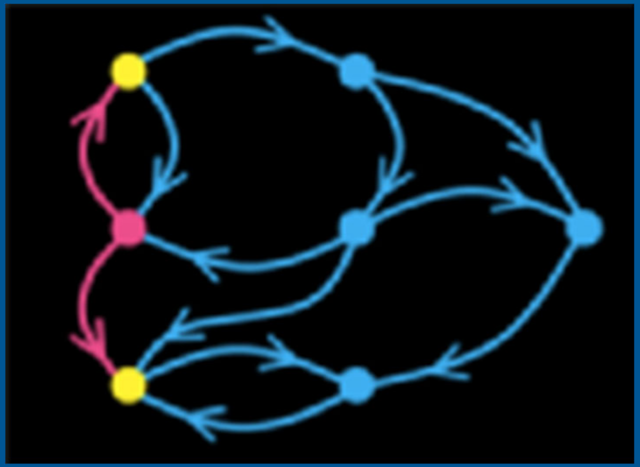
GraphBLAS Forum Updates

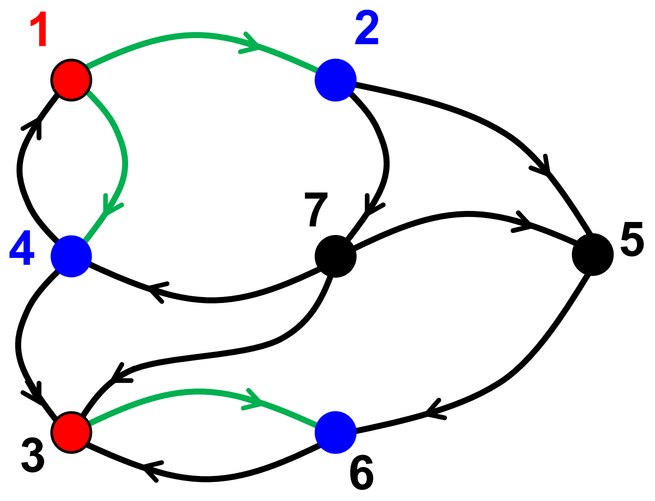
Scott McMillan

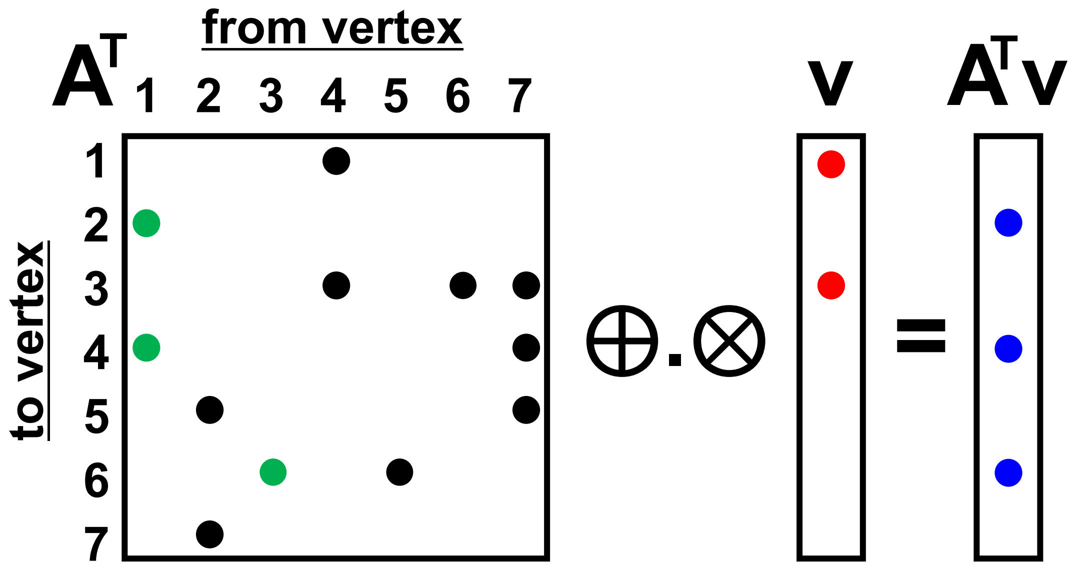
Software Engineering Institute
Carnegie Mellon University



Content provided by Benjamin Brock (Intel), Tim Davis (TAMU), James Kitchen (Anaconda), Manoj Kumar (IBM), Roi Lipman (Redis), Tim Mattson (Intel), Erik Welch (NVIDIA) and many others.

