GraphBLAS Updates

SC BoF: HPC Graph Toolkits and GraphBLAS Forum
20 November 2019

Scott McMillan
Email: smcmillan@sei.cmu.edu

Carnegie Mellon University
Software Engineering Institute
Copyright 2019 Carnegie Mellon University.

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.

DM19-1214
A Brief History…

- Sep. 2013: GraphBLAS “position paper” at IEEE HPEC
- Jun. 2015: GraphBLAS Forum “east coast/west coast” kickoff meeting
- Dec. 2015: Formation of the “GraphBLAS Signature Proposal Subcommittee” (C API Subcommittee)
- May 2017: GraphBLAS C API Specification v1.0 released (“provisional”)
- Nov. 2017: SuiteSparse GraphBLAS v1.0 released
- May 2018: IBM GraphBLAS released, “provisional” removed from C API spec. (v1.2.0)
- Sep. 2019: GraphBLAS C API Specification v1.3.0 released
### GraphBLAS Primitives

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Mathematical Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mxm</td>
<td>Matrix multiplication (breadth-first traversal)</td>
<td>( C \langle \neg M, z \rangle = C \circ (A^T \oplus B^T) )</td>
</tr>
<tr>
<td>mxv, (vxm)</td>
<td></td>
<td>( c \langle \neg m, z \rangle = c \circ (A^T \oplus b) )</td>
</tr>
<tr>
<td>eWiseMult</td>
<td>Element-wise ‘multiplication’ (graph intersection)</td>
<td>( C \langle \neg M, z \rangle = C \circ (A^T \otimes B^T) )</td>
</tr>
<tr>
<td>eWiseAdd</td>
<td>Element-wise ‘addition’ (graph union)</td>
<td>( C \langle \neg M, z \rangle = C \circ (A^T \oplus B^T) )</td>
</tr>
<tr>
<td>reduce (row/col)</td>
<td>Reduce along rows/cols (vertex degree)</td>
<td>( c \langle \neg m, z \rangle = c \circ [\bigoplus, A^T(:,j)] )</td>
</tr>
<tr>
<td>apply</td>
<td>Apply unary function to each element (edge modification)</td>
<td>( C \langle \neg M, z \rangle = C \circ f(A^T) )</td>
</tr>
<tr>
<td>transpose</td>
<td>Swap rows and columns (reverse directed edges)</td>
<td>( C \langle \neg M, z \rangle = C \circ A^T )</td>
</tr>
<tr>
<td>extract</td>
<td>Extract a sub-matrix (sub-graph selection)</td>
<td>( C \langle \neg M, z \rangle = C \circ A^T(i,j) )</td>
</tr>
<tr>
<td>assign</td>
<td>Assign to a sub-matrix (sub-graph assignment)</td>
<td>( C \langle \neg M, z \rangle(i,j) = C(i,j) \circ A^T )</td>
</tr>
<tr>
<td>kronecker</td>
<td>Compute the Kronecker product of two matrices</td>
<td>( C \langle \neg M, z \rangle = C \circ (A^T \otimes B^T) )</td>
</tr>
<tr>
<td>build (meth.)</td>
<td>Build a matrix from row, column, value tuples</td>
<td>( C = S_{m \times n}(i,j, v, \circ) )</td>
</tr>
<tr>
<td>extractTuples (meth.)</td>
<td>Extract row, column, value tuples from a matrix</td>
<td>( (i,j, v) = A )</td>
</tr>
</tbody>
</table>

Notation: i,j – index arrays, v – scalar array, m – 1D mask, other bold-lower – vector (column), M – 2D mask, other bold-caps – matrix, T – transpose, \( \neg \) – structural complement, z – clear output, \( \oplus \) monoid/binary function, \( \otimes \) semiring, blue – optional parameters, red – optional modifiers.
GraphBLAS C API Spec. v1.3.0: Updates at a Glance

- Execution model updates (especially “completion”).
- New API methods and operations
- New predefined types
- Language Clarifications
- Miscellany
Language Regarding Completion Changed

The Concept of “Completion”.

