GraphBLAS: A Programming Specification for Graph Analysis

Scott McMillan, PhD

DM-0004100

Template updated 12.8.2016

GraphBLAS:
A Programming Specification for Graph Analysis

Scott McMillan, PhD

In collaboration with:

MIT/LL, LBNL, Intel, IBM, UC Davis, Indiana, and *many* others.

Summary

Graph analysis is *important* and *pervasive* in the DoD community.

GraphBLAS Forum (a world-wide consortium of researchers)

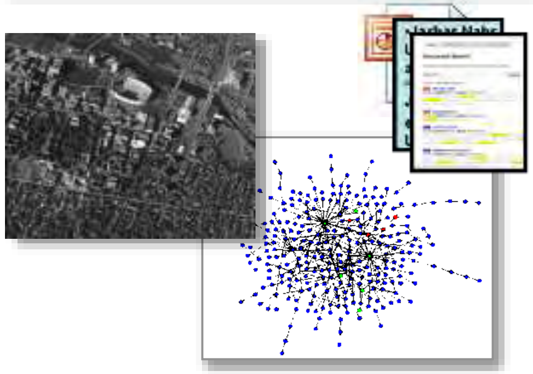
- Government/FFRDCs, academia, and industry
- Goal: application programming specification (API) for graph analysis
- GraphBLAS: Graph Basic Linear Algebra Subprograms

SEI contributions

- Member of the Graph BLAS Forum
- Member of the C API Specification committee
- Early implementation of a C++ library: GraphBLAS Template Library (open-source)

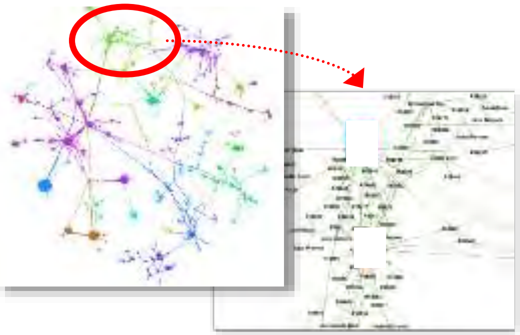
Graph Analysis is *Important* and *Pervasive*

ISR



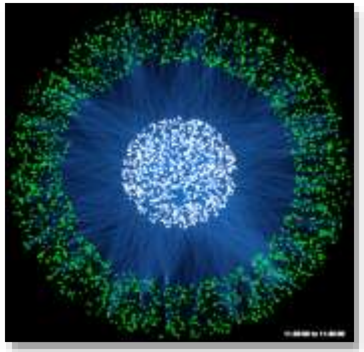
- Graphs represent entities and relationships detected through multi-INT sources
- 1,000s – 1,000,000s tracks and locations
- GOAL: Identify anomalous patterns of life

Social



- Graphs represent relationships between individuals or documents
- 10,000s – 10,000,000s individual and interactions
- GOAL: Identify hidden social networks

