Sparse Matrices Beyond Solvers - Graphs, Biology, and Machine Learning

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

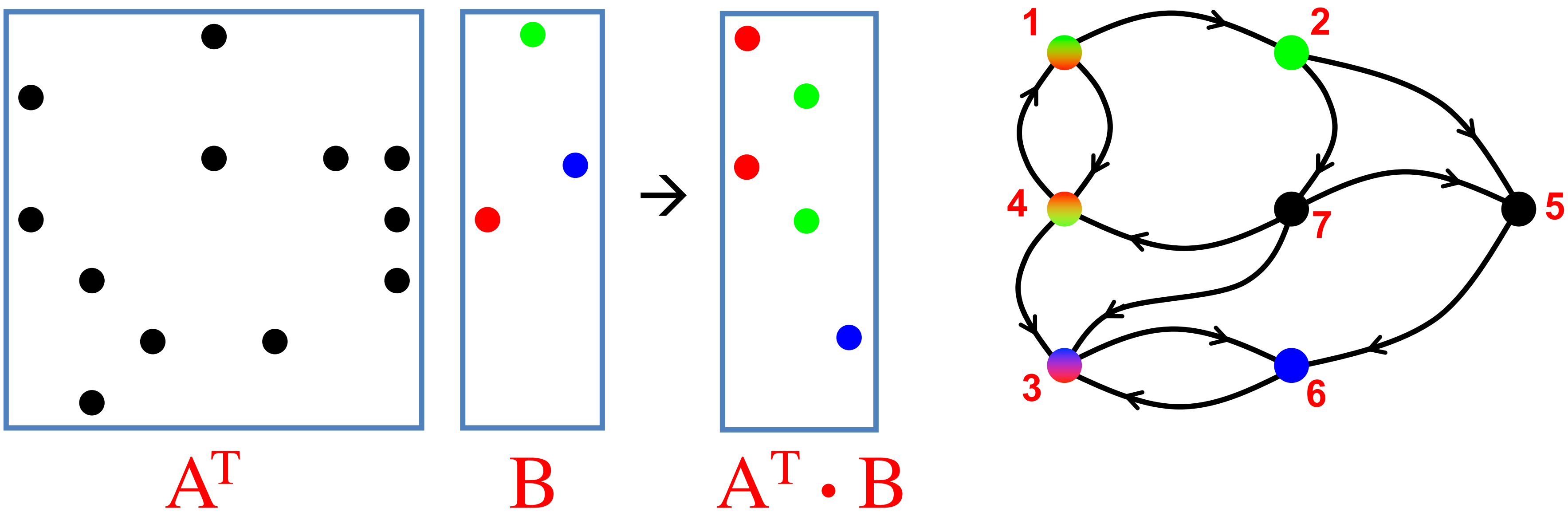
2020 SIAM Conference on
Mathematics of Data Science

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

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

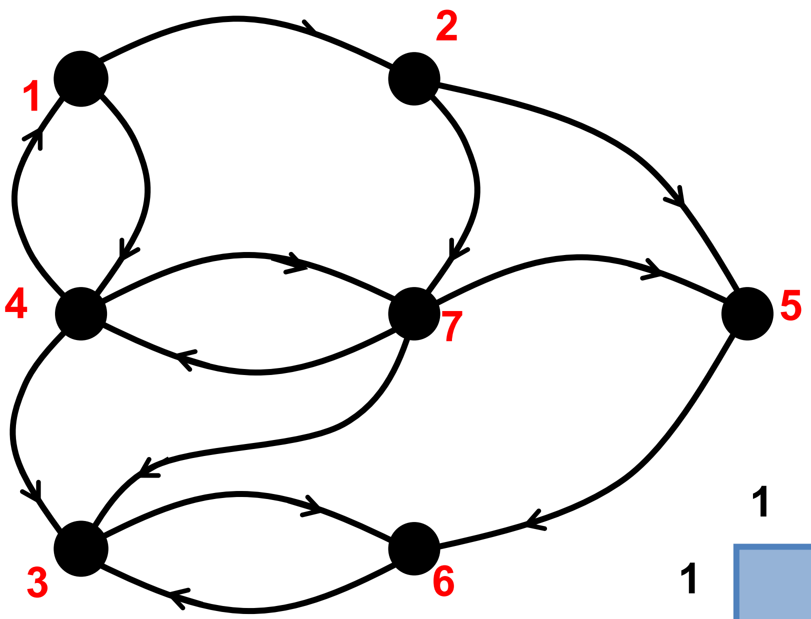
- **Goal:** A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an actual Application Programming Interface (API) that
  i. is faithful to the mathematics as much as possible, and
  ii. enables efficient implementations on modern hardware.
- **Impact:** All graph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

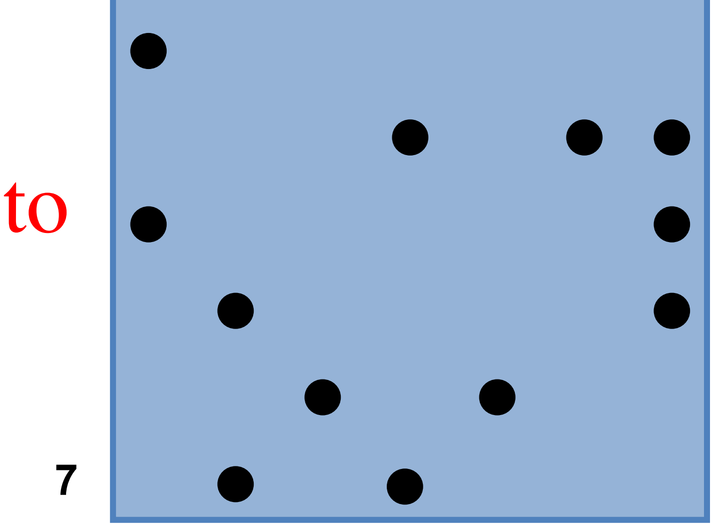
```
GrB_info GrB_mxm(GrB_Matrix                *C,      // destination
                 const GrB_Matrix           Mask,
                 const GrB_BinaryOp         accum,
                 const GrB_Semiring         op,
                 const GrB_Matrix           A,
                 const GrB_Matrix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

