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

GraphBLAS Template Library (GBTL) v2.0: Design and Implementation

10 April 2019

Scott McMillan
Principal Member of Technical Staff
Emerging Technology Center
Software Engineering Institute
Carnegie Mellon University

Copyright 2019 Carnegie Mellon University. All Rights Reserved.

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.

DM19-0383

History of GraphBLAS and GBTL

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

A Brief History…

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

- Sep. 2013: HPEC "GraphBLAS position paper"
- Oct. 2014: Work on C++ GraphBLAS Template Library (GBTL) begins
- Jun. 2015: GraphBLAS "east coast/west coast" kickoff meeting
- Nov. 2015: GBTL v1.0 released (with CPU and GPU implementations).
- Dec. 2015: Formation of the "GraphBLAS Signature Proposal 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)**
- May 2018: GBTL v2.0 released (C++, mathematically equivalent to C API spec, CPU only)

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

Current/Future Activities

- May 2019:
  - GraphBLAS C API Version 1.3
  - IPDPS: “LAGraph position paper”
- Now:
  - API updates to GBTL API.
  - Optimized sequential implementation of primitives.
  - GBTL Algorithms development.
- H2 2019:
  - GraphBLAS C++ API Specification to begin
  - GraphBLAS Test Generation framework (advanced combinatorial testing)
  - Parallel and distributed implementations (GBTL + CombBLAS or HavoqGT)
  - Python DSL

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

Tim Mattson[‡], Timothy A. Davis[◦], Manoj Kumar[¶], Aydın Buluç[†], Scott McMillan[§], José Moreira[¶], Carl Yang[∗,†]

[‡]Intel Corporation [†]Computational Research Division, Lawrence Berkeley National Laboratory
[◦]Texas A&M University [¶]IBM Corporation [§]Software Engineering Institute, Carnegie Mellon University
[∗]Electrical and Computer Engineering Department, University of California, Davis

Abstract—In 2013, we released a position paper to launch a community effort to define a common set of building blocks for constructing graph algorithms in the language of linear algebra. This led to the GraphBLAS. We released a specification for the C programming language binding to the GraphBLAS in 2017. Since that release, multiple libraries that conform to the GraphBLAS C specification have been produced.
In this position paper, we launch the next phase of this ongoing community effort: a project to assemble a set of high level graph algorithms built on top of the GraphBLAS. While

GraphBLAS. We call this library of algorithms *LAGraph*. Just as we launched the GraphBLAS project with a position paper, we are launching this next phase of the project with a position paper.

The goals of the LAGraph effort include, first and foremost, bringing together the full range of known graph algorithms that can be constructed with the GraphBLAS. From this collection we will be able to systematically assess the coverage of graph

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

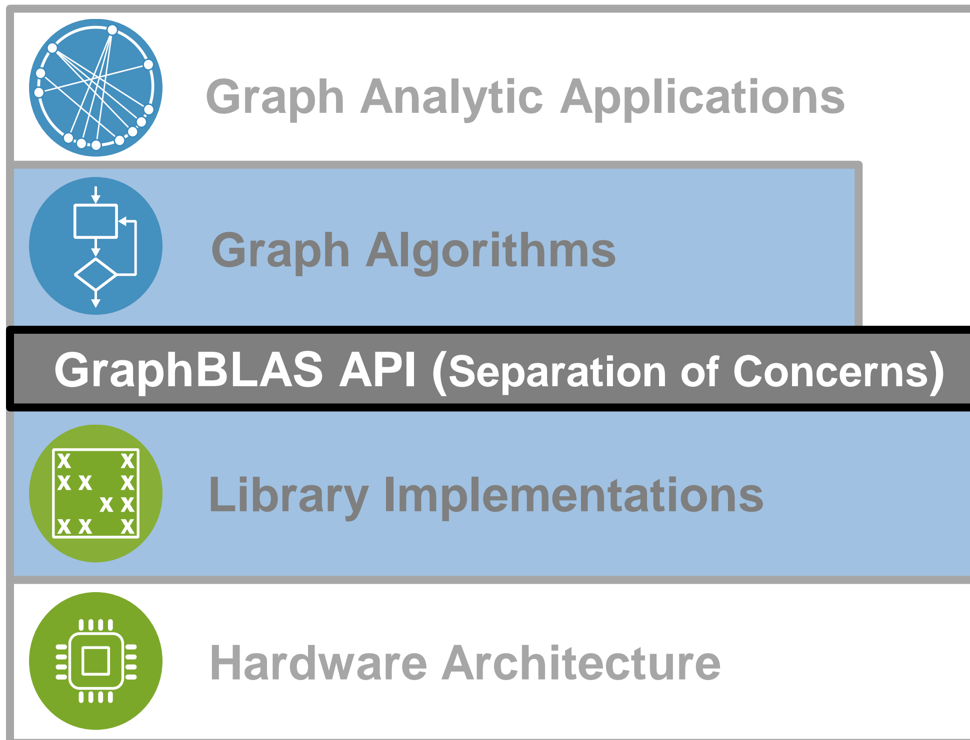
Overview of GBTL Library

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

GraphBLAS API

Goal: Achieve a ***separation of concerns*** between algorithm development "using" primitives and "implementation" of primitives with low-level hardware optimizations

