Mathematical Foundations of the GraphBLAS

Jeremy Kepner (MIT Lincoln Laboratory Supercomputing Center), Peter Aaltonen (Indiana University),
David Bader (Georgia Institute of Technology), Aydın Buluç (Lawrence Berkeley National Laboratory),
Franz Franchetti (Carnegie Mellon University), John Gilbert (University of California, Santa Barbara),
Dylan Hutchison (University of Washington), Manoj Kumar (IBM),
Andrew Lumsdaine (Indiana University), Henning Meyerhenke (Karlsruhe Institute of Technology),
Scott McMillan (CMU Software Engineering Institute), Jose Moreira (IBM),
John D. Owens (University of California, Davis), Carl Yang (University of California, Davis),
Marcin Zalewski (Indiana University), Timothy Mattson (Intel)

Abstract—**The GraphBLAS standard** (GraphBlas.org) **is being developed to bring the potential of matrix-based graph algorithms to the broadest possible audience. Mathematically, the GraphBLAS defines a core set of matrix-based graph operations that can be used to implement a wide class of graph algorithms in a wide range of programming environments. This paper provides an introduction to the mathematics of the GraphBLAS. Graphs represent connections between vertices with edges. Matrices can represent a wide range of graphs using adjacency matrices or incidence matrices. Adjacency matrices are often easier to analyze while incidence matrices are often better for representing data. Fortunately, the two are easily connected by matrix multiplication. A key feature of matrix mathematics is that a very small number of matrix operations can be used to manipulate a very wide range of graphs. This composability of a small number of operations is the foundation of the GraphBLAS. A standard such as the GraphBLAS can only be effective if it has low performance overhead. Performance measurements of prototype GraphBLAS implementations indicate that the overhead is low.**

I. Introduction

Graphs are among the most important abstract data structures in computer science, and the algorithms that operate on them are critical to applications in bioinformatics [Georganas et al 2014], computer networks, and social media [Ediger et al 2010], [Ediger et al 2011], [Riedy et al 2012], [Riedy & Bader 2013]. Graphs have been shown to be powerful tools for modeling complex problems because of their simplicity and generality [Staudt et al 2016], [Bergamini & Meyerhenke 2016]. For this reason, the field of graph algorithms has become one of the pillars of theoretical computer science, informing research in such diverse areas as combinatorial optimization, complexity theory, and topology. Graph algorithms have been adapted and implemented by the military, commercial industry, and researchers in academia, and have become essential in controlling the power grid, telephone systems, and, of course, computer networks.

Parallel graph algorithms are notoriously difficult to implement and optimize [Ediger et al 2012], [Ediger & Bader 2013], [McLaughlin & Bader 2014a], [McLaughlin & Bader 2014b], [McLaughlin et al 2014], [Staudt & Meyerhenke 2016]. The irregular data access patterns and inherently high communication-to-computation ratios found in graph algorithms mean that even the best algorithms will have parallel efficiencies that decrease as the number of processors is increased [Buluç & Gilbert 2012], [Azad et al 2015]. Recent work on communication-avoiding algorithms, and their applications to graph computations [Ballard et al 2013], [Solomonik et al 2013], might defer but not completely eliminate the parallel scalability bottleneck. Consequently, novel hardware architectures will also be required [Song et al 2010], [Song et al 2013]. A common graph processing interface provides a useful tool for optimizing both software and hardware to provide high performance graph applications.

The duality between the canonical representation of graphs as abstract collections of vertices and edges and a matrix representation has been a part of graph theory since its inception [Konig 1931], [Konig 1936]. Matrix algebra has been recognized as a useful tool in graph theory for nearly as long (see [Harary 1969] and references therein, in particular [Sabadusi 1960], [Weischel 1962], [McAndrew 1963], [Teh & Yap 1964], [McAndrew 1965], [Harary & Tauth 1964], [Brualdi 1967]). The modern description of the duality between graph algorithms and matrix mathematics (or sparse linear algebra) has been extensively covered in the literature and is summarized in the cited text [Kepner & Gilbert 2011]. This text has further spawned the development of the GraphBLAS math library standard (GraphBLAS.org)[Mattson et al 2013] that has been developed in a series of proceedings [Mattson 2014a], [Mattson 2014b], [Mattson 2015], [Buluç 2015], [Mattson 2016] and implementations [Buluç & Gilbert 2011], [Kepner et al 2012], [Ekanadham et al 2016], [Hutchison et al 2015], [Anderson et al 2016], [Zhang et al 2016]. This paper describes the mathematical properties that have been developed since [Kepner & Gilbert 2011] to support the GraphBLAS.

The foundational mathematical concepts for matrix-based graph analysis are the adjacency matrix and incidence matrix

This material is based in part upon work supported by the NSF under grant number DMS-1312831, by DOE ASCR under contract number DE-AC02-05CH11231, and by the DoD under contract number FA8721-05-C-0003. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Department of Energy, or the Department of Defense.

representations of graphs. From these concepts, a more formal definition of a matrix can be constructed. How such a matrix can be manipulated depends on the types of values the matrix holds and the operations allowed on those values. Furthermore, the mathematical properties of the matrix values determine the mathematical properties of the whole matrix. This paper describes the key mathematical concepts of the GraphBLAS and presents preliminary results that show the overhead of the GraphBLAS is minimal (as compared to their underlying matrix libraries).

II. Adjacency Matrix

Given an adjacency matrix $\mathbf{A}$, if

$$\mathbf{A}(i,j) = 1$$

then there exists an edge going from vertex i to vertex j (see Figure 1). Likewise, if

$$\mathbf{A}(i,j) = 0$$

then there is no edge from i to j. Adjacency matrices can have direction, which means that $\mathbf{A}(i,j)$ may not be the same as $\mathbf{A}(j,i)$. Adjacency matrices can also have edge weights. If

$$\mathbf{A}(i,j) = a \neq 0$$

then the edge going from i to j is said to have weight a. Adjacency matrices provide a simple way to represent the connections between vertices in a graph. Adjacency matrices are often square, and both the out-vertices (rows) and the in-vertices (columns) are the same set of vertices. Adjacency matrices can be rectangular, in which case the out-vertices (rows) and the in-vertices (columns) are different sets of vertices. Such graphs are often called bipartite graphs. In summary, adjacency matrices can represent a wide range of graphs, which include any graph with any set of the following properties: directed, weighted, and/or bipartite.

