High-Performance Linear Algebra-based Graph Framework on GPU
PhD Exit Seminar

Carl Yang

University of California, Davis

May 1, 2019
Overview

1. Introduction
2. Goals
3. Challenges
4. Conclusion
Overview

1. Introduction
2. Goals
3. Challenges
4. Conclusion
Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Mellon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

Abstract-- It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitive building blocks. This paper is a position paper defining the problem and announcing our intention to launch an open effort to define this standard.

http://graphblas.org
Graph traversal is sparse matrix multiplication

\[ G = (V,E) \]

\[ A^T \times x \xrightarrow{} A^T x \]

---

\(^1\) Denes Konig, 1931
Ingredient 1: Operations

“Our mission is to build up a linear algebra sense to the extent that vector-level thinking becomes as natural as scalar-level thinking.”
- Charles Van Loan

\[(a)\ eWiseAdd \quad (b)\ eWiseMult \quad (c)\ mxv \quad (d)\ mxm\]
Ingredient 2: Operators

Semiring notation: (Add, Multiply, Domain)

<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>,</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>
Semiring notation: (Add, Multiply, Domain)

<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

- **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

---

1. [http://graphblas.org](http://graphblas.org)
Overview

1. Introduction
2. Goals
3. Challenges
4. Conclusion
Goals of GraphBLAS

1. Provide portable performance
2. Allow concise expression
3. Match state-of-the-art in performance
4. Be effective at small scale and exascale
Goal 1: Provide portable performance

Open standard with well-defined API specification ensures portability.

Q: But how to future-proof spec against future innovations in data structures and algorithms?
Opaque Vector and Matrix

Open standard with well-defined API specification ensures portability.

Q: But how to future-proof spec against future innovations in data structures and algorithms?

```
Info mxv(Vector w,
       Vector mask,
       BinaryOp accum,
       Semiring op,
       Matrix A,
       Vector u,
       Descriptor desc);
```
## Goal 2: Allow concise expression

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Ligra</th>
<th>GraphIt</th>
<th>Gunrock</th>
<th>GraphBLAST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Breadth-first-search</td>
<td>30</td>
<td>22</td>
<td>2732</td>
<td>25</td>
</tr>
<tr>
<td>Graph coloring</td>
<td>N/A</td>
<td>N/A</td>
<td>893</td>
<td>22</td>
</tr>
<tr>
<td>Local graph clustering</td>
<td>84</td>
<td>N/A</td>
<td>875</td>
<td>45</td>
</tr>
<tr>
<td>Maximal independent set</td>
<td>51</td>
<td>N/A</td>
<td>N/A</td>
<td>26</td>
</tr>
<tr>
<td>Single-source shortest-path</td>
<td>55</td>
<td>25</td>
<td>760</td>
<td>25</td>
</tr>
</tbody>
</table>

**Table:** Lines of C++ application code counted by ‘cloc’.

---

2. Yunming Zhang et al. “GraphIt: A High-performance Graph DSL.” *OOPSLA ’18*
Goal 3: Match state-of-the-art in performance

1 Zhang, Peter, et al. “GBTL-CUDA: Graph algorithms and primitives for GPUs.” IPDPSW ’16.

Sparse-matrix sparse-vector multiplication necessary

GraphBLAS primitives in increasing arithmetic intensity

eWiseAdd | eWiseMult  
mxv  
mxm

SpMV | SpMSpV

Yang, Wang, Owens [IPDPSW '15]

---

Two load-balancing algorithms for SpMM

GraphBLAS primitives in increasing arithmetic intensity

- eWiseAdd
- eWiseMult
- mxv
- mxm

- SpMV
- SpMSpV
- SpGEMM
- SpMM

Yang, Wang, Owens [IPDPSW '15]
Yang, Bulu¸c, Owens [EuroPar '18]

