Design and Implementation of the GraphBLAS Template Library (GBTL)

Scott McMillan, Samantha Misurda

Marcin Zalewski, Peter Zhang, Andrew Lumsdaine

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

© 2016 Carnegie Mellon University

Distribution Statement A: Approved for Public Release;
Distribution is Unlimited

Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Department of Defense.

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.

DM-0003820

What is this talk about?

- GraphBLAS
  - an emerging paradigm for graph computation
  - programs new graph algorithms in a highly abstract *language of linear algebra*.
  - executes in a wide variety of programming environments
- Our implementation of GraphBLAS
  - Graph BLAS Template Library (GBTL)
  - High-level C++ *frontend* (some features still in progress)
  - Algorithms written in terms of the API
  - Released at: https://github.com/cmu-sei/gbtl

Graph BLAS Forum
http://www.graphblas.org

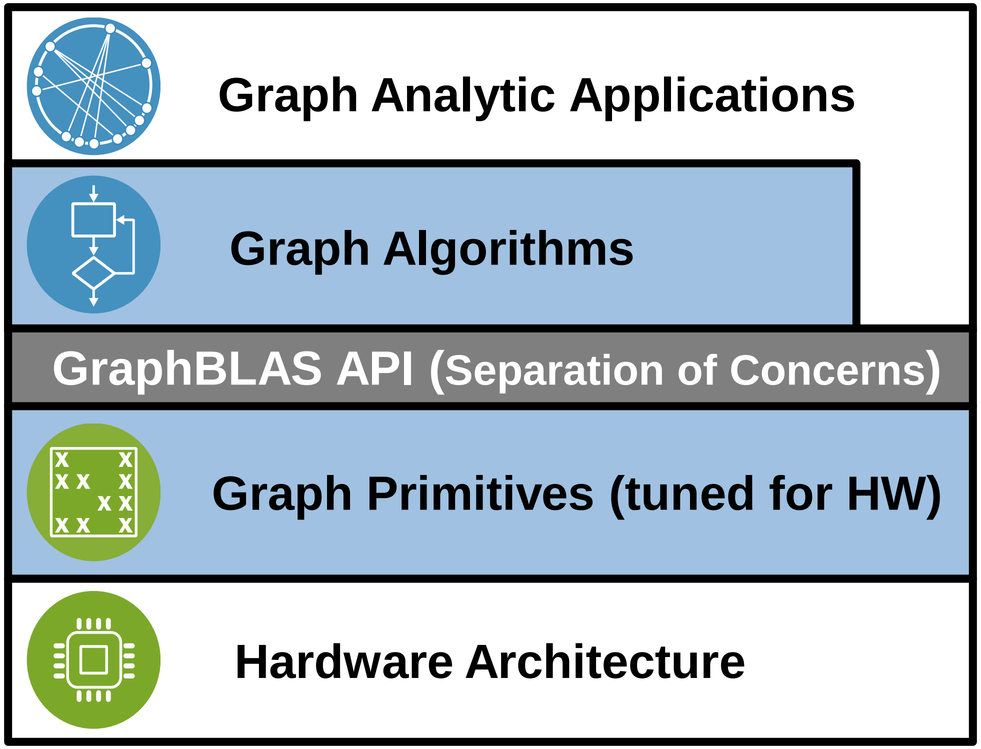
Software Engineering Institute
Carnegie Mellon University

CREST
Indiana University

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Design Goals

- Separation of concerns:
  - Algorithm development
  - Hardware tuning
- Low overhead
- Support patterns for scalable, high-performance computing



- Templated C++ implementation
  - Allows for generic programming and template metaprogramming
  - Allows generic semantic checks (compile time)
- Similarity to the C API specification (under development)
- Easy to use

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Contents

© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Contents

- **Operations Overview**
- Object Design
- Operations – Function Signatures
- Example Algorithm
- Summary and Future Work

© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

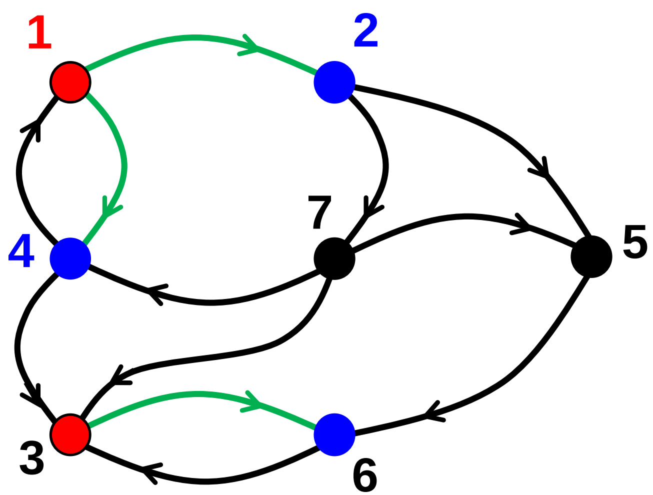
Background: GraphBLAS Operations

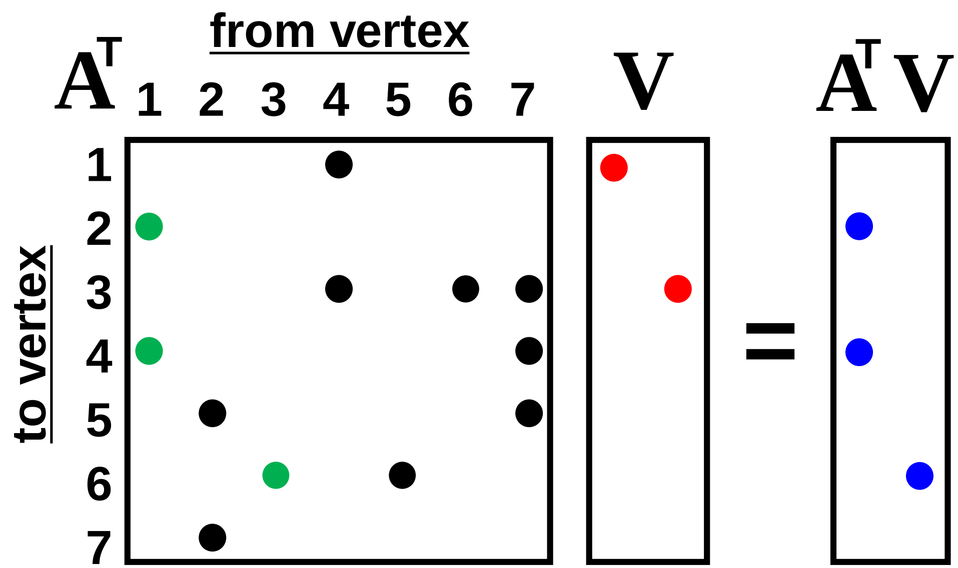
Operation	Description	Old Name
MxM, MxV, VxM	Perform matrix *multiplication* (e.g., BFS traversal)	SpGEMM
EwiseAdd, EwiseMult	Element-wise *addition* and *multiplication* of matrices (e.g., graph union, intersection)	SpEWiseX
Extract	Extract a sub-matrix from a larger matrix (e.g., sub-graph selection)	SpRef
Assign	Assign to a sub-matrix of a larger matrix (e.g., sub-graph assignment)	SpAsgn
Apply	Apply *unary function* to each element of matrix (e.g., edge weight modification)	Apply
Reduce	*Reduce* along columns or rows of matrices (e.g., vertex degree)	Reduce
Transpose	Swaps the rows and columns of a matrix (e.g., reverse directed edges)	Transpose
BuildMatrix	Build a matrix from row, column, value tuples	Sparse
ExtractTuples	Extract the row, column, value tuples from a matrix	Find

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Background: Matrix Multiply (MxM)

$$C = A \oplus . \otimes B$$





- The **Semiring** ($\oplus . \otimes$) determines how this computation is carried out.
- Consists of two **Monoids** (**Binary Function** + identity)
  - $\otimes$, e.g., (multiply, 1)
  - $\oplus$, e.g., (add, 0)
- These can be user defined, not adhering strictly to Semiring properties

$$c_{i,j} = \sum_{l=1}^{k} a_{i,l} \times b_{l,j}$$

© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GraphBLAS Operation: MxM example

$$\mathbf{C} = \mathbf{A} \oplus . \otimes \mathbf{B}$$

- Required:
  - Two input matrices: A_{MxK} and B_{KxN}
  - One output matrix: C_{MxN}
  - One semiring: $\oplus . \otimes$

© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GraphBLAS Operation: MxM example

$$C \oplus= A^T \oplus.\otimes B^T$$

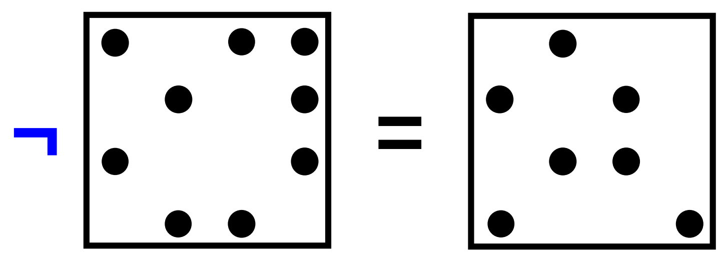
- Required:
  - Two input matrices: A^T_{MxK} and B^T_{KxN}
  - One output matrix: C_{MxN}
  - One semiring: $\oplus.\otimes$
- Optional:
  - Matrix Transpose – only necessary on inputs
  - Accumulation – binary function, can be different from Semiring's $\oplus$.

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GraphBLAS Operation: MxM example

$$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{\mathbf{T}} \oplus.\otimes\ \mathbf{B}^{\mathbf{T}}$$

- Required:
  - Two input matrices: A^T_{MxK} and B^T_{KxN}
  - One output matrix: C_{MxN}
  - One semiring: $\oplus.\otimes$
- Optional:
  - Matrix Transpose – only necessary on inputs
  - Accumulation – binary function, can be different from Semiring's $\oplus$.
  - Output Mask: M_{MxN} – specifies which locations in C can be modified
  - Mask Complement
    - Invert the structure of stored values (sparse)
    - Invert boolean values (dense)



© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GraphBLAS Operation: MxM example

$$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{\mathbf{T}} \oplus.\otimes \mathbf{B}^{\mathbf{T}}$$

- Required:
  - Two input matrices: A^T_{MxK} and B^T_{KxN}
  - One output matrix: C_{MxN}
  - One semiring: $\oplus.\otimes$
- Optional:
  - Matrix Transpose – only necessary on inputs
  - Accumulation – binary function, can be different from Semiring's $\oplus$.
  - Output Mask: M_{MxN} – specifies which locations in C can be modified
  - Mask Complement
    - Invert the structure of stored locations (sparse)
    - Invert boolean values (dense)