Cyber



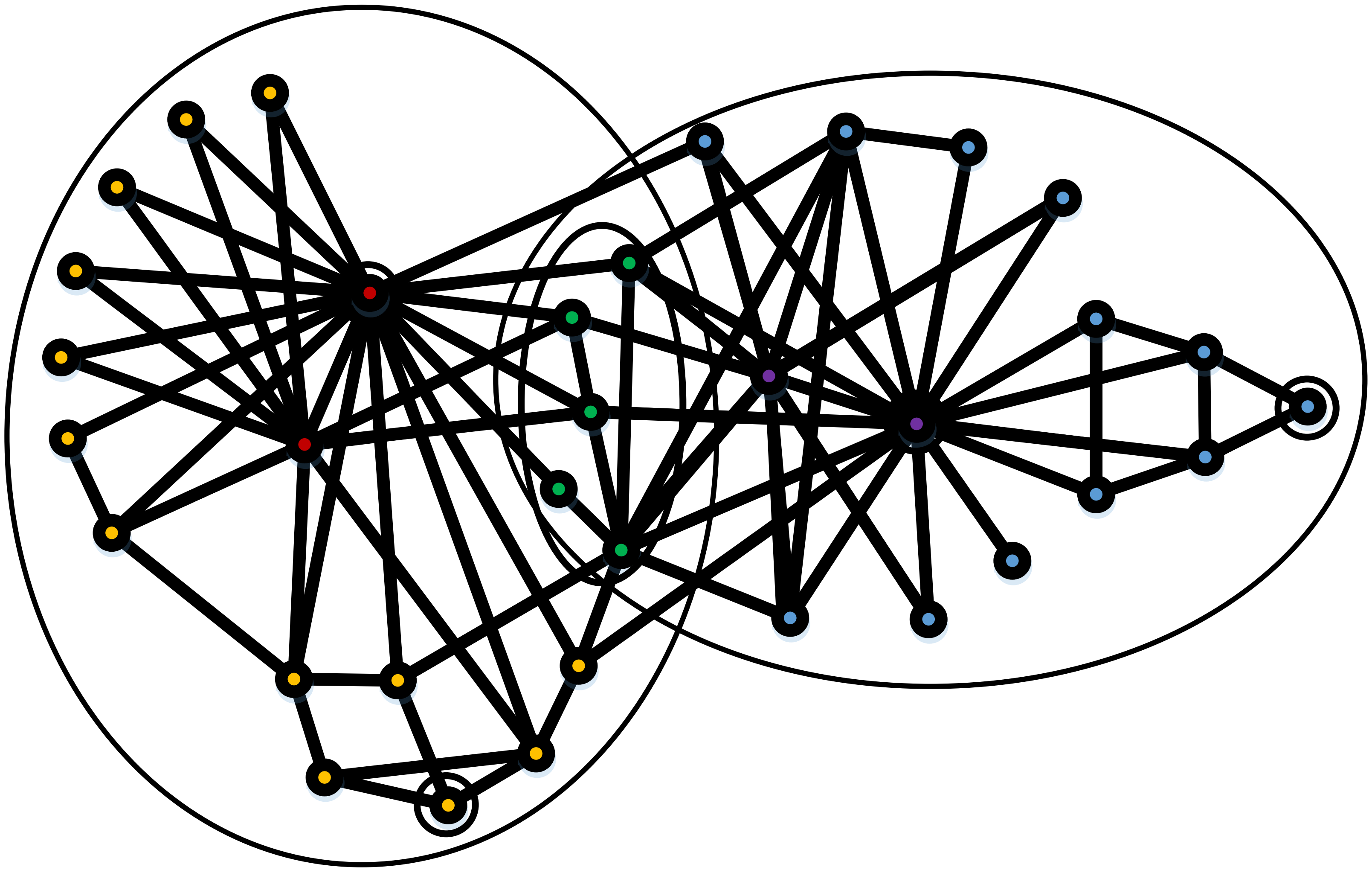
- Graphs represent communication patterns of computers on a network
- 1,000,000s – 1,000,000,000s network events
- GOAL: Identify cyber attacks or malicious software

Common Goal: Detection of subtle patterns in massive graphs

Slide credit: Jeremy Kepner, et al. “Mathematical Foundations of the GraphBLAS”, IEEE HPEC, Sept. 2016.