Introduction





GraphBLAS Forum Information

Website: http://graphblas.org

- Lists workshops and conferences
- Link to the latest C API Specification
- Lists teams developing implementations
- Other useful resources including the “The Math Document”

Mailing list: Graphblas@lists.lbl.gov

- Hosted by LBL (mailto:abuluc@lbl.gov)
- Join the Forum by joining the list

Monthly teleconference:

- Second Friday of every month, 12pm Eastern Time
- Send email to Jeremy Kepner to receive the calendar invite and Zoom ID.

GraphBLAS API Committees (we have reorganized)

C++ Subcommittee: Aydın Buluç, Tim Mattson, Scott McMillan, José Moreira, Benjamin Brock.

- C++ API Specification: under development
- Future: Distributed computing

C Subcommittee: Jim Kitchen, Erik Welch, Tim Mattson, Manoj Kumar, Will Kimmerer.

- C API Specification: Version 2.1 with type introspection and enhancements to address the needs of emerging applications (such as GNNs, and graph database queries).

"Math" Subcommittee: TBD.

- Defines the mathematical behaviour that should be implemented by a GraphBLAS library and can be referenced by any language API.

Note: We are _not_ planning to create committees/APIs for languages other than C/C++

GraphBLAS in the real world

- Language Bindings
  - C
  - Python
  - C++
  - Julia
  - others on the way…Go, Java, etc.
- Reference implementation: SuiteSparse:GraphBLAS
- LAGraph Algorithms Repository
- Commercial endeavors
  - Mathworks: MATLAB
  - RedisLabs: RedisGraph database
  - …and all the customers using those packages

GraphBLAS in the real world

- Language Bindings
  - C
  - Python
  - C++
  - Julia
  - others on the way…Go, Java, etc.
- Reference implementation: SuiteSparse:GraphBLAS
- LAGraph Algorithms Repository
- Commercial endeavors
  - Mathworks: MATLAB
  - RedisLabs: RedisGraph database
  - …and all the customers using those packages

Not just a research project anymore.

C API

Will Kimmerer, Jim Kitchen, Manoj Kumar, Tim Mattson, Erik Welch

C API Updates

https://github.com/GraphBLAS/graphblas-api-c

Expecting a new **minor release in 2023, version 2.1**

- **Type introspection** – the LAGraph work has shown this is essential for building libraries on top of the GraphBLAS.

Also considering numerous additional features including:

- Macros to identify library release information
- User defined Monoids with terminal values
- Query interface for monoids, semirings, operator domains, and execution modes.
- User-specified allocators/deallocators to use
- Miscellaneous refinements to existing operators and operations

Python

Jim Kitchen, Eric Welch

python-graphblas (was grblas)

https://python-graphblas.readthedocs.io
https://github.com/python-graphblas/python-graphblas

SSSP

```
while True:
  w = v.dup()
  v(binary.min) << semiring.min_plus(v @ M)
  if v.isequal(w):
    break
```

- Python wrapper around SuiteSparse:GraphBLAS with a more functional programming style
- Provides access to all GxB features in SuiteSparse
- Additional features:
  - Call Recorder – automatically generate equivalent C calls from Python code
  - Aggregators – advanced reductions (ex. avg, stdev, root mean square)
  - selectk – select the [first|last|smallest|largest|random] k elements from each row
- Easy to install (win/mac/linux, x86/arm64, wheels or conda)

graphblas-algorithms

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

- Similar concept to LAGraph, but written using python-graphblas
- 40+ algorithms so far (goal to implement majority of Networkx algorithms)
- **Will be used in NetworkX 3.0 fast-dispatching feature**

C++ API Specification

Benjamin Brock, Scott McMillan,
Tim Mattson, José Moreira, and Aydın Buluç

C++ Spec “Aspirations” (Design Goals)

- Better support for **user-defined types**
  - **First-class user-defined types**
  - **Non-memcpy-able scalar types**
  - **User-defined index types**
  - **First-class user-defined operators** (including lambdas)

- **Interoperability** with Standard Template Library

- Pathway for **advanced features**
  - **Distributed memory**
  - **GPU (device) support**