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

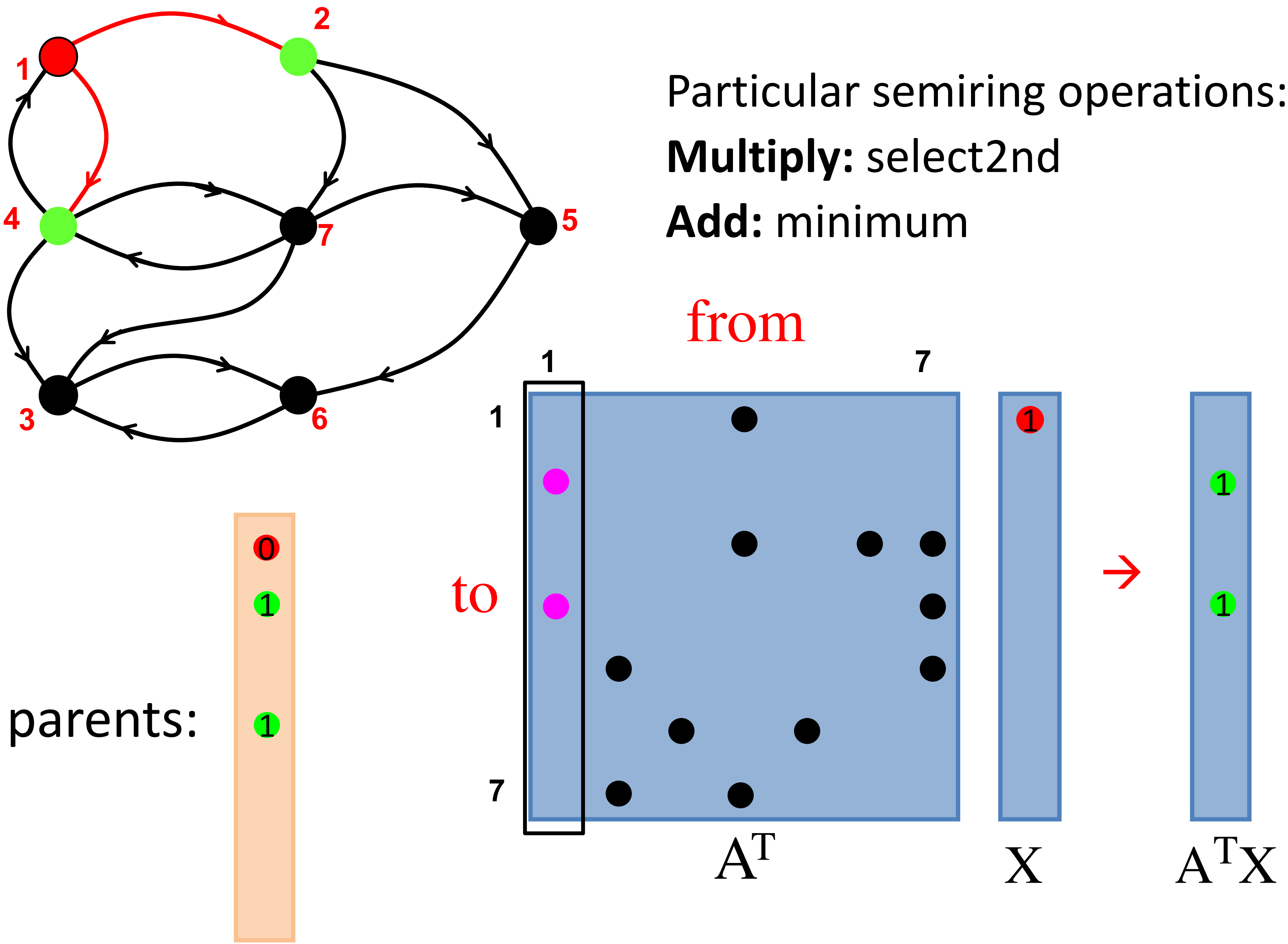
A.Buluç, T. Mattson, S. McMillan, J. Moreira, C. Yang. “The GraphBLAS C API Specification”, version 1.3.0

Breadth-first search in the language of matrices





A^T

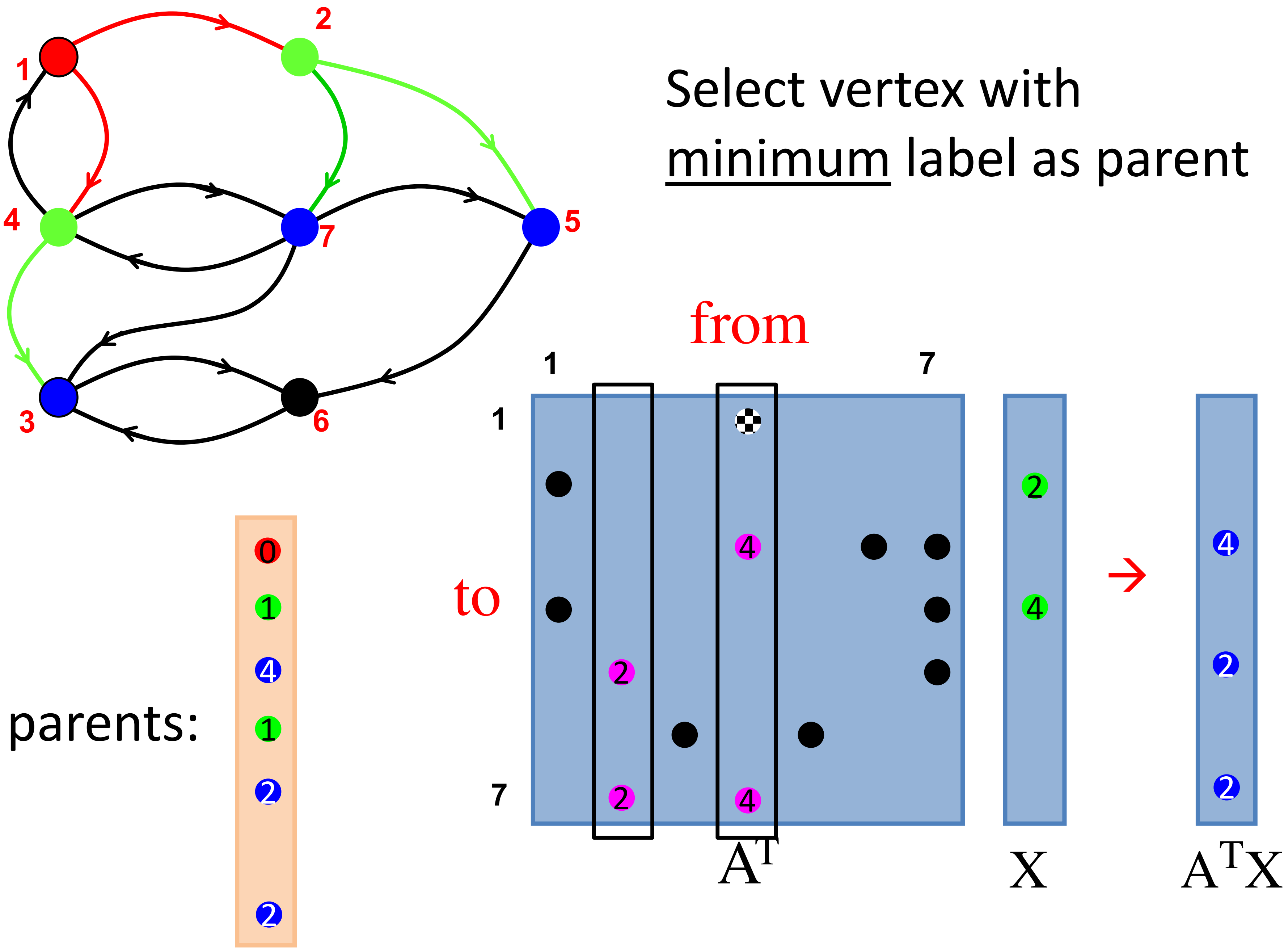


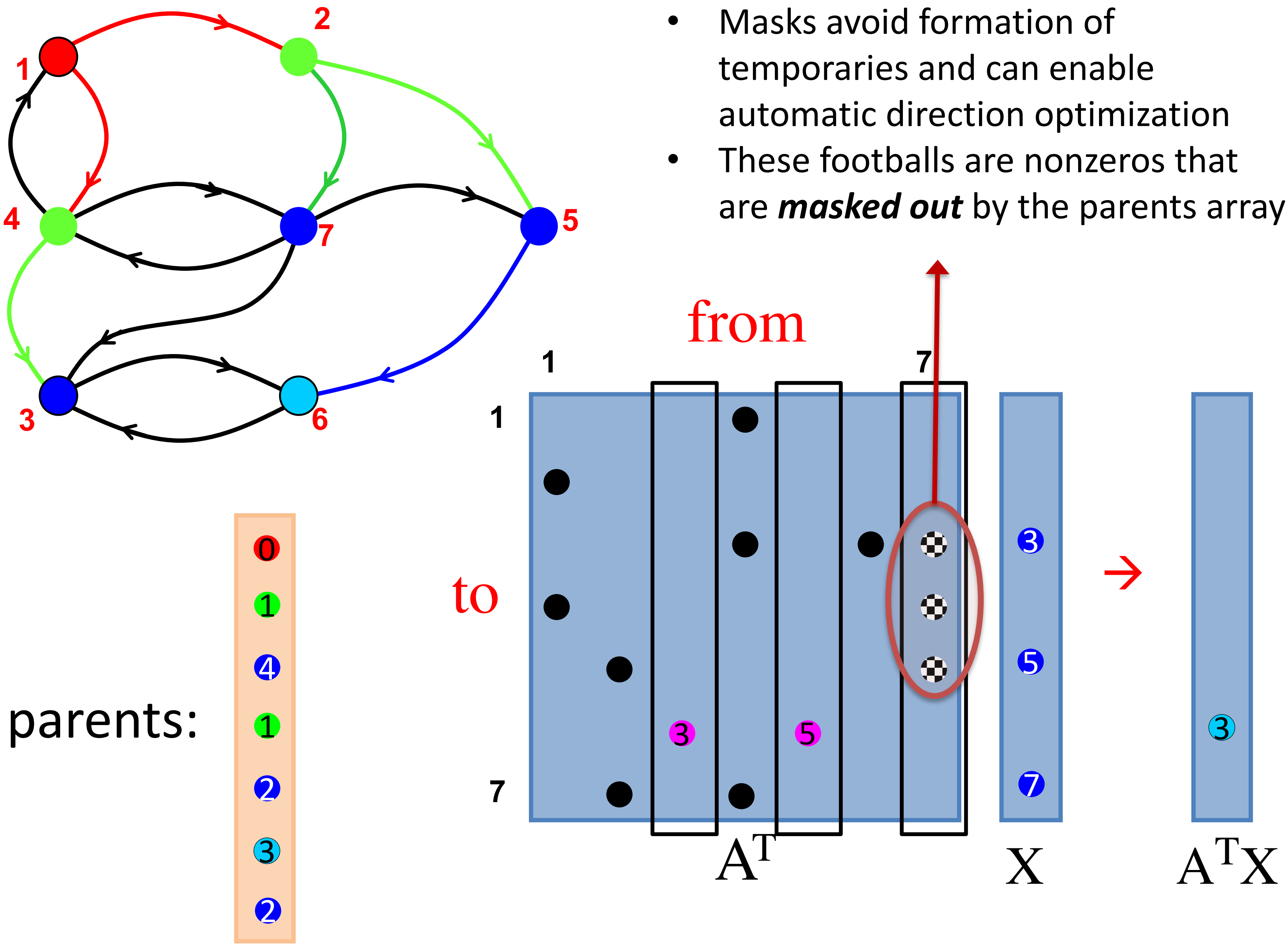
Particular semiring operations:

Multiply: select2nd

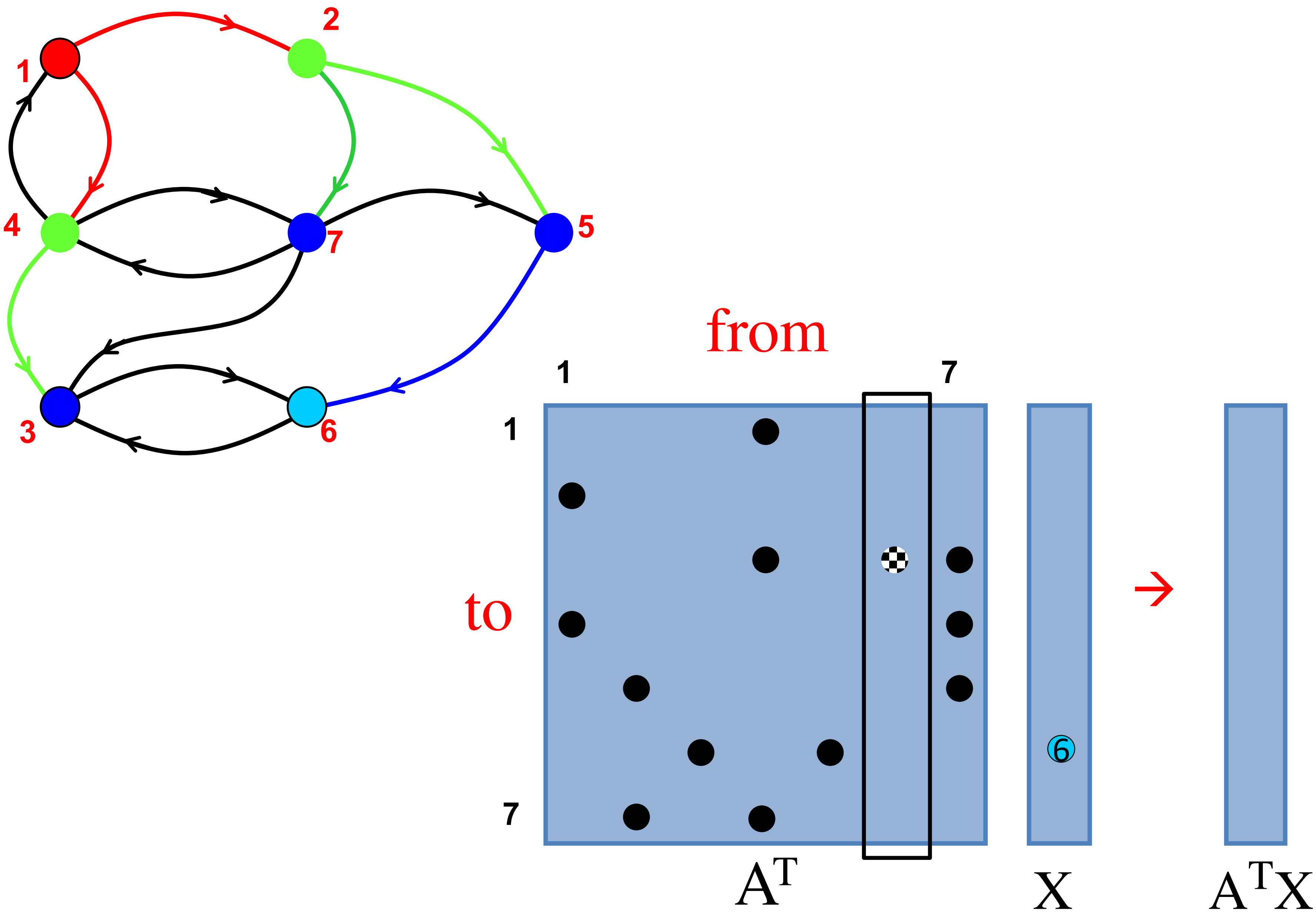
Add: minimum

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





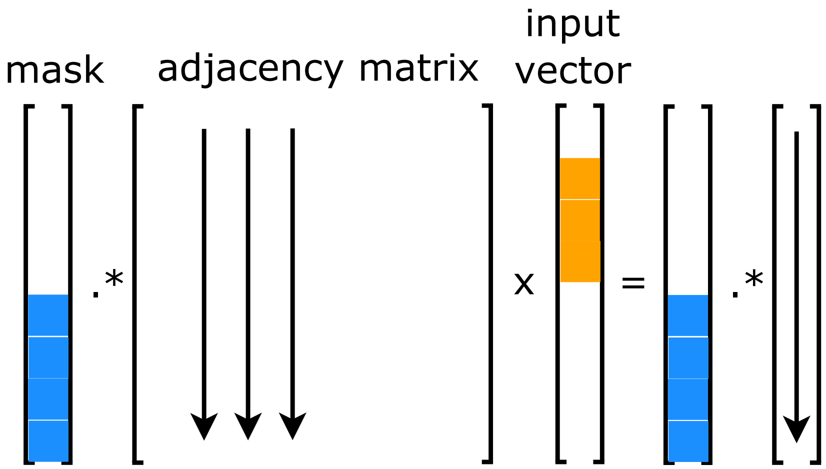
- Masks avoid formation of temporaries and can enable automatic direction optimization
- These footballs are nonzeros that are ***masked out*** by the parents array


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 $x = A^T x$.* 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

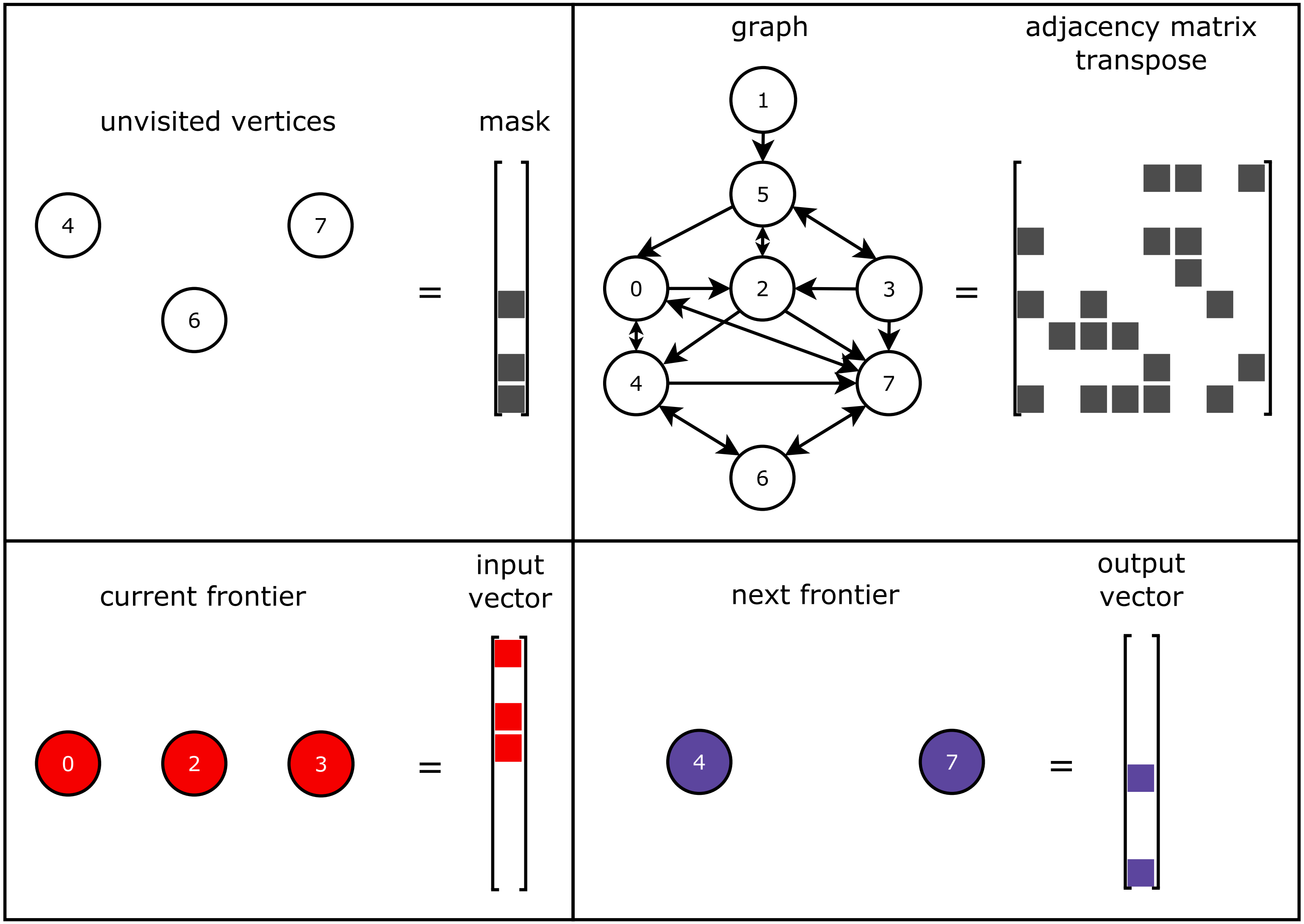
Idea was to run the same column-based algorithm, but checking against a mask before writing to output

Push-pull ≡ column-row matvec

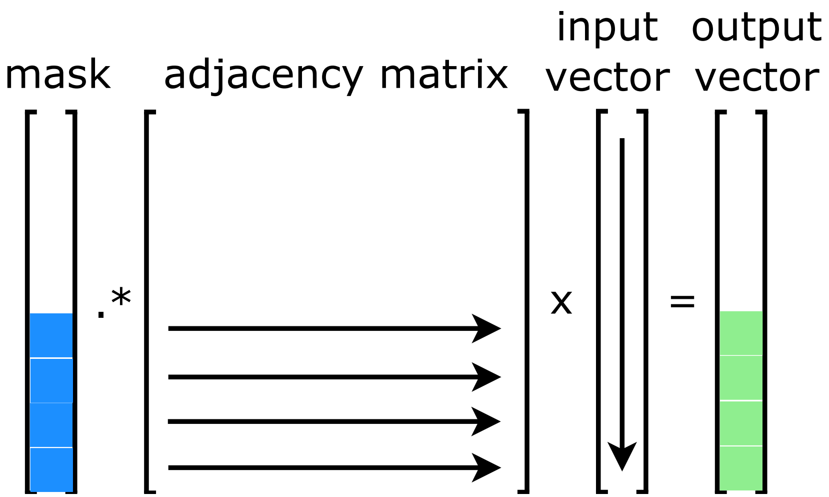
This is a story on how languages (and in this case APIs) change our thinking and drive our creative process

