Linear Algebra is the Right Way to Think About Graphs

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

[1] Department of Electrical and Computer Engineering, UC Davis
[2] Computational Research Division, LBNL
[3] Department of Electrical Engineering and Computer Sciences, UC Berkeley

Problem

- High-performance graph processing is an important problem
- Increasing graph sizes require accelerators (e.g. GPUs)
- Mapping irregular graph problems onto the GPU is hard
- What is the "best" interface for graph processing?

Matrix-graph duality

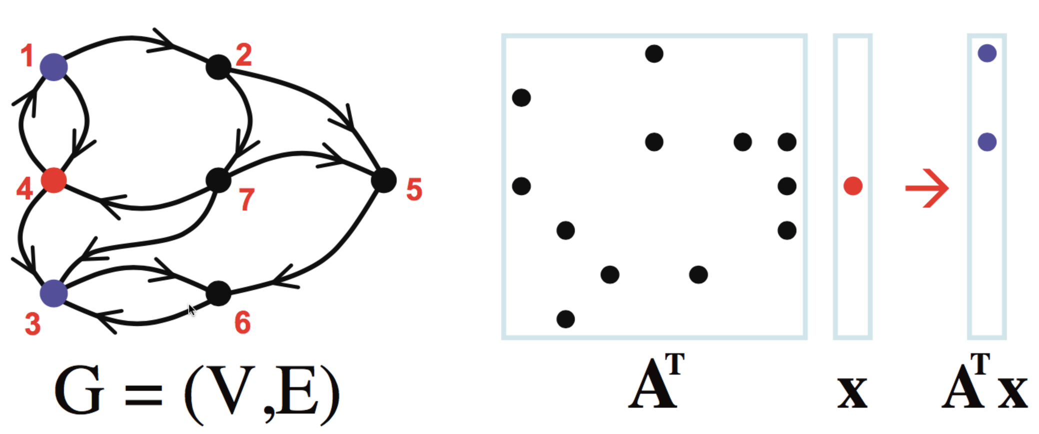


Figure: We rely on this connection first noted by Konig.

