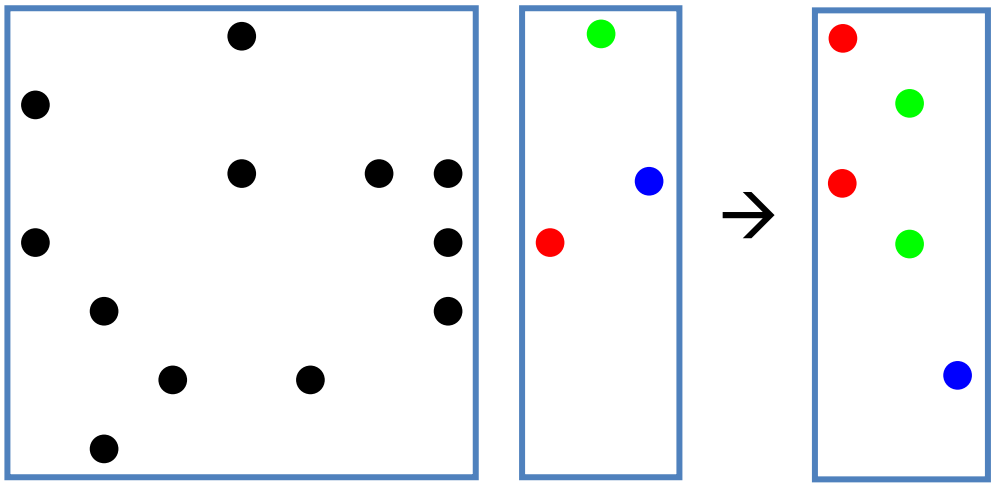
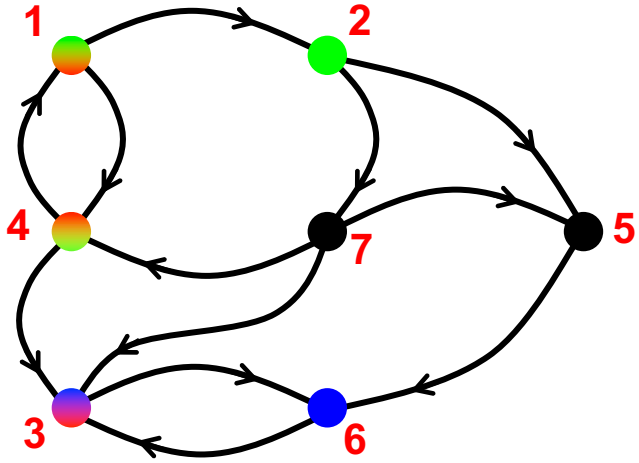
GraphBLAS in Data Analytics and Machine Learning

Aydın Buluç
Computational Research Division, LBNL
EECS Department, UC Berkeley

ACS Reverse Site Visit
March 25, 2019

Graphs in the language of matrices

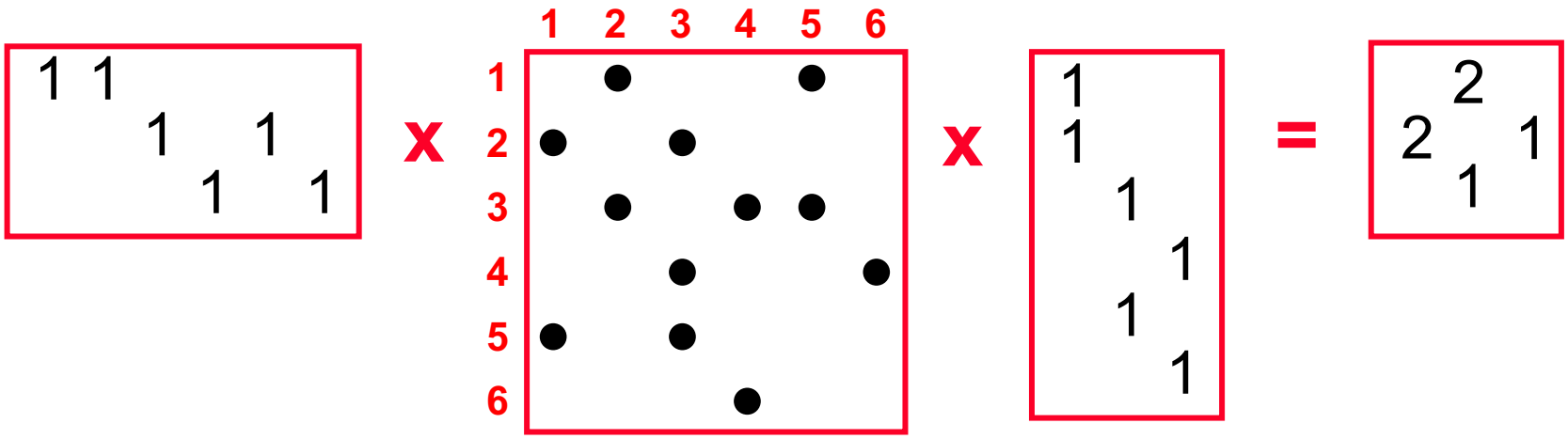


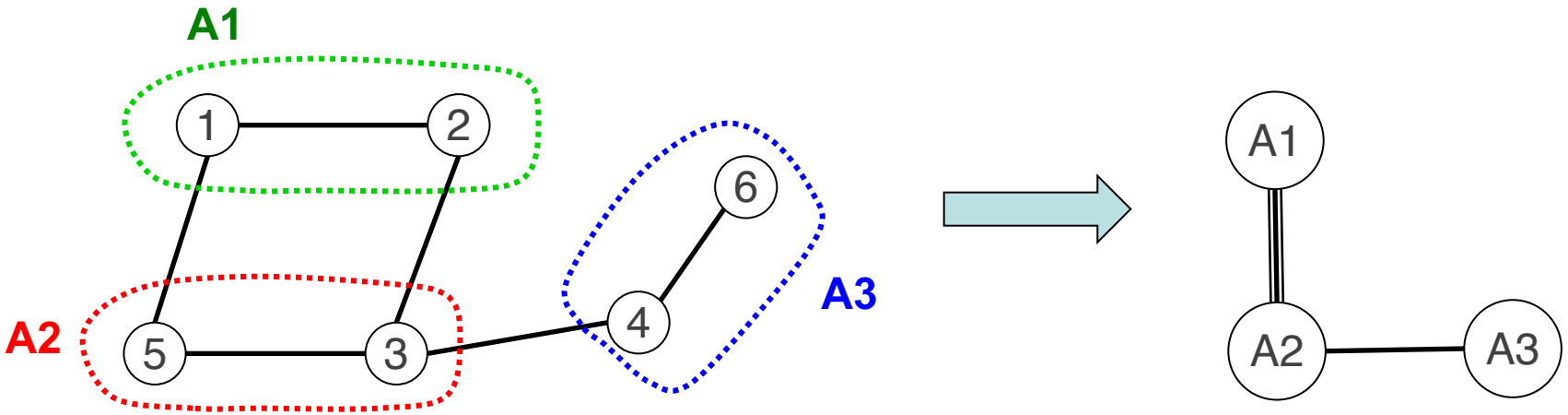


- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
- Highly-parallel implementation for Betweenness Centrality*

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

Coarsening via sparse matrix-matrix products





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

The GraphBLAS effort

Standards for Graph Algorithm Primitives

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

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

- The GraphBLAS Forum: http://graphblas.org
- Graphs: Architectures, Programming, and Learning (GrAPL @IPDPS): http://hpc.pnl.gov/grapl/

GraphBLAS Status: C API 1.2 released and in use

- Implementations of the GraphBLAS C specification:
  - SuiteSparse http://faculty.cse.tamu.edu/davis/suitesparse.html
  - IBM https://github.com/IBM/ibmgraphblas
  - Test suite for validating an implementation of the C-spec from SEI/CMU

    … to be released “soon”

- Systems using the GraphBLAS
  - RedisGraph v1.0 preview release:
    - RedisGraph is a graph database architecture implemented as a Redis Module, using GraphBLAS sparse matrices for internal data representation and linear algebra for query execution.
    - https://redislabs.com/blog/release-redisgraph-v1-0-preview/
  - Lincoln Labs GraphProcessor designed around the GraphBLAS.

- C++ bindings to the GraphBLAS
  - GBTL from SEI/CMU: https://github.com/cmu-sei/gbtl
  - Gunrock for GPUs: https://github.com/gunrock/gunrock-grb

GraphBLAS C API

- A binding of the GraphBLAS math to the C programming language.