- 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

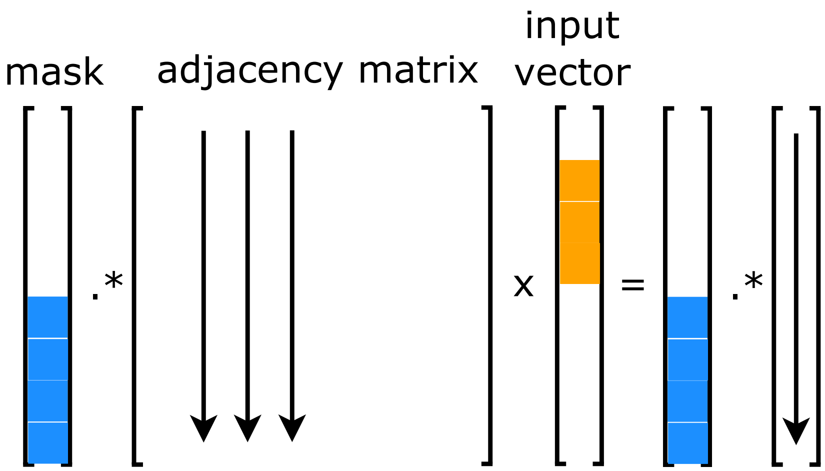
Enter “masks”


unvisited vertices
mask
4
7
6
=
graph
adjacency matrix transpose
1
5
0
2
3
4
7
6
=
current frontier
input vector
0
2
3
=
next frontier
output vector
4
7
=

Masks make "pull" implementable competitively in GraphBLAS



Row-based matvec w/ mask



Column-based matvec w/ mask

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

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

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:

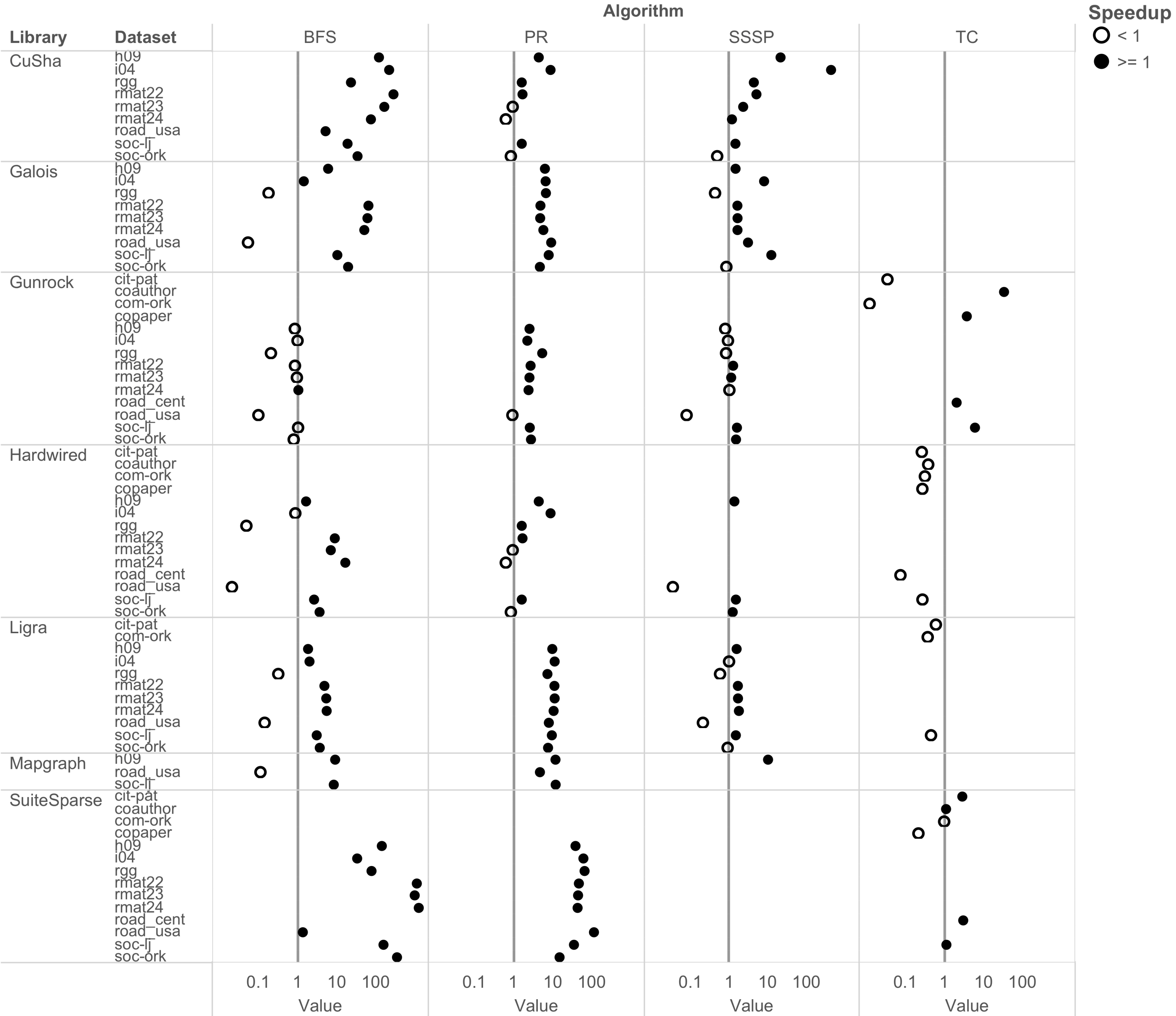
1. Exploit input sparsity => direction-optimization
2. Exploit output sparsity => masking
3. Proper load-balancing => key for GPU implementations

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

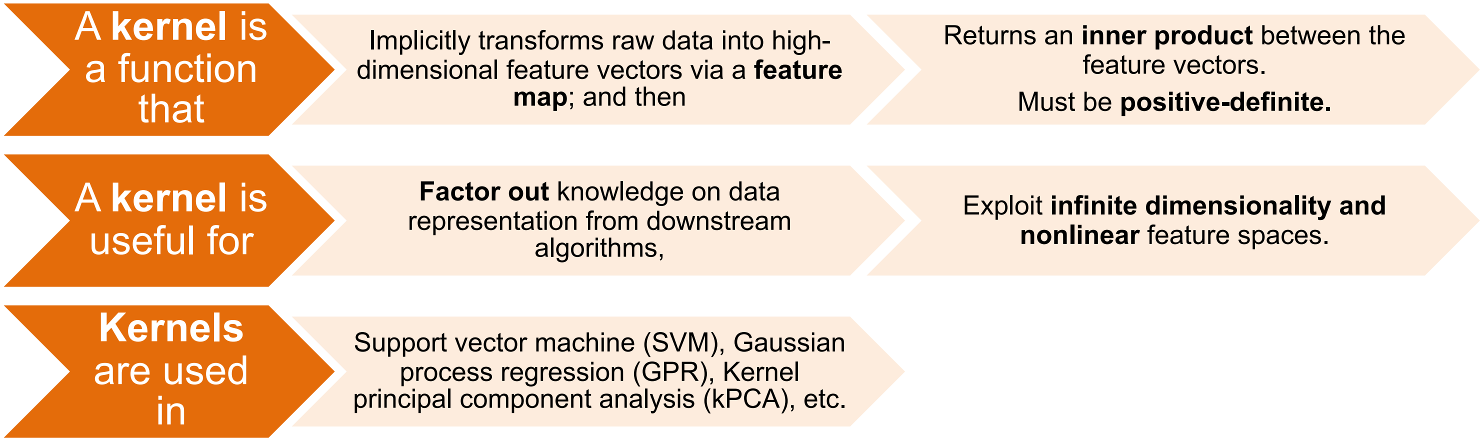
- Breadth-first-search (BFS)
- Single-source shortest-path (SSSP)
- PageRank (PR)
- Triangle counting (TC)

https://github.com/gunrock/graphblast

Yang, B., Owens, “GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU”, arXiv