- A GraphBLAS Object is “complete” when it resides in memory and is “available” for another process or thread to access.
- This is required to construct “Happens Before” relations … which are the foundational concept for reasoning about concurrency

In GraphBLAS 1.0 to 1.2

- Completion of all involved GraphBLAS objects was required anytime a transparent object (e.g. nvals) was returned.
  - We combined the concepts of delivering a visible result and completion.
- This inhibits a number of key optimizations:
  - Example: triangle counting where we do matrix products and compute reductions on that matrix, but we never need the full matrix.

Solution for GraphBLAS 1.3

- Completion is a separate concept implemented by a call to GrB_wait(). This allows the obvious optimization for triangle count.
- We added a per-object GrB_wait() … e.g. GrB_wait(A)
New Operation: Kronecker matrix product

GrB_Info GrB_kronecker(GrB_Matrix C, 
    const GrB_Matrix Mask, 
    const GrB_BinaryOp accum, 
    const GrB_BinaryOp op, // or monoid/semiring 
    const GrB_Matrix A, 
    const GrB_Matrix B, 
    const GrB_Descriptor desc);

GrB_kronecker computes the Kronecker product $C = A \otimes B$ or, if an optional binary accumulation operator ($\odot$) is provided, $C = C \odot (A \otimes B)$ (where matrices $A$ and $B$ can be optionally transposed). The Kronecker product is defined as follows:

$$C = A \otimes B = \begin{bmatrix} A_{0,0} \otimes B & A_{0,1} \otimes B & \cdots & A_{0,n_A-1} \otimes B \\ A_{1,0} \otimes B & A_{1,1} \otimes B & \cdots & A_{1,n_A-1} \otimes B \\ \vdots & \vdots & \ddots & \vdots \\ A_{m_A-1,0} \otimes B & A_{m_A-1,1} \otimes B & \cdots & A_{m_A-1,n_A-1} \otimes B \end{bmatrix}$$

where $A : S^{m_A \times n_A}$, $B : S^{m_B \times n_B}$, and $C : S^{m_A m_B \times n_A n_B}$. More explicitly, the elements of the Kronecker product are defined as

$$C(i_{A m_B} + i_B, j_{A n_B} + j_B) = A_{i_A j_A} \otimes B_{i_B j_B}$$
New variants of apply operation: binary op + scalar

• “Bind first”

\[ \text{C} \langle \neg M, z \rangle \bigoplus = f_b(s, A) \]

\[
\text{GrB_Info GrB_apply} (\text{GrB_Matrix} C, \text{const GrB_Matrix Mask, \text{const GrB_BinaryOp acc, \text{const GrB_BinaryOp op, \text{<type> s, \text{const GrB_Matrix A, \text{const GrB_Descriptor desc})}})}}
\]

\[ w \langle \neg m, z \rangle \bigoplus = f_b(s, u) \]

\[
\text{GrB_Info GrB_apply} (\text{GrB_Vector w, \text{const GrB_Vector mask, \text{const GrB_BinaryOp acc, \text{const GrB_BinaryOp op, \text{<type> s, \text{const GrB_Vector u, \text{const GrB_Descriptor desc})}})}}
\]

• “Bind second”

\[ \text{C} \langle \neg M, z \rangle \bigoplus = f_b(A, s) \]

\[
\text{GrB_Info GrB_apply} (\text{GrB_Matrix C, \text{const GrB_Matrix Mask, \text{const GrB_BinaryOp accum, \text{const GrB_BinaryOp op, \text{<type> s, \text{const GrB_Matrix A, \text{const GrB_Descriptor desc})}})}}
\]

\[ w \langle \neg m, z \rangle \bigoplus = f_b(u, s) \]

\[
\text{GrB_Info GrB_apply} (\text{GrB_Vector w, \text{const GrB_Vector mask, \text{const GrB_BinaryOp accum, \text{const GrB_BinaryOp op, \text{<type> s, \text{const GrB_Vector u, \text{const GrB_Descriptor desc})}})}}
\]
New matrix and vector methods

• Resize matrices and vectors (larger and smaller):

```c
GrB_Info GrB_Matrix_resize(GrB_Matrix C,
                          GrB_Index new_nrows,
                          GrB_Index new_ncols);
GrB_Info GrB_Vector_resize(GrB_Vector w,
                         GrB_Index new_nsize);
```