- GraphBLAS C API Specification
- GBTL C++ API ("frontend")
- PyGB Python DSL



GraphBLAS Primitives (the Math)

Operation	Description	Mathematical Description
mxm	Matrix multiplication (bread-first traversal)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \oplus.\otimes \mathbf{B}^T)$
mxv, (vxm)		$\mathbf{c}\langle\neg\mathbf{m}, z\rangle = \mathbf{c} \odot (\mathbf{A}^T \oplus.\otimes \mathbf{b})$
eWiseMult	Element-wise 'multiplication' (graph intersection)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \otimes \mathbf{B}^T)$
eWiseAdd	Element-wise 'addition' (graph union)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot (\mathbf{A}^T \oplus \mathbf{B}^T)$
reduce (row/col)	Reduce along rows/cols (vertex degree)	$\mathbf{c}\langle\neg\mathbf{m}, z\rangle = \mathbf{c} \odot [\oplus_j \mathbf{A}^T(:,j)]$
apply	Apply unary function to each element (edge modification)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot f(\mathbf{A}^T)$
transpose	Swap rows and columns (reverse directed edges)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}^T$
extract	Extract a sub-matrix (sub-graph selection)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}^T(\mathbf{i},\mathbf{j})$
assign	Assign to a sub-matrix (sub-graph assignment)	$\mathbf{C}\langle\neg\mathbf{M}, z\rangle (\mathbf{i},\mathbf{j}) = \mathbf{C}(\mathbf{i},\mathbf{j}) \odot \mathbf{A}^T$
build (meth.)	Build a matrix from row, column, value tuples	$\mathbf{C} = \mathbb{S}^{mxn}(\mathbf{i},\mathbf{j},\mathbf{v},\odot)$
extractTuples (meth.)	Extract row, column, value tuples from a matrix	$(\mathbf{i},\mathbf{j},\mathbf{v}) = \mathbf{A}$

S. McMillan, et al., "Design and Implementation of the GraphBLAS Template Library (GBTL)," SIAM Annual Meeting (AN16), July 2016. Updated IEEE HPEC Conference, Sep 2017.

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, z – clear output, ⊕ monoid/binary function, ⊕.⊗ semiring, **blue** – optional parameters, **red** – optional modifiers

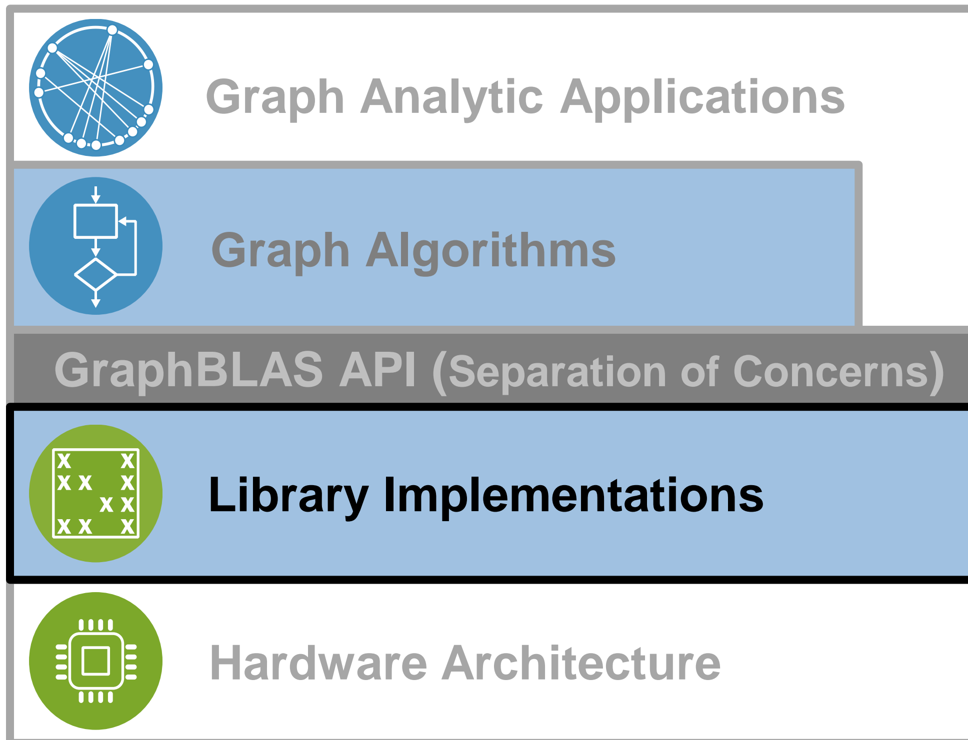
[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Library Implementations

Goal: Tune primitives for specific architecture.

