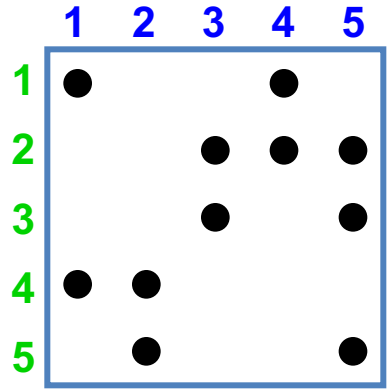

Berkeley
UNIVERSITY OF CALIFORNIA

# Communication-Avoiding Sparse Matrix Algorithms for Large Graph and Machine Learning Problems

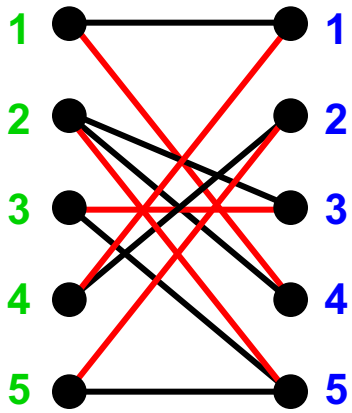
Aydın Buluç
Computational Research Division, LBNL
EECS Department, UC Berkeley

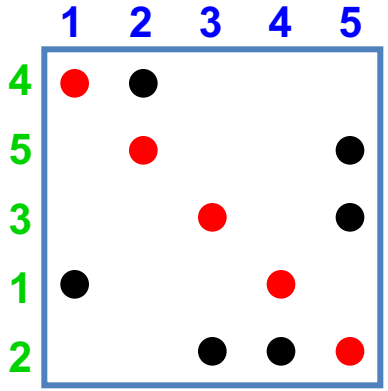
New Architectures and Algorithms
IPAM (UCLA), Nov 28, 2018

# Large Graphs in Scientific Discoveries



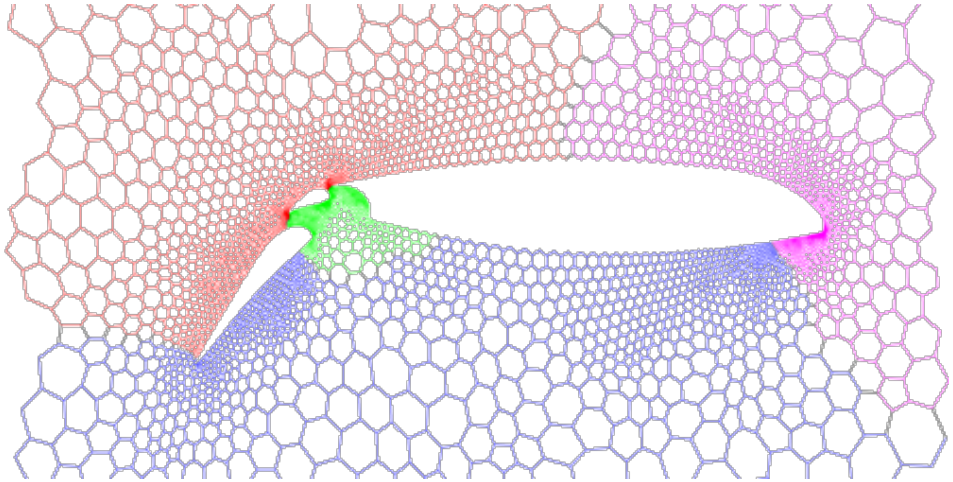

A




PA

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



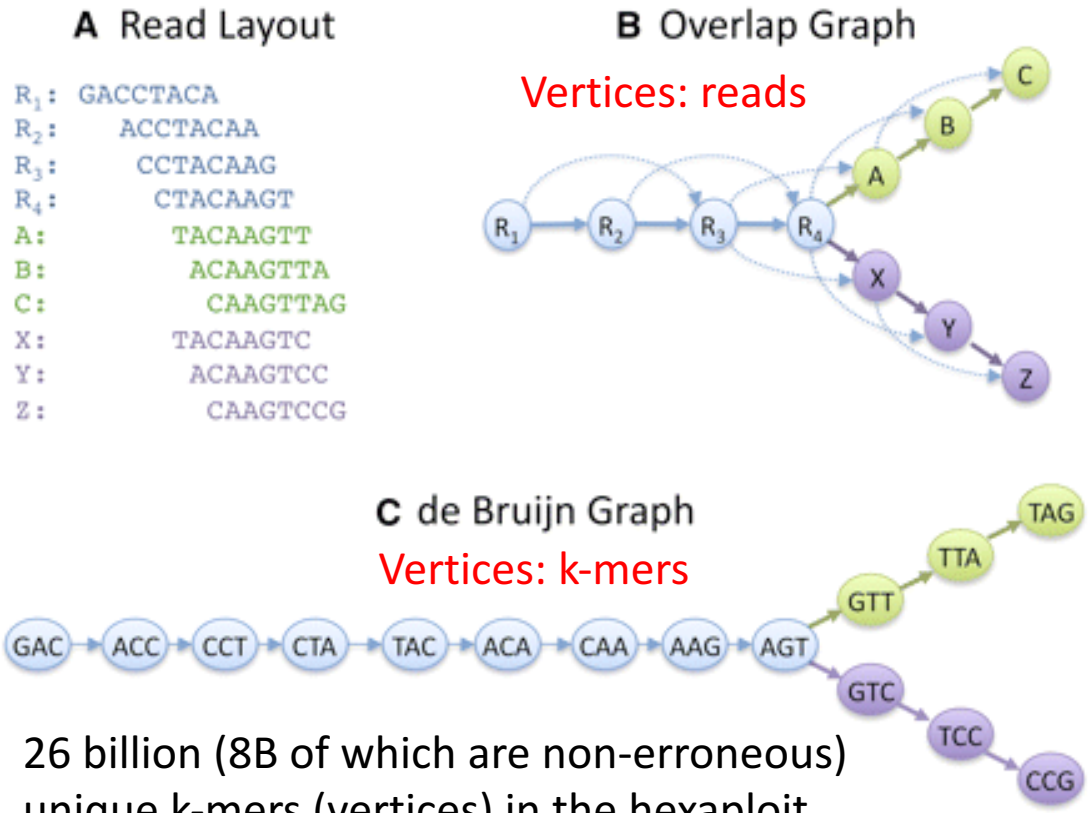
**Graph partitioning:** *Dynamic load balancing* in parallel simulations

Picture (left) credit: Sanders and Schulz

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

# Large Graphs in Scientific Discoveries

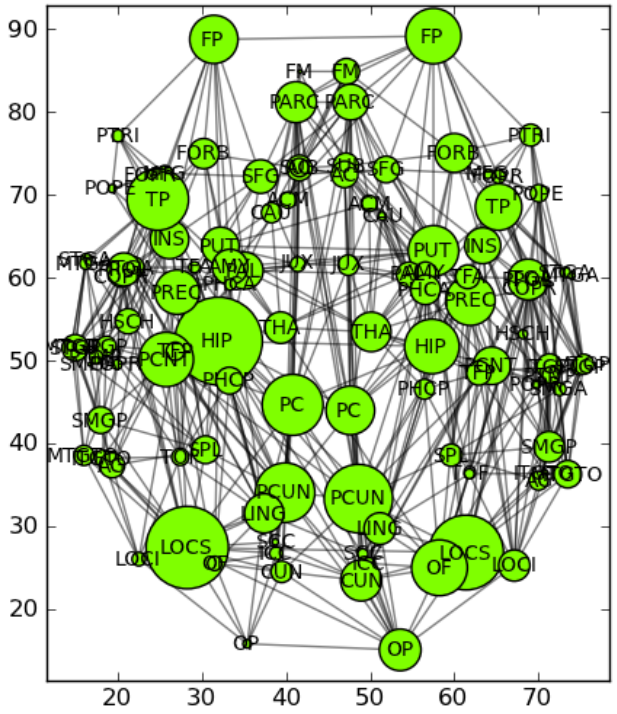
## Whole genome assembly




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

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

## Graph Theoretical analysis of Brain Connectivity



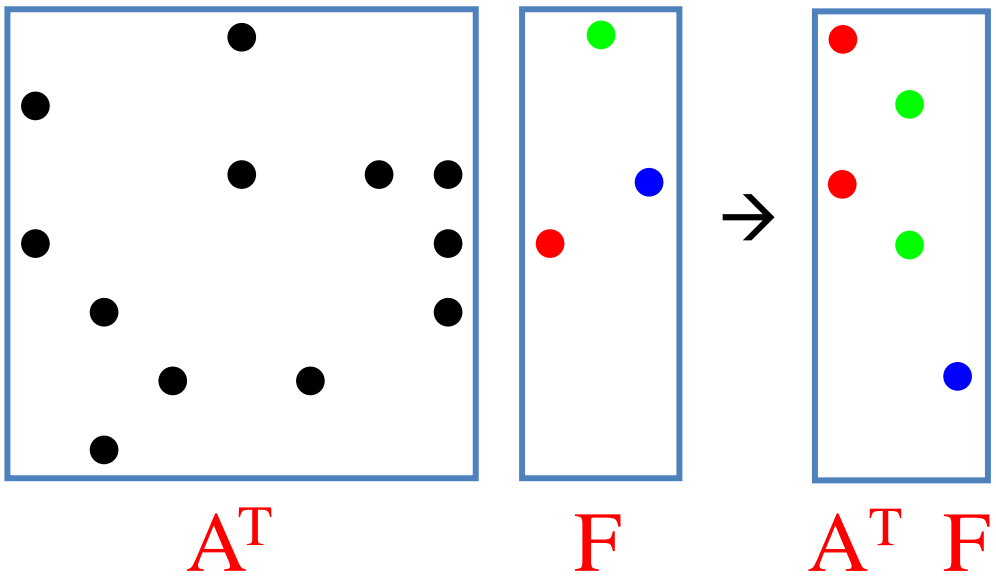

Potentially millions of neurons and billions of edges with developing technologies

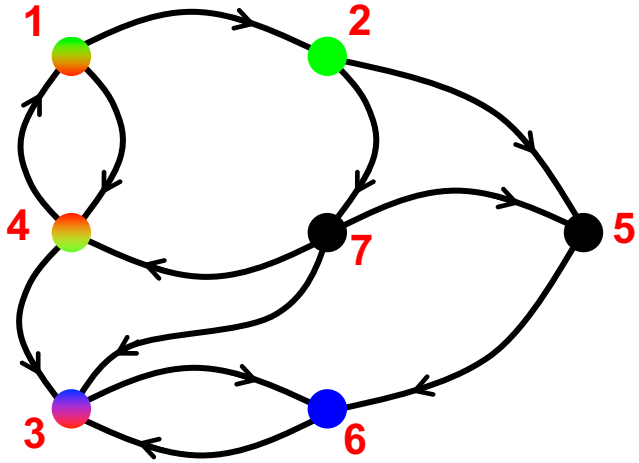
# Sparse Matrices

"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

# Graphs in the language of matrices

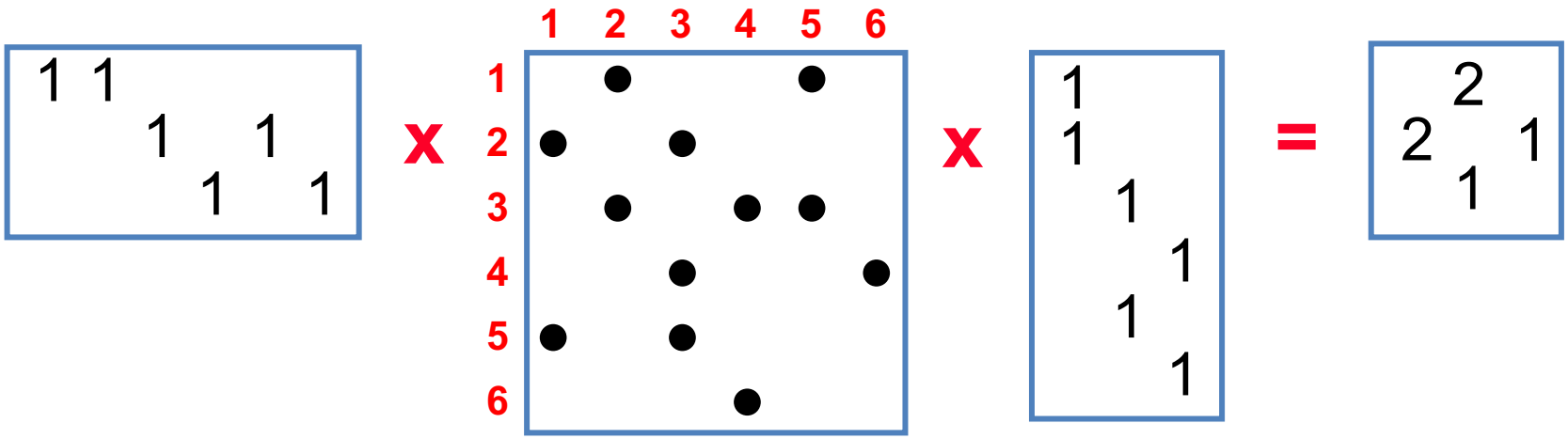


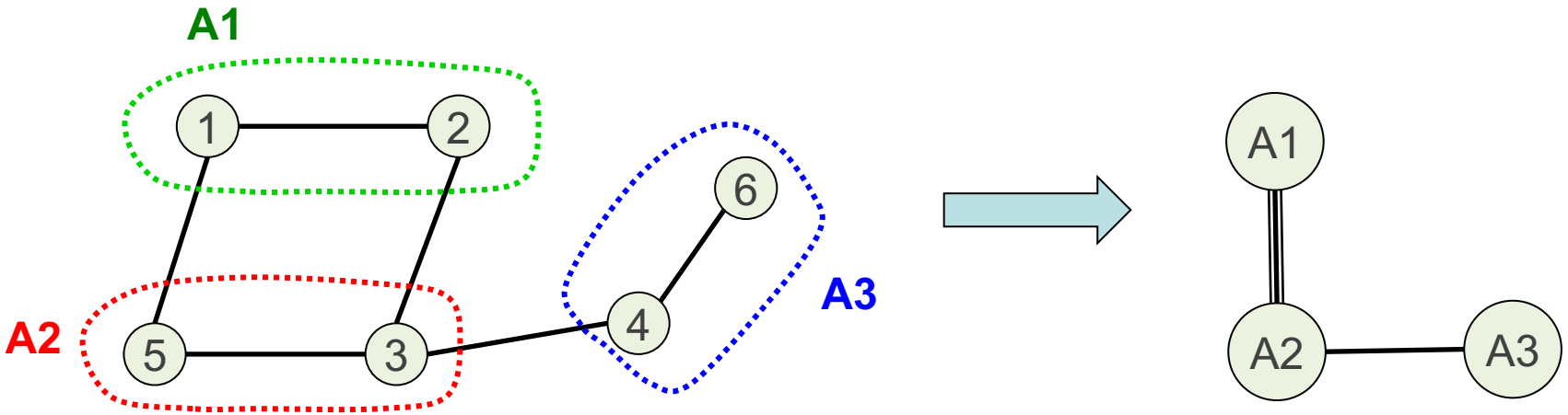




- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
- Highly-parallel implementation for Betweenness Centrality*

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

# Graph 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 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

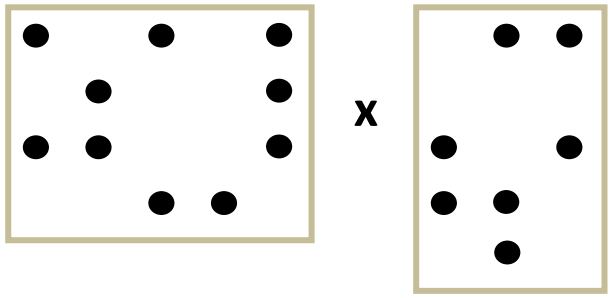
# The case for sparse matrices

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

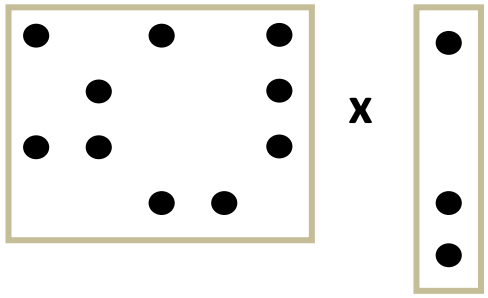
| Traditional graph computations | Graphs in the language of linear algebra |
|---|---|
| Data driven, unpredictable communication. | Fixed communication patterns |
| Irregular and unstructured, poor locality of reference | Operations on matrix blocks exploit memory hierarchy |
| Fine grained data accesses,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