- Requires C99 extended with function polymorphism based on static-types and number-of-parameters.
  - All modern C compilers in common use today support these extensions

- Basic include file with function prototypes, types, and constants
  - **`#include <GraphBLAS.h>`**

- Includes a few types and opaque objects (e.g. matrices and vectors) to give implementations maximum flexibility

  `GrB_Index` → An integer type used to set dimensions and index into arrays

  `GrB_Matrix` → A 2D sparse array, row indices, column indices and values

  `GrB_Vector` → A 1D sparse Array

  - … plus additional opaque objects we'll describe later (descriptors, semirings, binary operators, and unary operators)

GraphBLAS C API: Basic definitions

- **Opaque object**: An object manipulated strictly through the GraphBLAS API whose implementation is not defined by the GraphBLAS specification.
- **Transparent object**: an object whose structure is fully exposed to the programmer. E.g.: an array of tuples <i, j, value>
- **Method**: Any C function that manipulates a GraphBLAS opaque object.
- **Domain**: the set of available values used for the elements of matrices, the elements of vectors, and when defining operators.
  - Examples are `GrB_UINT64, GrB_INT32, GrB_BOOL, GrB_FP32`
- **Operation**: a method that corresponds to an operation defined in the GraphBLAS math spec. http://www.mit.edu/~kepner/GraphBLAS/GraphBLAS-Math-release.pdf
  - Examples: matrix multiply, matrix vector multiply, reduction, apply

Design principles of the GraphBLAS C API

- Object-oriented
  - **All objects are opaque, represented by handles**
  - Only GraphBLAS methods can manipulate those objects
- **Separation of data (matrices and vectors) and operations**
  - Only explicitly defined elements of a matrix or vector have values
  - The “structural zeros” are undefined
  - Any matrix/vector can be used with any semiring of compatible domain
  - Semantics are defined so that the “zero” value does not matter (most of the time)
- Blocking and nonblocking modes
  - **Blocking:** each method completes before returning
  - **Nonblocking:** methods may return early (must verify correctness of call)
  - Facilitated by opaqueness of objects
- Procedural specification
  - Semantics of each method is defined through process to compute output
  - Any implementation that produces the same output is conforming

GraphBLAS C API Spec (http://graphblas.org)

