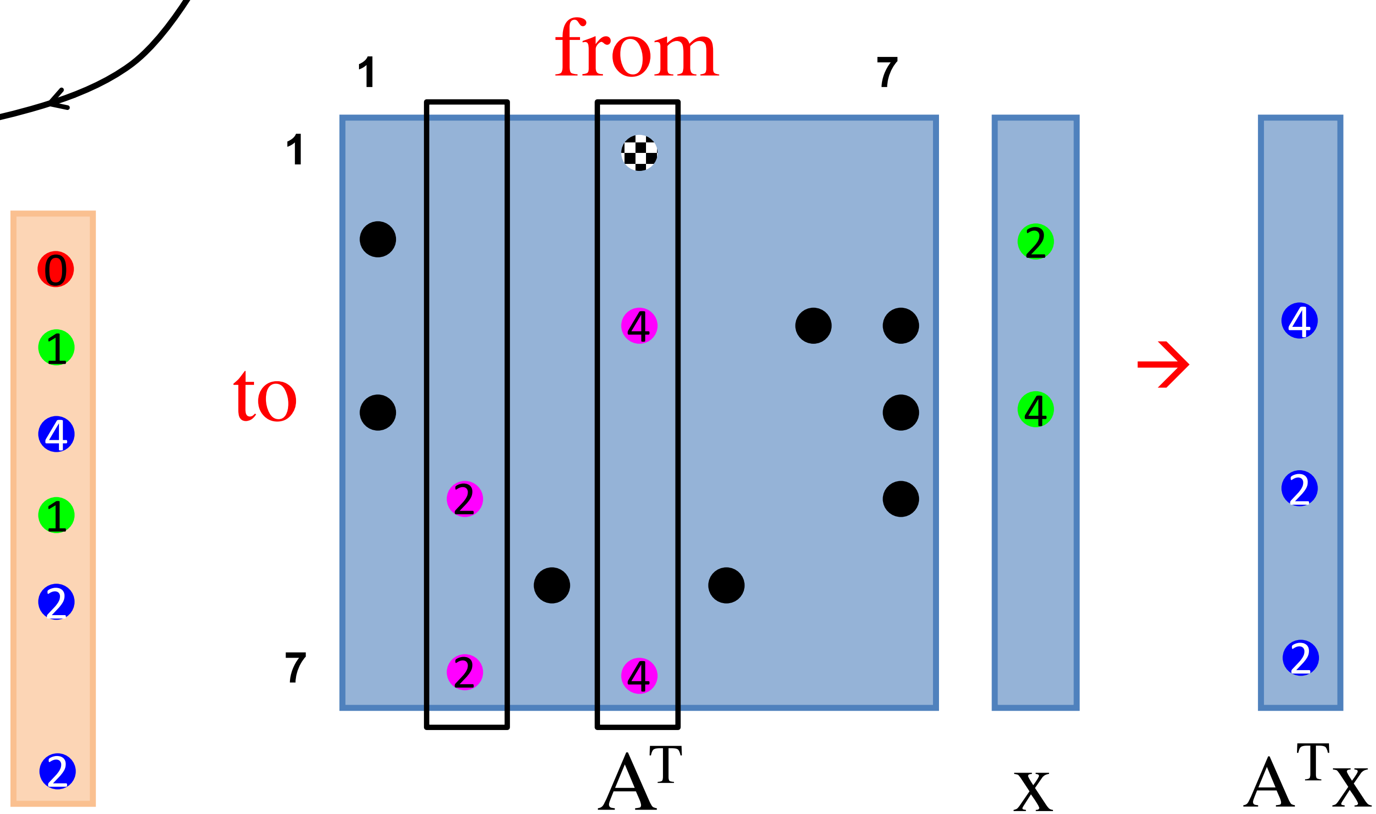
Single-source traversal:

BFS, connected components, matching, ordering, etc.

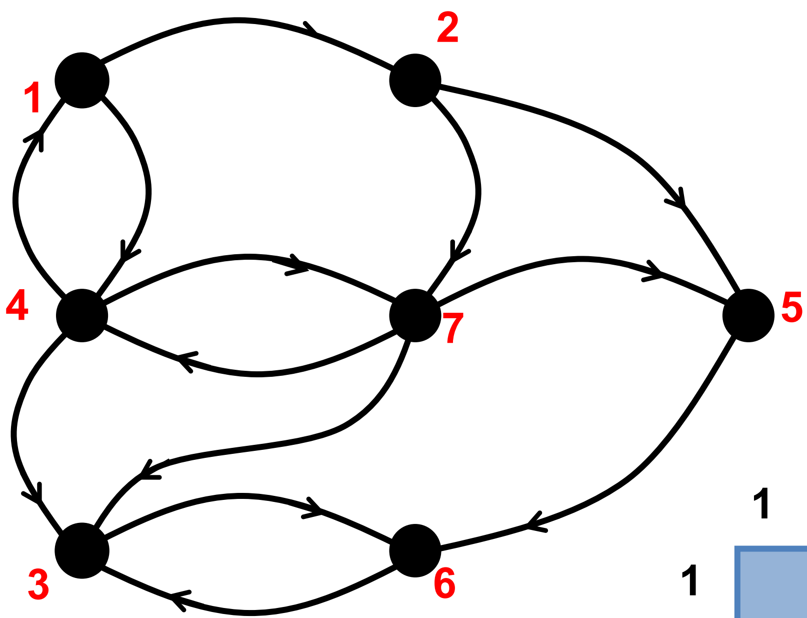
GrB_mxv(y, p, <semiring>, A, x, <desc>)

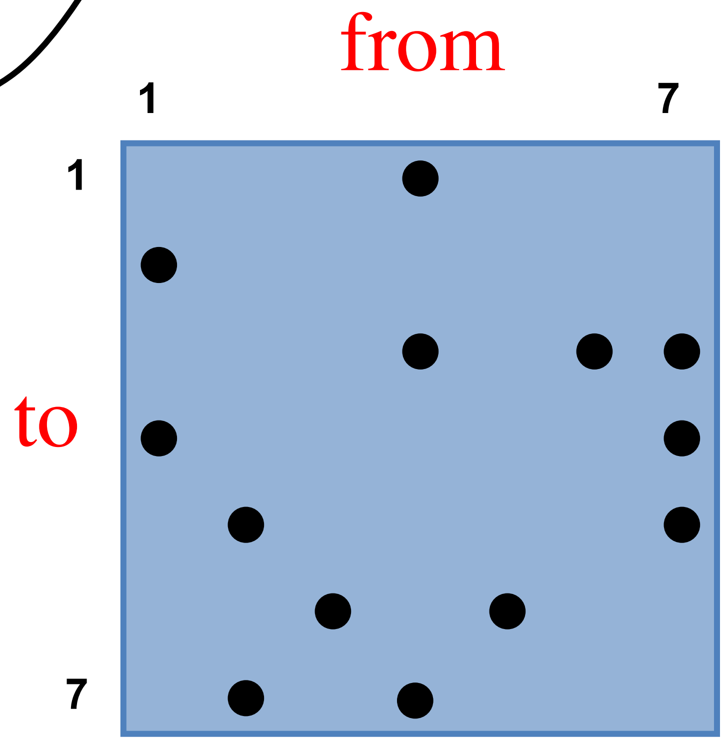
A: sparse adjacency matrix
x: sparse input vector (previous frontier)
p: mask (already discovered vertices)

parents (p):

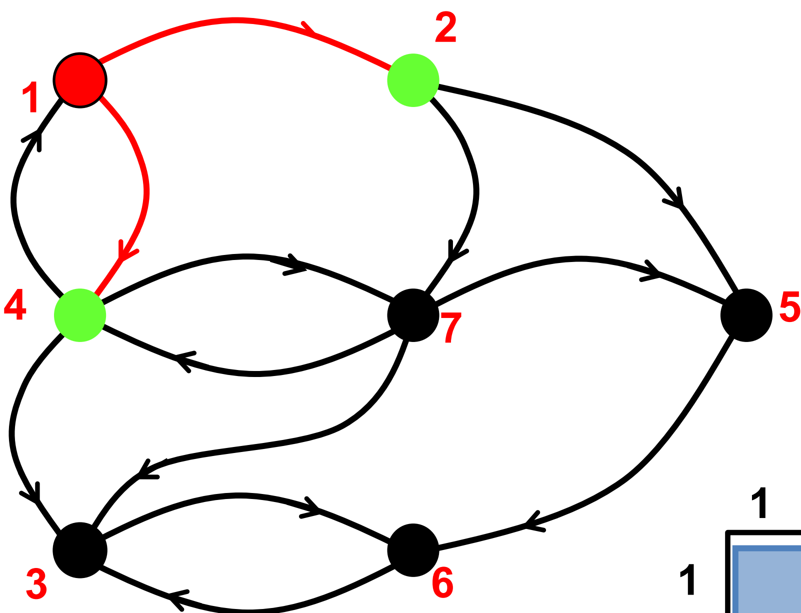


Breadth-first search in the language of matrices





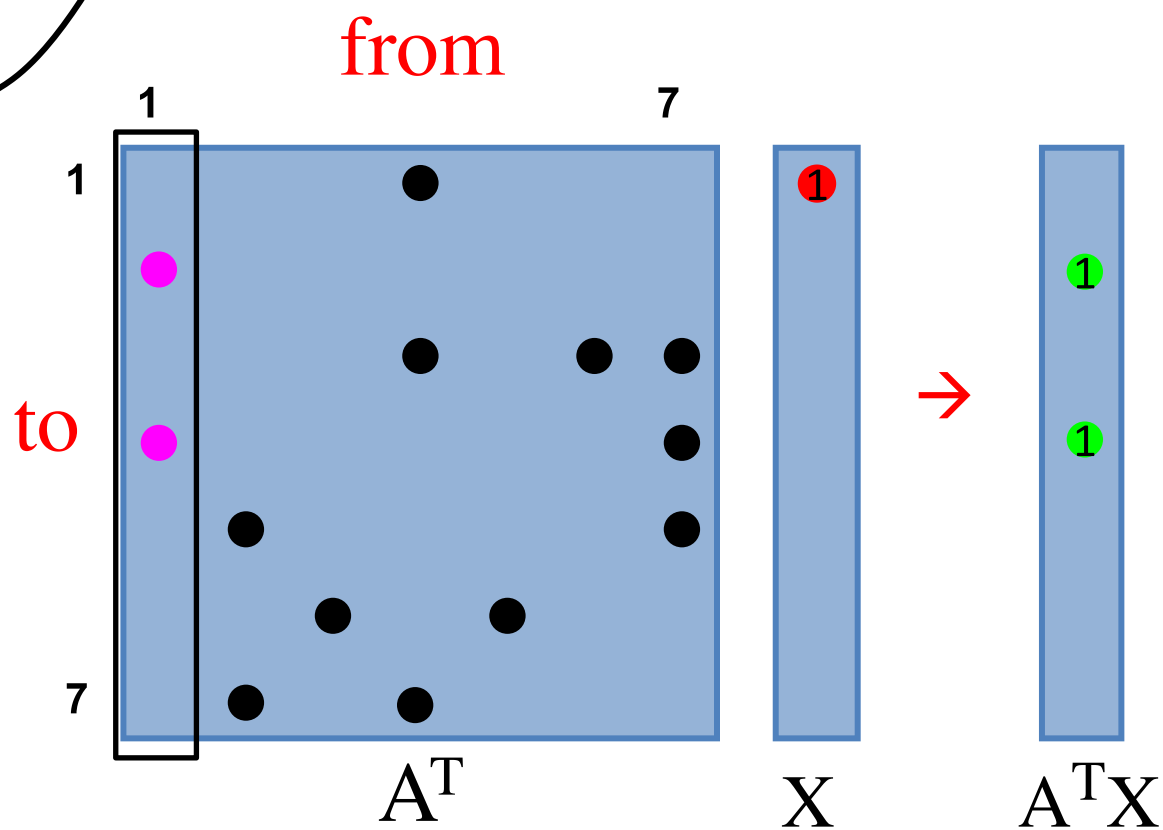
A^T



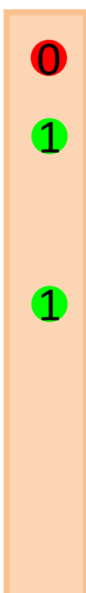
Particular semiring operations:

Multiply: select2nd

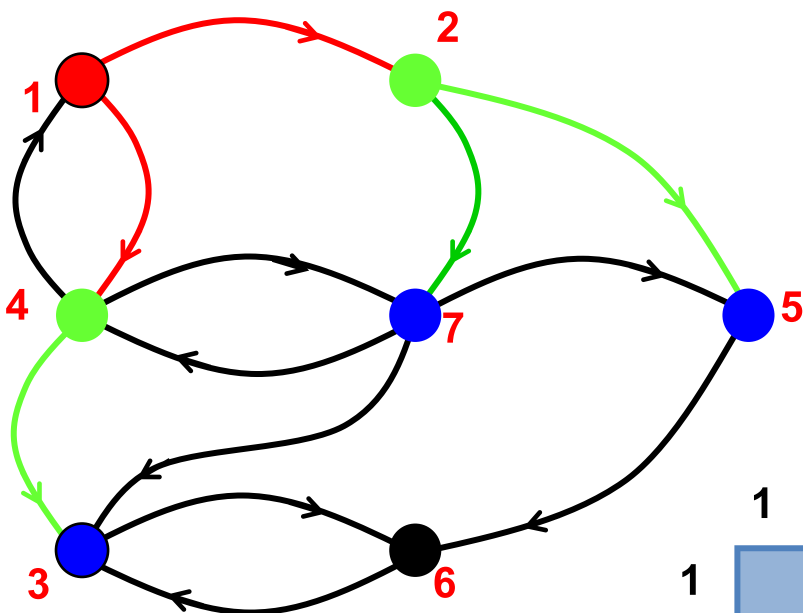
Add: minimum

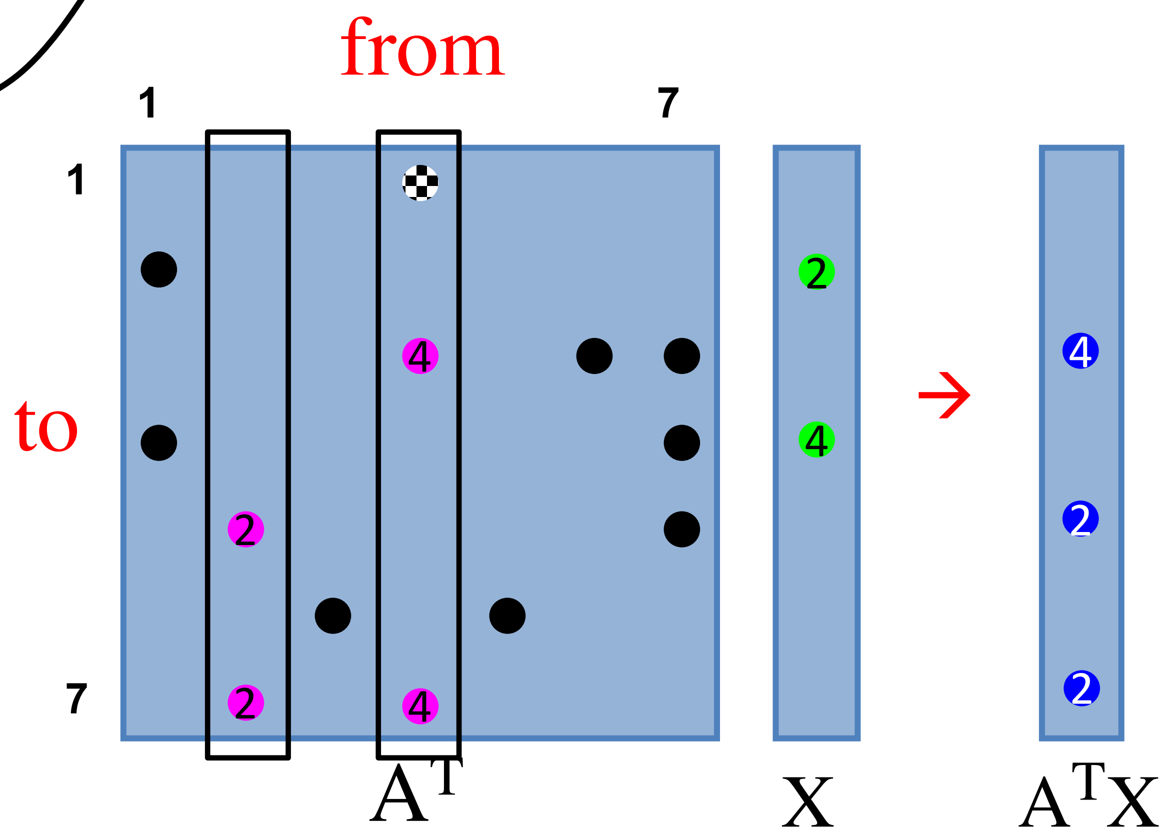


parents:

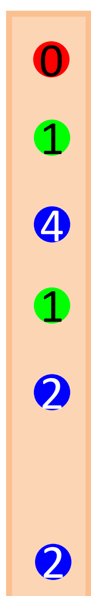


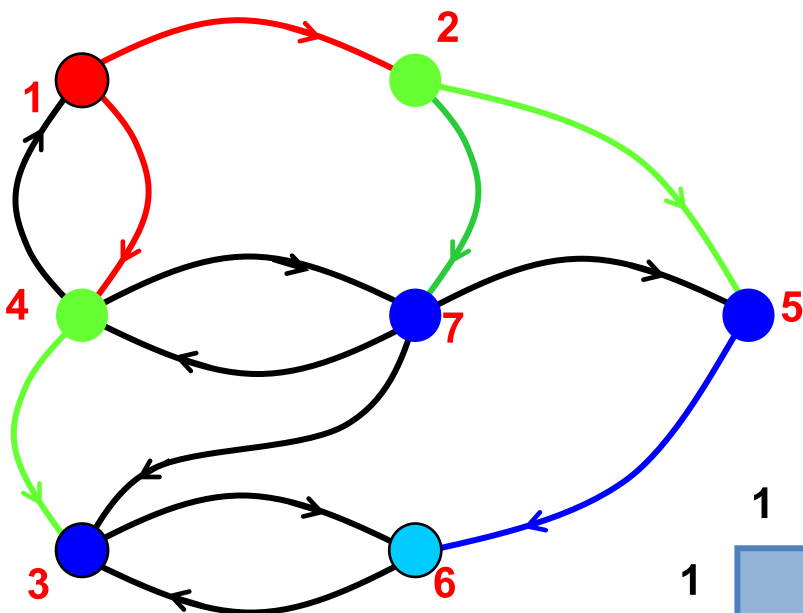
Select vertex with <u>minimum</u> label as parent





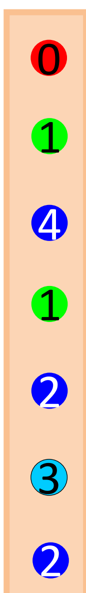
parents:

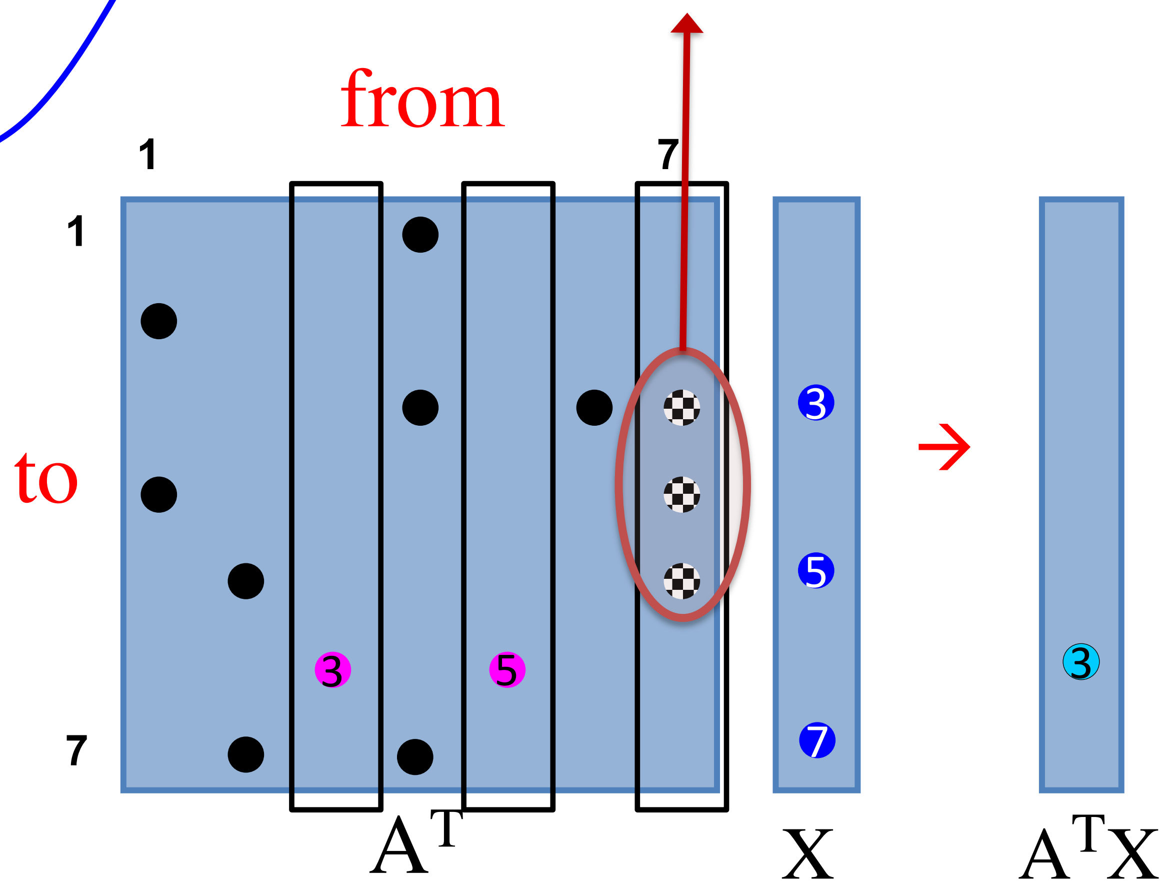


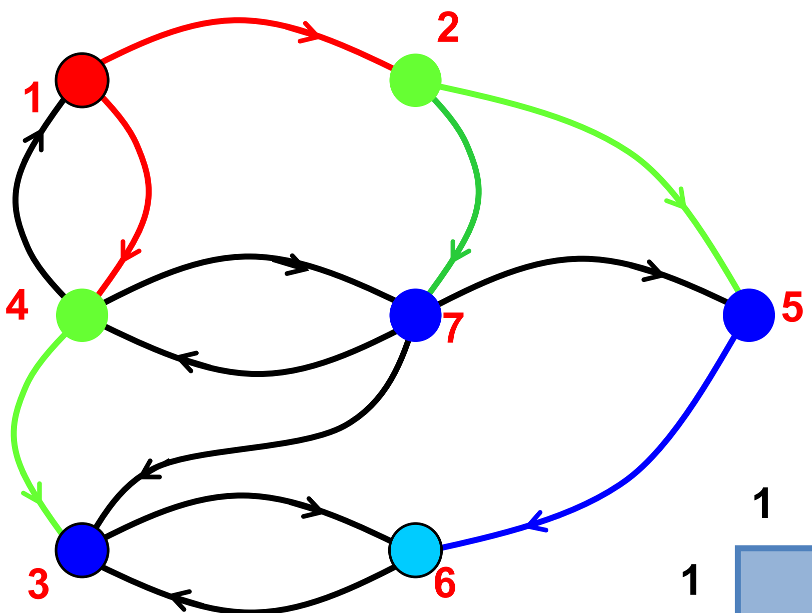


- Masks avoid formation of temporaries and can enable automatic direction optimization
- These footballs are nonzeros that are ***masked out*** by the parents array

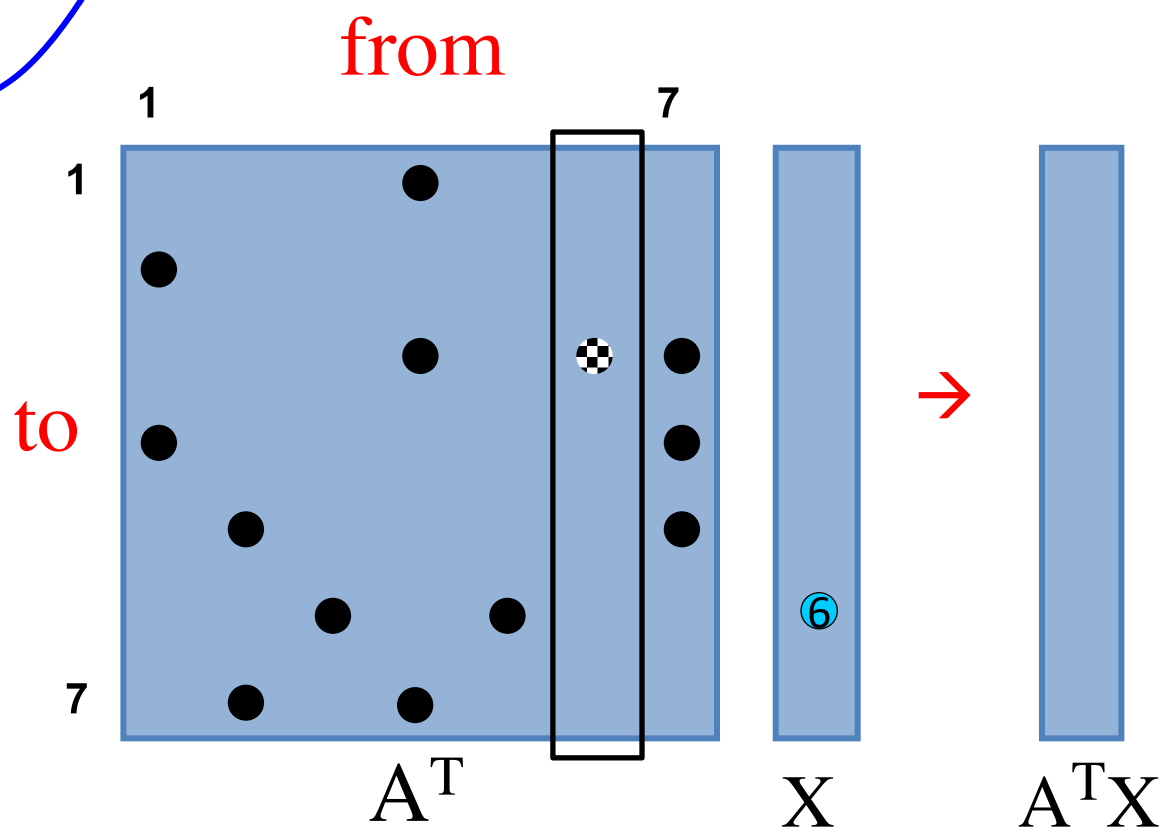
parents:







1
2
4
7
5
3
6

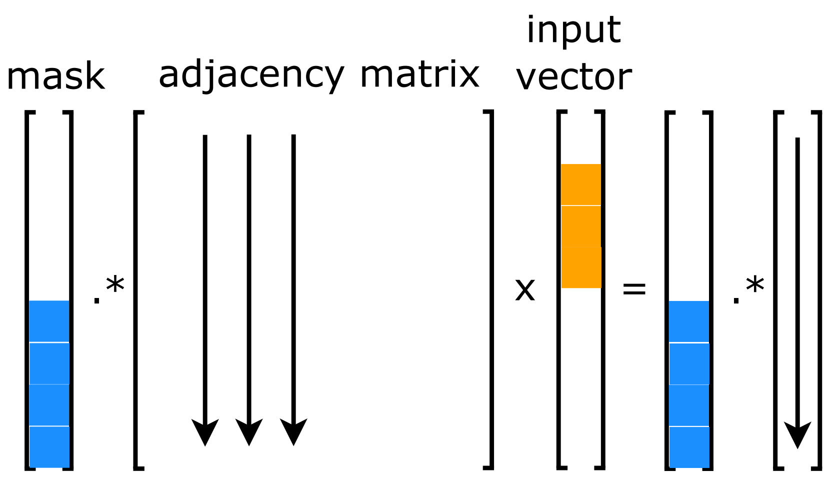


from
1
7
1
to
7
6
A^T
X
A^T X

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

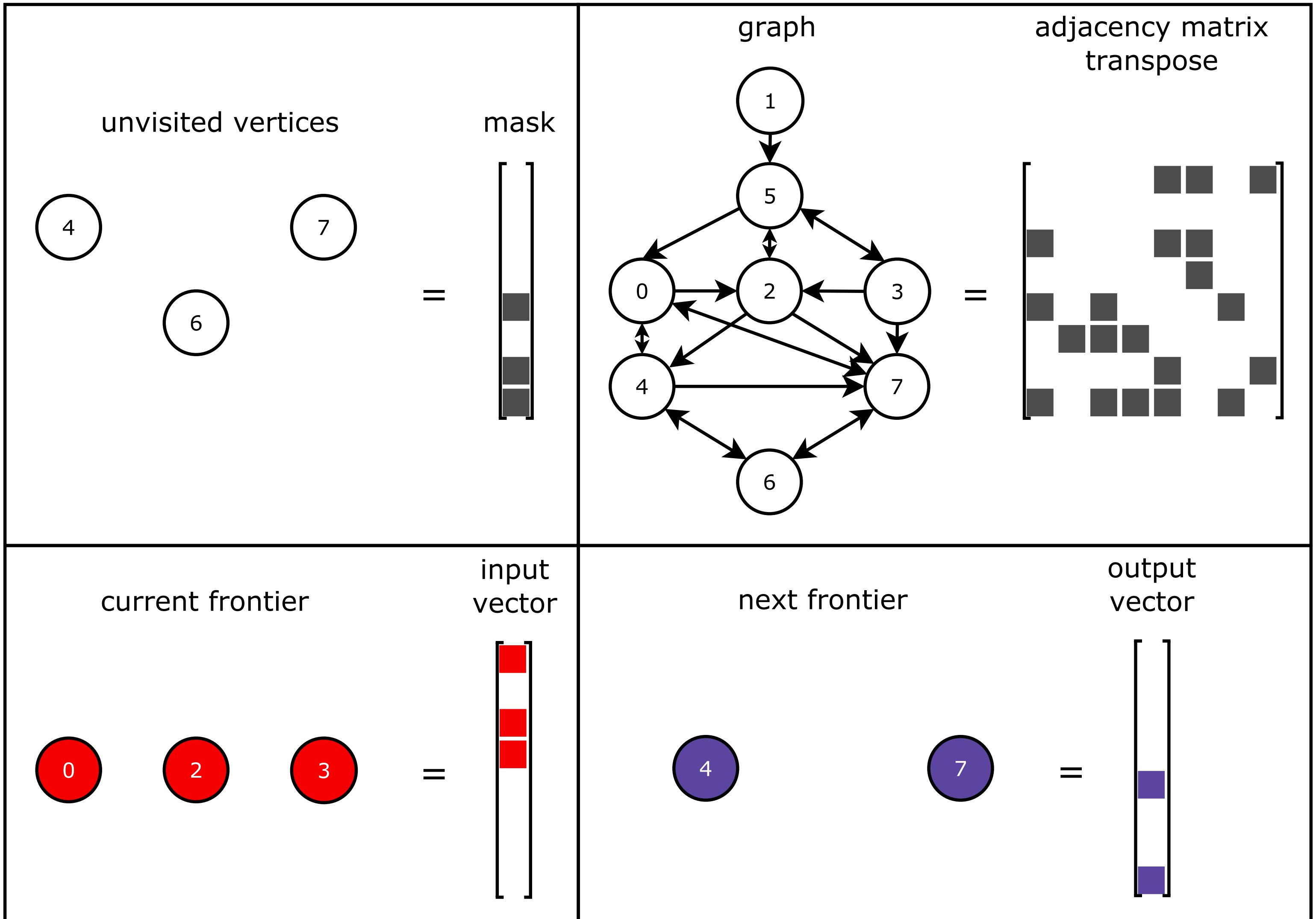
Idea was to run the same column-based algorithm, but checking against a mask before writing to output

