GraphBLAS: Graph Algorithms in the Language of Linear Algebra

John R. Gilbert
University of California, Santa Barbara

June 27, 2019

Support: Intel, Microsoft, DOE Office of Science, NSF
Multiple-source breadth-first search

- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges
Coarsening via sparse matrix-matrix products

Counting triangles (clustering coefficient)

Cluster coefficient:
- \( \Pr (\text{wedge } i-j-k \text{ makes a triangle with edge } i-k) \)
- \( 3 \times \frac{\# \text{ triangles}}{\# \text{ wedges}} \)
- \( 3 \times \frac{4}{19} = 0.63 \) in example
- may want to compute for each vertex \( j \)

Low-hinge algorithm to count triangles:
- Count triangles by lowest-degree vertex.
- Enumerate “low-hinged” wedges.
- Keep wedges that close.

Counting triangles (clustering coefficient)

\[
A = L + U \quad \text{(hi->lo + lo->hi)}
\]

\[
L \times U = B \quad \text{(wedge, low hinge)}
\]

\[
A \wedge B = C \quad \text{(closed wedge)}
\]

\[
\text{sum}(C)/2 = 4 \text{ triangles}
\]
Counting triangles (clustering coefficient)

A = L + U \quad \text{(hi->lo + lo->hi)}
L \times U = B \quad \text{(wedge, low hinge)}
A \wedge B = C \quad \text{(closed wedge)}
\text{sum}(C)/2 = 4 \text{ triangles}

**Spoiler:** \((L \times L) \wedge L\) works better in practice \([Wolf \ et \ al. \ 2017]\)
Graph algorithms in the language of linear algebra

- Kepner et al. study [2006]: fundamental graph algorithms including min spanning tree, shortest paths, independent set, max flow, clustering, …
- SSCA#2 / centrality [2008]
- Basic breadth-first search / Graph500 [2010]
- Combinatorial BLAS [2010]
But why do it this way in practice?

Picking the right level of abstraction:
High enough to optimize,
Low enough to be broadly useful

<table>
<thead>
<tr>
<th>Vertex/edge graph computations</th>
<th>Graphs in the language of linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpredictable, data-driven communication patterns</td>
<td>Fixed communication patterns</td>
</tr>
<tr>
<td>Irregular data accesses, with poor locality</td>
<td>Matrix block operations exploit memory hierarchy</td>
</tr>
<tr>
<td>Fine grained data accesses, dominated by latency</td>
<td>Coarse grained parallelism, limited by bandwidth not latency</td>
</tr>
</tbody>
</table>
The (original) BLAS: Separation of concerns

The Basic Linear Algebra Subroutines had a revolutionary impact on computational linear algebra.

<table>
<thead>
<tr>
<th>BLAS 1</th>
<th>vector ops</th>
<th>Lawson, Hanson, Kincaid, Krogh, 1979</th>
<th>LINPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS 2</td>
<td>matrix-vector ops</td>
<td>Dongarra, Du Croz, Hammarling, Hanson, 1988</td>
<td>LINPACK on vector machines</td>
</tr>
<tr>
<td>BLAS 3</td>
<td>matrix-matrix ops</td>
<td>Dongarra, Du Croz, Duff, Hammarling, 1990</td>
<td>LAPACK on cache based machines</td>
</tr>
</tbody>
</table>

- Experts in mapping algorithms to hardware tuned BLAS for specific platforms.
- Experts in numerical linear algebra built software on top of the BLAS to get high performance “for free.”

Today every computer, phone, etc. comes with /usr/lib/libblas
Combinatorial BLAS  (2010)
[Azad, Buluc, JG, Lugowski, …]

An extensible distributed-memory library offering a small but powerful set of linear algebraic operations specifically targeting graph analytics.

- Aimed at graph algorithm designers/programmers who are not expert in mapping algorithms to parallel hardware.
- Flexible templated C++ interface.
- Scalable performance from laptop to 100,000-processor HPC.
- Open source software: people.eecs.berkeley.edu/~aydin/CombBLAS/html/
- Version 1.6.2 released April 2018.

Launching the GraphBLAS effort

- Manifesto, HPEC 2013:

  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)

  “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.”