- **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]);
```

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

$\oplus . \otimes B^T$

A.Buluç, T. Mattson, S. McMillan, J. Moreira, **C. Yang**. "The GraphBLAS C API Specification", version 1.2.0

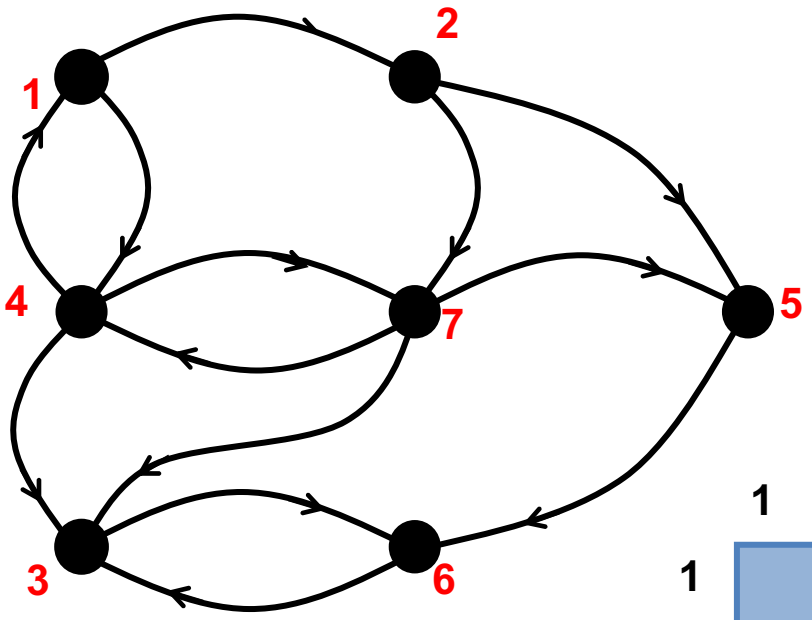
Examples of semirings in graph algorithms

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

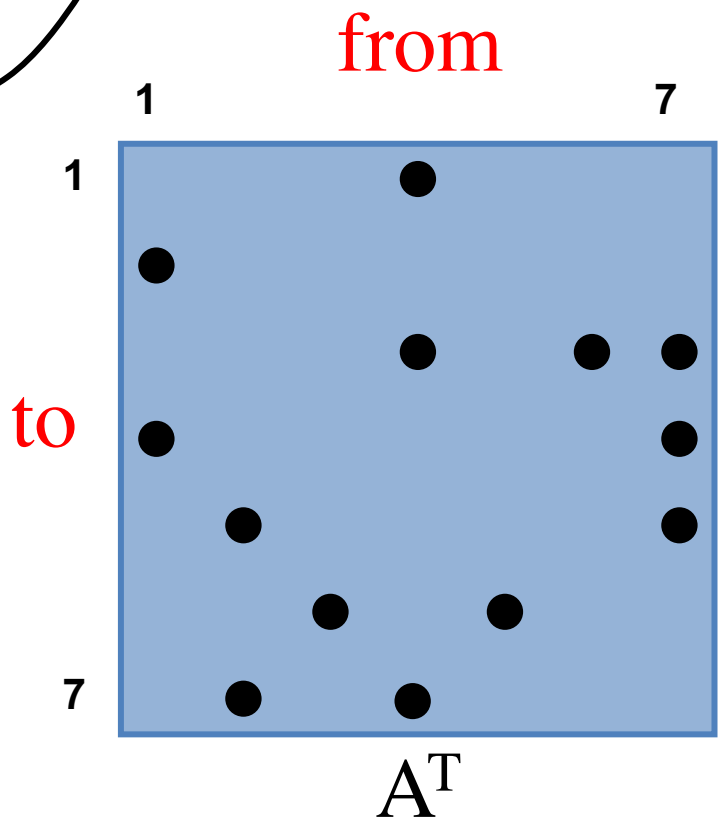
- **Shortened semiring notation: (Set, Add, Multiply)**. Both identities omitted.
- **Add:** Traverses edges, **Multiply:** Combines edges/paths at a vertex
- Neither add nor multiply needs to have an inverse.
- Both **add** and **multiply** are **associative**, **multiply distributes** over **add**

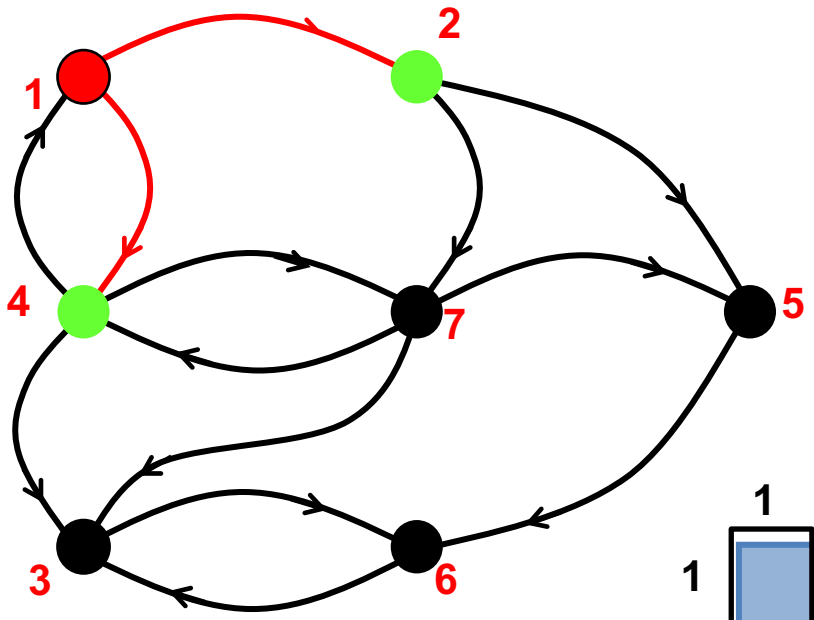
What does Mask accomplish?

- Masks avoids computation and materialization of intermediate objects.
- All masks are “write” masks (i.e. they apply to the output)
- Any object (not just Boolean) can be passed as a mask
- The ***structural complement*** of a mask can be used ***without materialization***
- Check the spec for the intricate semantics of mixing masks & accumulators.
- Masks are useful in domains outside graph analysis:
  - **Neural network** pruning: “Caffe was modified to add a mask which disregards pruned parameters during network operation for each weight tensor” (Han, Pool, Tran, and Dally, NIPS 2015)
  - Personalized PageRank (avoid converged vertices to receive messages)
  - “The sampled dense-dense matrix product (SDDMM) is written $P = A *_S B = (AB) \circ (S > 0)$. P’s values are the elements of the product AB evaluated at the nonzeros of S, and zero elsewhere. SDDMM is a bottleneck operation in all of the ***factor analysis algorithms*** (ALS, SFA, LDA, GaP).” (Canny and Zhao, 2013)



Breadth-first search in the language of matrices

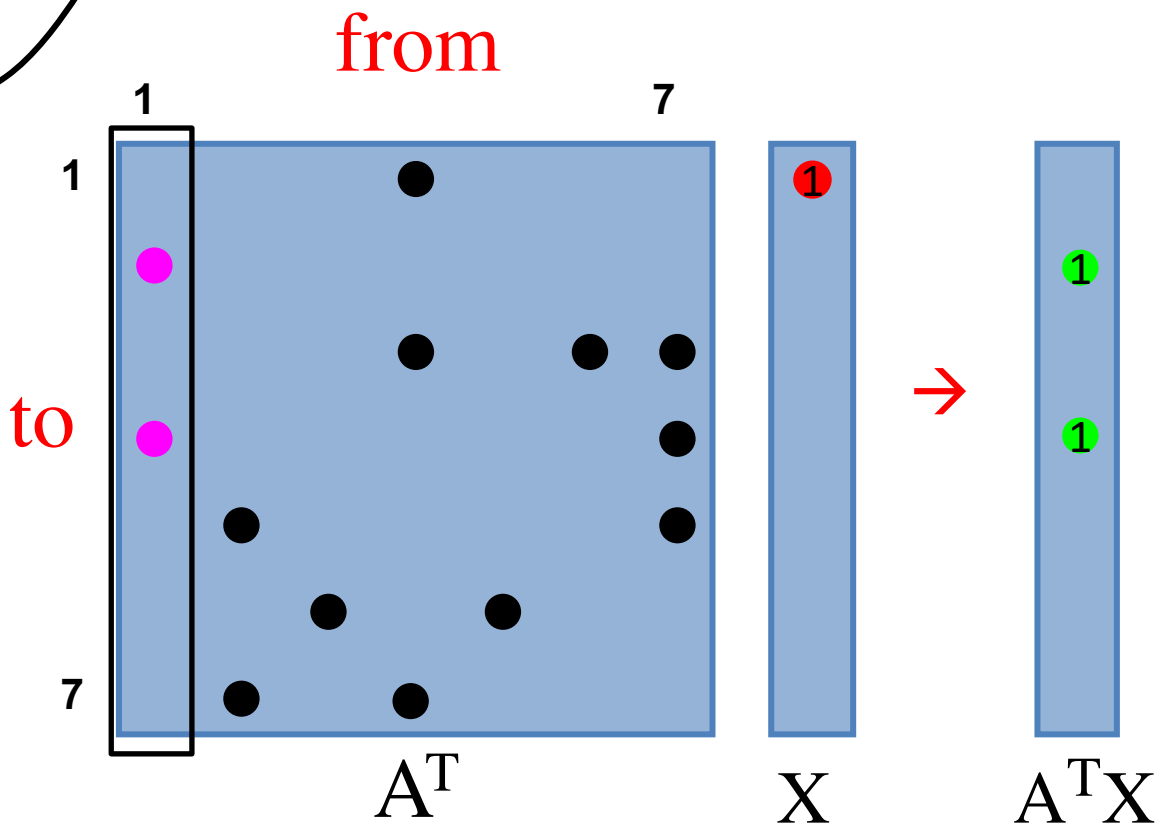




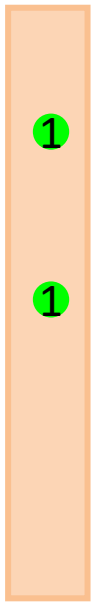
Particular semiring operations:

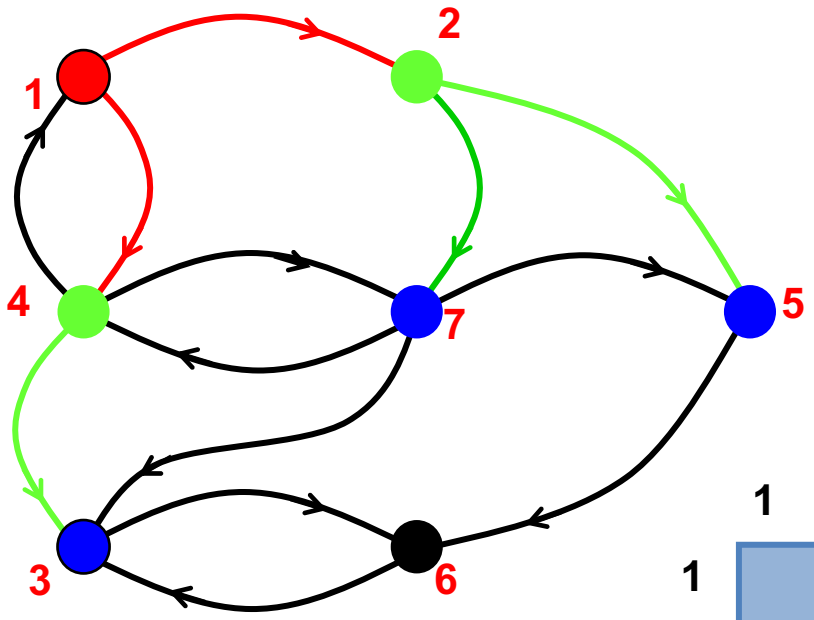
Multiply: select

Add: minimum

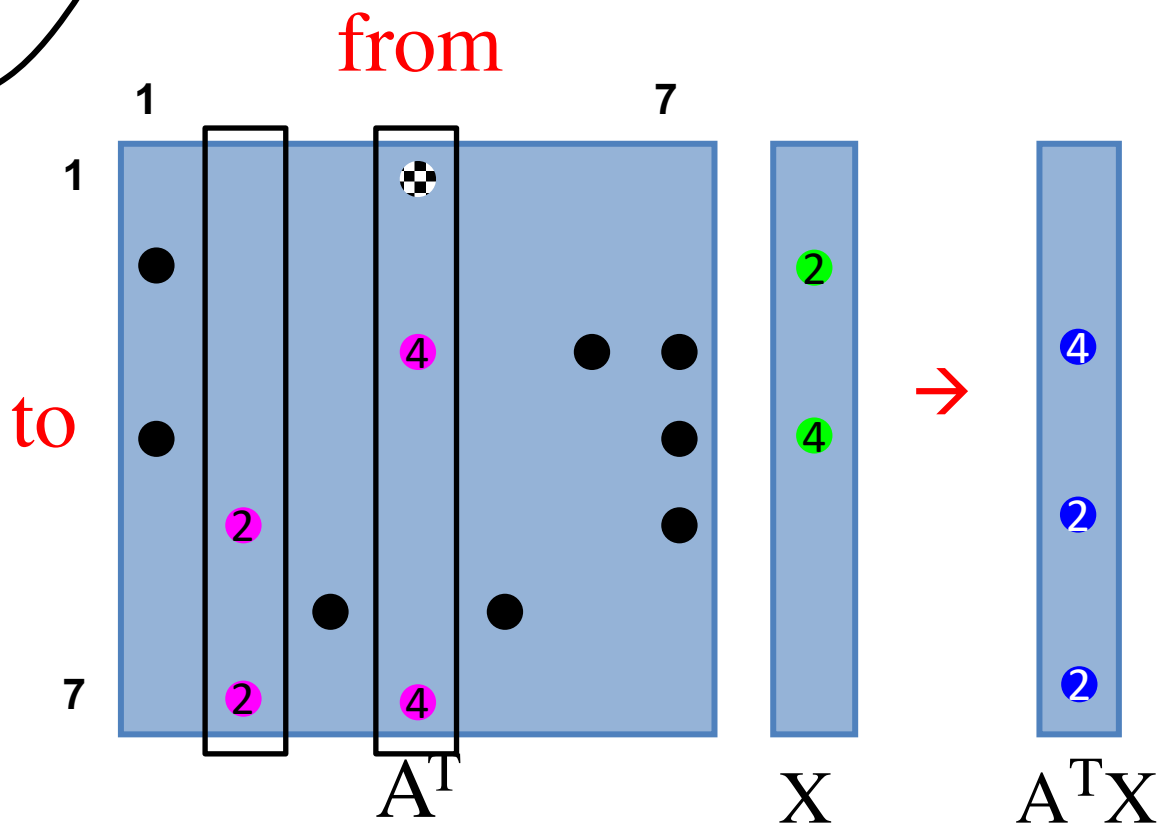


parents:

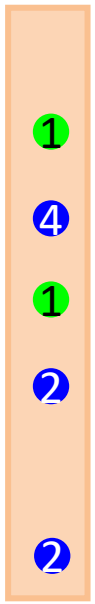


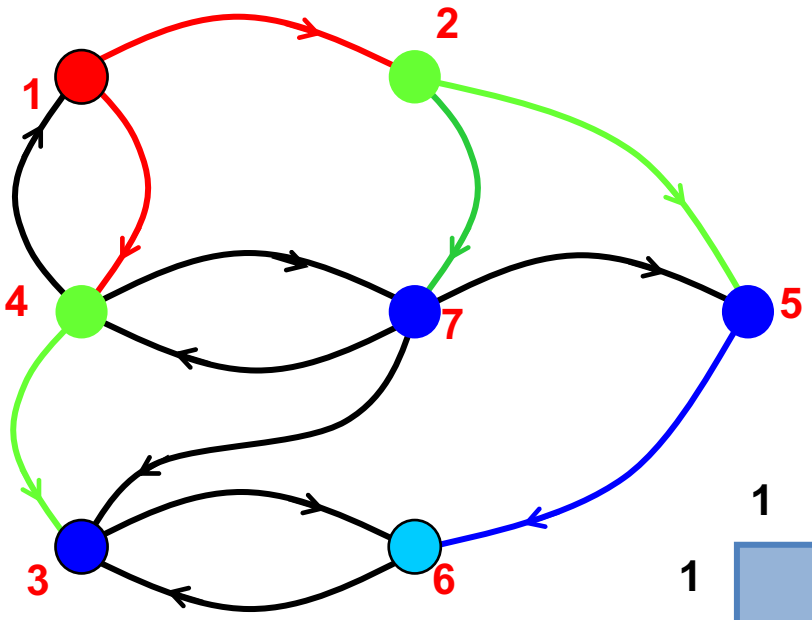


Select vertex with
minimum label as parent

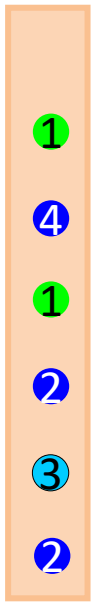


parents:



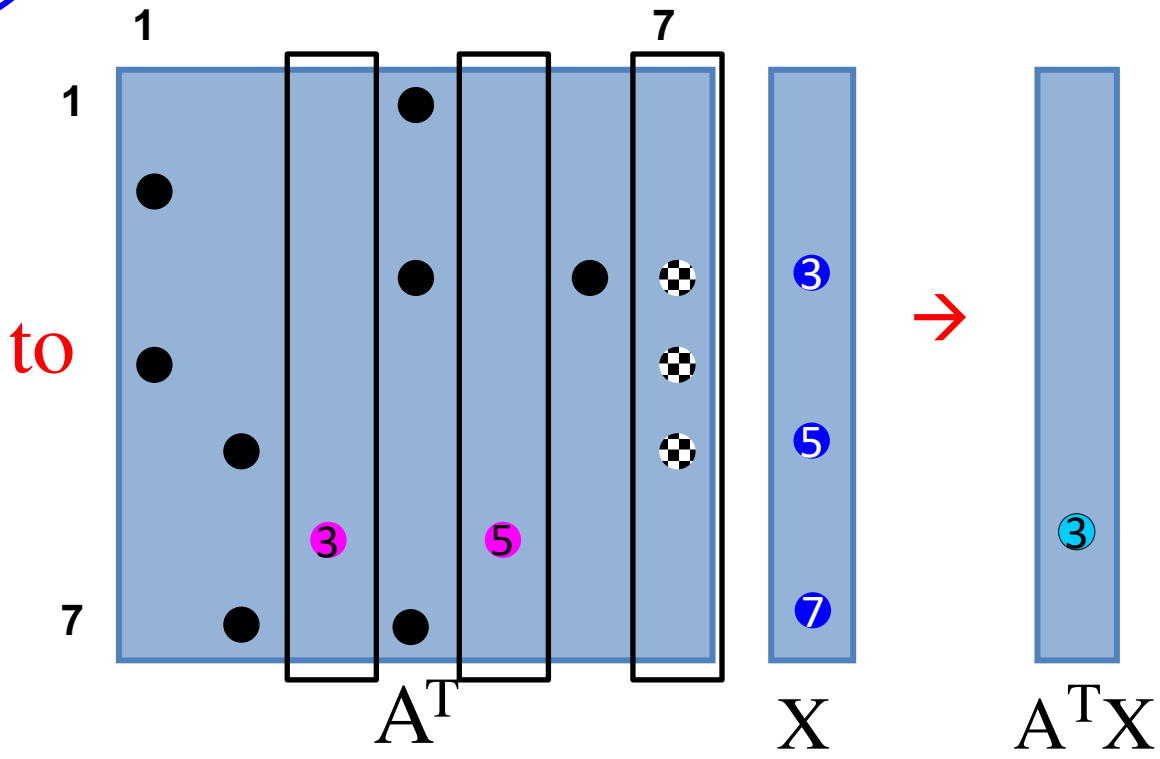


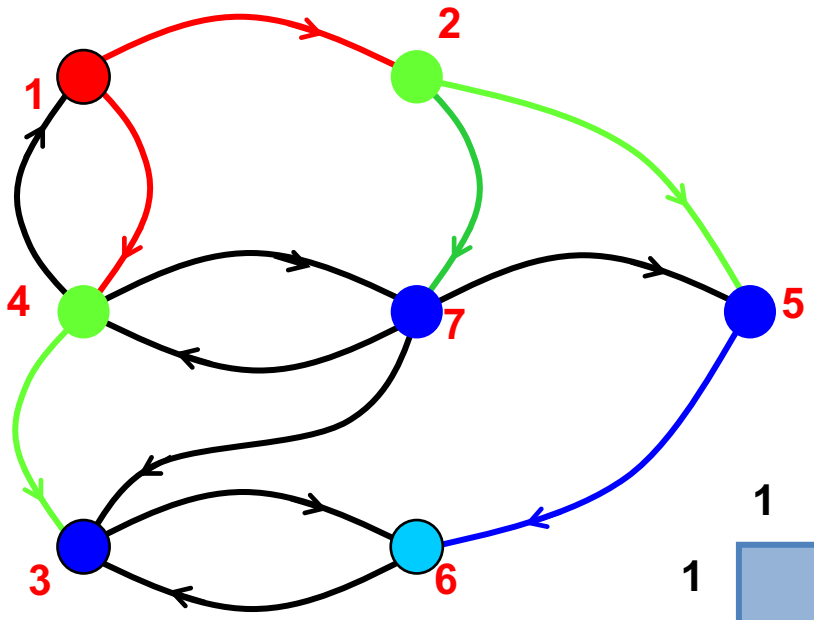
parents:



from

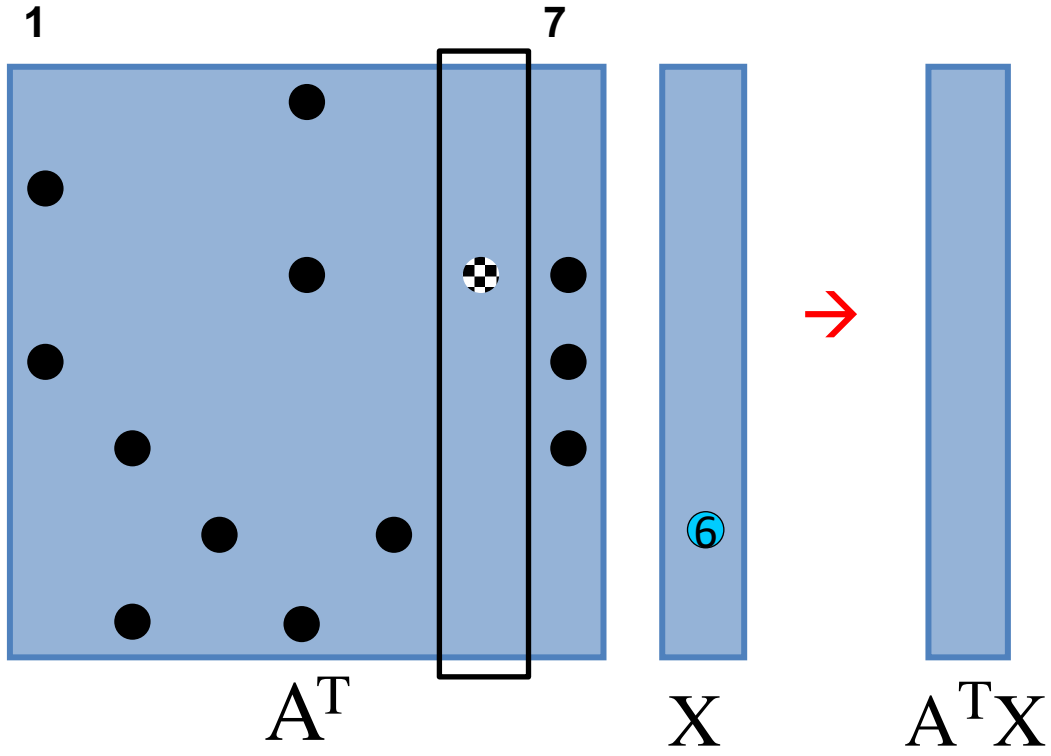
to





from

to

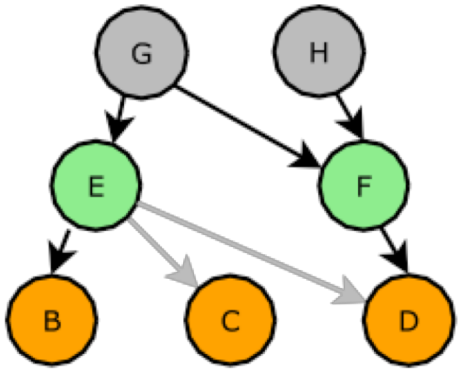


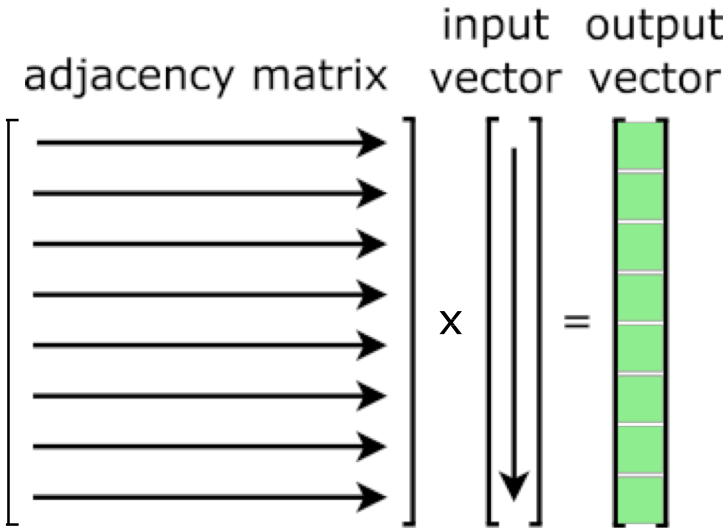
BFS in GraphBLAS with Masks

