Linear Algebra is the Right Way to Think About Graphs

Supercomputing Conference 2018

Carl Yang[1,3], Aydın Buluç[2,3] and John D. Owens[1]

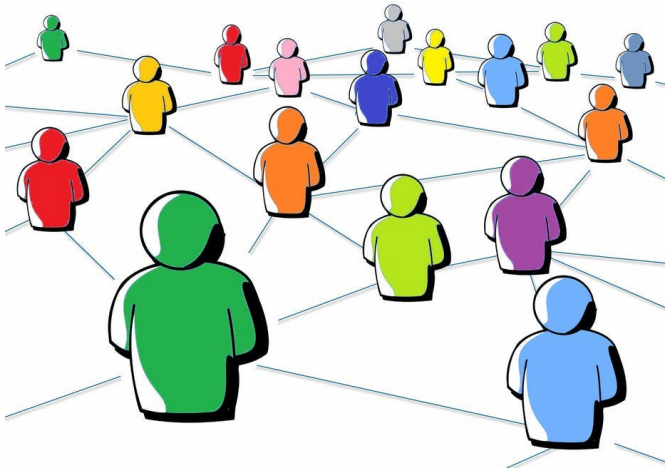
[1] University of California, Davis
[2] University of California, Berkeley
[3] Lawrence Berkeley National Laboratory

November 13, 2018

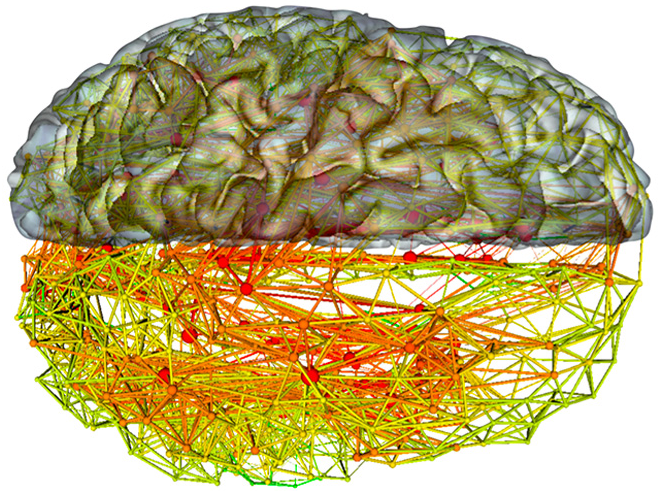
Overview

Overview

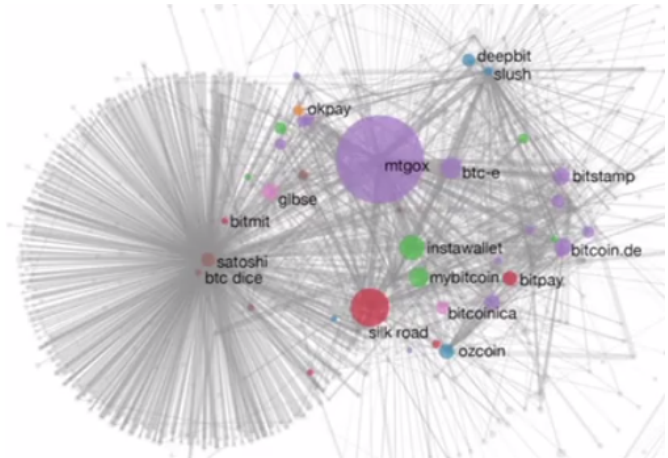
What's in common?



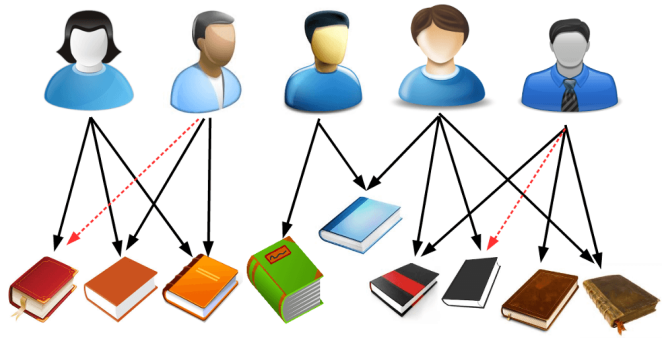
(a) Social network analysis



(b) Bioinformatics

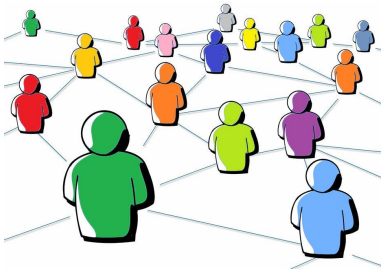


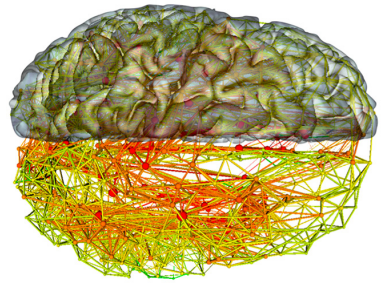
(c) Computer forensics

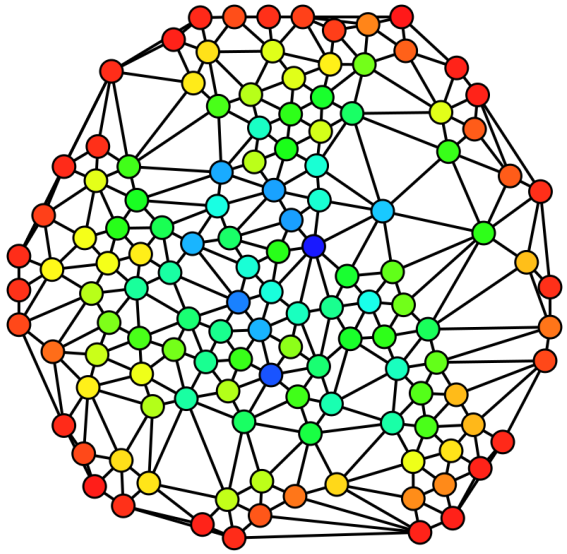


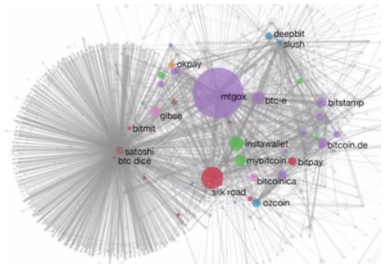
(d) Recommender systems

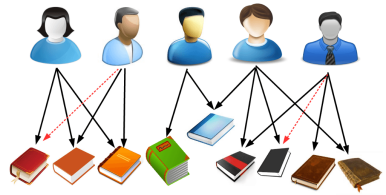
What's in common?



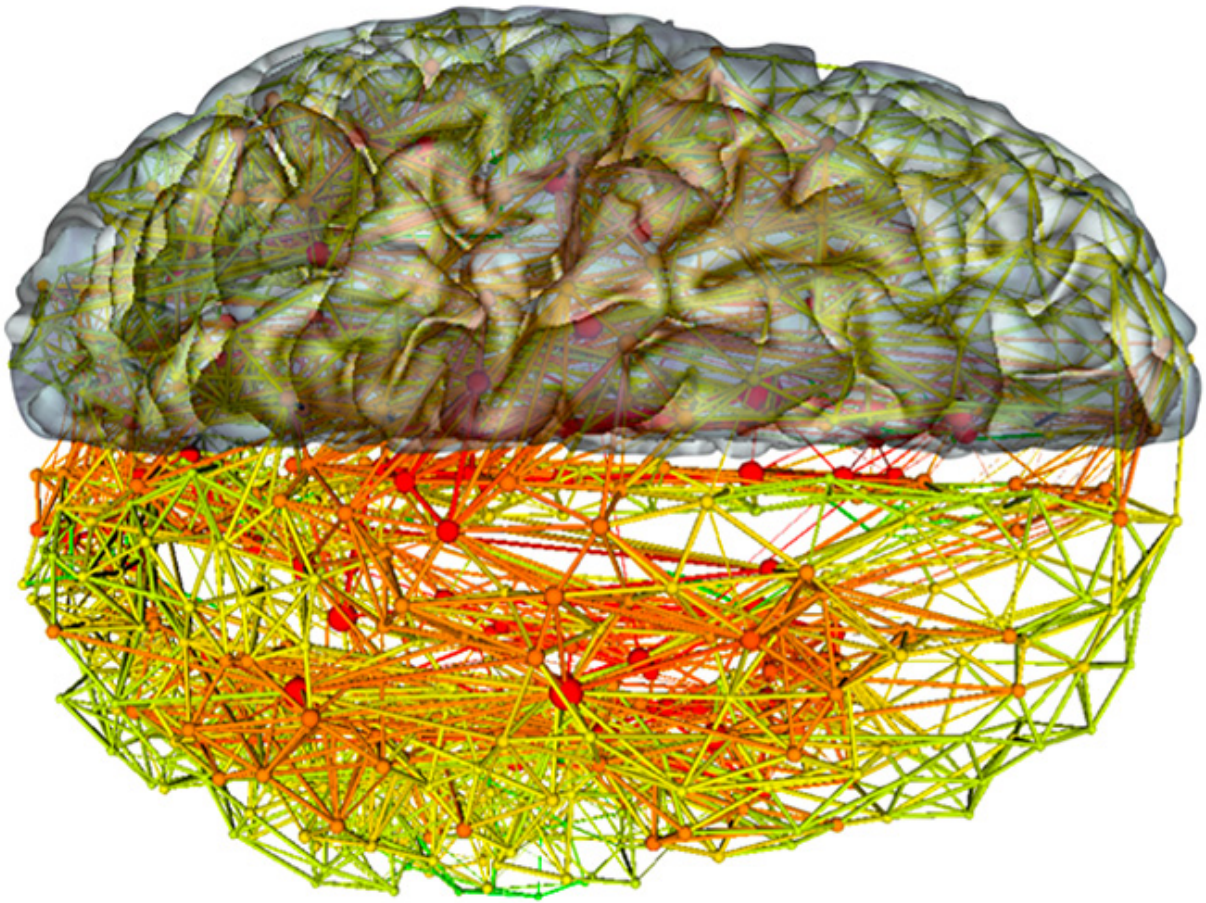








Problem: Why high-performance?



[0]Van Den Heuvel, Martijn P., and Olaf Sporns. "Rich-club organization of the human connectome." *Journal of Neuroscience* 31.44 (2011): 15775-15786.

Solution: Graph frameworks

CPU: Pregel, GraphLab, Cyclops, GraphX, Powergraph, GoFFish, Blogel, Gremlin, Haloop, Apache Giraph, Apache Hama, GPS, Mizan, Giraphx, Seraph, GiraphUC, Pregel+, Pregelix, Apache Tinkerpop, LFGraph, Gelly, Trinity, Ligra

GPU: Gunrock, Medusa, Totem, Frog, VertexAPI2, MapGraph, CuSha

Problem: What are the right primitives?

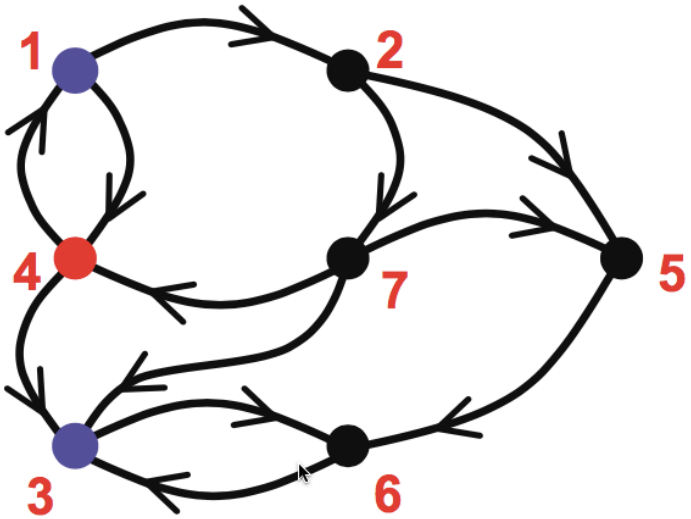
1. Concise
2. Portable
3. High-performance
4. Expressible

Thesis statement

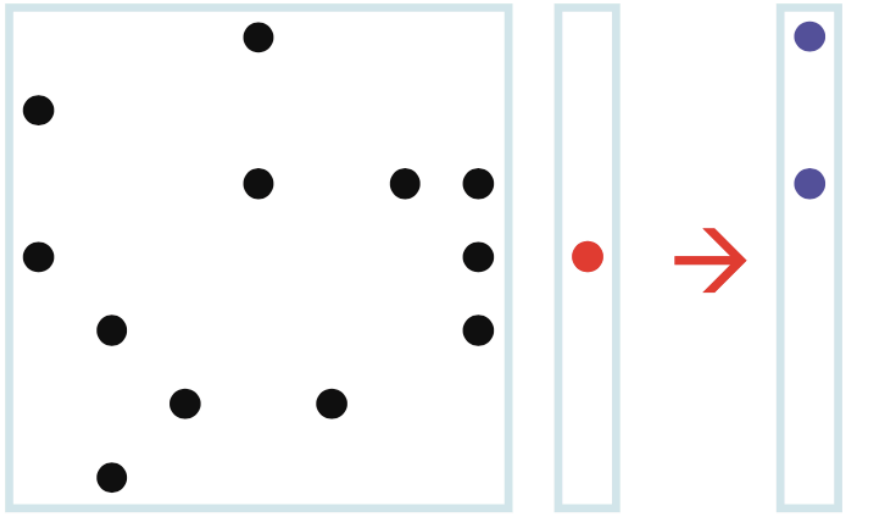
Linear algebra is the right way to think about graph algorithms.