All currently single-threaded CPU.

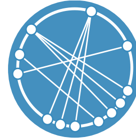
- IBM-graphBLAS (C, reference)
- SuiteSparse GraphBLAS (C, optimizations)
- GBTL "backend" (C++ reference)

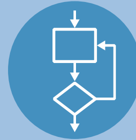


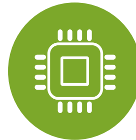
[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution.

Graph Algorithms

- Algorithms written in GBTL
  - Metrics: triangle count*, diameter, radius, eccentricity, degree, etc.
  - Traversals: Breadth-First Search (BFS)
  - SSSP, Delta-stepping*, Bellman-Ford, filtered Bellman-Ford, APSP
  - Maximal Independent Set
  - Betweenness Centrality
  - Connected Components
  - Minimum Spanning Tree
  - Clustering/Community Detection (peer pressure, Markov, Louvain*)
  - PageRank
  - K-truss Enumeration*
  - MaxFlow*
  - DFS* (in progress)







* Algorithms translated from vertex programs (CMU collaborator, T. Low)

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

GBTL Repository

- URL: https://github.com/cmu-sei/gbtl
- Build system: `cmake -DPLATFORM=<dirname>` (configures one 'backend' to build)

- `src/`

```
└ graphblas/                     - GraphBLAS API (public API  or 'frontend')
  └ detail/                      - helper code for frontend
  └ platforms/                   - Library Implementations (one per subdir)
    └ sequential/                - 'backend': reference implementation
    └ optimized_sequential/      - 'backend': work in progress (not on master)
    └ <your backend here>        - 'backend': MKL? TBB?
└ algorithms/                    - Graph Algorithms
└ test/                          - unit tests for public API*
```


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

Compare and Contrast GBTL C++ and GraphBLAS C

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

GraphBLAS in C++ or C…Which would you choose?!

```
// from GBTL src/algorithms/bfs.h

#include "graphblas.hpp"
using namespace GraphBLAS;

GEN_GRAPHBLAS_MONOID(LOrMonoid, LogicalOr, false);
GEN_GRAPHBLAS_SEMIRING(LogicalSemiring, LOrMonoid, LogicalAnd);

template <typename MatrixT, typename LevelVecT>
void bfs_level(LevelVecT &levels, MatrixT const &graph, IndexType src)
{
   IndexType N(graph.nrows());
   Vector<bool> wavefront(N);

   // BFS traversal and label the vertices.
   IndexType level = 0;
   while (wavefront.nvals() > 0) {
      ++level; // Increment the level

      // Apply the level to all newly visited nodes
      BinaryOp_Bind2nd<IndexType, Times<IndexType>> set_level(level);
      apply(levels, NoMask(), Plus<IndexType>(), set_level, wavefront);

      // Advance wavefront and mask out nodes already assigned levels
      mxv(wavefront, complement(levels), NoAccumulate(),
          LogicalSemiring<bool>(), transpose(graph), wavefront, true);
   }
}
```