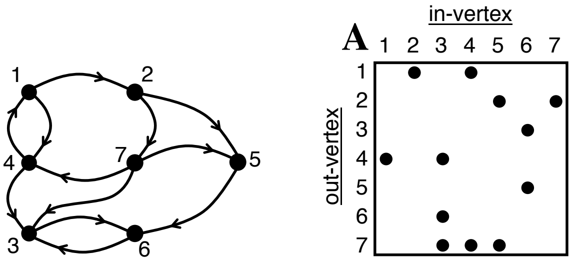

Fig. 1. (left) Seven-vertex graph with 12 edges. Each vertex is labeled with an integer. (right) 7×7 adjacency matrix $\mathbf{A}$ representation of the graph. $\mathbf{A}$ has 12 nonzero entries corresponding to the edges in the graph.

III. Incidence Matrix

An incidence, or edge matrix $\mathbf{E}$, uses the rows to represent every edge in the graph and the columns to represent every vertex. There are a number of conventions for denoting an edge in an incidence matrix. One such convention is to use two incidence matrices

$$\mathbf{E}_{\text{out}}(k,i) = 1 \qquad \text{and} \qquad \mathbf{E}_{\text{in}}(k,j) = 1$$

to indicate that edge k is a connection from i to j (see Figure 2). Incidence matrices are useful because they can easily represent multi-graphs, hyper-graphs, and multipartite graphs. These complex graphs are difficult to capture with an adjacency matrix. A multi-graph has multiple edges between the same vertices. If there was another edge, k', from i to j, this relationship can be captured in an incidence matrix by setting

$$\mathbf{E}_{\text{out}}(k',i) = 1 \qquad \text{and} \qquad \mathbf{E}_{\text{in}}(k',j) = 1$$

(see Figure 3) [Note: Another convention is to use +1 and -1, in which case the resulting matrix multiplication is the graph Laplacian.] In a hyper-graph, one edge can connect more than two vertices. For example, to denote that edge k has a connection from i to j and j' can be accomplished by also setting

$$\mathbf{E}_{\text{in}}(k,j') = 1$$

(see Figure 3). Furthermore, i, j, and j' can be drawn from different classes of vertices. $\mathbf{E}$ can be used to represent multipartite graphs by defining an additional incidence array $\mathbf{E}'_{\text{in}}$ and seting

$$\mathbf{E}'_{\text{in}}(k,j') = 1$$

Thus, an incidence matrix can be used to represent a graph with any set of the following graph properties: directed, weighted, multipartite, multi-edge, and/or hyper-edge.

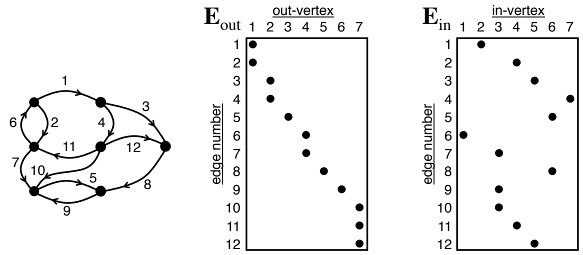

Fig. 2. (left) Seven-vertex graph with 12 edges. Each edge is labeled with an integer; the vertex labels are the same as in Figure 1. (middle) 12×7 incidence matrix $\mathbf{E}_{\text{out}}$ representing the out-vertices of the graph edges. (right) 12×7 incidence matrix $\mathbf{E}_{\text{in}}$ representing the in-vertices of the graph edges. Both $\mathbf{E}_{start}$ and $\mathbf{E}_{\text{in}}$ have 12 nonzero entries corresponding to the edges in the graph.

IV. Matrix Values

A typical matrix has m rows and n columns of real numbers. Such a matrix can be denoted as

$$\mathbf{A} : \mathbb{R}^{m \times n}$$

The row and and column indexes of the matrix $\mathbf{A}$ are

$$i \in I = \{1, \ldots, m\}$$

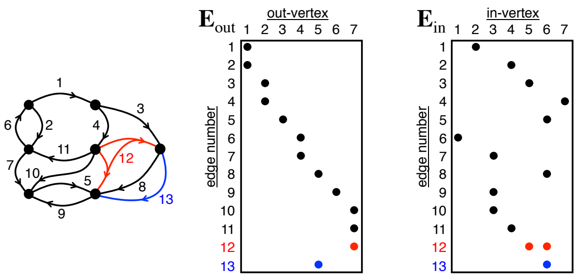

Fig. 3. Graph and incidence matrices from Figure 2 with a hyper-edge (edge 12) and a multi-edge (edge 13). The graph is a hyper-graph because edge 12 has more than one in-vertex. The graph is a multi-graph because edge 8 and edge 13 have the same out- and in-vertex.

and

$$j \in J = \{1,\ldots,n\}$$

so that any particular value $\mathbf{A}$ can be denoted as $\mathbf{A}(i,j)$. The row and column indices of matrices are natural numbers $I,J : \mathbb{N}$. [Note: a specific *implementation* of these matrices might use IEEE 64-bit double-precision floating point numbers to represent real numbers, 64-bit unsigned integers to represent row and column indices, and the compressed sparse rows (CSR) format or the compressed sparse columns (CSC) format to store the nonzero values inside the sparse matrix.]

A matrix of complex numbers

$$\mathbb{C} = \{x + y\sqrt{\text{-}1} : x,y \in \mathbb{R}\}$$

is denoted

$$\mathbf{A} : \mathbb{C}^{m\times n}$$

A matrix of integers

$$\mathbb{Z} = \{\ldots,-1,0,1,\ldots\}$$

is denoted

$$\mathbf{A} : \mathbb{Z}^{m\times n}$$

A matrix of natural numbers

$$\mathbb{N} = \{1,2,3,\ldots\}$$

is denoted

$$\mathbf{A} : \mathbb{N}^{m\times n}$$

Using the above concepts, a matrix is defined as the following two-dimensional (2D) mapping

$$\mathbf{A} : I \times J \to \mathbb{S}$$

where the indices $I,J : \mathbb{Z}$ are finite sets of integers with m and n elements, respectively, and

$$\mathbb{S} \in \{\mathbb{R},\mathbb{Z},\mathbb{N},\ldots\}$$

is a set of scalars. Without loss of generality, matrices can be denoted

$$\mathbf{A} : \mathbb{S}^{m\times n}$$

A *vector* is a matrix in which either $m = 1$ or $n = 1$. A column vector is denoted $\mathbf{v} : \mathbb{S}^{m\times 1}$ or simply $\mathbf{v} : \mathbb{S}^{m\times 1}$. A row vector can be denoted $\mathbf{v} : \mathbb{S}^{1\times n}$ or simply $\mathbf{v} : \mathbb{S}^{n}$. A scalar is a single element of a set $s \in \mathbb{S}$ and has no matrix dimensions.