Generic Containers

Matrix Data Structure

```
grb::matrix<float>
```

Type of stored values

Matrix Data Structure

```
grb::matrix<float, int>
```

Type of stored values

(Integer) type used to store indices

Matrix Data Structure

```
grb::matrix<float, int, grb::column>
```

Type of stored values

(Integer) type used to store indices

Compile-time hint about storage format

Matrix Data Structure

```
grb::matrix<float, int, grb::column, my_alloc<float>>
```

Type of stored values

(Integer) type used to store indices

Compile-time hint about storage format

Allocator

Matrix Data Structure

```
grb::matrix<float, int, grb::column, my_alloc<float>>
```

Type of stored values

(Integer) type used to store indices

Compile-time hint about storage format

Allocator

Optional

Interoperability

- **Allocators** support
  - GPUs/device memory
  - Persistent memory
  - Any framework that wants to control memory allocation
- **Concepts** support interoperability with the **C++ Standard Library** *and* **user applications**
  - Matrices and vectors are Ranges
  - Views on matrices and vectors (transposes, rows, transforms, etc…)
  - Matrix concepts allow users to adapt *their* data structures to GraphBLAS

```
// Select a particular GPU
auto my_gpu = sycl::device(sycl::gpu_selector());

// Create allocator for `my_gpu`
auto alloc =
      sycl_tools::device_allocator<int>(my_gpu);

// Create matrix using GPU allocator
grb::matrix<float, int, grb::row,
            sycl_tools::device_allocator<int> >
       matrix({1024, 1024}, alloc);


// ...


// Using STL Algorithms
auto r = std::ranges::reduce(matrix);
```

Algorithms (GraphBLAS Operations)

GraphBLAS Operations – Overloading and Optional Arguments

Current draft introduces **multiply**, which multiplies **vectors** and/or **matrices**

Optional arguments and **overloading** results in cleaner syntax.

```
grb::matrix<float> a("chesapeake.mtx");

grb::vector<bool> x(a.shape()[1]);
x[5] = true;

// Default plus/times operators, "full mask"
auto b = grb::multiply(a, x);

// Equivalent, but explicitly declare operators
auto b_p = grb::multiply(a, x, grb::plus(), grb::times());

// Multiply with an explicit mask
auto next = grb::multiply(a, b, grb::plus(), grb::times(), x);
```

Operators

Binary Operators

Binary operators are implemented similarly to **STL's `<functional>`**

Can **specify one or more types**, or leave them to be **deduced**

Allows use of inline specifications like lambdas (not shown)

```
// Automatically deduce types of plus, times
auto b_p = grb::multiply(a, x, grb::plus(),
                               grb::times());

// Everything in floating point
auto next = grb::multiply(a, b, grb::plus<float>(),
                                grb::times<float>());

// Multiply in float, reduce in double
auto next = grb::multiply(
                 a, b,
                 grb::plus<double>(),
                 grb::times<float, float, double>());
```

Draft spec is accessible…Feedback Welcome

Check out **the spec**:

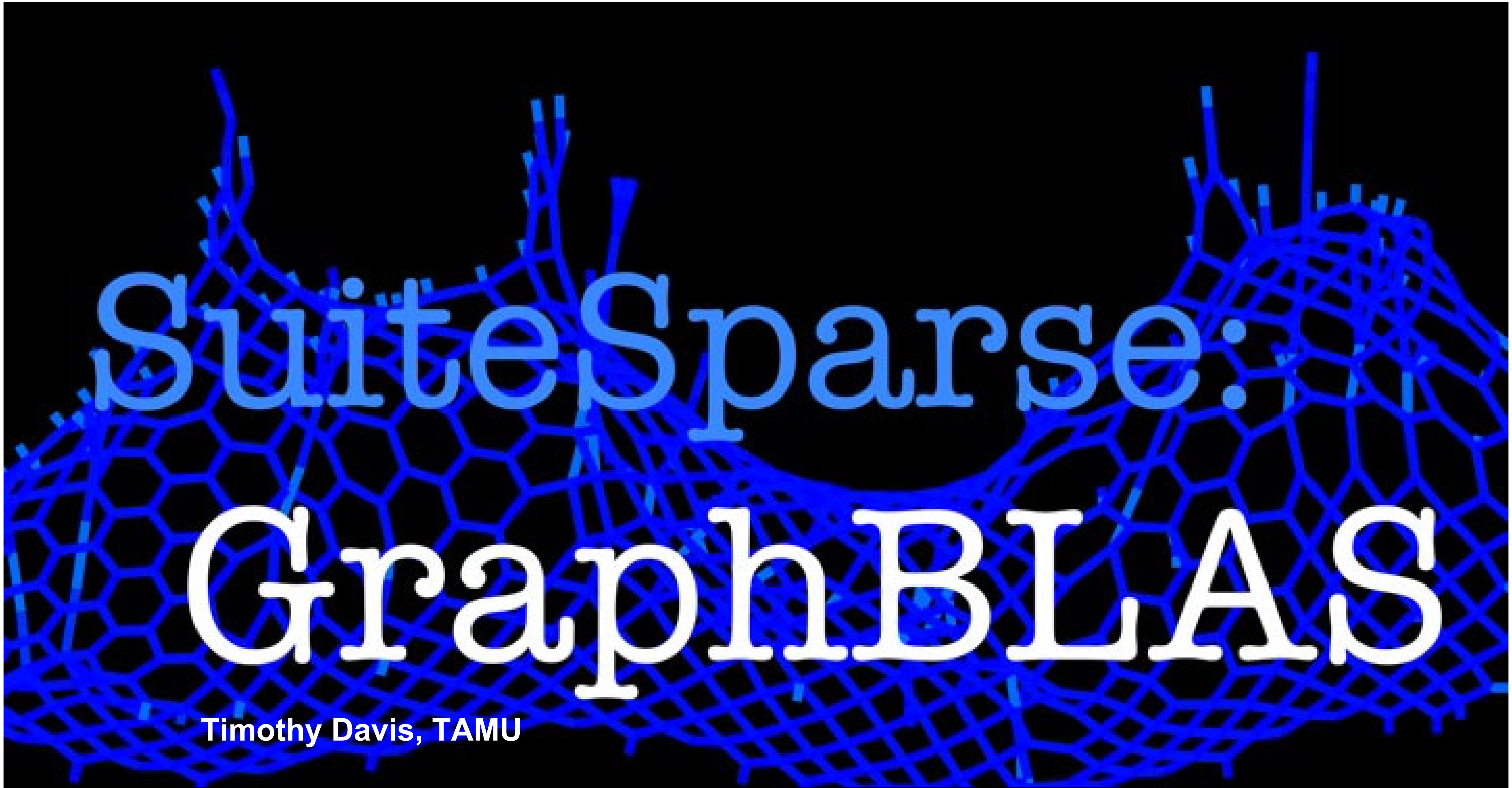
https://github.com/GraphBLAS/graphblas-api-cpp

Check out the rgri **reference implementation**:

https://github.com/GraphBLAS/rgri

SuiteSparse: GraphBLAS

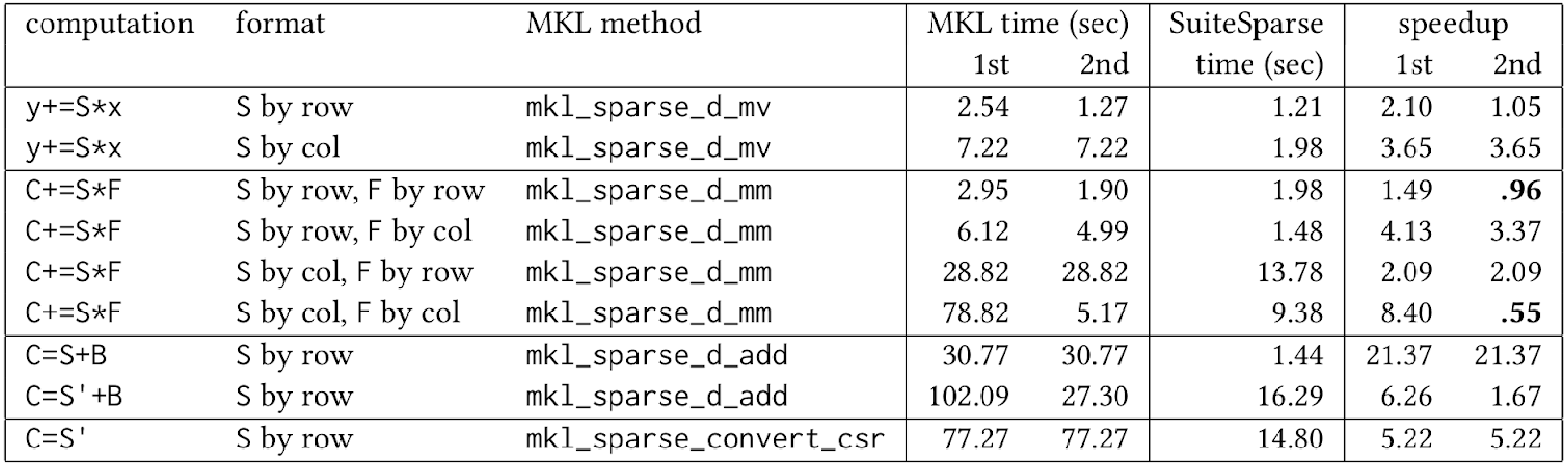
Timothy Davis, TAMU



SuiteSparse:GraphBLAS v7.2.0. Progress since 2021

- **Conforms to the v2.0 C API** (Nov 2021)
  - GrB_Scalar, GrB_IndexUnaryOp, GrB_serialize/deserialize with ZSTD compresssion
- New GxB features:
  - pack/unpack (O(1)-time move semantics)
  - named types and operators (for future JIT)
  - matrix and vector sort
  - eWiseUnion (like eWiseAdd but with 2 scalars; all entries in output go through the operator)
  - matrix and vector iterators
  - matrix reshape
- Performance:
  - GrB_mxm, particularly with sparse-times-dense or dense-times-sparse. AVX2 and AVX512 exploit
  - faster MATLAB interface
- Port to Octave 7
- Supported by Intel, NVIDIA, Redis, MIT Lincoln Lab, MathWorks, Julia Computing

SuiteSparse versus the Intel MKL sparse library



computation	format	MKL method	MKL time (sec)		SuiteSparse	speedup	
			1st	2nd	time (sec)	1st	2nd
`y+=S*x`	S by row	`mkl_sparse_d_mv`	2.54	1.27	1.21	2.10	1.05
`y+=S*x`	S by col	`mkl_sparse_d_mv`	7.22	7.22	1.98	3.65	3.65
`C+=S*F`	S by row, F by row	`mkl_sparse_d_mm`	2.95	1.90	1.98	1.49	**.96**
`C+=S*F`	S by row, F by col	`mkl_sparse_d_mm`	6.12	4.99	1.48	4.13	3.37
`C+=S*F`	S by col, F by row	`mkl_sparse_d_mm`	28.82	28.82	13.78	2.09	2.09
`C+=S*F`	S by col, F by col	`mkl_sparse_d_mm`	78.82	5.17	9.38	8.40	**.55**
`C=S+B`	S by row	`mkl_sparse_d_add`	30.77	30.77	1.44	21.37	21.37
`C=S'+B`	S by row	`mkl_sparse_d_add`	102.09	27.30	16.29	6.26	1.67
`C=S'`	S by row	`mkl_sparse_convert_csr`	77.27	77.27	14.80	5.22	5.22

Table 4. SuiteSparse vs MKL 2022 with the GAP-Twitter matrix

Work in progress and future work

- **Faster hypersparse matrices** (the “hyperhash”, avoids binary search), in v7.3.0beta
- **CUDA acceleration** (with J. Eaton and C. Nolet, NVIDIA)**: 3x to 9x speedup in GrB_mxm**
- **Julia integration (just announced v0.7), replacing Julia SparseArrays**
- more MATLAB integration
- further Python integration
- RedisGraph future: faster, more features
- JIT for faster user-defined types and operations
- aggressive non-blocking mode, kernel fusion
- x=A\b over a field
- more built-in types (FP16, complex integers, …)
- faster kernels (GrB_mxm for sampled dense-dense matrix multiply)
- matrices with shallow components
- …

https://github.com/DrTimothyAldenDavis/GraphBLAS

LAGraph: graph algorithms library

Tim Davis, Scott McMillan, Gabor Szarnyas, Tim Mattson,
Jim Kitchen, Eric Welch, David Bader, Roi Lipman, and contributors.