B.2 Example: level BFS in GraphBLAS using apply

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <stdint.h>
4  #include <stdbool.h>
5  #include "GraphBLAS.h"
6
7  int32_t level = 0;                 // level = depth in BFS traversal, roots=1, unvisited=0
8  void return_level(void *out, const void *in) {
9    bool element = *(bool*)in;
10   *(int32_t*)out = level;
11 }
12
13 /*
14  * Given a boolean n x n adjacency matrix A and a source vertex s, performs a BFS traversal
15  * of the graph and sets v[i] to the level in which vertex i is visited (v[s] == 1).
16  * If i is not reachable from s, then v[i] = 0. (Vector v should be empty on input.)
17  */
18 GrB_Info BFS(GrB_Vector *v, const GrB_Matrix A, GrB_Index s)
19 {
20   GrB_Index n;
21   GrB_Matrix_nrows(&n,A);                              // n = # of rows of A
22
23   GrB_Vector_new(v,GrB_INT32,n);                       // Vector<int32_t> v(n) = 0
24
25   GrB_Vector q;                                        // vertices visited in each level
26   GrB_Vector_new(&q,GrB_BOOL,n);                       // Vector<bool> q(n) = false
27   GrB_Vector_setElement(q,(bool)true,s);               // q[s] = true, false everywhere else
28
29   GrB_Monoid Lor;                                      // Logical-or monoid
30   GrB_Monoid_new(&Lor,GrB_LOR,false);
31
32   GrB_Semiring Boolean;                                // Boolean semiring
33   GrB_Semiring_new(&Boolean,Lor,GrB_LAND);
34
35   GrB_Descriptor desc;                                 // Descriptor for vxm
36   GrB_Descriptor_new(&desc);
37   GrB_Descriptor_set(desc,GrB_MASK,GrB_SCMP);          // invert the mask
38   GrB_Descriptor_set(desc,GrB_OUTP,GrB_REPLACE);       // clear the output before assignment
39
40   GrB_UnaryOp apply_level;
41   GrB_UnaryOp_new(&apply_level,return_level,GrB_INT32,GrB_BOOL);
42
43   /*
44    * BFS traversal and label the vertices.
45    */
46   level = 0;
47   GrB_Index nvals;
48   do {
49     ++level;                                           // next level (start with 1)
50     GrB_apply(*v,GrB_NULL,GrB_PLUS_INT32,apply_level,q,GrB_NULL);  // v[q] = level
51     GrB_vxm(q,*v,GrB_NULL,Boolean,q,A,desc);           // q[!v] = q ||.&& A ; finds all the
52                                                        // unvisited successors from current q
53     GrB_Vector_nvals(&nvals, q);
54   } while (nvals);                                     // if there is no successor in q, we are done.
55
56   GrB_free(&q);                                        // q vector no longer needed
57   GrB_free(&Lor);                                      // Logical or monoid no longer needed
58   GrB_free(&Boolean);                                  // Boolean semiring no longer needed
59   GrB_free(&desc);                                     // descriptor no longer needed
60
61   return GrB_SUCCESS;
62 }
```


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

Operations signatures: mxv (from graphblas/operations.h)

```
template<typename WVectorT,
         typename MaskT,
         typename AccumT,
         typename SemiringT,
         typename AMatrixT,
         typename UVectorT>
inline void mxv(WVectorT          &w,
                MaskT       const &mask,
                AccumT             accum,
                SemiringT          op,
                AMatrixT    const &A,
                UVectorT    const &u,
                bool               replace_flag = false);
```

```
// equivalent signature in C API.

GrB_Info GrB_mxv(GrB_Vector                w,
                 GrB_Vector const          mask,
                 GrB_BinaryOp const        accum,
                 GrB_Semiring const        op,
                 GrB_Matrix const          A,
                 GrB_Vector const          u,
                 GrB_Descriptor const      desc);
```

- Goal: be "similar" to C API Spec **in use**…(*not always obvious when reading template declarations*).
- Almost all parameters *individually* templated (handles replaced with pass-by-"value", "&", and "const &" as needed)
  - allows arbitrary matrix backend and scalar types (can be different types for each operand)
  - allows GrB_NULL and GrB_ALL to be replaced with more strongly typed objects for specialization purposes.
- Descriptor replaced:
  - Views – vector and matrix wrappers – for transpose and complement
  - Replace flag (needs further discussion)
- Return type replaced with exceptions (needs further discussion)

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

Matrix (and Vector) C API methods become public member funcs (graphblas/Matrix.hpp)

```
template<typename ScalarT, typename... TagsT>
class Matrix
{
public:
   Matrix() = delete;
   Matrix(IndexType num_rows, IndexType num_cols);
   Matrix(Matrix<ScalarT,TagsT...> const &rhs); // copy ctor

   void clear();
   IndexType nrows() const;
   IndexType ncols() const;
   IndexType nvals() const;

   // other overloads of build provided (e.g., with iterators)
   template<typename ValueT, typename BinaryOpT=Second<ScalarT>>
   void build(IndexArrayType      const &row_indices,
              IndexArrayType      const &col_indices,
              std::vector<ValueT> const &vals,
              BinaryOpT                 dup = BinaryOpT());

   bool hasElement(IndexType row, IndexType col) const;
   void setElement(IndexType row, IndexType col, ScalarT const &val);
   ScalarT extractElement(IndexType row, IndexType col) const;
   /// @todo support a clearElement()?

   // other overloads of extractTuples provided
   template<typename RAIterI, typename RAIterJ, typename RAIterV>
   void extractTuples(RAIterI row_it, RAIterJ col_it, RAIterV vals);
   . . .
};
```

Signatures “close” to C API Spec.