Graph primitives based in linear algebra are superior ones based on existing vertex-centric graph frameworks in conciseness, portability, performance and expressibility.

Graph traversal is sparse matrix multiplication



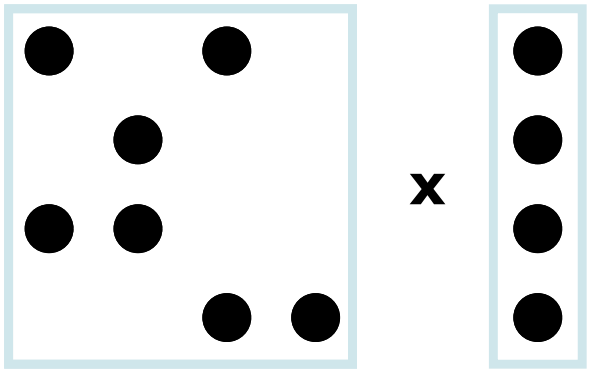
$$G = (V, E)$$



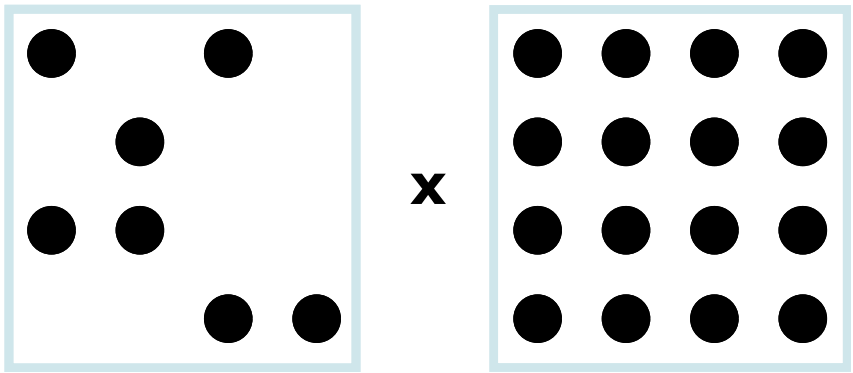
$$A^T \quad x \quad A^T x$$

[1] Denes Konig, 1931

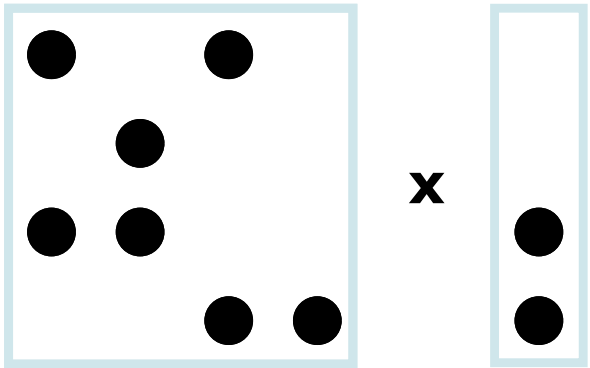
Ingredient 1: Operations



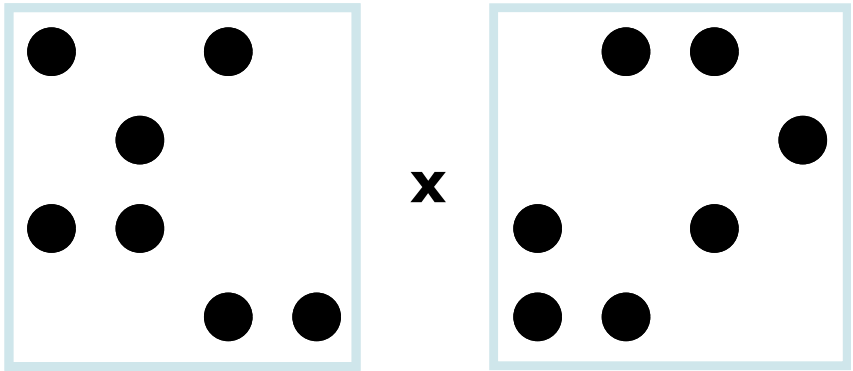
(a) SpMV
Yang et al., ICPP '18



(b) SpMM
Yang et al., EuroPar '18



(c) SpMSpV
Yang et al., IPDPSW '15, ICPP '18



(d) SpGEMM

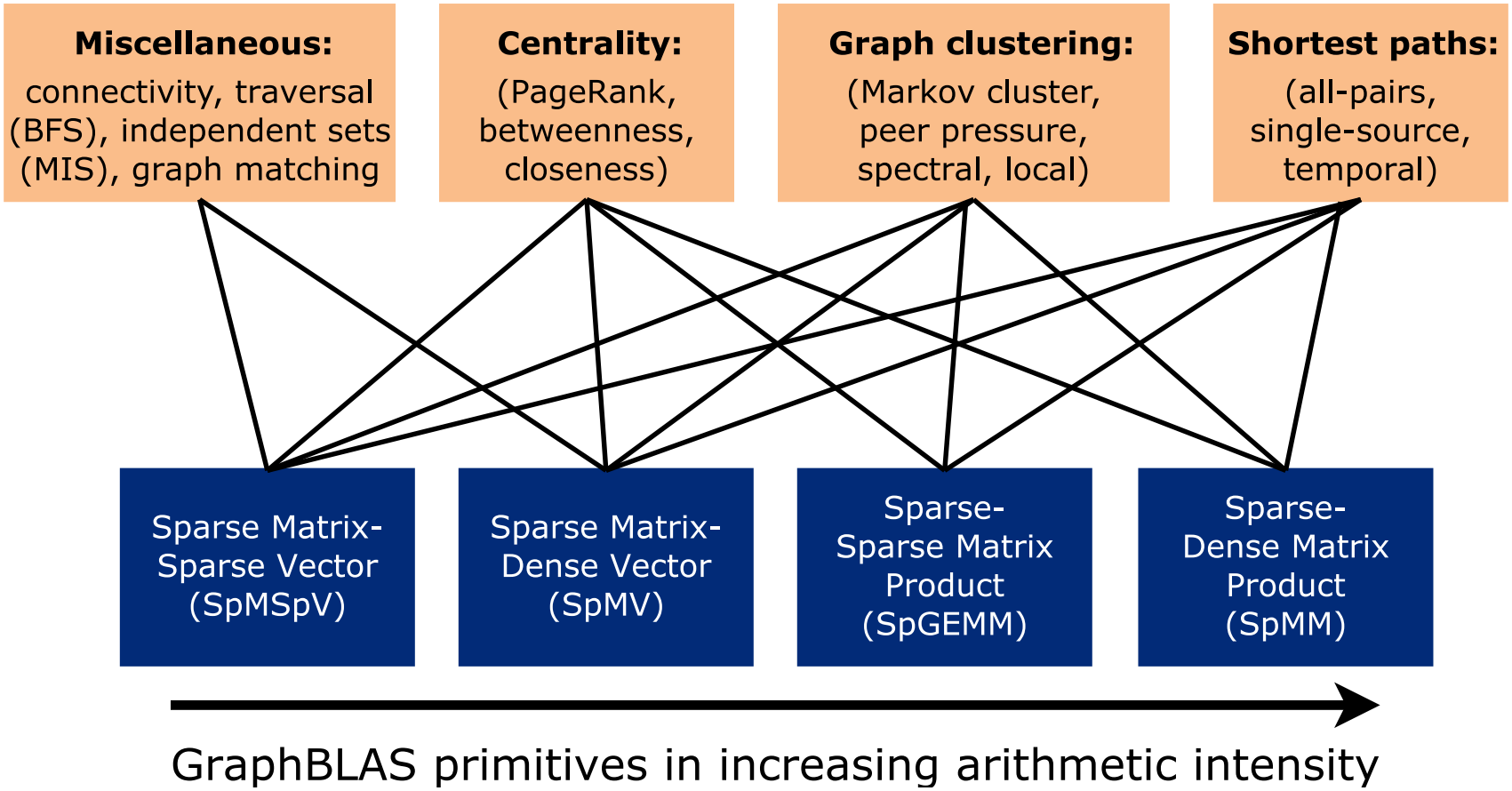
Ingredient 2: Operators

Semiring notation: (Add, Multiply, Domain)

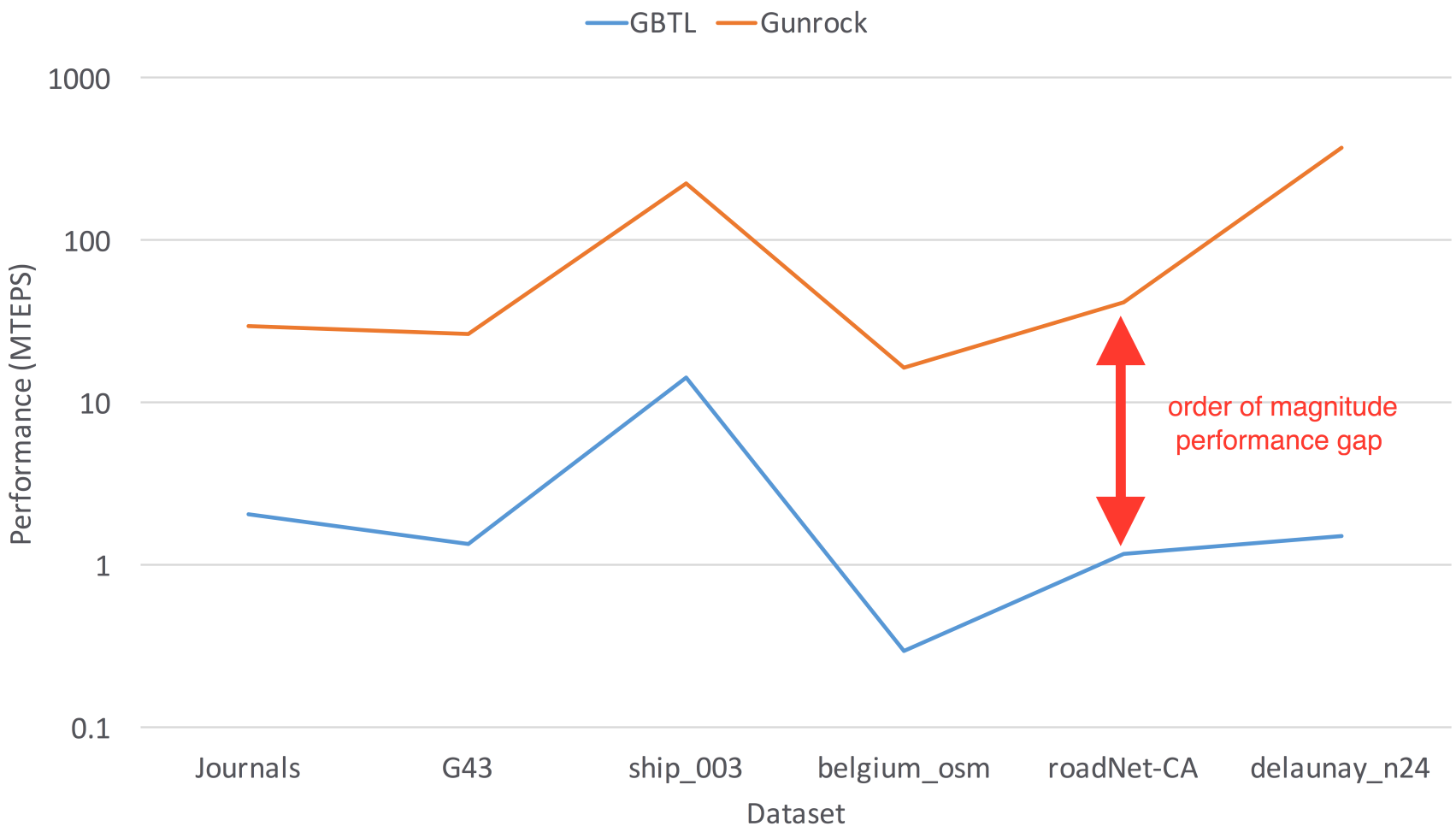
- Add: How to combine edges
- Multiply: How to combine vertex with incident edges
- Domain: Vertex/edge attributes

Name	Semiring	Application
Real field	$\{+, \times, \mathbb{R}\}$	Classical numerical linear algebra
Boolean	$\{\|, \&, \{0, 1\}\}$	Graph connectivity
Tropical	$\{\min, +, \mathbb{R} \cup \{\infty\}\}$	Shortest path
Max-plus	$\{\max, +, \mathbb{R}\}$	Graph matching
Min-times	$\{\min, \times, \mathbb{R}\}$	Maximal independent set

