A GraphBLAS solution to the SIGMOD 2014 Programming Contest using multi-source BFS

M. Elekes, A. Nagy, D. Sándor, J.B. Antal, T.A. Davis, G. Szárnyas
MOTIVATION

- Graph algorithms are challenging to program
  - irregular access patterns → poor locality
  - caching and parallelization are difficult

- Optimizations often limit portability

- **GraphBLAS** introduces an abstraction layer using the language of linear algebra
  - graph ≡ sparse matrix
  - navigation step ≡ matvec on semirings
MOTIVATION

- Graph algorithms are challenging to program
  - irregular access patterns → poor locality
  - caching and parallelization are difficult

- Optimizations often limit portability

- **GraphBLAS** introduces an abstraction layer using the language of linear algebra
  - graph ≡ sparse matrix
  - navigation step ≡ matvec on semirings
GRAPHBLAS STACK

- Graph analytical applications
  - Algorithm library
    - GraphBLAS C API
    - GraphBLAS implementation
    - Hardware architecture

Application developers and algorithm designers
Sparse matrix experts and hardware designers

Textbook algos: BFS, PageRank, triangle count
Untyped graphs

GraphBLAS-based Cypher engine: RedisGraph
Property graphs

RQ: How to formulate mixed workloads on property graphs?
SIGMOD 2014 PROGRAMMING CONTEST

Annual contest

- Teams compete on database-related programming tasks
- Highly-optimized C++ implementations

2014 event

- Tasks on the LDBC social network graph
  - Benchmark data set for property graphs
  - People, forums, comments, hashtags, etc.
- 4 queries
  - Mix of filtering operations and graph algorithms
I. Compute an induced subgraph over Person-knows-Person

II. Run the graph algorithm on the subgraph

I. Create induced subgraph from (pA)-[:knows]-{pB}.

II. In the subgraph, compute the closeness centrality value for each Person p, then return top-k Persons with the highest values.

exact closeness centrality

key kernel: all-source BFS
OVERVIEW OF QUERIES 1, 2, 3

I. Filter the induced subgraph(s)

Create induced subgraph from (pA)\{-[:knows]-:pB\}
where \(\text{count}(p) > x\) and \(\text{count}(c) > x\).

II. Run the graph algorithm

Run the graph algorithm

unweighted shortest path

connected components

pairwise reachability
GRAPHBLAS SOLUTION OF THE QUERIES

- **Loading** includes relabelling UINT64 vertex IDs to a contiguous sequence $0 \ldots N - 1$.

- **Filtering the induced subgraph** from the property graph is mostly straightforward and composable with the algorithms.

- **The algorithms** can be concisely expressed in GraphBLAS:
  - Connected components $\rightarrow$ FastSV [Zhang et al., PPSC’20]
  - BFS
  - Bidirectional BFS
  - All-source BFS + bitwise optimization
  - Multi-source bidirectional BFS
BFS
**BFS: BREADTH-FIRST SEARCH**

Boolean matrices and vectors

- **seen**:矩阵和向量
- **frontier**: 决策边界
- \( \neg \text{seen} \) mask

- \( \oplus \): logical OR
- \( \otimes \): logical AND

\[
\text{next}(\neg \text{seen}) = A \lor \text{LAND} \ \text{frontier} \\
\text{seen}' = \text{seen} \lor \text{next}
\]
BFS: BREADTH-FIRST SEARCH

mask prevents redundant computations

next\langle \neg \text{seen} \rangle = \text{A LOR}, \text{LAND}, \text{frontier}

\text{seen'} = \text{seen LOR next}
All-source BFS
Q4 computes the top-$k$ Person vertices based on their exact closeness centrality values:

$$ CCV(p) = \frac{(C(p) - 1)^2}{(n - 1) \cdot s(p)} $$

where

- $C(p)$ is the size of the connected component of vertex $p$,
- $n$ is the number of vertices in the induced graph,
- $s(p)$ is the sum of geodesic distances to all other reachable persons from $p$.

$s(p)$ is challenging: needs unweighted all-pairs shortest paths.
BOOOLEAN ALL-SOURCE BFS ALGORITHM

Frontier t1 t2 t3 t4 t5

seen′ t1 t2 t3 t4 t5

next(¬seen) = A LOR . LAND Frontier

seen′ = seen LOR next
BOOLEAN ALL-SOURCE BFS ALGORITHM

Frontier $t_1 \ t_2 \ t_3 \ t_4 \ t_5$

Seen $t_1 \ t_2 \ t_3 \ t_4 \ t_5$

A $\begin{array}{c|c|c|c|c|c}
1 & 2 & 3 & 4 & 5 \\
\hline
1 & & & & \\
2 & & & & \\
3 & & & & \\
4 & & & & \\
5 & & & & \\
\end{array}$

Next($\neg$Seen) = A LOR LAND Frontier

Seen' $t_1 \ t_2 \ t_3 \ t_4 \ t_5$

Seen' = Seen LOR Next
Bitwise all-source BFS
BITWISE ALL-SOURCE BFS ALGORITHM

- For large graphs, the all-source BFS algorithm might need to run 500k+ traversals

- Two top-ranking teams used bitwise operations to process traversals in batches of 64 [Then et al., VLDB’15]