blue – optional parameters
red – optional modifiers

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Operations

Operation	Math	Out	Inputs
MxM	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{\mathbf{T}} \oplus.\otimes \mathbf{B}^{\mathbf{T}}$	$\mathbf{C}$	$\neg$, $\mathbf{M}$, $\oplus$, $\mathbf{A}$, $\mathbf{T}$, $\oplus.\otimes$, $\mathbf{B}$, $\mathbf{T}$
MxV (VxM)	$\mathbf{c}(\neg\mathbf{m}) \oplus= \mathbf{A}^{\mathbf{T}} \oplus.\otimes \mathbf{b}$	$\mathbf{c}$	$\neg$, $\mathbf{m}$, $\oplus$, $\mathbf{A}$, $\mathbf{T}$, $\oplus.\otimes$, $\mathbf{b}$
EwiseMult	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{\mathbf{T}} \otimes \mathbf{B}^{\mathbf{T}}$	$\mathbf{C}$	$\neg$, $\mathbf{M}$, $\oplus$, $\mathbf{A}$, $\mathbf{T}$, $\otimes$, $\mathbf{B}$, $\mathbf{T}$
EwiseAdd	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{\mathbf{T}} \oplus \mathbf{B}^{\mathbf{T}}$	$\mathbf{C}$	$\neg$, $\mathbf{M}$, $\oplus$, $\mathbf{A}$, $\mathbf{T}$, $\oplus$, $\mathbf{B}$, $\mathbf{T}$
Reduce (row)	$\mathbf{c}(\neg\mathbf{m}) \oplus= \oplus_j \mathbf{A}^{\mathbf{T}}(:,j)$	$\mathbf{c}$	$\neg$, $\mathbf{m}$, $\oplus$, $\mathbf{A}$, $\mathbf{T}$, $\oplus$
Apply	$\mathbf{C}(\neg\mathbf{M}) \oplus= f(\mathbf{A}^{\mathbf{T}})$	$\mathbf{C}$	$\neg$, $\mathbf{M}$, $\oplus$, $\mathbf{A}$, $\mathbf{T}$, f
Transpose	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{\mathbf{T}}$	$\mathbf{C}$	$\neg$, $\mathbf{M}$, $\oplus$, $\mathbf{A}$ $(\mathbf{T})$
Extract	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{\mathbf{T}}(\mathbf{i},\mathbf{j})$	$\mathbf{C}$	$\neg$, $\mathbf{M}$, $\oplus$, $\mathbf{A}$, $\mathbf{T}$, $\mathbf{i}$, $\mathbf{j}$
Assign	$\mathbf{C}(\neg\mathbf{M})\ (\mathbf{i},\mathbf{j}) \oplus= \mathbf{A}^{\mathbf{T}}$	$\mathbf{C}$	$\neg$, $\mathbf{M}$, $\oplus$, $\mathbf{A}$, $\mathbf{T}$, $\mathbf{i}$, $\mathbf{j}$
BuildMatrix	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbb{S}^{mxn}(\mathbf{i},\mathbf{j},\mathbf{v},\oplus)$	$\mathbf{C}$	$\neg$, $\mathbf{M}$, $\oplus$, $\oplus$, m, n, $\mathbf{i}$, $\mathbf{j}$, $\mathbf{v}$
ExtractTuples	$(\mathbf{i},\mathbf{j},\mathbf{v}) = \mathbf{A}(\neg\mathbf{M})$	$\mathbf{i},\mathbf{j},\mathbf{v}$	$\neg$, $\mathbf{M}$, $\mathbf{A}$

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, **¬** - structural complement,
$\oplus$ monoid/binary function, $\oplus.\otimes$ semiring,
blue – optional parameters, **red** – optional modifiers

© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Contents

© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Objects

- Index and IndexArray
- Matrices and vectors
  - Structure and values
  - Sparse (and dense, but not today)
- Modifiers
  - Structural Complement (and Masks)
  - Transpose
- Mathematical operations
  - Binary functions vs. Monoids
  - Semirings
  - Accumulation (just another binary function)

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

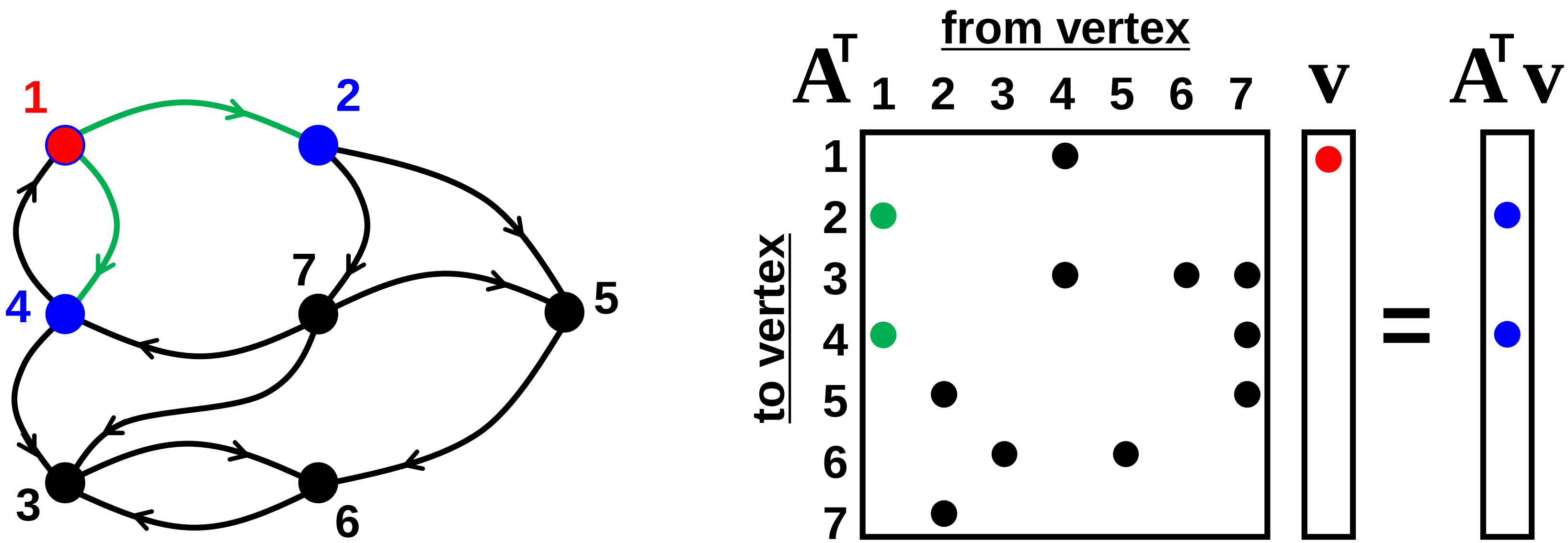
Indices

- Index – a value used to locate a position in vectors or matrices (pair of indices).
- IndexArray – multiple values used to locate multiple positions

```
// Some typedefs to give GraphBLAS names to some concepts
typedef uint64_t                    IndexType;
typedef std::vector<IndexType>  IndexArrayType;
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Sparse Matrices and Vectors



Matrices (e.g. adjacency or incidence)

- Adjacency matrices:
  - “Stored values” – edge
  - “Structural zeros” – no edge/storage
- Two index arrays for positions (structure)
- Scalar array for edge attribute (values)

Vectors (e.g. wavefronts)

- Current: (m x 1) or (1 x n) matrices
- In progress: a different object
  - More efficient storage managing only one index dimension
  - Is an implicit orientation necessary (column in mathematics)?

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Achieving Opaque Matrices

- Frontend: Interface here, type/storage details are opaque