• Remove single elements from matrices and vectors.

```c
GrB_Info GrB_Matrix_removeElement(GrB_Matrix C,
                                   GrB_Index row_index,
                                   GrB_Index col_index);
GrB_Info GrB_Vector_removeElement(GrB_Vector w,
                                   GrB_Index index);
```
Mask changes

- **Added GrB_STRUCTURE** (structure only)
  - Original mode (evaluate to true) is still the default
- **Deprecating GrB_SCMP symbol for GrB_COMP**
- **Can compose GrB_COMP with GrB_STRUCTURE** (complement of the structure only)

All 32 possible combinations of descriptor flags are now predefined:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>GrB.OUTP</th>
<th>GrB.MASK</th>
<th>GrB.INP0</th>
<th>GrB.INP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrB.NULL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.T0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.T0T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.S</td>
<td></td>
<td>GrB.COMP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.CT1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.ST1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.ST0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.CT0T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.ST0T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.SC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.SCT1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.SCT0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.SCT0T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RT1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RT0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RT0T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RCT1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RST1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RCT0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RST0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RCT0T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RST0T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RSC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RSTC1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RSTC0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GrB.DESC.RSTC0T1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
New Predefined Operators and Monoids

- **Unary Operators**
  - `GrB_ABS_T` – absolute value
  - `GrB_BNOT_I` – bitwise comp. (integers only)

- **Monoids (and their identities)**
  - `GrB_PLUS_MONOID_T` – (+, 0)
  - `GrB_TIMES_MONOID_T` – (x, 1)
  - `GrB_MIN_MONOID_T` – (min, uint/int_max or infinity)
  - `GrB_MAX_MONOID_T` – (max, 0, int_min or -infinity)
  - `GrB_LOR_MONOID_BOOL` – logical OR
  - `GrB_LAND_MONOID_BOOL` – logical AND
  - `GrB_LXOR_MONOID_BOOL` – logical XOR
  - `GrB_LXNOR_MONOID_BOOL` – logical XNOR

- **Binary Operators**
  - `GrB_LXNOR` – logical XNOR (bool)
  - `GrB_BOR_I` – bitwise OR (ints)
  - `GrB_BAND_I` – bitwise AND (ints)
  - `GrB_BXOR_I` – bitwise XOR (ints)
  - `GrB_BXNOR_I` – bitwise XNOR (ints)
### New Predefined Semirings

#### “True” semirings:

(additive identity == multiplicative annihilator)

<table>
<thead>
<tr>
<th>GraphBLAS identifier</th>
<th>$(T \times T \to T)$</th>
<th>$+ \text{ identity}$</th>
<th>$\times \text{ annihilator}$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrB_PLUS_TIMES_SEMIRING_T</td>
<td>UINTx</td>
<td>0</td>
<td>0</td>
<td>arithmetic semiring</td>
</tr>
<tr>
<td>GrB_MIN_PLUS_SEMIRING_T</td>
<td>INTx</td>
<td>0</td>
<td>INTx_MAX</td>
<td>min-plus semiring</td>
</tr>
<tr>
<td>GrB_MAX_PLUS_SEMIRING_T</td>
<td>FPx</td>
<td>0</td>
<td>INFINITY</td>
<td>max-plus semiring</td>
</tr>
<tr>
<td>GrB_MIN_TIMES_SEMIRING_T</td>
<td>UINTx</td>
<td>0</td>
<td>INTx_MAX</td>
<td>min-times semiring</td>
</tr>
<tr>
<td>GrB_MIN_MAX_SEMIRING_T</td>
<td>INTx</td>
<td>0</td>
<td>INTx_MAX</td>
<td>min-max semiring</td>
</tr>
<tr>
<td>GrB_MAX_MIN_SEMIRING_T</td>
<td>FPx</td>
<td>0</td>
<td>INFINITY</td>
<td>max-min semiring</td>
</tr>
<tr>
<td>GrB_MAX_TIMES_SEMIRING_T</td>
<td>UINTx</td>
<td>0</td>
<td>INTx_MIN</td>
<td>max-times semiring</td>
</tr>
<tr>
<td>GrB_PLUS_MIN_SEMIRING_T</td>
<td>INTx</td>
<td>0</td>
<td>-INFINITY</td>
<td>plus-min semiring</td>
</tr>
<tr>
<td>GrB_LOR_LAND_SEMIRING_BOOL</td>
<td>BOOL</td>
<td>false</td>
<td>true</td>
<td>Logical semiring</td>
</tr>
<tr>
<td>GrB_LAND_LOR_SEMIRING_BOOL</td>
<td>BOOL</td>
<td>true</td>
<td>false</td>
<td>”and-or” semiring</td>
</tr>
<tr>
<td>GrB_LXOR_LAND_SEMIRING_BOOL</td>
<td>BOOL</td>
<td>false</td>
<td>true</td>
<td>same as NEQ_LAND</td>
</tr>
<tr>
<td>GrB_LXNOR_LAND_SEMIRING_BOOL</td>
<td>BOOL</td>
<td>true</td>
<td>true</td>
<td>same as EQ_LOR</td>
</tr>
</tbody>
</table>
**New Predefined Semirings**