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

Kernel methods in Machine Learning



A **kernel** is a function that

Implicitly transforms raw data into high-dimensional feature vectors via a **feature map**; and then

Returns an **inner product** between the feature vectors.

Must be **positive-definite.**

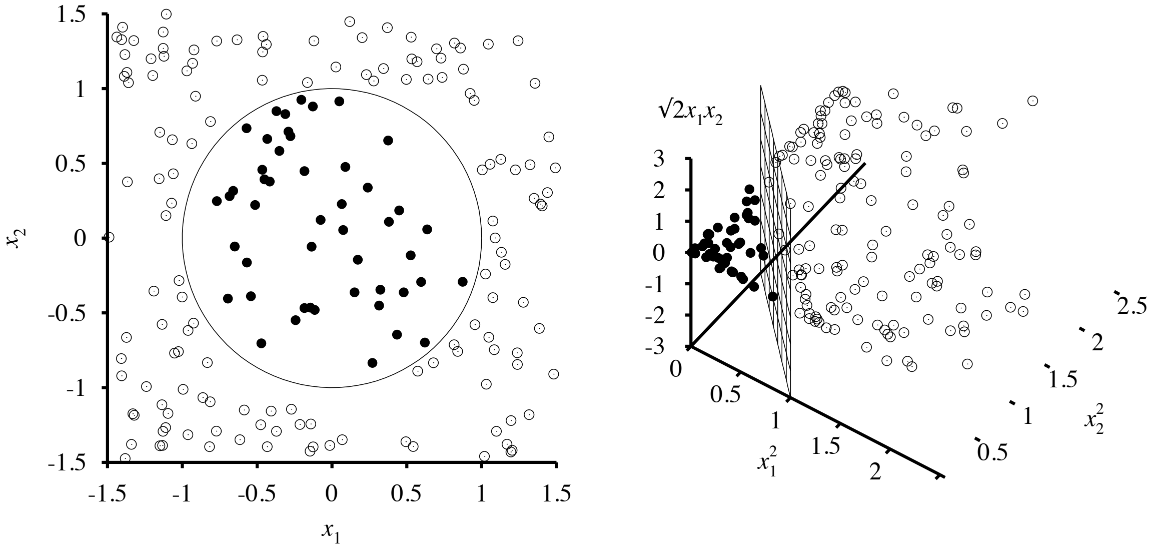
A **kernel** is useful for

Factor out knowledge on data representation from downstream algorithms,

Exploit **infinite dimensionality and nonlinear** feature spaces.

Kernels are used in

Support vector machine (SVM), Gaussian process regression (GPR), Kernel principal component analysis (kPCA), etc.



(a)

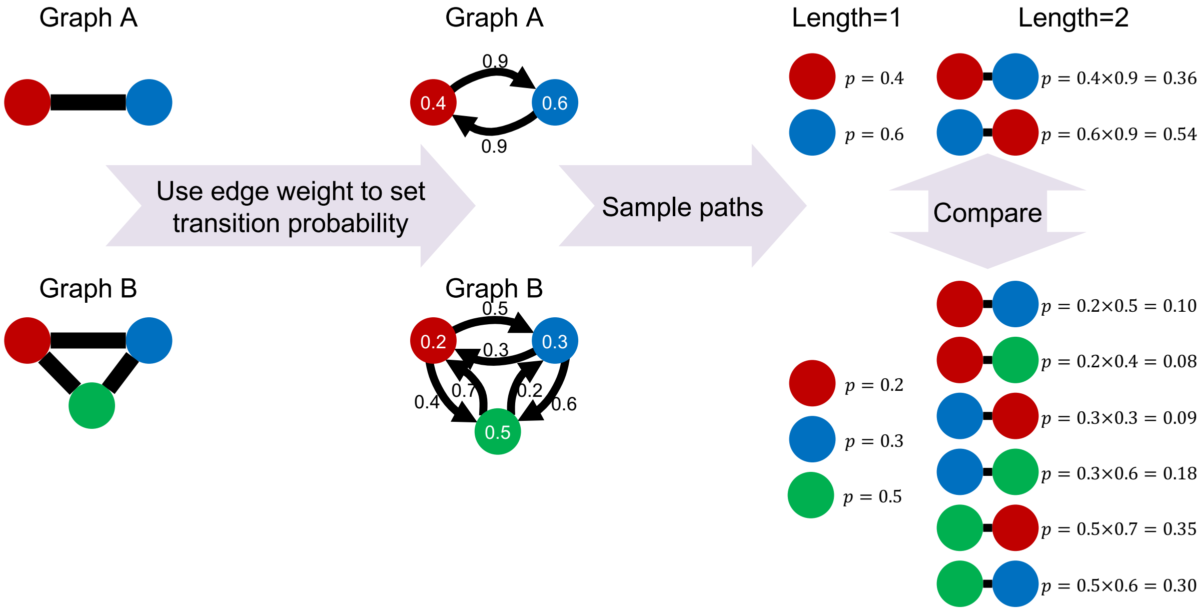
(b)

Figure source: Russell & Norvig

The circular decision boundary in 2D (a) becomes a linear boundary in 3D (b) using the following transformation: $\phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$

Marginalized Graph Kernels

The inner product between two graphs is the statistical average of the inner product of simultaneous random walk paths on the two graphs.