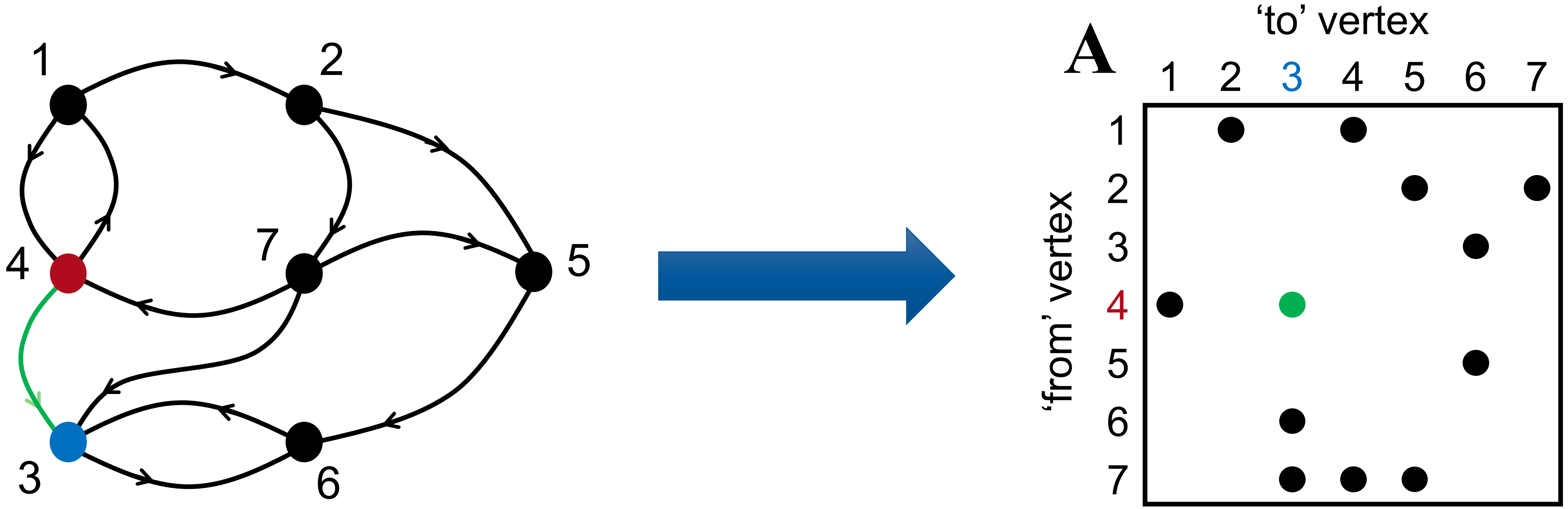
Common Operations

- Finding Neighbors
- Shortest Paths
- Clustering
- “Important” nodes
  - Influencers
  - Bottlenecks
  - Outliers



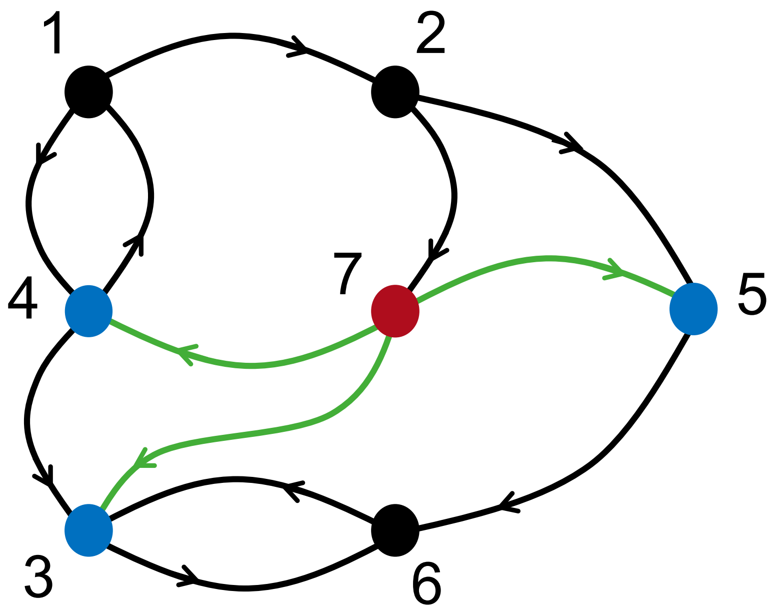
W. W. Zachary, An information flow model for conflict and fission in small groups, Journal of Anthropological Research 33, 452-473 (1977).

