Building Blocks for Graph Algorithms in the Language of Linear Algebra

John R. Gilbert
University of California, Santa Barbara

with: David Bader (Georgia Tech), Aydı̇n Bulu̇ç (LBNL), Jeremy Kepner (MIT-LL),
Tim Mattson (Intel), Henning Meyerhenke (Karlsruhe)

CIMI Workshop on Innovative Clustering Methods
July 7, 2015

Support at UCSB: Intel, Microsoft, DOE Office of Science, NSF
Outline

- Motivation: Graph applications
- Mathematics: Sparse matrices for graph algorithms
- Software: CombBLAS, KDT, QuadMat
- Standards: The Graph BLAS effort
Computational models of the physical world

- Cortical bone
- Trabecular bone
Large graphs are everywhere...

- Internet structure
- Social interactions
- Scientific datasets: biological, chemical, cosmological, ecological, ...

WWW snapshot, courtesy Y. Hyun

Yeast protein interaction network, courtesy H. Jeong
Social network analysis (1993)

Co-author graph from 1993 Householder symposium
Facebook graph: 
> 1,000,000,000 vertices
The middleware challenge for graph analysis

- Continuous physical modeling
  - Linear algebra
    - Computers

- Discrete structure analysis
  - Graph theory
    - Computers
34 Petaflops (Top500) \[ P A = L \times U \] 24 Terateps (Graph500)

34 Peta / 24 Tera is about 1,400
Dense linear algebra vs. graphs, November 2014

34 Petaflops (Top500)

\[
P A = L x U
\]

24 Terateps (Graph500)

Nov 2014: 34 Peta / 24 Tera \(\sim 1,400\)

Nov 2010: 2.5 Peta / 6.6 Giga \(\sim 380,000\)
The middleware challenge for graph analysis

- By analogy to numerical scientific computing. . .

- What should the combinatorial BLAS look like?

Basic Linear Algebra Subroutines (BLAS):
   Ops/Sec vs. Matrix Size

\[
C = A \times B
\]

\[
y = A \times x
\]

\[
\mu = x^T y
\]
Coarse-grained parallelism can be exploited by abstractions at the right level.

<table>
<thead>
<tr>
<th>Vertex/edge graph computations</th>
<th>Graphs in the language of linear algebra</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unpredictable, data-driven communication patterns</td>
<td>Fixed communication patterns</td>
</tr>
<tr>
<td>Irregular data accesses, with poor locality</td>
<td>Matrix block operations exploit memory hierarchy</td>
</tr>
<tr>
<td>Fine grained data accesses, dominated by latency</td>
<td>Coarse grained parallelism, limited by bandwidth not latency</td>
</tr>
</tbody>
</table>
Sparse array primitives for graphs

Sparse matrix-sparse matrix multiplication

Sparse matrix-sparse vector multiplication

Element-wise operations

Sparse matrix indexing

Matrices over various semirings: (+, ×), (and, or), (min, +), ...
Multiple-source breadth-first search

\[ A^T \]

\[ B \]
Multiple-source breadth-first search

- Sparse array representation $\Rightarrow$ space efficient
- Sparse matrix-matrix multiplication $\Rightarrow$ work efficient
- Three possible levels of parallelism: searches, vertices, edges
Examples of semirings in graph algorithms

<table>
<thead>
<tr>
<th>( “values”: edge/vertex attributes, “add”: vertex data aggregation, “multiply”: edge data processing )</th>
<th>General schema for user-specified computation at vertices and edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real field: (R, +, *)</td>
<td>Numerical linear algebra</td>
</tr>
<tr>
<td>Boolean algebra: ({0 1},</td>
<td>, &amp;)</td>
</tr>
<tr>
<td>Tropical semiring: (R U {∞}, min, +)</td>
<td>Shortest paths</td>
</tr>
<tr>
<td>(S, select, select)</td>
<td>Select subgraph, or contract nodes to form quotient graph</td>
</tr>
</tbody>
</table>
Graph contraction via sparse triple product

Contract

[Diagram of graph contraction]

[Matrix multiplication calculation]