Operations + Operators = GraphBLAS



Problem: Existing implementations not performant enough

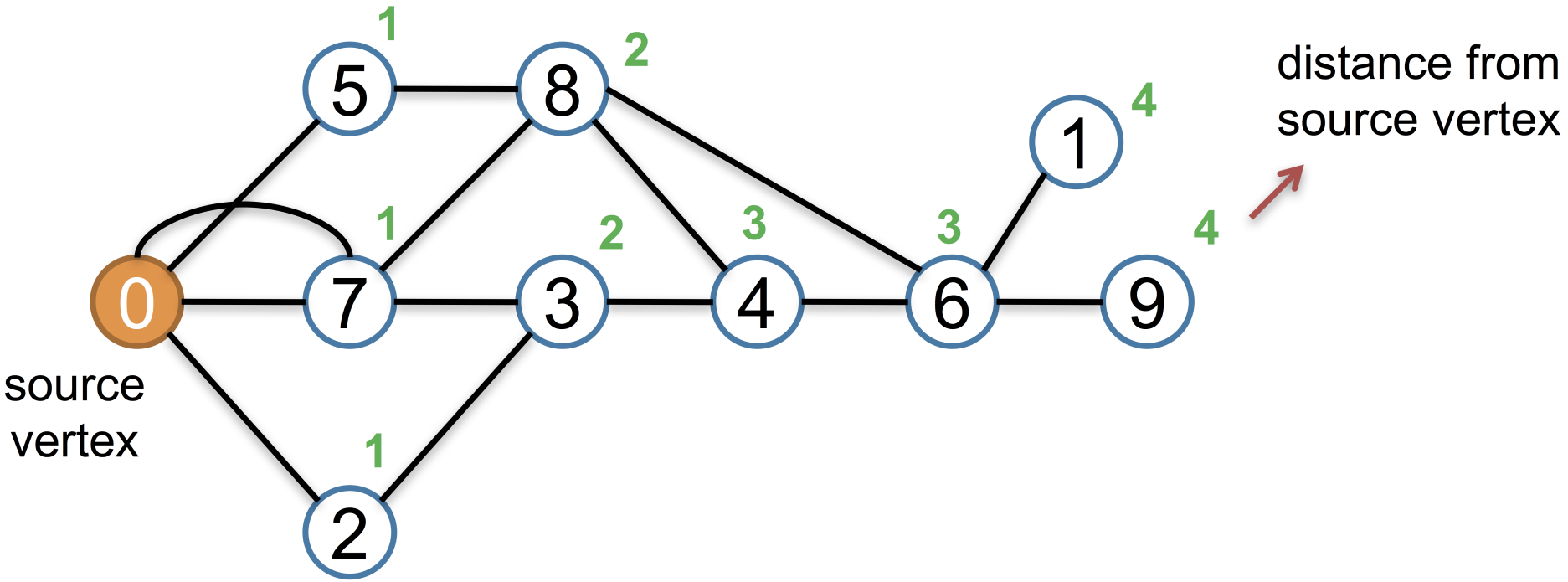


[1]Zhang, Peter, et al. "GBTL-CUDA: Graph algorithms and primitives for GPUs." *IPDPSW* '16.

[2]Wang, Yangzihao, et al. "Gunrock: GPU graph analytics." *ACM TOPC* '17.

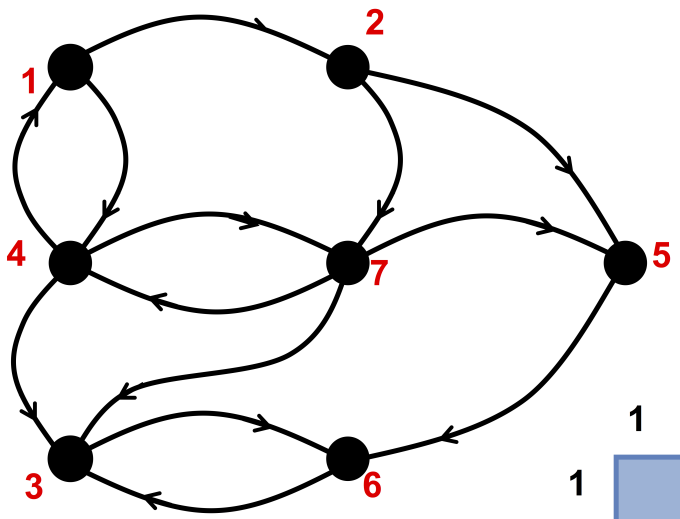
Overview

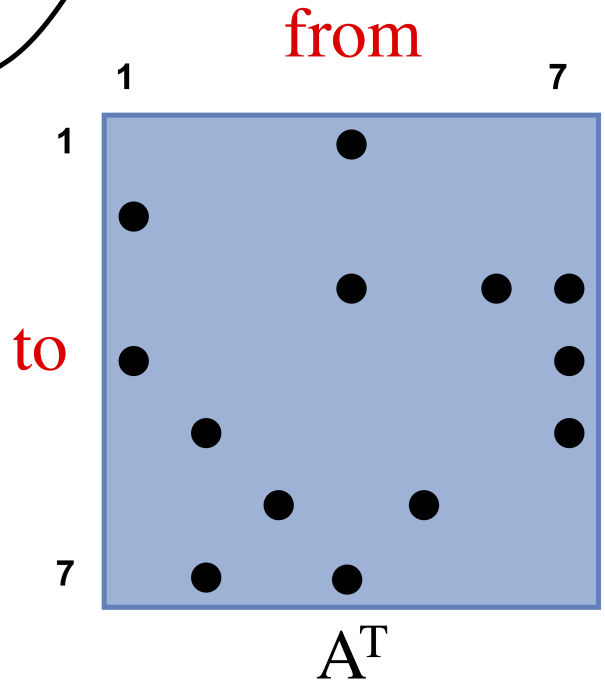
Breadth First Search (BFS)



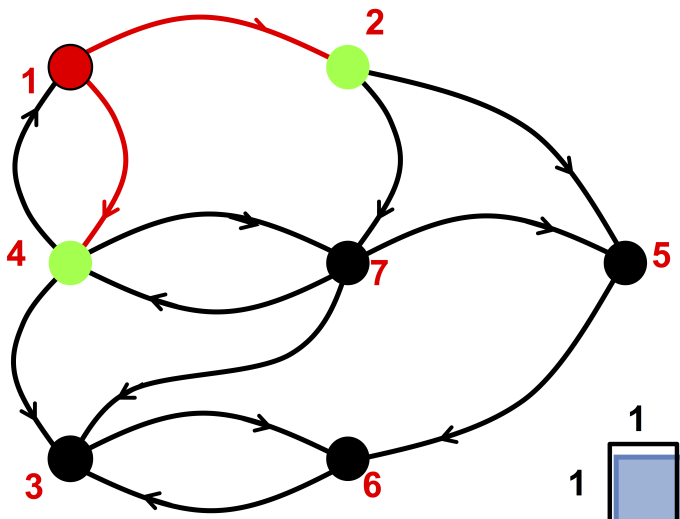
Breadth first search is a very important building block for other graph algorithms such as bipartite matching, maximum flow, strongly connected components, betweenness centrality, etc.

Breadth First Search Using Matrix Algebra

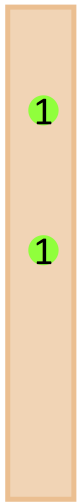


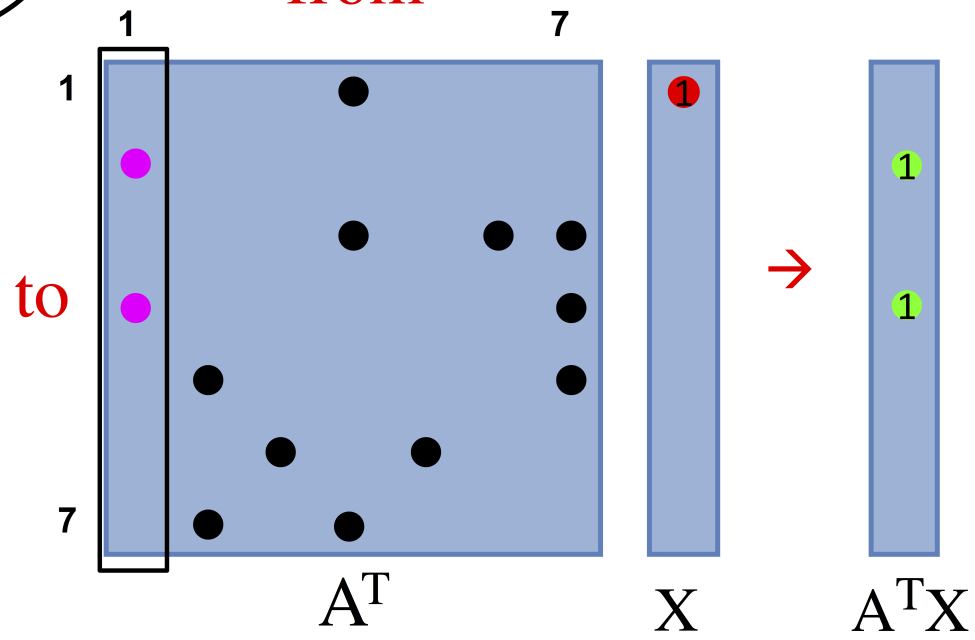


Iteration 1

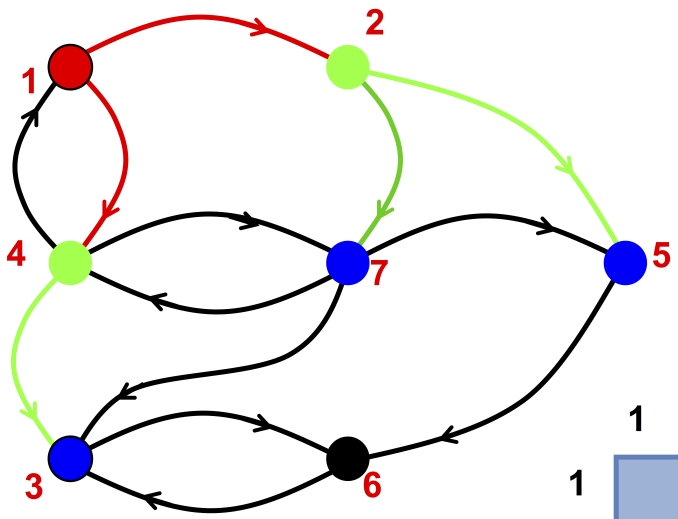


parents:

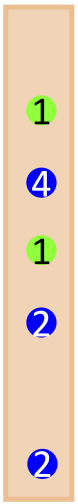




Iteration 2

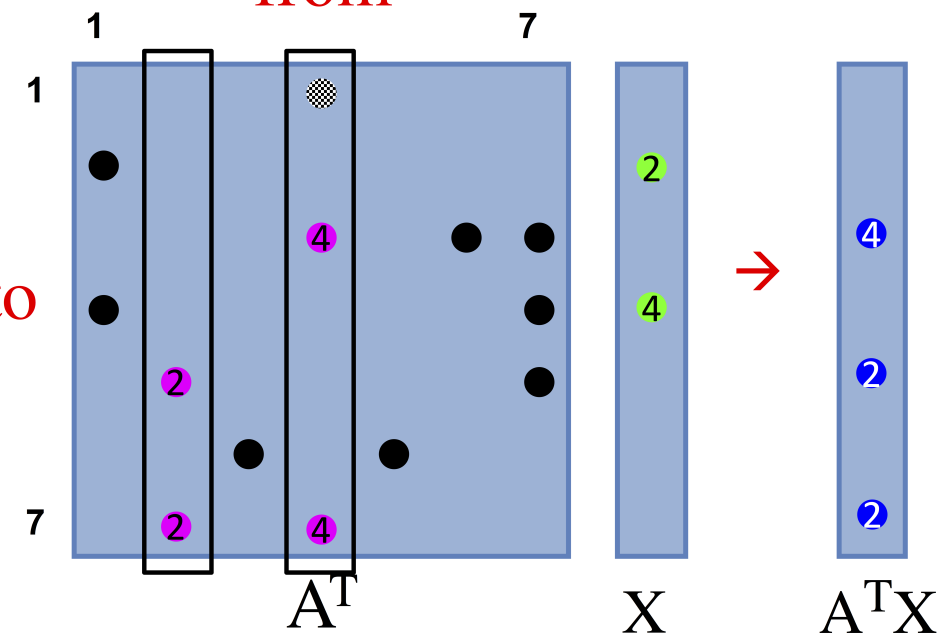


parents:

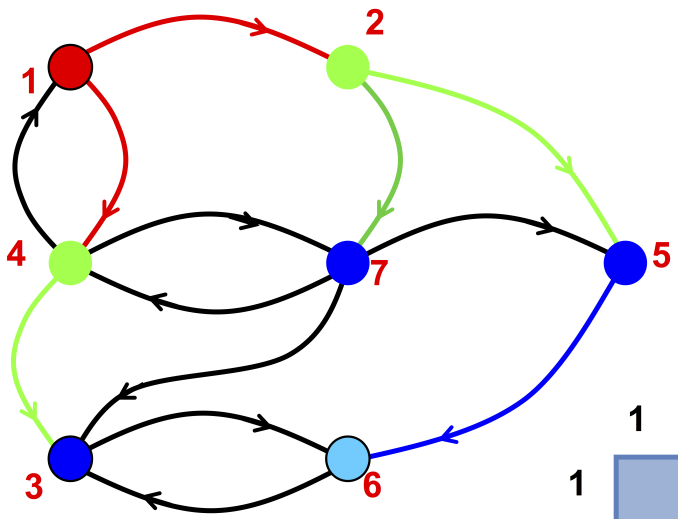


from

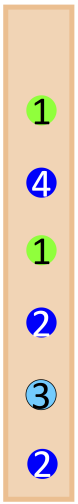
to

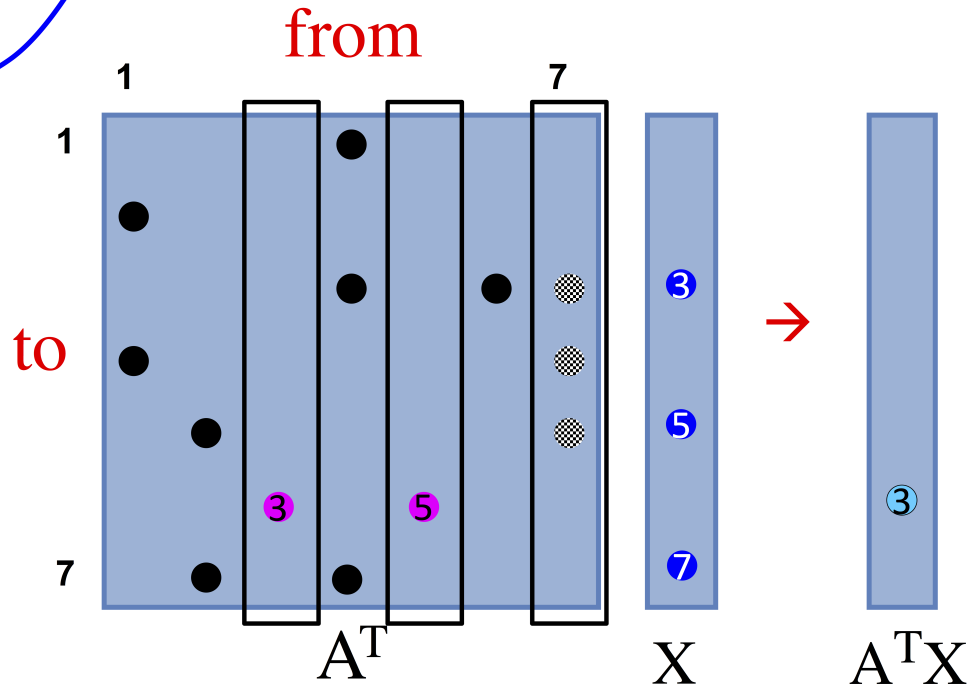


Iteration 3

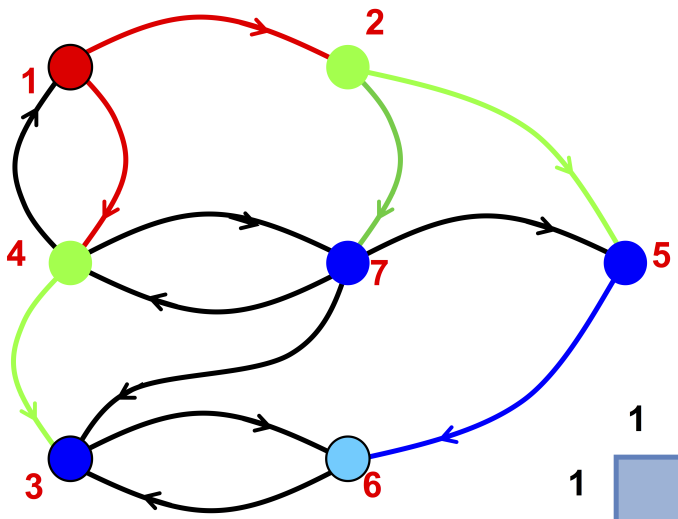


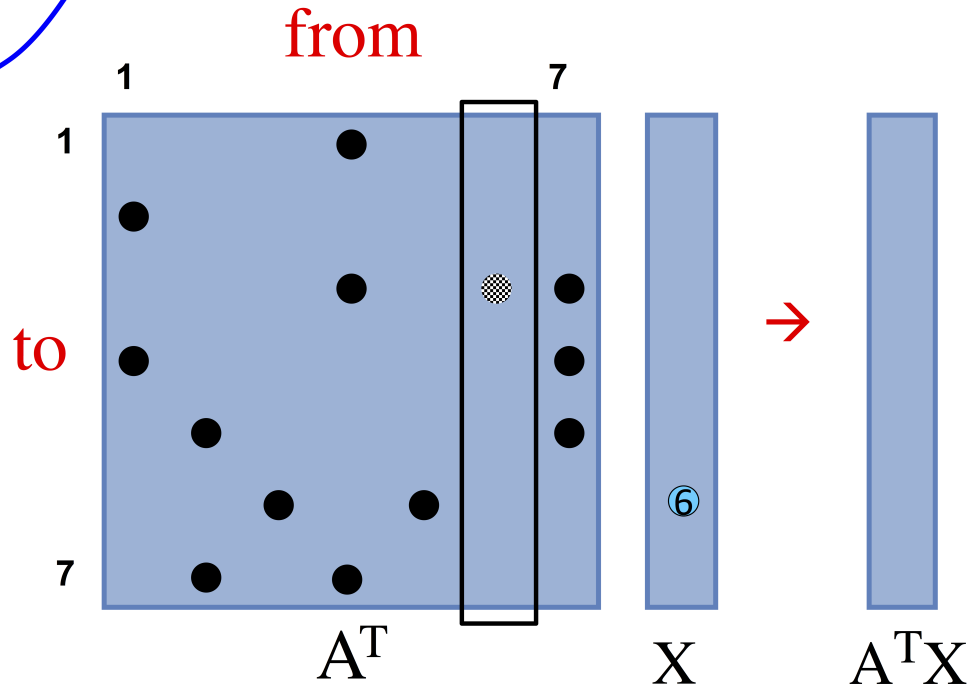
parents:



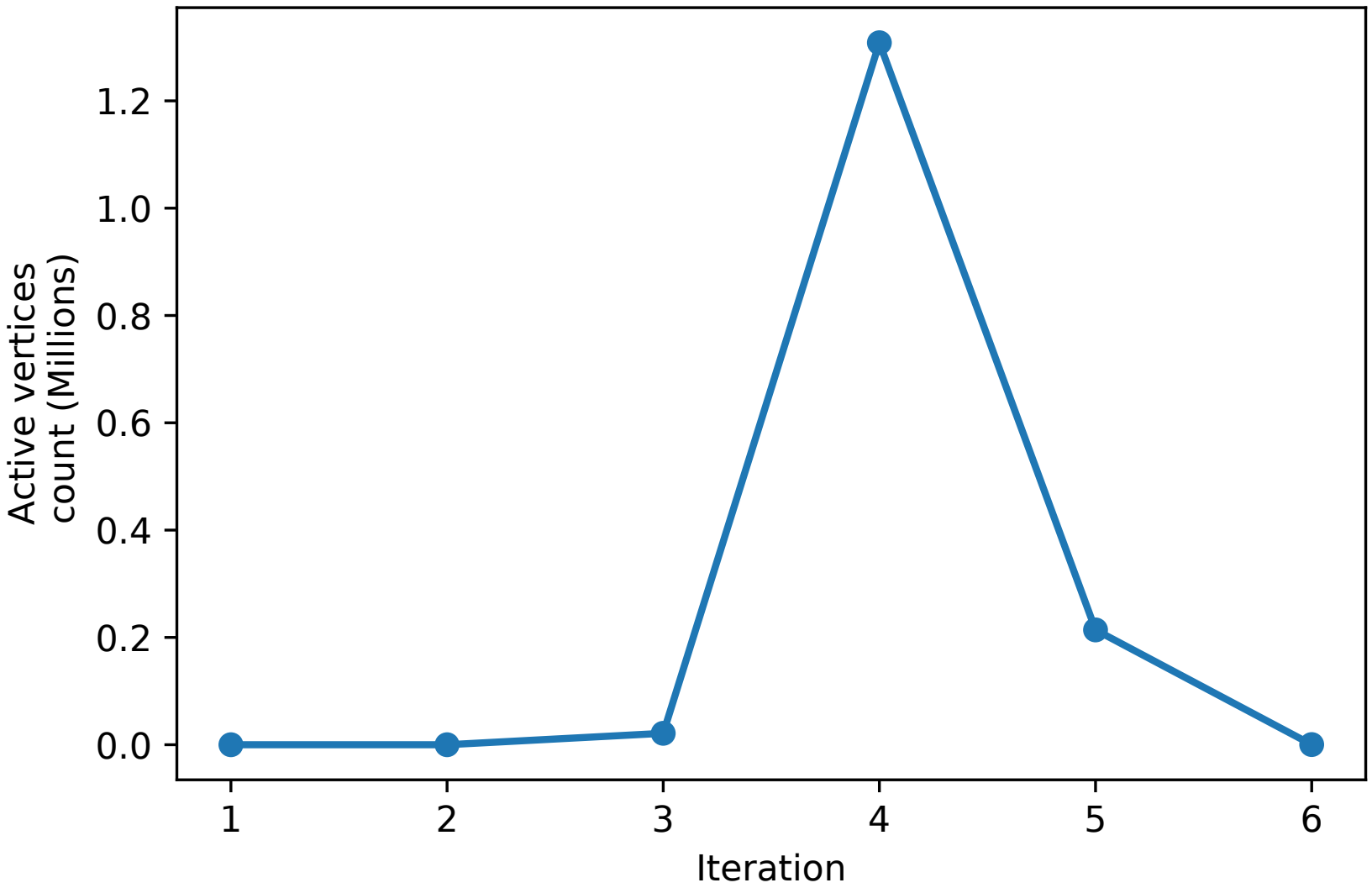


Final Iteration





Problem: Many active vertices = Slow



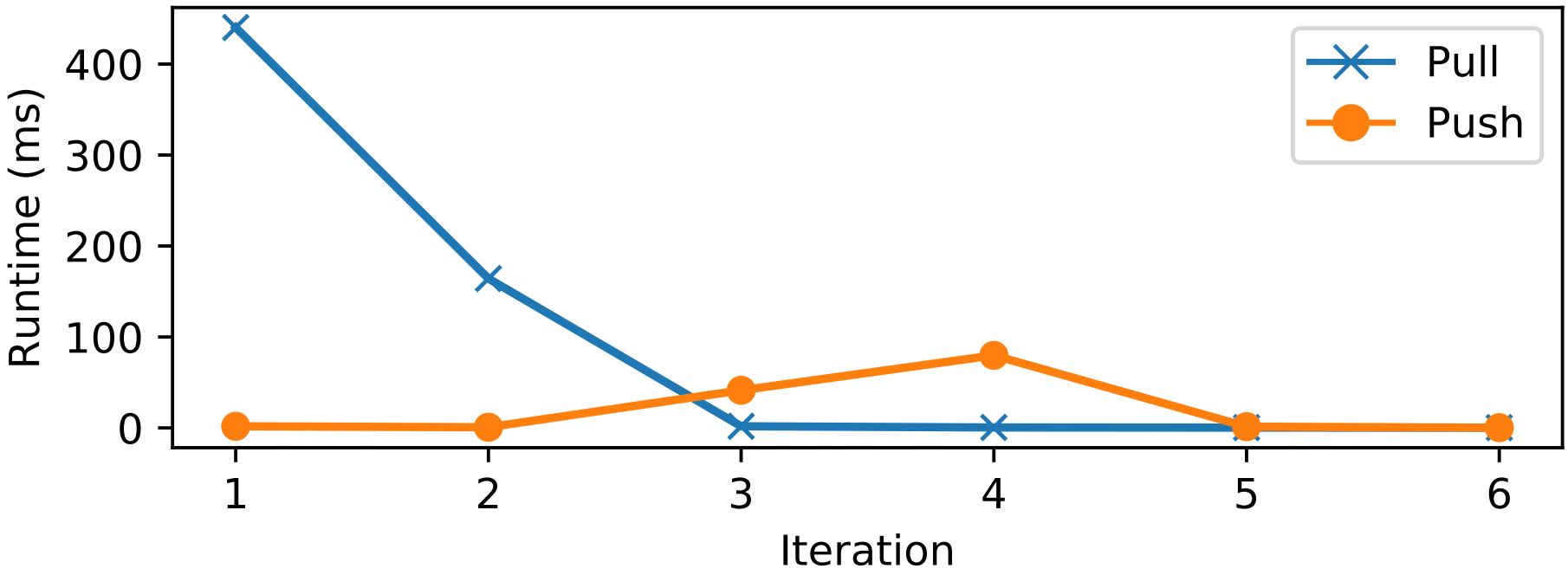
Overview

Direction-Optimized BFS

What if we instead of beginning with the active vertices and looking for their children, we start from unvisited vertices and look for their parents?