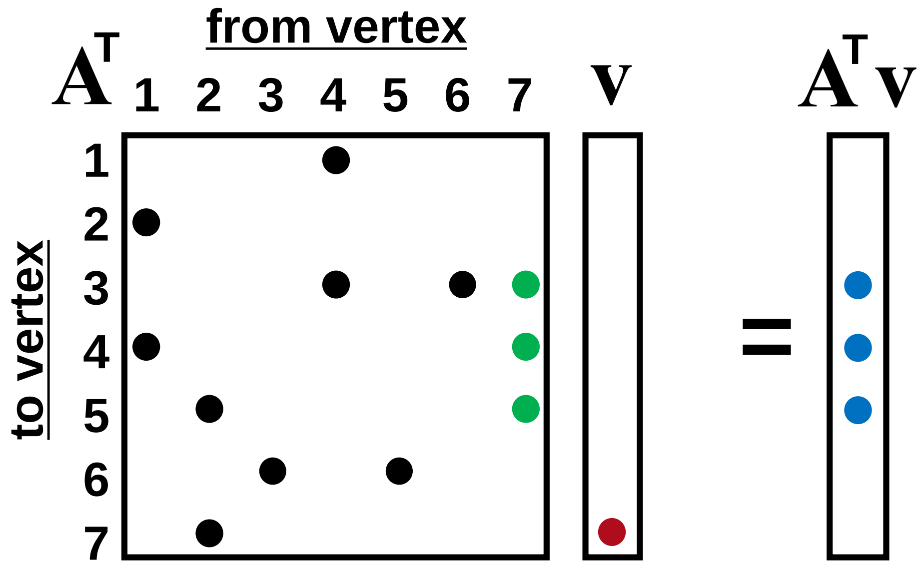
Graphs as Matrices



Graphs are represented as adjacency matrices that have *irregular* and *sparse* structure.

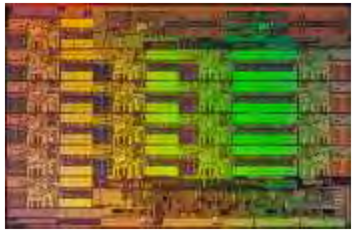
Graph Operations as Matrix Operations





- Matrix multiply → find neighbors (most important primitive)
- Used in breadth-first traversal, shortest paths, and many others
- Sparsity and irregularity of matrix structure is a barrier to high performance