Graph A

Graph B

Use edge weight to set transition probability

Sample paths

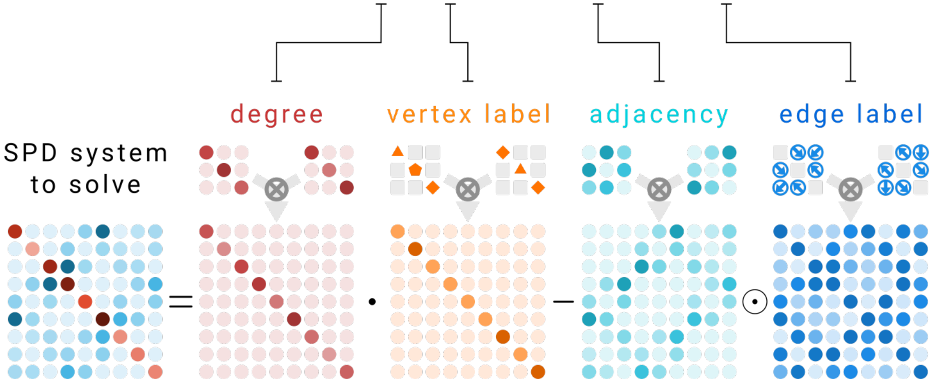
Length=1

Length=2

Compare

Graph A
0.9
0.4
0.6
0.9
Graph B
0.5
0.2
0.3
0.3
0.4
0.7
0.2
0.6
0.5
$p = 0.4$
$p = 0.6$
$p = 0.4\times0.9 = 0.36$
$p = 0.6\times0.9 = 0.54$
$p = 0.2\times0.5 = 0.10$
$p = 0.2\times0.4 = 0.08$
$p = 0.3\times0.3 = 0.09$
$p = 0.3\times0.6 = 0.18$
$p = 0.5\times0.7 = 0.35$
$p = 0.5\times0.6 = 0.30$
$p = 0.2$
$p = 0.3$
$p = 0.5$

$$K(G,G') = \mathbf{p}_\times^\mathsf{T}\left(\mathbf{D}_\times \mathbf{V}_\times^{-1} - \mathbf{A}_\times \odot \mathbf{E}_\times\right)^{-1} \mathbf{D}_\times \mathbf{q}_\times$$

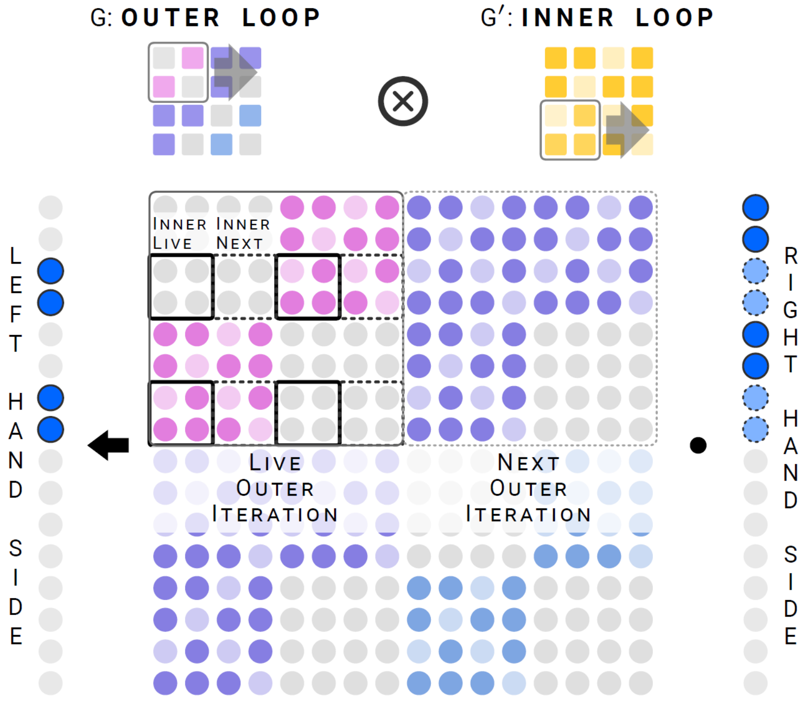


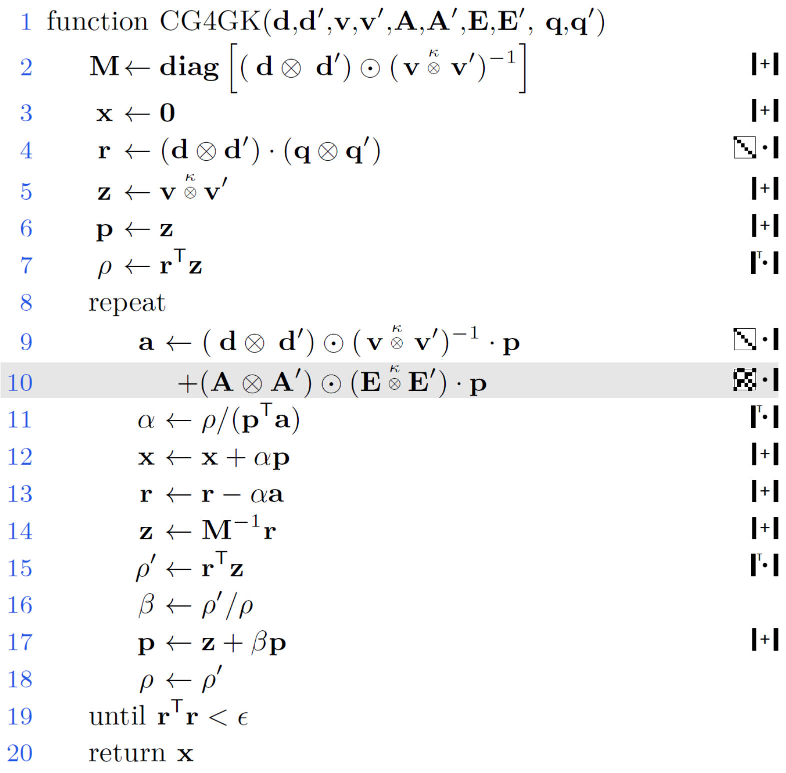
The marginalized graph kernel in linear algebra form represents a modified graph Laplacian

Solving the Graph Kernel PSD system

Streaming Kronecker matrix-vector multiplication

- Regenerates the product linear system on the fly by streaming 8-by-8 tiles.
- Tiles staged in shared memory.
- Trade FLOPS for GB/s, but asymptotic arithmetic complexity stays the same.



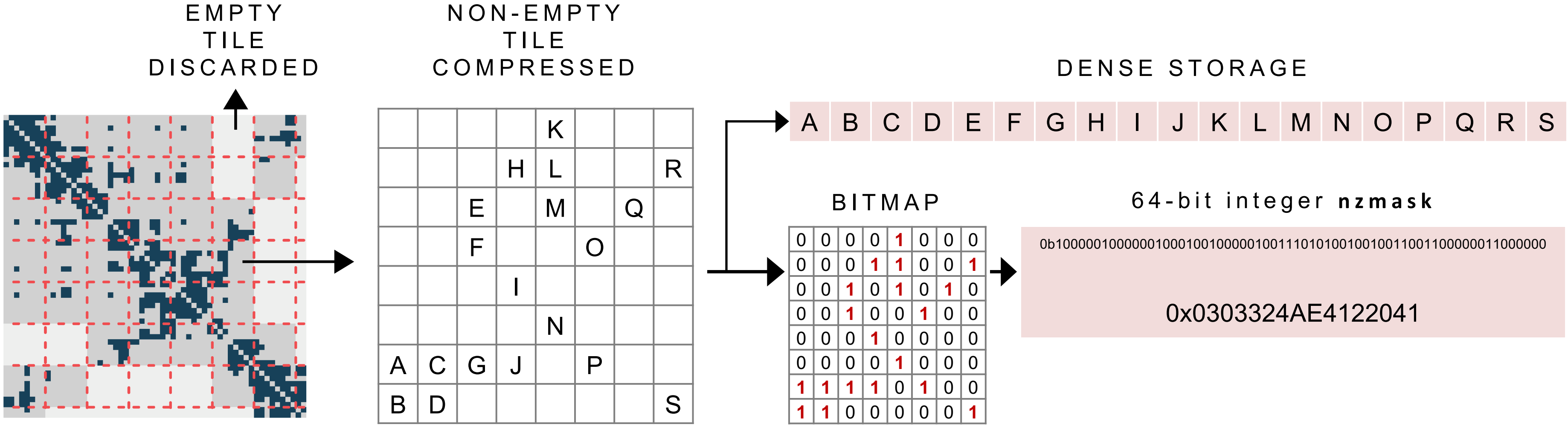


```
function CG4GK($\mathbf{d},\mathbf{d}',\mathbf{v},\mathbf{v}',\mathbf{A},\mathbf{A}',\mathbf{E},\mathbf{E}', \mathbf{q},\mathbf{q}'$)
    $\mathbf{M} \leftarrow \mathbf{diag}\left[(\mathbf{d} \otimes \mathbf{d}') \odot (\mathbf{v} \overset{\kappa}{\otimes} \mathbf{v}')^{-1}\right]$
    $\mathbf{x} \leftarrow \mathbf{0}$
    $\mathbf{r} \leftarrow (\mathbf{d} \otimes \mathbf{d}') \cdot (\mathbf{q} \otimes \mathbf{q}')$
    $\mathbf{z} \leftarrow \mathbf{v} \overset{\kappa}{\otimes} \mathbf{v}'$
    $\mathbf{p} \leftarrow \mathbf{z}$
    $\rho \leftarrow \mathbf{r}^\mathsf{T}\mathbf{z}$
    repeat
        $\mathbf{a} \leftarrow (\mathbf{d} \otimes \mathbf{d}') \odot (\mathbf{v} \overset{\kappa}{\otimes} \mathbf{v}')^{-1} \cdot \mathbf{p}$
            $+(\mathbf{A} \otimes \mathbf{A}') \odot (\mathbf{E} \overset{\kappa}{\otimes} \mathbf{E}') \cdot \mathbf{p}$
        $\alpha \leftarrow \rho/(\mathbf{p}^\mathsf{T}\mathbf{a})$
        $\mathbf{x} \leftarrow \mathbf{x} + \alpha\mathbf{p}$
        $\mathbf{r} \leftarrow \mathbf{r} - \alpha\mathbf{a}$
        $\mathbf{z} \leftarrow \mathbf{M}^{-1}\mathbf{r}$
        $\rho' \leftarrow \mathbf{r}^\mathsf{T}\mathbf{z}$
        $\beta \leftarrow \rho'/\rho$
        $\mathbf{p} \leftarrow \mathbf{z} + \beta\mathbf{p}$
        $\rho \leftarrow \rho'$
    until $\mathbf{r}^\mathsf{T}\mathbf{r} < \epsilon$
    return $\mathbf{x}$
```

Exploiting Sparsity

- Most discrete systems have natural sparsity (e.g. not all atoms are connected).