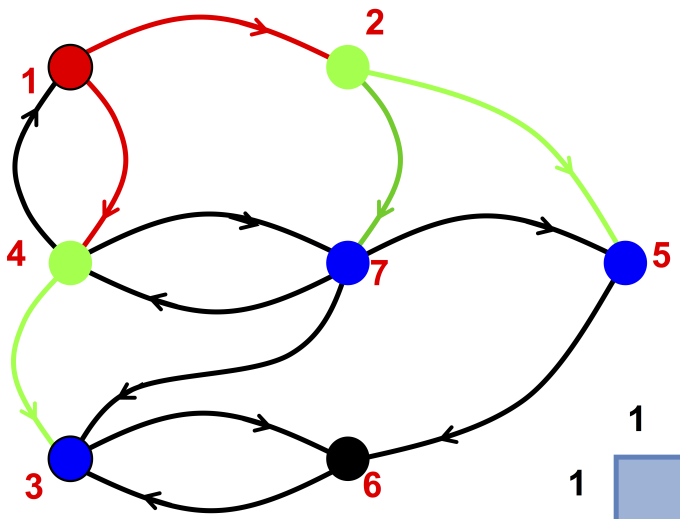
[1]Scott Beamer, Krste Asanovic, and David Patterson. "Direction-Optimizing Breadth-First Search". *SC* '12

Direction-optimized BFS

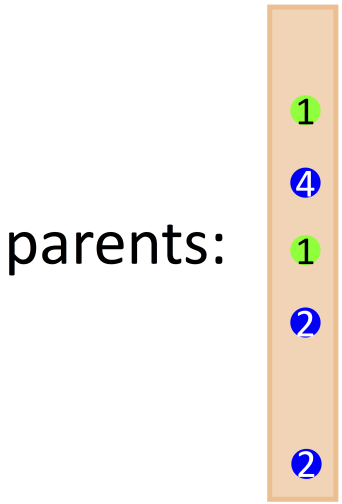


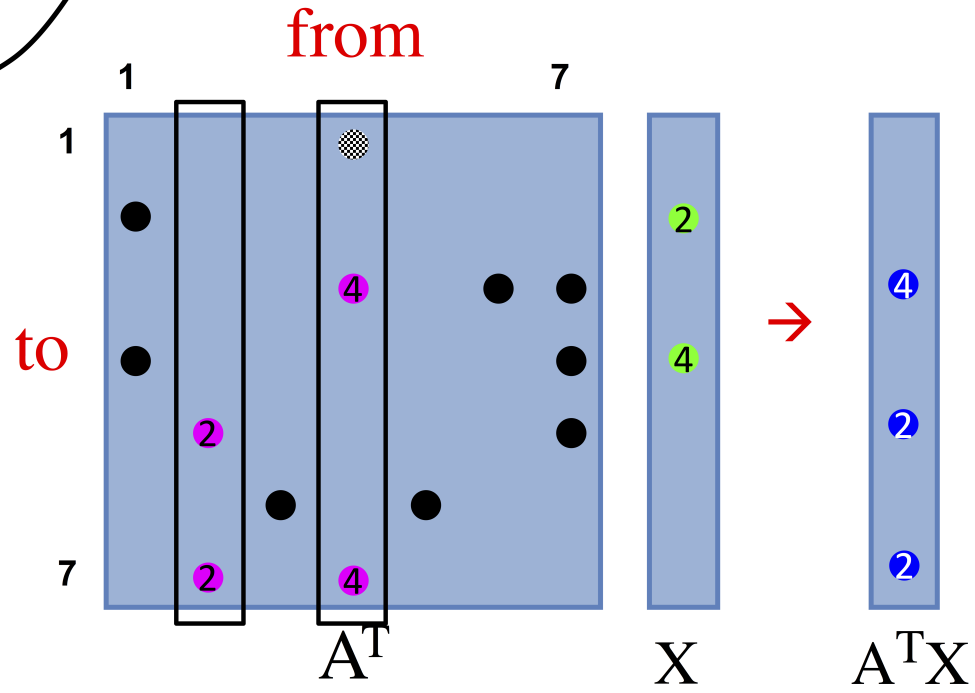
Problem: How to represent this in linear algebra?

Using Push (standard BFS, 5 memory accesses)

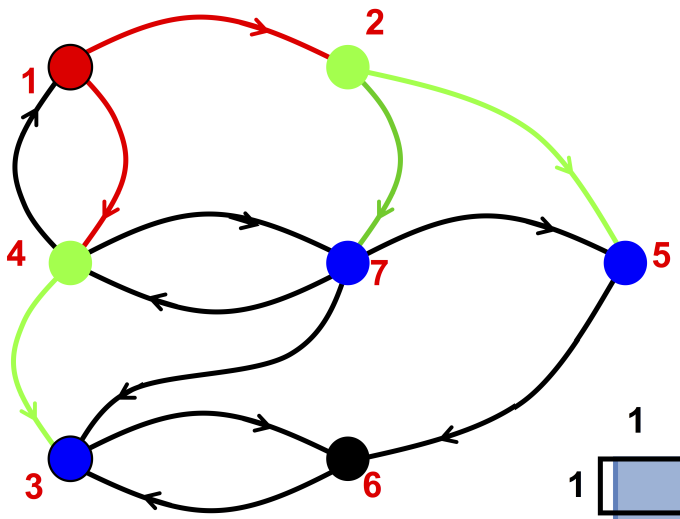


Column 2 means
Vertex 2's children



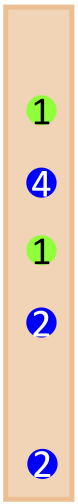


Using Pull (13 memory accesses)



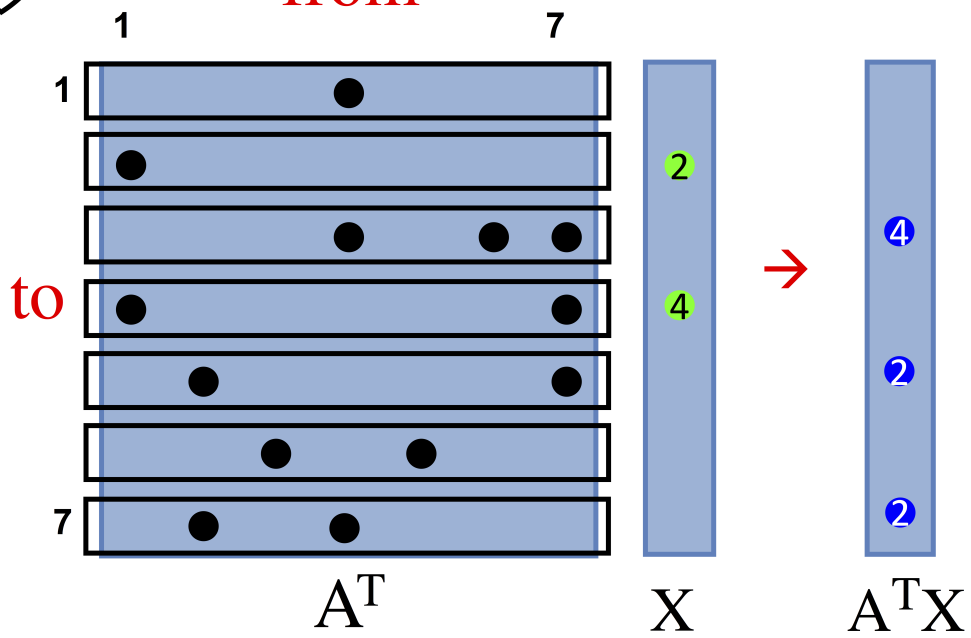
Row 2 means
Vertex 2's parents

parents:

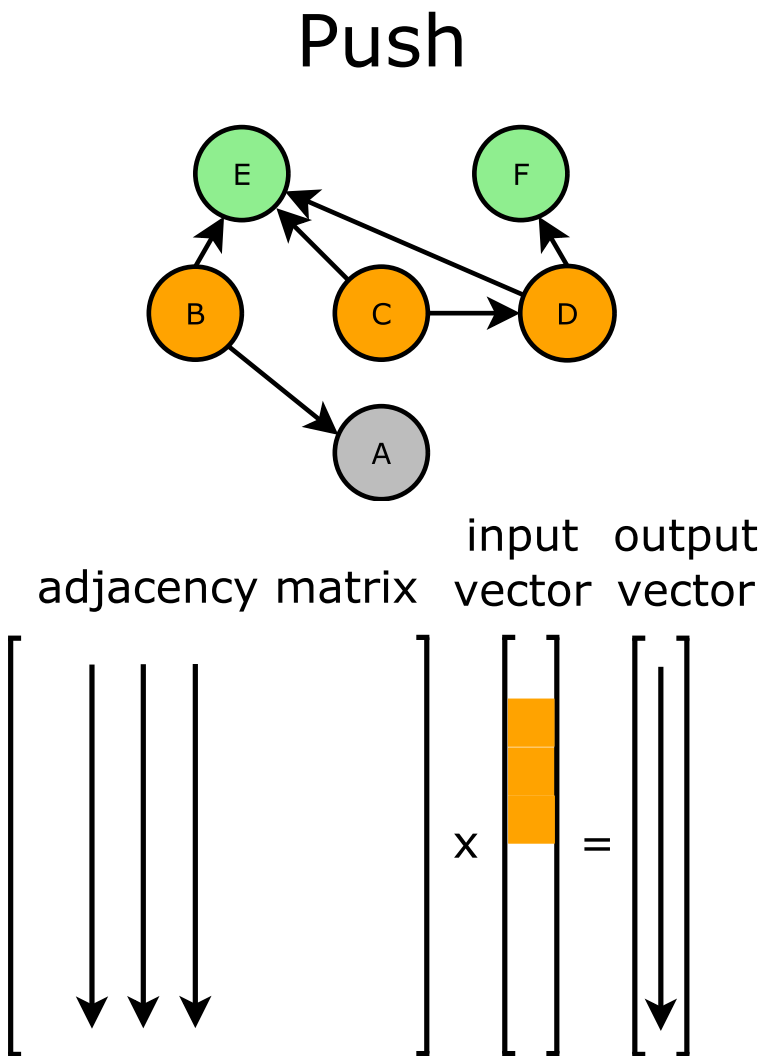


from

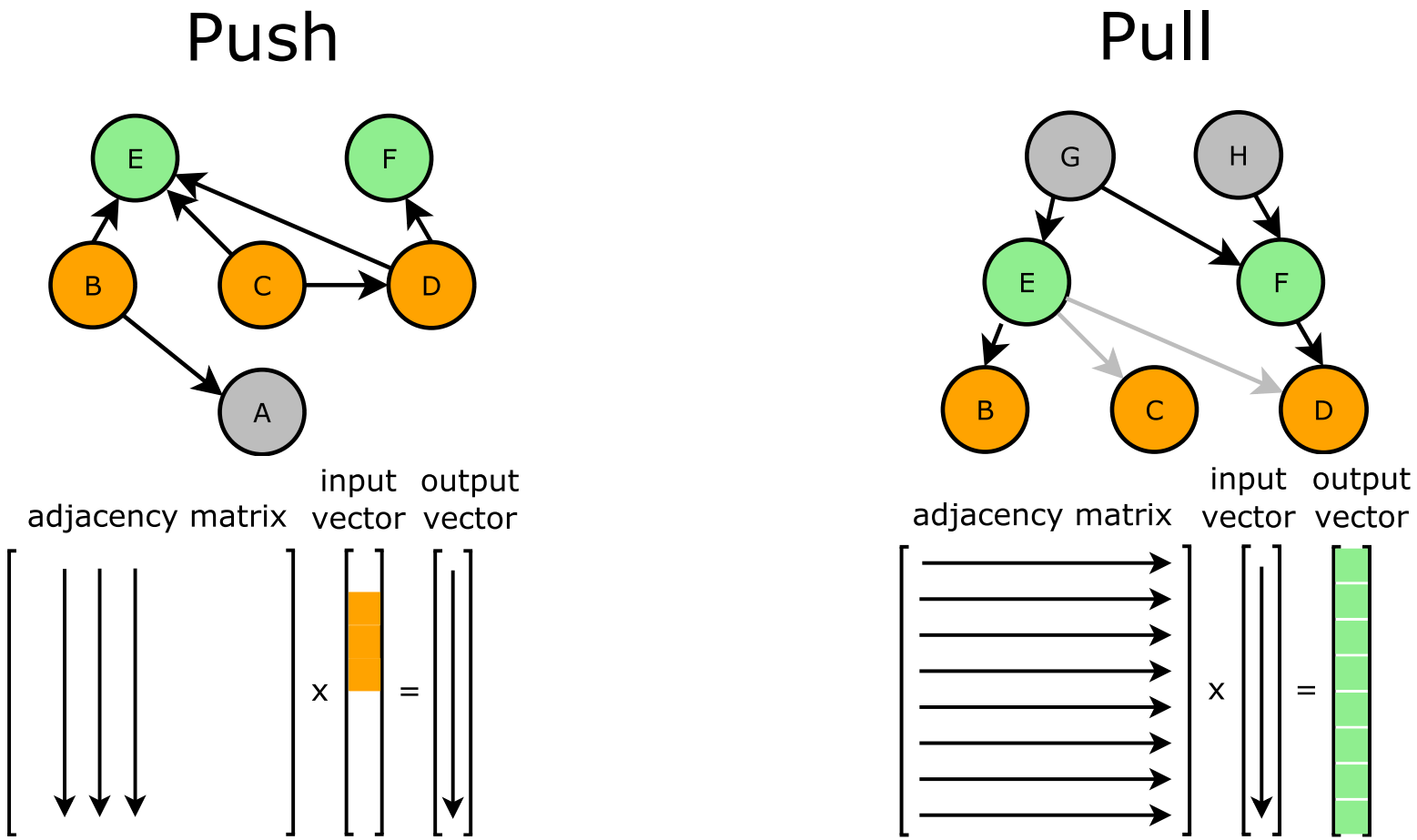
to



Push-pull is column- and row-based SpMV!