---

Push-pull maps nicely to linear algebra

GraphBLAS primitives in increasing arithmetic intensity

- eWiseAdd
- eWiseMult
- mxv
- mxm
- SpMV
- SpMSpV
- SpGEMM
- SpMM
- Masking
- Early exit
- Op reuse

Yang, Wang, Owens [IPDPSW '15]
Yang, Buluç, Owens [EuroPar '18]
Yang, Buluç, Owens [ICPP '18]

---

3Carl Yang, Aydin Buluç and John D. Owens. “Implementing push-pull efficiently in GraphBLAS” ICPP '18.
Comparable performance to state-of-the-art
GraphBLAST has closed the $10 \times$ performance gap
Goal 4: Be effective at small scale and exascale

![Graph showing performance of different systems across various scales.]

- Top CPU Graph500 entry: 82944 nodes, K Computer
- Top GPU Graph500 entry: 1366 nodes x 3 GPUs/node, TSUBAME 2.0
- Pan, Pearce and Owens: 31 nodes x 4 GPUs/node, CORAL early access

The graph illustrates performance metrics (GTEPS) against problem scale.
Overview

1. Introduction

2. Goals

3. Challenges

4. Conclusion
Challenges of GraphBLAS

1. Hard for users to learn
2. Bulk-synchronous
3. GraphBLAS C API limited to C
4. Dynamic graphs
5. Kernel fusion
Challenge 1: Hard for users to learn

1. Where is the graph?
2. Where are the edges?
3. Where are the vertices?
Challenge 2: Bulk-synchronous

Some say the world is moving towards asynchronous backends: AM++, HavoqGT, Galois, STAPL, HPX, Charm++

Q: How to reconcile GraphBLAS’s inherently bulk-synchronous model with these asynchronous runtime systems?
Example: HavoqGT’s delegates partitioning

\[ \text{(a) 1D Partitioning} \quad \text{(b) Distributed Delegates Partitioning.} \]

\[ P_1 \text{ contains the controller.} \]

---

\(^1\) Roger Pearce, Maya Gokhale and Nancy M. Amato. “Faster Parallel Traversal of Scale Free Graphs at Extreme Scale with Vertex Delegates.” *SC ’14*

\(^2\) Yuechao Pan, Roger Pearce and John D. Owens. “Scalable Breadth-First Search on a GPU Cluster.” *IPDPS ’18*
Delegates partitioning in linear algebra

\[ E_{\text{high}} \quad E_{\text{low}} \]

```
P0...P3
P0
P1
P2
P3
```

```
P0...P3
P0
P1
P2
P3
```
Delegates partitioning in linear algebra
Standard 1D partition communication for $E_{low}$

$E_{low}$

P0  P1  P2  P3

$\times$

P0  P1  P2  P3

Send to P0...P3 tree

Send to P0

Send to P1

Send to P2

Send to P3
Reduced communication for $E_{\text{high}}$

$E_{\text{high}}$

- P0
- P1
- P2
- P3

Root  P0  P1  P2  P3

MPI_Broadcast to P0...P3

No communication needed

No communication needed

No communication needed

No communication needed
Open questions

Q: How to build a distributed framework that is decoupled from GraphBLAST, so that we can get code reuse?

Q: Can we build a distributed framework that is decoupled from the implementation, so that every GraphBLAS backend (e.g. single-threaded CPU, multi-threaded CPU, GPU) gets distributed computing for free?

- This is what Horovod does for deep learning, but it needs to make assumptions on data structures (i.e. dense tensor) and supports only 3 MPI communication ops.
- Can we do the same by limiting to CSR and a minimal set of ops?
Challenge 3: GraphBLAS C API limited to C
Use modern practices such as hourglass model

1Stephanus Dutoit. “Hourglass Interfaces for C++ APIs.” CppCon ’14
Use modern practices such as hourglass model

Enables goodies like:
- Operator overloading in Python and C++ frontends
  - Use + and × with previously used Semiring and Descriptor
- Templates can be used in C++ backend for codegen
- RAII for memory safety
- Can build C++ spec atop existing C spec

---

1Stephanus Dutoit. “Hourglass Interfaces for C++ APIs.” *CppCon ’14*
Challenge 4: Dynamic graphs

Q: What are the motivating applications for dynamic graphs?

prototype $\rightarrow$ full implementation $\rightarrow$ intuit patterns $\rightarrow$ extract frameworks

Q: How should system be designed so as to maximize decoupling between code and code specific to each graph representation?