V. Scalar Operations

Matrix operations are built on top of scalar operations that can be used for combining and scaling graph edge weights. The primary scalar operations are standard arithmetic addition, such as

$$1 + 1 = 2$$

and arithmetic multiplication, such as

$$2 \times 2 = 4$$

These scalar operations of addition and multiplication can be defined to be a wide variety of functions. To prevent confusion with standard arithmetic addition and arithmetic multiplication, $\oplus$ will be used to denote scalar addition and $\otimes$ will be used to denote scalar multiplication. In this notation, standard arithmetic addition and arithmetic multiplication of real numbers

$$a,b,c \in \mathbb{R}$$

where

$$\oplus \equiv + \qquad \text{and} \qquad \otimes \equiv \times$$

results in

$$c = a \oplus b \qquad \Rightarrow \qquad c = a + b$$

and

$$c = a \otimes b \qquad \Rightarrow \qquad c = a \times b$$

Generalizing $\oplus$ and $\otimes$ to a variety of operations enables a wide range of algorithms on scalars of all different types (not just real or complex numbers).

Certain $\oplus$ and $\otimes$ combinations over certain sets of scalars are particularly useful because they preserve essential mathematical properties, such as additive commutativity

$$a \oplus b = b \oplus a$$

multiplicative commutativity

$$a \otimes b = b \otimes a$$

additive associativity

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

multiplicative associativity

$$(a \otimes b) \otimes c = a \otimes (b \otimes c)$$

and the distributivity of multiplication over addition

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

The properties of commutativity, associativity, and distributivity are *extremely* useful properties for building graph applications because they allow the builder to swap operations without changing the result. Example combinations of $\oplus$ and

$\otimes$ that preserve scalar commutativity, associativity, and distributivity include (but are not limited to) standard arithmetic

$$\oplus \equiv + \qquad \otimes \equiv \times \qquad a,b,c \in \mathbb{R}$$

max-plus algebras

$$\oplus \equiv \max \qquad \otimes \equiv + \qquad a,b,c \in \{-\infty \cup \mathbb{R}\}$$

max-min algebras

$$\oplus \equiv \max \qquad \otimes \equiv \min \qquad a,b,c \in \{\text{-}\infty \cup \mathbb{R}_{\leq 0}\}$$

finite (Galois) fields such as GF(2)

$$\oplus \equiv \text{xor} \qquad \otimes \equiv \text{and} \qquad a,b,c \in \{0,1\}$$

and power set algebras

$$\oplus \equiv \cup \qquad \otimes \equiv \cap \qquad a,b,c \subset \mathbb{Z}$$

Other functions that do not preserve the above properties can also be defined for $\oplus$ and $\otimes$. For example, it is often useful for $\oplus$ or $\otimes$ to pull in other data, such as vertex indices of a graph.

VI. Matrix Properties

Associativity, distributivity, and commutativity are very powerful properties that enable the construction of composable graph algorithms (i.e., operations can be reordered with the knowledge that the answers will remain unchanged). Composability makes it easy to build a wide range of graph algorithms with just a few functions. Given matrices

$$\mathbf{A},\mathbf{B},\mathbf{C} \in \mathbb{S}^{m\times n}$$

let their elements be specified by

$$a = \mathbf{A}(i,j) \qquad b = \mathbf{B}(i,j) \qquad c = \mathbf{C}(i,j)$$

Commutativity, associativity, and distributivity of scalar operations translates into similar properties on matrix operations in the following manner.

Additive commutativity allows graphs to be swapped and combined via matrix element-wise addition (see Figure 4) without changing the result

$$a \oplus b = b \oplus a \qquad \Rightarrow \qquad \mathbf{A} \oplus \mathbf{B} = \mathbf{B} \oplus \mathbf{A}$$

where matrix element-wise addition is given by

$$\mathbf{C}(i,j) = \mathbf{A}(i,j) \oplus \mathbf{B}(i,j)$$

Multiplicative commutativity allows graphs to be swapped, intersected, and scaled via matrix element-wise multiplication (see Figure 5) without changing the result

$$a \otimes b = b \otimes a \qquad \Rightarrow \qquad \mathbf{A} \otimes \mathbf{B} = \mathbf{B} \otimes \mathbf{A}$$

where matrix element-wise (Hadamard) multiplication is given by

$$\mathbf{C}(i,j) = \mathbf{A}(i,j) \otimes \mathbf{B}(i,j)$$

Additive associativity allows graphs to be combined via matrix element-wise addition in any grouping without changing the result

$$(a \oplus b) \oplus c = a \oplus (b \oplus c) \quad \Rightarrow \quad (\mathbf{A} \oplus \mathbf{B}) \oplus \mathbf{C} = \mathbf{A} \oplus (\mathbf{B} \oplus \mathbf{C})$$

Multiplicative associativity allows graphs to be intersected and scaled via matrix element-wise multiplication in any grouping without changing the result

$$(a \otimes b) \otimes c = a \otimes (b \otimes c) \quad \Rightarrow \quad (\mathbf{A} \otimes \mathbf{B}) \otimes \mathbf{C} = \mathbf{A} \otimes (\mathbf{B} \otimes \mathbf{C})$$

Element-wise distributivity allows graphs to be intersected and/or scaled and then combined or vice versa without changing the result

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \;\Rightarrow\; \mathbf{A} \otimes (\mathbf{B} \oplus \mathbf{C}) = (\mathbf{A} \otimes \mathbf{B}) \oplus (\mathbf{A} \otimes \mathbf{C})$$

Matrix multiply distributivity allows graphs to be transformed via matrix multiply and then combined or vice versa without changing the result

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \quad \Rightarrow \quad \mathbf{A}(\mathbf{B} \oplus \mathbf{C}) = (\mathbf{A}\mathbf{B}) \oplus (\mathbf{A}\mathbf{C})$$

where matrix multiply

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

is given by

$$\mathbf{C}(i,j) = \bigoplus_{k=1}^{l} \mathbf{A}(i,k) \otimes \mathbf{B}(k,j)$$

for matrices with dimensions

$$\mathbf{A} : \mathbb{S}^{m\times l} \qquad \mathbf{B} : \mathbb{S}^{l\times m} \qquad \mathbf{C} : \mathbb{S}^{m\times n}$$

Matrix multiply associativity is another implication of scalar distributivity and allows graphs to be transformed via matrix multiplication in various orderings without changing the result

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c) \qquad \Rightarrow \qquad (\mathbf{A}\mathbf{B})\mathbf{C} = \mathbf{A}(\mathbf{B}\mathbf{C})$$