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

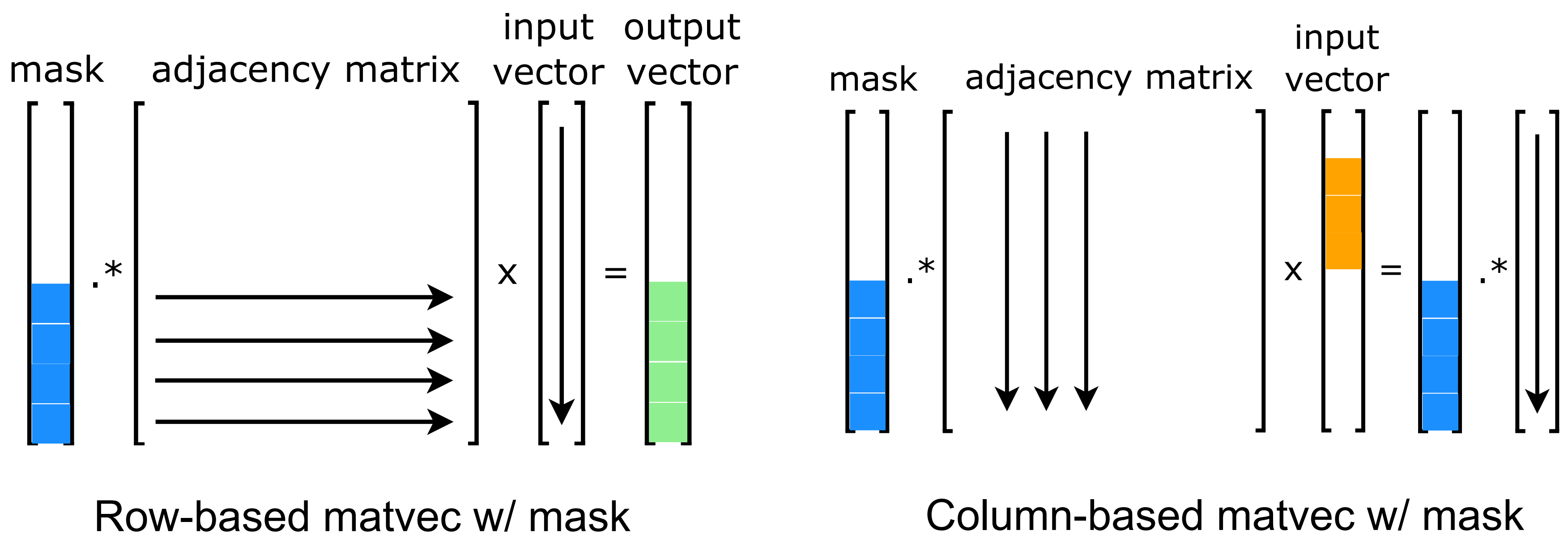
- Carl Yang and I pondered quite a bit on whether it was possible to implement direction optimization in the language of matrices *
- Push-pull (also known as direction optimization) was just about running a row- vs. column-based matvec
- But it wouldn't be competitive it its pure form because you were 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
4
7
6
=
graph
1
5
0
2
3
4
7
6
=
adjacency matrix transpose
current frontier
input vector
0
2
3
=
next frontier
output vector
4
7
=