# Linear-algebraic primitives for graphs

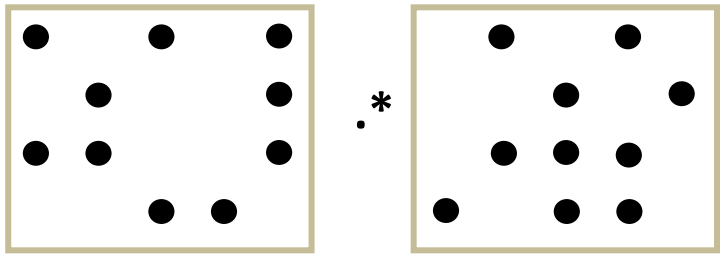
Sparse matrix X sparse matrix



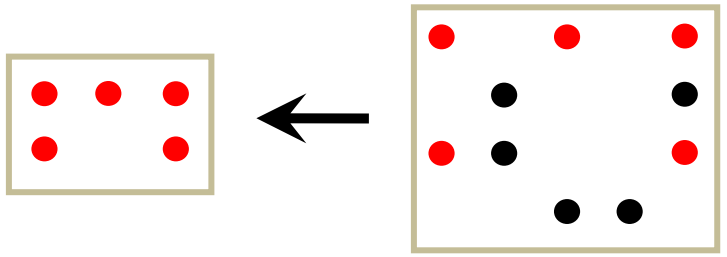
Sparse matrix X sparse vector



Element-wise operations



Sparse matrix indexing

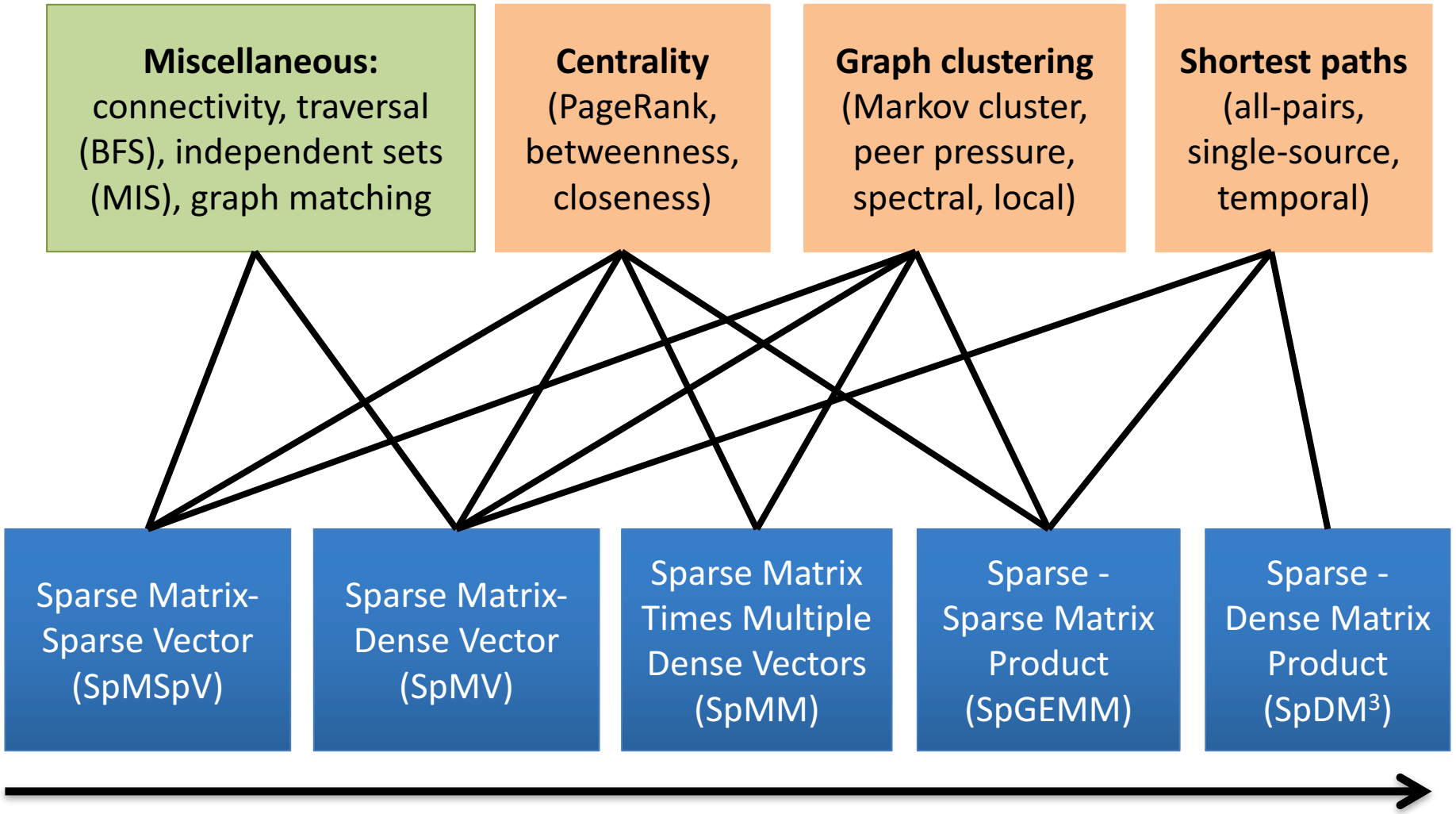


Is **think-like-a-vertex** really more productive?

"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**

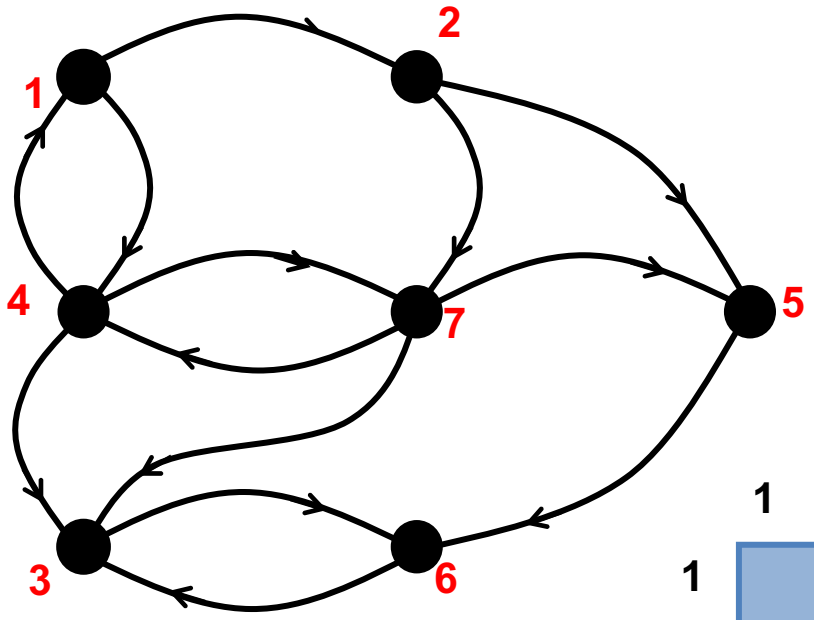
# Graph Algorithms on GraphBLAS



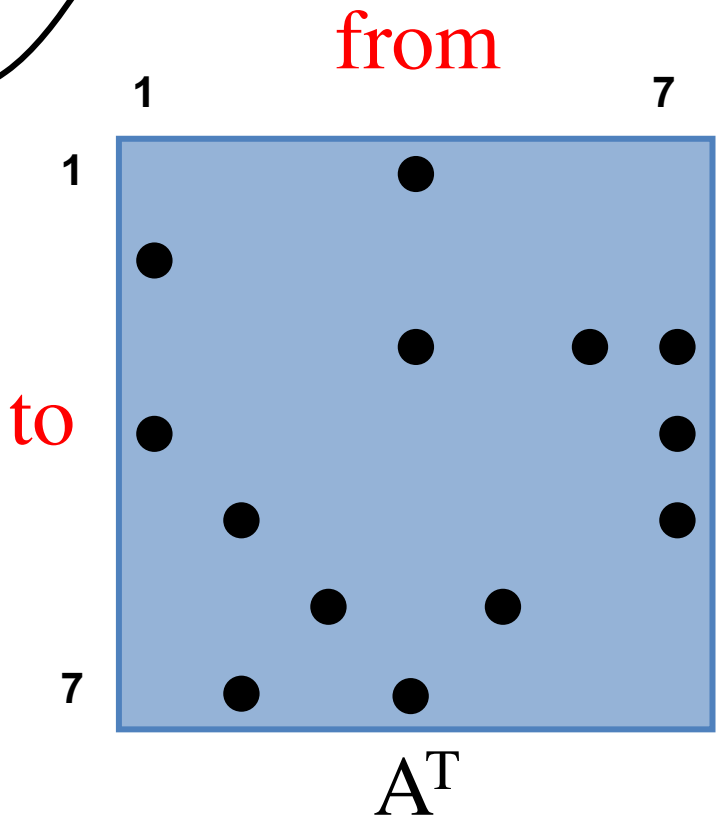
# Examples of semirings in graph algorithms

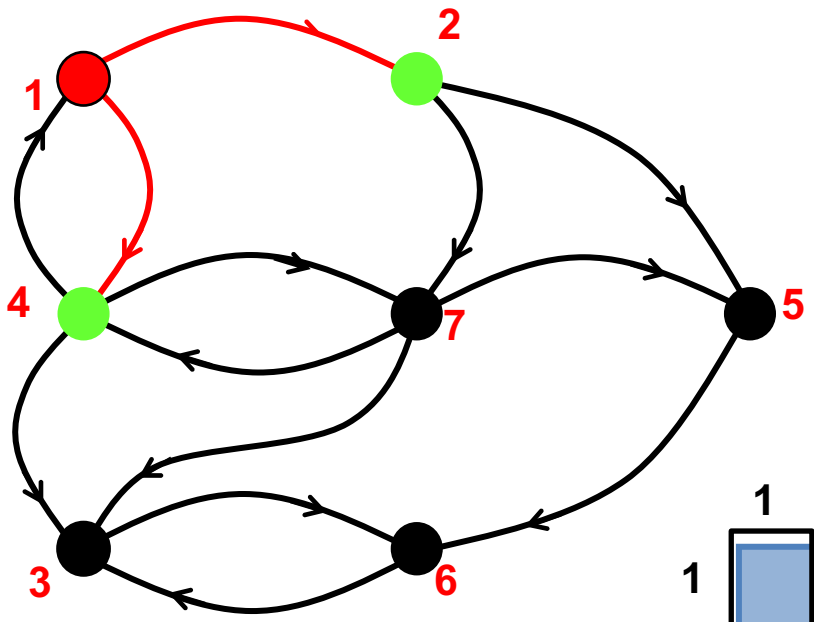
| | |
|---|---|
| Real field: **(R, +, x)** | Classical numerical linear algebra |
| Boolean algebra: **({0 1}, \|, &)** | Graph connectivity |
| Tropical semiring: **(R U {∞}, min, +)** | Shortest paths |
| **(S, select, select)** | Select subgraph, or contract nodes to form quotient graph |
| (edge/vertex 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**




# Breadth-first search in the language of matrices

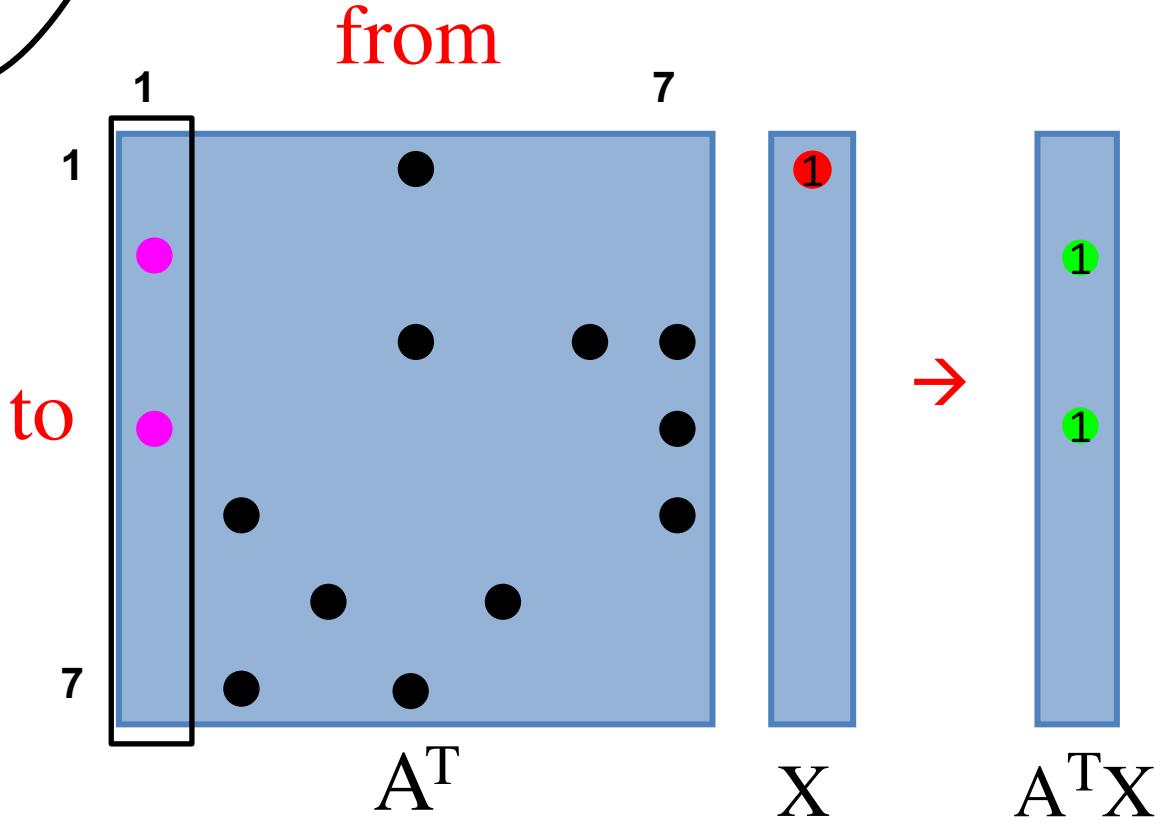




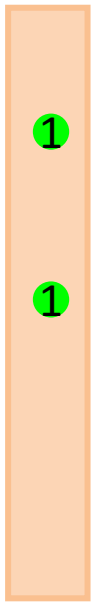

Particular semiring operations:

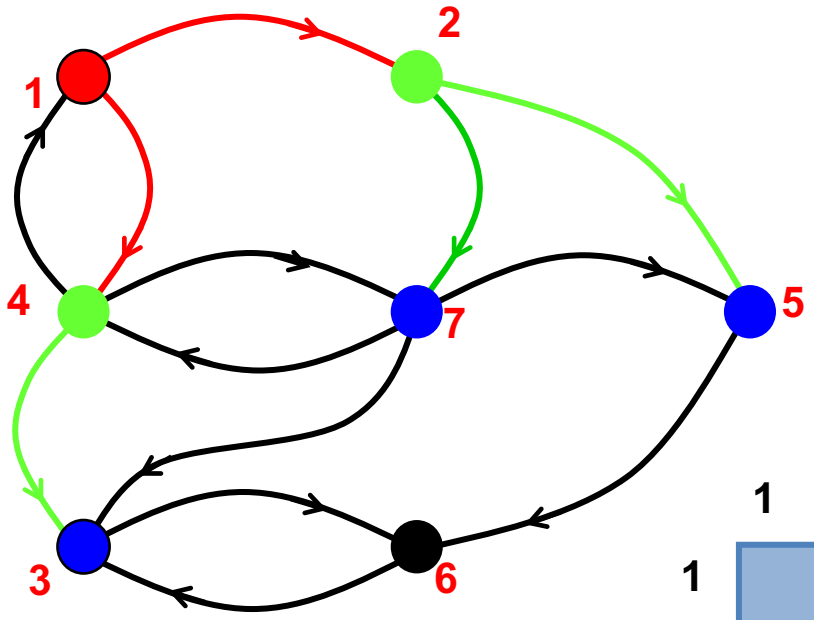
**Multiply:** select

**Add:** minimum



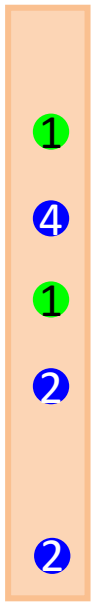

parents:





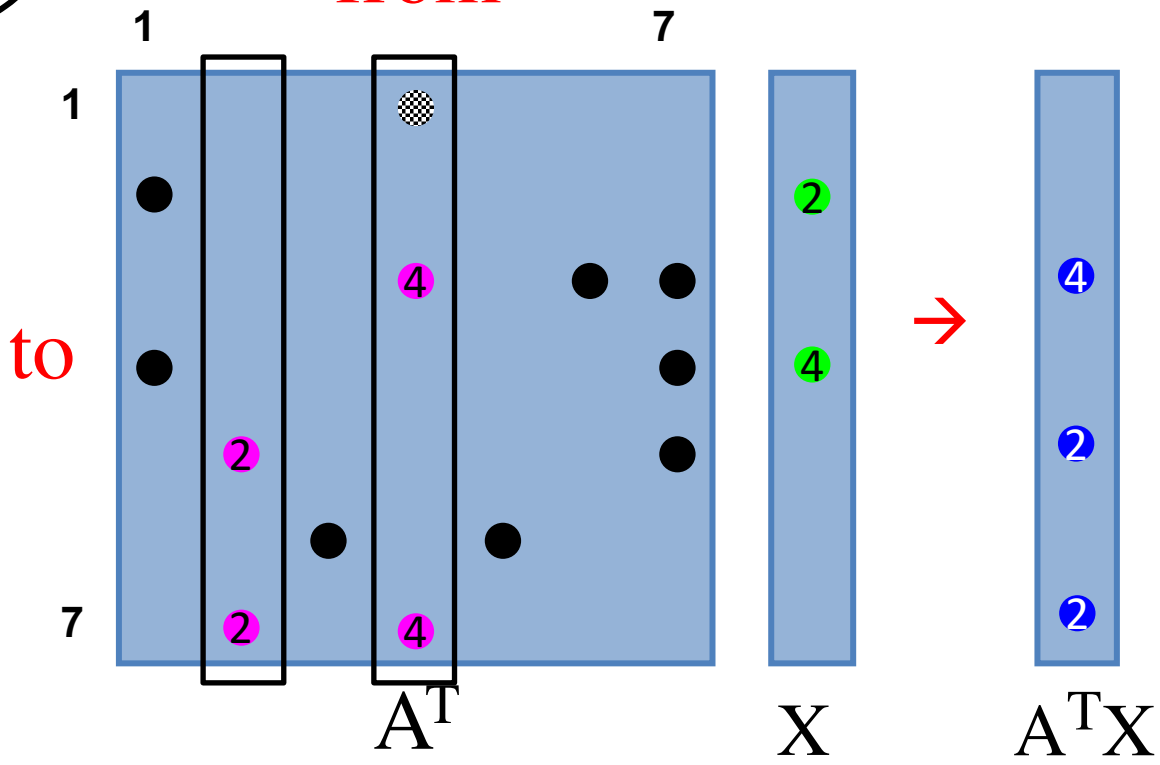

Select vertex with
minimum label as parent

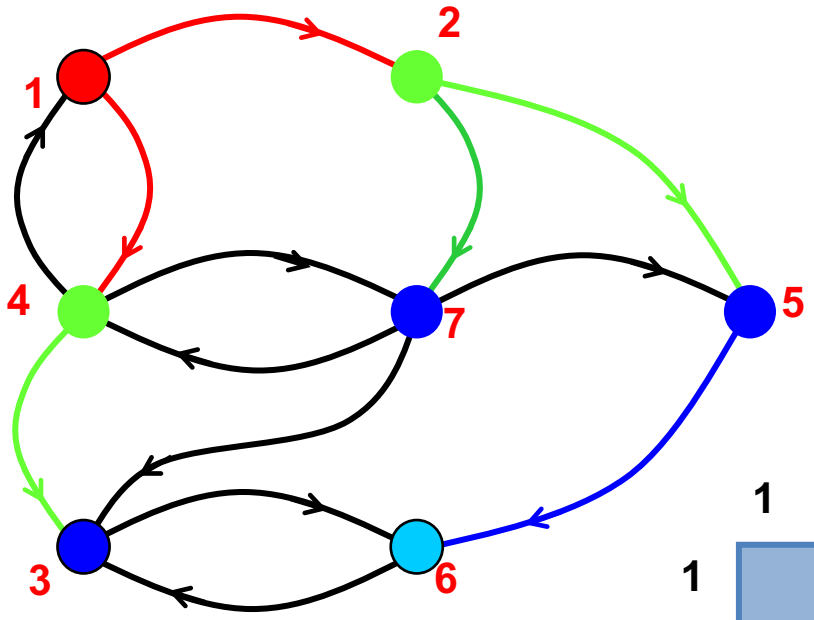
parents:



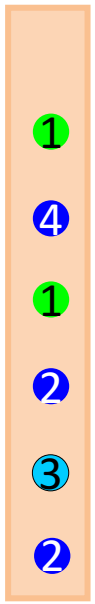

from

to



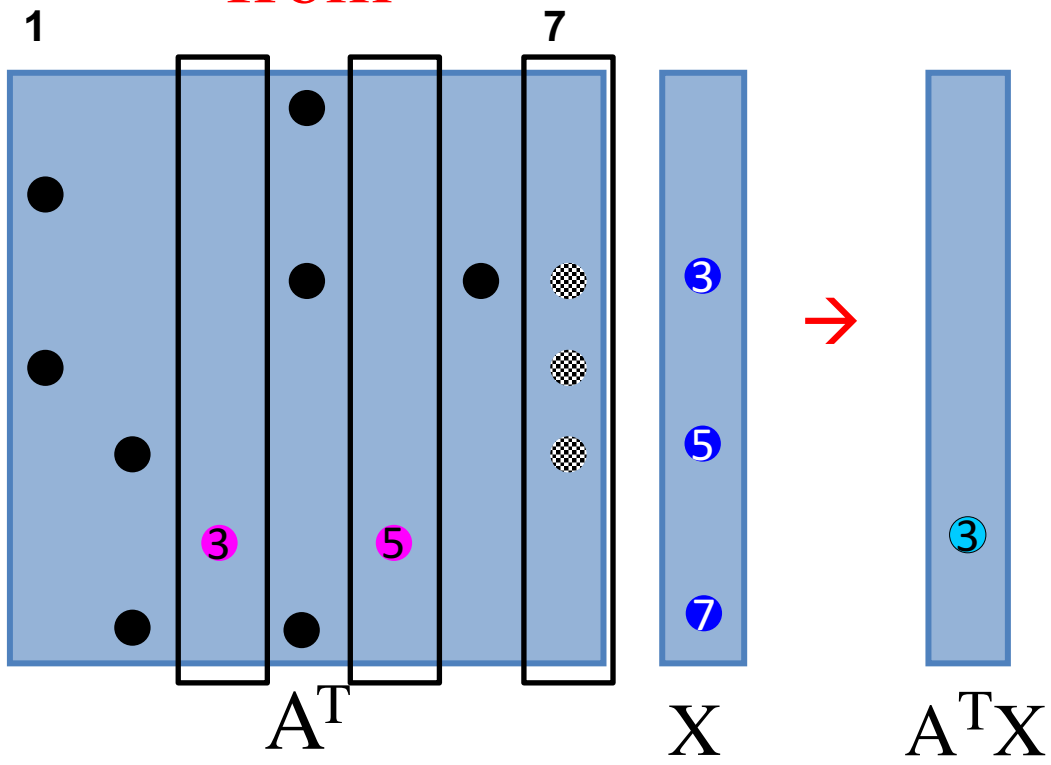


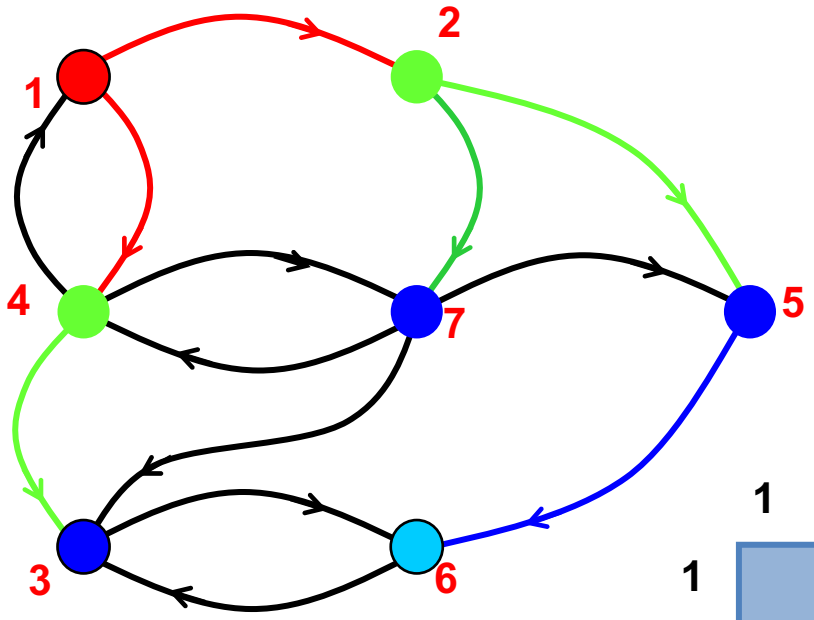

parents:




from

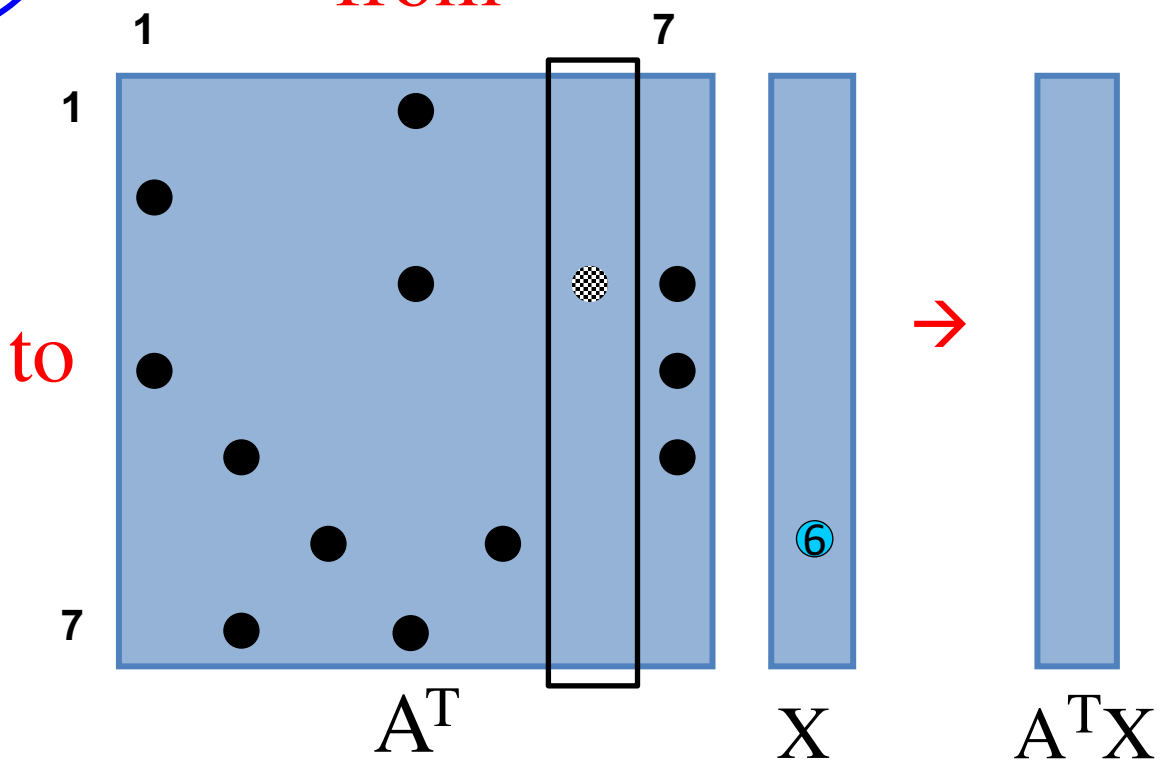
to






from

to

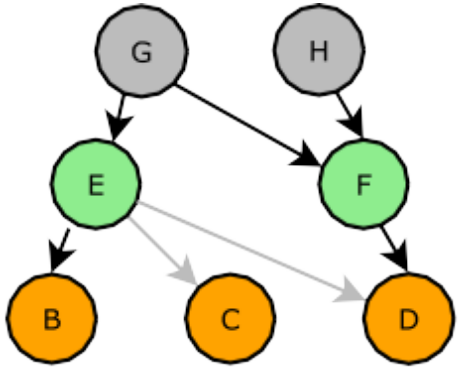


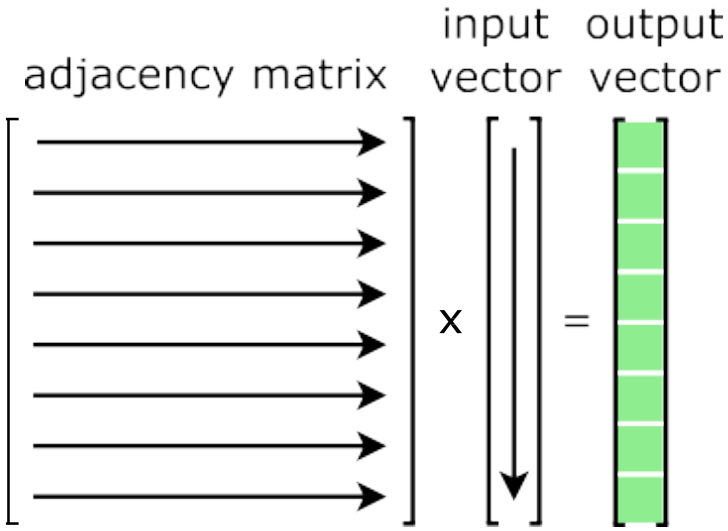
# BFS in GraphBLAS with Masks

```
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.
```

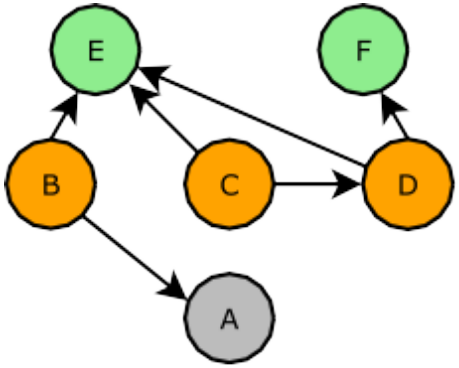
# Push-pull ≡ column-row matvec!