Matrix multiply commutativity can be achieved when combined with the transpose operation

$$(\mathbf{A}\mathbf{B})^{\mathsf{T}} = \mathbf{B}^{\mathsf{T}}\mathbf{A}^{\mathsf{T}}$$

where the transpose of a matrix is given by

$$\mathbf{A}^{\mathsf{T}}(j,i) = \mathbf{A}(i,j)$$

VII. 0-Element: No Graph Edge

Sparse matrices play an important role in graphs. Many implementations of sparse matrices reduce storage by not storing the 0-valued elements in the matrix. In adjacency matrices, the 0 element is equivalent to no edge from the vertex that is represented by the row to the vertex that is represented by the column. In incidence matrices, the 0 element is equivalent to the edge represented by the row not including the vertex that is represented by the column. In most cases, the 0 element is standard arithmetic 0, but in other cases it can be a different value. Nonstandard 0 values can be helpful when combined with different $\oplus$ and $\otimes$ operations. For example, in different

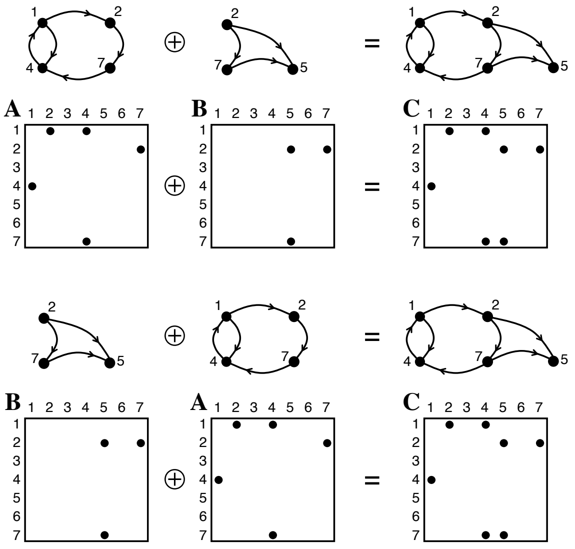

Fig. 4. Illustration of the commutative property of the element-wise addition of two graphs and their corresponding adjacency matrix representations.

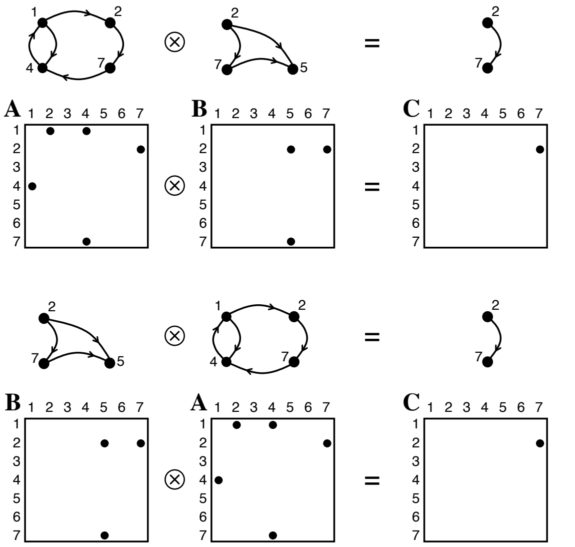

Fig. 5. Illustration of the commutative property of the element-wise multiplication of two graphs and their corresponding adjacency matrix representations.

contexts 0 might be $+\infty$, $-\infty$, or $\emptyset$ (empty set). For any value of 0, if the 0 element has certain properties with respect to scalar $\oplus$ and $\otimes$, then the sparsity of matrix operations can be managed efficiently. These properties are the additive identity

$$a \oplus 0 = a$$

and the multiplicative annihilator

$$a \otimes 0 = 0$$

Example combinations of $\oplus$ and $\otimes$ that exhibit the additive identity and multiplicative annihilator include

- standard arithmetic $(+.\times)$ on real numbers $\mathbb{R}$
- max-plus algebra $(\max.+)$ on real numbers with a defined minimal element $\{-\infty \cup \mathbb{R}\}$
- min-plus algebra $(\min.+)$ using real numbers with a defined maximal element $\{\mathbb{R} \cup \infty\}$
- max-min algebra $(\max.\min)$ using non-negative real numbers $[0, \infty)$
- min-max algebra $(\min.\max)]$ using non-positive real numbers $(-\infty, \leq 0]$
- max-min algebra $(\max.\min)$ using non-positive real numbers with a minimal element $\{-\infty \cup \mathbb{R}_{\leq 0}\}$
- min-max algebra $(\min.\max)$ using non-negative real numbers with a maximal element $\{\mathbb{R}_{\geq 0} \cup \infty\}$
- Galois field (xor.and) over a set of two numbers $\{0, 1\}$
- power set $(\cup.\cap)]$ on any subset of integers $\mathbb{Z}$

The above examples are a small selection of the operators and sets that are useful for building graph algorithms. Many more are possible. The ability to change the scalar values and operators while preserving the overall behavior of the graph operations is one of the principal benefits of using matrices for graph algorithms.

VIII. Matrix Graph Operations

The main benefit of a matrix approach to graphs is the ability to perform a wide range of graph operations on diverse types of graphs with a small number of matrix operations. These core matrix operations and some example graph operations they support are as follows

- building a sparse matrix from row, column, and value triples, which corresponds to constructing a graph from a set of out-vertices, in-vertices, and edge weights
- extracting the row, column, and value tuples corresponding to the nonzero elements in a sparse matrix, which corresponds to extracting graph edges from the matrix representation of a graph
- transposing the rows and the columns of a sparse matrix, which is equivalent to swapping the out-vertices and the in-vertices of a graph
- using matrix multiplication to perform single-source breadth-first search, multisource breadth-first search, and weighted breadth-first search on a graph
- extracting a sub-matrix from a larger matrix is equivalent to selecting a sub-graph from a larger graph
- assigning a matrix to a set of indices in a larger matrix inserts a sub-graph into a graph
- using element-wise addition of matrices and element-wise multiplication of matrices to perform graph union and intersection along with edge weight scaling and combining

The above collection of functions has been shown to be useful for implementing a wide range of graph algorithms. These functions strike a balance between providing enough functions to be useful to application builders while being few enough that they can be implemented effectively.