[Result of graph contraction]
Subgraph extraction via sparse triple product

Extract

\[
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 &  & 1 &  &  &  \\
2 &  &  &  &  &  \\
3 &  &  &  &  &  \\
\end{array}
\times
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 &  &  &  &  &  \\
2 &  &  &  &  &  \\
3 &  &  &  &  &  \\
\end{array}
\times
\begin{array}{cccccc}
1 & 2 & 3 & 4 & 5 & 6 \\
1 &  &  &  &  &  \\
2 &  &  &  &  &  \\
3 &  &  &  &  &  \\
\end{array}
= \begin{array}{cc}
\cdot & \cdot \\
\cdot & \cdot \\
\cdot & \cdot \\
\end{array}
\]
Counting triangles (clustering coefficient)

Clustering coefficient:

- \( \Pr(\text{wedge } i-j-k \text{ makes a triangle with edge } i-k) \)
- \( 3 \times \frac{\# \text{ triangles}}{\# \text{ wedges}} \)
- \( 3 \times \frac{4}{19} = 0.63 \) in example
- may want to compute for each vertex } j
Counting triangles (clustering coefficient)

Cluster coefficient:
- \( \text{Pr} \) (wedge i-j-k makes a triangle with edge i-k)
- \( 3 \times \frac{\# \text{triangles}}{\# \text{wedges}} \)
- \( 3 \times \frac{4}{19} = 0.63 \) in example
- may want to compute for each vertex j

“Cohen’s” algorithm to count triangles:
- Count triangles by lowest-degree vertex.
- Enumerate “low-hinged” wedges.
- Keep wedges that close.
Counting triangles (clustering coefficient)

\[ A = L + U \quad \text{(hi->lo + lo->hi)} \]

\[ L \times U = B \quad \text{(wedge, low hinge)} \]

\[ A \wedge B = C \quad \text{(closed wedge)} \]

\[ \text{sum}(C)/2 = 4 \text{ triangles} \]
Combinatorial BLAS

http://gauss.cs.ucsb.edu/~aydin/CombBLAS

An extensible distributed-memory library offering a small but powerful set of linear algebraic operations specifically targeting graph analytics.

- Aimed at graph algorithm designers/programmers who are not expert in mapping algorithms to parallel hardware.
- Flexible, templated C++ interface.
- Scalable performance from laptop to 100,000-processor HPC.
- Open source software, version 1.4.0 released Jan 2014.
Combinatorial BLAS in distributed memory

Combinatorial BLAS functions and operators

DistMat

CommGrid

FullyDistVec

DenseDistMat

SpDistMat

SpMat

SpDistVec

DenseDistVec

DCSC

CSC

Triples

CSB

HAS A

Polymorphism

Enforces interface only
Matrix/vector distributions, interleaved on each other.

Default distribution in **Combinatorial BLAS**.

Scalable with increasing number of processes

- 2D matrix layout wins over 1D with large core counts and with limited bandwidth/compute
- 2D vector layout sometimes important for load balance

2D Layout for Sparse Matrices & Vectors
Combinatorial BLAS was fastest among all tested graph processing frameworks on 3 out of 4 benchmarks in an independent study by Intel.

Satish et al. "Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets", in SIGMOD’14
Combinatorial BLAS “users” (Sep 2014)

- IBM (T. J. Watson, Zurich, & Tokyo)
- Intel
- Cray
- Microsoft
- Stanford
- UC Berkeley
- Carnegie-Mellon
- Georgia Tech
- Ohio State
- U Texas Austin
- Columbia
- U Minnesota
- NC State
- UC Santa Barbara
- UC San Diego

- Berkeley Natl Lab
- Sandia Natl Labs
- SEI
- Paradigm4
- IHPC (Singapore)
- King Fahd U (Saudi Arabia)
- Tokyo Inst of Technology
- Chinese Academy of Sciences
- U Ghent (Belgium)
- Bilkent U (Turkey)
- U Canterbury (New Zealand)
- Purdue
- Indiana U
- UC Merced
- Mississippi State
Knowledge Discovery Toolbox
http://kdt.sourceforge.net/

- Aimed at domain experts who know their problem well but don’t know how to program a supercomputer
- Easy-to-use Python interface
- Runs on a laptop as well as a cluster with 10,000 processors
- Open source software (New BSD license)

A general graph library with operations based on linear algebraic primitives
Attributed semantic graphs and filters

Example:

- **Vertex types:** Person, Phone, Camera, Gene, Pathway
- **Edge types:** PhoneCall, TextMessage, CoLocation, SequenceSimilarity
- **Edge attributes:** Time, Duration

- Calculate centrality just for emails among engineers sent between given start and end times

```python
def onlyEngineers(self):
    return self.position == Engineer

def timedEmail(self, sTime, eTime):
    return ((self.type == email) and (self.Time > sTime) and (self.Time < eTime))

G.addVFilter(onlyEngineers)
G.addEFilter(timedEmail(start, end))

# rank via centrality based on recent email transactions among engineers
bc = G.rank('approxBC')
```
SEJITS for filter/semiring acceleration

- **Standard KDT**
  - KDT Algorithm
  - Filter (Py)
  - Semiring (Py)

- **KDT+SEJITS**
  - KDT Algorithm
  - Filter (C++)
  - Semiring (C++)
  - SEJITS Translation

**Embedded DSL: Python for the whole application**
- Introspect, translate Python to equivalent C++ code
- Call compiled/optimized C++ instead of Python
Filtered BFS with SEJITS

Time (in seconds) for a single BFS iteration on scale 25 RMAT (33M vertices, 500M edges) with 10% of elements passing filter. Machine is NERSC’s Hopper.
A few other graph algorithms we’ve implemented in linear algebraic style

- Maximal independent set (KDT/SEJITS) [BDFGKLOW 2013]
- Peer-pressure clustering (SPARQL) [DGLMR 2013]
- Time-dependent shortest paths (CombBLAS) [Ren 2012]
- Gaussian belief propagation (KDT) [LABGRTW 2011]
- Markoff clustering (CombBLAS, KDT) [BG 2011, LABGRTW 2011]
- Betweenness centrality (CombBLAS) [BG 2011]
- Geometric mesh partitioning (Matlab 😊) [GMT 1998]
Graph algorithms in the language of linear algebra

- Kepner et al. study [2006]: fundamental graph algorithms including min spanning tree, shortest paths, independent set, max flow, clustering, ...
- SSCA#2 / centrality [2008]
- Basic breadth-first search / Graph500 [2010]
- Beamer et al. [2013] direction-optimizing breadth-first search, implemented in CombBLAS
The (original) BLAS

The Basic Linear Algebra Subroutines had a revolutionary impact on computational linear algebra.

<table>
<thead>
<tr>
<th>BLAS  1</th>
<th>vector ops</th>
<th>Lawson, Hanson, Kincaid, Krogh, 1979</th>
<th>LINPACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLAS  2</td>
<td>matrix-vector ops</td>
<td>Dongarra, Du Croz, Hammarling, Hanson, 1988</td>
<td>LINPACK on vector machines</td>
</tr>
<tr>
<td>BLAS  3</td>
<td>matrix-matrix ops</td>
<td>Dongarra, Du Croz, Duff, Hammarling, 1990</td>
<td>LAPACK on cache based machines</td>
</tr>
</tbody>
</table>

- Experts in mapping algorithms to hardware tune BLAS for specific platforms.
- Experts in numerical linear algebra build software on top of the BLAS to get high performance “for free.”

Today every computer, phone, etc. comes with `/usr/lib/libblas`
Can we standardize a “Graph BLAS”? 
Can we standardize a “Graph BLAS”? 

**No**, it’s not reasonable to define a universal set of building blocks.

- Huge diversity in matching graph algorithms to hardware platforms.
- No consensus on data structures or linguistic primitives.
- Lots of graph algorithms remain to be discovered.
- Early standardization can inhibit innovation.
Yes, it is reasonable to define a universal set of building blocks…

… for graphs as linear algebra.

- Representing graphs in the language of linear algebra is a mature field.
- Algorithms, high level interfaces, and implementations vary.
- But the core primitives are well established.
The Graph BLAS effort

- Manifesto, HPEC 2013:

Stands 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 Mellon 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 (Cothy Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

Abstract-- 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.