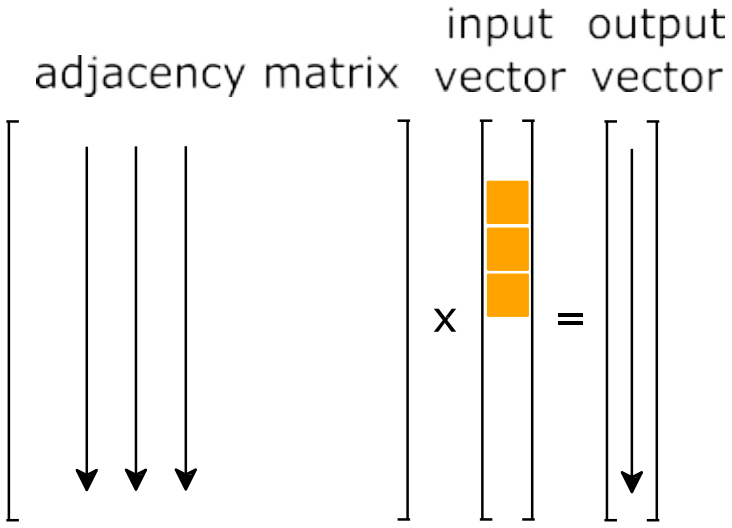
## Pull



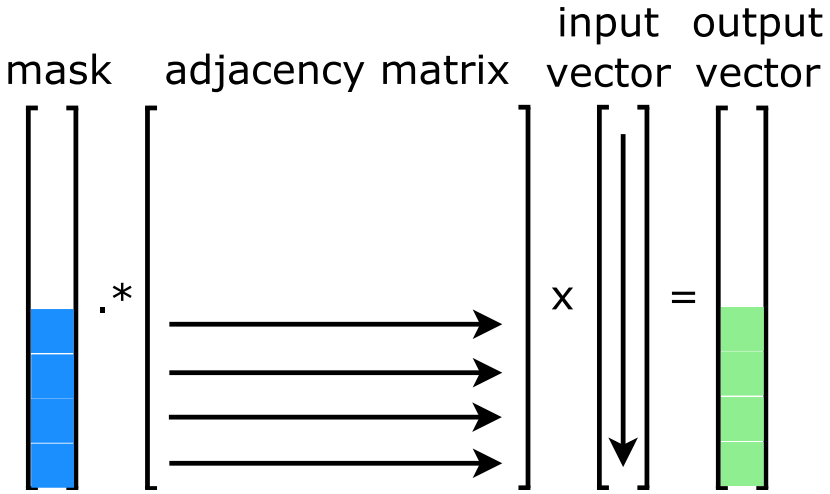

## Push



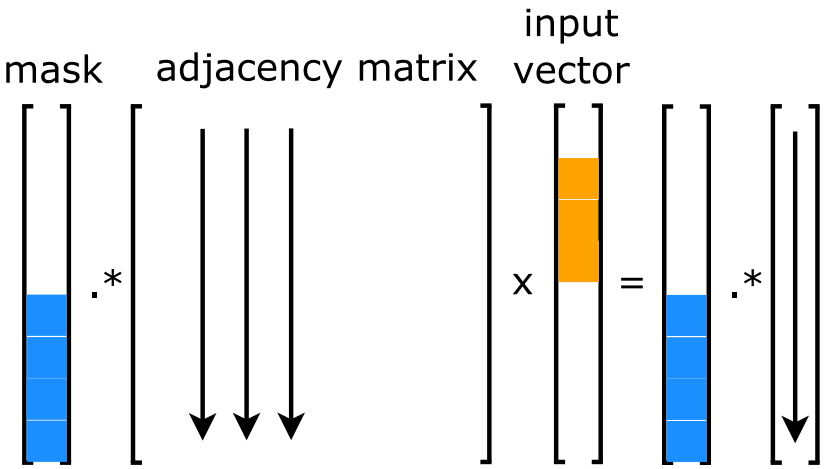




Yang, C., Buluc, A. and Owens, J.D., Implementing Push-Pull Efficiently in GraphBLAS. *ICPP'18*

# Masks make “pull” implementable in GraphBLAS




Row-based matvec w/ mask



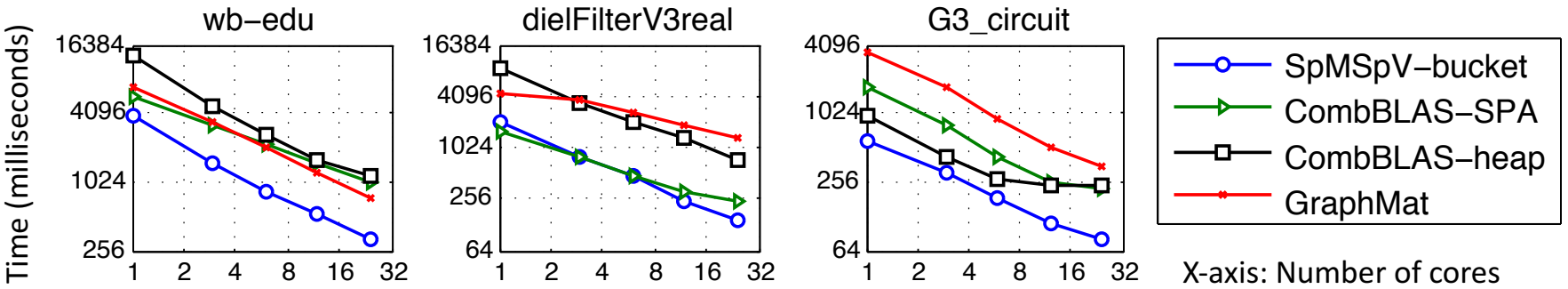

Column-based matvec w/ mask

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

# A work-efficient parallel algorithm for sparse matrix-sparse vector multiplication (SpMSpV)

- **Goal:** A scalable SpMSpV algorithm without doing more work on higher concurrency
- **Application:** Breadth-first search, graph matching, support vector machines, etc.
- **Algorithmic innovation:**
  - Attains work-efficiency by arranging necessary columns of the matrix into buckets where each bucket is processed by a single thread
  - Avoids synchronization by row-wise partitioning of the matrix on the fly
- **Performance:**
  - First ever work-efficient algorithm for SpMSpV that attains up to 15x speedup on a 24-core Intel Ivy Bridge processor and up to 49x speedup on a 64-core KNL processor
  - Up to an order of magnitude faster than its competitors, especially for sparser vector

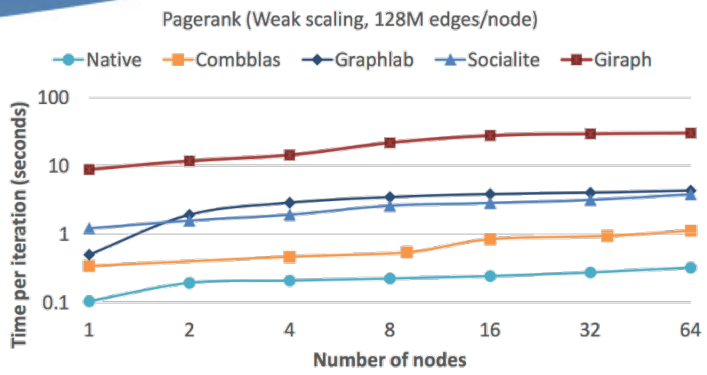



A.Azad, A. Buluç. A work-efficient parallel sparse matrix-sparse vector multiplication algorithm. IPDPS'17

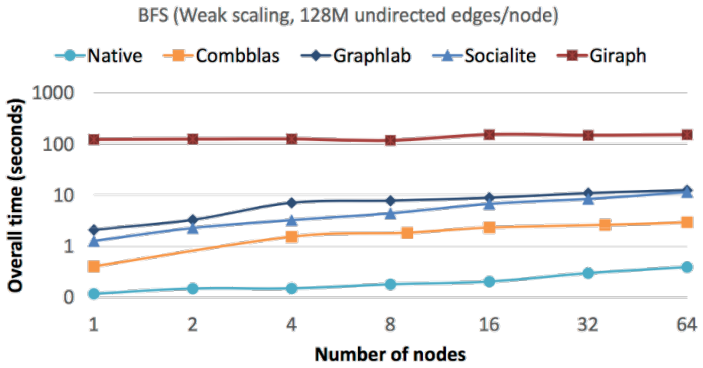
# Performance of Linear Algebraic Graph Algorithms

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

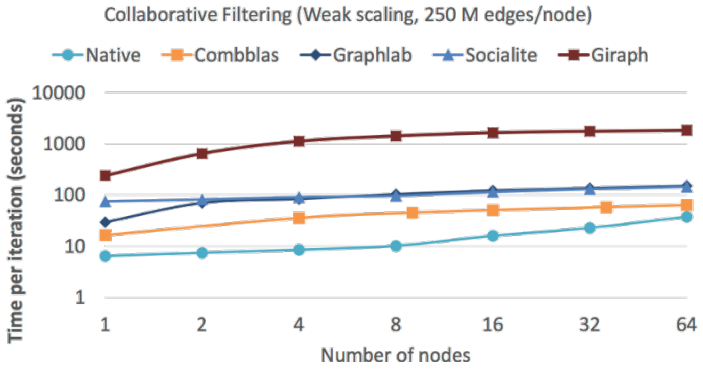
*The linear algebra abstraction enables high performance, within 4X of native performance for PageRank and Collaborative filtering.*



(a) PageRank



(b) Breadth-First Search



(c) Collaborative Filtering

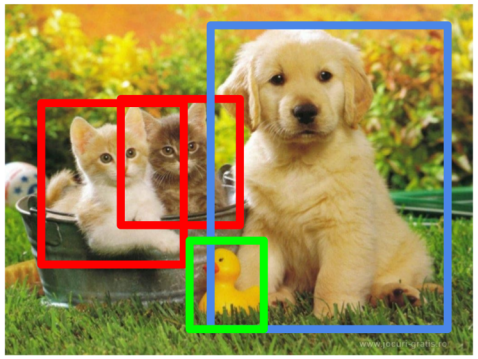
Satish, Nadathur, et al. "Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets", in SIGMOD'14

# Machine Learning for Science

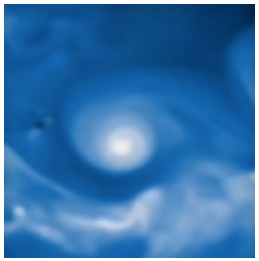
**Classification**

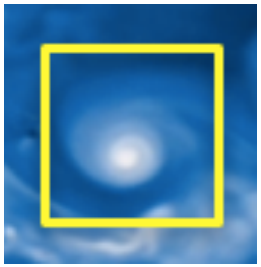
**Classification + Localization**

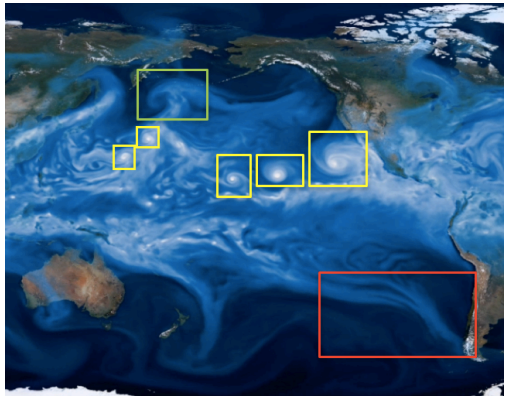
**Object Detection**

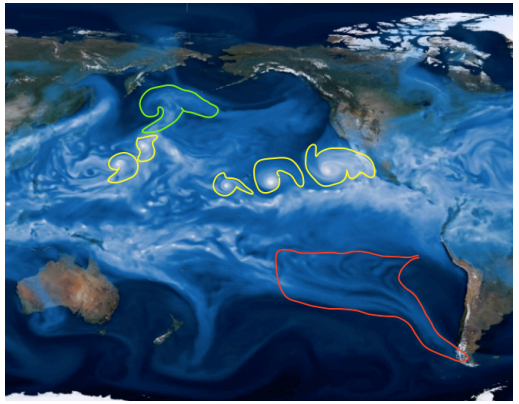


**Instance Segmentation**



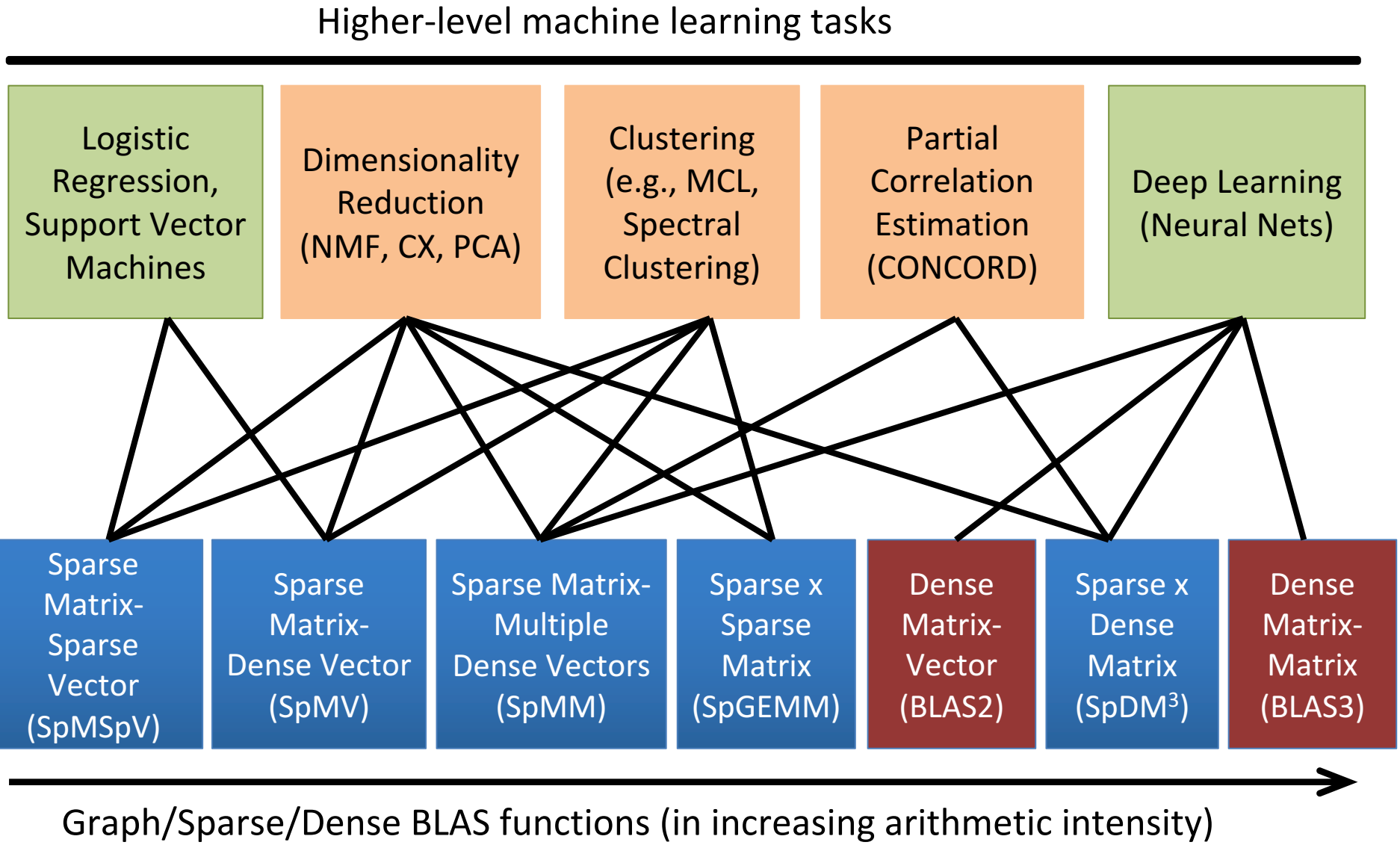






Slide source: Prabhat

# Machine Learning relies a lot on Linear Algebra



## Parallelism in Machine Learning

**Implicit Parallelization:** Keep the overall algorithm structure (the sequence of operations) intact and parallelize the individual operations.