```
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.
```

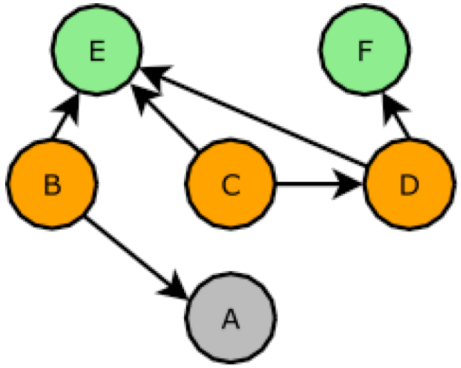
Push-pull ≡ column-row matvec!

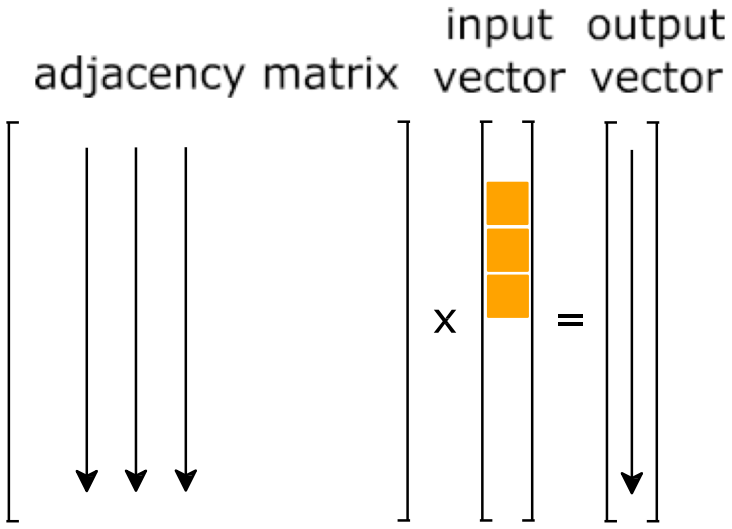
Pull





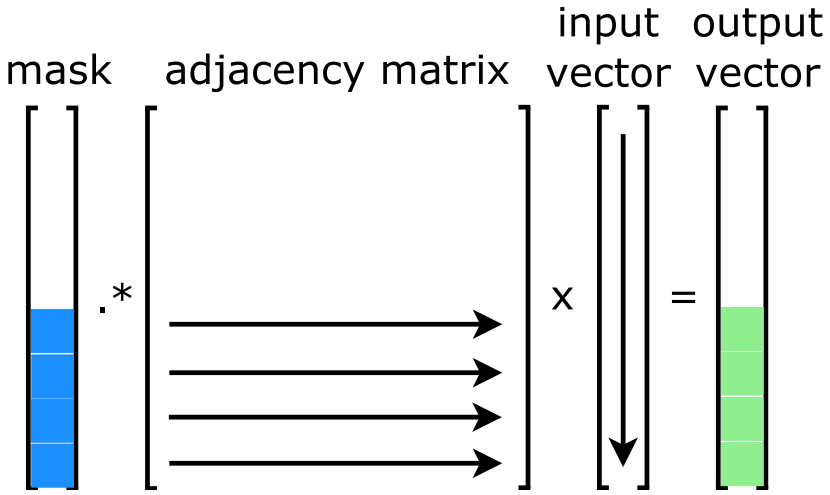
Push



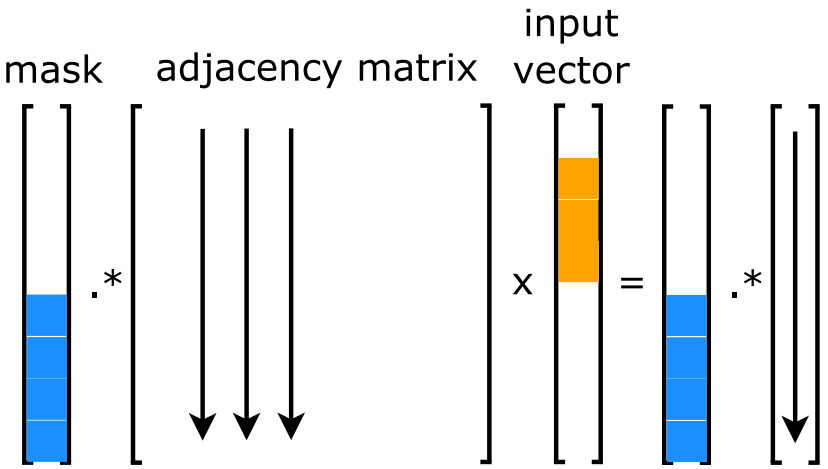


Yang, C., Buluc, A. and Owens, J.D., Implementing Push-Pull Efficiently in GraphBLAS. *ICPP'18*

Masks make "pull" implementable in GraphBLAS



Row-based matvec w/ mask

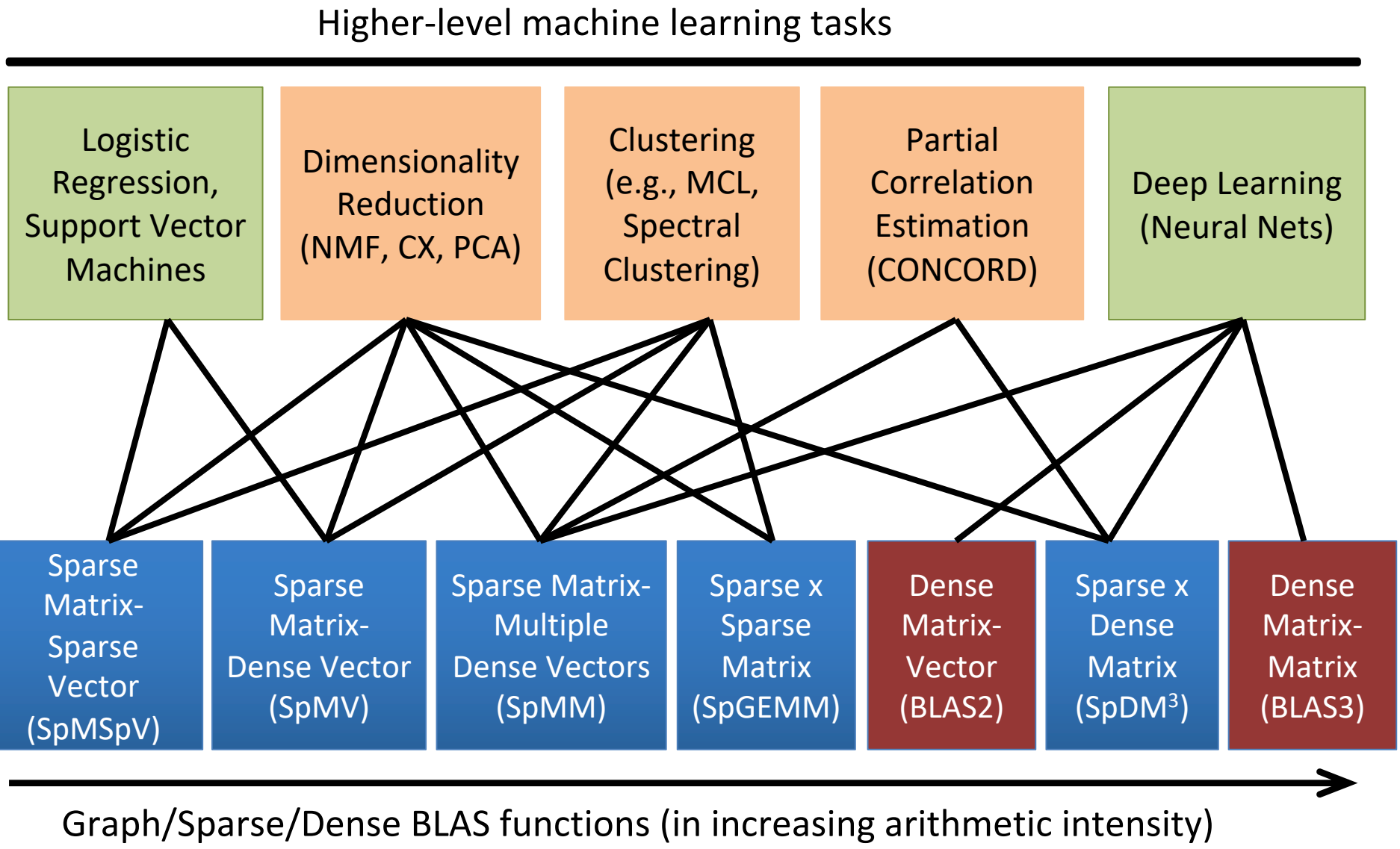


Column-based matvec w/ mask

Complexity: *O(dN)* → *O(d nnz(m))*

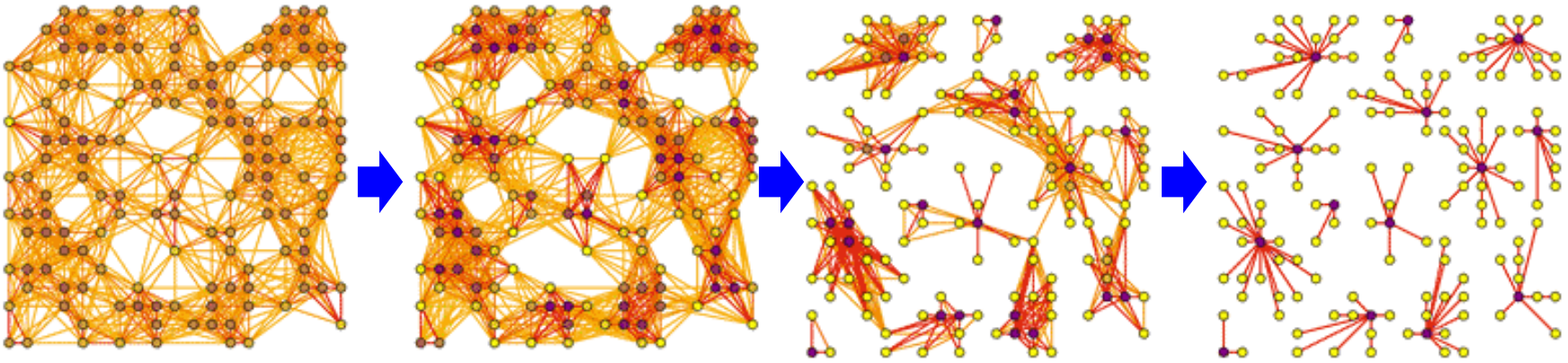
- d: average vertex degree
- *nnz(m)* is the number nonzeros in the mask
- *N* is the matrix/vector length

Machine Learning relies a lot on Linear Algebra too



Markov Cluster Algorithm (MCL)

Widely popular and successful algorithm for discovering clusters in protein interaction and protein similarity networks



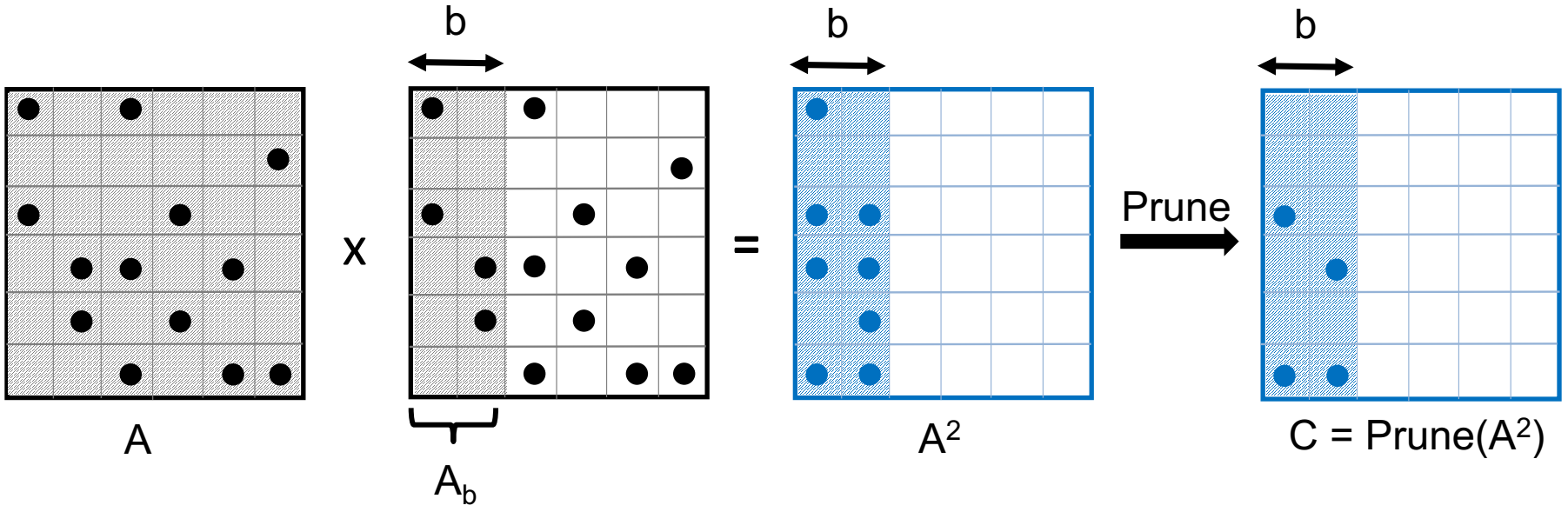
At each iteration:

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

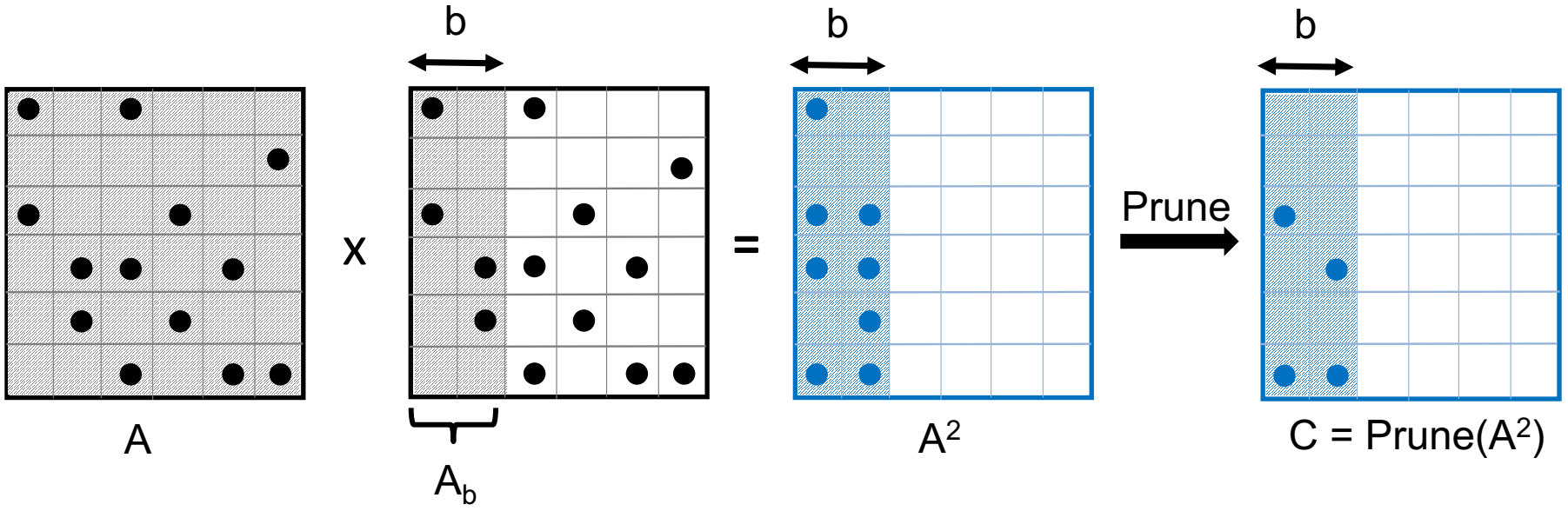
Step 2 (Inflation) : taking powers entry-wise

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

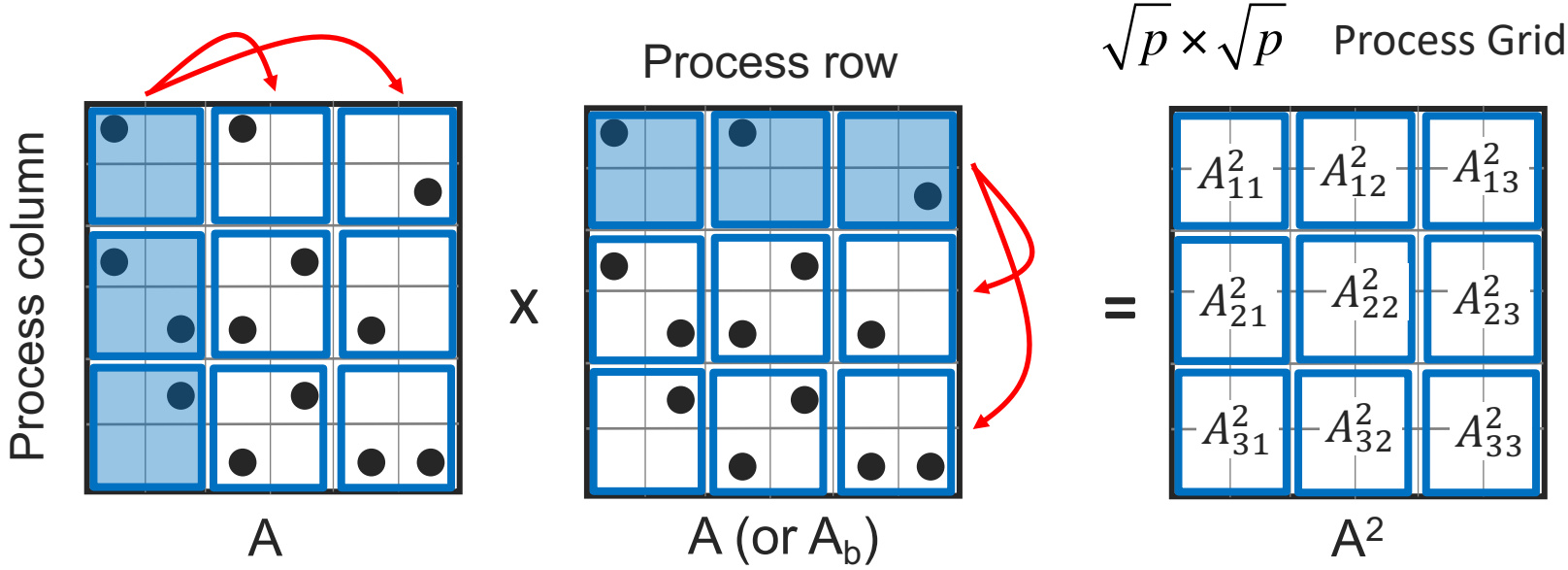
A combined expansion and pruning step



- 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

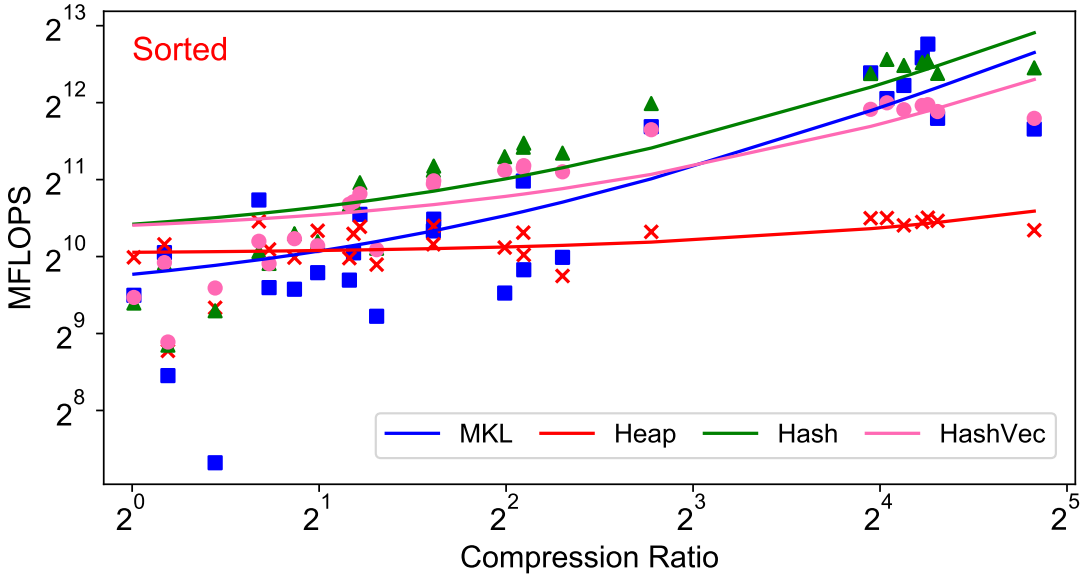
- MCL process is both **computationally expensive** and **memory hungry**, limiting the sizes of networks that can be clustered
- HipMCL overcomes such limitation via **sparse parallel algorithms**.
- **Up to 1000X times faster** than original MCL with same accuracy.

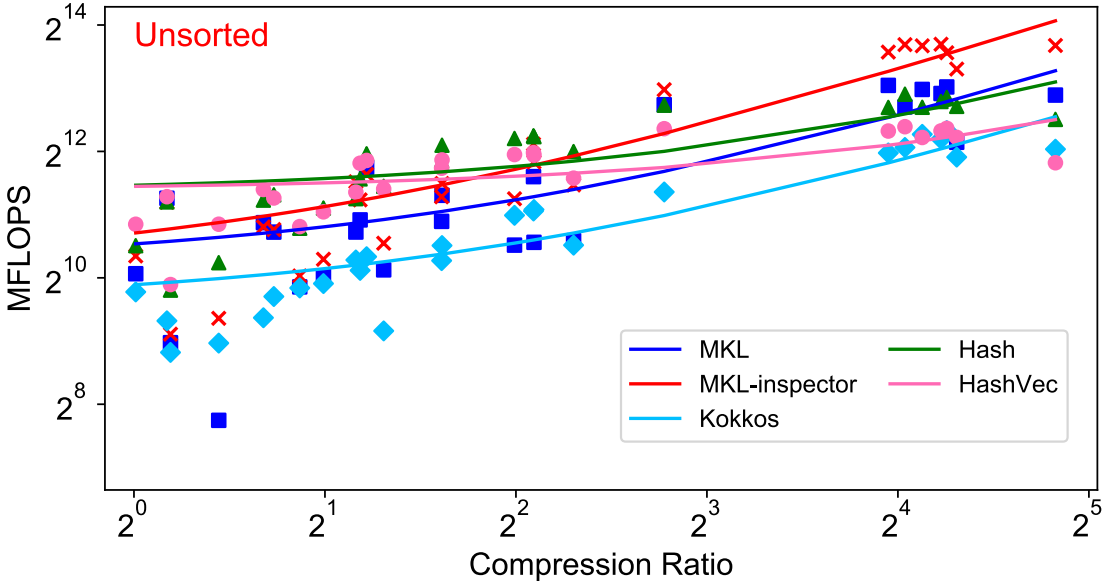


A. Azad, G. Pavlopoulos, C. Ouzounis, N. Kyrpides, A. Buluç; HipMCL: a high-performance parallel implementation of the Markov clustering algorithm for large-scale networks, *Nucleic Acids Research, 2018*

New shared-memory SpGEMM kernels

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





- We will integrate hash algorithms to CombBLAS and HipMCL

Yusuke Nagasaka, Satoshi Matsuoka, Ariful Azad, and Aydin Buluc. High-performance sparse matrix-matrix products on intel KNL and multicore architectures. In ICPPW, 2018.

SpGEMM on GPUs: tested 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

If these authors implemented the standard GrB_mxm, things would be much more portable. But we are still doing great compared to 5+ 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.

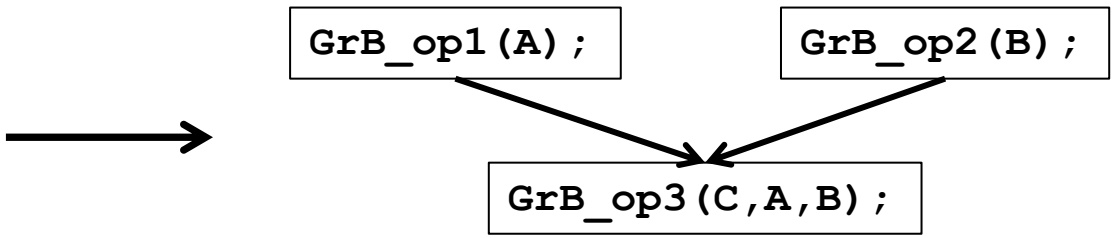
[2] Gremse, Felix, Kerstin Küpper, and Uwe Naumann. "Memory-Efficient Sparse Matrix-Matrix Multiplication by Row Merging on Many-Core Architectures." SIAM Journal on Scientific Computing 40, no. 4 (2018): C429-C449.

[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.

Execution modes

- A GraphBLAS program defines a DAG of operations.
- Objects are defined by the sequence of GraphBLAS method calls, but the value of the object is not assured until a GraphBLAS method queries its state.
- This gives an implementation flexibility to optimize the execution (fusing methods, replacing method sequences by more efficient ones, etc.)