3 Types of Sparsity

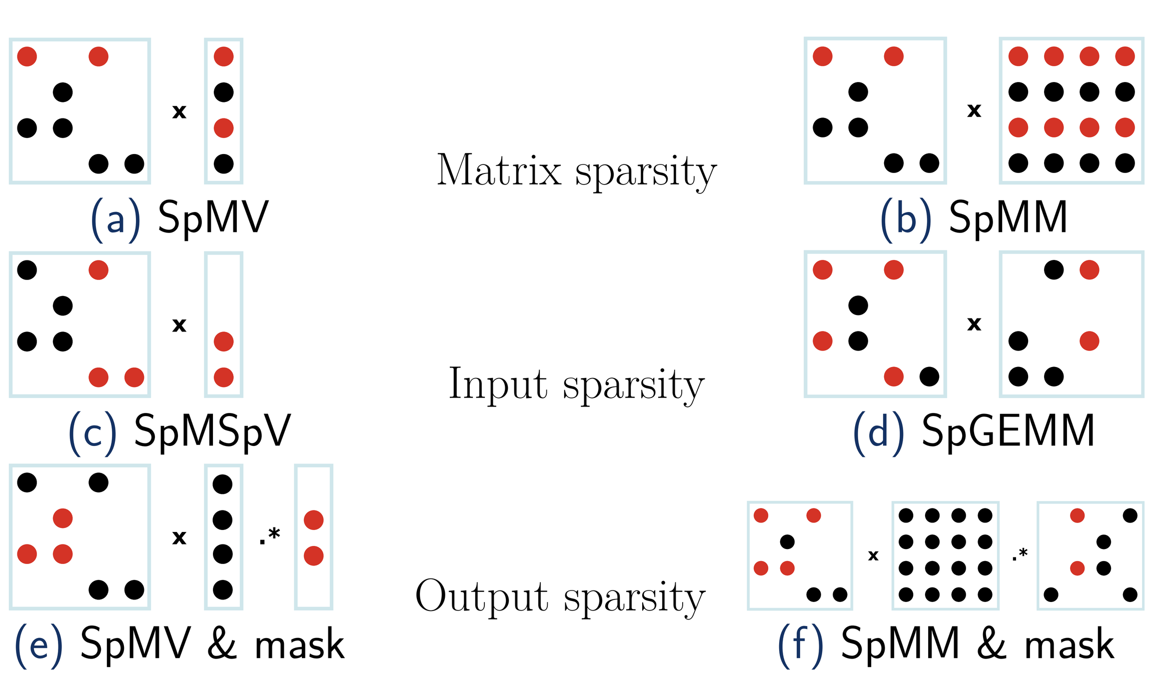


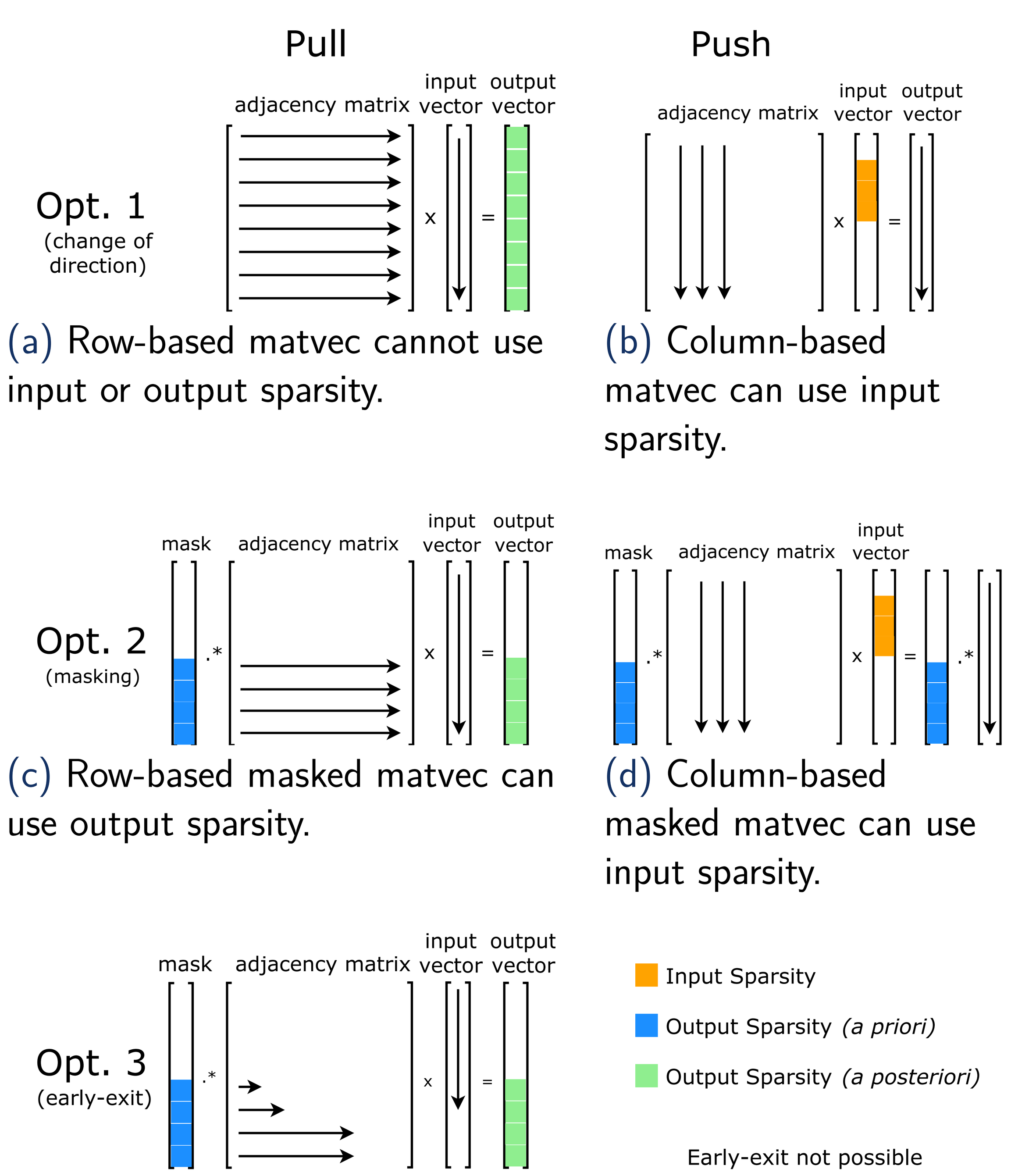
Figure: We take advantage of 3 types of sparsity in our work: (a) matrix sparsity or lefthand-side sparsity, (b) input sparsity or righthand-side sparsity, (c) output sparsity or mask sparsity. Their complexity results are given below.