```
// Variadic template parameters provide hints for backend matrix type
template <typename ScalarT, typename... TagsT>
class Matrix
{
public:
    // construct an empty matrix with immutable dimensions
    Matrix(IndexType num_rows, IndexType num_cols);

    // Interface, forwards calls to backend
    IndexType get_nnz() const    { return m_matrix.get_nnz(); }
    IndexType get_nrows() const  { return m_matrix.get_nrows(); }
    IndexType get_ncols() const  { return m_matrix.get_ncols(); }
    //...

private:
    detail::matrix_type_generator::result<     // template metaprogramming
        ScalarT,                               // to select backend type
        detail::SparsenessCategoryTag,         // at compile time.
        detail::DirectednessCategoryTag,
        TagsT...>::type
            m_mat;

    friend void template<...> mxm(...);      // all frontend ops are friends
};
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Achieving Opaque Matrices (what the user writes)

- Frontend `Matrix` construction:
  - User provides hints (tags) through template parameter packing,
  - backend can make decisions based on hints.

```
Matrix<double, DenseMatrixTag, DirectedMatrixTag>
    matrix(num_rows, num_columns);
```

- “Sparseness” and “directedness” hints currently implemented.
  - Future support could include layout hints like “fast-column access”
- Backend `Matrix` classes:
  - Specialized for hardware and implementation
  - Does not have to support or adhere to hints

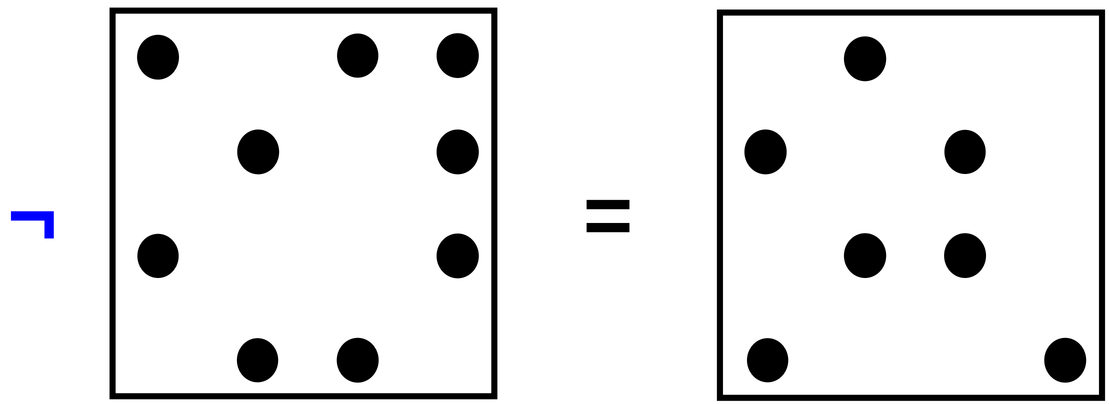
© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Modifiers

- For two cases
  - Mask complement
  - Matrix transpose
- Purpose:
  - Reduce the number of parameters in operation signatures.
  - Improve readability/usability
- Lightweight wrappers for certain input arguments
- Non-mutating
  - Does not proscribe creation of temporaries

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Masks and Structural Complement (WIP)



- Current:
  - Masks also implemented as matrices
  - Values = don’t care
  - Complement changes structure (structural zeros become “1”)
- Future: a different object
  - Potential: more efficient storage with no scalar values

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Complement Modifier (WIP)

- Currently called NegateView (bad name! – will change)
- Templated operations will accept either Matrix or Mask.

```
template <typename MaskT>
class NegateView
{
public:
    NegateView(MaskT const &mask) : m_mask(mask) {}
    IndexType get_nnz() const {
        return m_mask.get_nrows()*m_mask.get_ncols() - m_mask.get_nnz(); }
    //...
private:
    MaskT const &m_mask;
};
```

- Corresponding operation returns a wrapped (*backend*) mask.

```
template<typename MaskT>
NegateView<MaskT> negate(MaskT const &m) {
    return NegateView<MaskT>(backend::negate(m.m_mat));
}
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Transpose Modifier

- Effectively swaps row and column access.

```
template <typename MatrixT>
class TransposeView
{
public:
    TransposeView(MatrixT const &matrix) : m_matrix(matrix) {}

    IndexType get_nrows() const { return m_matrix.get_ncols(); }
    IndexType get_ncols() const { return m_matrix.get_nrows(); }
    // ...
private:
    MatrixT const &m_matrix;
};
```

- Corresponding operation returns a wrapped (*backend*) matrix:

```
template<typename MatrixT>
TransposeView<MatrixT> transpose(MatrixT const &A) {
    return TransposeView<MatrixT>(backend::transpose(A.mat));
}
```


© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Mathematical Operations: $\oplus$, $\otimes$, $\oplus$=, $\oplus.\otimes$

- Binary functions on multiple domains: D1 x D2 → D3

```
template <typename ResultT, typename Arg1T=ResultT, typename Arg2T=ResultT>
struct ArithmeticMultiplyFunc
{
    ResultT operator()(Arg1T const &lhs, Arg2T const &rhs) const {
        return static_cast<ResultT>(lhs) * static_cast<ResultT>(rhs); }
};
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Mathematical Operations: $\oplus$, $\otimes$, $\oplus$=, $\oplus.\otimes$

- Binary functions on multiple domains: D1 x D2 → D3
- Monoids (if needed), include the identity for the operation

```
template <typename ResultT, typename Arg1T=ResultT, typename Arg2T=ResultT>
struct ArithmeticAddMonoid
{
    ResultT operator()(Arg1T const &lhs, Arg2T const &rhs) const {
        return static_cast<ResultT>(lhs) + static_cast<ResultT>(rhs);

    ResultT identity() { return static_cast<ResultT>(0) };
};
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Mathematical Operations: $\oplus$, $\otimes$, $\oplus$=, $\oplus.\otimes$

- Binary functions on multiple domains: D1 x D2 → D3
- Monoids (if needed), include the identity for the operation
- Semirings for matrix multiply
  - Two binary functions (and identities, if needed)
  - Addition monoid is defined on one domain: D3 x D3 → D3

