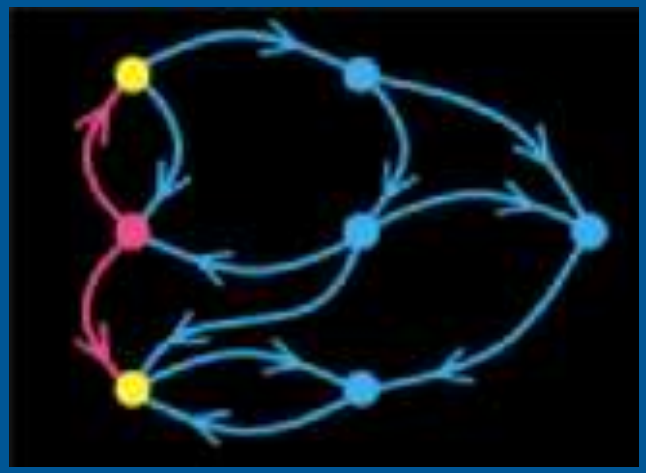
GraphBLAS BoF @ HPEC19

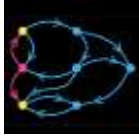


Co-chairs:

Tim Mattson, Intel

Marcin Zalewski, NIAC/PNNL

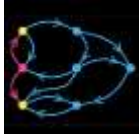
Scott McMillan, CMU/SEI



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.

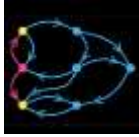
DM19-0930

Agenda



- **The GraphBLAS Forum Overview**
- C API Specification “Report”
- Short Topics
  - Trevor Steil/Roger Pearce: *Distributed GraphBLAS*
- Invited Speaker: Michel Pelletier
  - *pygraphblas: a Python API for GraphBLAS*
- Open discussion

GraphBLAS Forum Information

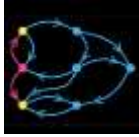


Steering Committee: David Bader, Aydın Buluç, John Gilbert, Jeremy Kepner, Tim Mattson, Henning Meyerhenke

Mailing list: Graphblas@lists.lbl.gov

- Hosted by LBL (mailto:abuluc@lbl.gov)
- Join the Forum by joining the list
- Current membership is 100+ (as of August 2019)

GraphBLAS Forum Information



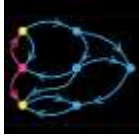
Website: http://graphblas.org

- Hosted by UCSB
- Lists workshops and conferences
- 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.

GraphBLAS Forum Updates: C API

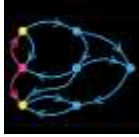


C API Subcommittee: Aydın Buluç, Tim Mattson, Scott McMillan, José Moreira, Carl Yang (graduating), Benjamin Brock (incoming).

C API Specification Schedule:

- Version 1.2.0 released May 2018
  - Removed “provisional” with two complete implementations.
- Version 1.3.0 released today! (September 2019)
  - More details in a minute
- Regular specification committee meetings still held
  - Answer questions regarding the specification language
  - Discuss proposals for changes and extensions
- Next priorities: parallel/distributed, C++ API, testing

GraphBLAS Implementations



SuiteSparse library (Texas A&M): First fully conforming GraphBLAS release.

- http://faculty.cse.tamu.edu/davis/suitesparse.html

GraphBLAS C (IBM): the second fully conforming release,

- https://github.com/IBM/ibmgraphblas

GBTL: GraphBLAS Template Library (CMU/SEI/PNNL/IU): GraphBLAS C++ API

- https://github.com/cmu-sei/gbtl

GraphBLAST: A C++ implementations for GraphBLAS for GPUs (UC Davis),

- https://github.com/gunrock/graphblast

Python bindings:

- PyGraphBLAS: A Python Wrapper around SuiteSparse GraphBLAS
  - https://github.com/michelp/pygraphblas
- PyGB: A Python Wrapper around GBTL (UW/PNNL/CMU)
  - https://github.com/jessecoleman/gbtl-python-binding

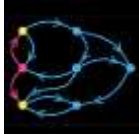
pggraphblas: A PostgreSQL wrapper around Suite Sparse GraphBLAS

- https://github.com/michelp/pggraphblas

Matlab and Julia wrappers around SuiteSparse GraphBLAS

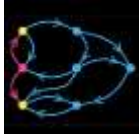
- https://aldenmath.com

Agenda



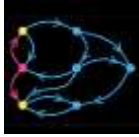
- The GraphBLAS Forum Overview
- **C API Specification “Report”**
- Short Topics
  - Trevor Steil/Roger Pearce: *Distributed GraphBLAS*
- Invited Speaker: Michel Pelletier
  - *pygraphblas: a Python API for GraphBLAS*
- Open discussion

GraphBLAS C API Spec. v1.3.0 is Released!