Example: parallelizing the BLAS operations in previous figure

+ Often achieves exactly the same accuracy (e.g., model parallelism in DNN training)

- Scalability can be limited if the critical path of the algorithm is long

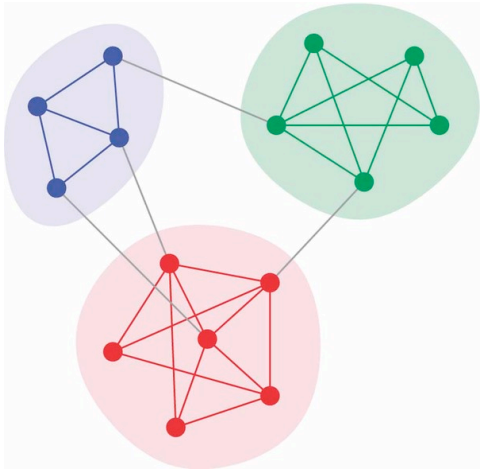
**Explicit Parallelization:** Modify the algorithm to extract more parallelism, such as working on individual pieces whose results can later be combined

Examples: CA-SVM and data parallelism in DNNs

+ Significantly better scalability can be achieved

- No longer the same algorithmic properties (e.g. HogWild!).

# Philosophy of the Markov Cluster Algorithm (MCL)



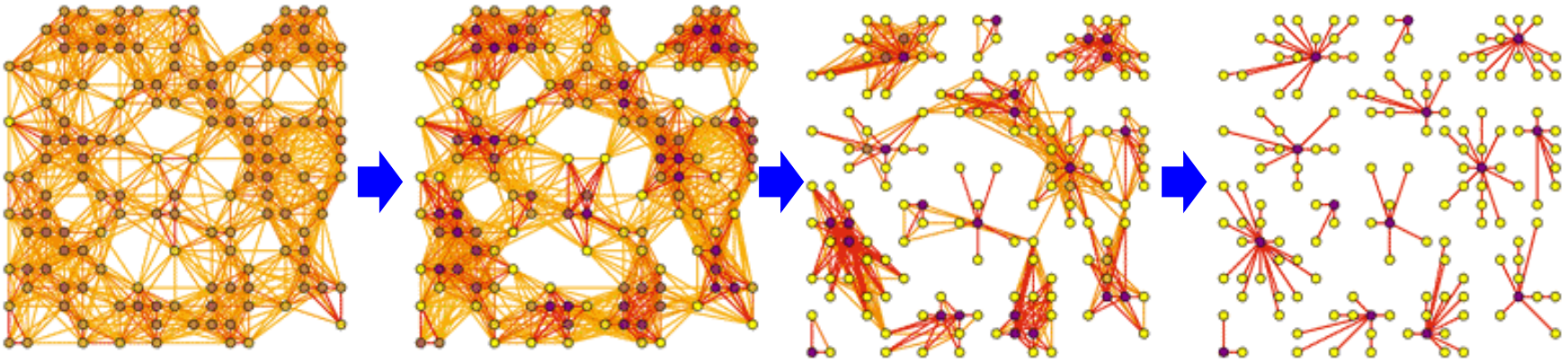
The number of **edges or higher-length paths** between two arbitrary nodes in a cluster is greater than the number of paths between nodes from different clusters

**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

# Markov Cluster Algorithm (MCL)

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



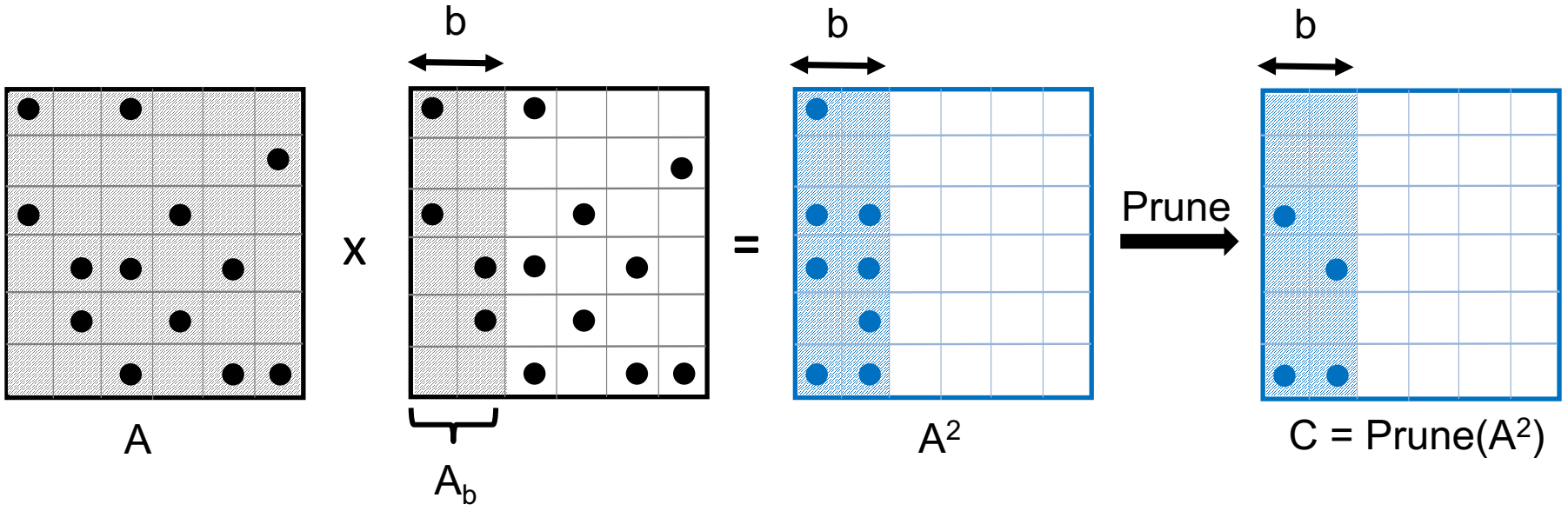

**At each iteration:**

**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

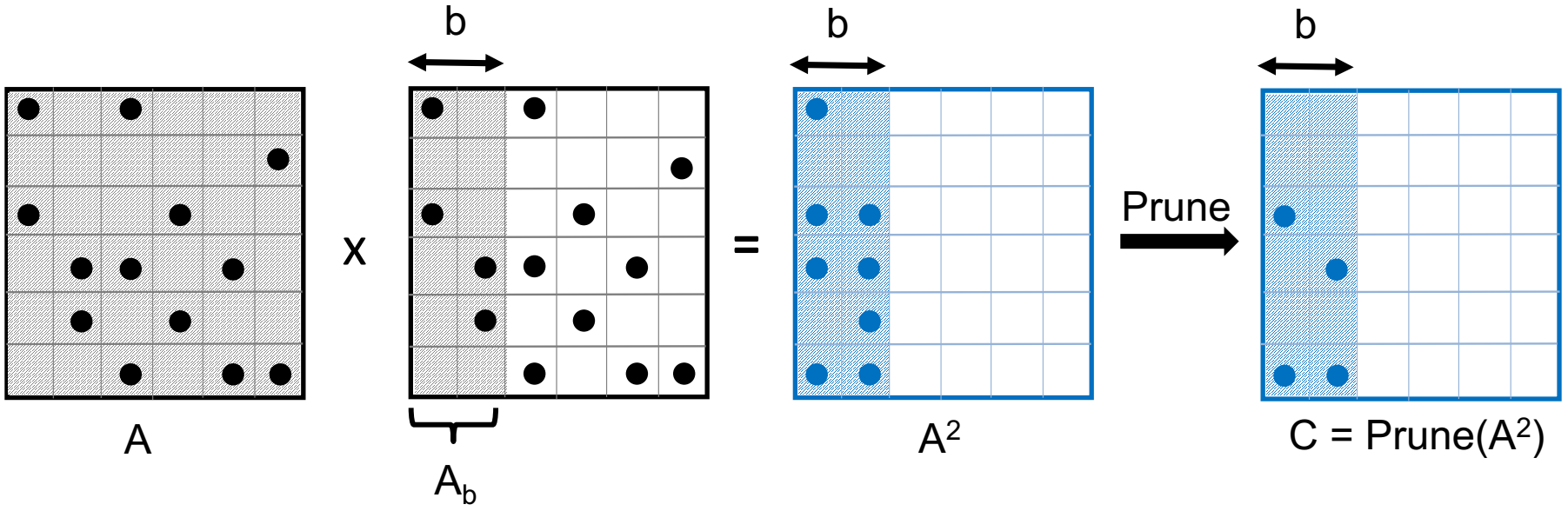
**Step 2** (Inflation) : taking powers entry-wise

# 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

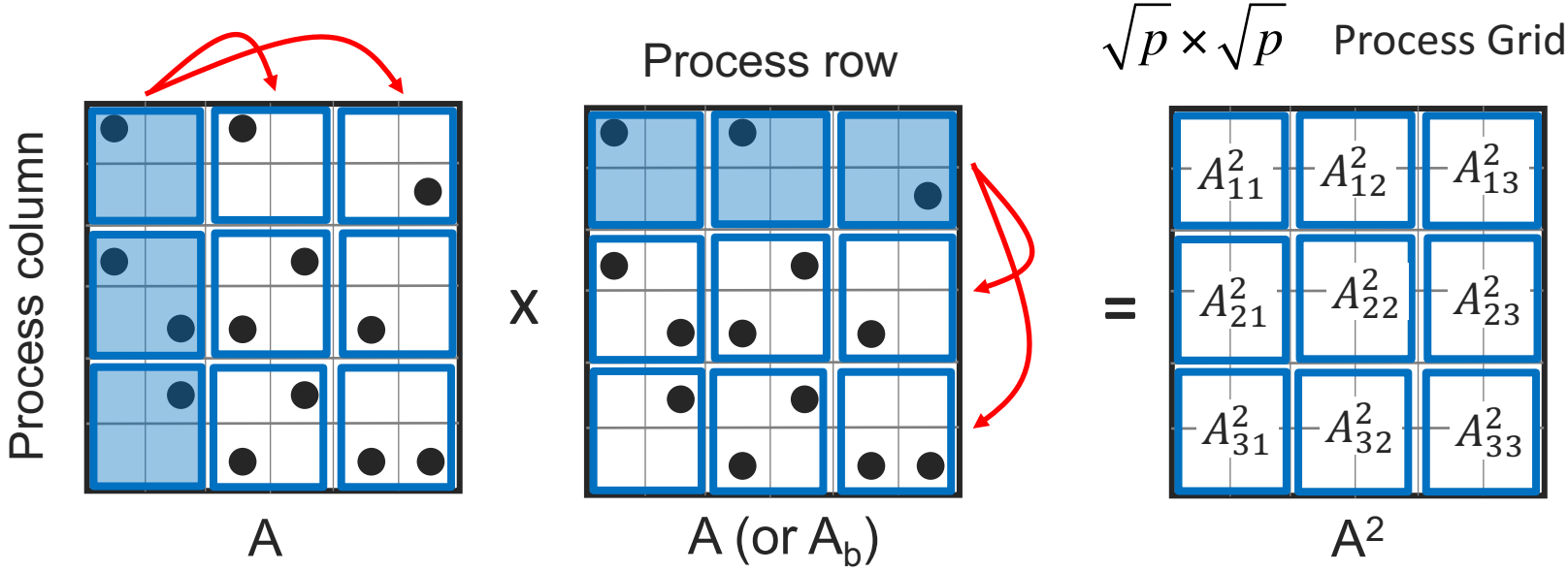
# A combined expansion and pruning step




- 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*

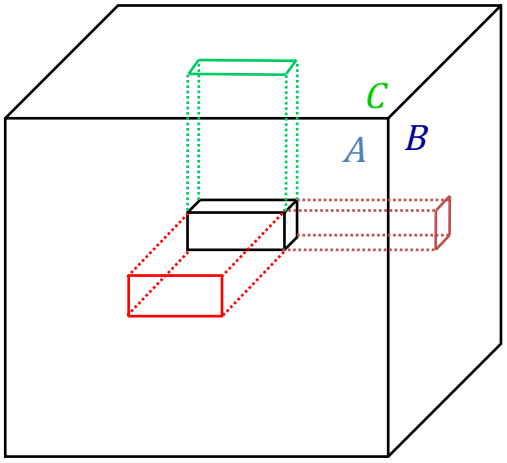
# HipMCL on large networks

| 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

# The computation cube of matrix-matrix multiplication

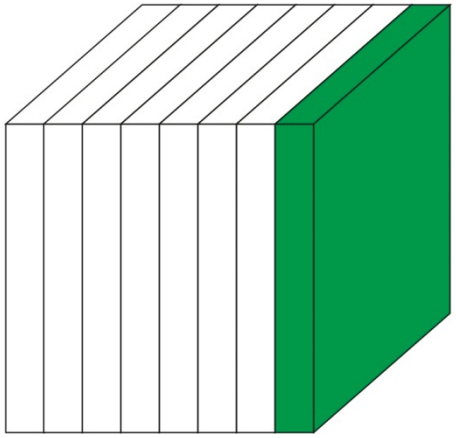
Matrix multiplication: $\forall(i,j) \in n \times n, \quad C(i,j) = \Sigma_k A(i,k)B(k,j),$



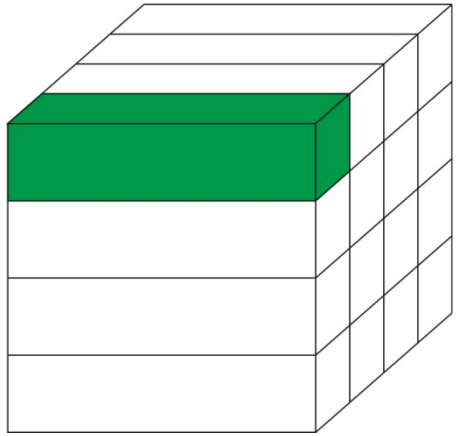

The *computation (discrete) cube*:

- A face for each (input/output) matrix
- A grid point for each multiplication

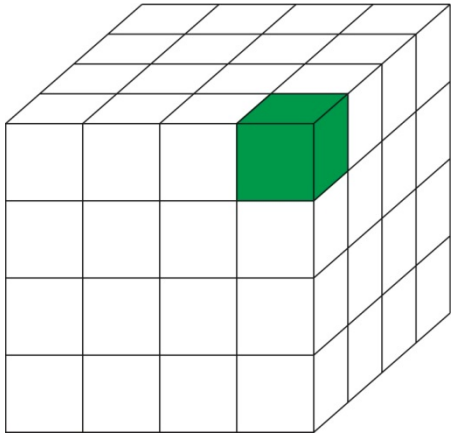
How about sparse algorithms?