```
        GrB_Matrix_new without dimensions not allowed
4.2.3.1 GrB_Matrix_new(&mat, ScalarT, nrows, ncols);
4.2.3.2 GrB_Matrix_dup(&mat, rhs);

4.2.3.3 GrB_Matrix_clear(mat);
4.2.3.4 GrB_Matrix_nrows(&nrows, mat);
4.2.3.5 GrB_Vector_ncols(&ncols, mat);
4.2.3.6 GrB_Vector_nvals(&nvals, mat);

4.2.3.7 GrB_Matrix_build(mat, &irows, &icols, &vals,
                         n, dup);

No GraphBLAS C API equivalent for hasElement (yet)
4.2.3.8 GrB_Matrix_setElement(mat, val, row, col);
4.2.3.9 GrB_Matrix_extractElement(&val, mat, row, col);

4.2.3.10 GrB_Matrix_extractTuples(&irows, &icols, &vals,
                                  n, mat);
```


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

Achieving "Opaque" Matrices and Vectors (from graphblas/Matrix.hpp, graphblas/detail/param_unpack.hpp)

```
// the frontend class.
template<typename ScalarT, typename... TagsT>
class Matrix
{
public:
    .
    .  // public interface: forwards to backend
    .
private:
   // the backend "opaque" type/object
   detail::matrix_generator::result<
           ScalarT,
           detail::SparsenessCategoryTag,
           detail::DirectednessCategoryTag,
           TagsT... ,
           detail::NullTag,
           detail::NullTag>::type m_mat;
};


// Example construction:
Matrix<double, DirectedMatrixTag> A(N, N);
```

Template parameters used to specify "complete" type

- ScalarT template parameter replaces Domains
  - Can be any type (not just C API predefined types)
- TagsT… is a variable number of template parameters
  - Used to provide information to the backend system to help determine the backend container type.
  - Examples include: DenseTag, SparseTag, DirectedMatrixTag (for matrices), etc..
  - We could send directives like: DistributedTag, ReplicatedTag, FastColumnAccessTag, etc.
- matrix_generator "parses" template parameters at compile time to compute the backend data type.
- Backend opaque type:
  - **Goal: Specialized for hardware and implementation**
  - Does not have to adhere to the "hints"
  - `GraphBLAS::backend::Matrix<ScalarT,SparseTag>`
  - This is a subclass of a specific type of matrix, i.e. `GraphBLAS::backend::LilSparseMatrix<ScalarT>`

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

Descriptors (from graphblas/{operations.hpp, TransposeView.hpp, ComplementView.hpp})

```
template<typename MatrixT>
class TransposeView
{
public:
   typedef typename backend::TransposeView<
      typename MatrixT::BackendType> BackendType;

   TransposeView(BackendType backend_mat)
      : m_mat(backend_mat) {}

   // Implements entire Matrix interface passes to m_mat

private:
   BackendType m_mat;
};
```

Descriptors are replaced with the following

- GrB_TRAN: TransposeView "wraps" input matrices
- GrB_SCMP: ComplementView "wraps" mask
- GrB_REPLACE: bool `replace_flag` parameter

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

Wrapper function used to instantiate the class properly. Not to be confused with transpose operation.

Mask structural complement done the same way with a "complement" wrapper function.

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

Algebra: Type, UnaryOp, BinaryOp (from src/graphblas/algebra.hpp)

```
// Unary Operator
template <typename D1, typename D2 = D1>
struct AdditiveInverse
{
   typedef D2 result_type;
   inline D2 operator()(D1 input) { return -input; }
};

typedef GraphBLAS::AdditiveInverse<bool>      GrB_AINV_BOOL;
typedef GraphBLAS::AdditiveInverse<int8_t>    GrB_AINV_INT8;
typedef GraphBLAS::AdditiveInverse<uint8_t>   GrB_AINV_UINT8;
...
```

*_new functions not necessary (Type, Unary, Binary, etc)

GBTL supports domains through template parameters (same order as std::unary_function

Typedefs make it look more like the C API. These are not necessary.

```
// Binary Operator
template<typename D1, typename D2 = D1, typename D3 = bool>
struct GreaterThan
{
   typedef D3 result_type;
   inline D3 operator()(D1 lhs, D2 rhs) { return lhs > rhs; }
};
```

`GrB_GreaterThan_<D1>`, returns bool by default but can be changed.

- NOTE: there is no penalty for user-defined operators when defined as functors like these.
- We need to investigate the use of lambda functions

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

Algebra: BinaryOp → UnaryOp, (from src/graphblas/algebra.hpp)

```
// Turn Binary Operator into Unary Operator by binding rhs
template <typename ConstT, typename BinaryOpT>
struct BinaryOp_Bind2nd
{
   typedef typename BinaryOpT::result_type result_type;