Masks make “pull” implementable competitively in GraphBLAS



- **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.
```

Pattern 2: Sparse matrix times sparse matrix (SpGEMM)

Multi-source traversal:

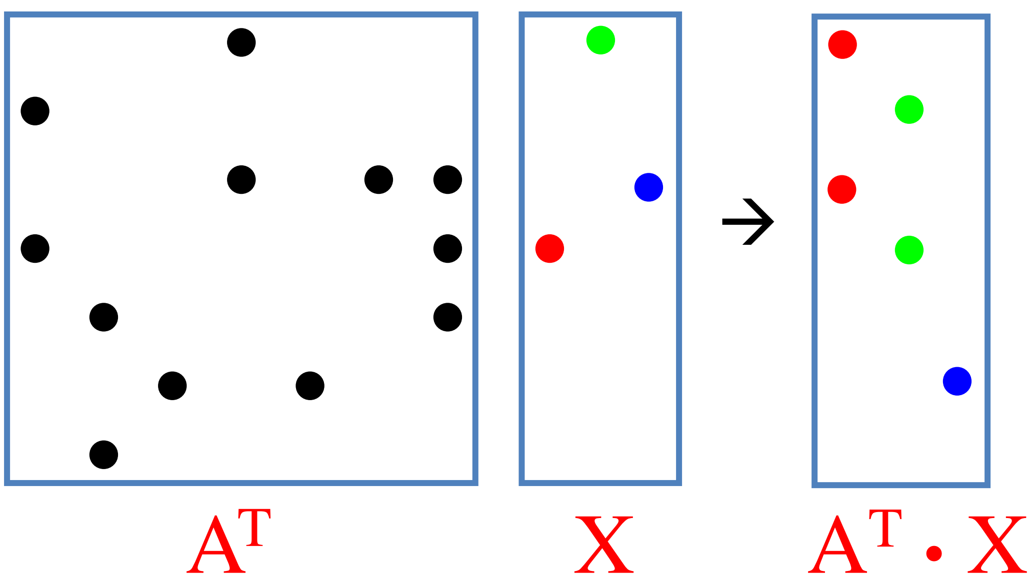
Ex: multi-source BFS, betweenness centrality, triangle counting[*], Markov clustering[*]

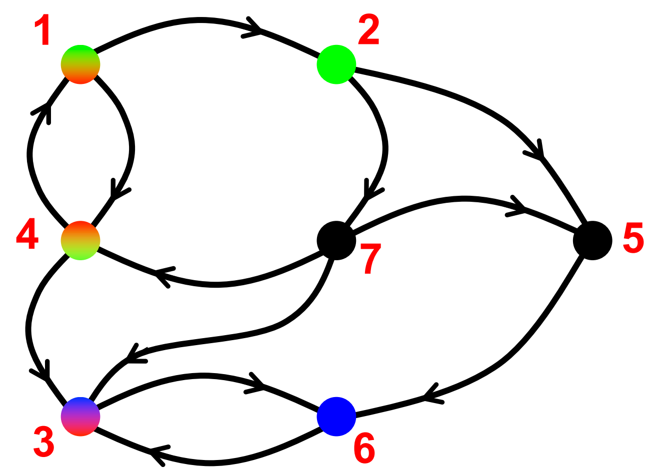
GrB_mxm(Y, P, <semiring>, A, X, <desc>)

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

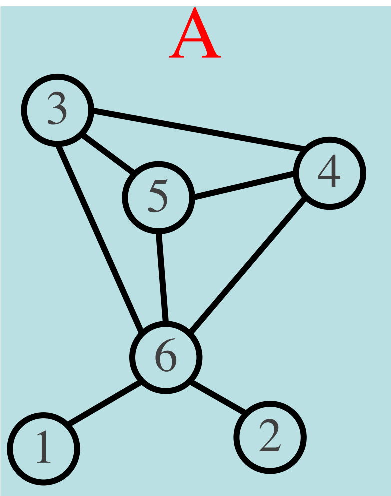




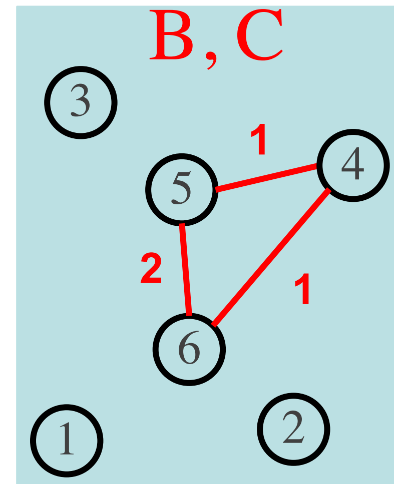
Pattern 2: Sparse matrix times sparse matrix (SpGEMM)

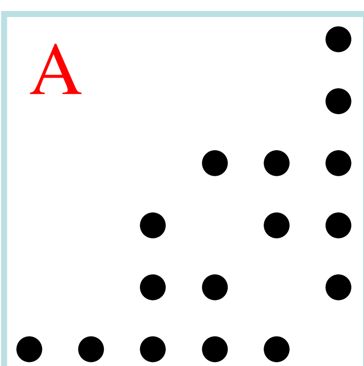
Triangle counting is also multi-source(in fact, all sources) traversal:
It just stops after one traversal iteration only, discovering all wedges

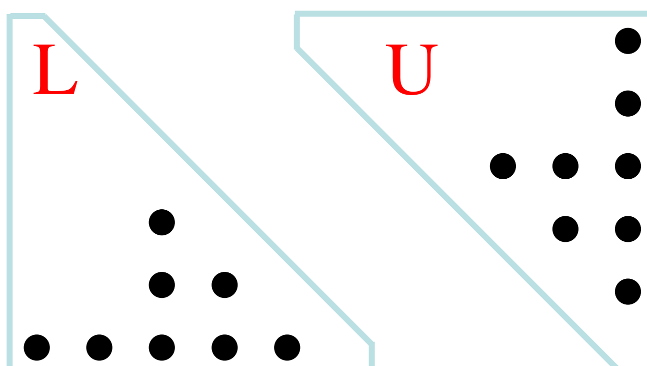
GrB_mxm(C, A, <semiring>, L, U, <desc>)

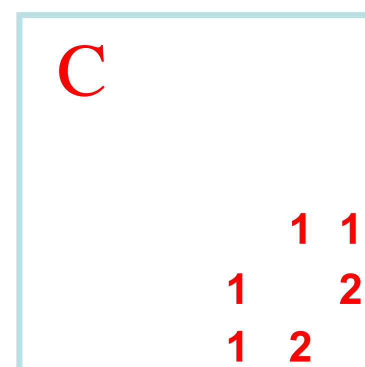


$A = L + U$	(hi->lo + lo->hi)
$L \times U = B$	(wedge, low hinge)
$A \wedge B = C$	(closed wedge)
sum(C)/2 =	**4 triangles**

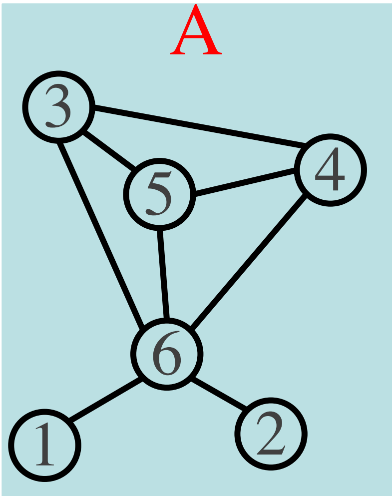








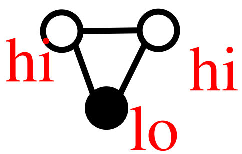
Counting triangles



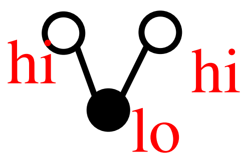
Clustering coefficient:

- 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

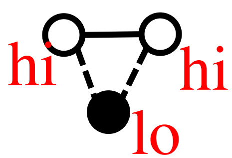
Cohen's algorithm to count triangles:



- 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

SpGEMM use case #3: Markov Clustering

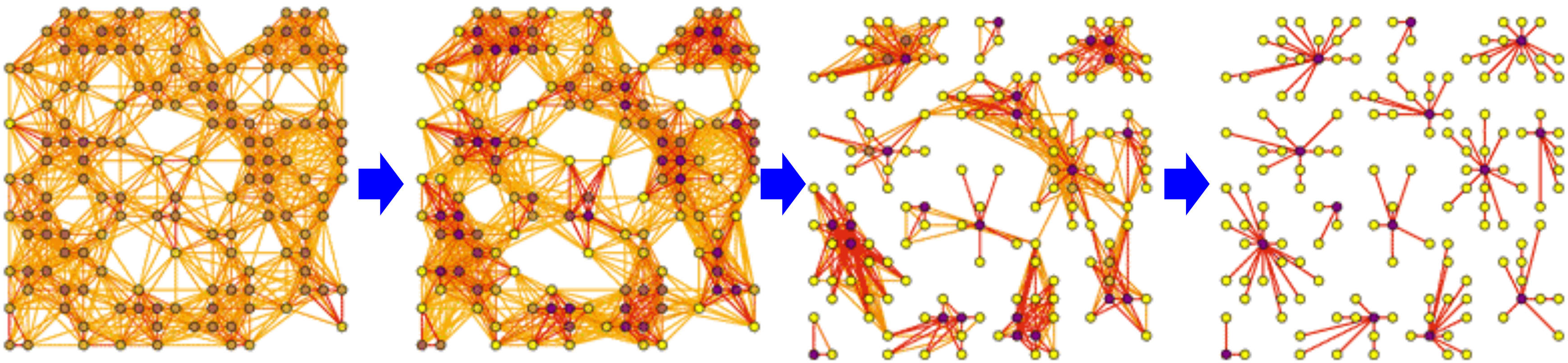
Markov clustering is also multi-source (in fact, all sources) traversal:

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



Initial network | Iteration 1 | Iteration 2 | Iteration 3

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

High-level outline

- Sparse matrices for graph algorithms
- **Sparse matrices for graph 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"

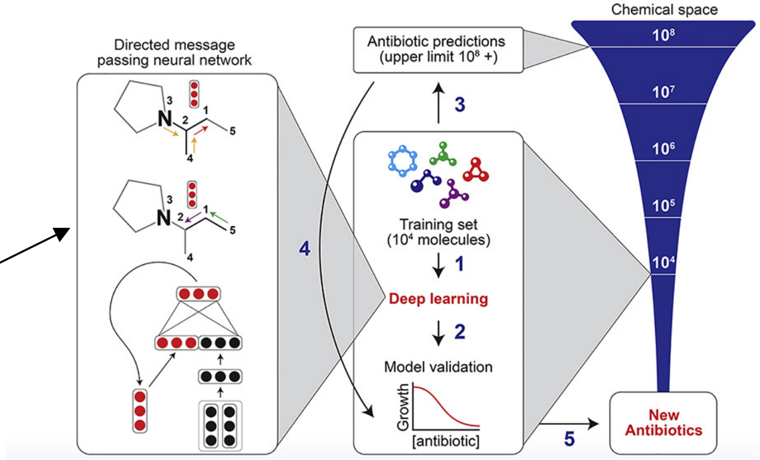
Bronstein, Michael M., et al. "Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges." (2021)

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



Article | Published: 09 June 2021

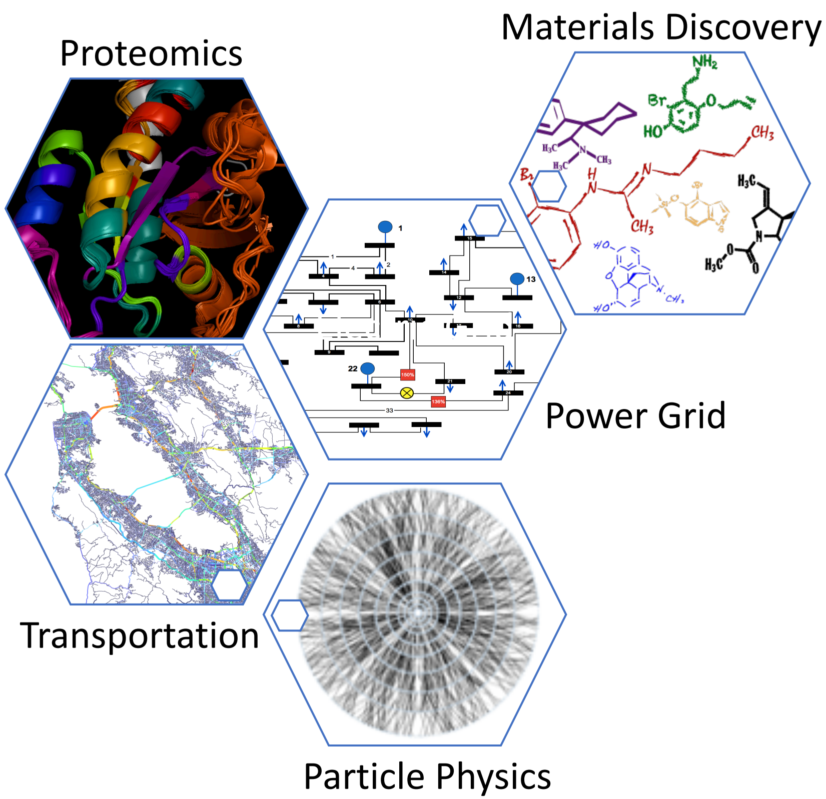
A graph placement methodology for fast chip design

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?

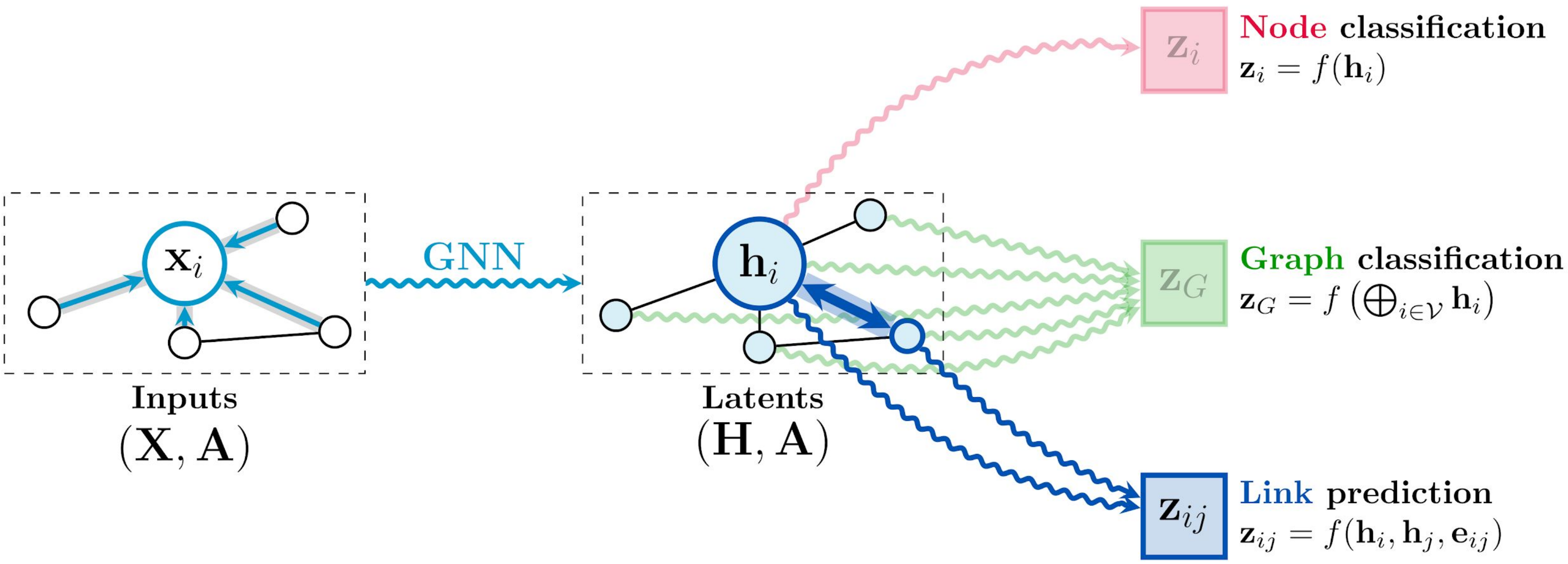


Figure source: Petar Veličković

Full-graph vs. mini-batch SGD

Full-graph training:

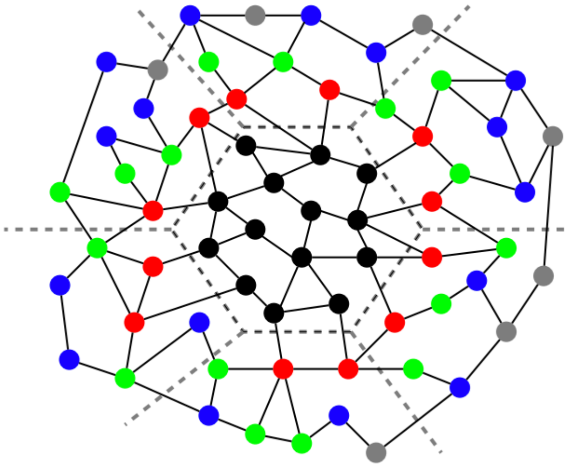
- Train on **entire** training set
- Slower convergence per epoch
- Faster training per epoch
- **Focus of this work**

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

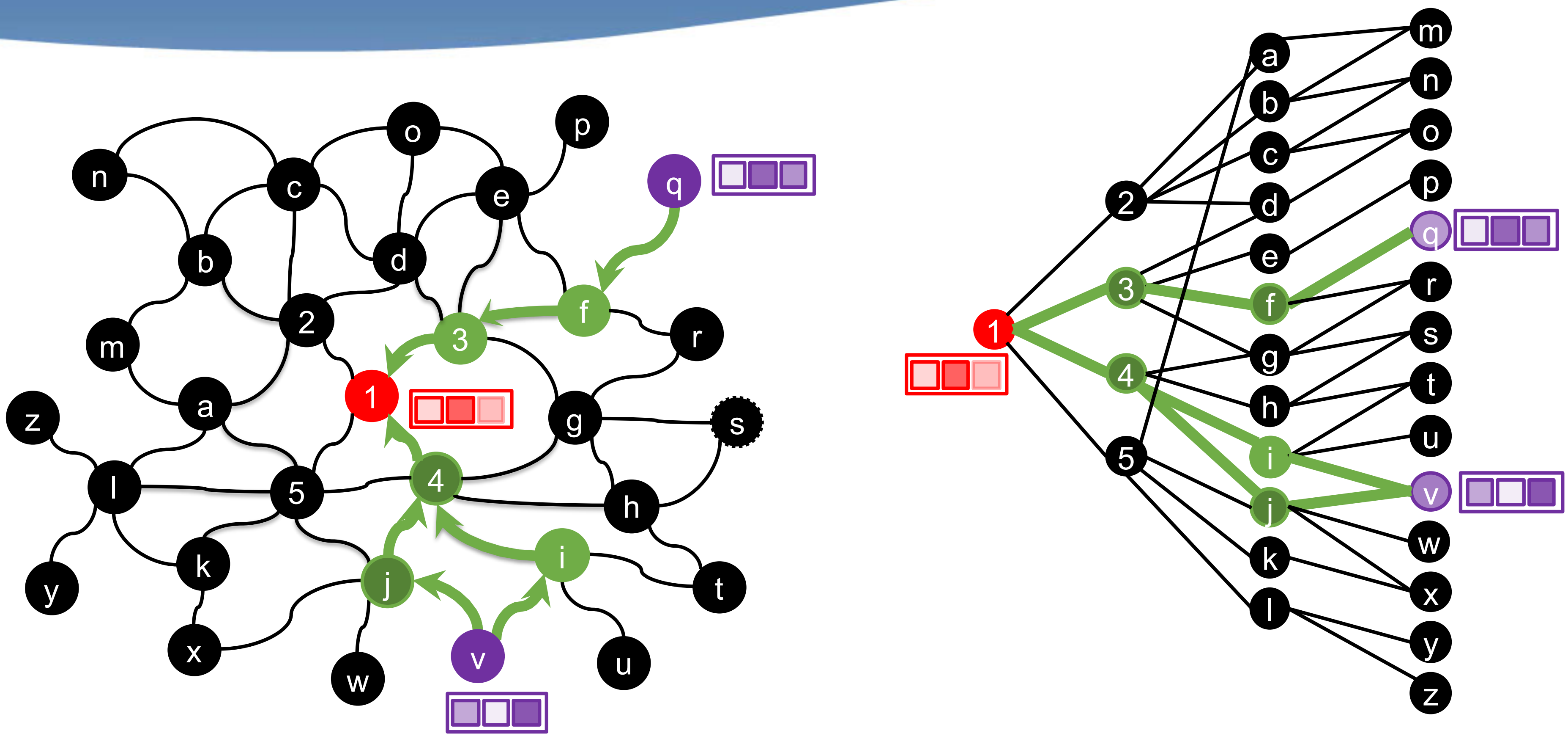
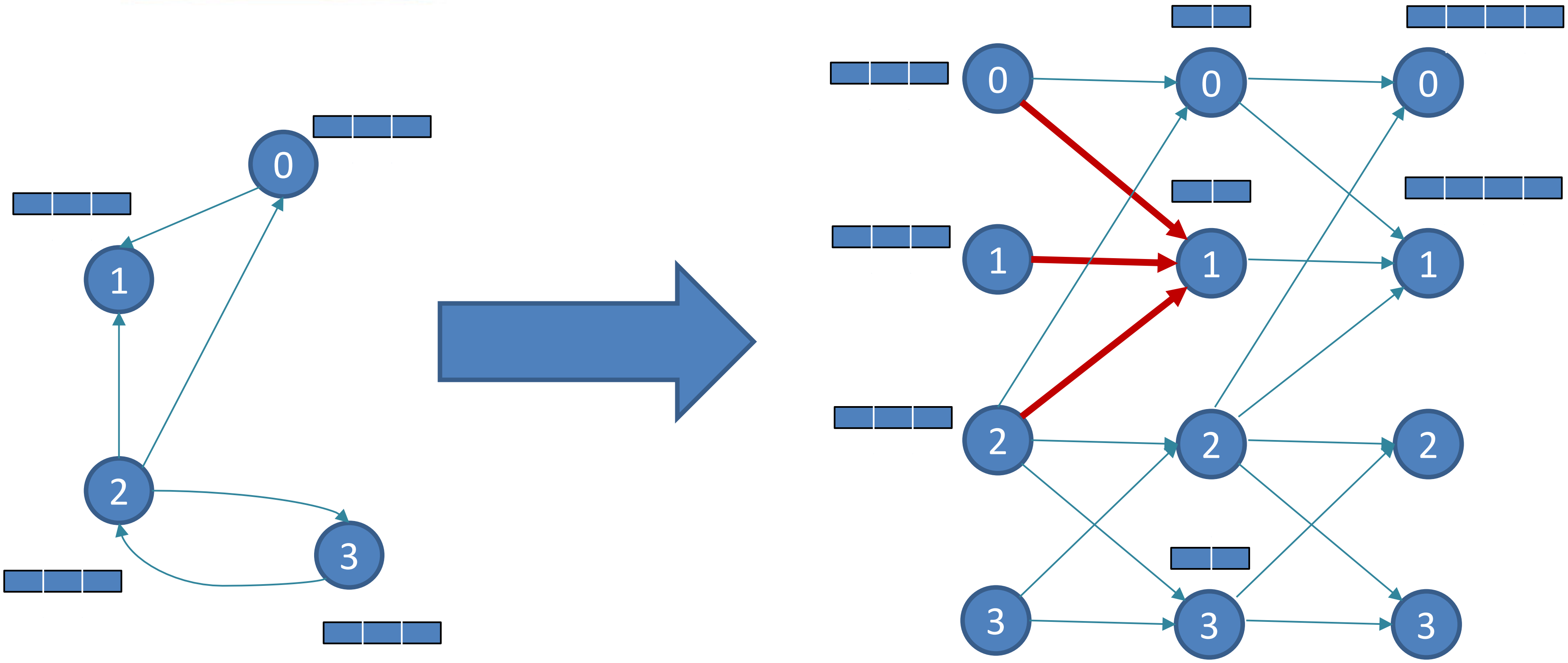


Illustration of the information flow in a Graph Neural Network (GNN). On the left is the graph in its natural form. The features (the shaded boxes) of vertices v and q are aggregated at vertex 1 through intermediate (green) vertices and edges. Features of other nodes are not shown but are also propagated. 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

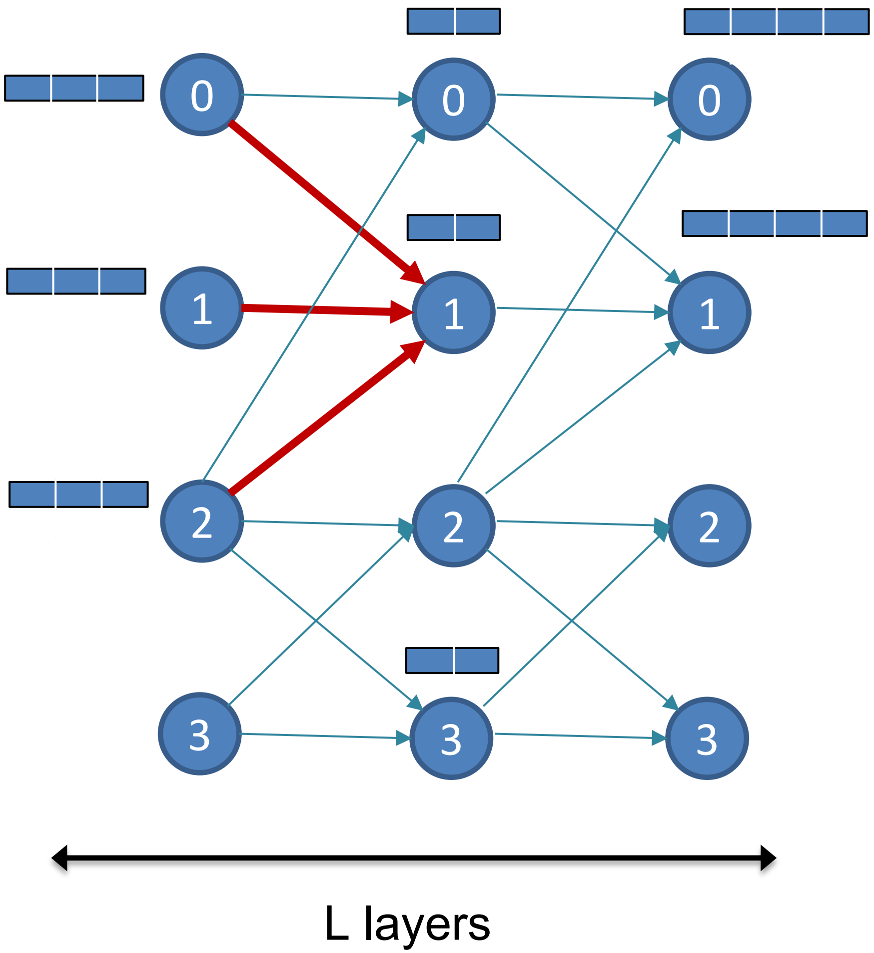


Input Graph

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
        Zˡ = Aᵀ * Hˡ⁻¹ * Wˡ
        Hˡ = σ(Zˡ)
    ...
    for l = L-1 … 1
        Gˡ = A * Gˡ⁺¹ * (Wˡ⁺¹)ᵀ ⊙ σ’(Zˡ)
       dH/dW = (Hˡ⁻¹)ᵀ * A * Gˡ
```