1D algorithms

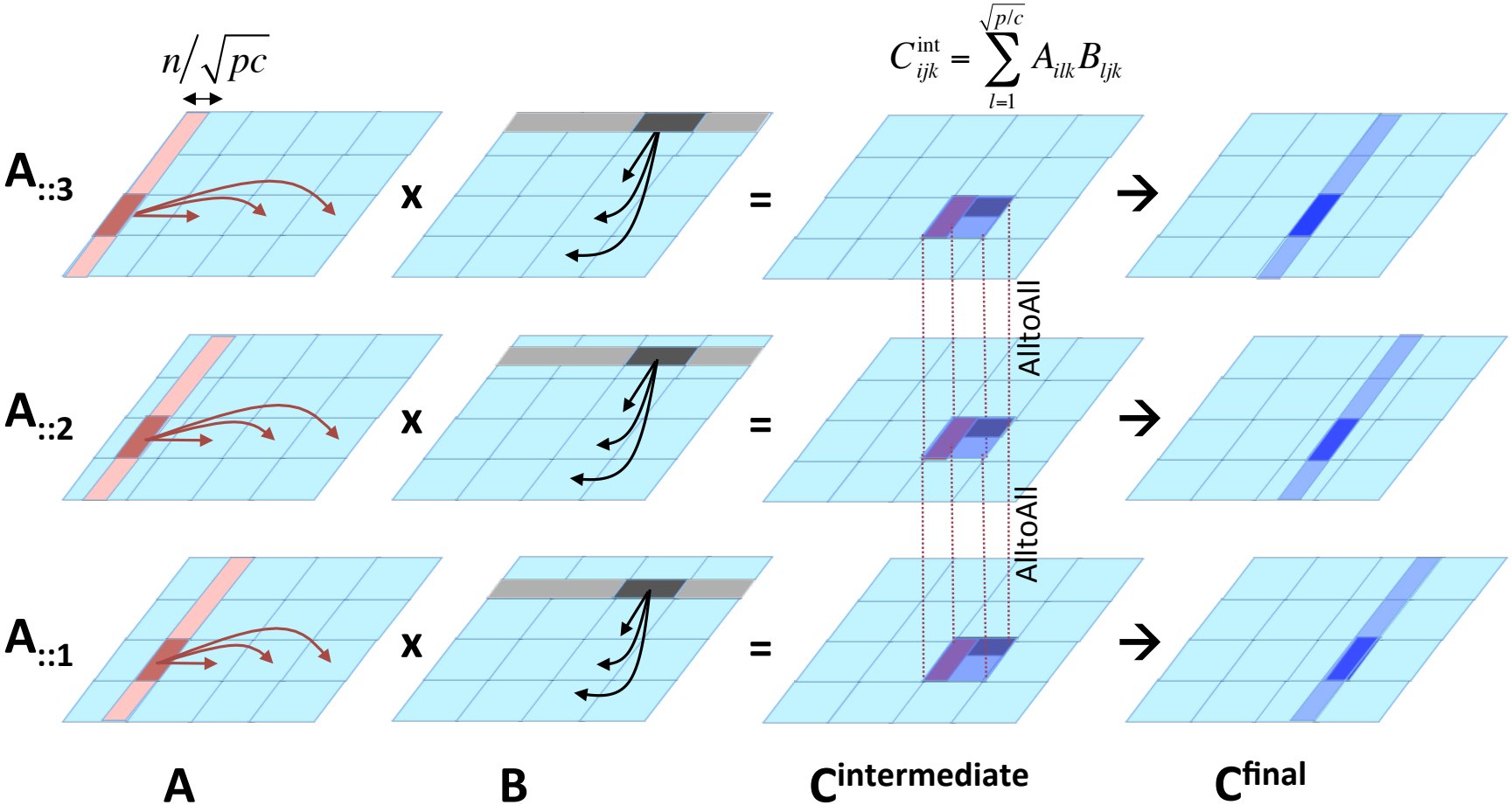


2D algorithms



3D algorithms

# 3D parallel SpGEMM in a nutshell



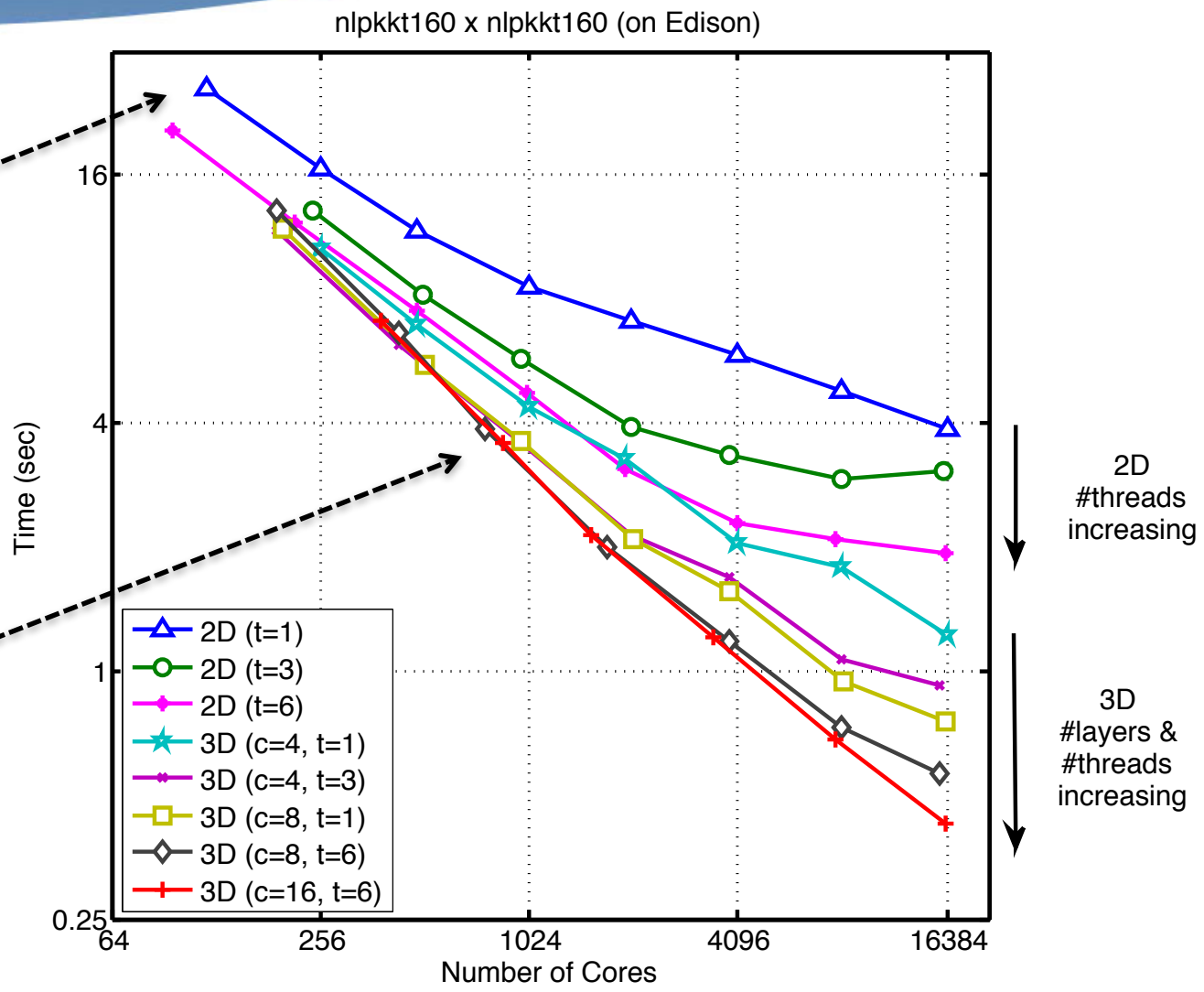


Azad, A., Ballard, G., Buluc, A., Demmel, J., Grigori, L., Schwartz, O., Toledo, S. and Williams, S., 2016. Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication. *SIAM Journal on Scientific Computing*, *38*(6), pp.C624-C651.

# 3D SpGEMM performance

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

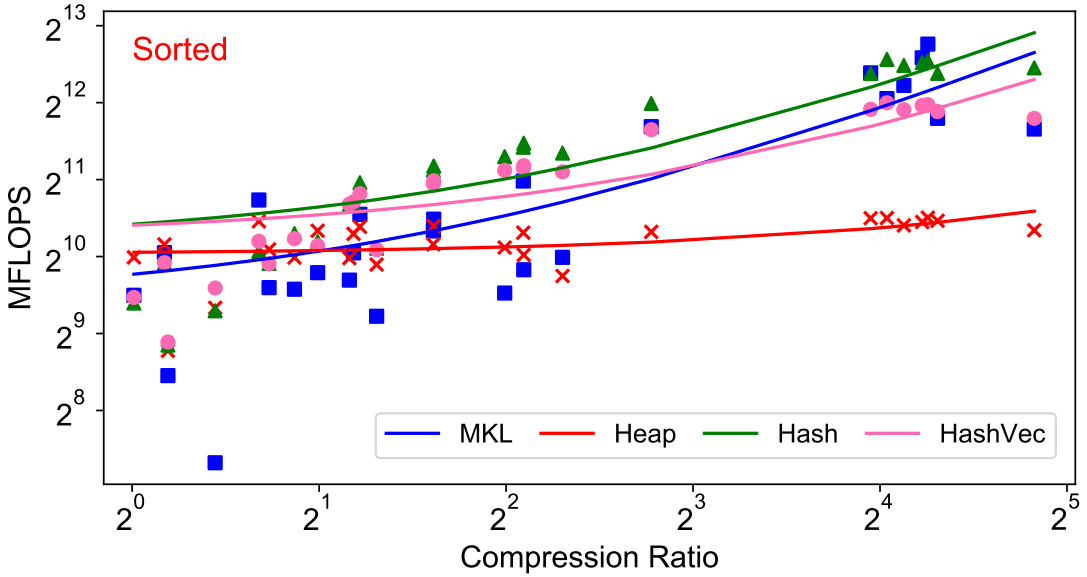
3D (threaded) – first presented here – beats it by 8X at large concurrencies

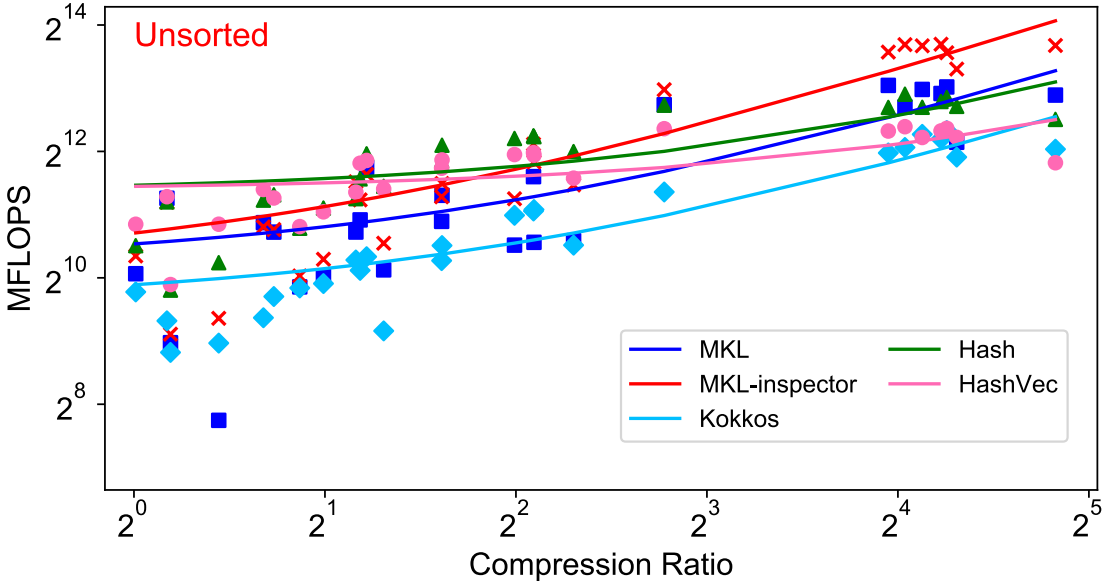



Strong scaling of different variants of 2D and 3D algorithms when squaring of nlpkkt160 matrix on Edison.

# 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.

## Sparse Inverse Covariance Matrix Estimation

- **Precision matrix = Inverse covariance matrix**
- **Goal**: Estimating graphical model structure
- "The zeros of a precision matrix correspond to zero partial correlation, a necessary and sufficient condition for conditional independence (Lauritzen, 1996)"
- Sparsity often enforced by regularization
- One algorithm (HP-CONCORD)'s objective function:

$$\underset{\Omega \in \mathbf{R}^{p \times p}}{\text{minimize}} \; -\log\det(\Omega_D^2) + \mathbf{tr}(\Omega S \Omega) + \lambda_1 \|\Omega_X\|_1 + \frac{\lambda_2}{2}\|\Omega\|_F^2,$$

- Ω is the sparse inverse covariance matrix we are trying to estimate

# Why care? Finding Direct Associations

**Partial Correlation (a.k.a. sparse inverse covariance estimation): direct association without confounders**

- Gene Regulatory Network (GRN) estimation
- Joint modeling of SNPs and GRN
- Linkage Disequilibrium (LD) estimation
- Canonical Correlation Analysis (CCA)
- Genome-wide association studies (GWAS)

**Data-driven hypothesis generation!**

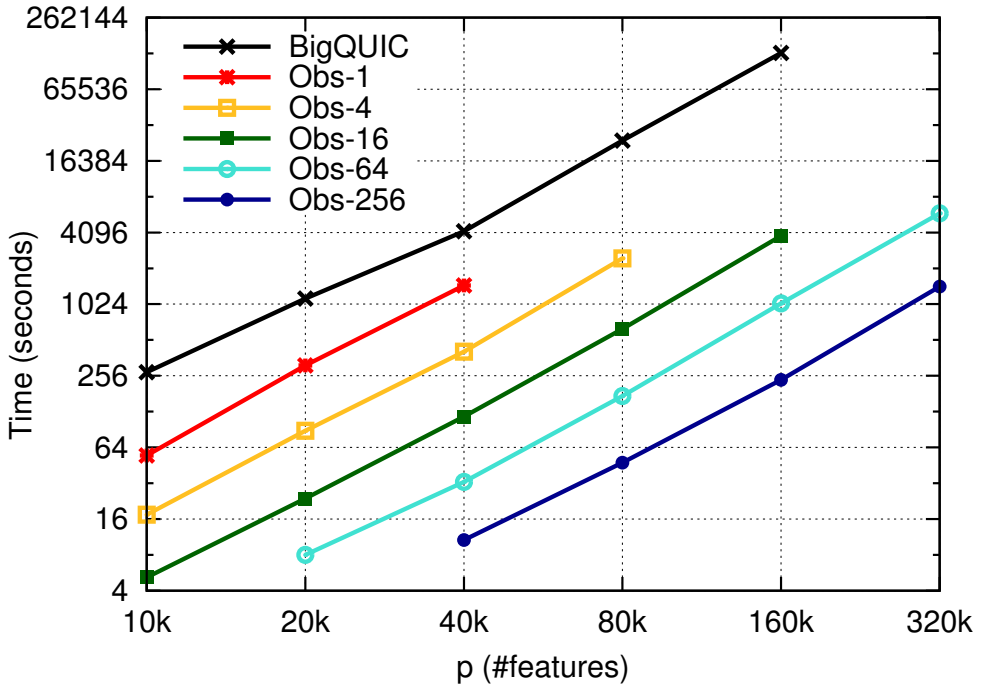
**Fig. 1.** Conditionally on the height of snow, the number of snowmen is independent of the intensity of traffic jams. This is represented by a two edges graph.

- Computationally challenging;
- **HP-CONCORD** on distributed memory increases scalability

# HP-CONCORD Advantages

- HP-CONCORD makes fewer assumptions about the data (in particular, no Gaussianity is assumed) compared to competitors
- Thanks to **communication-avoiding matrix algorithms**, it reaches unprecedented scales

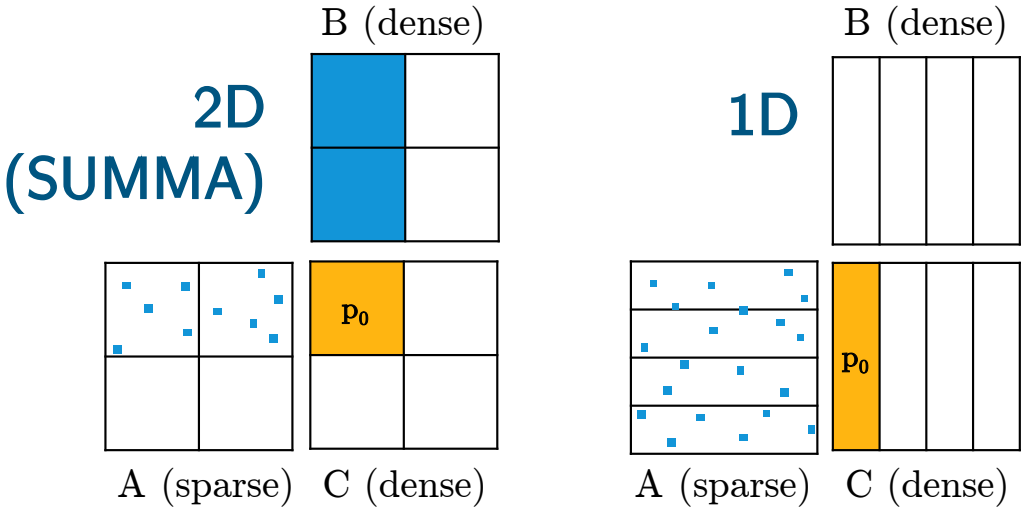
  - BigQUIC: previous state-of-the-art
  - Obs-K are our HP-CONCORD algorithm (K: number of nodes)
  - Experiment is trying to recover a random graph structure.