- 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

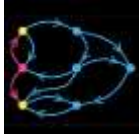
In GraphBLAS 1.0 to 1.2

- 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);
```

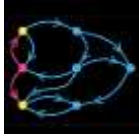
GrB_kronecker computes the Kronecker product $C = A \otimes B$ or, if an optional binary accumulation operator ($\odot$) is provided, $C = C \odot (A \otimes B)$ (where matrices A and B can be optionally transposed). The Kronecker product is defined as follows:

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

where $A : \mathbb{S}^{m_A \times n_A}$, $B : \mathbb{S}^{m_B \times n_B}$, and $C : \mathbb{S}^{m_A m_B \times n_A n_B}$. 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



```
GrB_Info GrB_apply(
   GrB_Matrix           C,
   const GrB_Matrix     Mask,
   const GrB_BinaryOp   accum,
   const GrB_BinaryOp   op,
   <type>               s,
   const GrB_Matrix     A,
   const GrB_Descriptor desc);
```

$$\mathbf{C}\langle\neg\mathbf{M},z\rangle \oplus= f_b(s, \mathbf{A})$$

```
GrB_Info GrB_apply(
   GrB_Vector           w,
   const GrB_Vector     mask,
   const GrB_BinaryOp   accum,
   const GrB_BinaryOp   op,
   <type>               s,
   const GrB_Vector     u,
   const GrB_Descriptor desc);
```

$$\mathbf{w}\langle\neg\mathbf{m},z\rangle \oplus= f_b(s, \mathbf{u})$$

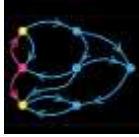
```
GrB_Info GrB_apply(
   GrB_Matrix           C,
   const GrB_Matrix     Mask,
   const GrB_BinaryOp   accum,
   const GrB_BinaryOp   op,
   const GrB_Matrix     A,
   <type>               s,
   const GrB_Descriptor desc);
```

$$\mathbf{C}\langle\neg\mathbf{M},z\rangle \oplus= f_b(\mathbf{A}, s)$$

```
GrB_Info GrB_apply(
   GrB_Vector           w,
   const GrB_Vector     mask,
   const GrB_BinaryOp   accum,
   const GrB_BinaryOp   op,
   const GrB_Vector     u,
   <type>               s,
   const GrB_Descriptor desc);
```

$$\mathbf{w}\langle\neg\mathbf{m},z\rangle \oplus= f_b(\mathbf{u}, s)$$

New matrix and vector methods



- Resize matrices and vectors (larger and smaller):

```
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.

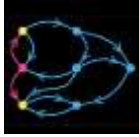
```
GrB_Info GrB_Matrix_removeElement(GrB_Matrix   C,
                                  GrB_Index    row_index,
                                  GrB_Index    col_index);
GrB_Info GrB_Vector_removeElement(GrB_Vector   w,
                                  GrB_Index    index);
```

Mask changes

- Added **`GrB_STRUCTURE`** (structure only)
  - Original mode (evaluate to true) is still the default
- Deprecating **`GrB_SCMP`** for **`GrB_COMP`**
- Can compose **`GrB_COMP`** with **`GrB_STRUCTURE`** (complement of the structure only)
- All 32 possible combinations are now predefined:

