Graph Processing using GraphBLAS

Scott McMillan, CMU Software Engineering Institute

with collaborators Benjamin Brock, Tim Mattson, Jose E. Moreira, Aydin Buluc, Tim Davis, Gabor Szarnyas, Roi Lipman, Jim Kitchen, Erik Welch, and many more…
This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM23-0646
Outline

• Background & Motivation
• Graphs can be represented as matrices
• Basic graph operations can be performed with linear algebra
• These operations can be composed to implement useful algorithms
The GraphBLAS Application Programming Interface (API)

**Goal:** separate the concerns of hardware/library & application designers.

1979: BLAS Basic Linear Algebra Subprograms (BLAS 2 ’88, BLAS 3 ’90)
The GraphBLAS API

**Goal:** separate the concerns of hardware/library & application designers.

1979: BLAS  
Basic Linear Algebra Subprograms (BLAS 2 ’88, BLAS 3 ’90)

2001: Sparse BLAS  
an extension to BLAS (little uptake)
The GraphBLAS API

Goal: separate the concerns of hardware/library & application designers.

1979: BLAS  
Basic Linear Algebra Subprograms (BLAS 2 ’88, BLAS 3 ’90)

2001: Sparse BLAS  
an extension to BLAS (little uptake)

2013: GraphBLAS  
an effort to define standard building blocks for graph algorithms in the language of linear algebra

Numerical applications
LINPACK/LAPACK
API: Separation of concerns
BLAS
Hardware architecture

Graph analytic apps
LAGraph
API: Separation of concerns
GraphBLAS
Hardware architecture
GraphBLAS C/C++ Timeline

Book — Papers — GraphBLAS API version — SuiteSparse:GraphBLAS releases


Graph Algorithms in the Language of Linear Algebra

Standards for graph algorithm primitives, HPEC

Seven good reasons, ICCS

Mathematical foundations, HPEC

C API, GABB@ IPDPS

LAGraph, GrAPL@ IPDPS

C++ API Roadmap, Distributed GraphBLAS GrAPL@ IPDPS

Intro. to GraphBLAS v2.0, GrAPL@ IPDPS

C++ Iterators, GrAPL@ IPDPS

C++ Concepts, GrAPL@ IPDPS

© 2023 Carnegie Mellon University

DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.
GraphBLAS Primitives

• Basic objects (opaque types)
  – Matrices (sparse or dense), vectors (sparse or dense), algebraic operators (semirings)
• Fundamental operations over these objects

...plus reduction, transpose, Kronecker product, filtering, transform, etc.
Graphs as Adjacency Matrices

$$A_{ij} = \begin{cases} 1 & (v_i, v_j) \in E \\ 0 & (v_i, v_j) \notin E \end{cases}$$
Graphs as Adjacency Matrices

\[
A_{ij} = \begin{cases} 
\bullet & (v_i, v_j) \in E \\
\emptyset & (v_i, v_j) \notin E 
\end{cases}
\]
Graphs as Adjacency Matrices

\[ A_{ij} = \begin{cases} \bullet & (v_i, v_j) \in E \\ \emptyset & (v_i, v_j) \notin E \end{cases} \]
Graph Operations as Matrix Operations

Matrix-vector multiply $\rightarrow$ find neighbors
- In-neighbors: use $A$
- Out-neighbors: use $A^T$
Graph Operations as Matrix Operations

Finding out-neighbors is used many graph algorithms.

• Matrix-vector multiply $\to$ find neighbors
  - In-neighbors: use $A$
  - Out-neighbors: use $A^T$
Graph Operations as Matrix Operations

Another way to look at matrix-vector multiply…

\[
\begin{pmatrix}
A^T \\
\end{pmatrix}
\begin{pmatrix}
f \\
\end{pmatrix}
\]
One more thing… write masks: \( \langle m \rangle \)

Often not interested in some nodes…
Often not interested in some nodes…

ANOTHER feature of GraphBLAS: All operations support a write mask.

\[ f'\langle m \rangle = A^T \oplus. \otimes f \]
Algorithm: Breadth-First Search (BFS)
Example: Breadth-First Search (levels)