   BinaryOp_Bind2nd(ConstT const &value,
                    BinaryOpT     operation = BinaryOpT() )
      : n(value), op(operation)
   {}

   result_type operator()(result_type const &value)
   {
      return op(value, n);
   }

   ConstT n;
   BinaryOpT op;
};
```

You can turn a BinaryOp into a UnaryOp by binding an input operand (the second one in this example) to a constant "value".

The UnaryOp interface

```
BinaryOp_Bind2nd<unsigned int,
                 Minus<unsigned int>> subtract_1(1);
```

Example: `subtract_1`: A UnaryOp that subtracts one from any element.

NOTE: same type of mechanism can be used to turn Semirings into Monoids or BinaryOps.

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

Algebra: Monoids (from src/graphblas/algebra.hpp)

```
// Monoids
#define GEN_GRAPHBLAS_MONOID(M_NAME, BINARYOP, IDENTITY)\
template <typename ScalarT>                            \
struct M_NAME                                          \
{                                                      \
public:                                                \
   typedef ScalarT ScalarType;                         \
   typedef ScalarT result_type;                        \
                                                       \
   ScalarT identity() const {                          \
      return static_cast<ScalarT>(IDENTITY);           \
   }                                                   \
                                                       \
   ScalarT operator()(ScalarT lhs, ScalarT rhs) const { \
      return BINARYOP<ScalarT,                         \
                      ScalarT,                         \
                      ScalarT>()(lhs, rhs);            \
   }                                                   \
};

// Building a few common Monoids
GEN_GRAPHBLAS_MONOID(PlusMonoid, Plus, 0)
GEN_GRAPHBLAS_MONOID(TimesMonoid, Times, 1)
GEN_GRAPHBLAS_MONOID(MinMonoid, Min, numeric_limits<ScalarT>::max())
GEN_GRAPHBLAS_MONOID(MaxMonoid, Max, 0)
GEN_GRAPHBLAS_MONOID(LogicalOrMonoid, LogicalOr, false)
```

- We can generate monoid templates from any BinaryOp using this macro
- Using this generator is not necessary. Any class with this interface will work.
- NOTE: Monoid Interface satisfies the BinaryOp requirements: Monoids can be passed where BinaryOps are required.
- Enforcing one domain on the BINARYOP by specifying ScalarT three times.

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

Algebra: Semirings (from src/graphblas/algebra.hpp)

```
// Semirings
#define GEN_GRAPHBLAS_SEMIRING(SRNAME,ADD_MONOID,MULT_BINARYOP)\
template <typename D1, typename D2=D1, typename D3=D1>        \
class SRNAME                                                  \
{                                                             \
public:                                                       \
   typedef D3 ScalarType;                                     \
   typedef D3 result_type;                                    \
                                                              \
   D3 add(D3 a, D3 b) const                                   \
   { return ADD_MONOID<D3>()(a, b); }                         \
                                                              \
   D3 mult(D1 a, D2 b) const                                  \
   { return MULT_BINARYOP<D1,D2,D3>()(a, b); }                \
                                                              \
   ScalarType zero() const                                    \
   { return ADD_MONOID<D3>().identity(); }                    \
};


// Building a few common Semirings
GEN_GRAPHBLAS_SEMIRING(ArithmeticSemiring, PlusMonoid, Times)
GEN_GRAPHBLAS_SEMIRING(LogicalSemiring, LogicalOrMonoid, LogicalAnd)
...
```

We can generate semiring templates from any Monoid and BinaryOp using this macro.

The MULT_BINARYOP could also be a monoid.

Using this generator is not necessary. Any class with this interface will work.

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

Level BFS algorithm in GBTL (from src/algorithms/bfs.hpp)

```
#include "graphblas.hpp"
using namespace GraphBLAS;

GEN_GRAPHBLAS_MONOID(LOrMonoid, LogicalOr, false);
GEN_GRAPHBLAS_SEMIRING(LogicalSemiring, LOrMonoid, LogicalAnd);

template <typename MatrixT, typename LevelVecT>
void bfs_level(LevelVecT &levels, MatrixT const &graph, IndexType src)
{
   IndexType N(graph.nrows());
   Vector<bool> wavefront(N);

   // BFS traversal and label the vertices.
   IndexType level = 0;
   while (wavefront.nvals() > 0) {
      ++level; // Increment the level

      // Apply the level to all newly visited nodes
      BinaryOp_Bind2nd<IndexType, Times<IndexType>> set_level(level);
      apply(levels, NoMask(), Plus<IndexType>(), set_level, wavefront);