A. Building a Matrix: Edge List to Graph

Graph data can often be represented as triples of vectors $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{v}$ corresponding to the nonzero elements in the sparse matrix. Constructing an $m\times n$ sparse matrix from vector triples can be denoted

$$\mathbf{C} \;=\; \mathbb{S}^{m\times n}(\mathbf{i},\mathbf{j},\mathbf{v},\oplus)$$

where

$$\mathbf{i} : I^l \qquad \mathbf{j} : J^l \qquad \mathbf{v} : \mathbb{S}^l$$

are all l element vectors. The optional $\oplus$ operation defines how multiple entries with the same row and column are handled.

B. Extracting Tuples: Graph to Vertex List

Extracting the nonzero tuples from a sparse matrix can be denoted mathematically as

$$(\mathbf{i},\mathbf{j},\mathbf{v}) = \mathbf{A}$$

C. Transpose: Swap Out-Vertices and In-Vertices

Swapping the rows and columns of a sparse matrix is a common tool for changing the direction of vertices in a graph (see Figure 6). The transpose is denoted as

$$\mathbf{C} \;=\; \mathbf{A}^{\mathsf{T}}$$

or more explicitly

$$\mathbf{C}(j,i) \;=\; \mathbf{A}(i,j)$$

where $\mathbf{A} : \mathbb{S}^{m\times n}$ and $\mathbf{C} : \mathbb{S}^{n\times m}$

Transpose also can be implemented using triples as follows

$$(\mathbf{i},\mathbf{j},\mathbf{v}) = \mathbf{A}$$

$$\mathbf{C} = \mathbb{S}^{n\times m}(\mathbf{j},\mathbf{i},\mathbf{v})$$

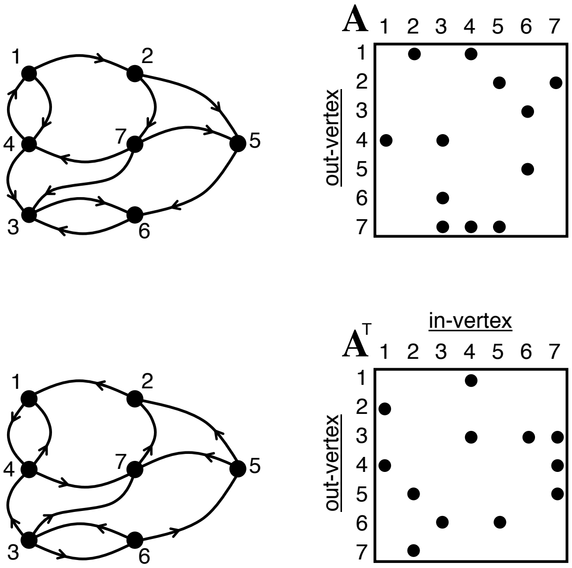

Fig. 6. Transposing the adjacency matrix of a graph switches the directions of its edges.

D. Matrix Multiplication: Breadth-First-Search, and Adjacency Matrix Construction

Matrix multiplication is the most important matrix operation and can be used to implement a wide range of graph algorithms. Examples include finding the nearest neighbors of a vertex (see Figure 7) and constructing an adjacency matrix from an incidence matrix (see Figure 8). In its most common form, matrix multiplication using standard arithmetic addition and multiplication is given by

$$\mathbf{C} = \mathbf{A}\mathbf{B}$$

or more explicitly

$$\mathbf{C}(i,j) = \sum_{k=1}^{l} \mathbf{A}(i,k)\mathbf{B}(k,j)$$

where

$$\mathbf{A} : \mathbb{R}^{m\times l} \qquad \mathbf{B} : \mathbb{R}^{l\times n} \qquad \mathbf{C} : \mathbb{R}^{m\times n}$$

Matrix multiplication has many important variants that include non-arithmetic addition and multiplication

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

where

$$\mathbf{A} : \mathbb{S}^{m\times l} \qquad \mathbf{B} : \mathbb{S}^{l\times n} \qquad \mathbf{C} : \mathbb{S}^{m\times n}$$

and the notation $\oplus.\otimes$ makes explicit that $\oplus$ and $\otimes$ can be other functions.

One of the most common uses of matrix multiplication is to construct an adjacency matrix from an incidence matrix representation of a graph. For a graph with out-vertex incidence matrix $\mathbf{E}_{\text{out}}$ and in-vertex incidence matrix $\mathbf{E}_{\text{in}}$, the corresponding adjacency matrix can be computed by

$$\mathbf{A} = \mathbf{E}^{\mathsf{T}}_{\text{out}}\mathbf{E}_{\text{in}}$$

where the individual values in $\mathbf{A}$ can be computed via

$$\mathbf{A}(i,j) = \bigoplus_k \mathbf{E}^{\mathsf{T}}_{\text{out}}(i,k) \otimes \mathbf{E}_{\text{in}}(k,j)$$

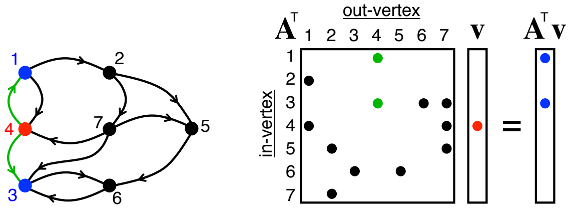

Fig. 7. (left) Breadth-first search of a graph starting at vertex 4 and traversing to vertices 1 and 3. (right) Matrix-vector multiplication of the adjacency matrix of a graph performs the equivalent operation.

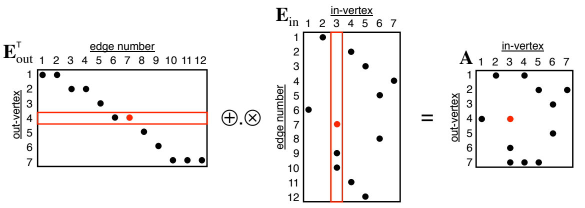

Fig. 8. Construction of an adjacency matrix of a graph from its incidence matrices via matrix-matrix multiply. The entry $\mathbf{A}(4,3)$ is obtained by combining the row vector $\mathbf{E}^{\mathsf{T}}_{\text{out}}(4,k)$ with the column vector $\mathbf{E}_{\text{in}}(k,3)$ via matrix-matrix product $\mathbf{A}(4,3) = \bigoplus_{k=1}^{12} \mathbf{E}^{\mathsf{T}}_{\text{out}}(4,k) \otimes \mathbf{E}_{\text{in}}(k,3)$.