LAGraph: graph algorithm library

https://github.com/GraphBLAS/LAGraph

Version 1.0 released in September 2022

6 polished, stable algorithms (the GAP benchmark):

- Breadth-first search
- Betweenness-centrality
- PageRank
- Connected Components
- Single-source Shortest-Path
- Triangle Counting

Stable utilities

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

Many experimental algorithms to be curated

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

Experimental utilities

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

LAGraph: graph algorithm library

https://github.com/GraphBLAS/LAGraph

Version 1.0 released in September 2022

6 polished, stable algorithms (the GAP benchmark):

- Breadth-first search
- Betweenness-centrality
- PageRank
- Connected Components
- Single-source Shortest-Path
- Triangle Counting

Stable utilities

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

Many experimental algorithms to be curated

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

Experimental utilities

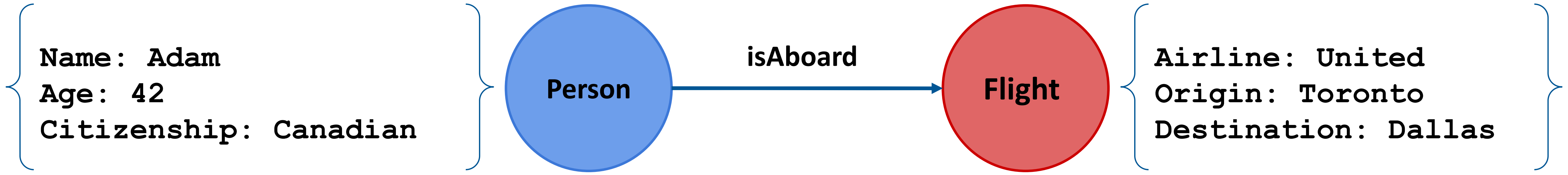
- random matrix, vector generators
- Binary matrix file I/O (very fast), serialize/deserialize, parallel LZ4 comp.

RedisGraph

Roi Lipman

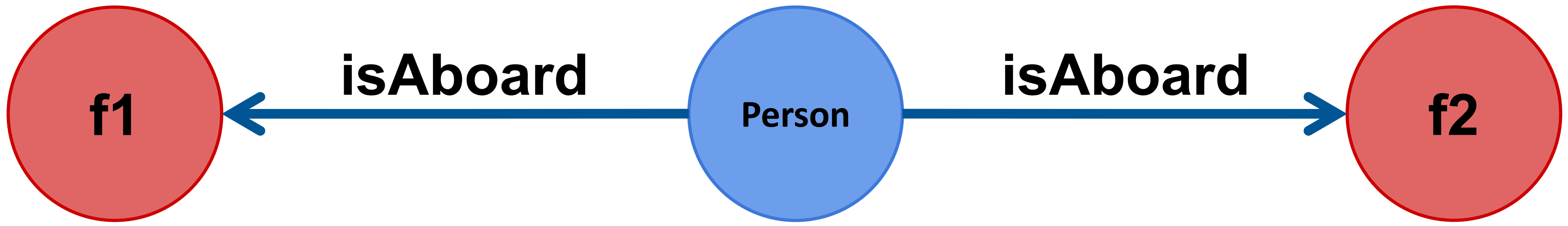
Property Graph DB

- Nodes represent entities
- Edges represent relations between entities
- Nodes & Edges associated with Attributes



Query graphs

MATCH (f1:Flight)<-[:isAboard]-(:Person)-[:isAboard]->(f2:Flight)



$$\left[\ \text{Flight}\ \right] * \left[\ \text{isAboard}\ \right]^{T} * \left[\ \text{Person}\ \right] * \left[\ \text{isAboard}\ \right] * \left[\ \text{Flight}\ \right]$$

RedisGraph Use Cases

- Social networks
- Supply chain optimization
- Fraud detection/prevention
- Resource management
- Access control

To learn more

https://redis.io/docs/stack/graph/

Questions?

Website: http://graphblas.org

- Lists workshops and conferences
- Links to the latest API Specifications
- Teams developing implementations
- Other useful resources

Mailing list: Graphblas@lists.lbl.gov

- Hosted by LBL (mailto:abuluc@lbl.gov)
- Join the Forum by joining the list

Monthly teleconference:

- Second Friday of every month, 12pm Eastern Time
- Send email to Jeremy Kepner to receive the calendar invite and Zoom ID.

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

SuiteSparse:GraphBLAS:
https://github.com/DrTimothyAldenDavis/GraphBLAS

Python GraphBLAS bindings
https://github.com/python-graphblas/python-graphblas

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

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

C++ reference implementation:
https://github.com/GraphBLAS/rgri

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

RedisGraph:
https://redis.io/docs/stack/graph