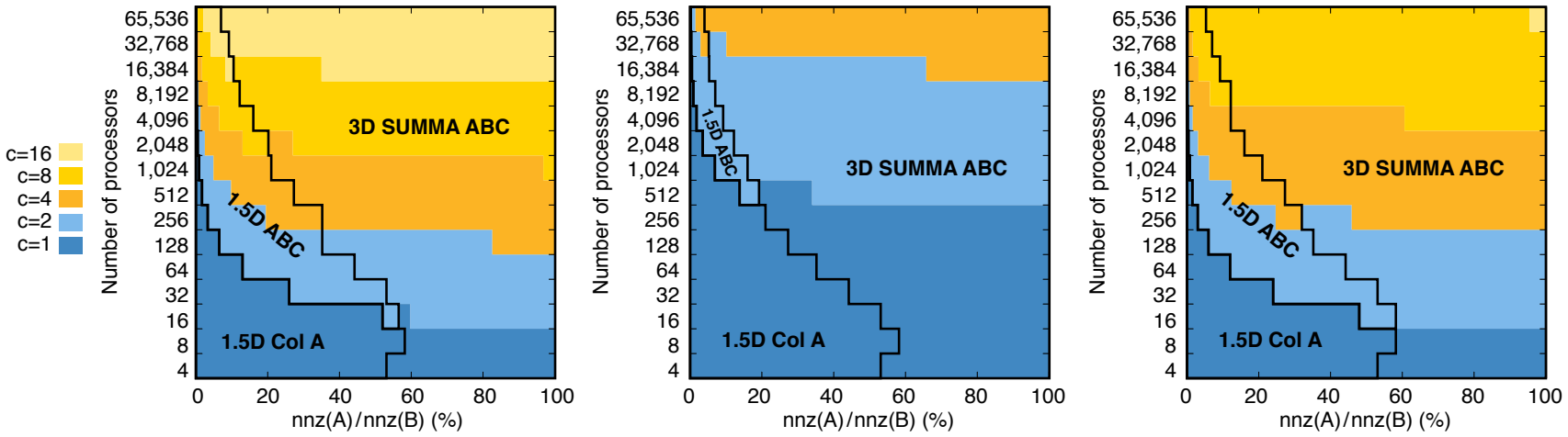



P. Koanantakool, A. Ali, A. Azad, A. Buluç, D. Morozov, S. Oh, L.Oliker, and K. Yelick "Communication-Avoiding Optimization Methods for Massive-Scale Graphical Model Structure Learning". In: International Conference on Artificial Intelligence and Statistics (AISTATS). 2018.

# SpDM$^3$: Sparse x Dense Matrix: The workhorse of HP-CONCORD

- The best algorithm when multiplying two matrices with unequal nonzero counts?
- Depends on the concurrency!

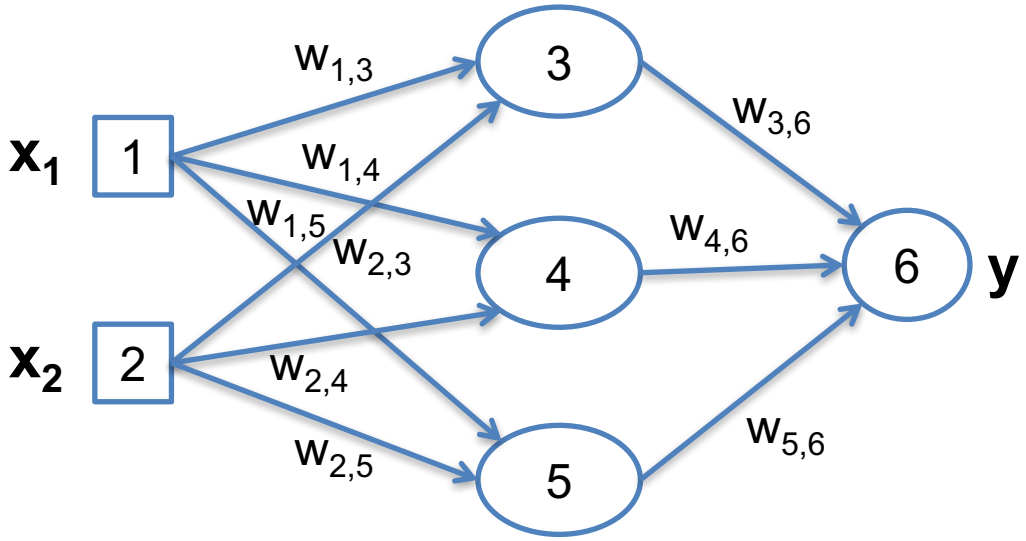



Koanantakool, Penporn, et al. "Communication-avoiding parallel sparse-dense matrix-matrix multiplication." IPDPS, 2016

# Training Neural Networks

- Training is to **adjust the weights (W)** in the connections of the neural network, in order to change the function it represents.




Only parameters are weights for simplicity (i.e. ignore bias parameters)

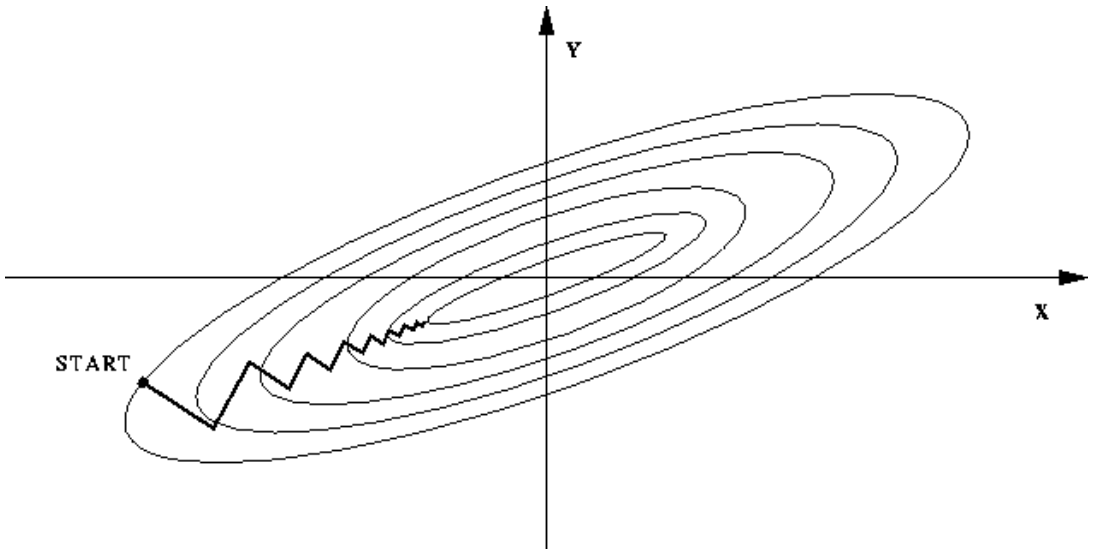
**W:** the matrix of weights

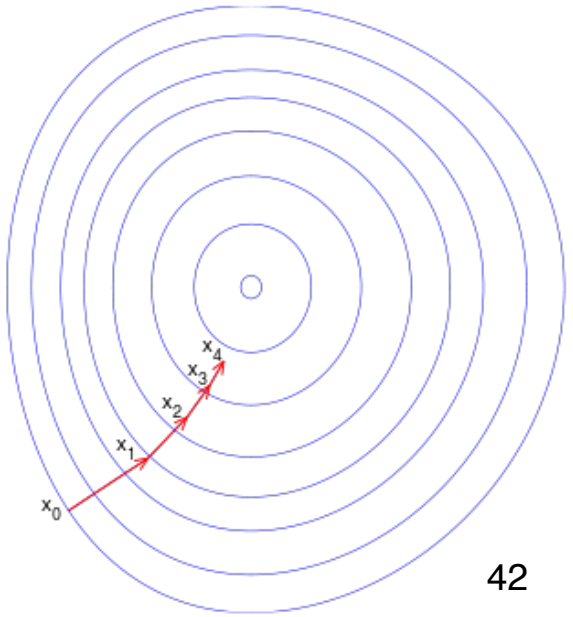
A "shallow" neural network with only one hidden layer (nodes 3,4,5), two inputs and one output.

# Gradient Descent

$$W^{t+1} \leftarrow W^t - \alpha \cdot \nabla_W f(W^t, x)$$

- Also called the steepest descent algorithm
- In order to minimize a function, move towards the opposite direction of the gradient at a rate of α.
- α is the step size (also called the learning rate)
- Used as the ***optimization backend*** of many other machine learning methods (example: NMF)



# Stochastic Gradient Descent (SGD)

$$\text{Assume } f(W^t, x) = \frac{1}{n}\sum_{i=1}^{n} f_i(W^t, x)$$

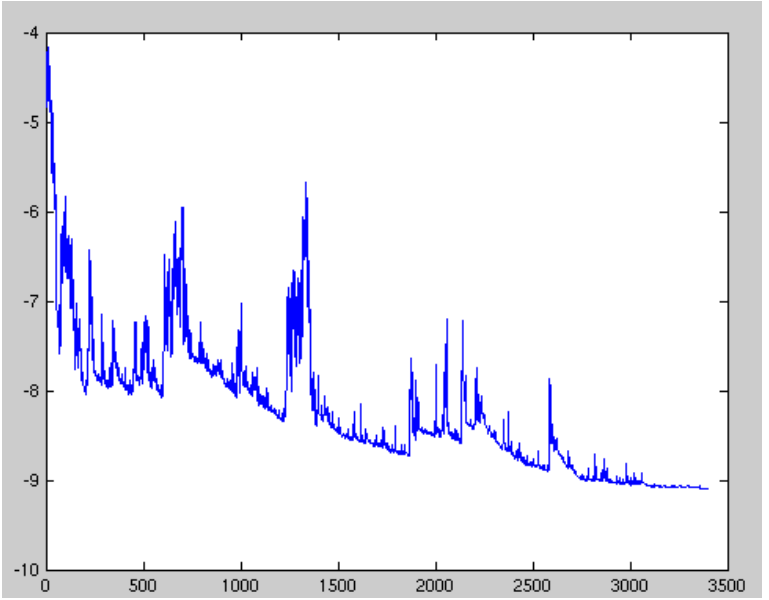
$$W^{t+1} \leftarrow W^t - \alpha \cdot \nabla_W f_i(W^t, x)$$

Pure SGD: compute gradient using 1 sample

$$W^{t+1} \leftarrow W^t - \alpha \cdot \frac{1}{b}\sum_{i=k+1}^{k+b} \nabla_W f_i(W^t, x)$$

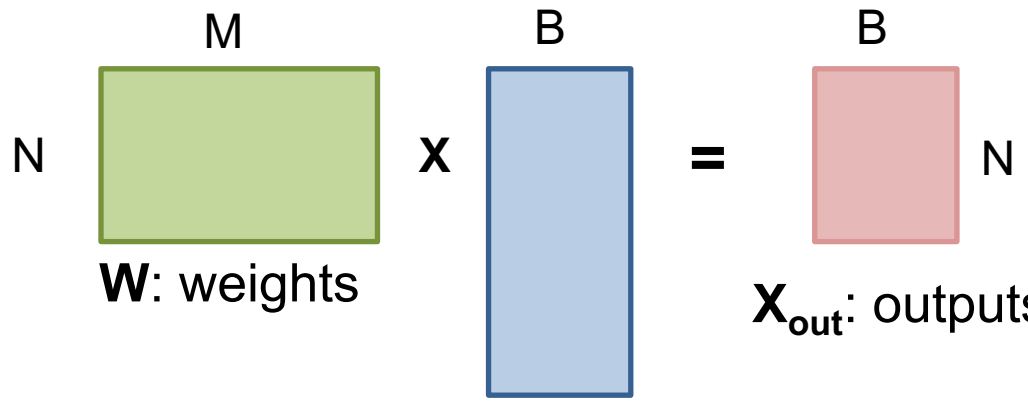
Mini-batch: compute gradient using b samples

f is not going down for every iteration




- Actually the name is a misnomer, *this is not a "descent" method*
- But we will stick to it anyway to avoid confusion.
- Performance and parallelism requires batch training
- Larger batch sizes hurt convergence as they get trapped easily
- SGD escapes sharp local minima due to its "noisy" gradients

# SGD training of NNs as matrix operations

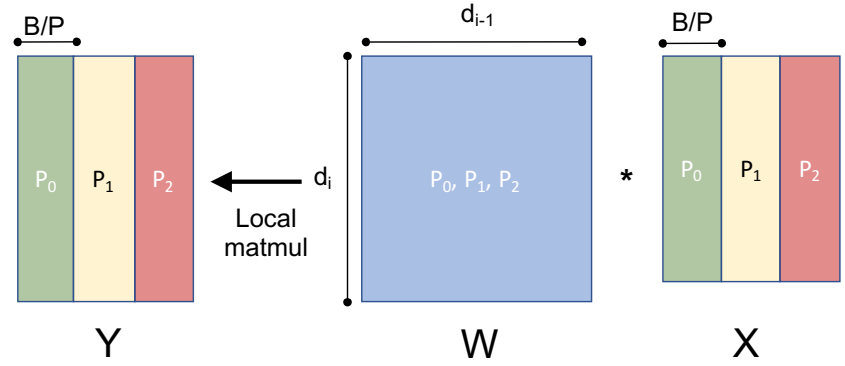



N = the number of outputs
M = the number of inputs
B = the size of the minibatch

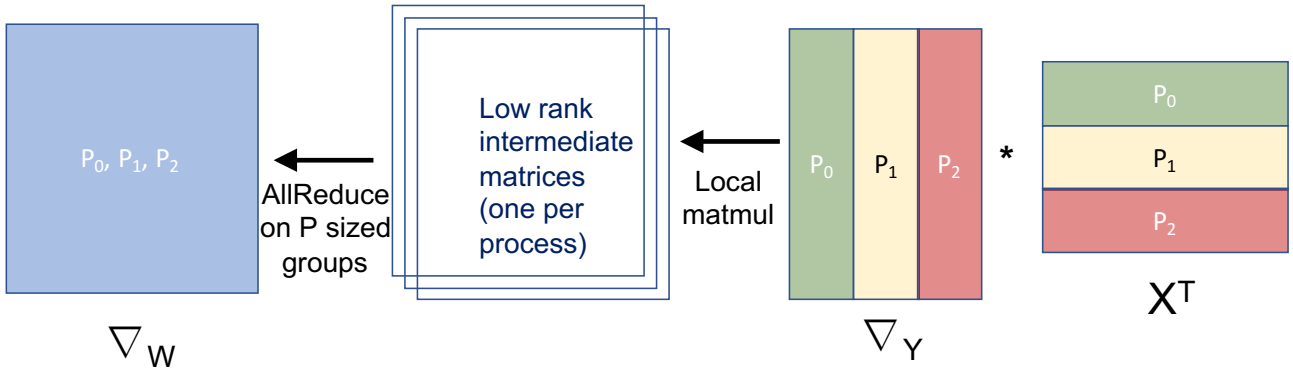
**The impact to parallelism:**

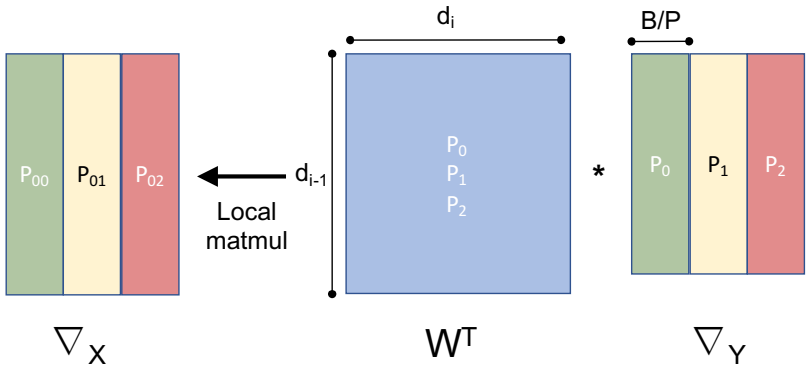
- **W** is replicated to processor, so it doesn't change
- $\mathbf{X_{in}}$ and $\mathbf{X_{out}}$ gets skinnier if we only use data parallelism, i.e. distributing **b=B/p** mini-batches per processor
- GEMM performance suffers as *matrix dimensions get smaller* and *more skewed*
- **Result:** Data parallelism can hurt single-node performance