$A \in n\ x\ n$

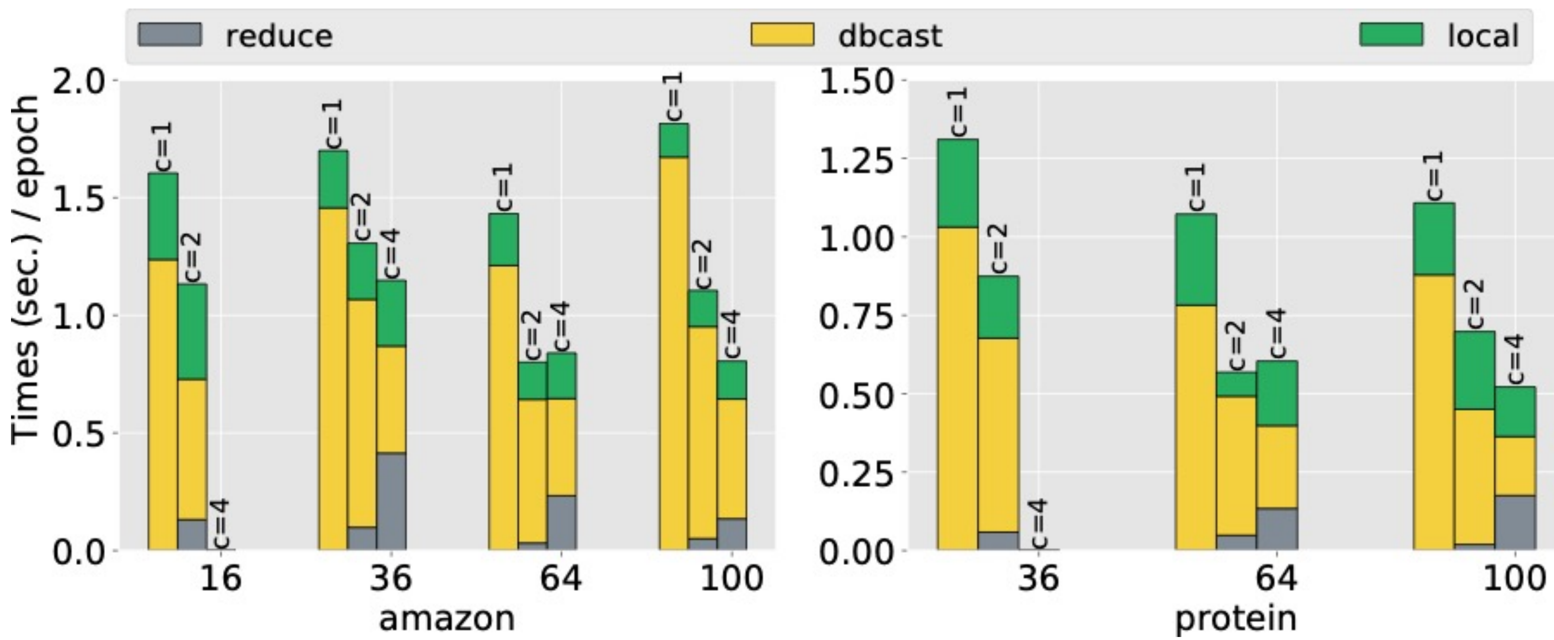
$H^l \in n\ x\ f^l$

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

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

- 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

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

Feature aggregation from neighbors:

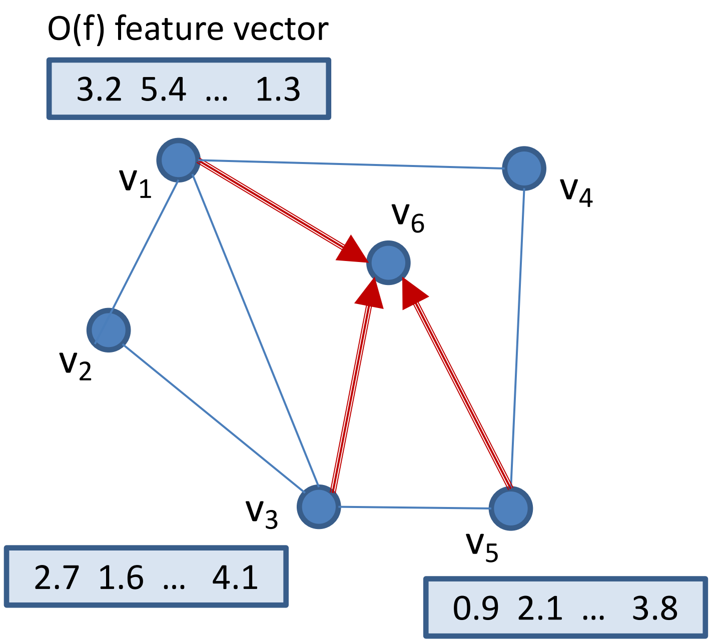
Used in Graph neural networks, graph embedding, etc.

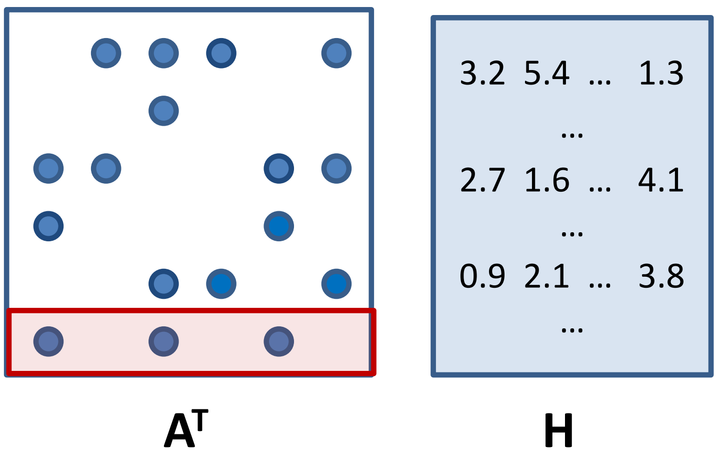
GrB_mxm(W, GrB_NULL, <semiring>, A, H, <desc>)

A: sparse adjacency matrix, n-by-n

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

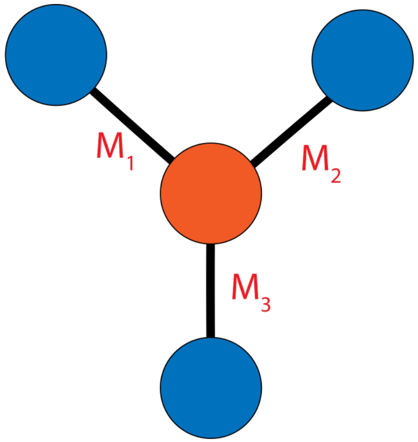
W: output dense matrix, new features





[More] Sparse Kernels in Graph Learning

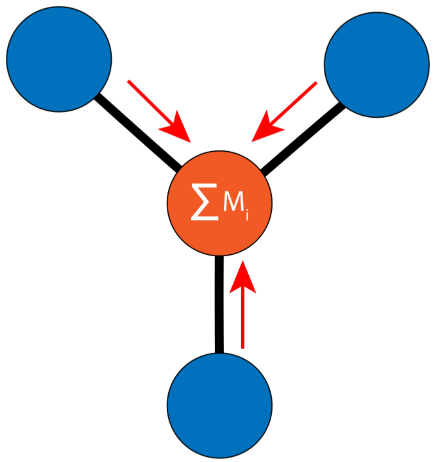
- Sampled Dense-Dense Matrix Multiplication (SDDMM) and Sparse-times-Dense Matrix Multiplication (SpMM) appear in a variety of applications:
  - Graph Neural Networks with Self-Attention
  - Collaborative Filtering with Alternating Least Squares
  - Document Clustering by Wordmover's Distance



Message Generation

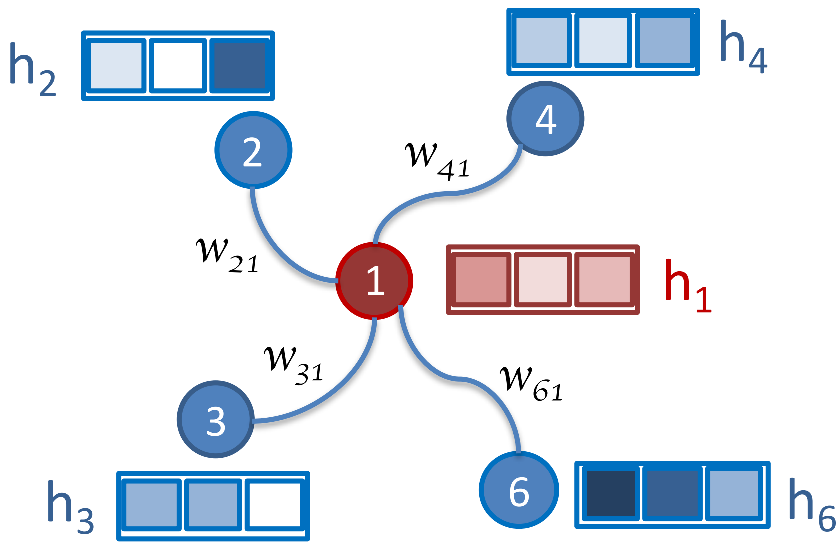
- Both kernels involve a single sparse matrix and two (typically tall-skinny) dense matrices. Typically, applications use both operations in sequence.

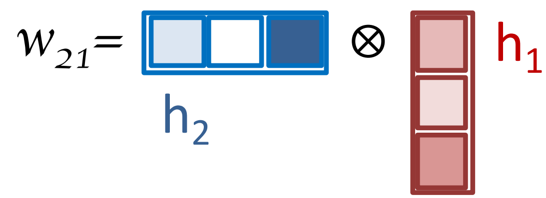
- When the sparse matrix is the adjacency matrix of a graph, we interpret the kernels as follows:
  - SDDMM generates a message on each edge
  - SpMM aggregates messages from incident edges



Message Aggregation

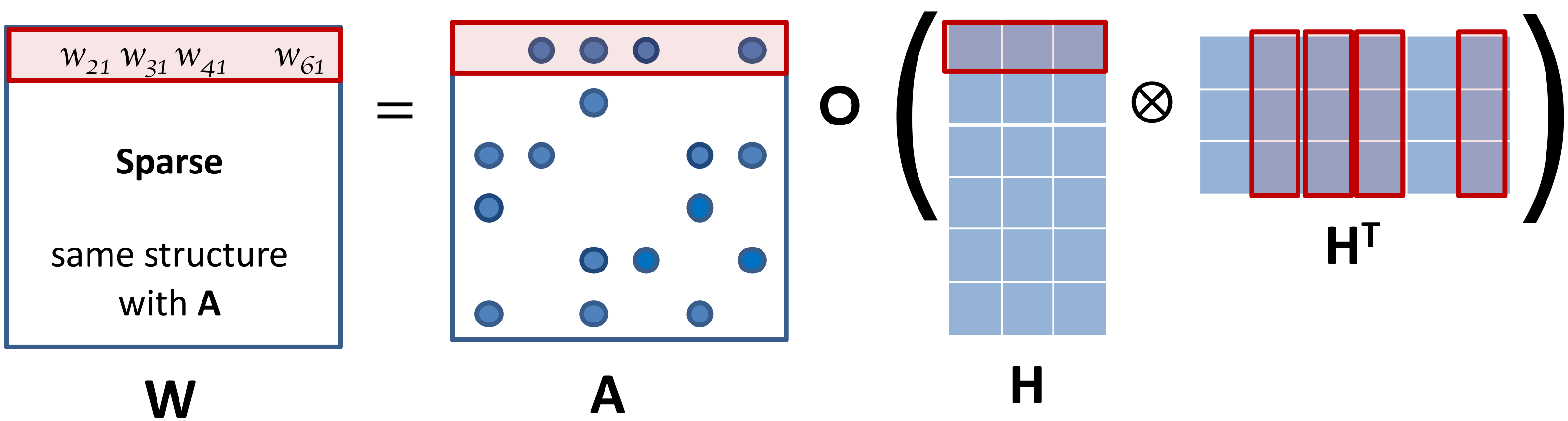
Graph attention: making edge weights learnable





SDDMM: Sampled dense-dense matrix multiplication

GrB_mxm(W, A, H, H, …);



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

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

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

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)

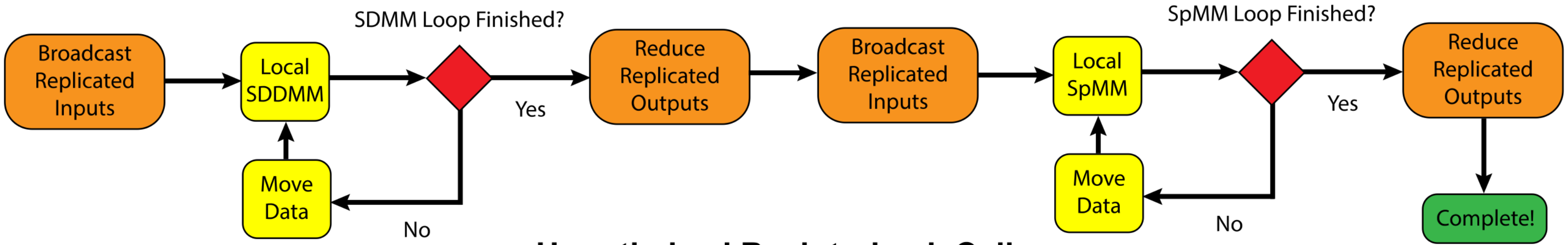
- Co-locate S_{ij}, A_{ik}, B_{jk} on a single processor
- Perform the update $A_{ik} \mathrel{+}= S_{ij} B_{jk}$

Transform this algorithm as follows:

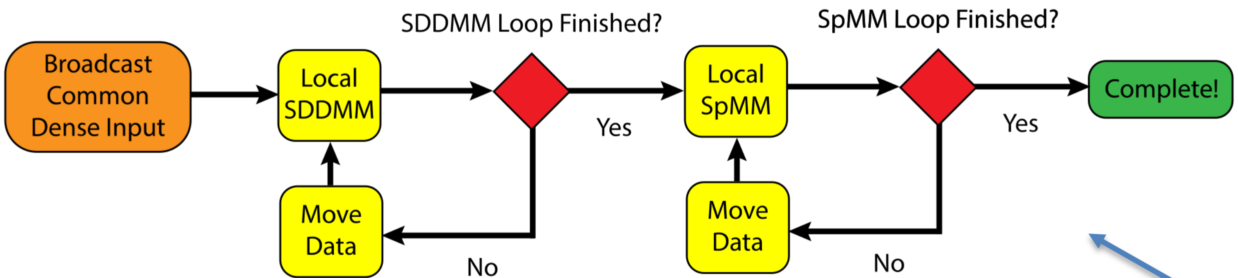
1. Change the input sparse matrix S to an output that is initialized to 0.
2. Change A from an output to an input.
3. Have each processor execute the local update: $S_{ij} \mathrel{+}= A_{ik} B_{jk}$