\( f(src) = \bullet \)
Example: Breadth-First Search (levels)

\[ \text{level} = 0 \]
\[ \mathbf{v} += \text{level} \times \mathbf{f} \]
Example: Breadth-First Search (levels)

\[ \text{level} = 0 \]
\[ \text{v} += \text{level} \times \text{f} \quad \text{// Use v as a mask, } \langle \overline{v} \rangle. \]
Example: Breadth-First Search (levels)

\( level = 0 \)
\( v += level \times f \)
\( f'(\bar{v}) = A^T \oplus \otimes f \)  // Boolean semiring
Example: Breadth-First Search (levels)

level = 0

\[ \mathbf{v} += level \times f \]

\[ f'(\langle \overline{\mathbf{v}} \rangle) = A^\top \odot \otimes f \]

\[ f = f' \]
Example: Breadth-First Search (levels)

\( level = 1 \)
\( \mathbf{v} += level \times \mathbf{f} \)
\( \mathbf{f}'(\mathbf{v}) = \mathbf{A}^\top \Theta \mathbf{x} \mathbf{f} \)
\( \mathbf{f} = \mathbf{f}' \)
Example: Breadth-First Search (levels)

\[ \text{level} = 2 \]
\[ \mathbf{v} += \text{level} \times \mathbf{f} \]
\[ \mathbf{f'}(\mathbf{v}) = \mathbf{A}^\top \otimes \mathbf{f} \]
\[ \mathbf{f} = \mathbf{f'} \]
Example: Breadth-First Search (levels)

\[ \text{level} = 3 \]

\[ \mathbf{v} += \text{level} \times \mathbf{f} \]

\[ \mathbf{f}'(\langle \mathbf{v} \rangle) = \mathbf{A}^\top \bigoplus \mathbf{\otimes f} \]

\[ \mathbf{f} = \mathbf{f}' \]

if \( \mathbf{f}.\text{empty()} \) return \( \mathbf{v} \)
Example: Breadth-First Search (levels)

- **Input:** adjacency matrix $A$ (Boolean), source vertex $src$ (integer)
- **Output:** visited vertices vector, $v$ (integer)
- **Workspace:** frontier vector $f$ (Boolean)

1. $f(src) = true$
2. $level = 0$
3. while ! $f$.empty() 
4. $v += level \ast f$
5. $f(\bar{v}) = A^T \oplus \otimes f$ // using the Boolean semiring (OR.AND)
6. $++level$
Resources/Activities (some covered in the next talk?)

- C API Specification
  - https://github.com/GraphBLAS/graphblas-api-c

- C API Implementation: SuiteSparse:GraphBLAS
  - https://github.com/DrTimothyAldenDavis/GraphBLAS

- LAGraph Algorithms Repository
  - https://github.com/GraphBLAS/LAGraph

- Language Bindings: python, Julia, postgres, etc
  - https://github.com/python-graphblas/python-graphblas
  - https://github.com/JuliaSparse/SuiteSparseGraphBLAS.jl
  - https://github.com/michelp/pggraphblas

- IN PROGRESS: C++ API Specification and Reference Lib.
  - https://github.com/GraphBLAS/graphblas-api-cpp
  - https://github.com/GraphBLAS/rgri
Questions?

Website: [http://graphblas.org](http://graphblas.org)
- Lists workshops and conferences
- Links to the latest API Specifications
- Teams developing implementations
- Other useful resources

Mailing list: [Graphblas@lists.lbl.gov](mailto:Graphblas@lists.lbl.gov)
- Hosted by LBL ([mailto:abuluc@lbl.gov](mailto:abuluc@lbl.gov))
- Join the Forum by joining the list

Monthly teleconference:
- Second Friday of every month, 12pm Eastern Time
- Send email ([mailto:kepner@ll.mit.edu](mailto:kepner@ll.mit.edu)) to receive the calendar invite and Zoom ID.
Backups
Graphs (GraphBLAS) and Storage (TileDB) as Sparse Linear Algebra

Timothy G. Mattson, Intel
Graph Algorithms and Linear Algebra

This is not a new idea
• At least since the 1950’s
• There is even has a book.