```
template <typename ResultT, typename Arg1T=ResultT, typename Arg2T=ResultT>
struct ArithmeticSemiring
{
    // Additive Monoid
    ResultT zero() const { return static_cast<ResultT>(0); }
    ResultT add(ResultT const &lhs, ResultT const &rhs) const {
        return (lhs + rhs); }

    // Multiplicative Monoid
    ResultT one() const { return static_cast<ResultT>(1); }
    ResultT mult(Arg1T const &lhs, Arg2T const &rhs) const {
        return static_cast<ResultT>(lhs) * static_cast<ResultT>(rhs); }
};
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Accumulation: ⊕, ⊗, ⊕=, ⊕.⊗

- Select one of two binary functions

```
// No accumulation (select rhs), used as the default parameter value
template <typename ResultT>
struct Assign
{
    ResultT operator()(ResultT lhs, ResultT rhs) { return rhs; }
};



// Accumulation: using arithmetic addition as the default
template <typename ResultT,
          typename BinaryFuncT = ArithmeticAddFunc<ResultT> >
struct Accum
{
    ResultT operator()(ResultT lhs, ResultT rhs) {
               return BinaryFuncT()(lhs, rhs);
    }
};
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Contents

- Design Overview – Separation of Concerns
- Operations Overview
- Object Design
- **Operations – Function Signatures**
- Example Algorithm
- Summary and Future Work

© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

$$C(\neg M) \oplus= A^T \oplus.\otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string(“mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```


© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

$$C(\neg M) \oplus= A^T \oplus . \otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string("mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```


© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

$$C(\neg M) \oplus= A^T \oplus.\otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string(“mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

$$C(\neg M) \oplus= A^T \oplus.\otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string(“mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

$$C(\neg M) \oplus= A^T \oplus.\otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string("mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```


© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

$$C(\neg M) \oplus= A^T \oplus.\otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string("mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

$$C(\neg M) \oplus = A^T \oplus . \otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string(“mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

Modifiers appear at the call site.

$$C(\neg M) \oplus= A^{T} \oplus.\otimes B^{T}$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string(“mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```


© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

$$C(\neg M) \oplus= A^T \oplus.\otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string(“mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```

Simple code like dimension checks in the frontend

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: MxM

$$C(\neg M) \oplus= A^T \oplus.\otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename SemiringT,
         typename AccumT    = math::Assign<typename CMatrixT::ScalarType> >
void mxmMasked(AMatrixT const &A,
               BMatrixT const &B,
               CMatrixT       &C,
               MaskT    const &M,
               SemiringT       sr,
               AccumT          accum = AccumT())
{
    same_dimension_check(C, M, std::string(“mxmMasked"));
    multiply_dimension_check(A, B, C, std::string("mxmMasked"));
    backend::mxmMasked(A.m_mat, B.m_mat, C.m_mat, M.m_mat, sr, accum);
}
```

Forward work
to the backend

© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: EwiseMult

$$C(\neg M) \oplus= A^T \otimes B^T$$

```
template<typename AMatrixT,
         typename BMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename BinaryFunctionT,
         typename AccumT = Assign<typename CMatrixT::ScalarType> >
void ewiseaddMasked(AMatrixT const  &A,
                    BMatrixT const  &B,
                    CMatrixT        &C,
                    MaskT    const  &M,
                    BinaryFunctionT  func = BinaryFunctionT(),
                    AccumT           accum = AccumT())
{
    same_dimension_check(A, B, std::string("ewiseadd"));
    same_dimension_check(A, C, std::string("ewiseadd"));
    same_dimension_check(C, M, std::string("ewiseadd"));
    backend::ewiseadd(A.m_mat, B.m_mat, C.m_mat, M.m_mat, func, accum);
}
```


© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: Apply

$$C(\neg M) \oplus= f(A^T)$$

```
template<typename AMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename UnaryFunctionT,
         typename AccumT = Assign<typename CMatrixT::ScalarType> >
void applyMasked(AMatrixT const  &A,
                 CMatrixT        &C,
                 MaskT           &M,
                 UnaryFunctionT   func,
                 AccumT           accum = AccumT())
{
    same_dimension_check(A, C, std::string("apply"));
    same_dimension_check(C, M, std::string("apply"));
    backend::applyMasked(a.m_mat, c.m_mat, M.m_mat, func, accum);
}
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: Reduce (rows)

$$c(\neg m) \oplus= \oplus_j A^T(:,j)$$

```
template<typename AMatrixT,
         typename CVectorT,
         typename MaskT,
         typename MonoidT = ArithmeticAddMonoid<typename AMatrixT::ScalarType,
                                               typename BMatrixT::ScalarType,
                                               typename CMatrixT::ScalarType >,
         typename AccumT = Assign<typename CMatrixT::ScalarType> >
void rowReduceMasked(AMatrixT const &A,
                     CVectorT       &c,
                     MaskT          &m,
                     MonoidT         monoid = MonoidT(),
                     AccumT          accum = AccumT())
{
    same_dimension_check(c, m, std::string(“rowReduceMasked"));
    if (A.get_nrows() != c.get_nrows() || c.get_ncols() != 1) {
        throw graphblas::DimensionException("rowReduceMasked dimension error");
    }
    backend::rowReduceMasked(A.m_mat, c.m_mat, m.m_mat, moniod, accum);
}
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Signatures: Extract

$$C(\neg M) \oplus= A^T(i,j)$$

```
template<typename AMatrixT,
         typename CMatrixT,
         typename MaskT,
         typename AccumT = Assign<typename CMatrixT::ScalarType> >
void extractMasked(AMatrixT       const &A,
                   IndexArrayType const &i,
                   IndexArrayType const &j,
                   CMatrixT             &C,
                   MaskT                &M,
                   AccumT                accum = AccumT())
{
    same_dimension_check(C, M, std::string("extract"));
    assign_extract_dimension_check(A, C, i.begin(),j.begin());
    backend::extract(A.m_mat, i, j, C.m_mat, M.m_mat, accum);
}
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Contents

- Operations Overview
- Object Design
- Operations – Function Signatures
- **Example Algorithm**
- Summary and Future Work

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Algorithm: Multi-front, Level BFS v0