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

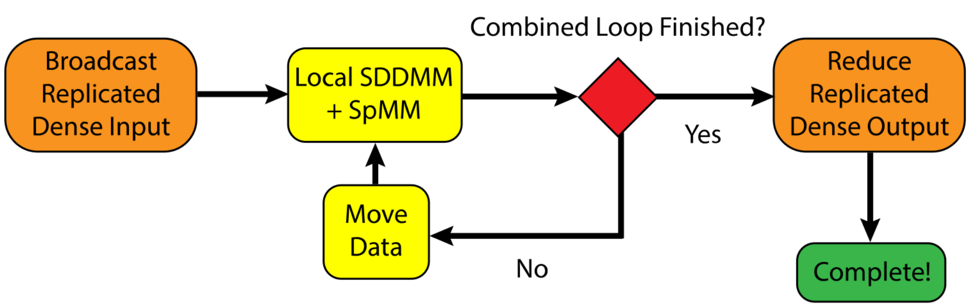


Unoptimized Back-to-back Calls



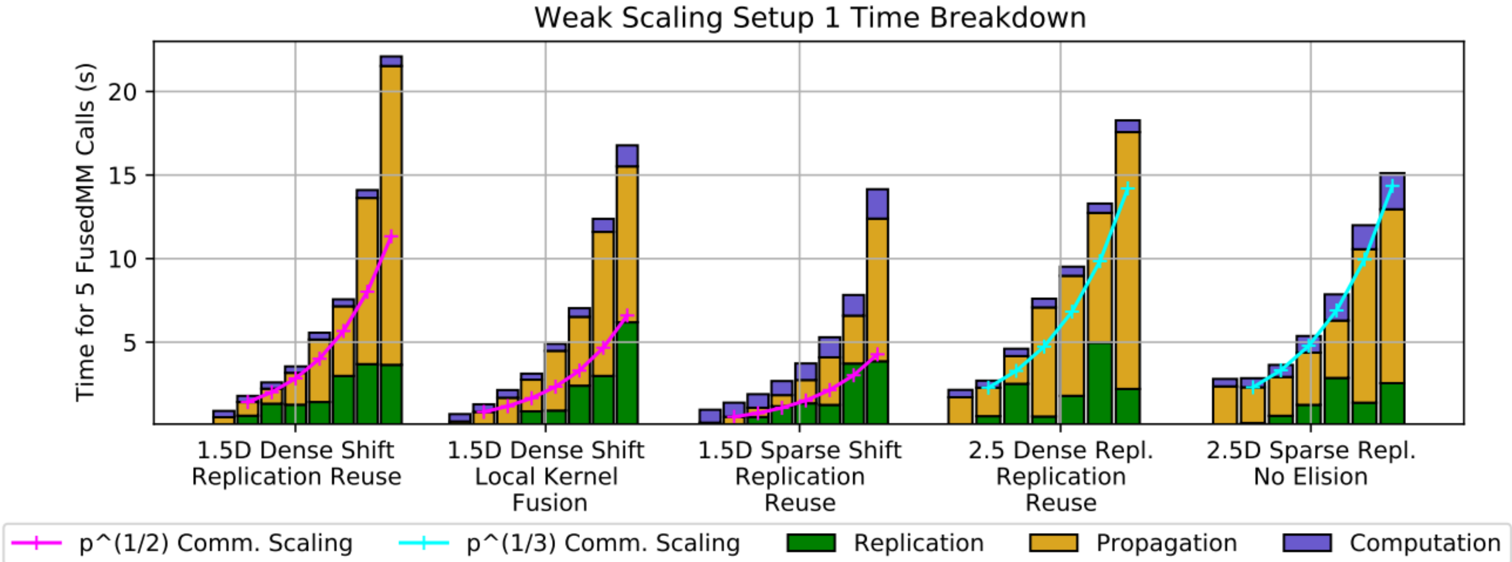
Replication Reuse

Mutually exclusive optimizations



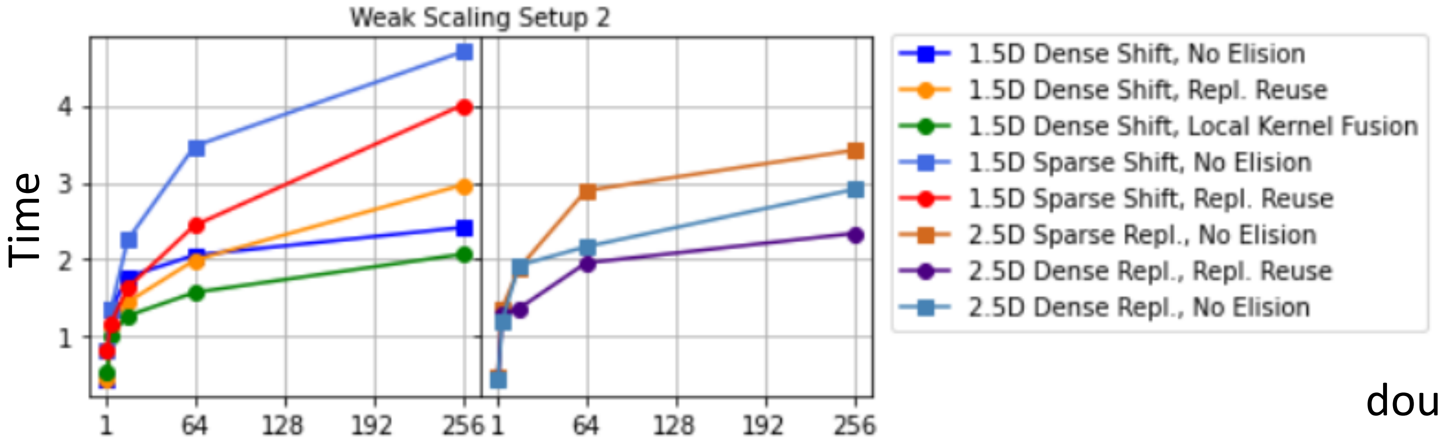
Local Kernel Fusion

Distributed FusedMM performance



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

remains constant



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

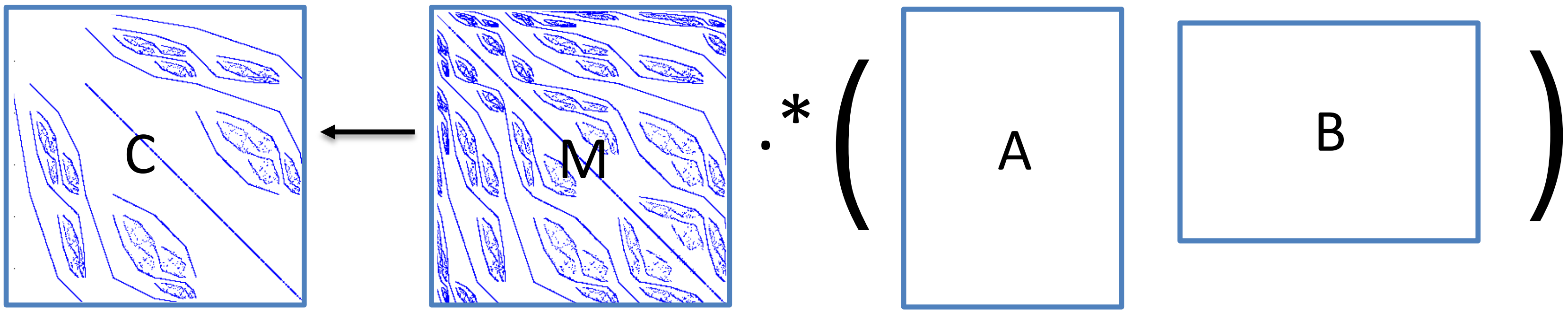
doubles at each process count quadrupling

High-level outline

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

Sparse matrix-matrix multiplication

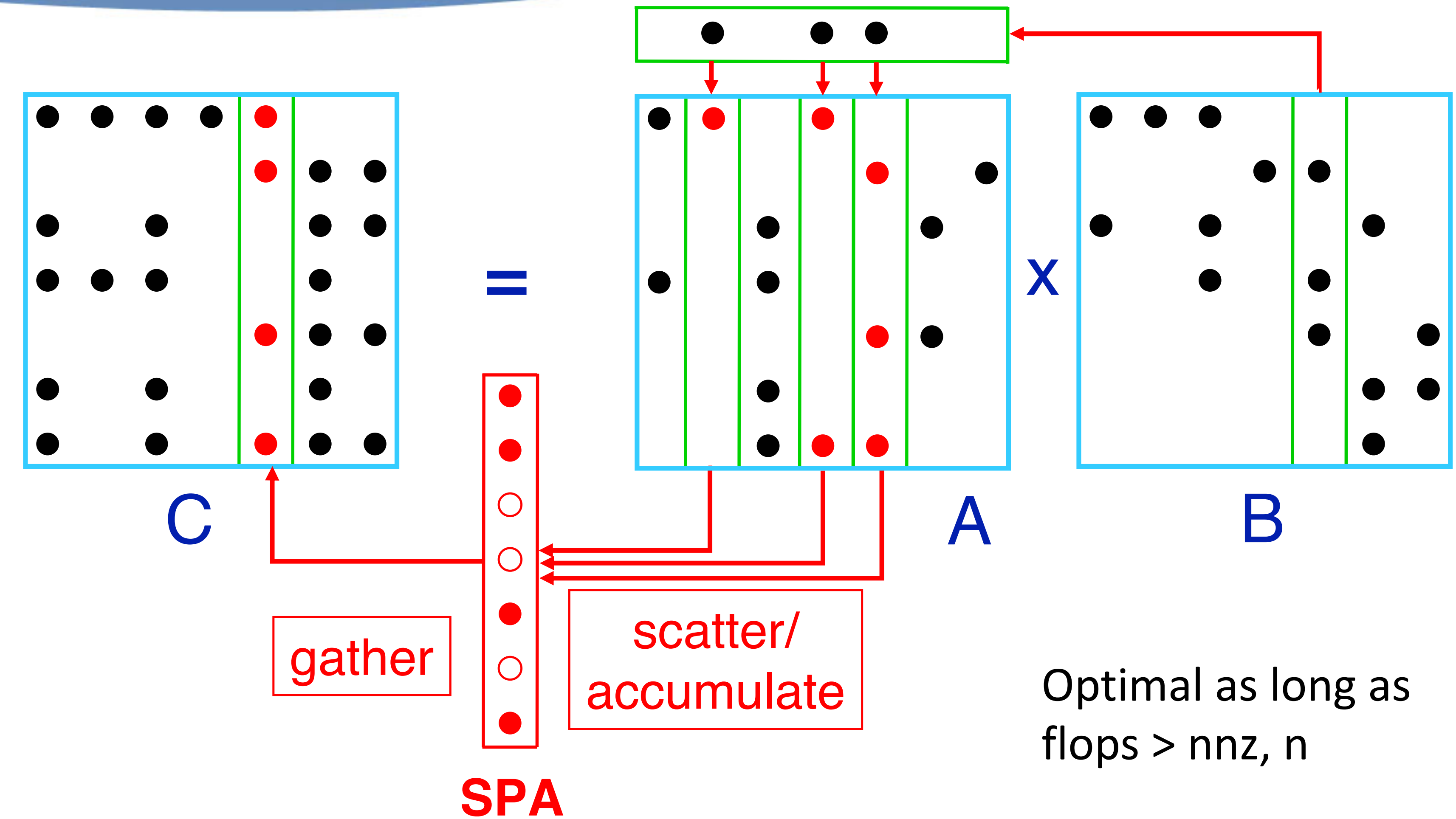
$$C(\neg M) \oplus = A^T \oplus . \otimes B^T$$



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

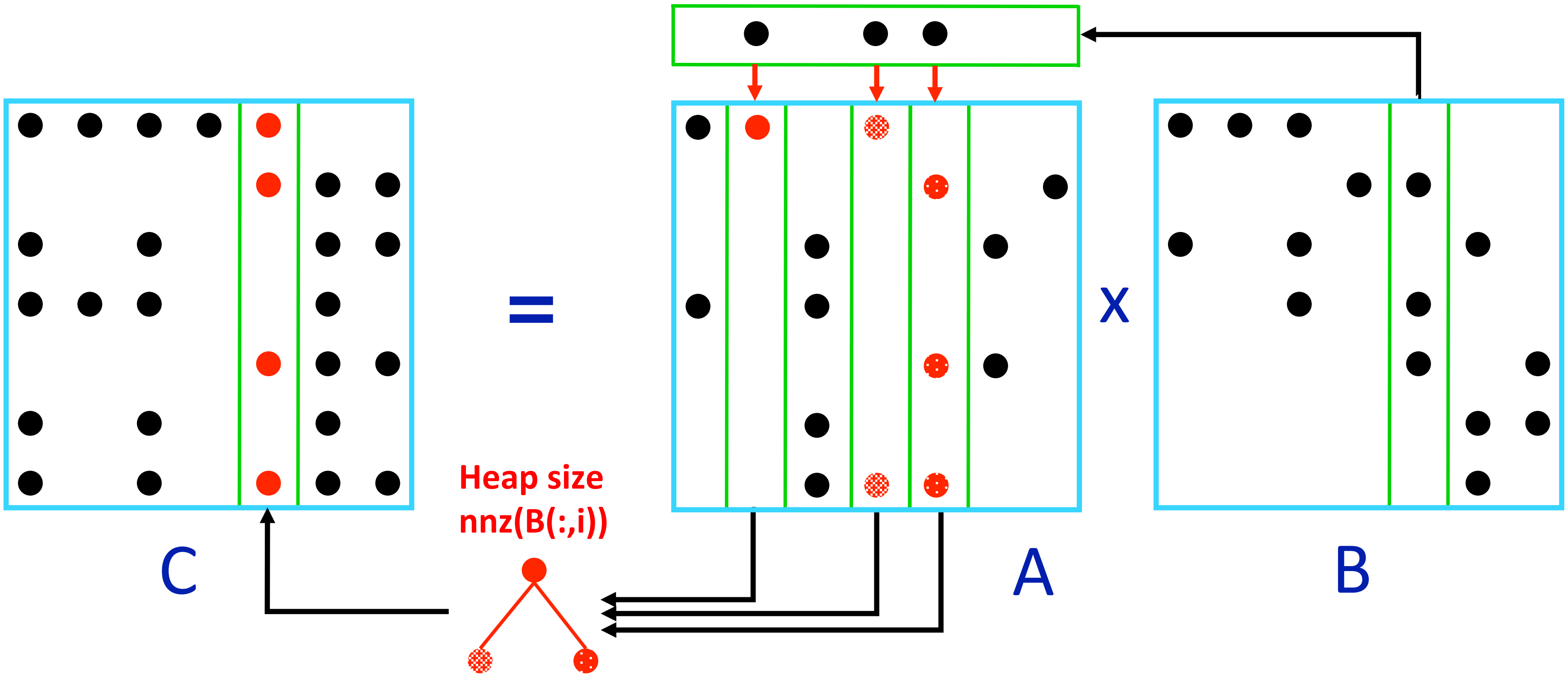
Basic serial SpGEMM (Gustavson, 1978)



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)



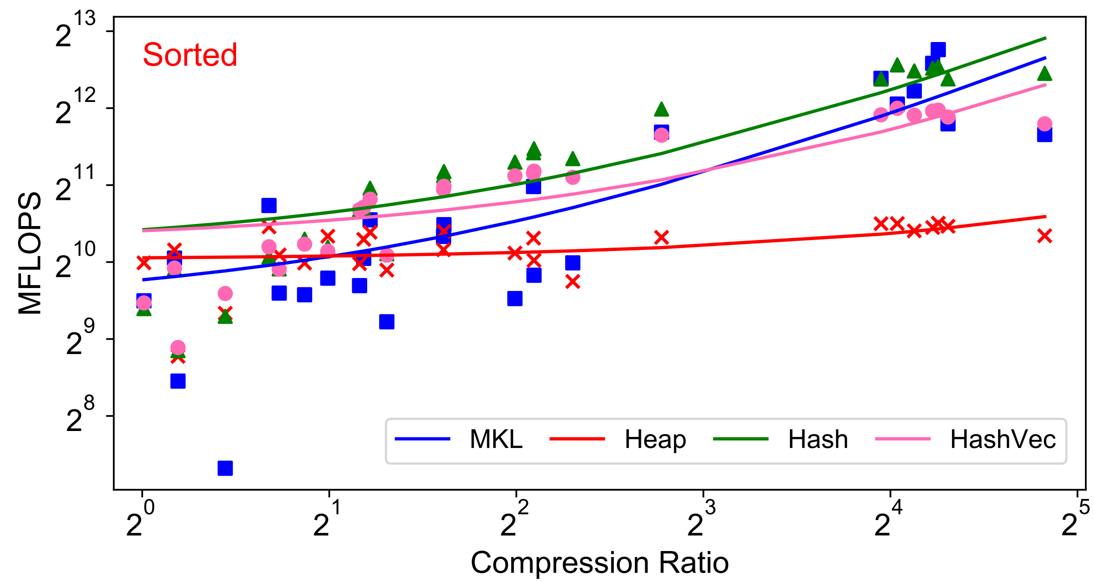
- Implemented in CombBLAS and SparseSuite:GraphBLAS
- Memory efficient and suitable for multithreading
- Not great for high compression ratio cases (more later)

