Implementing Push-Pull Efficiently in GraphBLAS
International Conference on Parallel Processing

Carl Yang\textsuperscript{1,3}, Aydın Buluç\textsuperscript{2,3} and John D. Owens\textsuperscript{1}

\textsuperscript{1} University of California, Davis
\textsuperscript{2} University of California, Berkeley
\textsuperscript{3} Lawrence Berkeley National Laboratory

August 16, 2018
Overview

1. Problem

2. Breadth First Search Using Linear Algebra

3. Direction-Optimized Breadth First Search

4. Evaluation

5. Conclusion
Overview

1. Problem

2. Breadth First Search Using Linear Algebra

3. Direction-Optimized Breadth First Search

4. Evaluation

5. Conclusion
What’s in common?

(a) Social network analysis

(b) Bioinformatics

(c) Computer forensics

(d) Recommender systems
What’s in common?
Problem: Why high-performance?

---

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

$G = (V,E)$

$A^T x \rightarrow A^T x$

\(^1\) 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
Semiring notation: (Add, Multiply, Domain)

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Semiring</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real field</td>
<td>{+ , \times , \mathbb{R}}</td>
<td>Classical numerical linear algebra</td>
</tr>
<tr>
<td>Boolean</td>
<td>{</td>
<td>, &amp;, {0, 1}}</td>
</tr>
<tr>
<td>Tropical</td>
<td>{\text{min}, +, \mathbb{R} \cup {\infty}}</td>
<td>Shortest path</td>
</tr>
<tr>
<td>Max-plus</td>
<td>{\text{max}, +, \mathbb{R}}</td>
<td>Graph matching</td>
</tr>
<tr>
<td>Min-times</td>
<td>{\text{min}, \times, \mathbb{R}}</td>
<td>Maximal independent set</td>
</tr>
</tbody>
</table>
Operations + Operators = GraphBLAS

**Operations**:
- Sparse Matrix-Sparse Vector (SpMSpV)
- Sparse Matrix-Dense Vector (SpMV)
- Sparse-Sparse Matrix Product (SpGEMM)
- Sparse-Dense Matrix Product (SpMM)

**Operators**:
- Miscellaneous: connectivity, traversal (BFS), independent sets (MIS), graph matching
- Centrality: PageRank, betweenness, closeness
- Graph clustering: Markov cluster, peer pressure, spectral, local
- Shortest paths: all-pairs, single-source, temporal

GraphBLAS primitives in increasing arithmetic intensity
Problem: Existing implementations not performant enough

Overview

1. Problem

2. Breadth First Search Using Linear Algebra

3. Direction-Optimized Breadth First Search

4. Evaluation

5. Conclusion
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

from

to

AT

1 2 3 4 5 6 7
Iteration 1

Breadth First Search Using Linear Algebra

CY, AB, JDO (UCD, UCB, LBNL)

Implementing Push-Pull in GraphBLAS

August 16, 2018 18 / 48
Iteration 2

Breadth First Search Using Linear Algebra

Implementing Push-Pull in GraphBLAS

August 16, 2018 19 / 48
Iteration 3

parents:

from

to

$AT^T X$
Final Iteration

Breadth First Search Using Linear Algebra

Implementing Push-Pull in GraphBLAS
Problem: Many active vertices $\Rightarrow$ Slow
Overview

1 Problem

2 Breadth First Search Using Linear Algebra

3 Direction-Optimized Breadth First Search

4 Evaluation

5 Conclusion
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
Problem: How to represent this in linear algebra?
Using Push (standard BFS, 5 memory accesses)
Using Pull (13 memory accesses)

Row 2 means Vertex 2's parents

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

Push

adjacency matrix

\[
\begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\]

input vector

output vector

\[
\begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\]

\[
X = \begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\end{bmatrix}
\]
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 Ligra⁴.

Optimization 1: Change of direction

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

Complexity improves from $O(dM)$ to $O(dn n z(m))$, where $m$ is the mask vector. This yielded a 154% speedup on power law graphs. Where is the rest of the performance?
Using Masking, from 13 down to 9 memory accesses

Row 2 means Vertex 2's parents

from

to

parents:
Optimization 2: Masking

Complexity improves from $O(dM)$ to $O(d \: \text{nnz}(m))$, where $m$ 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

This yielded 302% speedup on power law graphs.
Using Early-Exit, from 9 down to 5 memory accesses

Row 2 means Vertex 2's parents

from

to

parents:
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

1. Problem
2. Breadth First Search Using Linear Algebra
3. Direction-Optimized Breadth First Search
4. Evaluation
5. Conclusion
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): $O(dM)$

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

Column-based (mask and no mask): $O(d \ \text{nnz}(f) \log M)$
Runtime explained by crossing of frontier and unvisited node counts

![Graph representing runtime explained by crossing of frontier and unvisited node counts.](image)
Performance close to Gunrock on scale-free graphs
Where the speedup comes from

![Evaluation Diagram]

- Push-only baseline: 1.0x
- Structure only: 1.62x
- Change of direction: 1.08x
- Masking: 2.58x
- Early exit: 4.02x
- Operand reuse: 2.68x
Overview

1. Problem
2. Breadth First Search Using Linear Algebra
3. Direction-Optimized Breadth First Search
4. Evaluation
5. Conclusion
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