Today’s Computing Landscape



Intel Xeon E5-2699v3
662 Gflop/s, 145 W
18 cores, 2.3 GHz
4-way/8-way AVX2

IBM POWER8
384 Gflop/s, 200 W
12 cores, 4 GHz
2-way/4-way VMX/VSX

NVIDIA Tesla P100
10.6 Tflop/s, 250 W
3584 cores, 1.48 GHz
64-way SIMT

Intel Xeon Phi
1.2 Tflop/s, 300 W
61 cores, 1.24 GHz
8-way/16-way LRBni

Qualcomm Snapdragon 810
10 Gflop/s, 2 W
4 cores, 2.5 GHz
A330 GPU, V50 DSP, NEON

Intel Atom C2750
29 Gflop/s, 20 W
8 cores, 2.4 GHz
2-way/4-way SSSE3

Dell PowerEdge R920
1.34 Tflop/s, 850 W
4x 15 cores, 2.8 GHz
4-way/8-way AVX

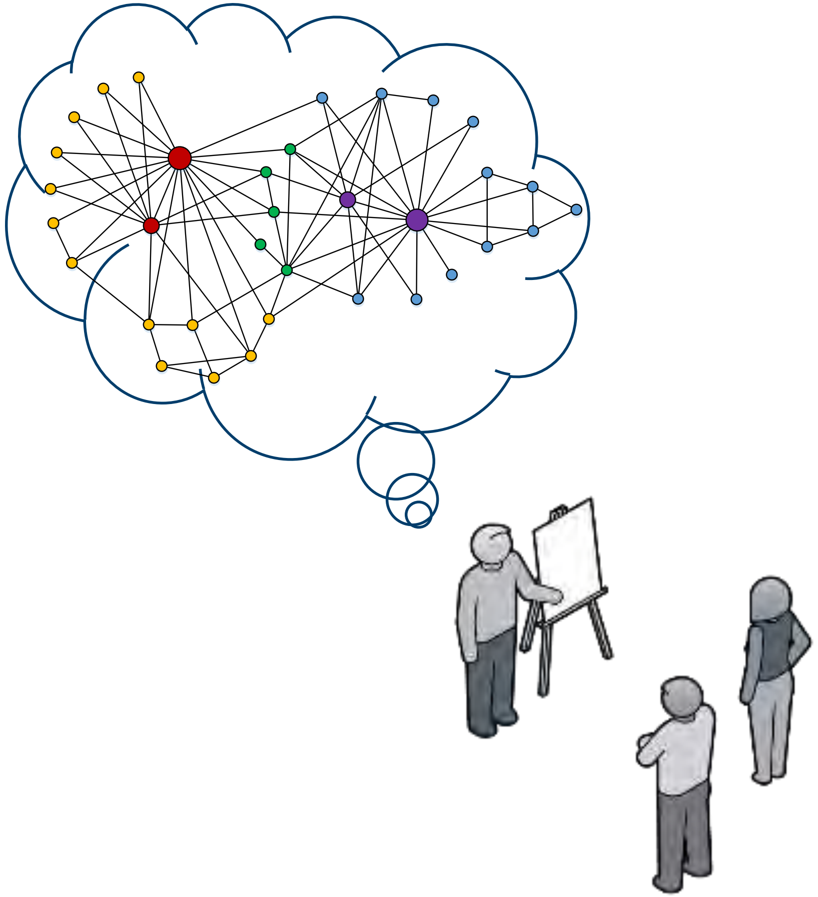
IBM BlueGene/Q
10 Pflop/s, 8 MW
48k x 16 cores, 1.6 GHz
4-way QPX

Slide credit: Franz Franchetti, “SPIRAL: Automated Code Generation of Performance Libraries,” 2016.

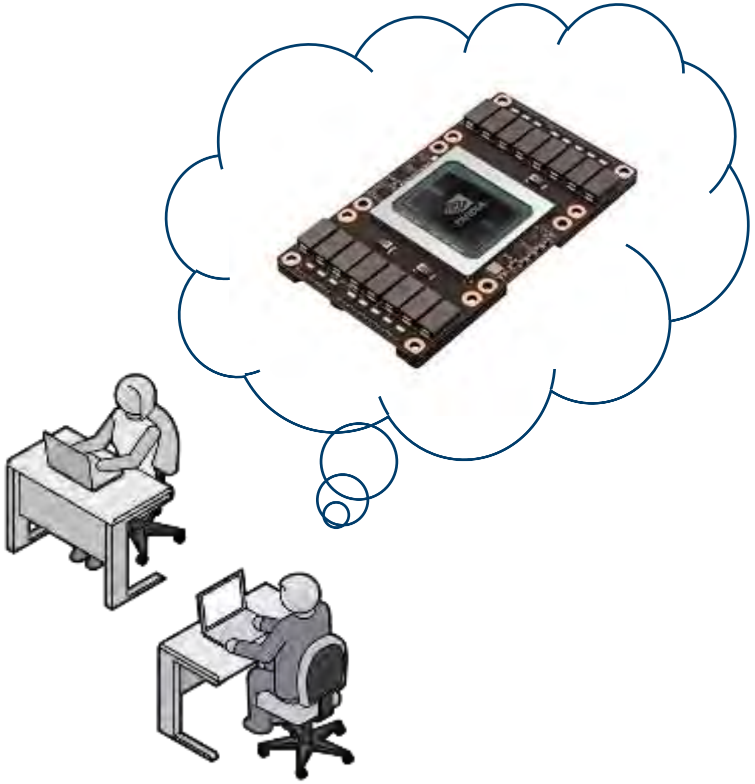
1 Gflop/s = one billion floating-point operations (additions or multiplications) per second

Separation of Concerns

Separate the complexity of graph analysis from the complexity of hardware systems:



Separation of Concerns



Graph Primitives Using Linear Algebra

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)

“It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of **a standard set of primitive building blocks**. This paper is a position paper defining the problem and **announcing our intention to launch an open effort to define this standard**.”

Presented at the IEEE High Performance Extreme Computing Conference. Waltham, MA, Sept. 2013.