- GraphBLAS Forum:  www.graphblas.org
- Workshops at HPEC, IPDPS, SC
- Working group telecons and meetings
The GraphBLAS are born

• First the math, HPEC 2016:

Mathematical Foundations of the GraphBLAS

Jeremy Kepner (MIT Lincoln Laboratory Supercomputing Center), Peter Aaltonen (Indiana University), David Bader (Georgia Institute of Technology), Aydın Buluç (Lawrence Berkeley National Laboratory), Franz Franchetti (Carnegie Mellon University), John Gilbert (University of California, Santa Barbara), Dylan Hutchison (University of Washington), Manoj Kumar (IBM), Andrew Lumsdaine (Indiana University), Henning Meyerhenke (Karlsruhe Institute of Technology), Scott McMillan (CMU Software Engineering Institute), Jose Moreira (IBM), John D. Owens (University of California, Davis), Carl Yang (University of California, Davis), Marcin Zalewski (Indiana University), Timothy Mattson (Intel)

• Then the language bindings, GABB 2017:

Design of the GraphBLAS API for C

Aydın Buluç†, Tim Mattson‡, Scott McMillan§, José Moreira¶, Carl Yang*,†

†Computational Research Division, Lawrence Berkeley National Laboratory
‡Intel Corporation
§Software Engineering Institute, Carnegie Mellon University
¶IBM Corporation
*Electrical and Computer Engineering Department, University of California, Davis, USA
GraphBLAS: Building blocks for graphs as linear algebra

<table>
<thead>
<tr>
<th>Operation name</th>
<th>Mathematical description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxm</td>
<td>[ C \odot = A \oplus . \otimes B ]</td>
</tr>
<tr>
<td>mxv</td>
<td>[ w \odot = A \oplus . \otimes v ]</td>
</tr>
<tr>
<td>vxm</td>
<td>[ w^T \odot = v^T \oplus . \otimes A ]</td>
</tr>
<tr>
<td>eWiseMult</td>
<td>[ C \odot = A \times B ]</td>
</tr>
<tr>
<td>eWiseAdd</td>
<td>[ w \odot = u \otimes v ]</td>
</tr>
<tr>
<td>reduce (row)</td>
<td>[ w \odot = \bigoplus_j A(:,j) ]</td>
</tr>
<tr>
<td>apply</td>
<td>[ C \odot = F_u(A) ]</td>
</tr>
<tr>
<td>transpose</td>
<td>[ C \odot = A^T ]</td>
</tr>
<tr>
<td>extract</td>
<td>[ C \odot = A(i,j) ]</td>
</tr>
<tr>
<td>assign</td>
<td>[ C(i,j) \odot = A ]</td>
</tr>
</tbody>
</table>

- Operators: \( \oplus, \otimes \): semiring “add” and “multiply”, \( \odot \): “accumulate”
- Objects: matrix, vector, monoid, semiring, ...
Examples of semirings in graph algorithms

<table>
<thead>
<tr>
<th>Real field: ((\mathbb{R}, +, \times))</th>
<th>Numerical linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean algebra: (({0, 1}, \text{or}, \text{and}))</td>
<td>Connectivity &amp; traversal</td>
</tr>
<tr>
<td>Tropical semiring: ((\mathbb{R} \cup {\infty}, \text{min}, +))</td>
<td>Shortest paths</td>
</tr>
<tr>
<td>((S, \text{select}, \text{select}))</td>
<td>Select subgraph, or contract nodes to form quotient graph</td>
</tr>
<tr>
<td>((\mathbb{R}, \text{max}, +))</td>
<td>Graph matching &amp; network alignment</td>
</tr>
<tr>
<td>((\mathbb{R}, \text{max}, \times))</td>
<td>Maximal independent set</td>
</tr>
<tr>
<td>“values”: edge/vertex attributes, “add”: vertex data aggregation, “multiply”: edge data processing</td>
<td>General schema for user-specified computation at vertices and edges</td>
</tr>
</tbody>
</table>
GraphBLAS C API