```
GrB_op1(A);
GrB_op2(B);
GrB_op3(C,A,B);
```



- An execution of a GraphBLAS program defines a context for the library.
- The execution runs in one of two modes:
  - **Blocking mode** … executes methods in program order with each method completing before the next is called
  - **Non-Blocking mode** … methods launched in order. Complete in any order consistent with the DAG. Objects do not exit in fully defined state until queried.
- Most implementations only support Blocking mode. SuiteSparse:GraphBLAS uses nonblocking for assign and setElement

Opportunities in non-blocking mode

- Suppose you are solving a linear system on the Kronecker product graph
- Actually happens when you are computing similarity between two graphs
- Using “graph kernels” enable machine learning on graph structures data, such as proteins and other molecules.

$$\underset{N\,x\,M}{\mathbf{C}} = \underset{}{\mathbf{A}} \otimes \underset{K\,x\,L}{\mathbf{B}} \doteq \underset{N*K\,x\,M*L}{\begin{pmatrix} a_{1,1}\mathbf{B} & a_{1,2}\mathbf{B} & \dots & a_{1,m}\mathbf{B} \\ a_{2,1}\mathbf{B} & a_{2,2}\mathbf{B} & \dots & a_{2,m}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & a_{n,2}\mathbf{B} & \dots & a_{n,m}\mathbf{B} \end{pmatrix}}$$

- The Kronecker product itself has huge memory footprint and lots of redundancy (NK+ML dimension but NKML apparent values)

Opportunities in non-blocking mode

- The only way to write this in GraphBLAS or any other library we know of:

```
GrB_kronecker(C, …, A, B, …); // C=A⊗B
GrB_mxv(y, …, C, x, …); // y=C x
```

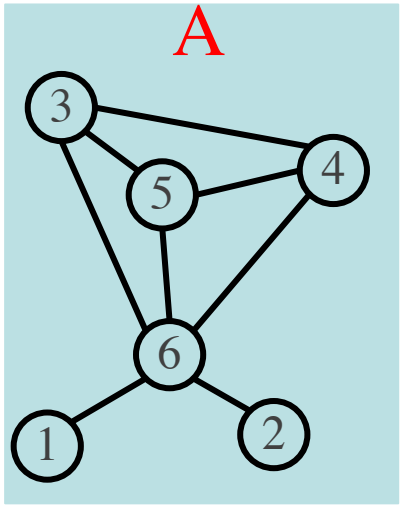
- What we would rather call:

```
GrB_kronxv(y, …, A, B, x …); // y= (A⊗B) x
```

- But that would result in API bloat and would lead us to a rabbit hole.
- There are many other examples:
  - KFAC (optimization method for deep learning),
  - Triple matrix product (graph contraction and AMG restriction),
  - Triangle counting (who needs the list of triangles when all we need is the count)

- **Solution**: **A JIT that performs automatic operator fusion**

Triangle counting in matrix algebra

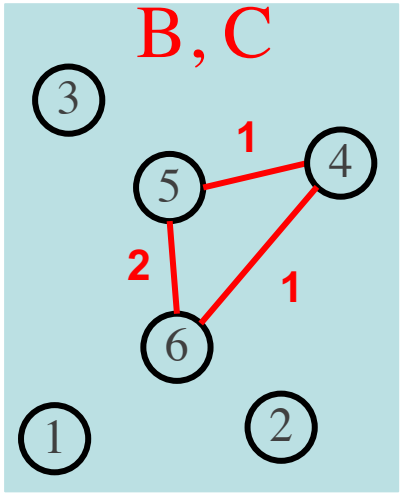


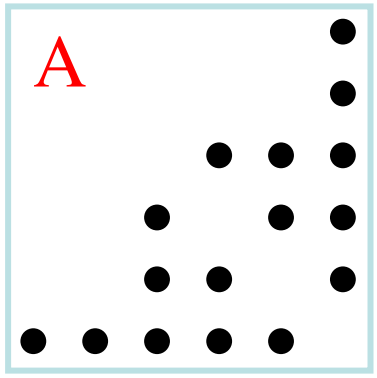
$A = L + U$ (hi->lo + lo->hi)

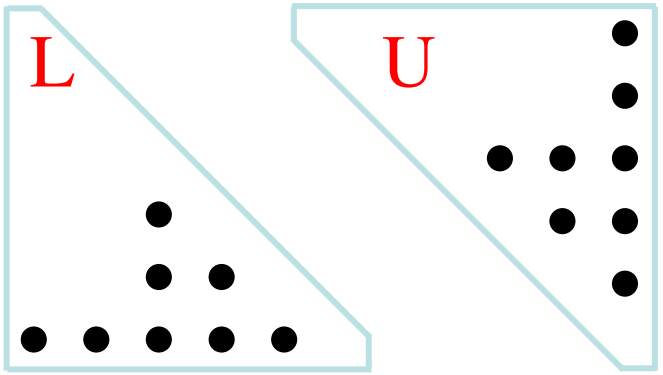
$L \times U = B$ (wedge, low hinge)

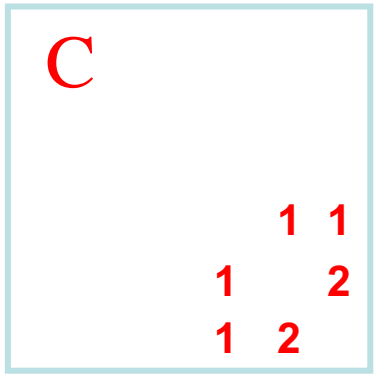
$A \wedge B = C$ (closed wedge)

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









Ariful Azad, Aydin Buluc, and John R Gilbert. “Parallel triangle counting and enumeration using matrix algebra”. *IPDPS Workshops, 2015*

Multithreading and the GraphLAS

- The language designers oath:
  - First, do no harm
- This is hard to do as soon as you introduce multithreading.
- We require that the GraphBLAS() methods are thread safe, but that was as far as we went.
- Only one thread can call GrB_init() and GrB_finalize().
- Basically, we created a spec that hides threads inside individual GraphBlas methods
  - In OpenMP jargon, GraphBLAS is fine if each method contains its own distinct parallel region
  - That approach has numerous problems … high thread management overhead and terrible memory movement costs.