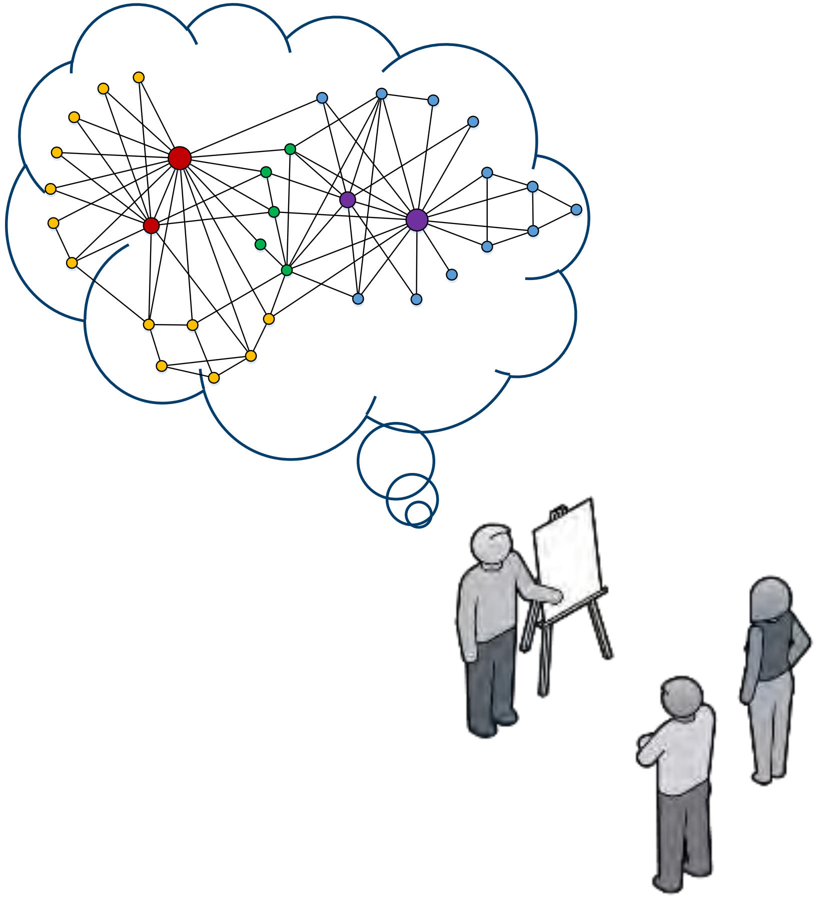
GraphBLAS Forum

FFRDCs	Industry	Academia
• MIT/Lincoln Labs*	• Intel* †	• UC Santa Barbara*, Davis †, Berkeley
• Lawrence Berkeley NL* †	• IBM †	• Georgia Tech*
• CMU/Software Engineering Institute †	• NVIDIA	• Karlsruhe (KIT)*
• Pacific Northwest NL	• Hauwei	• CMU
• Sandia NL	• Reservoir Labs	• Indiana U.
	• Galois	• MIT
	• Mellanox	• U. Washington
	• and others	• and others

* - steering committee, † - C API subcommittee

Separation of Concerns

GOAL: write once, run everywhere (with help from hardware experts).



GraphBLAS Application Programming Interface (API)

GraphBLAS Primitives

Operation	Description
mxm, mxv, vxm	Perform matrix *multiplication* (e.g., breadth-first traversal)
eWiseAdd, eWiseMult	Element-wise *addition* and *multiplication* of matrices (e.g., graph union, intersection)
extract	Extract a sub-matrix from a larger matrix (e.g., sub-graph selection)
assign	Assign to a sub-matrix of a larger matrix (e.g., sub-graph assignment)
apply	Apply *unary function* to each element of matrix (e.g., edge weight modification)
reduce	*Reduce* along columns or rows of matrices (vertex degree)
transpose	Swaps the rows and columns of a sparse matrix (e.g., reverse directed edges)
buildMatrix	Build an matrix representation from row, column, value tuples
extractTuples	Extract the row, column, value tuples from a matrix representation