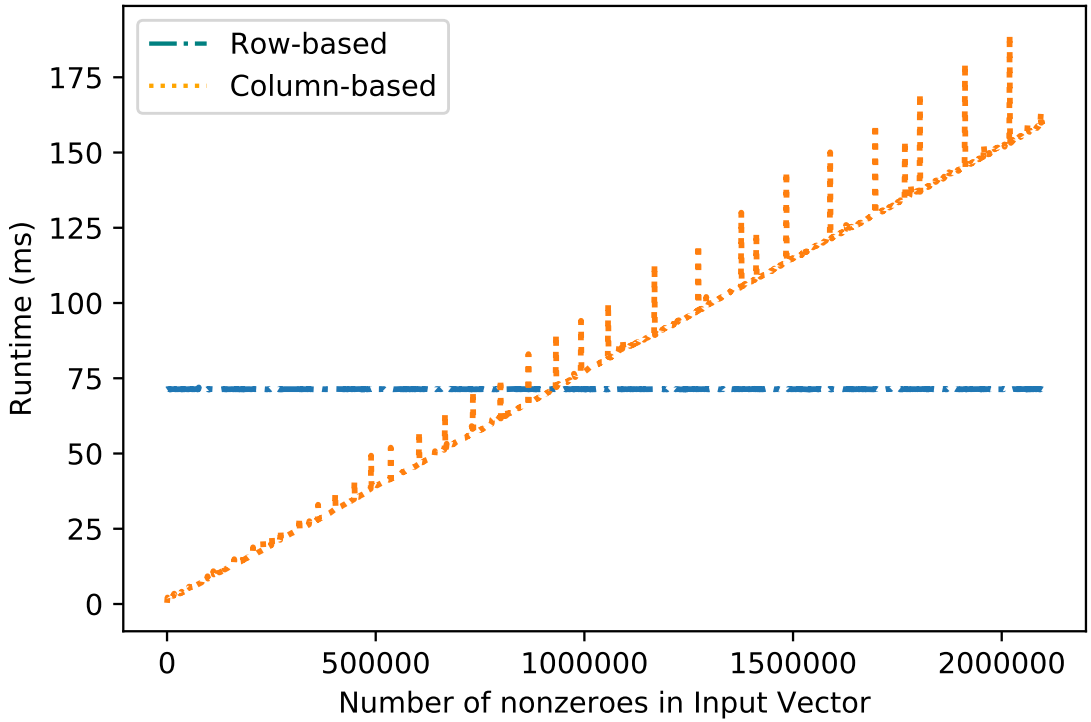
Push-pull is column- and row-based SpMV!



In linear algebra, these are merely two different memory access patterns for the same SpMV calculation.

But in graph analytics, they are two completely different algorithms!

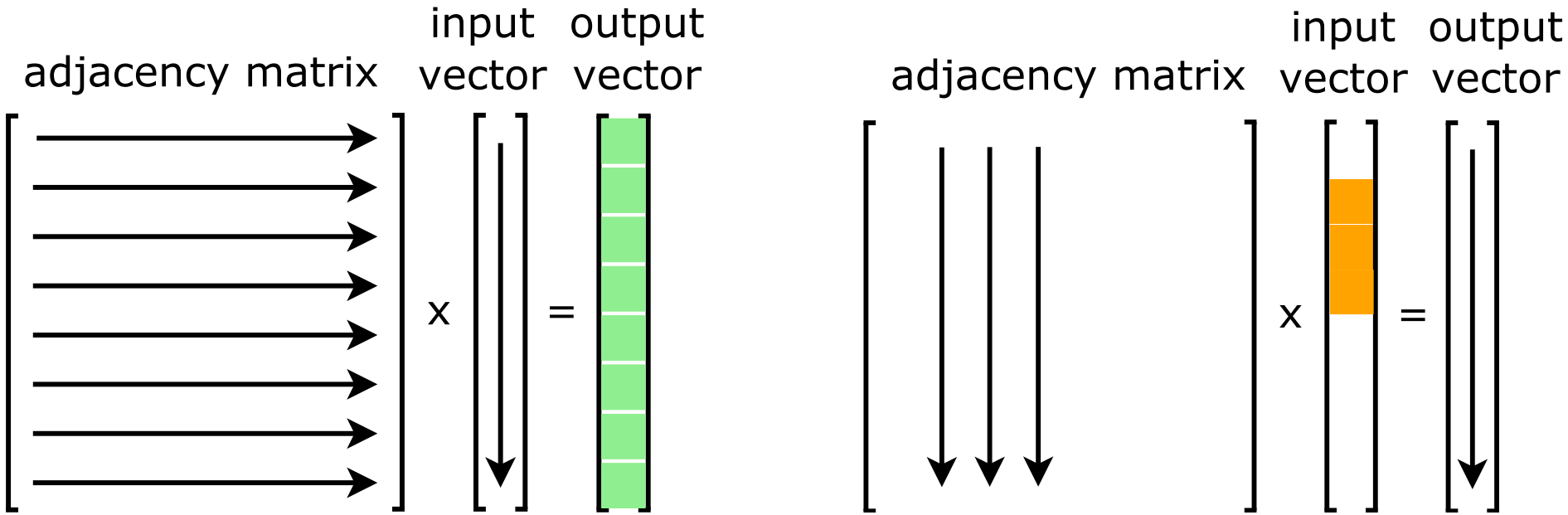
Column- vs. Row-based SpMV



- Column-based is faster when active vertices is sparse.
- Core optimization behind vertex-centric library Ligra[4].

[1]Shun, Julian, and Guy E. Blelloch. "Ligra: a lightweight graph processing framework for shared memory." *ACM SIGPLAN Notices* '13.

Optimization 1: Change of direction

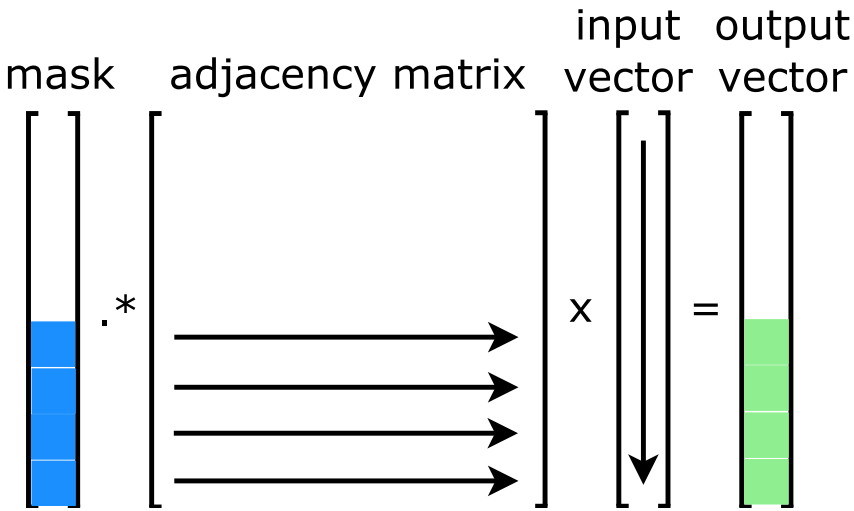


(a) Row-based SpMV

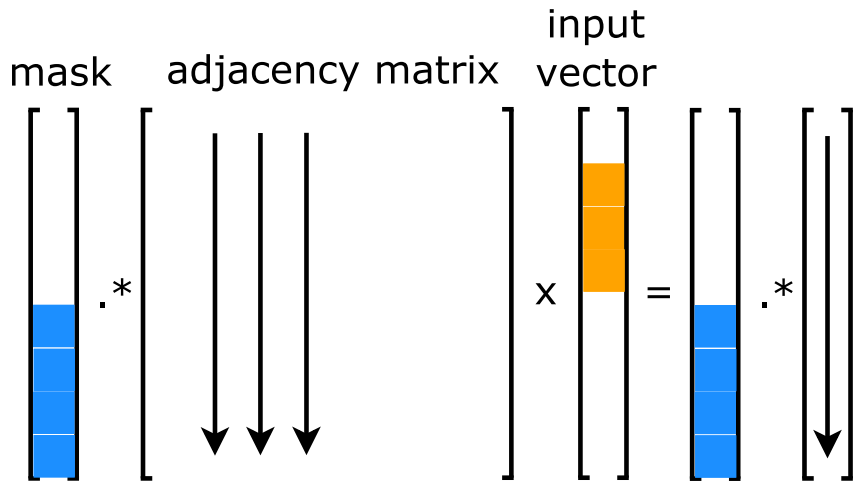
(b) Column-based SpMV

Previously known and used with success in Ligra, but only yields 8% speed-up on power law graphs.
Where is the missing performance?

Optimization 2: Masking

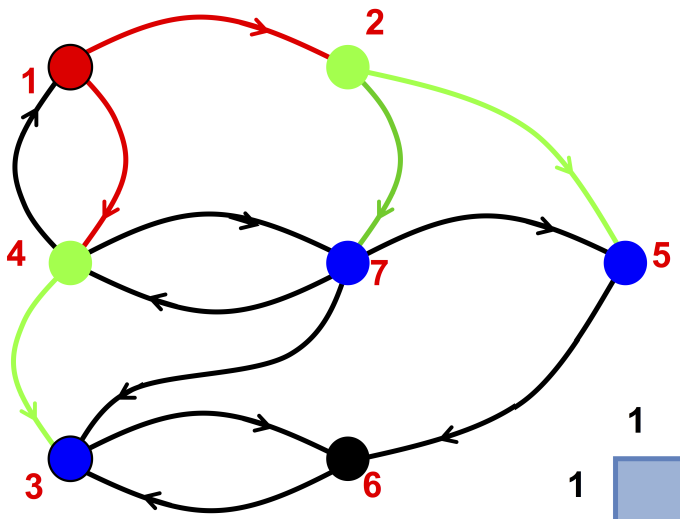


(a) Row-based SpMV (mask)



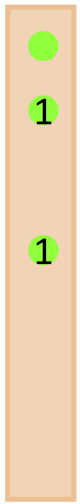
(b) Column-based SpMV (mask)

Using Masking, from 13 down to 9 memory accesses



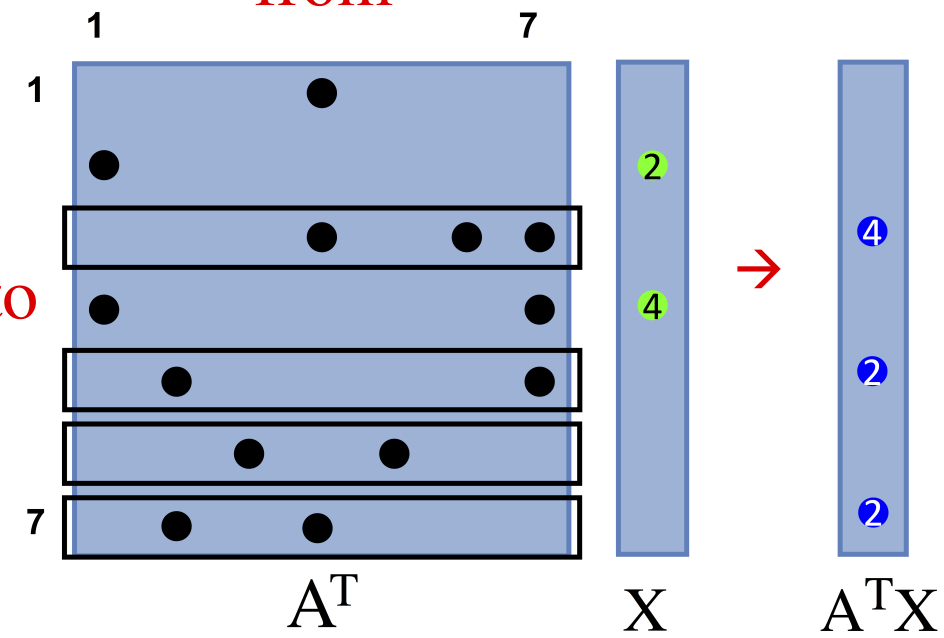
Row 2 means
Vertex 2's parents

parents:

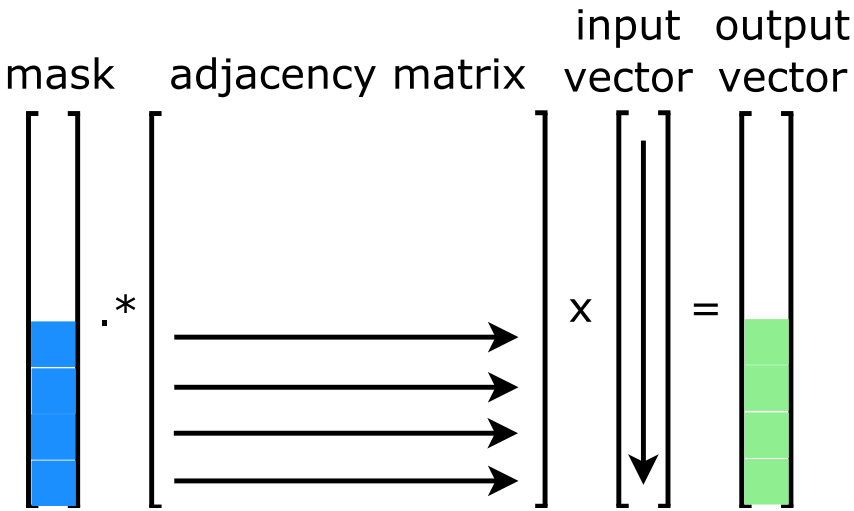


from

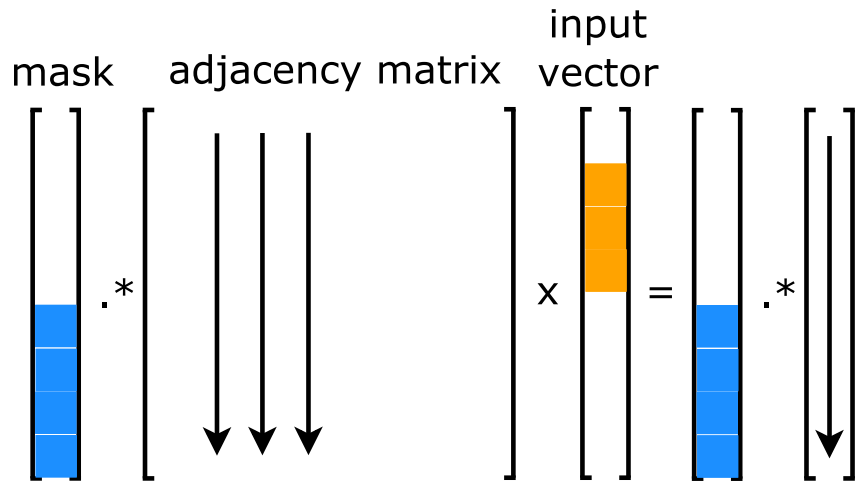
to



Optimization 2: Masking



(a) Row-based SpMV (mask)