```
template <typename MatrixT>              // MatrixT scalar type: Integer,..
void bfs_level(MatrixT const &graph,     // MxM adjacency matrix {0,1}
               MatrixT        wavefront, // MxR columns init'd with roots {0,1}
               MatrixT       &levels)    // MxR level results for each BFS
{
   using T = typename MatrixT::ScalarType;

   IndexType rows = wavefront.get_nrows();
   IndexType cols = wavefront.get_ncols();
   MatrixT not_visited(rows, cols);
   T current_level = 0;

   while (wavefront.get_nnz() > 0) {
      // Increment and apply current level to all newly visited nodes.
      arithmetic_n<T, Times<T> > apply_level(++current_level);
      apply(wavefront, levels, apply_level, Accum<T>());

      mxm(wavefront, graph, wavefront,
          IntBooleanSemiring<T>());

      // Remove previously visited vertices from the wavefront
      apply(levels, not_visited, IsZero<T>());
      ewisemult(not_visited, wavefront, wavefront, AndFunc<T>());
   }
}
```

$++c$

$\mathbf{L} += c\,\mathbf{W}$

$\mathbf{W} = \mathbf{W} \,|.\&\, \mathbf{A}$

$\overline{\mathbf{V}} = (\mathbf{L} == 0)$

$\mathbf{W} = \overline{\mathbf{V}} \,.\&\, \mathbf{W}$

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Algorithm: Multi-front, Level BFS v1

```
template <typename MatrixT>                // MatrixT scalar type: Integer,..
void bfs_level(MatrixT const &graph,       // MxM adjacency matrix {0,1}
               MatrixT        wavefront,   // MxR columns init'd with roots {0,1}
               MatrixT       &levels)      // MxR level results for each BFS
{
   using T = typename MatrixT::ScalarType;

   IndexType rows = wavefront.get_nrows();
   IndexType cols = wavefront.get_ncols();
   MatrixT not_visited(rows, cols);
   T current_level = 0;

   while (wavefront.get_nnz() > 0) {
      // Increment and apply current level to all newly visited nodes.
      arithmetic_n<T, Times<T> > apply_level(++current_level);
      apply(wavefront, levels, apply_level, Accum<T>());

      mxm(transpose(graph), wavefront, wavefront,
          IntBooleanSemiring<T>());

      // Remove previously visited vertices from the wavefront
      apply(levels, not_visited, IsZero<T>());
      ewisemult(not_visited, wavefront, wavefront, AndFunc<T>());
   }
}
```

$++c$

$\mathbf{L} += c\,\mathbf{W}$

$\mathbf{W} = \mathbf{A}^{\mathrm{T}} \,|.\&\, \mathbf{W}$

$\overline{\mathbf{V}} = (\mathbf{L} == 0)$

$\mathbf{W} = \overline{\mathbf{V}} \,.\&\, \mathbf{W}$

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Algorithm: Multi-front, Level BFS v2

```
template <typename MatrixT>                    // MatrixT scalar type: Integer,..
void bfs_level(MatrixT const &graph,           // MxM adjacency matrix {0,1}
               MatrixT        wavefront,       // MxR columns init'd with roots {0,1}
               MatrixT       &levels)          // MxR level results for each BFS
{
   using T = typename MatrixT::ScalarType;

   IndexType rows = wavefront.get_nrows();
   IndexType cols = wavefront.get_ncols();
   MatrixT not_visited(rows, cols);
   T current_level = 0;

   while (wavefront.get_nnz() > 0) {
      // Increment and apply current level to all newly visited nodes.
      arithmetic_n<T, Times<T> > apply_level(++current_level);
      apply(wavefront, levels, apply_level, Accum<T>());

      mxmMasked(transpose(graph), wavefront, wavefront,
                negate(levels), IntBooleanSemiring<T>());

      // Remove previously visited vertices from the wavefront
      apply(levels, not_visited, IsZero<T>());
      ewisemult(not_visited, wavefront, wavefront, AndFunc<T>());
   }
}
```

$$W(\neg L) = A^T \,|.\&\, W$$

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Algorithms: Maximal Independent Set

```
template <typename MatrixT>
void mis(MatrixT const &graph,              // NxN
         MatrixT       &independent_set,    // Nx1: !0 indicates node ‘in’ set.
         double         seed = 0)
{
    graphblas::IndexType rows, cols, r, c;
    graph.get_shape(rows, cols);
    // check dimensions...

    generator.seed(seed);  // for SetRandom functor (not shown)

    typedef Matrix<double, /*Tags...*/> RealMatrix;
    using T = typename MatrixT::ScalarType;

    // This will hold the set (non-zero) implies part of the set
    RealMatrix neighbor_max(rows, 1);
    RealMatrix new_members(rows, 1);
    RealMatrix new_neighbors(rows, 1);
    RealMatrix prob(rows, 1);

    RealMatrix candidates(rows, 1);  fill(candidates, 1.0);
    RealMatrix degrees(rows, 1);     fill(degrees, 1.0);

    // Compute degree of each node, add 1 to prevent divide by zero
    rowReduce(graph, degrees, ArithmeticAddMonoid<double, T, T>());
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

GBTL Algorithms: Maximal Independent Set

```
    while (candidates.get_nnz() > 0)
    {
        // Assign random values (scaled by degree) to all non-zero candidate elements.
        // Ensures that any ties that occur between neighbors will eventually be
        //   broken, and higher degree nodes are more likely selected.
        ewisemult(candidates, degrees, prob, SetRandom());

        // find the neighbor of each source node with the max random number
        mxm(graph, prob, neighbor_max, MaxSelect2ndSemiring<double>());

        // Select source node if its probability is > neighbor_max
        ewiseadd(prob, neighbor_max, new_members, GreaterThan<double>());

        // Add new members to independent set.
        ewiseadd(independent_set, new_members, independent_set, OrFn<double>());

        // Zero out candidates of new_members selected for independent set
        ewisemult(negate(new_members), candidates, candidates);

        if (candidates.get_nnz() == 0) { break; }   // Early exit

        // Neighbors of new members can also be removed
        mxm(graph, new_members, new_neighbors, MaxSelect2ndSemiring<double>());

        // Zero out candidates of new member neighbors
        ewisemult(negate(new_neighbors), candidates, candidates);
    }
}
```


© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Summary and Future Work

- GraphBLAS Template Library
  - Separation of concerns: hardware tuning vs. algorithm design
  - C++ templates
    - Expressive syntax in algorithm development (we could do more).
    - Low overhead (first session)
  - Similar to C API Specification (where possible/reasonable)

- Current and Future Work
  - Tracking C API Specification decisions
    - Multiple domains in monoids and semirings
    - Mask and structural complement
    - Variants of basic operations (reduce to scalar, assign/extract columns)
  - More algorithms
  - GraphBLAS C++ API Specification and Reference Implementation

© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Questions?