“Useful” semirings

(no multiplicative annihilator)

<table>
<thead>
<tr>
<th>GraphBLAS identifier</th>
<th>Domains, $T$</th>
<th>+ identity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GrB_MAX_PLUS_SEMIRING_T</td>
<td>UINT_x</td>
<td>0</td>
<td>max-plus semiring</td>
</tr>
<tr>
<td>GrB_MIN_TIMES_SEMIRING_T</td>
<td>INT_x</td>
<td>INT_x_MAX</td>
<td>min-times semiring</td>
</tr>
<tr>
<td></td>
<td>FP_x</td>
<td>INFINITY</td>
<td>max-times semiring</td>
</tr>
<tr>
<td>GrB_MAX_TIMES_SEMIRING_T</td>
<td>INT_x</td>
<td>INT_x_MIN</td>
<td>plus-min semiring</td>
</tr>
<tr>
<td></td>
<td>FP_x</td>
<td>-INFINITY</td>
<td></td>
</tr>
<tr>
<td>GrB_PLUS_MIN_SEMIRING_T</td>
<td>INT_x</td>
<td>0</td>
<td>min-select first semiring</td>
</tr>
<tr>
<td></td>
<td>FP_x</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>GrB_MIN_FIRST_SEMIRING_T</td>
<td>UINT_x</td>
<td>UINT_x_MAX</td>
<td>min-select second semiring</td>
</tr>
<tr>
<td></td>
<td>INT_x</td>
<td>INT_x_MAX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FP_x</td>
<td>INFINITY</td>
<td></td>
</tr>
<tr>
<td>GrB_MIN_SECOND_SEMIRING_T</td>
<td>UINT_x</td>
<td>UINT_x_MAX</td>
<td>min-select second semiring</td>
</tr>
<tr>
<td></td>
<td>INT_x</td>
<td>INT_x_MAX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FP_x</td>
<td>INFINITY</td>
<td></td>
</tr>
<tr>
<td>GrB_MAX_FIRST_SEMIRING_T</td>
<td>UINT_x</td>
<td>0</td>
<td>max-select first semiring</td>
</tr>
<tr>
<td></td>
<td>INT_x</td>
<td>INT_x_MIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FP_x</td>
<td>-INFINITY</td>
<td></td>
</tr>
<tr>
<td>GrB_MAX_SECOND_SEMIRING_T</td>
<td>UINT_x</td>
<td>0</td>
<td>max-select second semiring</td>
</tr>
<tr>
<td></td>
<td>INT_x</td>
<td>INT_x_MIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FP_x</td>
<td>-INFINITY</td>
<td></td>
</tr>
</tbody>
</table>
Miscellany

• Added run-time getVersion() and compile-time version macros
• Updated all code examples
  • Update to use new capabilities where possible
  • Bug fix in non-batch BC code
  • Added parent-BFS example
• Clarifications:
  • Distributive law
  • init/finalize errors
  • Boolean/integer division
  • Aliasing in user-defined operators
  • Freeing predefined objects
  • Removed unnecessary language about annihilators and implied zeros
GraphBLAS Implementations