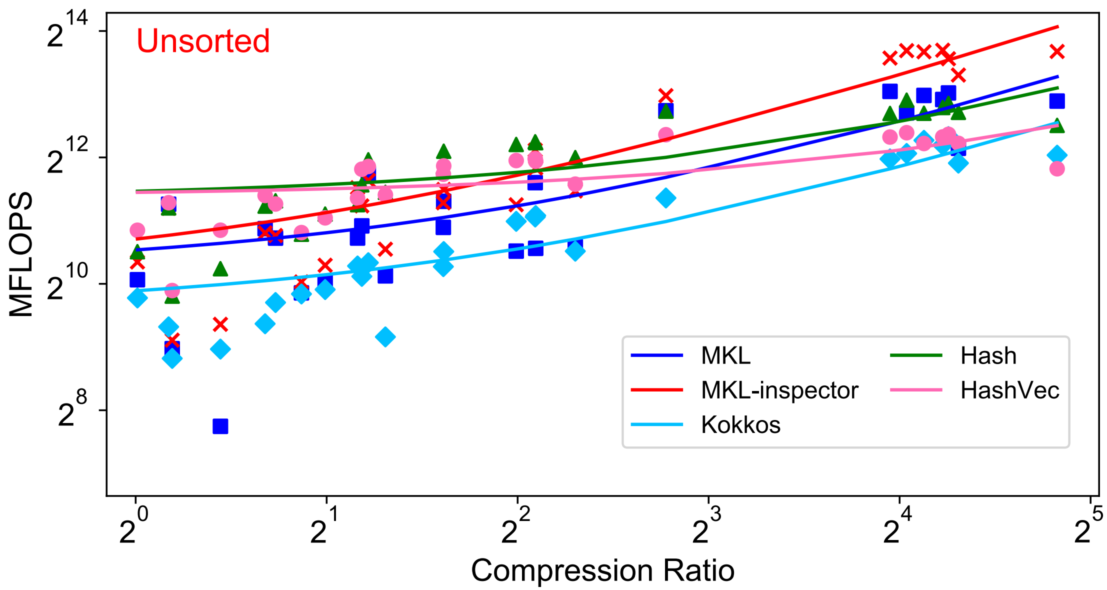
New shared-memory SpGEMM kernels

- 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



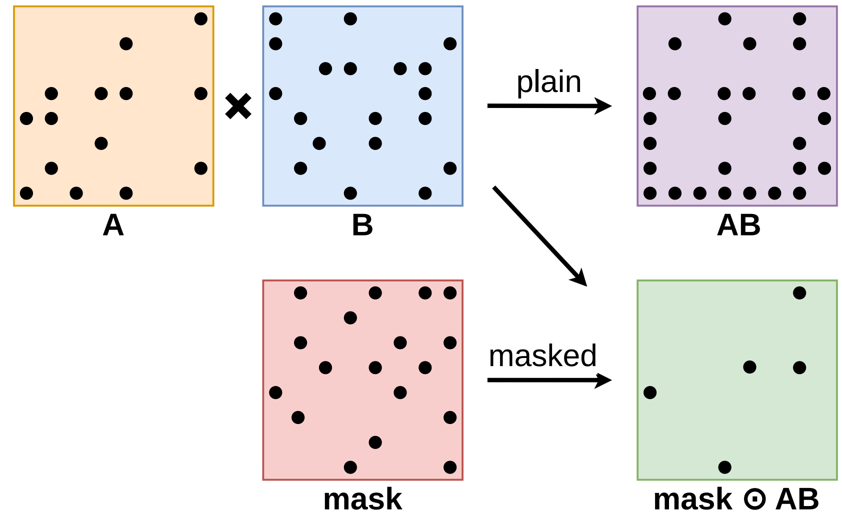


- We integrated hash algorithms to CombBLAS and HipMCL

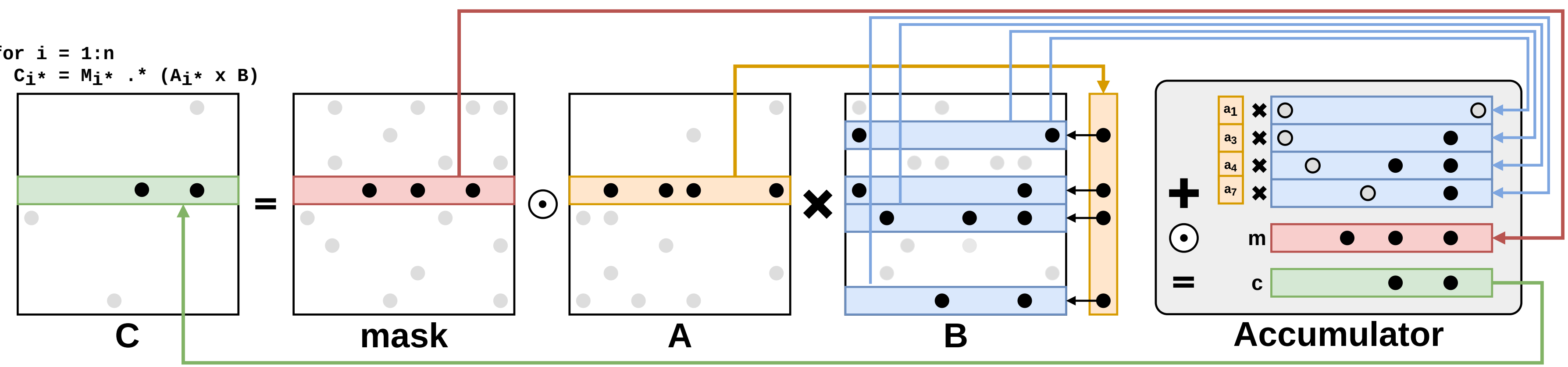
Yusuke Nagasaka, Satoshi Matsuoka, Ariful Azad, and Aydin Buluc. Performance optimization, modeling and analysis of sparse matrix-matrix products on multi-core and many-core processors. Parallel Computing, 90:102545, 2019.