• #include <GraphBLAS.h>

• Example: \[ C(-M) \oplus= A^T \oplus.\otimes B^T \]

```c
GrB_info GrB_mxm(GrB_Matrix *C,
                    const GrB_Matrix mask,
                    const GrB_BinaryOp accum,
                    const GrB_Semiring op,
                    const GrB_Matrix A,
                    const GrB_Matrix B
                   [, const GrB_Descriptor desc]);
```

• Opaque objects, e.g.:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrB_Type</td>
<td>Scalar type for semiring</td>
</tr>
<tr>
<td>GrB_BinaryOp</td>
<td>Binary operation</td>
</tr>
<tr>
<td>GrB_Semiring</td>
<td>Packages components of a semiring</td>
</tr>
<tr>
<td>GrB_Vector</td>
<td>1D (implicitly sparse) array</td>
</tr>
<tr>
<td>GrB_Matrix</td>
<td>2D sparse array, with row indices, column indices, and values</td>
</tr>
</tbody>
</table>

Notes on the C API

• Objects are opaque
  • Only GraphBLAS methods can manipulate them

• Data (matrices and vectors) are separate from operations
  • Only explicitly defined elements of a sparse matrix or vector have values
  • The “structural zeros” are undefined; semantics are defined so that the “implicit zero” value does not matter (most of the time)

• Blocking and non-blocking modes
  • Blocking: methods must complete before returning
  • Non-blocking: methods may return early
  • Facilitates operation fusion and efficient add/delete ops

• Most operations allow a “mask” parameter
  • Specifies that only a subset of output values are required
  • Avoids computation and materialization of intermediate objects
  • Turns out to be surprisingly useful
Breadth-first search in the language of matrices

levels:

from

1

1

0

to

7

7

7

A^T
Particular semiring operations

Multiply: logical and
Add: logical or

levels:

from

\begin{align*}
\begin{bmatrix}
A^T \\
X \\
A^{TX}
\end{bmatrix}
\end{align*}

(old frontier) (new frontier)
Could compute parents as well as levels (not shown)

levels:

from

A

T

X

\(A^TX\) (old frontier) (new frontier)
Masking omits already-reached vertices from result.
levels:

to

from

$A^T X$ (old frontier) (new frontier)
BFS in GraphBLAS with masks

(a) Pseudocode

```
1 Input: graph, frontier, levels
2 depth ← 0
3 while nvals(frontier) > 0:
4   depth ← depth + 1
5   levels[frontier] ← depth
6   frontier ← levels, replace> ← graph^T ⊕.⊗ frontier
    where ⊕.⊗ = ⊕.⊗(LogicalSemiring)
```
BFS in GraphBLAS with masks: C code

```c
GrB_Info bfs  // BFS of a graph (using vector assign & reduce)
{
    GrB_Vector *v_output, // v[i] is the BFS level of node i in the graph
    const GrB_Matrix A,  // input graph, treated as if boolean in semiring
    GrB_Index s          // starting node of the BFS
}
{
    GrB_Index n;          // # of nodes in the graph
    GrB_Vector q = NULL;  // nodes visited at each level
    GrB_Vector v = NULL;  // result vector
    GrB_Descriptor desc = NULL; // Descriptor for vxm
    GrB_Matrix_nrows (&n, A);  // n = # of rows of A
    GrB_Vector_new (&v, GrB_INT32, n); // Vector<int32_t> v(n) = 0
    GrB_Vector_new (&q, GrB_BOOL, n); // Vector<bool> q(n) = false
    GrB_Vector_setElement (q, true, s); // q[s] = true, false elsewhere
    GrB_Descriptor_new (&desc);
    GrB_Descriptor_set (desc, GrB_MASK, GrB_SCMP); // invert the mask
    GrB_Descriptor_set (desc, GrB_OUTP, GrB_REPLACE); // clear q first

    bool successor = true;  // true when some successor found
    for (int32_t level = 1; successor && level <= n; level++)
    {
        // v<<q> = level, using vector assign with q as the mask
        GrB_assign (v, q, NULL, level, GrB_ALL, n, NULL); // q<<v> = q ||.&& A ; finds all the unvisited
        // successors from current q, using !v as the mask
        GrB_vxm (q, v, NULL, GrB_LOR_LAND_BOOL, q, A, desc); // successor = |(|(q)
        GrB_reduce (&successor, NULL, GrB_LOR_BOOL_MONOID, q, NULL);
    }
    *v_output = v;          // return result
    GrB_free (&q);
    GrB_free (&desc);
    return (GrB_SUCCESS);
}
```