**SuiteSparse** library (Texas A&M): First fully conforming GraphBLAS C API release.
- [http://faculty.cse.tamu.edu/davis/suitesparse.html](http://faculty.cse.tamu.edu/davis/suitesparse.html)

**IBM-GraphBLAS**: the second fully conforming C API release,
- [https://github.com/IBM/ibmgraphblas](https://github.com/IBM/ibmgraphblas)

**GBTL**: GraphBLAS Template Library (CMU/SEI/PNNL/IU): GraphBLAS C++ API
- [https://github.com/cmu-sei/gbtl](https://github.com/cmu-sei/gbtl)

**GraphBLAST**: A C++ implementations for GraphBLAS for GPUs (UC Davis),
- [https://github.com/gunrock/graphblast](https://github.com/gunrock/graphblast)

**Distributed GBTL**: in progress (LLNL)
- See Roger Pearce

Python bindings:
- **pyGraphBLAS**: A Python Wrapper around SuiteSparse GraphBLAS
  - [https://github.com/michelp/pygraphblas](https://github.com/michelp/pygraphblas)
- **pyGB**: A Python Wrapper around GBTL (UW/PNNL/CMU)
  - [https://github.com/jessecoleman/gbtl-python-binding](https://github.com/jessecoleman/gbtl-python-binding)

**pggraphblas**: A PostgreSQL wrapper around SuiteSparse GraphBLAS
- [https://github.com/michelp/pggraphblas](https://github.com/michelp/pggraphblas)

Matlab and Julia wrappers around SuiteSparse GraphBLAS
- [https://aldenmath.com](https://aldenmath.com)

Etc...
LA Graph: Curating a collection of high-level graph algorithms

- Launched May 2019 at GrAPL’19.
- Algorithms (current/pending):
  - BFS (simple, push/pull)
  - PageRank (multiple)
  - Triangle counting
  - K-truss (and All K-truss)
  - Clustering (label propagation)
  - Local Clustering Coefficient
  - SSSP (Bellman-Ford x 2, delta-stepping)
  - Connected components (LACC, FastSV)
  - Betweenness Centrality (single node, batch)
  - MIS (Luby’s)
  - All GAP Benchmark Algorithms (under development).
- [https://github.com/GraphBLAS/LAGraph](https://github.com/GraphBLAS/LAGraph) (BSD 2 Clause).
GraphBLAS Forum Information

Steering Committee: David Bader, Aydın Buluç, John Gilbert, Jeremy Kepner, Tim Mattson, Henning Meyerhenke

API Subcommittee: Benjamin Brock, Aydın Buluç, Tim Mattson, Scott McMillan, Jose Moreira

Mailing list: Graphblas@lists.lbl.gov
  • Join the Forum by joining the list (mailto:abuluc@lbl.gov)

Website: http://graphblas.org
  • Link to the latest C API Specification
  • Lists teams developing implementations
  • Other useful resources including the “The Math Document”

Monthly teleconference:
  • Second Friday of every month, 12pm Eastern Time
  • Send email to Jeremy Kepner to receive the calendar invite.
Backups
GraphBLAS Signatures: mxm

\[ C \leftarrow M, z \right\} = C \odot \left( A^T \oplus . \otimes B^T \right) \]

// GBTL C++ API
namespace GraphBLAS
{
    template <typename CMatrixT,
              typename MaskT,
              typename AccumT,
              typename SemiringT,
              typename AMatrixT,
              typename BMatrixT>
    void mxm(CMatrixT &C,
              MaskT &M,
              AccumT &accum,
              SemiringT &op,
              AMatrixT &A,
              BMatrixT &B,
              bool replace_flag);
}
GraphBLAS Signatures: \texttt{mxm}

\[
C \langle \neg M, z \rangle = C \odot (A^T \oplus \odot B^T)
\]

\begin{verbatim}
// GraphBLAS C API
GrB_Info GrB_mxm(
    GrB_Matrix C,
    GrB_Matrix const M,
    GrB_BinaryOp const accum,
    GrB_Semiring const op,
    GrB_Matrix const A,
    GrB_Matrix const B,
    GrB_Descriptor const desc);
\end{verbatim}