```
GrB_init(GrB_BLOCKING);

GrB_new(A…);      GrB_build(A…);
GrB_new(B…);      GrB_build(B…);
GrB_new(C…);      GrB_build(C…);

GrB_index norm;
GrB_reduce(&norm, C …);

While(norm>0){
    GrB_eWiseMult(A, A, B …);
    GrB_mxm(A, B, C …);
    GrB_reduce(&norm, C);
}
GrB_wait();

GrB_mxm(A,B, C…):  // start a new sequence

GrB_finalize();
```

Multithreading and the GraphLAS

- Multithreading issues in current spec
  - Can multiple threads call init so each thread has its own independent sequence?
  - If multiple threads process one sequence, how to we define happens-before relations inside a sequence?
    - Maybe a barrier-like GrB_wait() is too much and we need a GrB_wait() at the object level.
  - How do we manage threads so multithreaded libraries compose with multithreaded applications.
  - GraphBLAS objects are opaque. That means we own the memory model defining how threads interact through GraphBLAS objects

```
GrB_init(GrB_BLOCKING);

GrB_new(A…);      GrB_build(A…);
GrB_new(B…);      GrB_build(B…);
GrB_new(C…);      GrB_build(C…);

GrB_index norm;
GrB_reduce(&norm, C …);

While(norm>0){
    GrB_eWiseMult(A, A, B …);
    GrB_mxm(A, B, C …);
    GrB_reduce(&norm, C);
}
GrB_wait();

GrB_mxm(A,B, C…):  // start a new sequence

GrB_finalize();
```

Long term spec work

- New major release for the C spec
  - Everything from our “short term” list that didn’t make it into the GrAPL release.
  - Dynamic Graph Support:
    - Edge deletion
    - Vertex removal/addition
  - A tuple iterator … iterate over the non-empty elements of an object
  - **Full support for distributed execution.**
    - **With MPI**
    - **Potentially another PGAS language or library (UPC++, BCL, etc.)**
- New Language bindings
  - **C++**
  - Python

Acknowledgments

Ariful Azad, David Bader, Tim Davis, John Gilbert, Jeremy Kepner, Nikos Kyrpides, Tim Mattson, Scott McMillan, Jose Moreira, Lenny Oliker, John Owens, Christos Ouzounis, Georgios Pavlopoulos, Oguz Selvitopi, Yu-Hang Tang, Carl Yang, Kathy Yelick.

- The GraphBLAS Forum: http://graphblas.org
- Graphs: Architectures, Programming, and Learning (GrAPL @IPDPS): http://hpc.pnl.gov/grapl/