E. Extract: Selecting Sub-graphs

Selecting sub-graphs is a very common graph operation (see Figure 9). This operation is performed by selecting out-vertices (row) and in-vertices (columns) from a matrix $\mathbf{A} : \mathbb{S}^{m \times n}$

$$\mathbf{C} \ = \ \mathbf{A}(\mathbf{i},\mathbf{j})$$

or more explicitly

$$\mathbf{C}(i,j) = \mathbf{A}(\mathbf{i}(i),\mathbf{j}(j))$$

where $i \in \{1,...,m_C\}$, $j \in \{1,...,n_C\}$, $\mathbf{i} : I^{m_C}$, and $\mathbf{j} : J^{m_C}$ select specific sets of rows and columns in a specific order. The resulting matrix $\mathbf{C} : \mathbb{S}^{m_C \times n_C}$ can be larger or smaller than the input matrix $\mathbf{A}$. This operation can also be used to replicate and/or permute rows and columns in a matrix.

Extraction can also be implemented with matrix multiplication as

$$\mathbf{C} = \mathbf{S}(\mathbf{i}) \ \mathbf{A} \ \mathbf{S}^{\mathsf{T}}(\mathbf{j})$$

where $\mathbf{S}(\mathbf{i})$ and $\mathbf{S}(\mathbf{j})$ are selection matrices given by

$$\mathbf{S}(\mathbf{i}) = \mathbb{S}^{m_C \times m}(\{1,...,m_C\},\mathbf{i},1)$$
$$\mathbf{S}(\mathbf{j}) = \mathbb{S}^{n_C \times n}(\{1,...,n_C\},\mathbf{j},1)$$

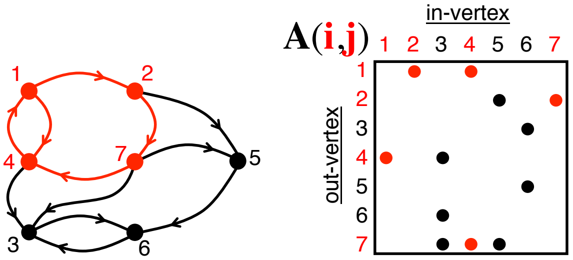

Fig. 9. Selection of a 4-vertex sub-graph from the adjacency matrix via selecting subsets of rows and columns $\mathbf{i} = \mathbf{j} = \{1,2,4,7\}$.

F. Assign: Modifying Sub-Graphs

Modifying sub-graphs is a very common graph operation. This operation is performed by selecting out-vertices (row) and in-vertices (columns) from a matrix $\mathbf{C} : \mathbb{S}^{m \times n}$ and assigning new values to them from another sparse matrix, $\mathbf{A} : \mathbb{S}^{m_A \times n_A}$

$$\mathbf{C}(\mathbf{i},\mathbf{j}) = \mathbf{A}$$

or more explicitly

$$\mathbf{C}(\mathbf{i}(i),\mathbf{j}(j)) = \mathbf{A}(i,j)$$

where $i \in \{1,...,m_A\}$, $j \in \{1,...,n_A\}$, $\mathbf{i} : I^{m_A}$ and $\mathbf{j} : J^{n_A}$ select specific sets of rows and columns.

G. Element-Wise Addition and Element-Wise Multiplication: Combining Graphs, Intersecting Graphs, and Scaling Graphs

Combining graphs along with adding their edge weights can be accomplished by adding together their sparse matrix representations

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

where $\mathbf{A}, \mathbf{B}, \mathbf{C} : \mathbb{S}^{m \times n}$ or more explicitly

$$\mathbf{C}(i,j) = \mathbf{A}(i,j) \oplus \mathbf{B}(i,j)$$

where $i \in \{1,...,m\}$, and $j \in \{1,...,n\}$.

Intersecting graphs along with scaling their edge weights can be accomplished by element-wise multiplication of their sparse matrix representations

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

where $\mathbf{A}, \mathbf{B}, \mathbf{C} : \mathbb{S}^{m \times n}$ or more explicitly

$$\mathbf{C}(i,j) = \mathbf{A}(i,j) \otimes \mathbf{B}(i,j)$$

where $i \in \{1,...,m\}$, and $j \in \{1,...,n\}$.

IX. Performance

A standard such as the GraphBLAS can only be effective if it does not impose unnecessary overhead on the computations it performs. One test of the overhead is to compare the GraphBLAS implementation to other standard sparse matrix libraries. Figure 10 shows the performance of one prototype GraphBLAS implementation compared to a state-of-the art GPU graph library (Gunrock) [Wang et al 2016].

The dataset used are random undirected Kronecker graphs with edge factor 32 and scale factor ranging from 16 to 21. Each experiment conducts a BFS starting from a high degree node in the graph. The GraphBLAS performance of sparse matrix - sparse vector multiplication is similar to Gunrock BFS performance. The similarity in performance indicates that the GraphBLAS is not introducing a high overhead. Each experiment is launched on these graphs from node 0 except on the scale 19 graph, which is launched from node 1. The runtime is an average of 10 runs to reduce variance.

We ran all experiments in this paper on a Linux workstation with 2× 3.50 GHz Intel 4-core E5-2637 v2 Xeon CPUs, 256 GB of main memory, and an NVIDIA K40c GPU with 12 GB on-board memory. The GPU programs were compiled with NVIDIA's nvcc compiler (version 7.5.17) using the `-O3` optimization level. The C code was compiled using gcc 4.8.5. All results ignore transfer time (from disk-to-memory and CPU-to-GPU). The Gunrock code was executed using the command-line configuration `--undirected --traversal-mode=1 --iteration-num=10`.

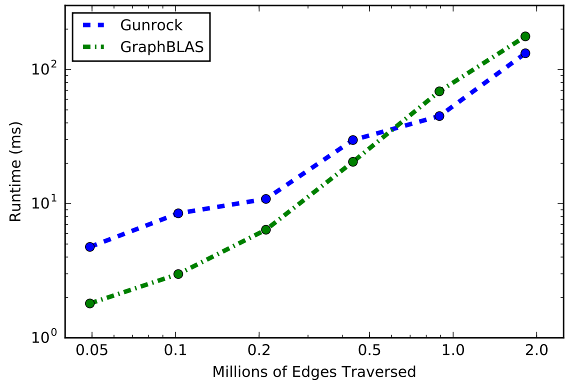

Fig. 10. Sparse matrix times sparse vector performance for a prototype GraphBLAS implementation as compared to an optimized GPU graph library (Gunrock) performing BFS in a similar manner.