- Workshops at IPDPS & HPEC – next in Sept 2015
- Periodic working group telecons and meetings
- Graph BLAS Forum:  http://graphblas.org
### Sparse array attribute survey

<table>
<thead>
<tr>
<th>Function</th>
<th>Graph BLAS</th>
<th>Comb BLAS</th>
<th>Sparse BLAS</th>
<th>STINGER</th>
<th>D4M</th>
<th>SciDB</th>
<th>Tensor Toolbox</th>
<th>Julia</th>
<th>Graph Lab</th>
<th>PBGL</th>
<th>Networ Kit</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
<td>1.4.0</td>
<td>2006</td>
<td>r633</td>
<td>2.5</td>
<td>13.9</td>
<td>2.5</td>
<td>0.2.0</td>
<td>2.2</td>
<td>2-1.0</td>
<td>3.3</td>
<td></td>
</tr>
<tr>
<td><strong>Language</strong></td>
<td>any</td>
<td>C++</td>
<td>F,C,C++,+</td>
<td>C</td>
<td>Matlab</td>
<td>C++</td>
<td>Matlab,Julia</td>
<td>C++</td>
<td>C++</td>
<td>C++</td>
<td></td>
</tr>
<tr>
<td><strong>Dim.</strong></td>
<td>2</td>
<td>1, 2</td>
<td>2</td>
<td>1-3</td>
<td>2</td>
<td>1-100</td>
<td>2, 3</td>
<td>1,2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td><strong>Index Base</strong></td>
<td>0 or 1</td>
<td>0</td>
<td>0 or 1</td>
<td>0</td>
<td>1</td>
<td>±N</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td><strong>Index Type</strong></td>
<td>uint64</td>
<td>any</td>
<td>int</td>
<td>int64</td>
<td>double, string</td>
<td>int64</td>
<td>double</td>
<td>any int</td>
<td>uint64</td>
<td>any int</td>
<td>uint64</td>
</tr>
<tr>
<td><strong>Value Type</strong></td>
<td>?</td>
<td>user</td>
<td>single, double,complex</td>
<td>int64</td>
<td>bool, string, double, complex</td>
<td>user</td>
<td>bool, double, complex</td>
<td>user</td>
<td>user</td>
<td>user</td>
<td>double</td>
</tr>
<tr>
<td><strong>Null</strong></td>
<td>0</td>
<td>user</td>
<td>0</td>
<td>0</td>
<td>≤0</td>
<td>null</td>
<td>0</td>
<td>0</td>
<td>int64(-1)</td>
<td>user</td>
<td>uint64(-1)</td>
</tr>
<tr>
<td><strong>Sparse Format</strong></td>
<td>?</td>
<td>tuple</td>
<td>undef</td>
<td>linked list</td>
<td>dense, csc, tuple</td>
<td>RLE</td>
<td>dense, csc</td>
<td>csc</td>
<td>csr/csc</td>
<td>linked list,csr, map</td>
<td>csr</td>
</tr>
<tr>
<td><strong>Parallel</strong></td>
<td>?</td>
<td>2D block</td>
<td>none</td>
<td>block</td>
<td>arbitrary</td>
<td>ND*</td>
<td>none</td>
<td>ND*</td>
<td>Edge*</td>
<td>1D (vertex)</td>
<td>Iterator and loop based</td>
</tr>
<tr>
<td><strong>+ op</strong></td>
<td>user?</td>
<td>user</td>
<td>+</td>
<td>user</td>
<td>+,*</td>
<td>user</td>
<td>+</td>
<td>user</td>
<td>user</td>
<td>user</td>
<td>user</td>
</tr>
<tr>
<td>*** op**</td>
<td>user?</td>
<td>user</td>
<td>*</td>
<td>user</td>
<td>user</td>
<td>user</td>
<td>user</td>
<td>user</td>
<td>user</td>
<td>user</td>
<td>+</td>
</tr>
</tbody>
</table>
Some Graph BLAS basic functions
(names not final)