      // Advance wavefront and mask out nodes already assigned levels
      mxv(wavefront, complement(levels), NoAccumulate(),
          LogicalSemiring<bool>(), transpose(graph), wavefront, true);
   }
}
```

BFS algorithm to compute level at which each node is reached. Root is level 1.

Templated functor generators optionally used for Monoids and Semirings

Bind 'times' BinaryOp with a constant to create UnaryOp for apply

mxv used to also illustrate the transpose wrapper.

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

Building an Optimized Backend

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

Optimizing mxm in GraphBLAS

- The GraphBLAS primitive is a bit more complicated…
  - Transpose either input matrix (4 options)
  - Optional accumulate with output matrix (2 options)
  - Optional write mask, optionally complemented (3 options)
  - Merge/Replace semantics (2 options, when mask used)

- Optimizations for 40 different code paths investigated:

Func name: mxm	A*B	A*B^T	A^T*B	A^T*B^T
NoMask_NoAccum	C := A*B	C := A*B'	C := A'*B	C := A'*B'
NoMask_Accum	C := C + (A*B)	C := C + (A*B')	C := C + (A'*B)	C := C + (A'*B')
Mask_NoAccum, REPLACE	C := M .* (A*B)	C := M .* (A*B')	C := M .* (A'*B)	C := M .* (A'*B')
Mask_NoAccum, MERGE	C := [!M .* C] U {M .* (A*B)}	C := [!M .* C] U {M .* (A*B')}	C := [!M .* C] U {M .* (A'*B)}	C := [!M .* C] U {M .* (A'*B')}
Mask_Accum, REPLACE	C := [M .* C] + {M .* (A*B)}	C := [M .* C] + {M.* (A*B')}	C := [M .* C] + {M.* (A'*B)}	C := [M .* C] + {M.* (A'*B')}
Mask_Accum, MERGE	C := [!M .* C] U {[M .* C] + M .* (A*B)}	C := [!M .* C] U {[M .* C] + M.* (A*B')}	C := [!M .* C] U {[M .* C] + M.* (A'*B)}	C := [!M .* C] U {[M .* C] + M.* (A'*B')}
CompMask_NoAccum, REPLACE	C := !M .* (A*B)	C := !M .* (A*B')	C := !M .* (A'*B)	C := !M .* (A'*B')
CompMask_NoAccum, MERGE	C := [M .* C] U!M .* (A*B)	C := [M .* C] U {!M .* (A*B')}	C := [M .* C] U {!M .* (A'*B)}	C := [M .* C] U {!M .* (A'*B')}
CompMask_Accum, REPLACE	C := (!M .* C) + !M .* (A*B)	C := {(!M .* C) + !M .* (A*B')}	C := {(!M .* C) + !M .* (A'*B)}	C := {(!M .* C) + !M .* (A'*B')}
CompMask_Accum, MERGE	C := [M .* C] U {(!M .* C) + !M .* (A*B)}	C := [M .* C] U {(!M .* C) + !M .* (A*B')}	C := [M .* C] U {(!M .* C) + !M .* (A'*B)}	C := [M .* C] U {(!M .* C) + !M .* (A'*B')}

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

Optimizing mxm in GraphBLAS

THIS IS WITHOUT MULTITHREADING, OR EXPLICIT VECTORIZATION

Input Matrices A, B, M: N=9,877, M=51,946

We can do better here at the cost of a temporary matrix.

A*B → sparse axpy

Operation	Reference, msec	Optimized, msec	Speedup
C = A*B	10,315.00	27.00	382.04
C = C + A*B	10,299.00	63.00	163.48
C<M,mrg> = A*B	10,439.00	50.00	208.78
C<M,rplc> = A*B	10,114.00	48.00	210.71
C<M,mrg> = C + A*B	10,226.00	30.00	340.87
C<M,rplc> = C + A*B	10,148.00	30.00	338.27
C<!M,mrg> = A*B	12,040.00	41.00	293.66
C<!M,rplc> = A*B	11,816.00	56.00	211.00
C<!M,mrg> = C + A*B	12,164.00	58.00	209.72
C<!M,rplc> = C + A*B	11,849.00	58.00	204.29

$A*B^T$ → dot product

Operation	Reference, msec	Optimized, msec	Speedup
C = A*B'	8,392.00	2,461.00	3.41
C = C + A*B'	8,647.00	2,782.00	3.11
C<M,mrg> = A*B'	8,606.00	187.00	46.02
C<M,rplc> = A*B'	8,413.00	182.00	46.23
C<M,mrg> = C + A*B'	8,521.00	146.00	58.36
C<M,rplc> = C + A*B'	8,548.00	144.00	59.36
C<!M,mrg> = A*B'	10,352.00	2,714.00	3.81
C<!M,rplc> = A*B'	9,959.00	2,764.00	3.60
C<!M,mrg> = C + A*B'	10,460.00	2,851.00	3.67
C<!M,rplc> = C + A*B'	10,165.00	2,730.00	3.72

A^T*B → sparse axpy + temp matrix

Ref: ~2 hours, Opt.: 30-74 msec, Speedups: 10^5

A^T*B^T → sparse axpy + temp $matrix^T$

Ref: DNF, Opt.: 22-63 msec, Speedups: $10^?$

Ubuntu 16.04 LTS (Vmware Workstation), g++7.4.0, Intel Xeon CPU E3-1505M v5 2.8Ghz

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

Recurring Sparse Primitives

- Sparse "axpy" using semiring :
  - Sparse set union with "plus", $\oplus$, operation where elements overlap, one set is scaled.
  - $T[i] = T[i] \oplus (a_{ik} \otimes B[k])$
- Masked sparse "axpy" using semiring:
  - Like "sparse axpy" where elements are filtered by the binary mask, M:
  - $T[i] = M[i] .* [T[i] \oplus (a_{ik} \otimes B[k])]$
- Sparse accumulate:
  - Sparse set union with accumulate, $\odot$, operation where elements intersect
  - $C[i] = C[i] \odot T[i]$
- Masked sparse accumulate:
  - $C[i] = (M[i] .* C[i]) \odot T[i]$
  - where T[i] has already been filtered by M[i] (as in masked sparse axpy).

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

In closing…

- GBTL is guiding our (API committee's) thinking about the C++ API.
  - Spend time with GBTL to find its strengths and weaknesses.
  - Help us refine it (e.g., update it for C++17) and use that to influence the coming work on the C++ API Specification.

- GBTL has focused on completeness and correctness relative to the C API
  - Let's collaborate to create a highly optimized backend for Intel CPUs.
  - GPU and Distributed backends are also on the horizon.

- "The future is verified-lifting into a high-level operator representation followed by code synthesis to target any platform we wish."
  - Intel and CMU/SEI (and PNNL/UW) should work on this together…perhaps through the Intel/NSF funded CAPA program at UW

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

Backup Slides

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

Miscellaneous GraphBLAS "types" (from graphblas/{types.hpp, indices.hpp, Matrix.hpp})