© 2016 Carnegie Mellon University

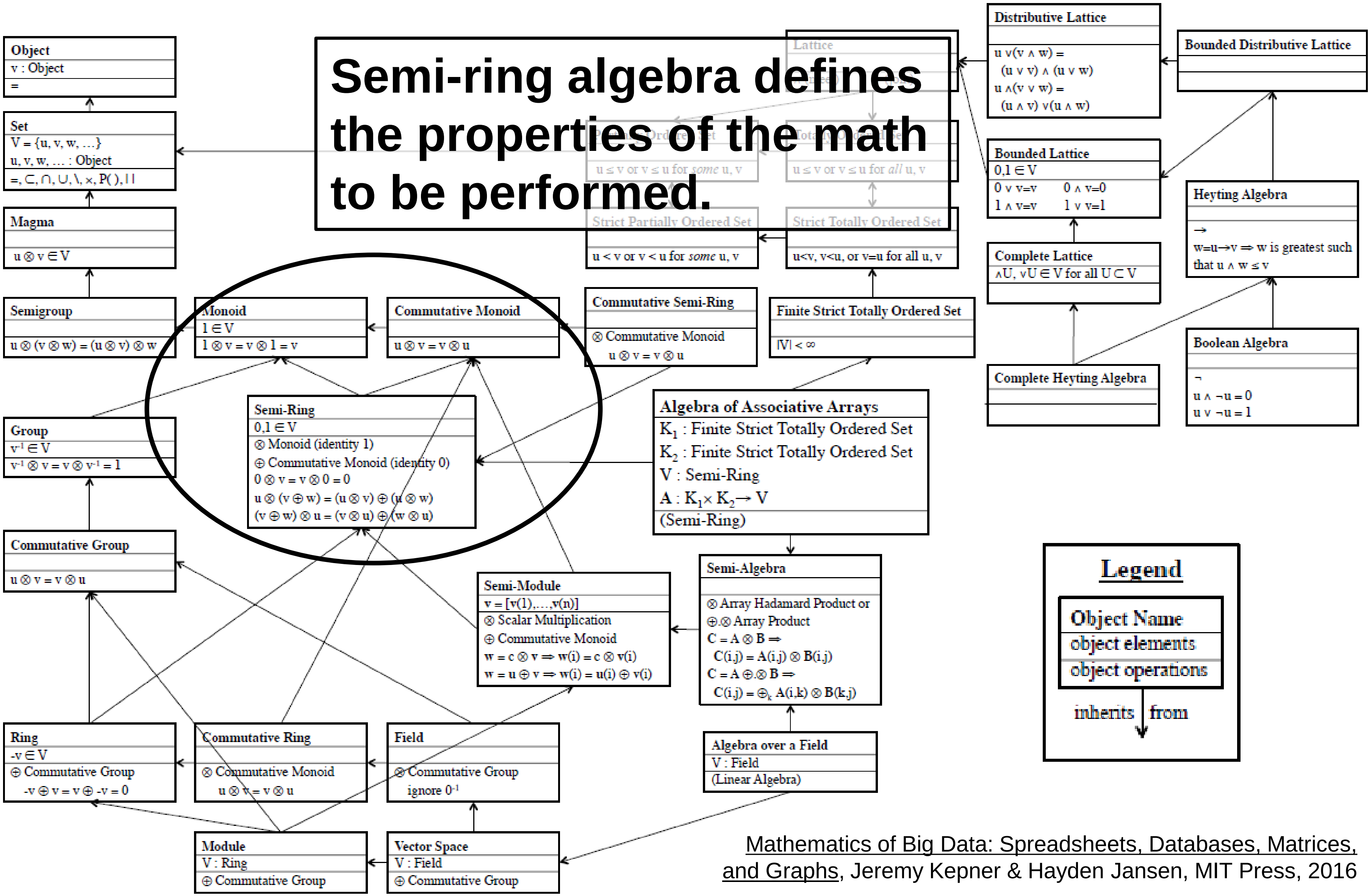
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

BACKUPS

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Background: Mathematics of Big Data