Figure 11 shows the overhead of a second prototype GraphBLAS implementation, the GraphBLAS Template Library (GBTL)[Zhang et al 2016].We measured the GraphBLAS API overhead using the GraphBLAS Template Library (GBTL) on a machine with an Intel i5-4670k processor and a GTX660 CUDA-capable graphics card. The overhead results reflect the difference in runtime, in terms of percentages, between the CUDA backend of GBTL invoked using GraphBLAS API and the direct calling of underlying implementation. We obtain the numbers by averaging the overhead of 16 runs on Erdős-Rényi random graphs generated using the same dimension and sparsity. The code is compiled using the -O2 optimization level on version 7.5.18 of the CUDA toolkit with gcc 4.9.3. The results indicate that the overhead of the GraphBLAS is small compared to the underlying math being performed.

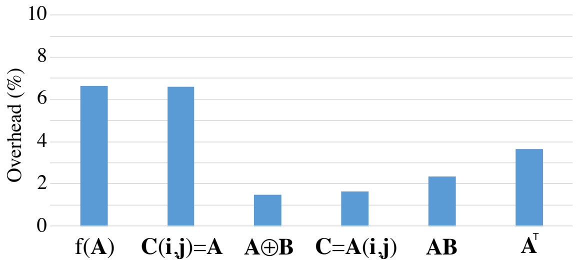

Fig. 11. Percentage overhead of the GraphBLAS Template Library prototype implementation on six different GraphBLAS operations.

X. Conclusions

Matrices are a powerful tool for representing and manipulating graphs. Adjacency matrices represent directed-weighted-graphs with each row and column in the matrix representing a vertex and the values representing the weights of the edges. Incidence matrices represent directed-weighted-multi-hyper-graphs with each row representing an edge and each column representing a vertex. Perhaps the most important aspects of matrix-based graphs are the mathematical properties of commutativity, associativity, and distributivity. These properties allow a very small number of matrix operations to be used to construct a large number of graphs. These properties of the matrix are determined by the element-wise properties of addition and multiplication on the values in the matrix. The GraphBLAS allows these matrix properties to be readily applied to graphs in a low-overhead manner.

Acknowledgments

The authors would like to thank Hedayat Alghassi, Michael Anderson, Ariful Azad, Muthu Baskaran, Paul Burkhardt, Steven Dalton, Tim Davis, Joe Eaton, Alan Edelman, Sterling Foster, Vijay Gadepally, Joseph Gonzalez, Torsten Hoefler, Erik Holk, Thejaka Kanewala, Tze Meng Low, Dave Martinez, John Matty, Asit Mishra, Samantha Misurda, Mostofa Patwary, Fabrizio Petrini, Albert Reuther, Jason Riedy, Victor Roytburd, Nadathur Satish, Narayanan Sundaram, Richard Veras, Michael Wolf, Albert-Jan Yzelman, Peter Zhang, and Xia Zhu.

References

[Anderson et al 2016] M. Anderson, N. Sundaram, N. Satish, M. Patwary, T. L. Willke, & P. Dubey, *GraphPad: Optimized Graph Primitives for Parallel and Distributed Platforms*, Proceedings of the IPDPS, 2016.

[Azad et al 2015] A. Azad, G. Ballard, A. Buluç, J. Demmel, L. Grigori, O. Schwartz, S. Toledo, & S. Williams, *Exploiting Multiple Levels of Parallelism in Sparse Matrix-Matrix Multiplication*, Technical Report 1510.00844.arXiv

[Ballard et al 2013] G. Ballard, A. Buluç, J. Demmel, L. Grigori, B. Lipshitz, O. Schwartz, & S. Toledo, *Communication optimal parallel multiplication of sparse random matrices*, In Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures (pp. 222-231), 2013

[Bergamini & Meyerhenke 2016] E. Bergamini & H. Meyerhenke, *Approximating Betweenness Centrality in Fully-dynamic Networks*. Accepted by Internet Mathematics. Taylor and Francis Group. To appear.

[Buluç & Gilbert 2011] A. Buluç & J. Gilbert, *The Combinatorial BLAS: Design, implementation, and applications*. International Journal of High Performance Computing Applications (IJHPCA), 2011

[Buluç & Gilbert 2012] A. Buluç & J. Gilbert, *Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments*, SIAM Journal on Scientific Computing 34.4 (2012): C170-C191

[Buluç 2015] A. Buluç, *GraphBLAS Special Session*, IEEE HPEC 2015, Waltham, MA

[Brualdi 1967] R.A. Brualdi, *Kronecker products of fully indecomposable matrices and of ultrastrong digraphs*, Journal of Combinatorial Theory, 2:135-139, 1967

[Chakrabarti 2004] D. Chakrabarti, Y. Zhan, and C. Faloutsos, *R-MAT: A recursive model for graph mining*. SIAM Data Mining, 2004.

[Ediger et al 2010] D. Ediger, K. Jiang, J. Riedy, and D.A. Bader, *Massive Streaming Data Analytics: A Case Study with Clustering Coefficients*, 4th Workshop on Multithreaded Architectures and Applications (MTAAP), Atlanta, GA, April 23, 2010

[Ediger et al 2011] D. Ediger, J. Riedy, H. Meyerhenke, and D.A. Bader, *Tracking Structure of Streaming Social Networks*, 5th Workshop on Multithreaded Architectures and Applications (MTAAP), Anchorage, AK, May 20, 2011

[Ediger et al 2012] D. Ediger, R. McColl, J. Riedy, and D.A. Bader, *STINGER: High Performance Data Structure for Streaming Graphs*, The IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, September 20-22, 2012

[Ediger & Bader 2013] D. Ediger and D.A. Bader, *Investigating Graph Algorithms in the BSP Model on the Cray XMT*, 7th Workshop on Multithreaded Architectures and Applications (MTAAP), Boston, MA, May 24, 2013

[Ekanadham et al 2016] K. Ekanadham, B. Horn, J. Jann, M. Kumar, J. Moreira, P. Pattnaik, M. Serrano, G. Tanase, H. Yu, *Graph programming interface (GPI): a linear algebra programming model for large scale graph computations*, Proceedings of the ACM International Conference on Computing Frontiers (CF'16), 72-81, 2016.

[Georganas et al 2014] E. Georganas, A. Buluç, J. Chapman, L. Oliker, D. Rokhsar and K. Yelick, *Parallel De Bruijn Graph Construction and Traversal for De Novo Genome Assembly*, Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14), 2014

[Harary & Tauth 1964] F. Harary & C.A. Tauth, *Connectedness of products of two directed graphs*, SIAM Journal on Applied Mathamatics, 14:250-254, 1966

[Harary 1969] F. Harary, *Graph Theory*, Reading:Addison-Wesley, 1969

[Hutchison et al 2015] D. Hutchison, J. Kepner, V. Gadepally, & A. Fuchs, *Graphulo implementation of server-side sparse matrix multiply in the Accumulo database*, IEEE High Performance Extreme Computing (HPEC) Conference, Walham, MA, September 2015.