Benefits of graphs as linear algebra
• Well suited to memory hierarchies of modern microprocessors
• Can utilize decades of experience in distributed/parallel computing from linear algebra in supercomputing.
• Easier to understand … for some people.
SuiteSparse:GraphBLAS: An Implementation of the C API

Tim Davis, Texas A&M University
SuiteSparse:GraphBLAS v7.4.x

- **Conforms to the v2.0 C API** (Nov 2021)
- **New features:**
  - faster hypersparse matrices (the “hyperhash”, avoids binary search), in v7.3.0beta
  - pack/unpack (O(1)-time move semantics)
  - named types and operators (for future JIT)
  - matrix and vector sort
  - eWiseUnion (like eWiseAdd but with 2 scalars; all entries in output go through the operator)
  - matrix and vector iterators
  - matrix reshape
- **Performance:**
  - GrB_mxm, particularly with sparse-times-dense or dense-times-sparse. AVX2 and AVX512 exploit
  - faster MATLAB interface
- **Port to Octave 7**
- **Supported by Intel, NVIDIA, Redis, MIT Lincoln Lab, MathWorks, Julia Computing**
SuiteSparse versus the Intel MKL sparse library

<table>
<thead>
<tr>
<th>computation</th>
<th>format</th>
<th>MKL method</th>
<th>MKL time (sec)</th>
<th>SuiteSparse time (sec)</th>
<th>speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1st</td>
<td>2nd</td>
<td></td>
</tr>
<tr>
<td>y+=S*x</td>
<td>S by row</td>
<td>mkl_sparse_d_mv</td>
<td>2.54</td>
<td>1.27</td>
<td>2.10</td>
</tr>
<tr>
<td>y+=S*x</td>
<td>S by col</td>
<td>mkl_sparse_d_mv</td>
<td>7.22</td>
<td>7.22</td>
<td>3.65</td>
</tr>
<tr>
<td>C+=S*F</td>
<td>S by row, F by row</td>
<td>mkl_sparse_d_mm</td>
<td>2.95</td>
<td>1.90</td>
<td>1.49</td>
</tr>
<tr>
<td>C+=S*F</td>
<td>S by row, F by col</td>
<td>mkl_sparse_d_mm</td>
<td>6.12</td>
<td>4.99</td>
<td>4.13</td>
</tr>
<tr>
<td>C+=S*F</td>
<td>S by col, F by row</td>
<td>mkl_sparse_d_mm</td>
<td>28.82</td>
<td>28.82</td>
<td>2.09</td>
</tr>
<tr>
<td>C+=S*F</td>
<td>S by col, F by col</td>
<td>mkl_sparse_d_mm</td>
<td>78.82</td>
<td>5.17</td>
<td>8.40</td>
</tr>
<tr>
<td>C=S+B</td>
<td>S by row</td>
<td>mkl_sparse_d_add</td>
<td>30.77</td>
<td>30.77</td>
<td>21.37</td>
</tr>
<tr>
<td>C=S' +B</td>
<td>S by row</td>
<td>mkl_sparse_d_add</td>
<td>102.09</td>
<td>27.30</td>
<td>6.26</td>
</tr>
<tr>
<td>C=S'</td>
<td>S by row</td>
<td>mkl_sparse_convert_csr</td>
<td>77.27</td>
<td>77.27</td>
<td>5.22</td>
</tr>
</tbody>
</table>

Table 4. SuiteSparse vs MKL 2022 with the GAP-Twitter matrix
Work in progress and future work

- **CUDA acceleration** (with J. Eaton and C. Nolet, NVIDIA): 3x to 9x speedup in GrB_mxm
- **Julia integration** (just announced v0.7), replacing Julia SparseArrays
- more MATLAB integration
- further Python integration
- JIT for faster user-defined types and operations
- aggressive non-blocking mode, kernel fusion
- $x = A \backslash b$ over a field
- more built-in types (FP16, complex integers, …)
- faster kernels (GrB_mxm for sampled dense-dense matrix multiply)
- matrices with shallow components

https://github.com/DrTimothyAldenDavis/GraphBLAS
LAGraph: graph algorithms library