BFS in GraphBLAS with masks: Python & C++

```python
def bfs(graph, frontier, levels):
    depth = 0
    while frontier.nvals > 0:
        depth += 1
        levels[frontier][] = depth
    with gb.LogicalSemiring, gb.Replace:
        frontier[~levels] = graph.T @ frontier
```

(b) PyGB

```cpp
template<class Mat, class Frontier, class Levels>
void bfs(Mat &graph, Frontier frontier, Levels &levels)
{
    GrB::IndexType depth = 0;
    while (frontier.nvals() > 0) {
        ++depth;
        GrB::assign(levels, frontier, GrB::NoAccumulate(),
                    depth, GrB::AllIndices(), false);
        GrB::mxv(frontier, GrB::complement(levels),
                 GrB::NoAccumulate(),
                 GrB::LogicalSemiring<GrB::IndexType>(),
                 GrB::transpose(graph), frontier, true);
    }
}
```

(c) GBTL C++

LAGraph: A community effort to collect graph algorithms built on top of the GraphBLAS. IPDPS 2019.
SuiteSparse:GraphBLAS

• From Tim Davis (Texas A&M)
• First conforming implementation of C API

Features:
• 960 semirings built in; also user-defined semirings
• Fast incremental updates using non-blocking mode and “zombies”
• Several sparse data structures & polyalgorithms under the hood
• Currently single-threaded
  • OpenMP near release, CUDA planned, MPI contemplated
• Performance on graph benchmarks (e.g. triangles, k-truss) comparable to highly-tuned custom C code
• Included in Debian and Ubuntu Linux distributions
• Used as computational engine in commercial RedisGraph product

Other GraphBLAS implementations

• IBM GraphBLAS (IBM)
  • Second fully conforming release of the GraphBLAS C API
  • Descendant of IBM Graph Programming Interface

• GraphBLAS Template Library (SEI/CMU, PNNL, Indiana U)
  • C++ implementation of GraphBLAS math spec
  • With a C interface conforming to the GraphBLAS C API

• GraphBLAST (UC Davis)
  • GraphBLAS for GPU’s, from the Gunrock graph library group

• PyGB (UW, PNNL, SEI/CMU)
  • Python DSL using the C++ GraphBLAS Template Library

Friends of GraphBLAS: Linear-algebra-based graph libraries

• Combinatorial BLAS (LBNL, UCSB): C++ with MPI and OpenMP
• Graphulo (MIT): Java/Accumulo
• D4M (MIT): Matlab/Octave
• Graph Programming Interface (IBM): C
• GraphPad (Intel): C++, OpenMP, MPI
Industrial impact: RedisGraph graph database

- From Redis Labs, enterprise / cloud database vendor.
- Emphasis on:
  - Query speed for large graphs
  - Mass insertion efficiency
  - Scaling Redis beyond single-node memory
- SuiteSparse:GraphBLAS under the hood.
- Benchmarks: Query time dramatically better than competing graph databases.

Machine Learning relies a lot on Linear Algebra too

Higher-level machine learning tasks

- Logistic Regression, Support Vector Machines
- Dimensionality Reduction (NMF, CX, PCA)
- Clustering (e.g., MCL, Spectral Clustering)
- Partial Correlation Estimation (CONCORD)
- Deep Learning (Neural Nets)

Graph/Sparse/Dense BLAS functions (increasing arithmetic intensity)
Sparse deep neural networks with GraphBLAS

• From MIT-LL & IBM:
  • DNN inference oscillates between two semirings: (+, ×) and (max, +)
  • Sparsity is at the frontier of DNN research

MIT/IEEE/Amazon Graph Challenge: Enable Large Sparse Deep Neural Networks

• Entries not limited to GraphBLAS solutions!