[Kepner & Gilbert 2011] J. Kepner & J. Gilbert (editors), *Graph Algorithms in the Language of Linear Algebra*, SIAM Press, Philadelphia, 2011

[Kepner et al 2012] J. Kepner, W. Arcand, W. Bergeron, N. Bliss, R. Bond, C. Byun, G. Condon, K. Gregson, M. Hubbell, J. Kurz, A. McCabe, P. Michaleas, A. Prout, A. Reuther, A. Rosa & C. Yee, *Dynamic Distributed Dimensional Data Model (D4M) Database and Computation System*, ICASSP (International Conference on Acoustics, Speech, and Signal Processing), 2012, Kyoto, Japan

[Konig 1931] D. Konig, *Graphen und Matrizen (Graphs and Matrices)*, Matematikai Lapok, 38:116-119, 1931.

[Konig 1936] D. Konig, *Theorie der endlichen und unendlichen graphen (Theory of finite and infinite graphs)*, Leipzig:Akademie Verlag M.B.H., 1936; see Richard McCourt (Birkhauser 1990) for an english translation of this classic work

[Leskovec 2005] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos. *Realistic, mathematically tractable graph generation and evolution, using Kronecker multiplication*. European Conference on Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2005), Porto, Portugal, 2005

[Mattson et al 2013] T. Mattson, D. Bader, J. Berry, A. Buluç, J. Dongarra, C. Faloutsos, J. Feo, J. Gilbert, J. Gonzalez, B. Hendrickson, J. Kepner, C. Leiserson, A. Lumsdaine, D. Padua, S. Poole, S. Reinhardt, M. Stonebraker, S. Wallach, & A. Yoo, *Standards for Graph Algorithms Primitives*, IEEE HPEC 2013, Waltham, MA

[Mattson 2014a] T. Mattson, *Workshop on Graph Algorithms Building Blocks*, IPDPS 2014, Pheoniz, AZ

[Mattson 2014b] T. Mattson, *GraphBLAS Special Session*, IEEE HPEC 2014, Waltham, MA

[Mattson 2015] T. Mattson, *Workshop on Graph Algorithms Building Blocks*, IPDPS 2015, Hyderabad, India

[Mattson 2016] T. Mattson, *Workshop on Graph Algorithms Building Blocks*, IPDPS 2016, Chicago, IL

[McAndrew 1963] M.H. McAndrew, *On the product of directed graphs*, Proceedings of the American Mathematical Society, 14:600-606, 1963

[McAndrew 1965] M.H. McAndrew, *On the polynomial of a directed graph*, Proceedings of the American Mathematical Society, 16:303-309, 1965

[McLaughlin & Bader 2014a] A. McLaughlin and D.A. Bader, *Revisiting Edge and Node Parallelism for Dynamic GPU Graph Analytics*, 8th Workshop on Multithreaded Architectures and Applications (MTAAP), Phoenix, AZ, May 23, 2014

[McLaughlin & Bader 2014b] A. McLaughlin and D.A. Bader, *Scalable and High Performance Betweenness Centrality on the GPU*, Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'14), New Orleans, LA, November 16-21, 2014

[McLaughlin et al 2014] A. McLaughlin, J. Riedy, and D.A. Bader, *Optimizing Energy Consumption and Parallel Performance for Betweenness Centrality using GPUs*, The 18th Annual IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, September 9-11, 2014

[Meyerhenke et al 2015] H. Meyerhenke, P. Sanders, C. Schulz, *Parallel Graph Partitioning for Complex Networks*, 29th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2015)

[Riedy & Bader 2013] J. Riedy & D.A. Bader, *Multithreaded Community Monitoring for Massive Streaming Graph Data*, 7th Workshop on Multithreaded Architectures and Applications (MTAAP), Boston, MA, May 24, 2014

[Riedy et al 2012] J. Riedy, H. Meyerhenke, and D.A. Bader, *Scalable Multi-threaded Community Detection in Social Networks*, 6th Workshop on Multithreaded Architectures and Applications (MTAAP), Shanghai, China, May 25, 2012

[Sabadusi 1960] G. Sabadusi, *Graph multiplication*, Mathematische Zeitschrift, 72:446-457, 1960

[Solomonik et al 2013] E. Solomonik, A. Buluç, & J. Demmel, *Minimizing communication in all-pairs shortest paths*. In IEEE International Symposium on Parallel & Distributed Processing (IPDPS), 548-559, 2013

[Song et al 2010] W.S. Song, J. Kepner, H.T. Nguyen, J.I. Kramer, V. Gleyzer, J.R. Mann, A.H. Horst, L.L. Retherford, R.A. Bond, N.T. Bliss, E.I. Robinson, S. Mohindra, and J. Mullen, *3-D Graph Processor*, Workshop on High Performance Embedded Computing, September 2010

[Song et al 2013] W.S. Song, J. Kepner, V. Gleyzer, H.T. Nguyen, and J.I. Kramer, *Novel Graph Processor Architecture*, MIT Lincoln Laboratory Journal, vol. 20, no. 1, pp. 92-104, 2013

[Staudt & Meyerhenke 2016] C.L. Staudt & H. Meyerhenke, *Engineering Parallel Algorithms for Community Detection in Massive Networks*, IEEE Transactions on Parallel and Distributed Systems vol. 27, no. 1, pp. 171-184, 2016.

[Staudt et al 2016] C.L. Staudt, A. Sazonovs, H. Meyerhenke, *NetworKit: A Tool Suite for Large-scale Network Analysis*, Network Science, Cambridge University Press

[Teh & Yap 1964] H.H. Teh & H.D. Yap, *Some construction problems of homogeneous graphs*, Bulletin of the Mathematical Society of Nanying University, 164-196, 1964

[Van Loan 2000] C.F.V. Loan. *The ubiquitous Kronecker product*. Journal of Computation and Applied Mathematics, 123(1-2):85–100, 2000

[Wang et al 2016] Y. Wang, A. Davidson, Y. Pan, Yuduo Wu, A. Riffel & J.D. Owens, *Gunrock: A high-performance graph processing library on the GPU*, 21th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2016, March 2016

[Weischel 1962] P.M. Weischel. *The Kronecker product of graphs*, Proceedings of the American Mathematical Society, 13(1):47–52, 1962

[Zhang et al 2016] P. Zhang, M. Zalewski, A. Lumsdaine, S. Misurda, & S. McMillan, *GBTL-CUDA: Graph Algorithms and Primitives for GPUs*, GABB workshop at IPDPS 2016