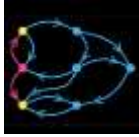
Identifier	GrB_OUTP	GrB_MASK	GrB_INP0	GrB_INP1
GrB_NULL	–	–	–	–
GrB_DESC_T1	–	–	–	GrB_TRAN
GrB_DESC_T0	–	–	GrB_TRAN	–
GrB_DESC_T0T1	–	–	GrB_TRAN	GrB_TRAN
GrB_DESC_C	–	GrB_COMP	–	–
GrB_DESC_S	–	GrB_STRUCTURE	–	–
GrB_DESC_CT1	–	GrB_COMP	–	GrB_TRAN
GrB_DESC_ST1	–	GrB_STRUCTURE	–	GrB_TRAN
GrB_DESC_CT0	–	GrB_COMP	GrB_TRAN	–
GrB_DESC_ST0	–	GrB_STRUCTURE	GrB_TRAN	–
GrB_DESC_CT0T1	–	GrB_COMP	GrB_TRAN	GrB_TRAN
GrB_DESC_ST0T1	–	GrB_STRUCTURE	GrB_TRAN	GrB_TRAN
GrB_DESC_SC	–	GrB_STRUCTURE, GrB_COMP	–	–
GrB_DESC_SCT1	–	GrB_STRUCTURE, GrB_COMP	–	GrB_TRAN
GrB_DESC_SCT0	–	GrB_STRUCTURE, GrB_COMP	GrB_TRAN	–
GrB_DESC_SCT0T1	–	GrB_STRUCTURE, GrB_COMP	GrB_TRAN	GrB_TRAN
GrB_DESC_R	GrB_REPLACE	–	–	–
GrB_DESC_RT1	GrB_REPLACE	–	–	GrB_TRAN
GrB_DESC_RT0	GrB_REPLACE	–	GrB_TRAN	–
GrB_DESC_RT0T1	GrB_REPLACE	–	GrB_TRAN	GrB_TRAN
GrB_DESC_RC	GrB_REPLACE	GrB_COMP	–	–
GrB_DESC_RS	GrB_REPLACE	GrB_STRUCTURE	–	–
GrB_DESC_RCT1	GrB_REPLACE	GrB_COMP	–	GrB_TRAN
GrB_DESC_RST1	GrB_REPLACE	GrB_STRUCTURE	–	GrB_TRAN
GrB_DESC_RCT0	GrB_REPLACE	GrB_COMP	GrB_TRAN	–
GrB_DESC_RST0	GrB_REPLACE	GrB_STRUCTURE	GrB_TRAN	–
GrB_DESC_RCT0T1	GrB_REPLACE	GrB_COMP	GrB_TRAN	GrB_TRAN
GrB_DESC_RST0T1	GrB_REPLACE	GrB_STRUCTURE	GrB_TRAN	GrB_TRAN
GrB_DESC_RSC	GrB_REPLACE	GrB_STRUCTURE, GrB_COMP	–	–
GrB_DESC_RSCT1	GrB_REPLACE	GrB_STRUCTURE, GrB_COMP	–	GrB_TRAN
GrB_DESC_RSCT0	GrB_REPLACE	GrB_STRUCTURE, GrB_COMP	GrB_TRAN	–
GrB_DESC_RSCT0T1	GrB_REPLACE	GrB_STRUCTURE, GrB_COMP	GrB_TRAN	GrB_TRAN

New Predefined Operators and Monoids



- Unary Operators
  - **GrB_ABS_***T* – absolute value
  - **GrB_BNOT_***I* – bitwise comp. (integers only)
- 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)
- 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

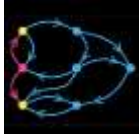
New Predefined Semirings



"True" semirings (additive identity == multiplicative annihilator)

GraphBLAS identifier	$(T \times T \to T)$	+ identity × annihilator	Description
GrB_PLUS_TIMES_SEMIRING_T	UINTx	0	arithmetic semiring
	INTx	0	
	FPx	0	
GrB_MIN_PLUS_SEMIRING_T	UINTx	UINTx_MAX	min-plus semiring
	INTx	INTx_MAX	
	FPx	INFINITY	
GrB_MAX_PLUS_SEMIRING_T	INTx	INTx_MIN	max-plus semiring
	FPx	-INFINITY	
GrB_MIN_TIMES_SEMIRING_T	UINTx	UINTx_MAX	min-times semiring
GrB_MIN_MAX_SEMIRING_T	UINTx	UINTx_MAX	min-max semiring
	INTx	INTx_MAX	
	FPx	INFINITY	
GrB_MAX_MIN_SEMIRING_T	UINTx	0	max-min semiring
	INTx	INTx_MIN	
	FPx	-INFINITY	
GrB_MAX_TIMES_SEMIRING_T	UINTx	0	max-times semiring
GrB_PLUS_MIN_SEMIRING_T	UINTx	0	plus-min semiring
GrB_LOR_LAND_SEMIRING_BOOL	BOOL	false	Logical semiring
GrB_LAND_LOR_SEMIRING_BOOL	BOOL	true	"and-or" semiring
GrB_LXOR_LAND_SEMIRING_BOOL	BOOL	false	same as NEQ_LAND
GrB_LXNOR_LOR_SEMIRING_BOOL	BOOL	true	same as EQ_LOR