LAGraph: graph algorithm library

Version 1.0 released in September 2022

6 polished, stable algorithms (the GAP benchmark):
- Breadth-first search
- Betweenness-centrality
- PageRank
- Connected Components
- Single-source Shortest-Path
- Triangle Counting

Stable utilities
- malloc/calloc/realloc/free wrappers
- create/destroy the LAGraph_Graph
- compute properties: degree, A’, # diag entries
- delete properties
- display graph
- Matrix Market file I/O (very slow)
- Sorting
- thread control
- timing
- type management

Graphalytics algorithms in next Release

Many experimental algorithms to be curated
- K-truss, All K-truss
- Bellman-Ford single-source shortest path
- Maximal independent set
- Triangle Centrality
- Community detection w/ label propagation
- Deep Neural Network Inference
- Strongly Connected Components
- Minimum Spanning Forest
- Local Clustering Coefficient
- K-core
- Counting all size-4 graphlets
- Triangle polling
- Fiedler vector

Experimental utilities
- random matrix, vector generators
- Binary matrix file I/O (very fast), serialize/deserialize, parallel LZ4 comp.
LAGraph: graph algorithm library

Version 1.0 released in September 2022

6 polished, stable algorithms (the GAP benchmark):
- Breadth-first search
- Betweenness-centrality
- PageRank
- Connected Components
- Single-source Shortest-Path
- Triangle Counting

Stable utilities
- malloc/calloc/realloc/free wrappers
- create/destroy the LAGraph_Graph
- compute properties: degree, $A'$, # diagonal
- delete properties
- display graph
- Matrix Market file I/O (very slow)
- Sorting
- thread control
- timing
- type management

HELP WANTED

Graphalytics algorithms in next Release

Many experimental algorithms to be curated
- K-truss, All K-truss
- Bellman-Ford single-source shortest path
- Maximal independent set
- Triangle Centrality
- Community detection w/ label propagation
- Deep Neural Network Inference
- Strongly Connected Components
- Minimum Spanning Forest
- Local Clustering Coefficient
- K-core
- Counting all size-4 graphlets
- Triangle polling
- Fiedler vector

Experimental utilities
- random matrix, vector generators
- Binary matrix file I/O (very fast), serialize/deserialize, parallel LZ4 comp.

https://github.com/GraphBLAS/LAGraph
python-graphblas + NetworkX

Jim Kitchen, Anaconda,
Eric Welch, NVIDIA,
and contributors.
Python package for accelerated GraphBLAS

• python-graphblas
  • package that dispatches to SuiteSparse:GraphBLAS for computation
  • Stays in sync with advances in SuiteSparse:GraphBLAS

• graphblas-algorithms
  • Like LAGraph, a set of graphblas algorithms
  • Built on top of python graphblas
Dispatching Example with graphblas-algorithms

```python
import networkx as nx

G = nx.erdos_renyi_graph(8000, 0.02)
k = nx.k_truss(G, 5)
```

8000 nodes, ~ 640,000 edges
This takes 10.7 seconds

The k-truss is the maximal induced subgraph of G with each edge belonging to at least k-2 triangles.

```python
import networkx as nx
import graphblas_algorithms as ga

G = nx.erdos_renyi_graph(8000, 0.02)
G2 = ga.Graph.from_networkx(G)
k = nx.k_truss(G2, 5)
```

conda install -c conda-forge graphblas-algorithms
-or-
pip install graphblas-algorithms (Linux Only)