- CSR, SlabHash, cuSTINGER, etc.

---

1Max Dama. “Max Dama on Automated Trading.” 2008
Challenge 5: Kernel fusion

1: \textbf{procedure} \texttt{GRAPHBLASCOLOR}(A, C)
2: \hspace{1em} \triangleright \textbf{Initialize colors to 0}
3: \texttt{GrB\_assign}(C, \texttt{GrB\_NULL}, 0, \texttt{GrB\_ALL}, \texttt{nrows}(A), \texttt{desc});
4: \hspace{1em} \triangleright \textbf{Assign random weight to each vertex}
5: \texttt{GrB\_apply}(weight, \texttt{GrB\_NULL}, \texttt{GrB\_NULL},
               \hspace{1em} \texttt{set\_random}(), weight, desc);
6: \hspace{1em} \textbf{for each color} = 1, \ldots, n \textbf{ do}
7: \hspace{1.5em} \triangleright \textbf{Find max of neighbors}
8: \hspace{1.5em} \texttt{GrB\_vxm}(max, \texttt{GrB\_NULL}, \texttt{GrB\_NULL},
               \hspace{1.5em} \texttt{GrB\_INT32MaxTimes}, weight, A, desc);

...
Thoughts on implementing kernel fusion

1. Treat each operation as node in computation graph
2. Traverse computation graph looking for nodes that should be fused together
   - Elementwise ops and simple math ops such as apply
3. Generate CUDA code
4. Use realtime compilation to generate compiled CUDA kernel
Overview

1. Introduction
2. Goals
3. Challenges
4. Conclusion
Our work is the first GraphBLAS implementation on the GPU that shows comparable performance to state-of-the-art
- 10× to 1000× speed-up compared to other GraphBLAS implementations

Linear algebra-based graph frameworks are the way to go:
- Just as expressible and performant as their vertex-centric counterparts
- More concise and portable
- Parallel traversals from different source vertices requires minimal code change (change Vector to Matrix)

Challenge for current students: Scale up to exascale problems
Roadmap assuming linear weak scaling
Acknowledgments

- My advisors, John D. Owens and Aydın Buluç
- Owens group and the Gunrock team
- GraphBLAS C API subcommittee
- 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?

Open-source code with Apache License:
https://github.com/gunrock/graphblast
Code example: BFS

```cpp
#include <graphblas/graphblas.hpp>
using namespace graphblas;

void bfs(Vector<float>* v,
           const Matrix<float>* A,
           Index s,
           Descriptor* desc) {
    Index A_nrows;
    A->nrows(&A_nrows);
    float d = 1.f;

    Vector<float> f1(A_nrows);
    Vector<float> f2(A_nrows);
    std::vector<Index> indices(1, s);
    std::vector<float> values(1, 1.f);
    f1.build(&indices, &values, 1, GrB_NULL);

    v->fill(0.f);

    float c = 1.f;
    while (c > 0) {
        assign(v, &f1, GrB_NULL, d, GrB_ALL, A_nrows, desc);
        desc->toggle(GrB_MASK);
        vxm(&f2, v, GrB_NULL, LogicalOrAndSemiring<float>(), &f1, A, desc);
        desc->toggle(GrB_MASK);
        f2.swap(&f1);
        reduce(&c, GrB_NULL, PlusMonoid<float>(), &f1, desc);
        d++;
    }
}
```