Complexity Results

Operation		Expected Cost
SpMV	no mask	$\mathcal{O}(dM)$
SpMV	masked	$\mathcal{O}(d\, nnz(\mathbf{m}))$
SpMSpV	no mask	$\mathcal{O}(d\, nnz(\mathbf{f}) \log nnz(\mathbf{f}))$
SpMSpV	masked	$\mathcal{O}(d\, nnz(\mathbf{f}) \log nnz(\mathbf{f}))$

Table: Four sparse matvec variants and their associated cost, measured in terms of number of expected memory accesses into the $M - by - M$ sparse matrix $\mathbf{A}$ required. We assume the sparse matrix is Erdős-Rényi with mean degree d. The $\mathbf{m}$ and $\mathbf{f}$ refer to the mask and righthand-side vector respectively.

Direction-optimized BFS



(a) Row-based matvec cannot use input or output sparsity.

(b) Column-based matvec can use input sparsity.

(c) Row-based masked matvec can use output sparsity.

(d) Column-based masked matvec can use input sparsity.

(e) Row-based masked matvec can use output sparsity and early-exit.

(f) Column-based masked matvec cannot early-exit.

Figure: 3 optimizations making up direction-optimized BFS.

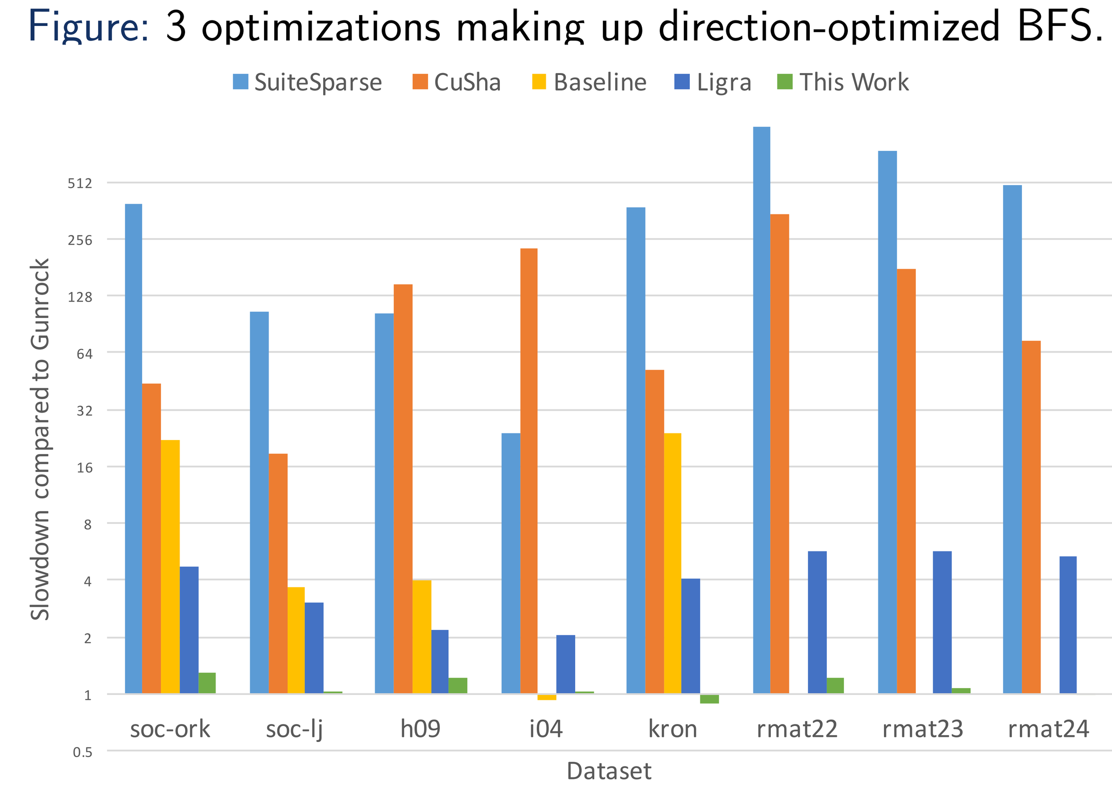
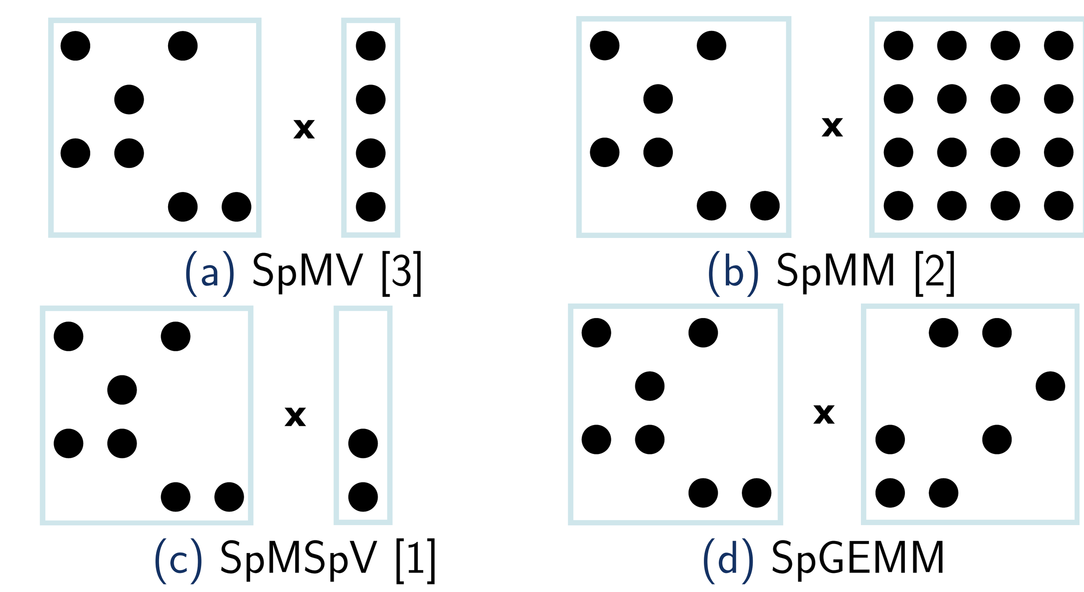


Figure: Comparison to other graph libraries (SuiteSparse, CuSha, a baseline push-based BFS, Ligra, and Gunrock)