```
namespace GraphBLAS
{
   typedef uint64_t                IndexType;

   typedef std::vector<IndexType>  IndexArrayType;

   class NoMask {...};

   class NoAccumulate {...};

   class AllIndices {...};
}
```

- **GraphBLAS** namespace replaces **GrB_** prefix.
- Replaces **GrB_Index**
- Replaces **GrB_Index* + size**
- Replaces **GrB_NULL** for the accumulate parameter
- Replaces **GrB_NULL** for the mask parameter
- Replaces **GrB_ALL** for the index array parameter in extract and assign

- With strong types (for the last three):
  - code paths are chosen at compile time via template specialization
  - No switch statements.

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

Vector (and Matrix) methods become public member funcs (from graphblas/Vector.hpp)

```
template<typename ScalarT, typename... TagsT>
class Vector
{
public:
   Vector() = delete;
   Vector(IndexType nsize);
   Vector(Vector<ScalarT,TagsT...> const &rhs); // copy ctor

   void clear();

   IndexType size() const;
   IndexType nvals() const;

   // other overloads of build provided (e.g., with iterators)
   template<typename ValueT, typename BinaryOpT=Second<ScalarT>>
   void build(IndexArrayType               indices,
              std::vector<ValueT> const &vals,
              BinaryOpT                    dup = BinaryOpT());

   bool hasElement(IndexType index) const;
   void setElement(IndexType index, ScalarT const &new_val);
   ScalarT extractElement(IndexType index) const;
   /// @todo support a clearElement()?

   // other overloads of extractTuples provided
   template<typename RAIterI, typename RAIterV>
   void extractTuples(RAIterI ids, RAIterV vals);

   . . .
};
```

Signatures "close" to C API Spec.

```
        GrB_Vector_new without dimensions not allowed
4.2.2.1 GrB_Vector_new(&vec, ScalarT, nsize);
4.2.2.2 GrB_Vector_dup(&vec, rhs);

4.2.2.3 GrB_Vector_clear(vec);

4.2.2.4 GrB_Vector_size(&nsize, vec);  // C++ return val
4.2.2.5 GrB_Vector_nvals(&nvals, vec); // C++ return val

4.2.2.6 GrB_Vector_build(vec, &indices, &vals, n, dup);

        No GraphBLAS C API equivalent
4.2.2.7 GrB_Vector_setElement(vec, val, index);
4.2.2.8 GrB_Vector_extractElement(&val, vec, index);

4.2.2.9 GrB_Vector_extractTuples(&ids, &vals, n, vec);
```


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