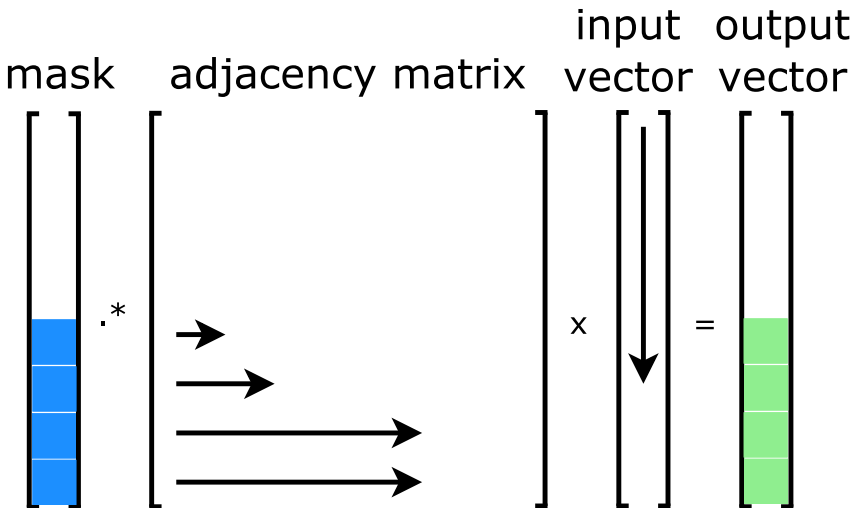


(b) Column-based SpMV (mask)

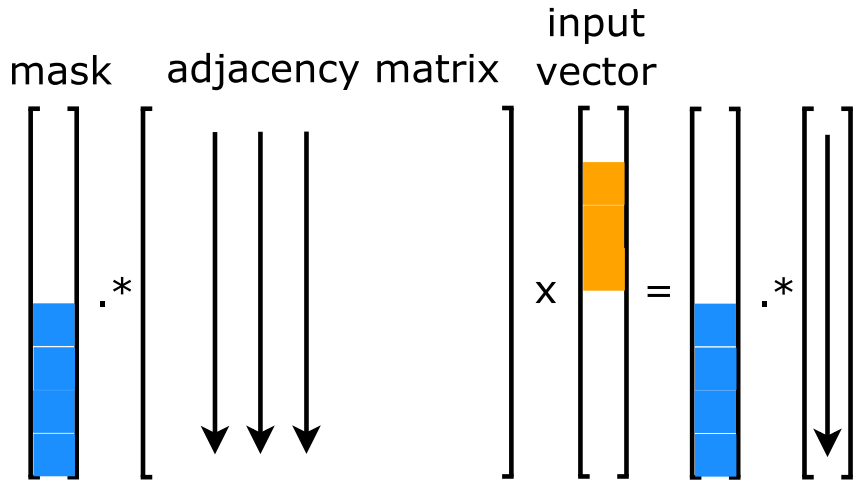
Complexity improves from $\mathcal{O}(dM)$ to $\mathcal{O}(d\ nnz(\mathbf{m}))$, where $\mathbf{m}$ is the mask vector.

This yielded a 154% speedup on power law graphs. Where is the rest of the performance?

Optimization 3: Early-exit

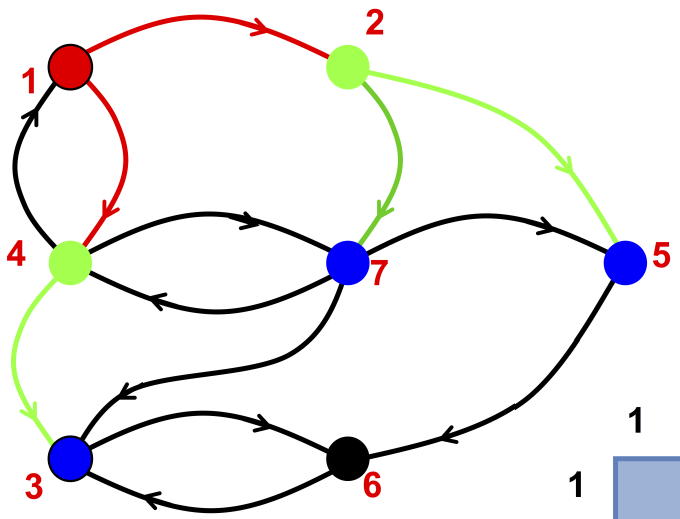


(a) Row-based SpMV (mask and early-exit)



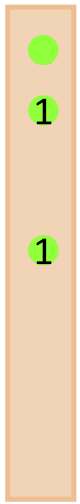
(b) Column-based SpMV (mask) cannot early-exit

Using Early-Exit, from 9 down to 5 memory accesses



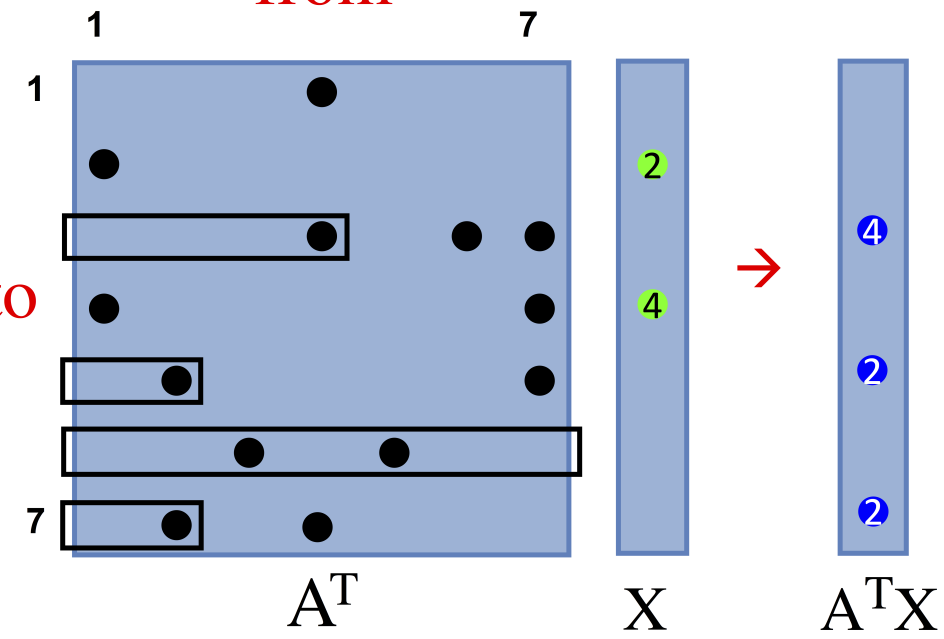
Row 2 means
Vertex 2's parents

parents:

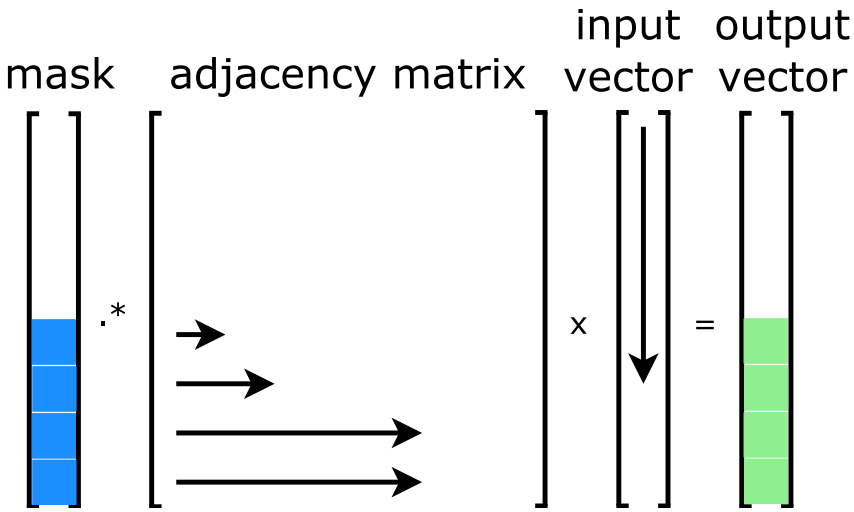


from

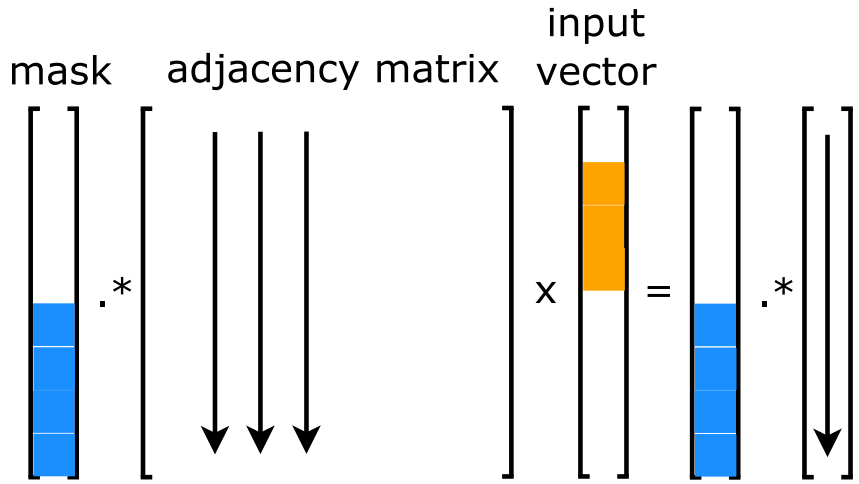
to



Optimization 3: Early-exit



(a) Row-based SpMV (mask and early-exit)



(b) Column-based SpMV (mask) cannot early-exit

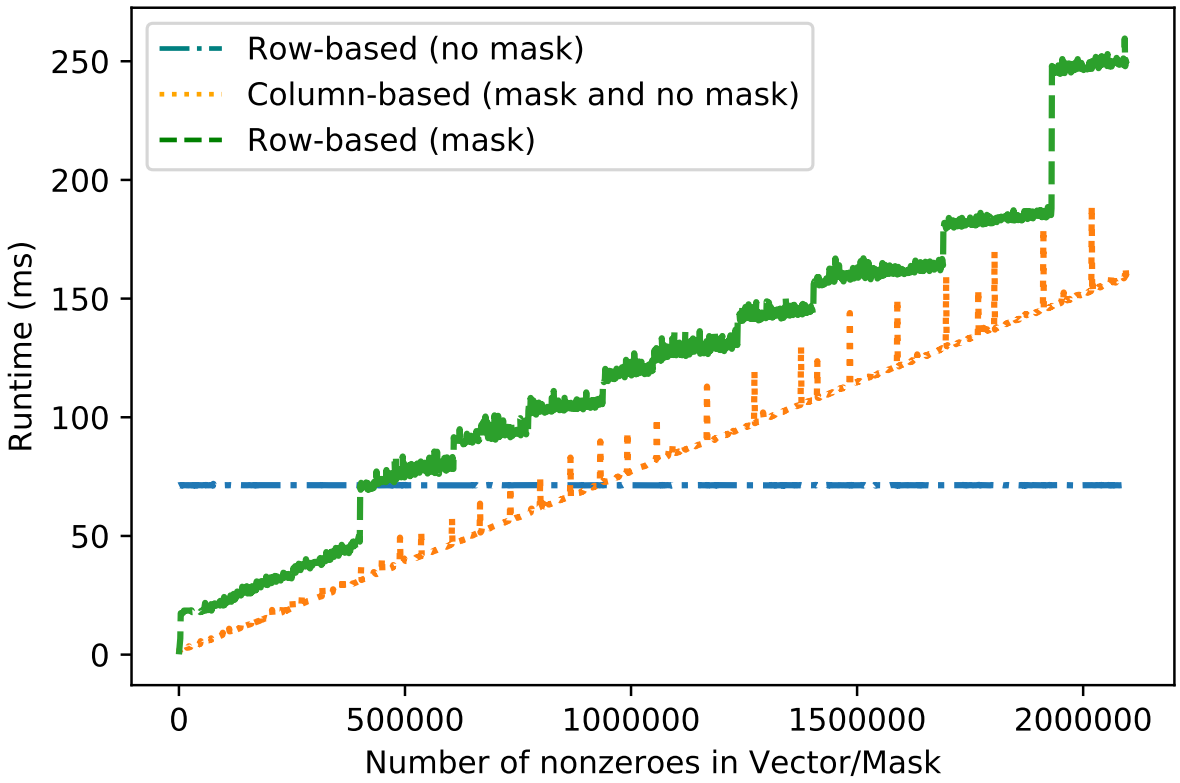
Only works for Boolean semirings.
This yielded 302% speedup on power law graphs.