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Matlab notation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>matmul</strong> (SpGEMM)</td>
<td>- sparse matrices A and B</td>
<td>sparse matrix</td>
<td>C = A * B</td>
</tr>
<tr>
<td></td>
<td>- optional unary functs</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>matvec</strong> (SpM{Sp}V)</td>
<td>- sparse matrix A</td>
<td>sparse/dense vector</td>
<td>y = A * x</td>
</tr>
<tr>
<td></td>
<td>- sparse/dense vector x</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ewisemult, add, ...</strong> (SpE WiseX)</td>
<td>- sparse matrices or vectors</td>
<td>in place or sparse</td>
<td>C = A .* B</td>
</tr>
<tr>
<td></td>
<td>- binary funct, optional unarys</td>
<td>matrix/vector</td>
<td>C = A + B</td>
</tr>
<tr>
<td><strong>reduce</strong> (Reduce)</td>
<td>- sparse matrix A and funct</td>
<td>dense vector</td>
<td>y = sum(A, op)</td>
</tr>
<tr>
<td><strong>extract</strong> (SpRef)</td>
<td>- sparse matrix A</td>
<td>sparse matrix</td>
<td>B = A(p, q)</td>
</tr>
<tr>
<td></td>
<td>- index vectors p and q</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>assign</strong> (SpAsgn)</td>
<td>- sparse matrices A and B</td>
<td>none</td>
<td>A(p, q) = B</td>
</tr>
<tr>
<td></td>
<td>- index vectors p and q</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>buildMatrix</strong> (Sparse)</td>
<td>- list of edges/triples (i, j, v)</td>
<td>sparse matrix</td>
<td>A = sparse(i, j, v, m, n)</td>
</tr>
<tr>
<td><strong>getTuples</strong> (Find)</td>
<td>- sparse matrix A</td>
<td>edge list</td>
<td>[i, j, v] = find(A)</td>
</tr>
</tbody>
</table>
Matrix times matrix over semiring

**Inputs**
- matrix A: $S^{M\times N}$ (sparse or dense)
- matrix B: $S^{N\times L}$ (sparse or dense)

**Optional Inputs**
- matrix C: $S^{M\times L}$ (sparse or dense)
- scalar “add” function $\oplus$
- scalar “multiply” function $\otimes$
- transpose flags for A, B, C

**Outputs**
- matrix C: $S^{M\times L}$ (sparse or dense)

**Implements**

\[
C \oplus= A \oplus \otimes B
\]

for $j = 1 : N$

\[
C(i,k) = C(i,k) \oplus (A(i,j) \otimes B(j,k))
\]

If input C is omitted, implements

\[
C = A \oplus \otimes B
\]

Transpose flags specify operation on $A^T, B^T, \text{and/or } C^T$ instead

**Notes**
- $S$ is the set of scalars, user-specified
- $S$ defaults to IEEE double float
- $\oplus$ defaults to floating-point $+$
- $\otimes$ defaults to floating-point $*$

**Specific cases and function names:**
- SpGEMM: sparse matrix times sparse matrix
- SpMSpV: sparse matrix times sparse vector
- SpMV: sparse matrix times dense vector
- SpMM: sparse matrix times dense matrix
Sparse matrix indexing & assignment

Inputs
matrix \( \mathbf{A} \): \( \mathbb{S}^{M \times N} \) (sparse)
matrix \( \mathbf{B} \): \( \mathbb{S}^{|p| \times |q|} \) (sparse)
vector \( p \subseteq \{1, \ldots, M\} \)
vector \( q \subseteq \{1, \ldots, N\} \)

Optional Inputs
none

Outputs
matrix \( \mathbf{A} \): \( \mathbb{S}^{M \times N} \) (sparse)
matrix \( \mathbf{B} \): \( \mathbb{S}^{|p| \times |q|} \) (sparse)

SpRef Implements \( \mathbf{B} = \mathbf{A}(p,q) \)

\[
\text{for } i = 1 : |p| \\
\quad \text{for } j = 1 : |q| \\
\quad \quad \mathbf{B}(i,j) = \mathbf{A}(p(i),q(j))
\]

SpAsgn Implements \( \mathbf{A}(p,q) = \mathbf{B} \)

\[
\text{for } i = 1 : |p| \\
\quad \text{for } j = 1 : |q| \\
\quad \quad \mathbf{A}(p(i),q(j)) = \mathbf{B}(i,j)
\]

Notes
\( \mathbb{S} \) is the set of scalars, user-specified
\( \mathbb{S} \) defaults to IEEE double float
\( |p| \) = length of vector \( p \)
\( |q| \) = length of vector \( q \)

Specific cases and function names
SpRef: get sub-matrix
SpAsgn: assign to sub-matrix
Element-wise operations