Ingredient I: Operations



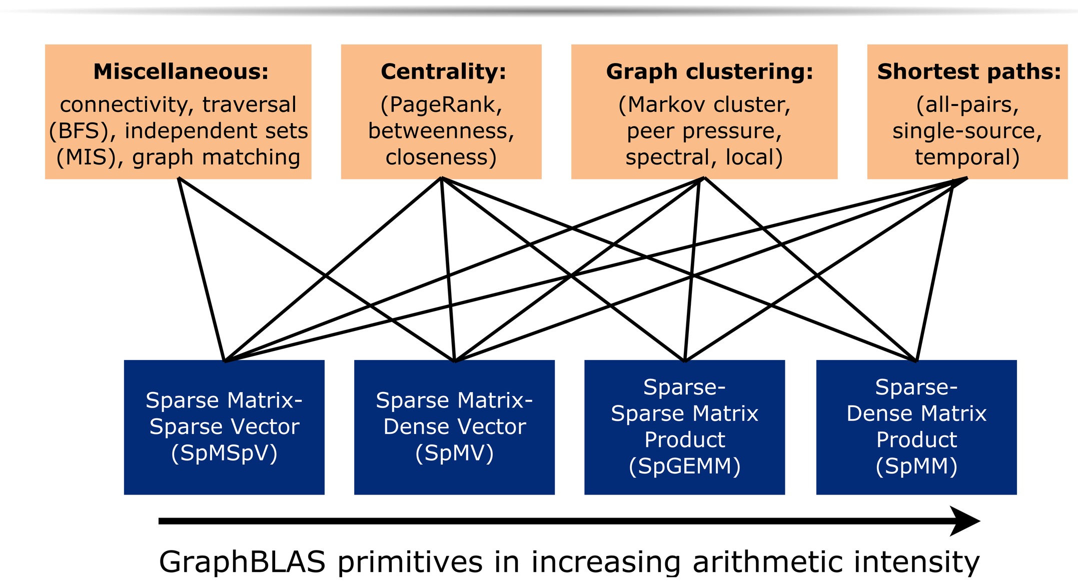
Ingredient II: Operators

Name	Semiring	Application
Real field	$\{+, \times, \mathbb{R}\}$	Classical 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

Semiring notation: (Add, Multiply, Domain)

- Add: Combines edges at a vertex
- Multiply: Traverses edges
- Domain: Vertex/edge attributes

Operations + Operators = GraphBLAS



Metrics

We evaluate graph frameworks using the following metrics:

- **Expressible:** Sparse matrix operations and semi-ring operators capture majority of parallelizeable graph algorithms
- **Concise:** "Think like a matrix." Only 3 lines need to be changed to convert your BFS code to SSSP
- **Performant:** "Optimize for the common case." Implementers write accelerator code for common sparse matrix operations and backend selects best one based on computational complexity
- **Backend-agnostic:** By using a linear algebra-based interface based on an open standard (e.g. **GraphBLAS**), user enjoys speed-up without having to write any accelerator code

Conclusion

- By its nature, linear algebra can be considered very **concise**
- GraphBLAS is an open standard, so there is consensus on a standard C API and it is possible to build **backend-agnostic** graph framework

Our work focuses on the other two metrics:

- We show that linear algebra is **expressible** enough to express a fairly complicated optimization: direction-optimized BFS
- We also demonstrate that our implementation is **performant** compared to traditional, vertex-centric graph frameworks

Contact Information

- Web: http://www.ece.ucdavis.edu/∼ctcyang/
- Email: ctcyang@ucdavis.edu
- Phone: +1 (916) 802-8178

[1] Carl Yang, Yangzihao Wang, and John D. Owens. Fast sparse matrix and sparse vector multiplication algorithm on the GPU. In *Graph Algorithms Building Blocks*, GABB 2015, pages 841–847, May 2015.

[2] Carl Yang, Aydın Buluç, and John D. Owens. Design principles for sparse matrix multiplication on the GPU. In *Proceedings of the 24th International European Conference on Parallel and Distributed Computing*, Euro-Par 2018, August 2018.

[3] Carl Yang, Aydın Buluç, and John D. Owens. Implementing push-pull efficiently in GraphBLAS. In *Proceedings of the International Conference on Parallel Processing*, ICPP 2018, August 2018.

Berkeley UNIVERSITY OF CALIFORNIA
BERKELEY LAB Bringing Science Solutions to the World
UCDAVIS UNIVERSITY OF CALIFORNIA