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)
```

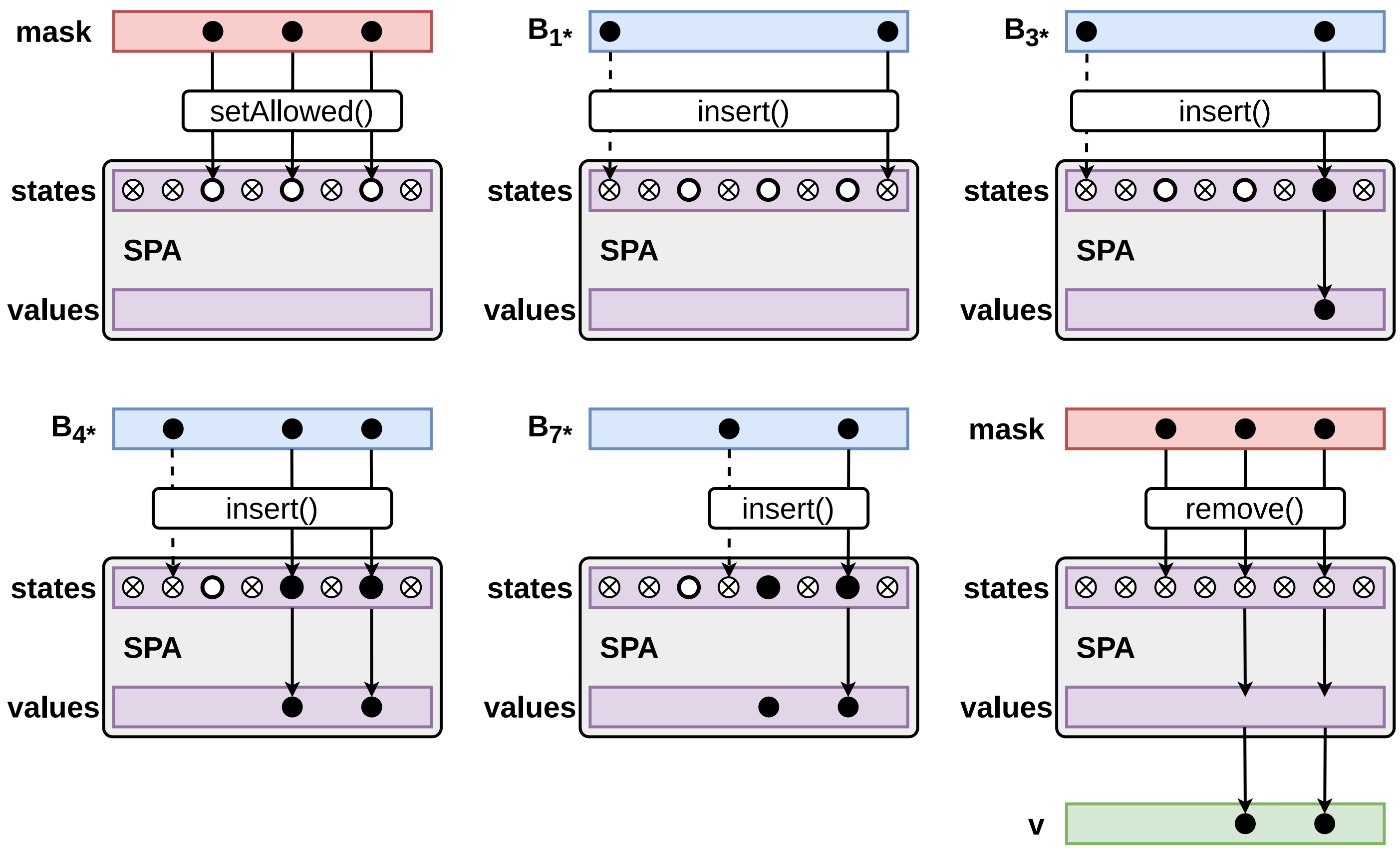


- Row-wise Masked SpGEMM using an accumulator to compute output row C_{i*}.
- The rows corresponding to the column indices of entries in row A_{i*} are merged and filtered through the respective mask entries to compute C_{i*}.
- This merging and filtering process can be performed in a number of ways.

Masked Sparse Accumulator (MSA)

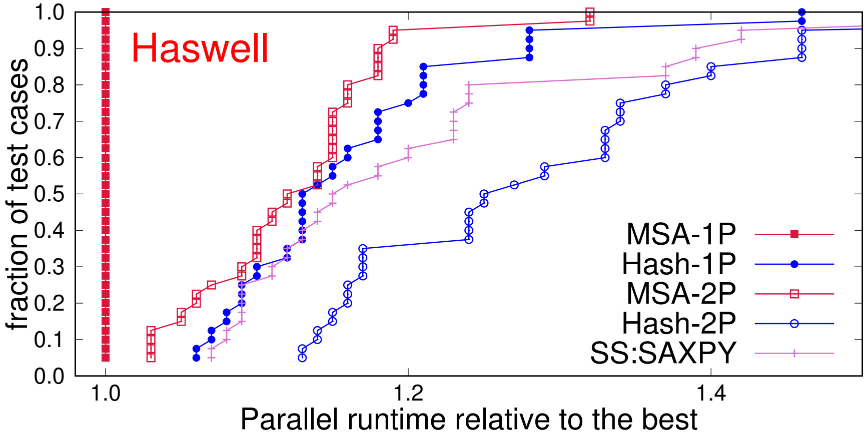
Execution of 1 row of SpGEMM with Masked Sparse Accumulator (MSA)

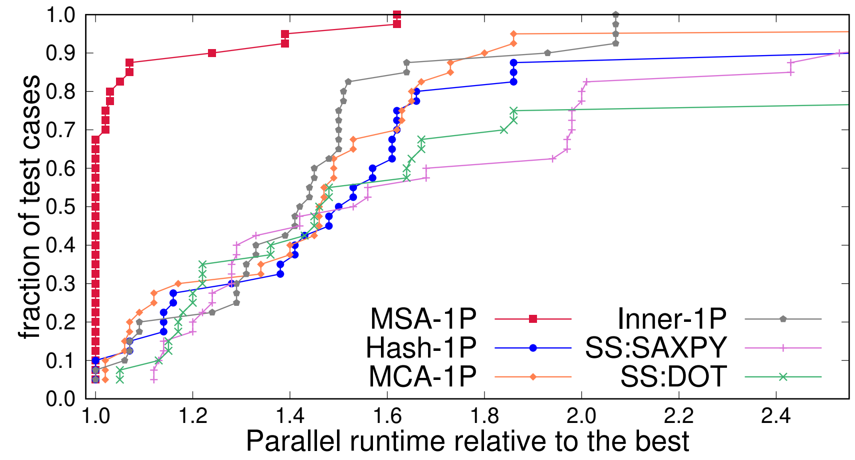
(a) initialize (b) MSA+=u_1 B_{1*} (c) MSA+=u_3 B_{3*} (d) MSA+=u_4 ×B_{4*} (e) MSA+=u_7 ×B_{7*} (f) output

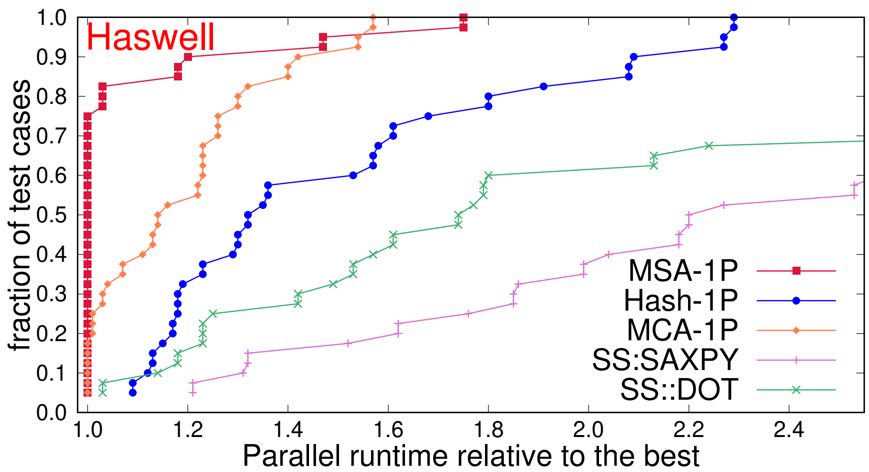


Srdjan Milaković, Oguz Selvitopi, Israt Nisa, Zoran Budimlić, and Aydın Buluç. Parallel algorithms for masked sparse matrix-matrix products. *arXiv preprint arXiv:2111.09947*, 2021 (Poster at PPOPP’22, paper at ICPP’22)

Performance of Masked SpGEMM algorithms







Top (left): Betweenness Centrality

Top (right): k-truss

Bottom: Triangle counting

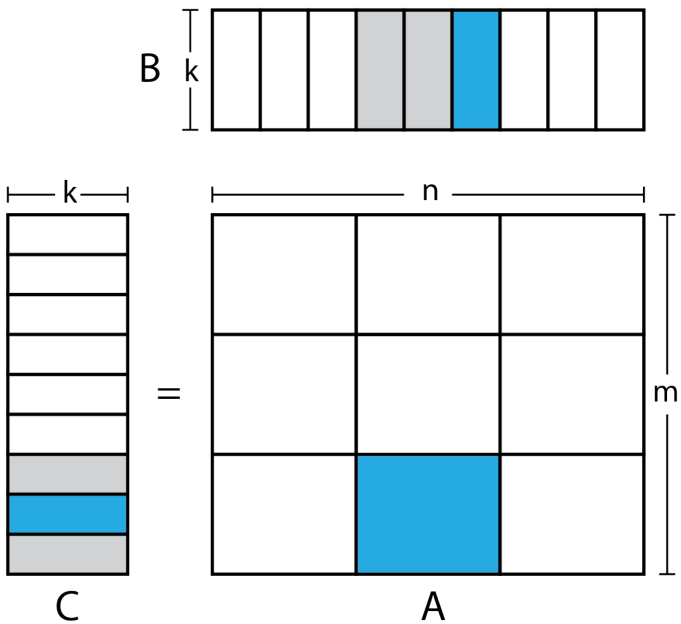
SS is the Suitesparse:GraphBLAS

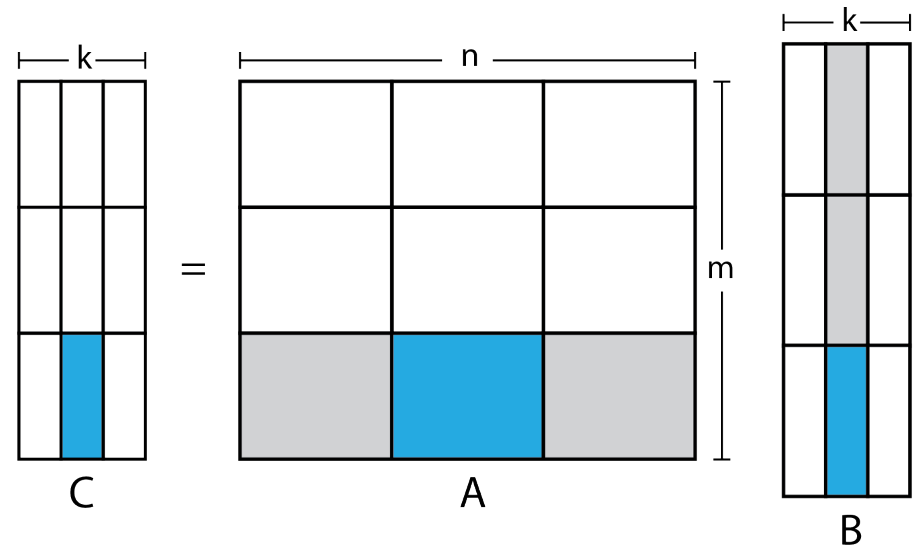
SS:DOT and Inner-1P do sparse dot products

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, paper at ICPP'22)

Distributed SpMM algorithms

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





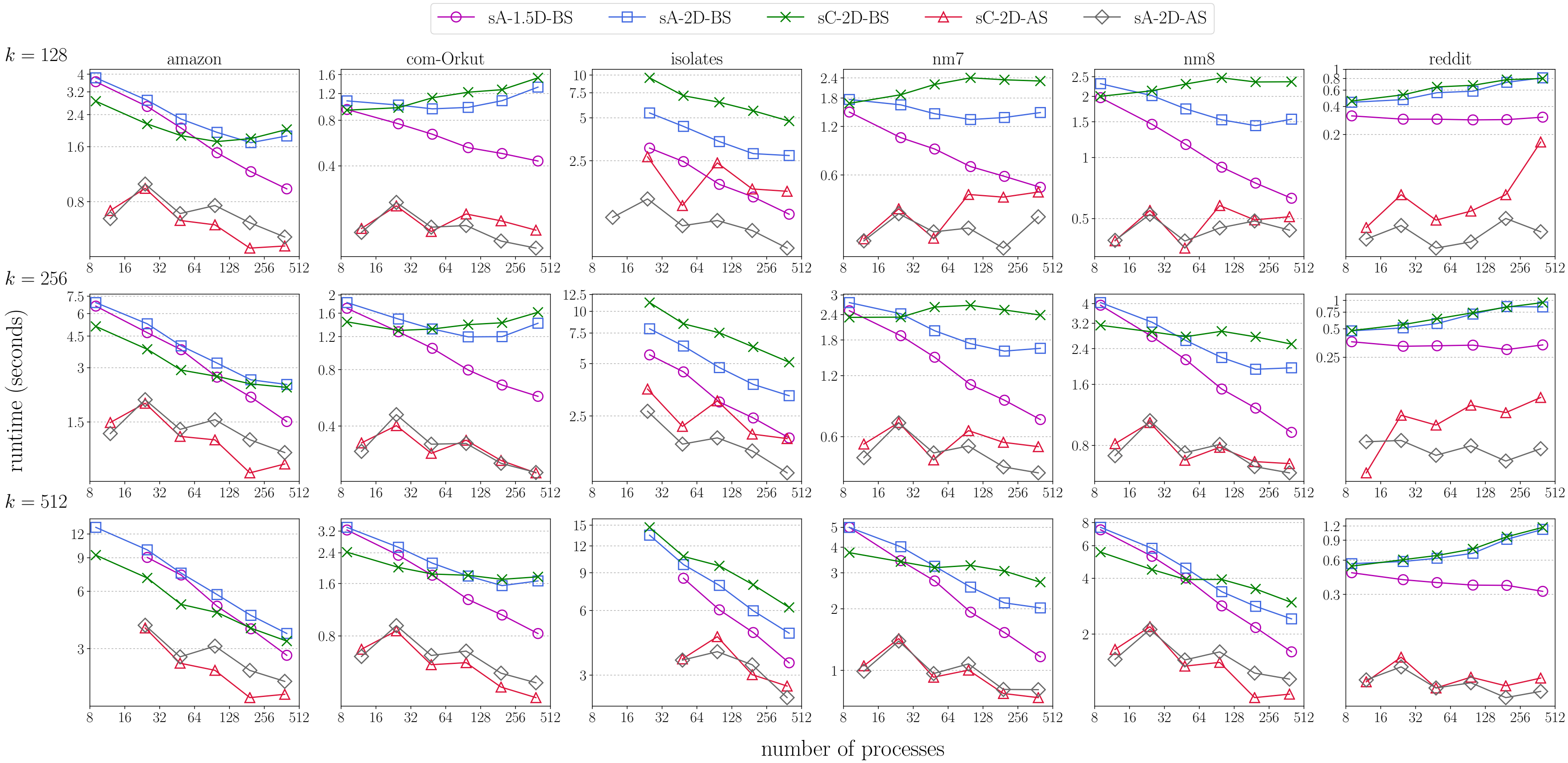
- Stationary A, 1.5D algorithm
- **A** is split on a p/c-by-c grid

- 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

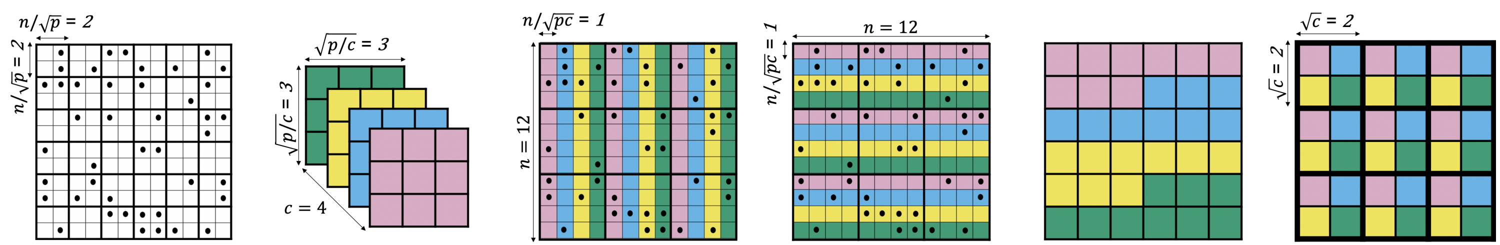
- Sparse matrices for graph algorithms
- Sparse matrices for graph learning
- Parallel algorithms for sparse matrix primitives
- **Available software**

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

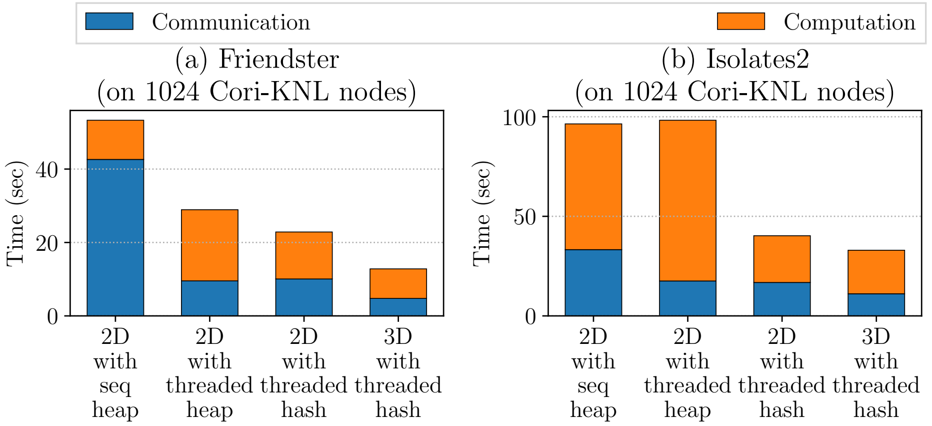
(d) Partitioning **B** into the 3D grid by splitting up the rows

(e) Converting a $6{\times}6$ grid to a $4{\times}3{\times}3$ grid in the regular way

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

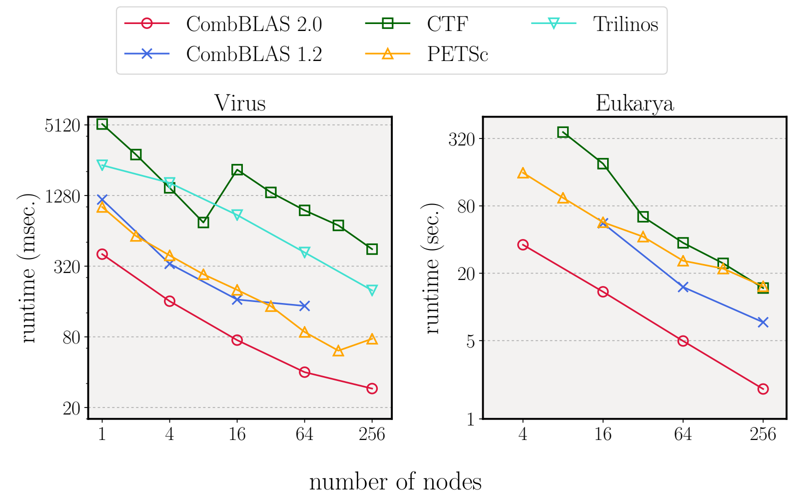
- communication avoiding algorithms,
- hierarchical parallelism via in-node multithreading,
- accelerator support via GPU kernels,
- generalized semiring support,
- implementations of key data structures and functions,
- scalable distributed I/O operations for human-readable files

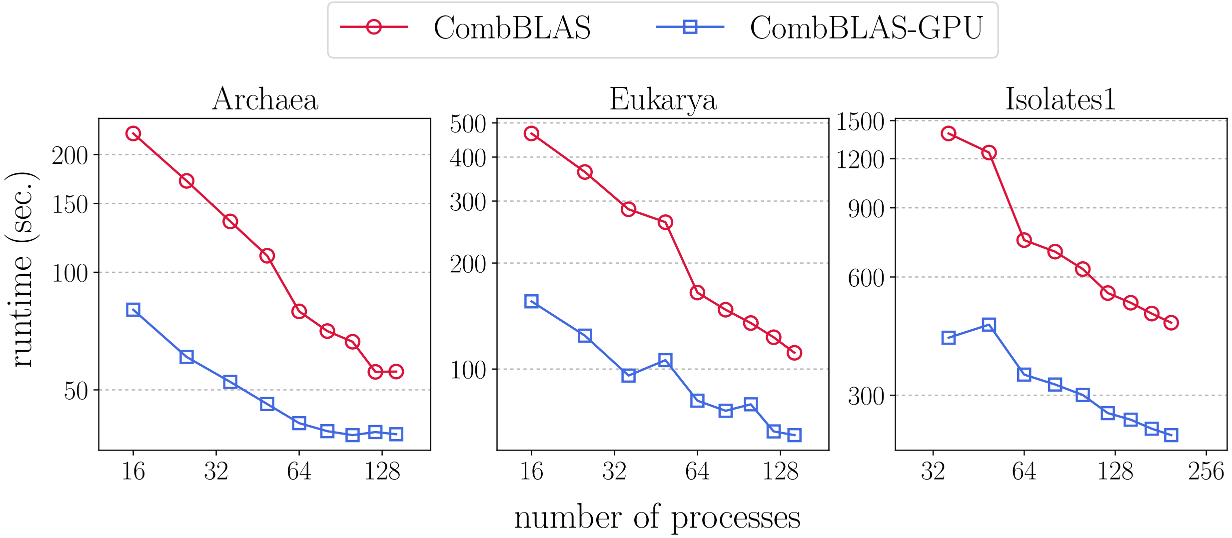
Combinatorial BLAS 2.0 performance



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:

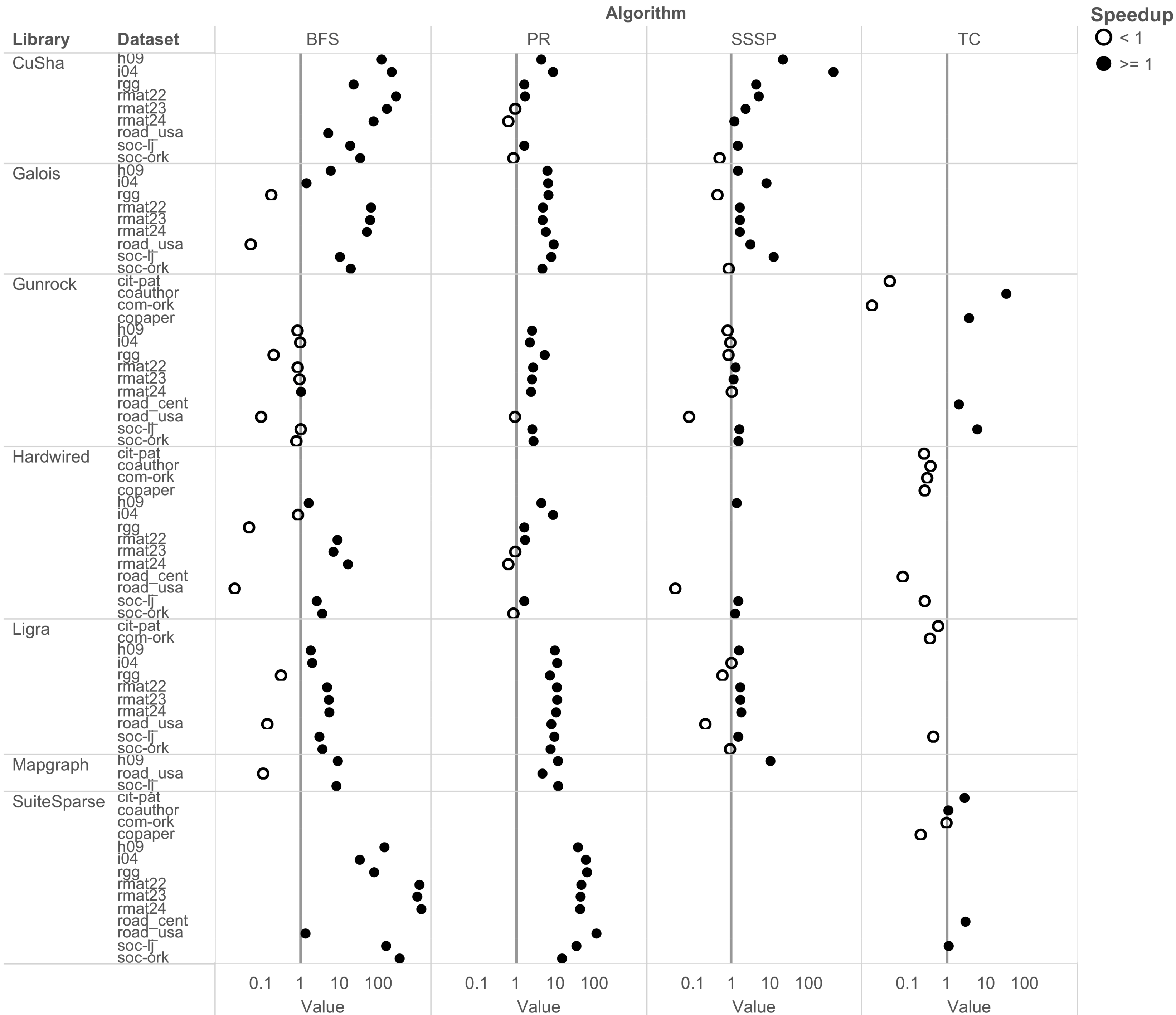
1. Exploit input sparsity => direction-optimization
2. Exploit output sparsity => masking
3. Proper load-balancing => key for GPU implementations

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
Galois
Gunrock
Hardwired
Ligra
Mapgraph
SuiteSparse
h09
i04
rgg
rmat22
rmat23
rmat24
road_usa
soc-lj
soc-ork
cit-pat
coauthor
com-ork
copaper
road_cent
0.1
1
10
100
Value

Conclusions

- Sparse matrix techniques underlie computations from disparate fields:
  a. Scientific computing
  b. Graph learning
  c. Graph algorithms
  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