- 2-level sparsity exploitation:
  i. Outer level: retain only non-empty tiles
  ii. Inner level: use bitmap + compact storage format
- Packing into compact format: on CPU as a preprocessing step
- Unpacking for Streaming Kronxv: on GPU using bit magic + warp intrinsics
- Partition-based graph ordering reduces # non-empty tiles

  ☛ Cost easily amortized because we reorder each graph, not their product



				K			
			H	L			R
		E		M		Q	
		F			O		
			I				
				N			
A	C	G	J		P		
B	D						S

DENSE STORAGE

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S

BITMAP

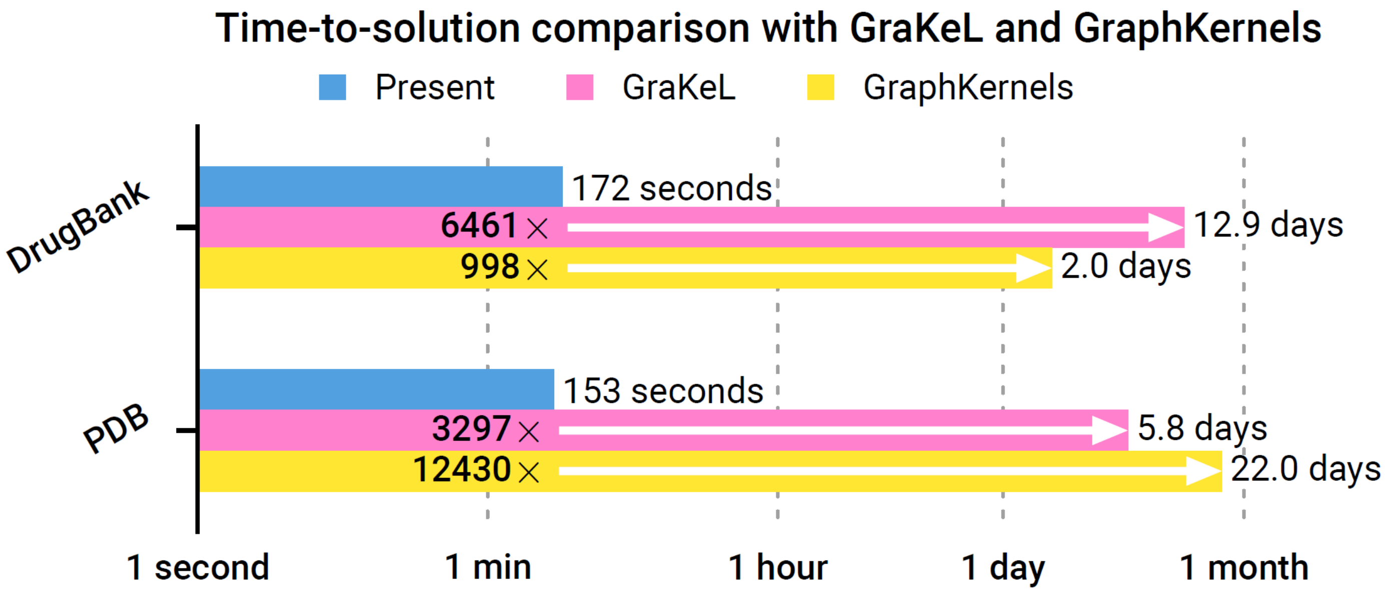
0	0	0	0	1	0	0	0
0	0	0	1	1	0	0	1
0	0	1	0	1	0	1	0
0	0	1	0	0	1	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
1	1	1	1	0	1	0	0
1	1	0	0	0	0	0	1

64-bit integer **nzmask**

0b1000000100000001000100100000010011101010010010011001100000011000000

0x0303324AE4122041

Performance of the Graph Kernel

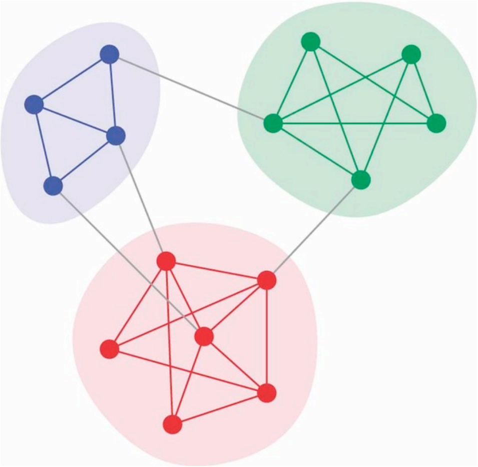


GraKeL: Cython, multi-threading
GraphKernels: Python, no parallelization

Yu-Hang Tang, Oguz Selvitopi, Doru Popovici, and Aydin Buluç. A high-throughput solver for marginalized graph kernels on GPU. In Proceedings of the IPDPS, 2020.

The Markov Cluster Algorithm (MCL)

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



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

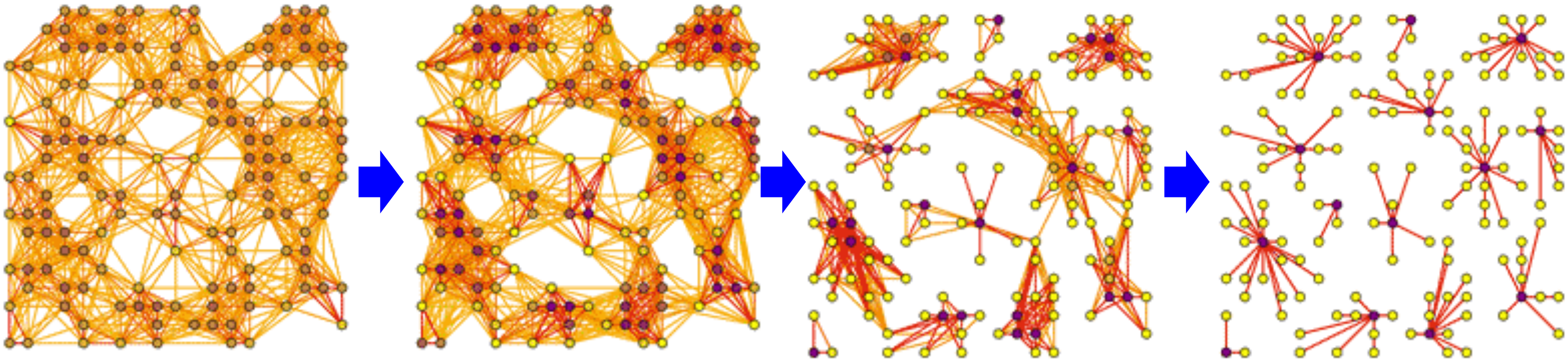
↓

Random walks on the graph will frequently remains within a cluster

↓

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

The Markov Cluster Algorithm (MCL)



Initial network

Iteration 1

Iteration 2

Iteration 3

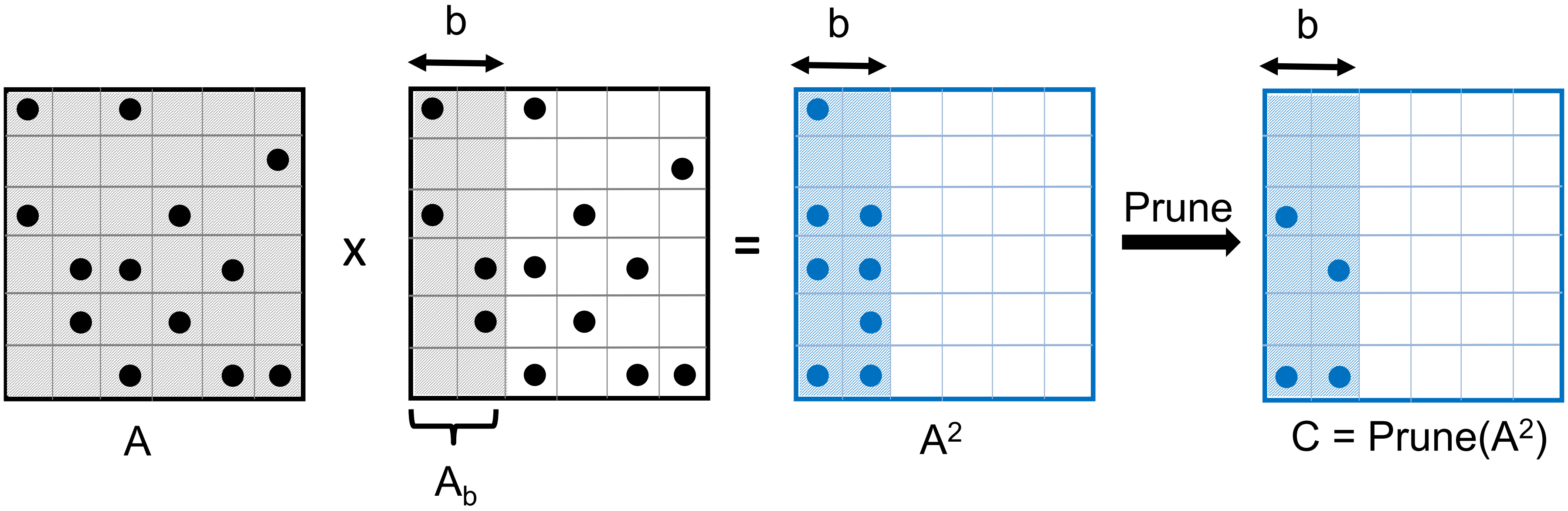
At each iteration:

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

A combined expansion and pruning step

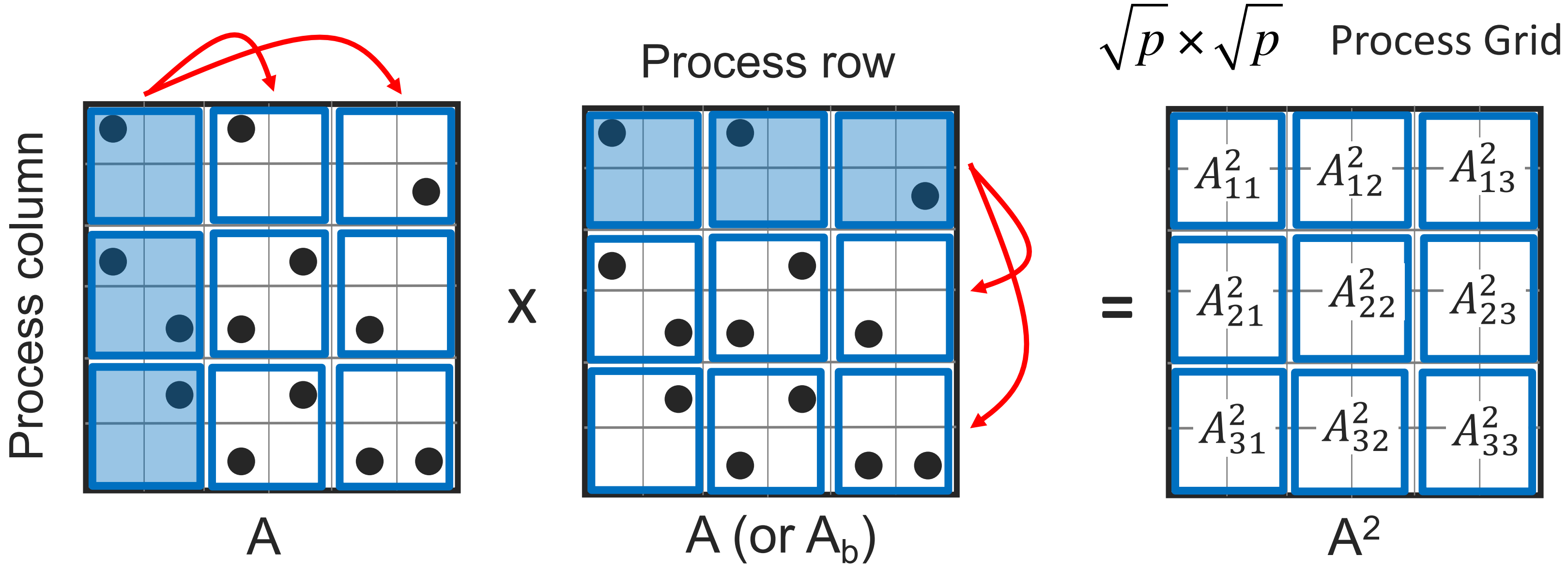