• Details at graphchallenge.org

New Challenge Goal:
Enable Large Sparse Deep Neural Networks

- Deep neural networks (DNNs) are at the heart of modern AI miracles
- Larger neural networks often perform better
  - Larger number of layers/features allow more non-linear boundaries
  - Problem: limited by expensive high-speed memory size
  - Solution: sparse (pruned) neural networks deliver comparable performance with less memory
Markov clustering algorithm

Popular and successful algorithm for discovering clusters in protein interaction and protein similarity networks

Initial network

Iteration 1

Iteration 2

Iteration 3

At each iteration:

Step 1 (Expansion): Square the matrix
Step 2 (Pruning): Remove small entries and dense columns
Step 3 (Inflation): Take powers entry-wise

Naïve implementation: sparse matrix-matrix product (SpGEMM), followed by column-wise top-k selection and column-wise pruning
HipMCL: High-performance Markov clustering

- MCL is both **computationally expensive** and **memory hungry**, limiting the sizes of networks that can be clustered.
- HipMCL overcomes these limitations via **sparse parallel algorithms**, with a combined expansion and pruning step.
- **Up to 1000X times faster** than original MCL with same accuracy.

LACC: Parallel Connected Components

Scientific Achievement
- LACC: Linear-Algebraic Connected Components
- New distributed-memory parallel connected component discovery algorithm using GraphBLAS primitives

Significance and Impact
- Finding connected components is a fundamental primitive for computing on graphs.
- More than 2x faster connected component identification across different scales.
- Orders of magnitude faster at large concurrences.

Research Details
- Based on Awerbuch-Shiloach PRAM algorithm
- Used with the Exascale Application HipMCL
- Proper sparsity exploitation is the key to high performance

A. Azad and A. Buluc. LACC: A Linear-Algebraic Algorithm for Finding Connected Components in Distributed Memory. IPDPS, 2019
Linear-algebra-based Kokkos graph infrastructure

- From Sandia Labs & Georgia Tech
- Sparse matrix-matrix multiplication (SpGEMM) kernel in Kokkos Kernels shared memory library
- Performance portable across architectures and parallel language infrastructures
- Benchmark: Counting triangles / clustering coefficient
- 2-time champion (2017, 2018) of IEEE HPEC Graph Challenge

Push-pull, or direction optimization in search

[C. Yang PhD thesis, UC Davis]

Top-down BFS is worst-case optimal, but **pessimistic** for low-diameter graphs because when the frontier is at its peak, almost all edge examinations fail to claim a child.

**Top-Down (Push)**

- for all \( v \) in frontier attempt to parent *all* neighbors(\( v \))

**Bottom-Up (Pull)**

- for all \( v \) in unvisited find *any* parent (neighbor(\( v \)) in frontier)

**Beamer’s direction-optimizing BFS**

- Switch from top-down/push to bottom-up/pull
- When the majority of the vertices have been reached.
- For 5+ years, thought to be impossible (efficiently) in the language of linear algebra
Efficient push-pull in GraphBLAS
[C. Yang PhD thesis, UC Davis]

• Push vs. pull corresponds to matvec multiplication by columns vs. rows.

• Row matvec is better for dense vector, column is better for sparse.

• Key insight: Masking changes the complexity!

• Three key optimizations:
  • Choose push or pull
  • Efficient use of masking
  • Early exit from \((\wedge, \vee)\) vector dot product

---

Impact: Up to 100 MTEPS on 1 Xeon 4-core CPU plus 1 Tesla GPU.

C. Yang, A. Buluc, J. Owens. Implementing push-pull efficiently in GraphBLAS. ICPP 2018
Matrix-based graph processor design at MIT-LL

[Song, Kepner, et al. 2010]

3-D Graph Processor

Lincoln Laboratory, Massachusetts Institute Technology, Lexington, MA 02420

Figure 1: Computational Throughput Differences between Conventional and Graph Processing.

Figure 2: Sparse Matrix Representation of Graph.