This takes 0.5 seconds
This takes 0.28 seconds

* Notice that dispatching is opt-in
### Benchmarks: GraphBLAS vs NetworkX

#### Speed-up

<table>
<thead>
<tr>
<th></th>
<th>amazon</th>
<th>google</th>
<th>pokec</th>
<th>enron</th>
<th>preferentialAttachment</th>
<th>dblp</th>
<th>citationCiteeseer</th>
<th>coAuthorsDBLP</th>
<th>as-Skitter</th>
<th>coPapersCiteeseer</th>
<th>coPapersDBLP</th>
</tr>
</thead>
<tbody>
<tr>
<td># of vertices</td>
<td>262,111</td>
<td>916,428</td>
<td>1,632,804</td>
<td>36,692</td>
<td>100,000</td>
<td>192,244</td>
<td>326,186</td>
<td>268,495</td>
<td>299,067</td>
<td>1,696,415</td>
<td>434,102</td>
</tr>
<tr>
<td># of edges</td>
<td>1,234,877</td>
<td>5,105,039</td>
<td>30,622,564</td>
<td>367,662</td>
<td>999,970</td>
<td>1,218,132</td>
<td>1,615,400</td>
<td>2,313,294</td>
<td>1,955,352</td>
<td>22,190,596</td>
<td>32,071,440</td>
</tr>
<tr>
<td>degree centrality</td>
<td>32</td>
<td>48</td>
<td>31</td>
<td>29</td>
<td>60</td>
<td>140</td>
<td>65</td>
<td>180</td>
<td>200</td>
<td>530</td>
<td>190</td>
</tr>
<tr>
<td>reciprocity</td>
<td>290</td>
<td>370</td>
<td>470</td>
<td>230</td>
<td>600</td>
<td>840</td>
<td>1600</td>
<td>1000</td>
<td>1400</td>
<td>1700</td>
<td>2200</td>
</tr>
<tr>
<td>generalized degree</td>
<td>N/A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k-truss (k=5)</td>
<td>140</td>
<td>160</td>
<td>190</td>
<td>150</td>
<td>220</td>
<td>150</td>
<td>1700</td>
<td>500</td>
<td>360</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pagerank</td>
<td>130</td>
<td>340</td>
<td>930</td>
<td>50</td>
<td>240</td>
<td>250</td>
<td>390</td>
<td>580</td>
<td>810</td>
<td>1800</td>
<td>3900</td>
</tr>
<tr>
<td>eigenvector centrality</td>
<td>53</td>
<td>120</td>
<td>150</td>
<td>61</td>
<td>650</td>
<td>740</td>
<td>1300</td>
<td>1100</td>
<td>1300</td>
<td>2000</td>
<td>5200</td>
</tr>
<tr>
<td>katz centrality</td>
<td>420</td>
<td>530</td>
<td>830</td>
<td>300</td>
<td>1100</td>
<td>1400</td>
<td>1700</td>
<td>2100</td>
<td>2300</td>
<td>3400</td>
<td>7500</td>
</tr>
<tr>
<td>clustering</td>
<td>160</td>
<td>900</td>
<td>620</td>
<td>370</td>
<td>370</td>
<td>290</td>
<td>280</td>
<td>540</td>
<td>380</td>
<td>11000</td>
<td>2600</td>
</tr>
<tr>
<td>transitivity</td>
<td>180</td>
<td>270</td>
<td>440</td>
<td>830</td>
<td>970</td>
<td>900</td>
<td>730</td>
<td>1600</td>
<td>970</td>
<td>20000</td>
<td>6600</td>
</tr>
<tr>
<td>square clustering</td>
<td>N/A</td>
<td></td>
<td></td>
<td>1200</td>
<td>950</td>
<td>1400</td>
<td>1800</td>
<td>1100</td>
<td>1300</td>
<td>DNF</td>
<td>DNF</td>
</tr>
<tr>
<td>pagerank (scipy)</td>
<td>3.4</td>
<td>14</td>
<td>23</td>
<td>2.1</td>
<td>3.3</td>
<td>3.8</td>
<td>6.3</td>
<td>9.8</td>
<td>11</td>
<td>20</td>
<td>23</td>
</tr>
</tbody>
</table>

#### Hardware: NVIDIA DGX-1
- CPU: Dual 20 Core Intel Xeon E5-2698 v4 2.2GHz
- RAM: 512 GB 2133 MHz DDR4 RDIMM

© 2023 Carnegie Mellon University
How to Try It Out

Dispatching is a feature in NetworkX 3.0

• Note: This is an experimental feature, and the API may change. Do not rely on this for production applications.

Install graphblas-algorithms and optional dependencies

• `conda install -c conda-forge graphblas-algorithms`
• `conda install pandas scipy`  # needed for display and converting to NetworkX

Try the Dispatch Example

• https://github.com/python-graphblas/graphblas-algorithms