❑ b: number of columns in the output constructed at once

- 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

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*

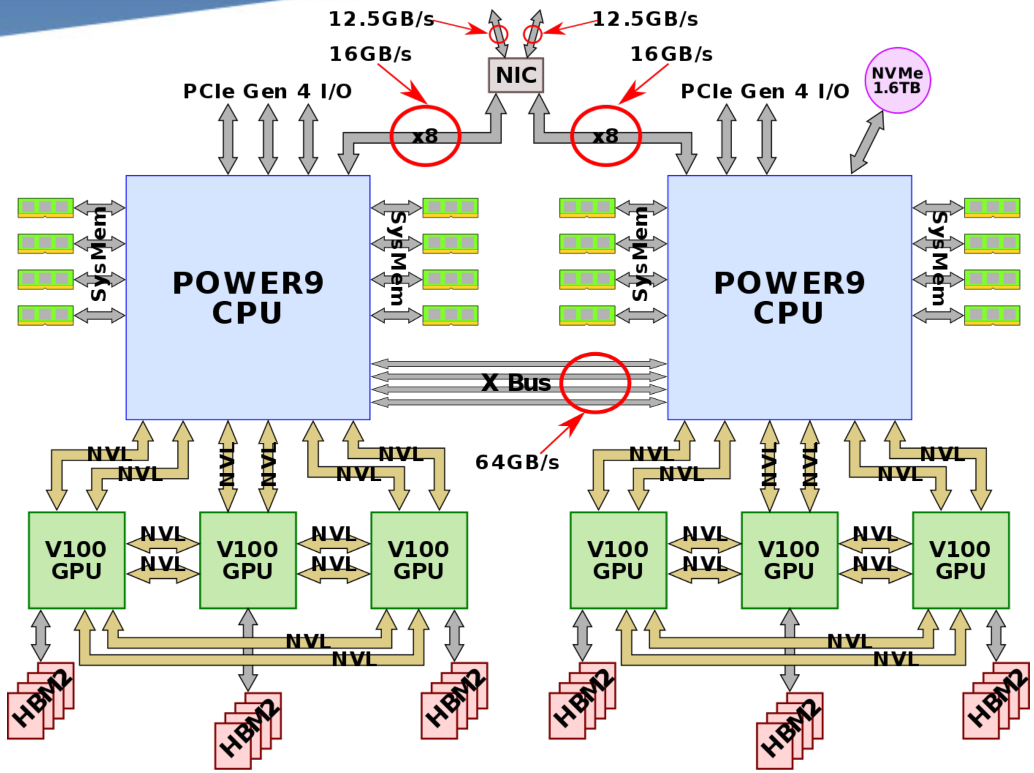
HipMCL on large networks

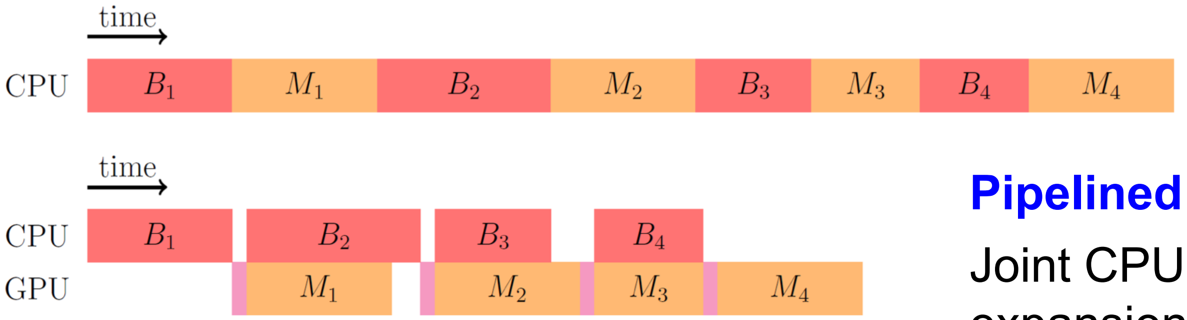
Data	Proteins	Edges	#Clusters	HipMCL time	platform
Isolate-1	47M	7 B	1.6M	1 hr	1024 nodes Edison
Isolate-2	69M	12 B	3.4M	1.66 hr	1024 nodes Edison
Isolate-3	70M	68 B	2.9M	2.41 hr	2048 nodes Cori KNL
MetaClust50	282M	37B	41.5M	3.23 hr	2048 nodes Cori KNL

MCL can not cluster these networks

HipMCL on Supercomputers with accelerators

- Recent top supercomputers are all accelerated (e.g. with GPUs)
- This is what a ORNL Summit node looks like
- There are 4608 such nodes in the system
- Challenges: (1) Utilizing all GPUs, (2) hiding the communication

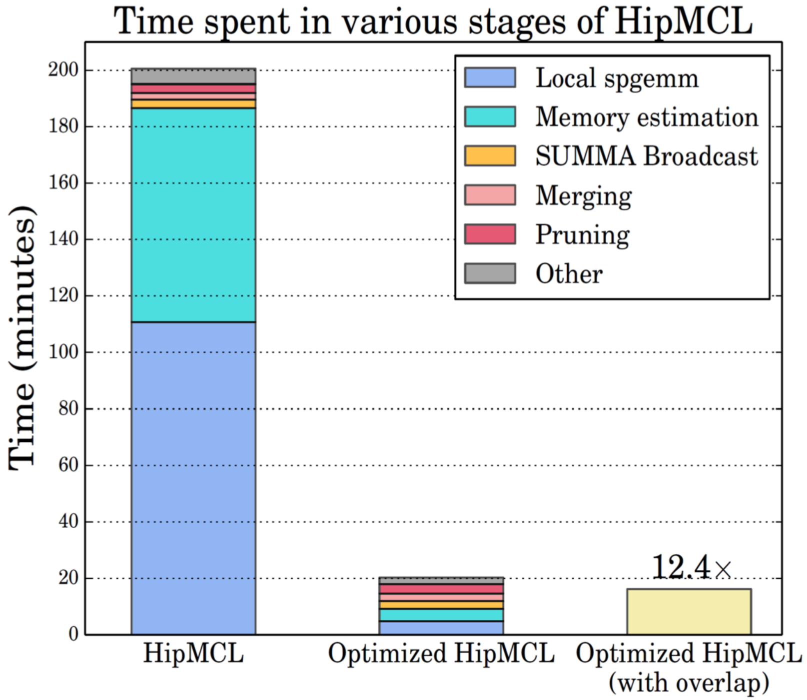


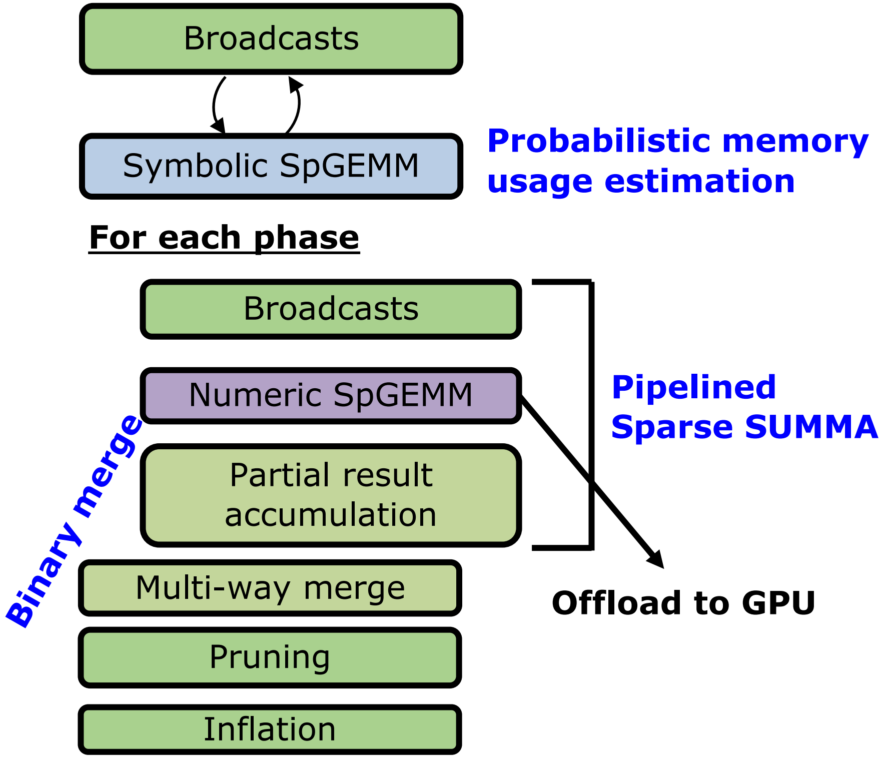


Pipelined Sparse SUMMA

Joint CPU-GPU distributed memory expansion of MCL algorithm

HipMCL on Supercomputers with accelerators





Broadcasts

Symbolic SpGEMM

Probabilistic memory usage estimation

For each phase

Broadcasts

Numeric SpGEMM

Partial result accumulation

Pipelined Sparse SUMMA

Binary merge

Multi-way merge

Offload to GPU

Pruning

Inflation

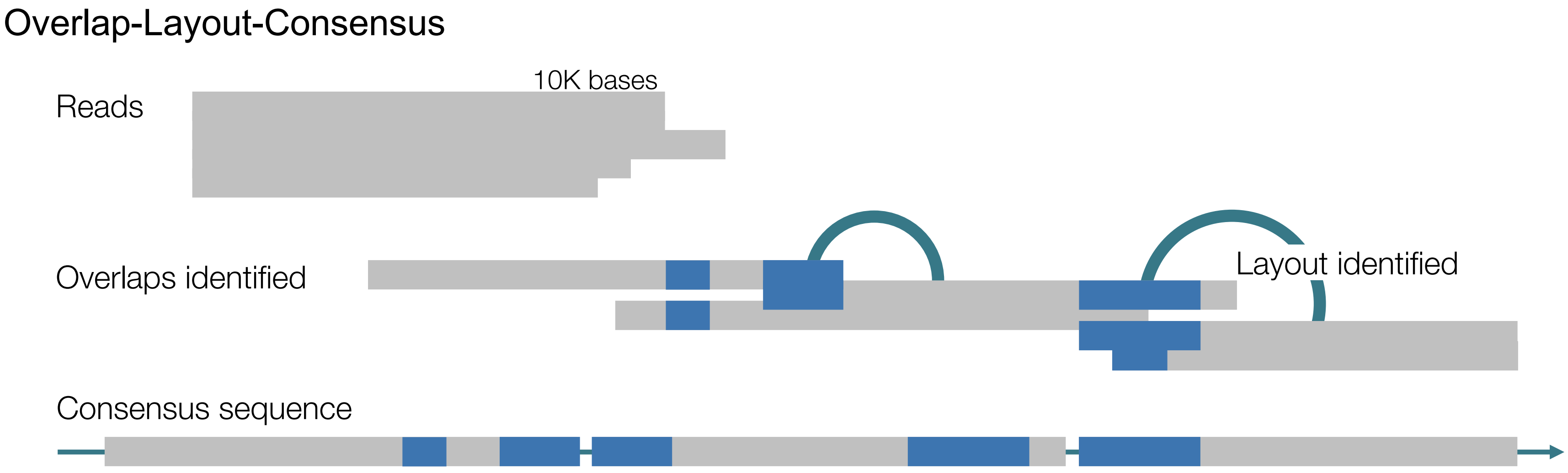
Other changes to HipMCL for the CPU-GPU workflow:

- *Randomized memory estimation algorithm* avoids symbolic phase
- New *eager binary merging* reduces memory footprint
- Integration of a much faster hash-based CPU SpGEMM algorithm

O. Selvitopi, M.T. Hussain, A. Azad, and A. Buluç. Optimizing high performance Markov clustering for pre-exascale architectures. IPDPS, 2020

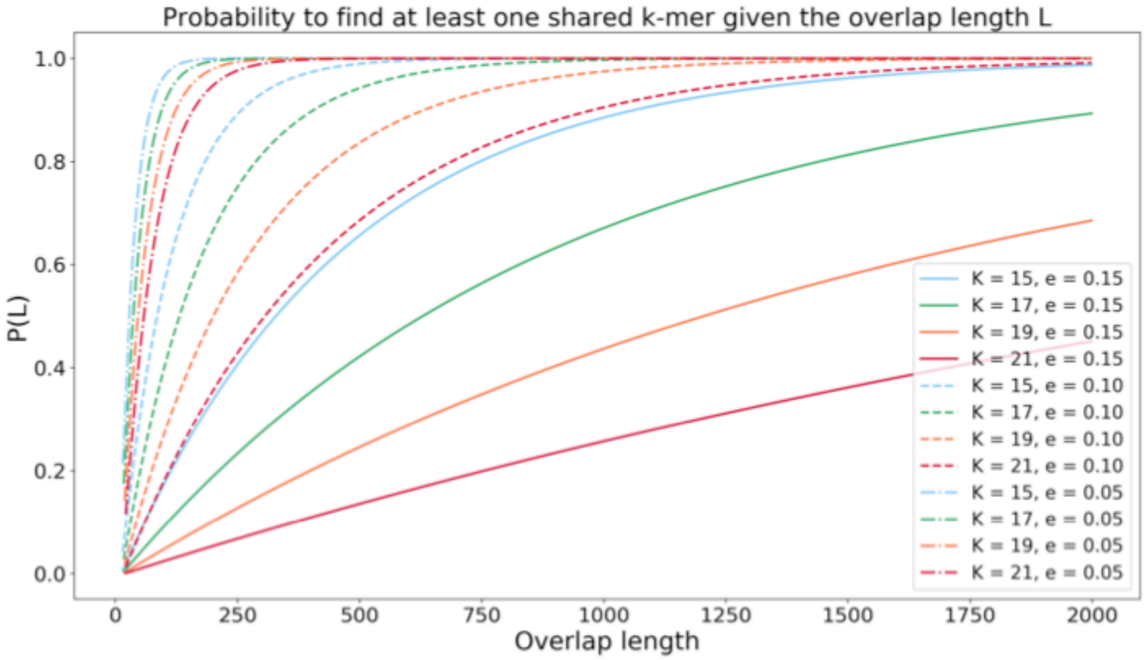
SpGEMM for DNA read overlapping

- **Long reads** from PacBio and Oxford Nanopore have the potential to revolutionize de-novo assembly