- \texttt{C} stores the result
- \texttt{C} is also used as input if an optional accumulation operator (\texttt{\odot}) is specified.
GraphBLAS Signatures: m\text{xm}

GraphBLAS C API

// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix C,
    GrB_Matrix const M,
    GrB_BinaryOp const accum,
    GrB_Semiring const op,
    GrB_Matrix const A,
    GrB_Matrix const B,
    GrB_Descriptor const desc);

\[
C^{\langle -M, z \rangle} = C \circ (A^T \oplus \otimes B^T)
\]

- Mask, \(M\), is optional.
- If not specified (GrB_NULL), the entire \(C\) matrix is overwritten.
GraphBLAS Signatures: mxm

\[ C \langle \neg M, z \rangle = C \odot (A^T \oplus \odot B^T) \]

// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix C,
                 GrB_Matrix const M,
                 GrB_BinaryOp const accum,
                 GrB_Semiring const op,
                 GrB_Matrix const A,
                 GrB_Matrix const B,
                 GrB_Descriptor const desc);

- The accumulation operator, \( \odot \), is optional.
- If not specified (GrB_NULL), the \( C \) matrix is used as output only (i.e., does not appear on the right hand side).
GraphBLAS Signatures: `mxm`

\[
C\langle -M, z \rangle = C \odot (A^T \oplus \otimes B^T)
\]

// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix C,
    GrB_Matrix const M,
    GrB_BinaryOp const accum,
    GrB_Semiring const op,
    GrB_Matrix const A,
    GrB_Matrix const B,
    GrB_Descriptor const desc);

- The semiring used in the matrix multiply.
  - \(\oplus\), a commutative monoid, replaces “plus”
  - \(\otimes\), a binary operator, replaces “times”
- More on this is a minute…
GraphBLAS Signatures: $\text{mxm}$

\[
\mathbf{C}^\langle -\mathbf{M}, z \rangle = \mathbf{C} \odot (\mathbf{A}^T \oplus \odot \mathbf{B}^T)
\]

// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix C,
  GrB_Matrix const M,
  GrB_BinaryOp const accum,
  GrB_Semiring const op,
  GrB_Matrix const A,
  GrB_Matrix const B,
  GrB_Descriptor const desc);
GraphBLAS Signatures: \texttt{m xm}

\[
\mathbf{C} \langle \neg \mathbf{M}, \mathbf{z} \rangle = \mathbf{C} \odot (\mathbf{A}^T \oplus \mathbf{e} \circ \mathbf{B}^T)
\]

// GraphBLAS C API
GrB_Info GrB_mxm(GrB_Matrix C,
    GrB_Matrix const M,
    GrB_BinaryOp const accum,
    GrB_Semiring const op,
    GrB_Matrix const A,
    GrB_Matrix const B,
    GrB_Descriptor const desc);

• Optional Descriptor can specify any or all of the following:
  - Complement the mask, \( \neg \)
  - Clear the output matrix before writing final result, \( \mathbf{z} \)
  - Transpose any input matrix, \( \mathbf{T} \)
Semi-ring algebra defines the properties of the math to be performed.

Mathematics of Big Data: Spreadsheets, Databases, Matrices, and Graphs, Hayden Jansen & Jeremy Kepner, MIT Press, 2018
Matrix Multiply using Semirings

\[
C = A \oplus \otimes B
\]

- The Semiring \((\oplus, \otimes)\) determines how this computation is carried out.
- Consists of two Monoids (binary operator, identity)
  - \(\oplus\), Commutative Monoid: e.g., \((\text{add}, 0)\)
  - \(\otimes\), Monoid: e.g., \((\text{multiply}, 1)\)
  
  where \(\oplus\)'s identity = \(\otimes\)'s annihilator
- In GraphBLAS:
  - \(\oplus\), is a Commutative Monoid with identity (same as above)
  - \(\otimes\), is a Binary Function (i.e., no identity required)
  - No enforcement that \(\oplus\)'s identity = \(\otimes\)'s annihilator, e.g. \(\oplus=\text{(min, } \infty)\) and \(\otimes=\text{‘second’}\)