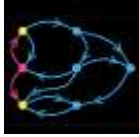
New Predefined Semirings



"Useful" semirings (no multiplicative annihilator)

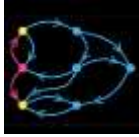
GraphBLAS identifier	Domains, T ($T \times T \to T$)	+ identity	Description
GrB_MAX_PLUS_SEMIRING_T	UINTx	0	max-plus semiring
GrB_MIN_TIMES_SEMIRING_T	INTx	INTx_MAX	min-times semiring
	FPx	INFINITY	
GrB_MAX_TIMES_SEMIRING_T	INTx	INTx_MIN	max-times semiring
	FPx	-INFINITY	
GrB_PLUS_MIN_SEMIRING_T	INTx	0	plus-min semiring
	FPx	0	
GrB_MIN_FIRST_SEMIRING_T	UINTx	UINTx_MAX	min-select first semiring
	INTx	INTx_MAX	
	FPx	INFINITY	
GrB_MIN_SECOND_SEMIRING_T	UINTx	UINTx_MAX	min-select second semiring
	INTx	INTx_MAX	
	FPx	INFINITY	
GrB_MAX_FIRST_SEMIRING_T	UINTx	0	max-select first semiring
	INTx	INTx_MIN	
	FPx	-INFINITY	
GrB_MAX_SECOND_SEMIRING_T	UINTx	0	max-select second semiring
	INTx	INTx_MIN	
	FPx	-INFINITY	

Miscellany



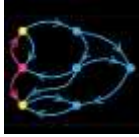
- Added run-time getVersion() and compile-time version macros
- Updated all code examples
  - Update to use new capabilities where possible
  - Bug fix in non-batch BC code
  - Added parent-BFS example
- Clarifications:
  - Distributive law
  - init/finalize errors
  - Boolean/integer division
  - Aliasing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

Agenda



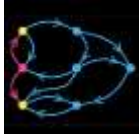
- The GraphBLAS Forum Overview
  - C API Specification “Report”
- **Short Topics**
  - **Trevor Steil/Roger Pearce:** ***Distributed GraphBLAS***
- Invited Speaker: Michel Pelletier
  - *pygraphblas: a Python API for GraphBLAS*
- Open discussion

Agenda



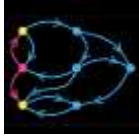
- The GraphBLAS Forum Overview
  - C API Specification “Report”
- Short Topics
  - Trevor Steil/Roger Pearce: *Distributed GraphBLAS*
- **Invited Speaker: Michel Pelletier**
  - ***pygraphblas: a Python API for GraphBLAS***
- Open discussion

Agenda



- The GraphBLAS Forum Overview
  - C API Specification “Report”
- Short Topics
  - Trevor Steil/Roger Pearce: *Distributed GraphBLAS*
- Invited Speaker: Michel Pelletier
  - *pygraphblas: a Python API for GraphBLAS*
- **Open discussion**

Open Discussion

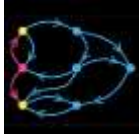


Possible future directions:

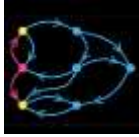
- What language APIs do you want?
- What is needed to make it easier/more efficient to interface to third party libraries?
- Parallel GraphBLAS with a formal memory model for execution with multiple threads.
- How does the API need to change to move into a distributed/MPI environment?

<insert your topic here>

Backups



C API Specification Overview



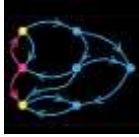
- Matrices and vectors as opaque types
  - Specify domain and size of dimensions (immutable)
  - Internal representation is hidden from user
  - They are not instantiated to be specifically sparse or dense.

```
GrB_Matrix graph;
GrB_Matrix_new(&graph, GrB_BOOL, 10, 10);       // 10x10 matrix of Booleans
```