Overview

Experimental setup

- CPU: Intel 4-core E5-2637 v2 Xeon CPU @ 3.50GHz, 556GB RAM
- GPU: NVIDIA K40c, 12GB RAM

Complexity summary

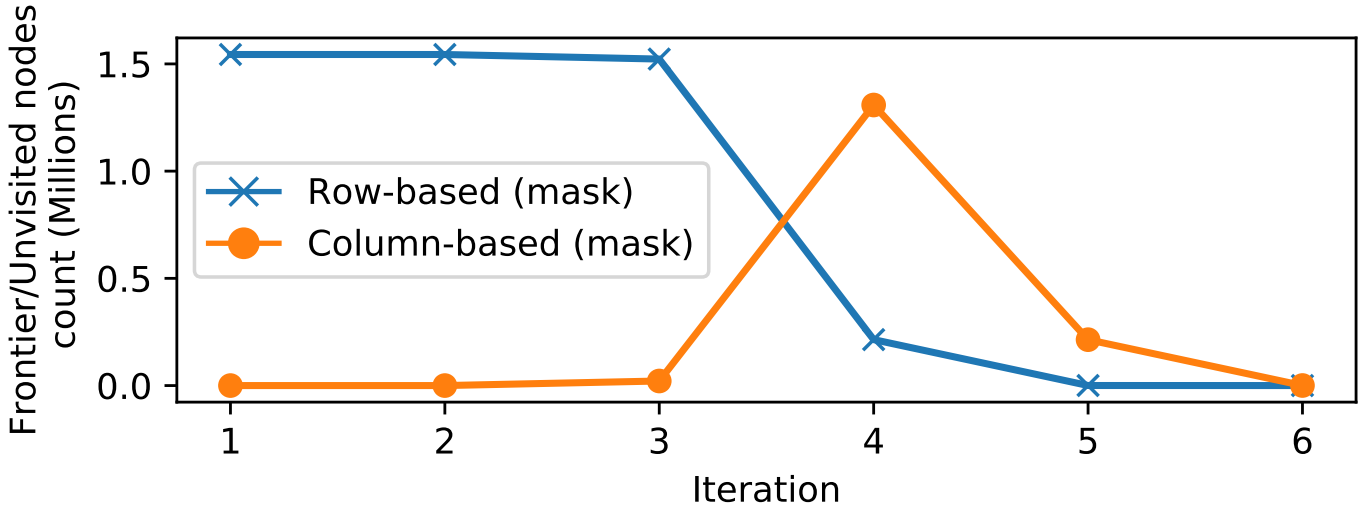


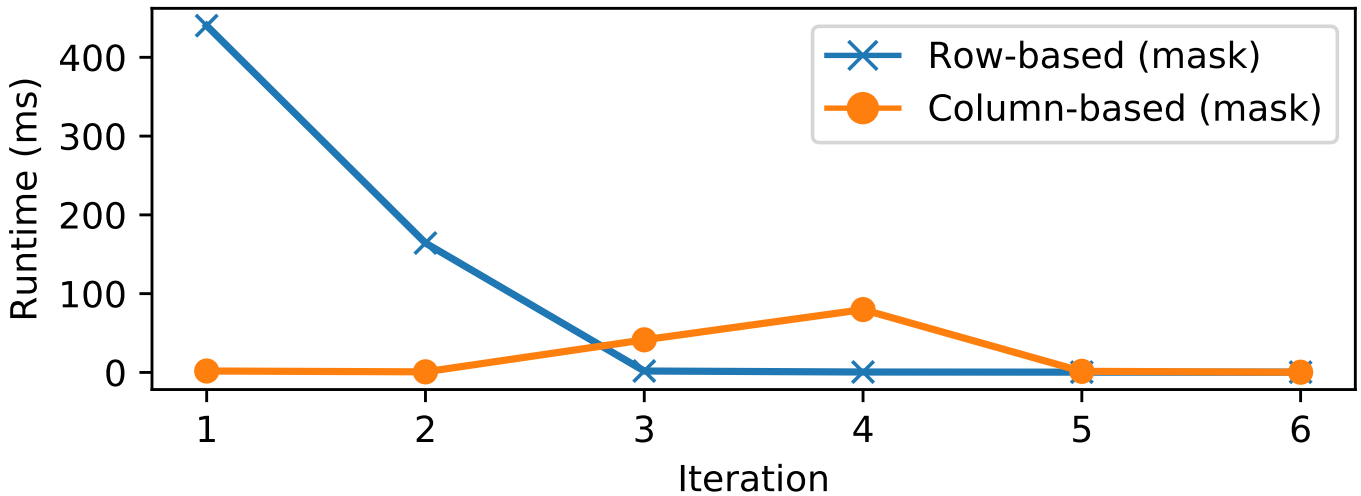
Row-based (no mask): $\mathcal{O}(dM)$

Row-based (mask): $\mathcal{O}(d\ nnz(\mathbf{m}))$

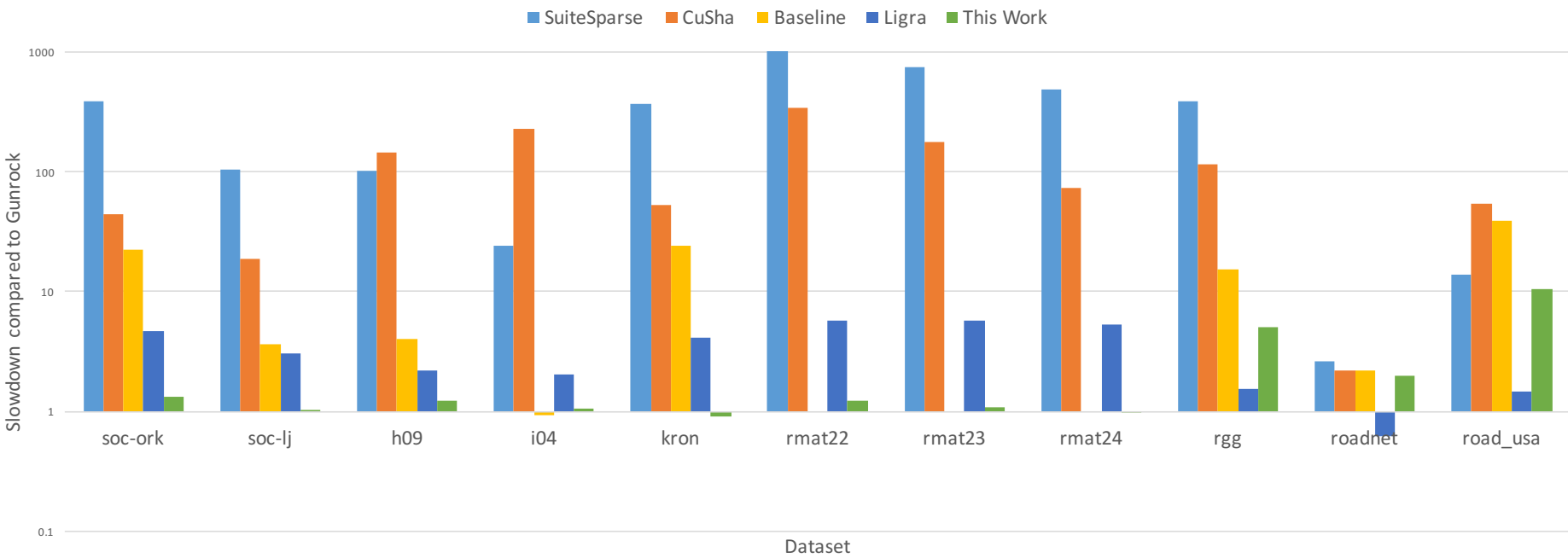
Column-based (mask and no mask): $\mathcal{O}(d\ nnz(\mathbf{f}) \log M)$

Runtime explained by crossing of frontier and unvisited node counts

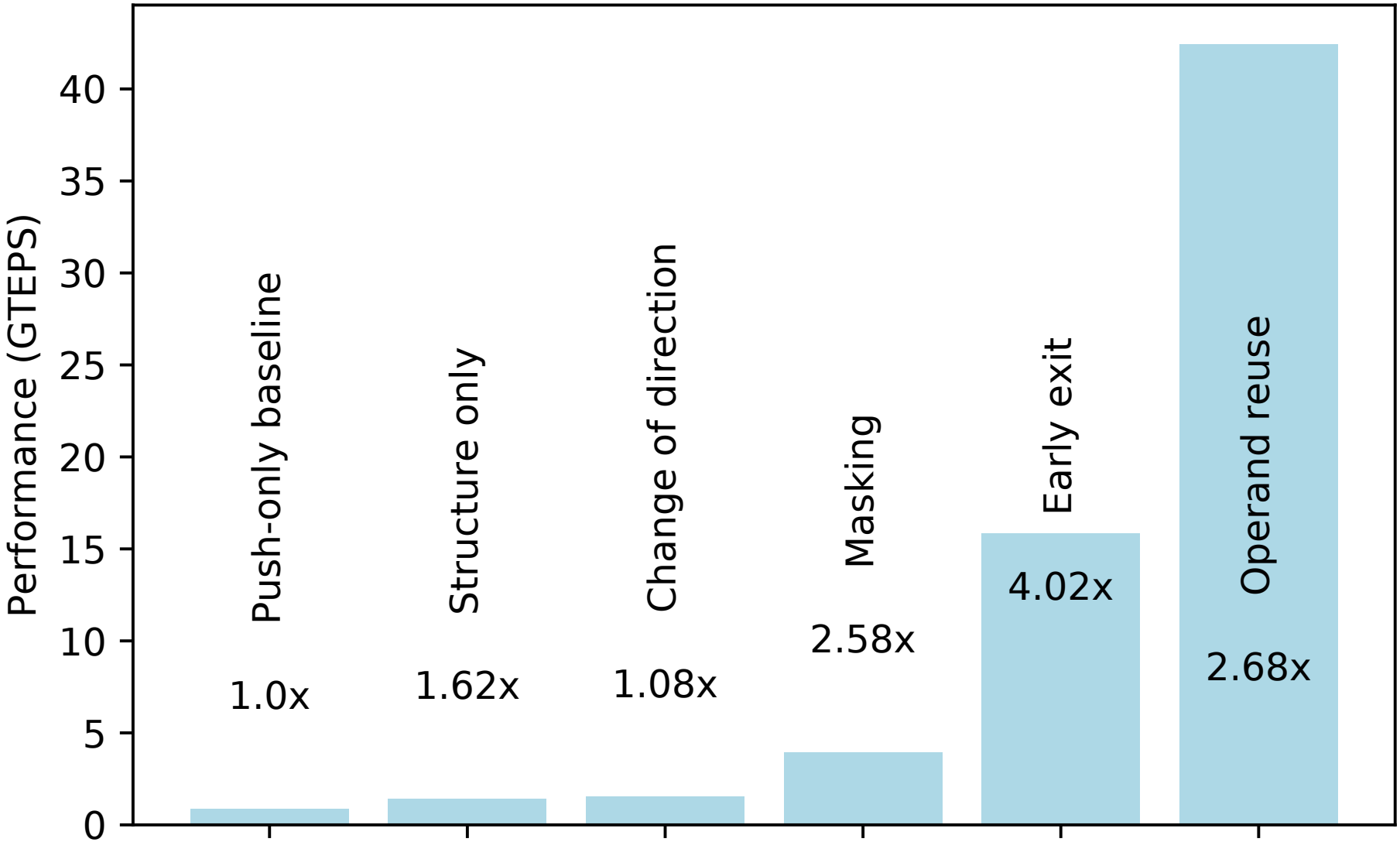




Performance close to Gunrock on scale-free graphs



Where the speedup comes from



Overview

Future work

- Task graph: Dynamically fuse operations in data-dependent way
- Distributed memory: Use this work as component in distributed, multi-GPU BFS
- Other graph algorithms: Ideas used here can be generalized to other graph algorithms such as adaptive PageRank and maximal independent set

Conclusion

- We decomposed direction-optimized BFS into 3 optimizations:
  1. Change of direction: column-based scales by number of active vertices, but row-based is a constant
  2. Masking: each vertex only needs to be visited once
  3. Early exit: finding a single parent that is an active vertex is sufficient
- Linear algebra-based graph frameworks are just as expressible and performant as their vertex-centric counterparts, while being more concise and portable
- Thinking about graphs in terms of linear algebra provides a systematic way of discovering and generalizing graph optimizations

Acknowledgments

- The Gunrock team
- Scott McMillan
- NVIDIA
  - for their generous hardware support
- Funding support
  - NSF Award # CCF-1629657
  - DARPA XDATA program under US Army award W911QX-12-C-0059
  - DARPA HIVE program under agreement number FA8650-18-2-7836
  - LBNL's DOE contract DE-AC02-05CH11231
  - DOE's OASCR contract DE-AC02-05CH11231
  - NNSA and DOE's Office of Science under the Exascale Computing Project 17-SC-20-SC

Questions?

Code is available at: https://github.com/owensgroup/push-pull