Figure 3: 3-D Graph Processor with Electromagnetic Coupling Communications between Processor Boards.
International impact of GraphBLAS

• KAUST, Saudi Arabia [Jamour et al., EuroSys 2019]:
  • Matrix algebra for RDF triple stores
  • Demonstrated scaling to 512B triples on 2048 nodes

• CMU-Qatar / Qatar U [Ahmad et al., VLDB 2018]:
  • Automatic translation of vertex-edge programs to matrix ops
  • Demonstrated high performance on clouds and clusters

• Budapest U [Szarnyas, FOSDEM 2019]:
  • Multiplex graph metrics with GraphBLAS and other matrix libs
  • Interesting connections to complexity theory for database joins

• Huawei:
  • GraphBLAS in the cloud and on the mobile phone
What’s next for the GraphBLAS?

LAGraph: A Community Effort to Collect Graph Algorithms Built on Top of the GraphBLAS

Tim Mattson†, Timothy A. Davis○, Manoj Kumar§, Aydın Buluç†, Scott McMillan§, José Moreira§, Carl Yang∗,†

https://github.com/GraphBLAS/LAGraph

[GrAPL 2019]

Motivation: Separation of Concerns
Algorithms in LAGraph today

<table>
<thead>
<tr>
<th>File Name</th>
<th>Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>LACC_GraphBLAS</td>
<td>Connected components</td>
</tr>
<tr>
<td>LAGraph_BF_basic</td>
<td>Bellman-Ford single source shortest paths</td>
</tr>
<tr>
<td>LAGraph_BF_full</td>
<td>Bellman-Ford single source shortest paths, with tree</td>
</tr>
<tr>
<td>LAGraph_allktruss</td>
<td>All k-trusses of a graph</td>
</tr>
<tr>
<td>LAGraph_bfs_pushpull</td>
<td>Direction optimizing breadth-first search</td>
</tr>
<tr>
<td>LAGraph_bfs_simple</td>
<td>Conventional breadth-first search</td>
</tr>
<tr>
<td>LAGraph_dnn.c</td>
<td>Sparse deep neural network</td>
</tr>
<tr>
<td>LAGraph_ktruss</td>
<td>k-truss of a graph</td>
</tr>
<tr>
<td>LAGraph_lcc</td>
<td>Local clustering coefficient</td>
</tr>
<tr>
<td>LAGraph_pagerank</td>
<td>PageRank</td>
</tr>
<tr>
<td>LAGraph_tricount</td>
<td>Count triangles in a graph</td>
</tr>
</tbody>
</table>

LAGraph is open source, and solicits:

- More algorithms and implementations
- Use cases and requests for algorithms!
Directions & challenges: LAGraph

- Additional practical use cases and uptake!
- Add algorithms with known linear algebraic formulations
  - Centrality, clustering, subgraph counting, ...
- Investigate important graph algorithms that haven’t yet been implemented in linear algebraic form
  - A* search, branch & bound, supervised learning, ...
- Integration with numerical linear algebra libraries
  - Spectral methods, Laplacian paradigm for graph algorithms, linear equation solvers, optimization
- Statistical perspective: distributions, stochastic graphs, ...
- Moving data in and out of opaque GraphBLAS objects
  - Data science workflows are often not opaque data: pandas frames, numpy arrays, CSR matrices
Directions & challenges: GraphBLAS

- Moving data in and out of GraphBLAS
  - Probably want import/export functions within GraphBLAS (SuiteSparse has prototypes of this)
  - Also want finer-grained operations, e.g. iterators over edges, adjacencies, etc.

- Exploit and extend nonblocking mode for method fusion, matrix triple product optimization, etc.

- Robust multithreading support
  - Current spec hides threads inside individual GraphBlas methods

- Robust support for distributed memory
  - MPI, PGAS, other models?

- Finalizing more API specs
  - Extensions to C API spec for iterators, distributed execution, ...
  - In progress: Language bindings for Python, C++, ...
Thanks …

Ariful Azad, David Bader, Jon Berry, Aydin Buluc, Tim Davis, Kevin Deweese, Joe Eaton, Jeremy Kepner, Manoj Kumar, Adam Lugowski, Andrew Lumsdaine, Tim Mattson, Scott McMillan, Henning Meyerhenke, Jose Moreira, Veronika Neeley, John Owens, Steve Reinhardt, Viral Shah, Bill Song, Michael Wolf, Carl Yang … and Intel, Microsoft, NSF, DOE Office of Science