# Data Parallel SGD training of NNs as matrix operations




1. Which matrices are replicated?
2. Where is the communication?
3. Which steps can be overlapped?



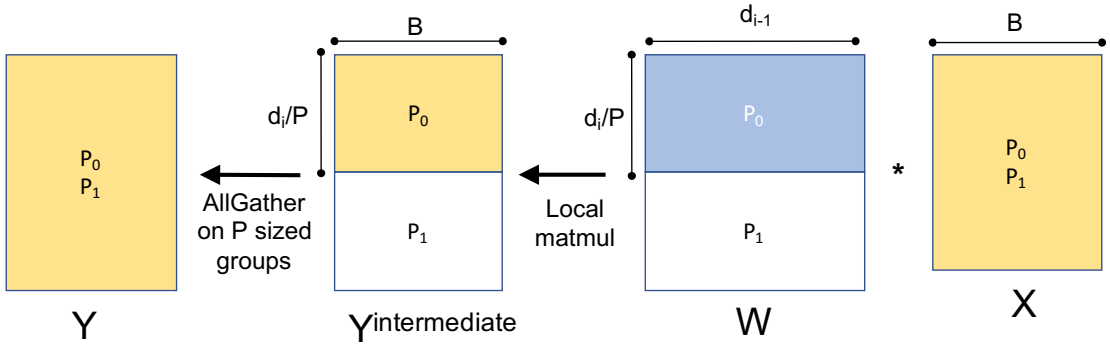



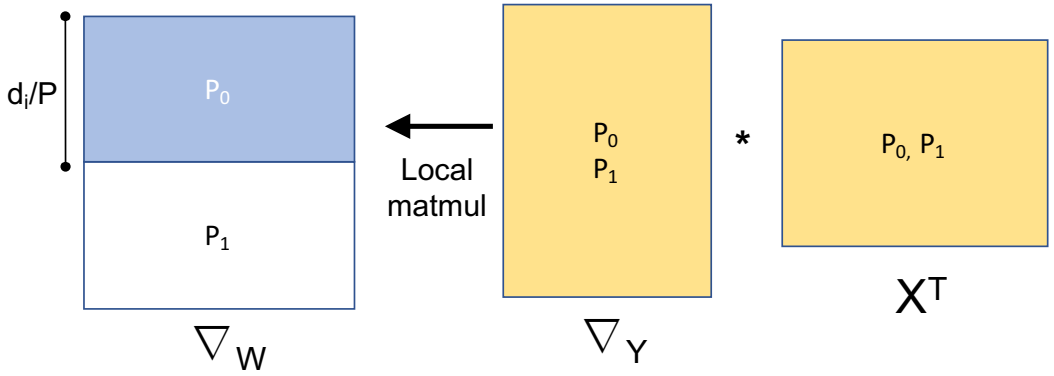

$\nabla_Y = \partial L / \partial Y$ = how did the loss function change as output activations changed?

$\nabla_X = \partial L / \partial X$

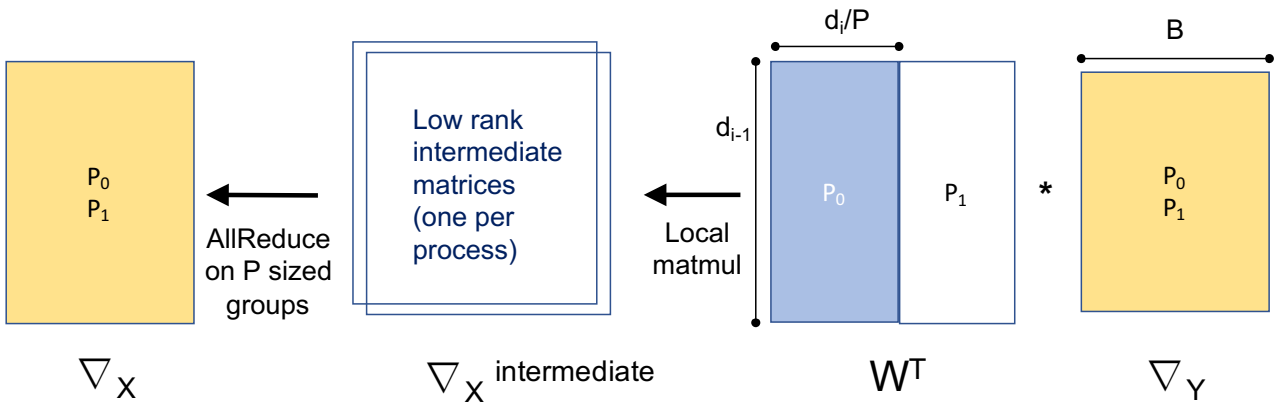
$\nabla_W = \partial L / \partial W$

# Model Parallel SGD training of NNs as matrix operations



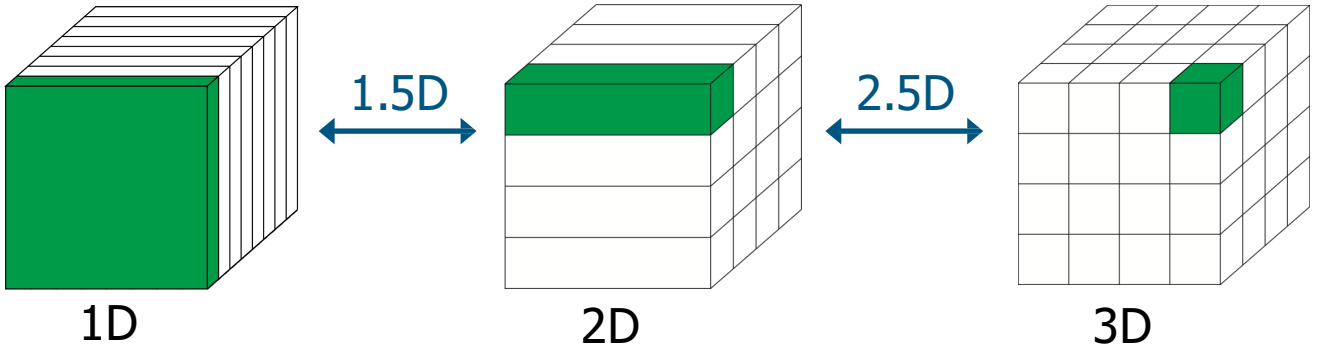

1. Which matrices are replicated?
2. Where is the communication?
3. How can matrix algebra capture both model and data parallelism?



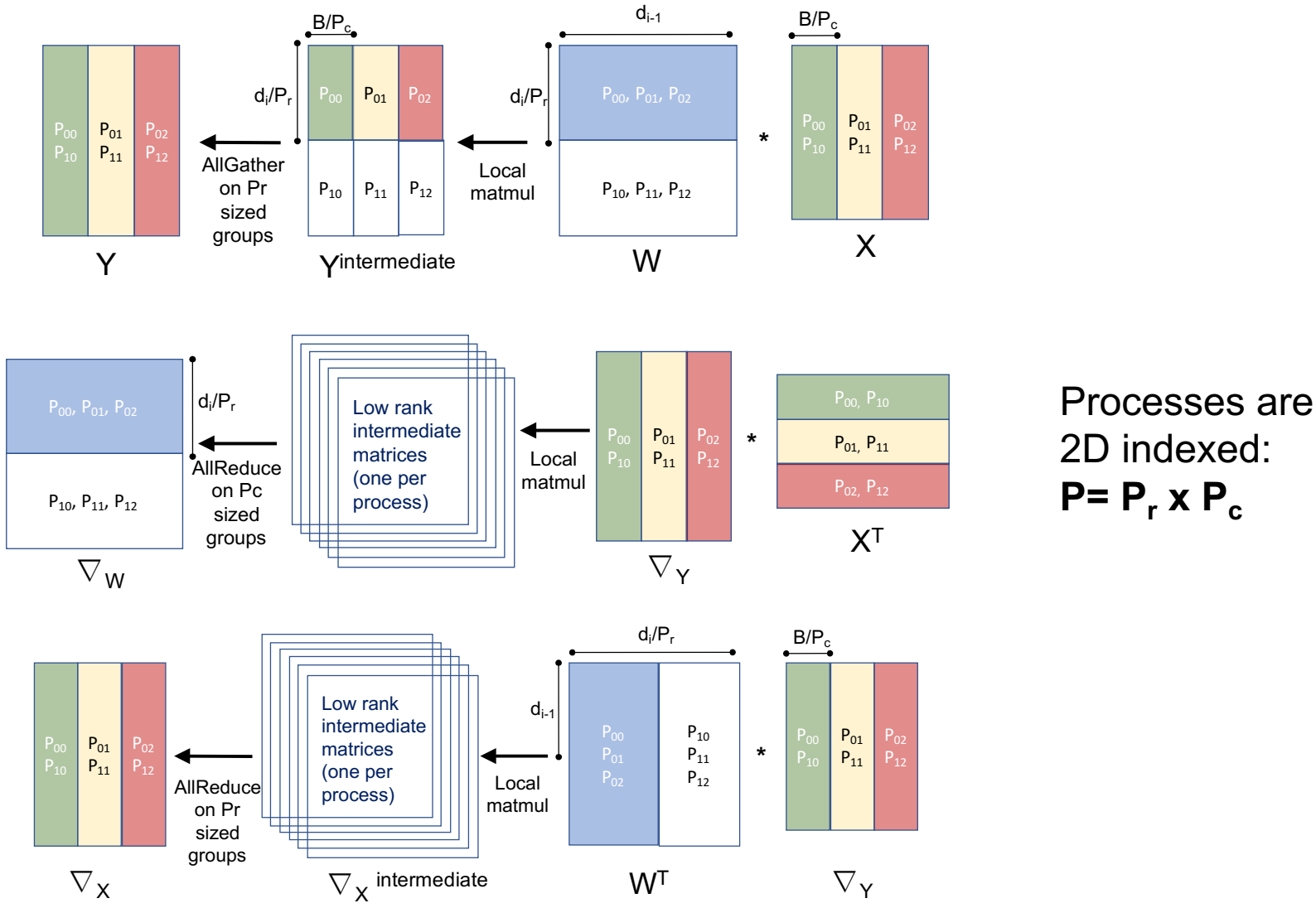
# Combinations of various parallelism opportunities

- There are various different ways to combine DNN training parallelism opportunities.
  - It helps to think in terms of matrices again.
- We will exploit ***communication-avoiding matrix algorithms***; which trade off some storage (judicious replication) at the expense of reduced communication.
  - Deep Learning community is already OK with data or model replication in many cases

## A succinct classification of parallel matrix multiplication algorithms



# Data & Model Parallel SGD training of NNs as matrix operations

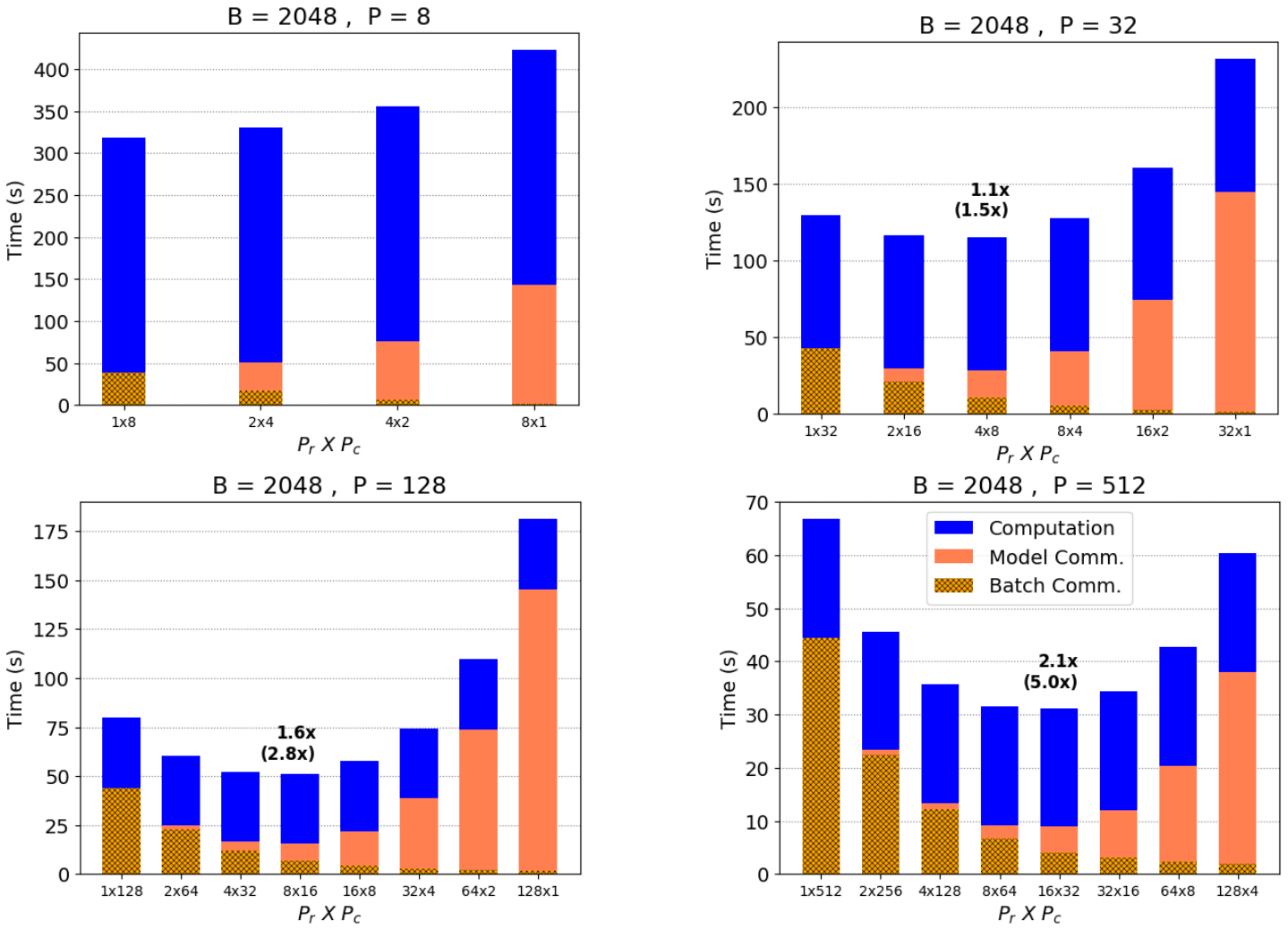



Processes are 2D indexed:
**$P = P_r \times P_c$**

A. Gholami, A. Azad, P. Jin, K. Keutzer, A. Buluç. Integrated Model, Batch, and Domain Parallelism in Training Neural Networks. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'18)

# Integrated Batch + Model Scaling

- **For large processes integrated could provide up to 2x speedup**



# Conclusions

- Both graph algorithms and machine learning have growing importance in scientific applications
- Not everything is [sparse] linear algebra, but a lot of things are!
- Transfer of techniques and knowledge is easier when your scientific base is not domain specific
- Communication-avoiding [sparse] linear algebra algorithms provide **unprecedented scaling** for problems outside traditional scientific computing, such as computational biology, graph analysis, and machine learning.
- Check out http://graphblas.org, HipMCL, and HP-CONCORD

My lab website: http://passion.lbl.gov

# Acknowledgments

Alnur Ali, Ariful Azad, David Bader, Grey Ballard, Scott Beamer, James Demmel, Amir Gholami, John Gilbert, Giulia Guidi, Laura Grigori, Peter Jin, Jeremy Kepner, Kurt Keutzer, Penporn Koanantakool, Nikos Kyrpides, Tim Mattson, Scott McMillan, Dmitriy Morozov, Jose Moreira, Yusuke Nagasaka, Sang-Yun Oh, Lenny Oliker, John Owens, Christos Ouzounis, Georgios Pavlopoulos, Prabhat, Dan Rokhsar, Oded Schwartz, Sivan Toledo, Sam Williams, Carl Yang, Kathy Yelick.

My work is funded by

U.S. DEPARTMENT OF ENERGY | Office of Science