- This idea can be adopted in the GraphBLAS algorithm by
  - using UINT64 values
  - performing the multiplication on the BOR \cdot SECOND semiring, where BOR is “bitwise or” and SECOND(x, y) = y

- 5-10x speedup compared to the Boolean all-source BFS
### BITWISE ALL-SOURCE BFS ALGORITHM

**Frontier**

<table>
<thead>
<tr>
<th></th>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>0100</td>
<td>0000</td>
</tr>
<tr>
<td>3</td>
<td>0010</td>
<td>0000</td>
</tr>
<tr>
<td>4</td>
<td>0001</td>
<td>0000</td>
</tr>
<tr>
<td>5</td>
<td>0000</td>
<td>1000</td>
</tr>
</tbody>
</table>

**Seen**

<table>
<thead>
<tr>
<th>A</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0100</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0010</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0001</td>
<td>0000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0000</td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Next** = A BOR . SECOND Frontier

**Seen’**

<table>
<thead>
<tr>
<th></th>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0101</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>1010</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>0101</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>1010</td>
<td>0000</td>
</tr>
<tr>
<td>5</td>
<td>0110</td>
<td>0000</td>
</tr>
</tbody>
</table>

Using UINT4s here.
BITWISE ALL-SOURCE BFS ALGORITHM

Frontier

<table>
<thead>
<tr>
<th></th>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0101</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>1010</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>0101</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>1010</td>
<td>0000</td>
</tr>
<tr>
<td>5</td>
<td>0110</td>
<td>0000</td>
</tr>
</tbody>
</table>

Seen

<table>
<thead>
<tr>
<th></th>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0101</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>1010</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>0101</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>1010</td>
<td>0000</td>
</tr>
<tr>
<td>5</td>
<td>0110</td>
<td>0000</td>
</tr>
</tbody>
</table>

A

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>•</td>
<td>•</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>4</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
</tbody>
</table>

Next = A BOR . SECOND Frontier

Seen'

<table>
<thead>
<tr>
<th></th>
<th>t1-t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1111</td>
<td>1000</td>
</tr>
<tr>
<td>2</td>
<td>1111</td>
<td>1000</td>
</tr>
<tr>
<td>3</td>
<td>1111</td>
<td>1000</td>
</tr>
<tr>
<td>4</td>
<td>1111</td>
<td>1000</td>
</tr>
<tr>
<td>5</td>
<td>1111</td>
<td>1000</td>
</tr>
</tbody>
</table>

Full VLDB paper on this algorithm vs. 9 GrB operations
Bidirectional BFS
BIDIRECTIONAL BFS

Advance frontiers alternately and intersect them

Length = 1 \times \text{next1} \cap \text{frontier2} = \text{next2}

Length = 2

\text{next1} \cap \text{next2} = \text{next1} \cap \text{next2}
Bidirectional MSBFS
BIDIRECTIONAL MSBFS ALGORITHM

- Pairwise reachability problem:
  From a given set of $k$ vertices, which pairs of vertices are reachable from each other with at most $h$ hops?

- Naïve solution:
  Run a $k$-source MSBFS for $h$ steps and check reachability. The frontiers get large as they grow exponentially.

- Better solution:
  Advance all frontiers simultaneously for $\lfloor h/2 \rfloor$ iterations.
**BIDIRECTIONAL MSBFS**

**Seen[1]:** reachability with ≤1 hops
# BIDIRECTIONAL MSBFS

**Seen[2]:** reachability with $\leq 2$ hops

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>S[1]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Seen[2]**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
To get paths of at most 4 hops, we compute $\text{Seen}[2]^T$. Here, we found paths between all pairs:
- from 1 to 5,
- from 1 to 6,
- from 5 to 6.

From vertex 1, we could get to these vertices with $\leq 2$ hops.

From vertex 5, we could get to these vertices with $\leq 2$ hops.
To get exactly 3-length paths we compute $\text{Next}[2]^T$.

We found two 3-length paths:
- from 5 to 1
- from 5 to 6.

From vertex 1, we could get to this vertex with 2 hops.

From vertex 5, we could get to these vertices with 1 hop.

From vertex 1, we could get to this vertex with 2 hops.
Benchmark results
BENCHMARK RESULTS

- The top solution of AWFY vs. SuiteSparse:GraphBLAS v3.3.3
- AWFY’s solution uses SIMD instructions → difficult to port
- GraphBLAS load times are slow (see details in paper)

80 executions with different parameters
This is optimization is subject to future work.

For low diameter graphs, it is worth to use push/pull phases.

Push/pull is simple to express in GraphBLAS.

- See [Yang et al., ICPP’18]

But deciding *when* to change is non-trivial.
SUMMARY

- An interesting case study, see technical report

- GraphBLAS can capture mixed workloads
  - Induced subgraph computations are simple to express
  - Algorithms are concise, bitwise optimizations can be adopted
  - Performance is sometimes on par with specialized solutions

- Future optimizations
  - Bitmap-based matrix/vector compression is WIP by Tim Davis
    \[ \rightarrow \approx 5 \times \text{speedup} \] without using bitwise operators in our code

GraphBLAS

Extended slide deck

sigmod2014-contest-graphblas
Ω