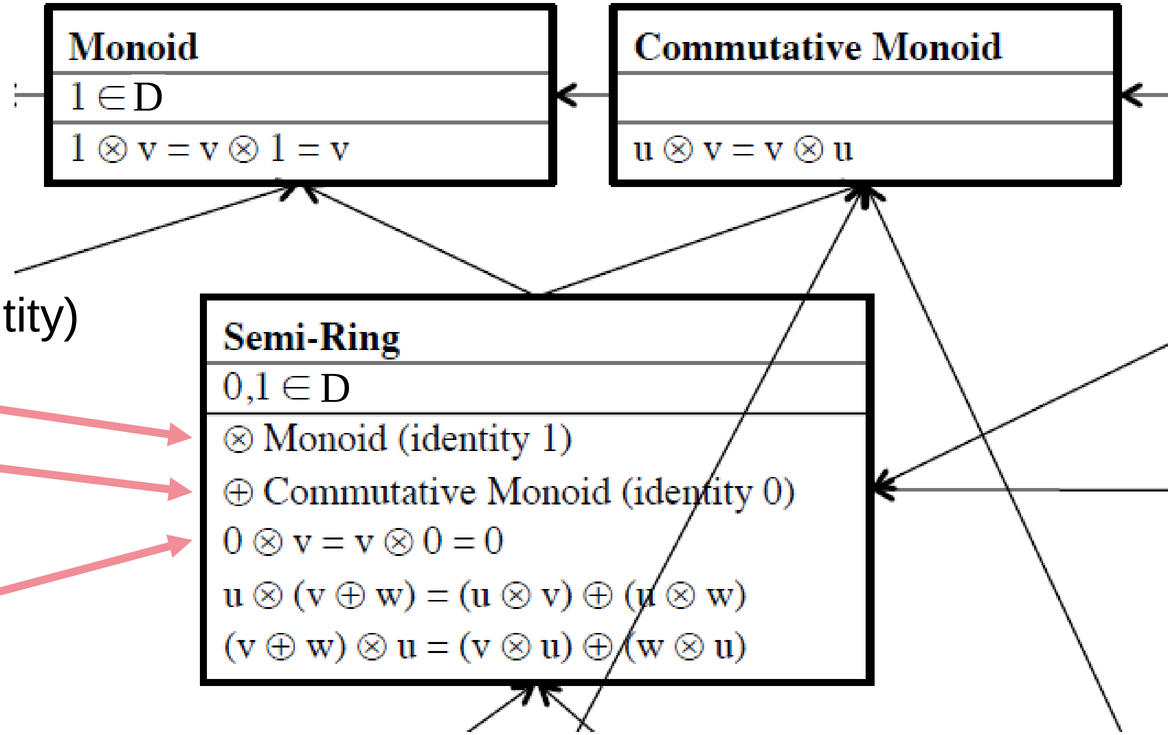
Semi-ring algebra defines the properties of the math to be performed.



Mathematics of Big Data: Spreadsheets, Databases, Matrices, and Graphs, Jeremy Kepner & Hayden Jansen, MIT Press, 2016

© 2016 Carnegie Mellon University
Distribution Statement A: Approved for Public Release; Distribution is Unlimited

Background: Semi-Ring, Monoid, Binary Function



One Domain, D

Two Monoids (binary func, identity)

- $\otimes$, ("multiply", "1")
- $\oplus$, ("add", "0")

Additive identity
= Multiplicative annihilator
= "Structural zero"

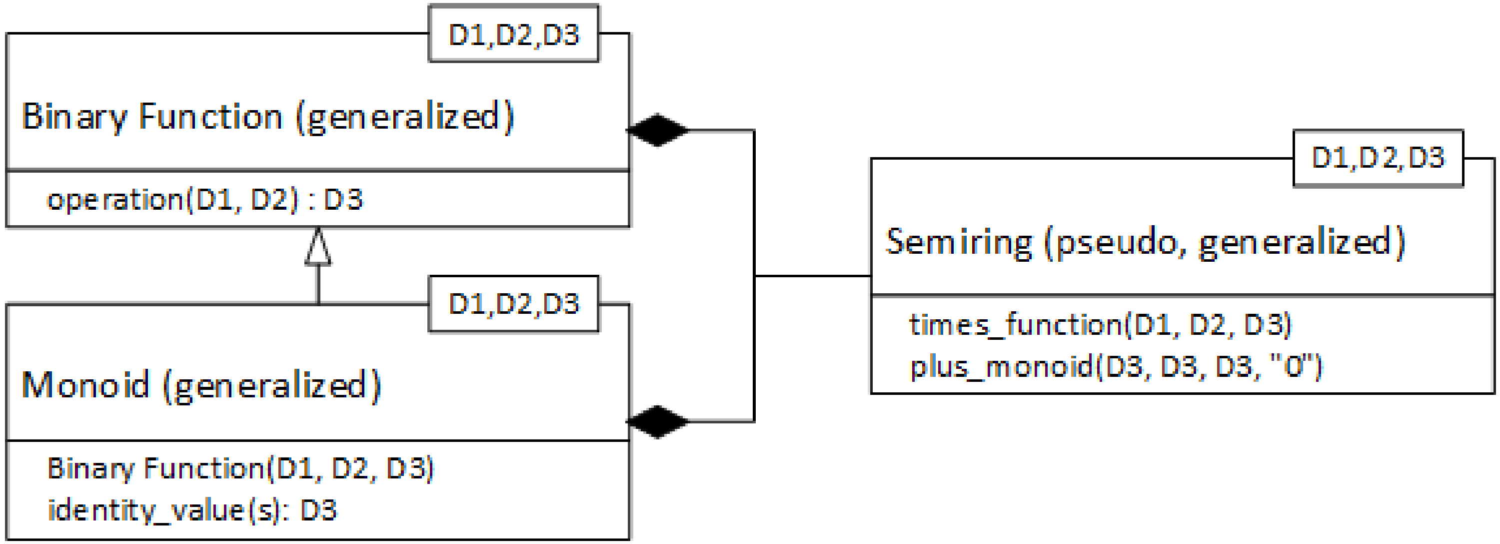
However…

- User can specify arbitrary binary function pairs (still refer to as $\otimes$ and $\oplus$)
- Operating over multiple domains
- That may not have a multiplicative annihilator

© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

"Generalized" Semiring Design

- Binary multiply function on multiple domains: D1 x D2 → D3
- Additive monoid on one domain: D3 x D3 → D3
- Additive identity specified on D3
- Current discussion:
  - Do we need a "Generalized Monoid" shown here?
  - Do we need to specify the multiplicative identity?
  - Do we require overlapping domains for the additive identity: $0 \in D_1 \cap D_2 \cap D_3$



© 2016 Carnegie Mellon University
Distribution Statement A:
Approved for Public Release; Distribution is Unlimited

Transpose Operation

Overloaded for 3 semantics:

1. Return a TransposeView wrapper around a *backend* Matrix.

```
// Modifier
template<typename MatrixT>
TransposeView<MatrixT> transpose(MatrixT const &A) {
    return TransposeView<MatrixT>(backend::transpose(A.mat));
}
```

2. Populate a new matrix with the scalar values transposed.
3. Modify the internal storage of existing matrix

```
// GraphBLAS operation: can check if A and C are the same object,
// and transpose in place if so.
template<typename MatrixT>
void transpose(MatrixT const &A, MatrixT &C) {...}
```


© 2016 Carnegie Mellon University

Distribution Statement A:
Approved for Public Release; Distribution is Unlimited