Inputs
 matrix $A: \mathbb{S}^{M\times N}$ (sparse or dense)
 matrix $B: \mathbb{S}^{M\times N}$ (sparse or dense)

Optional Inputs
 matrix $C: \mathbb{S}^{M\times N}$ (sparse or dense)
 scalar “add” function $\oplus$
 scalar “multiply” function $\otimes$

Outputs
 matrix $C: \mathbb{S}^{M\times N}$ (sparse or dense)

Implements
 $C \oplus= A \otimes B$

for $i = 1 : M$
 for $j = 1 : N$
 $C(i,j) = C(i,j) \oplus (A(i,j) \otimes B(i,j))$

If input $C$ is omitted, implements $C = A \otimes B$

Specific cases and function names:
SpEWiseX: matrix elementwise
$M=1$ or $N=1$: vector elementwise
Scale: when $A$ or $B$ is a scalar

Notes
 $\mathbb{S}$ is the set of scalars, user-specified
 $\mathbb{S}$ defaults to IEEE double float
 $\oplus$ defaults to floating-point $+$
 $\otimes$ defaults to floating-point $*$
Inputs
matrix $A$: $S^{MxN}$ (sparse or dense)

Optional Inputs
matrix $C$: $S^{MxN}$ (sparse or dense)
scalar “add” function $\oplus$
unary function $f()$

Outputs
matrix $C$: $S^{MxN}$ (sparse or dense)

Implements $C \oplus f(A)$

for $i = 1 : M$
  for $j = 1 : N$
    if $A(i,j) \neq 0$
      $C(i,j) = C(i,j) \oplus f(A(i,j))$

If input $C$ is omitted, implements $C = f(A)$

Notes
$S$ is the set of scalars, user-specified
$S$ defaults to IEEE double float
$\oplus$ defaults to floating-point $+$

Specific cases and function names:
Apply: matrix apply
M=1 or N=1: vector apply
Matrix / vector reductions

**Inputs**
matrix $A$: $\mathbb{S}^{M \times N}$ (sparse or dense)

**Optional Inputs**
vector $c$: $\mathbb{S}^M$ or $\mathbb{S}^N$ (sparse or dense)
scalar “add” function $\oplus$
dimension $d$: 1 or 2

**Outputs**
matrix $c$: $\mathbb{S}^{M \times N}$ (sparse or dense)

**Implements**
$c(i) \oplus = \bigoplus_j A(i,j)$

for $i = 1 : M$
  for $j = 1 : N$
    $c(i) = c(i) \oplus fA(i,j)$

If input $C$ is omitted, implements
$c(i) = \bigoplus_j A(i,j)$

**Notes**
$\mathbb{S}$ is the set of scalars, user-specified
$\mathbb{S}$ defaults to IEEE double float
$\oplus$ defaults to floating-point +
d $d$ defaults to 2

**Specific cases and function names:**
Reduce ($d = 1$): reduce matrix to row vector
Reduce ($d = 2$): reduce matrix to col vector
A few questions for the Graph BLAS Forum

• How does the API let the user specify the “semiring scalar” objects and operations?
  – How general can the objects be? Sets, lists, etc.?
  – Different levels of compliance for an implementation, beginning with just (double, +, *)?

• How does the API let the user “break out of the BLAS”?
  – In numeric BLAS and in sparse Matlab (but not in Sparse BLAS), the user can access the matrix directly, element-by-element, with a performance penalty.
    – “for each edge e incident on vertex v do ...”
    – “add / delete edge e”
Conclusions

• Graph analysis presents challenges in:
  – Performance (both serial and parallel)
  – Software complexity
  – Interoperability

• Implementing graph algorithms using matrix-based approaches provides several useful solutions to these challenges.

• Researchers from Intel, IBM, Nvidia, LBL, MIT, UCSB, GaTech, KIT, etc. have joined together to create the Graph BLAS standard.

• Several implementations in progress:
  – C++: CombBLAS (LBL, UCSB), GraphMAT (Intel)
  – C: Graph Programming Interface (IBM), Stinger (GaTech)
  – Java: Graphulo (MIT)
  – Python: NetworkKit (KIT)
Thanks …

Thank You!

Continuous physical modeling

Discrete structure analysis

Linear algebra & graph theory

Computers

http://graphblas.org