- **Overlap-Consensus-Layout** paradigm is more suitable than de Bruijn graph paradigm.
- **Overlapping** is the most computationally expensive step.



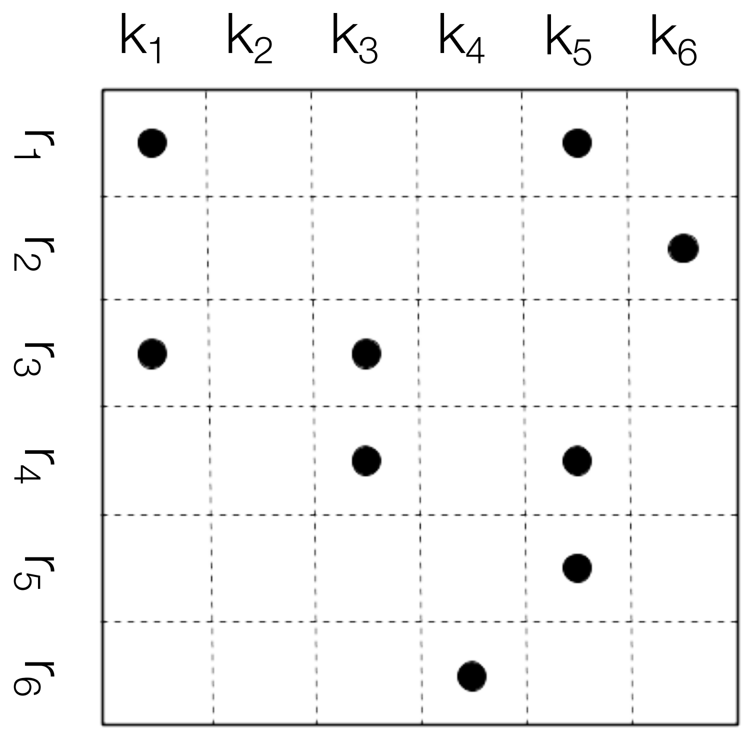
SpGEMM for DNA read overlapping

- We need to quickly determine pairs of reads that are *likely to* overlap, without resorting to O(n^2) comparisons
- If two reads do not share any subsequence of length k (aka a k-mer) for a reasonably short k, then they are unlikely to overlap



SpGEMM for DNA read overlapping

A matrix



	k_1	k_2	k_3	k_4	k_5	k_6
r_1	●				●	
r_2						●
r_3	●		●			
r_4			●		●	
r_5					●	
r_6				●		

- Suppose you have counted k-mers and only retained *reliable* k-mers
- Now you can generate this ***read-by-kmer*** sparse matrix **A**
- These are all linear time computations so far

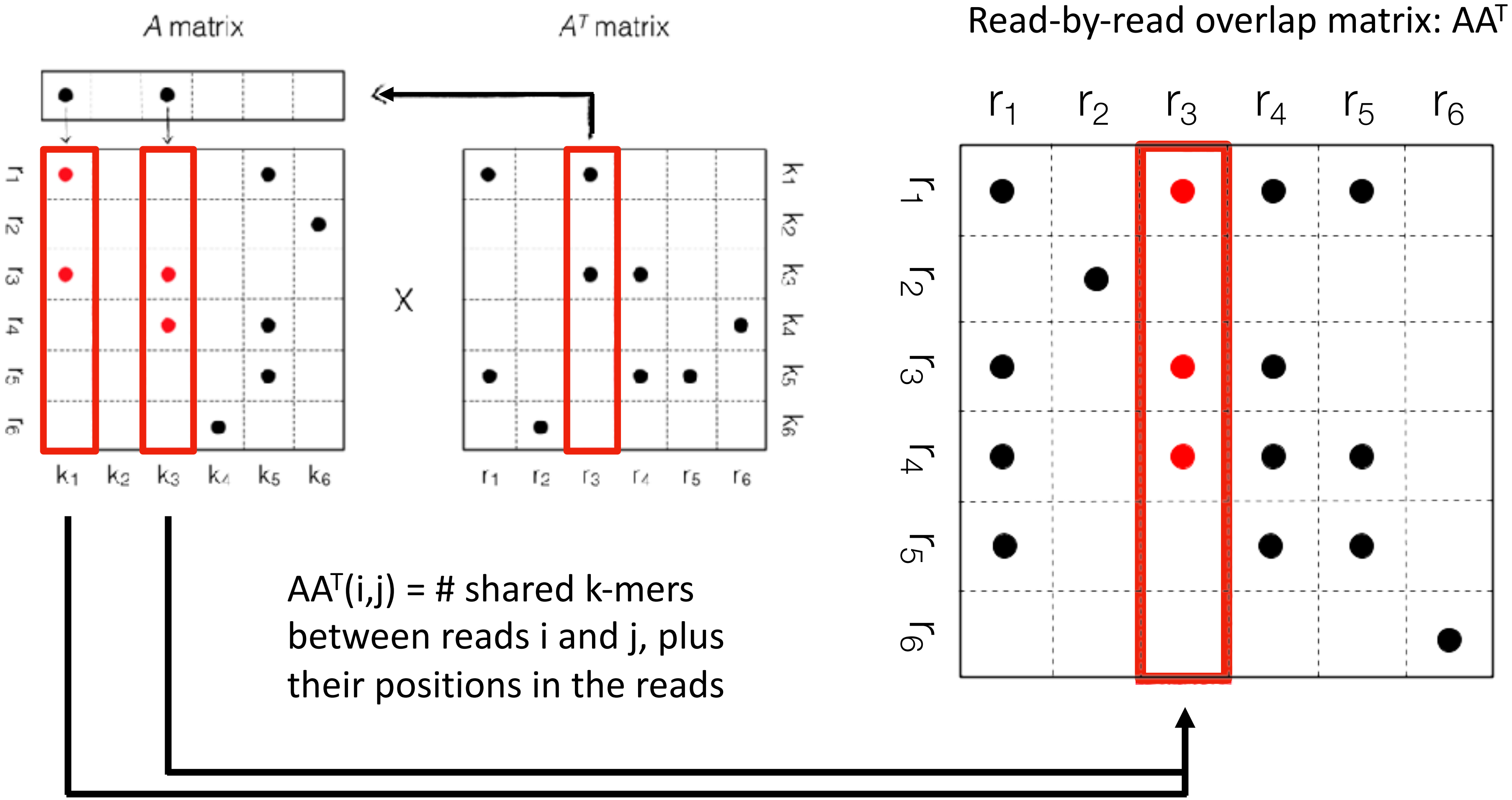
r_i = i^{th} read

k_j = j^{th} reliable *k*-mer

A(i,j) = presence of j^{th} reliable *k*-mer in i^{th} read, plus its position

Giulia Guidi, Marquita Ellis, Daniel Rokhsar, Kathy Yelick, Aydın Buluç, BELLA: Berkeley Efficient Long-read to Long-Read Overlapper and Aligner, Biorxiv, 2018

SpGEMM for DNA read overlapping


A matrix
A^T matrix
X
r_1 r_2 r_3 r_4 r_5 r_6
k_1 k_2 k_3 k_4 k_5 k_6

Read-by-read overlap matrix: AA^T

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

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

Acknowledgments

Ariful Azad, Tim Davis, Marquita Ellis, John Gilbert, Giulia Guidi, Jeremy Kepner, Nikos Krypides, Tim Mattson, Scott McMillan, Jose Moreira, John Owens, Georgios Pavlopoulos, Dan Rokhsar, Oguz Selvitopi, Yu-Hang Tang, Carl Yang, Kathy Yelick.

- My Research Team: http://passion.lbl.gov
- Our (new) Youtube Channel: http://shorturl.at/lpFRY
- The GraphBLAS Forum: http://graphblas.org