- Semirings used to describe operations (math document)
  - Some C functions can take monoids or arbitrary functions

```
GrB_Semiring arithmeticI32;          // <int32_t, float, float, +, *, 0, 1>
GrB_Semiring_new(&arithmeticI32FF,
                 GrB_INT32, GrB_FLOAT, GrB_FLOAT,
                 GrB_PLUS_FLOAT, GrB_TIMES_I32FF, 0, 1);
```

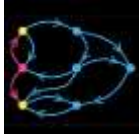
GraphBLAS Base Operations



Operation	Math	Out	Inputs
mxm	$\mathbf{C}\langle\neg\mathbf{M},z\rangle = \mathbf{C} \odot (\mathbf{A}^{\mathsf{T}} \oplus.\otimes \mathbf{B}^{\mathsf{T}})$	**C**	¬, **M**, z, ⊙, **A**, **T**, ⊕.⊗, **B**, **T**
mxv (vxm)	$\mathbf{c}\langle\neg\mathbf{m},z\rangle = \mathbf{c} \odot (\mathbf{A}^{\mathsf{T}} \oplus.\otimes \mathbf{b})$	**c**	¬, **m**, z, ⊙, **A**, **T**, ⊕.⊗, **b**
eWiseMult	$\mathbf{C}\langle\neg\mathbf{M},z\rangle = \mathbf{C} \odot (\mathbf{A}^{\mathsf{T}} \otimes \mathbf{B}^{\mathsf{T}})$	**C**	¬, **M**, z, ⊙, **A**, **T**, ⊗, **B**, **T**
eWiseAdd	$\mathbf{C}\langle\neg\mathbf{M},z\rangle = \mathbf{C} \odot (\mathbf{A}^{\mathsf{T}} \oplus \mathbf{B}^{\mathsf{T}})$	**C**	¬, **M**, z, ⊙, **A**, **T**, ⊕, **B**, **T**
reduce (row)	$\mathbf{c}\langle\neg\mathbf{m},z\rangle = \mathbf{c} \odot [\oplus_j \mathbf{A}^{\mathsf{T}}(:,j)]$	**c**	¬, **m**, z, ⊙, **A**, **T**, ⊕
apply	$\mathbf{C}\langle\neg\mathbf{M},z\rangle = \mathbf{C} \odot f(\mathbf{A}^{\mathsf{T}})$	**C**	¬, **M**, z, ⊙, **A**, **T**, *f*
transpose	$\mathbf{C}\langle\neg\mathbf{M},z\rangle = \mathbf{C} \odot \mathbf{A}^{\mathsf{T}}$	**C**	¬, **M**, z, ⊙, **A** (**T**)
extract	$\mathbf{C}\langle\neg\mathbf{M},z\rangle = \mathbf{C} \odot \mathbf{A}^{\mathsf{T}}(\mathbf{i},\mathbf{j})$	**C**	¬, **M**, z, ⊙, **A**, **T**, **i**, **j**
assign	$\mathbf{C}\langle\neg\mathbf{M},z\rangle(\mathbf{i},\mathbf{j}) = \mathbf{C}(\mathbf{i},\mathbf{j}) \odot \mathbf{A}^{\mathsf{T}}$	**C**	¬, **M**, z, ⊙, **A**, **T**, **i**, **j**
build (meth.)	$\mathbf{C} = \mathbb{S}^{m \times n}(\mathbf{i},\mathbf{j},\mathbf{v},\odot)$	**C**	⊙, m, n, **v**, **i**, **j**
extractTuples (meth.)	$(\mathbf{i},\mathbf{j},\mathbf{v}) = \mathbf{A}$	**i**,**j**,**v**	**A**

Notation: **i**,**j** – index arrays, **v** – scalar array, **m** – 1D mask, **bold-lower** – vector (column), **M** – 2D mask, **bold-upper** – matrix, **T** – transpose, ¬ - structural complement, **z** – replace ⊙ accumulate monoid/binary function, ⊕.⊗ semiring, **blue** – optional parameters, **red** – optional modifiers (using Descriptors)

GraphBLAS Signatures: mxm



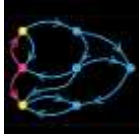
$$C\langle \neg M, z \rangle = C \odot (A^T \oplus.\otimes B^T)$$

```
GrB_info GrB_mxm(GrB_Matrix                C,      // destination
                 const GrB_Matrix          Mask,
                 const GrB_BinaryFunction  accum,
                 const GrB_Semiring        op,
                 const GrB_Matrix          A,
                 const GrB_Matrix          B
                 const Descriptor          desc);
```