GraphBLAS Primitives: The Math

Operation	Mathematical Description	Outputs	Inputs
mxm	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{T} \oplus.\otimes \mathbf{B}^{T}$	**C**	¬, **M**, ⊕, **A**, **T**, ⊕.⊗, **B**, **T**
mxv (vxm)	$\mathbf{c}(\neg\mathbf{m}) \oplus= \mathbf{A}^{T} \oplus.\otimes \mathbf{b}$	**c**	¬, **m**, ⊕, **A**, **T**, ⊕.⊗, **b**
eWiseMult	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{T} \otimes \mathbf{B}^{T}$	**C**	¬, **M**, ⊕, **A**, **T**, ⊗, **B**, **T**
eWiseAdd	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{T} \oplus \mathbf{B}^{T}$	**C**	¬, **M**, ⊕, **A**, **T**, ⊕, **B**, **T**
reduce (row)	$\mathbf{c}(\neg\mathbf{m}) \oplus= \oplus_{j} \mathbf{A}^{T}(:,j)$	**c**	¬, **m**, ⊕, **A**, **T**, ⊕
apply	$\mathbf{C}(\neg\mathbf{M}) \oplus= f(\mathbf{A}^{T})$	**C**	¬, **M**, ⊕, **A**, **T**, *f*
transpose	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{T}$	**C**	¬, **M**, ⊕, **A (T)**
extract	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbf{A}^{T}(\mathbf{i},\mathbf{j})$	**C**	¬, **M**, ⊕, **A**, **T**, **i**, **j**
assign	$\mathbf{C}(\neg\mathbf{M})\ (\mathbf{i},\mathbf{j}) \oplus= \mathbf{A}^{T}$	**C**	¬, **M**, ⊕, **A**, **T**, **i**, **j**
buildMatrix	$\mathbf{C}(\neg\mathbf{M}) \oplus= \mathbb{S}^{mxn}(\mathbf{i},\mathbf{j},\mathbf{v},\oplus)$	**C**	¬, **M**, ⊕, ⊕, m, n, **i**, **j**, **v**
extractTuples	$(\mathbf{i},\mathbf{j},\mathbf{v}) = \mathbf{A}(\neg\mathbf{M})$	**i**,**j**,**v**	¬, **M**, **A**

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

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

GraphBLAS Primitives: The Code

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

```
GrB_Info GrB_mxm(GrB_Matrix                    *C,      // output
                 const GrB_Matrix               Mask,
                 const GrB_BinaryFunction       accum,
                 const GrB_Semiring             op,
                 const GrB_Matrix               A,      // input matrix
                 const GrB_Matrix               B       // input matrix
               [, const Descriptor              desc]);
```

Common Elements

- Matrices (C, Mask, A, and B) are opaque data structures defined by the library implementers.
- Destination (C) is first.
- Output mask (M) specifies which output elements can assigned (optional).
- Accumulation function (accum) allows operation to combine with existing values (optional).
- Descriptor can specify transpose of input matrices or logical complement of the output mask.

The GraphBLAS Forum

- Meeting since 2014
- Teleconferences once per month (C API group more often)
- Forum meetings open to the public:
  - May: IEEE IPDPS (Symposium) – Graph Algorithms Building Blocks Workshop
  - Sept.: IEEE HPEC (Conference) – GraphBLAS BoF
  - Nov.: IEEE/ACM Supercomputing (Conference) – GraphBLAS Working Group
- Status and next steps:
  - Primitives established
  - C Application Programming Interface (API) – First draft review in August
  - Targeted release: November 2016 at Supercomputing
- For more information: http://graphblas.org

Current and Future Work

- Continue work on C++ API from the open-source GraphBLAS Template Library (GBTL) [collaboration with Indiana/PNNL]
- Line Project (FY17-18) on automated code generation for high performance graph libraries [collaboration with CMU ECE department]
  - Starting with the GraphBLAS primitives
  - Targeting COTS hardware (CPUs and GPUs)
- DARPA's HIVE (Hierarchical Identify Verify Exploit) BAA released in August
  - References the work of the GraphBLAS Forum
  - Goal: develop a special-purpose graph processing chip

Contact Information

Presenter / Point of Contact
Scott McMillan, PhD

Senior Member of Technical Staff

Telephone: +1 412.268.5156

Email: smcmillan@sei.cmu.edu