Common Elements:

- GrB_ “namespace”
- Destination (C) is first
- Mask matrix and accumulation function are next (if supported)
  - Pass GrB_NULL if not needed.
- Descriptor is always last (pass GrB_NULL if not needed)

GraphBLAS Signatures: mxm



$$C\langle \neg M, z \rangle = C \odot (A^T \oplus.\otimes B^T)$$

```
GrB_info GrB_mxm(GrB_Matrix               C,
                 const GrB_Matrix         Mask,
                 const GrB_BinaryFunction accum,
                 const GrB_Semiring       op,
                 const GrB_Matrix         A,
                 const GrB_Matrix         B
                 const Descriptor         desc);
```

API Error Return Values:

- GrB_NULL_POINTER
- GrB_INVALID_VALUE
- GrB_INVALID_INDEX
- GrB_DOMAIN_MISMATCH
- GrB_DIMENSION_MISMATCH
- GrB_OUTPUT_NOT_EMPTY
- GrB_NO_VALUE

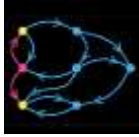
Execution Error Return Values:

- GrB_OUT_OF_MEMORY
- GrB_INDEX_OUT_OF_BOUNDS
- GrB_PANIC

Otherwise:

- GrB_SUCCESS

GraphBLAS Signatures: mxm



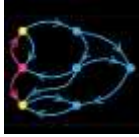
$$C\langle \neg M, z \rangle = C \odot (A^T \oplus.\otimes B^T)$$

```
GrB_info GrB_mxm(GrB_Matrix                C,      // GrB_OUTP
                 const GrB_Matrix          Mask,   // GrB_MASK
                 const GrB_BinaryFunction  accum,
                 const GrB_Semiring        op,
                 const GrB_Matrix          A,      // GrB_ARG0
                 const GrB_Matrix          B,      // GrB_ARG1
                 const Descriptor          desc);
```

Descriptors (modifiers to various arguments):

- GrB_SCMP ¬ – structural complement (MASK only).
- GrB_TRAN ^{T} – transpose the input matrix arguments (ARG0, ARG1)
- GrB_REPLACE z – replace all elements in the output argument (OUTP)

Counting Triangles (once) with GraphBLAS



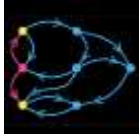
- Given:
  - Undirected graph G = {V, E}
  - L: boolean, lower-triangular portion of adjacency matrix
- **# triangles = $\|L \otimes (L \oplus.\otimes L^T)\|_1$**
  - Semiring can be Plus-AND or Plus-Times
  - Element-wise multiplication is equivalent to a mask operation

```
uint64_t triangle_count(GrB_Matrix L)   // L: NxN, lower-triangular, boolean
{
    GrB_Index N;
    GrB_Matrix_nrows(&N, L);
    GrB_matrix C;
    GrB_Matrix_new(&C, GrB_UINT64, N, N);

    GrB_mxm(C, L, GrB_NULL, GrB_UInt64AddMul, L, L, GrB_TB); // C<L> = L * L^T

    uint64_t count;
    GrB_reduce(&count, GrB_NULL, GrB_UInt64Add, C, GrB_NULL);// 1-norm of C
    return count;
}
```

Variants (assign, as an example)



- "Standard" variant (math document)

$$\mathbf{C}\langle\neg\mathbf{M},z\rangle(\mathbf{i},\mathbf{j}) \oplus= \mathbf{A}^{\mathsf{T}} \qquad \mathbf{c}\langle\neg\mathbf{m},z\rangle(\mathbf{i}) \oplus= \mathbf{a}$$

- "Column and row" variants (on matrices only)

row: $\mathbf{C}\langle\neg\mathbf{m}^{\mathsf{T}},z\rangle(i, \mathbf{j}) \oplus= \mathbf{a}^{\mathsf{T}}$ col: $\mathbf{C}\langle\neg\mathbf{m},z\rangle(\mathbf{i}, j) \oplus= \mathbf{a}$

- "Constant" variant

$$\mathbf{C}\langle\neg\mathbf{M},z\rangle(\mathbf{i},\mathbf{j}) \oplus= s \qquad \mathbf{c}\langle\neg\mathbf{m},z\rangle(\mathbf{i}) \oplus= s$$