GABB'14: Graph Algorithms Building Blocks workshop

http://www.graphanalysis.org/workshop2014.html

Workshop chair:

- Tim Mattson, *Intel Corp*

Steering committee:

- Jeremy Kepner, *MIT Lincoln Labs*
- John Gilbert, *UC Santa Barbara*
- David A. Bader, *Georgia Institute of Technology*
- Aydın Buluç, *LBNL*

Goals for the day

- Graph Algorithm Building Blocks (GABB):
  - A series of workshops exploring the fundamental building blocks of graph algorithms.
    - GABB’14 is composed of invited talks.
      - Speakers selected to put a wide range of viewpoints “on the table”.
    - GABB’XY (XY>14) will be “contributed papers” workshops.
      - Stay tuned for CFPs.
- We want GABB to be a real workshop … not a bunch of boring talks.
  - Please interact with the speakers.
  - Ask lots of questions, challenge assumptions, and help move the debate forward.

Schedule

09:00 - 09:30	Tim Mattson / Intel	Welcome, Goals, and a bit of Math
09:30 - 10:00	John Gilbert / UCSB	Examples and applications of graph algorithms in the language of linear algebra
10:00-10:30	Break	
10:30 - 11:00	Joseph Gonzalez / UC Berkeley	GraphX and Linear Algebra
11:00 - 11:30	David Mizell and Steven P. Reinhardt/ YarcData	Effective Graph-algorithmic Building Blocks for Graph Databases
11:30 - 12:00	Vijay Gadepally and Jeremy Kepner /MIT	Adjacency Matrices, Incidence Matrices, and Database Schemas
12:00 - 01:30	Lunch	
01:30 - 02:00	Dylan Stark / Sandia Nat Lab.	Graph Exploration: to Linear Algebra (and Beyond?)
02:00 - 02:30	Jason Riedy/ GaTech	Multi-threaded graph streaming
02:30 - 03:00	Saeed Maleki, G. Carl Evans, and David Padua/ UIUC	Linear algebra operator extensions for graph algorithms
03:00 - 03:30	Break	
03:30 - 04:00	Aydin Buluç, et. al. / LLBL and UC Berkeley	Communication-Avoiding Linear-Algebraic Primitives for Graph Analytics
04:00 - 04:30	Andrew Lumsdaine /Indiana University	Standardization: Lessons Learned
04:30 - 05:00	Panel	What's next for the "BLAS of Graph Algorithms"?

GraphBLAS: Motivation and Mathematical Foundations

http://istc-bigdata.org/GraphBlas/

Tim Mattson

Intel Labs

timothy.g.mattson@intel.com

… and the "GraphBLAS gang":

David Bader (GATech), Aydın Buluç (LBNL),
John Gilbert (UCSB), Joseph Gonzalez (UCB),
Jeremy Kepner (MIT Lincoln Labs)

Outline

➡ • Introduction: Graphs and Linear Algebra

• The Draft GraphBLAS primitives

• Conclusion/Summary

Motivation: History of BLAS

- BLAS: The Basic Linear Algebra subroutines

BLAS 1	$y \leftarrow \alpha x + y$	Lawson, Hanson, Kincaid and Krogh, 1979	LINPACK
BLAS 2	$y \leftarrow \alpha A x + \beta y$	Dongarra, Du Croz, Hammarling and Hanson, 1988	LINPACK on vector machines
BLAS 3	$C \leftarrow \alpha A B + \beta C$	Dongarra, Du Croz, Hammarling and Hanson, 1990	LAPACK on cache based machines

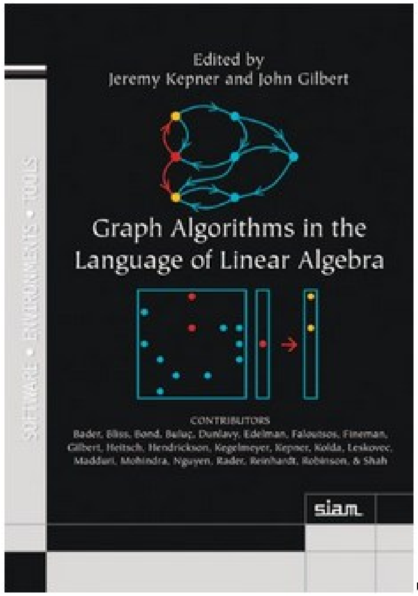
- The BLAS supported a separation of concerns:
  - HW/SW optimization experts tuned the BLAS for specific platforms.
  - Linear algebra experts built software on top of the BLAS .. high performance "for free".

- It is difficult to overestimate the impact of the BLAS … they revolutionized the practice of computational linear algebra.

Can we standardize "the BLAS" of graph algorithms

- No, it is not reasonable to define a common set of graph algorithm building blocks:
  - Matching Algorithms to the hardware platform results in too much diversity to support a common set of "graph BLAS".
  - There is little agreement on how to represent graph algorithms and data structures.
  - Early standardization can inhibit innovation by locking in a sub-optimum status quo

- Yes, it is reasonable to define a common set of graph algorithm building blocks … for Graphs in the language of Linear algebra.
  - Representing graphs in the language of linear algebra is a mature field … the algorithms, high level interfaces, and implementations vary, but the core primitives are well established .


Edited by
Jeremy Kepner and John Gilbert

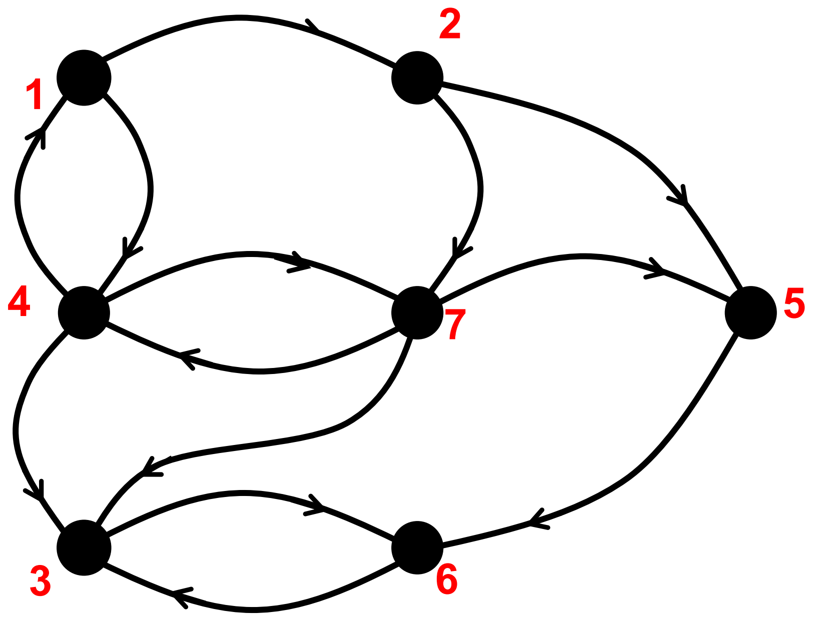
Graph Algorithms in the Language of Linear Algebra

CONTRIBUTORS
Bader, Bliss, Bond, Buluç, Dunlavy, Edelman, Faloutsos, Fineman, Gilbert, Heitsch, Hendrickson, Kegelmeyer, Kepner, Kolda, Leskovec, Madduri, Mohindra, Nguyen, Rader, Reinhardt, Robinson, & Shah

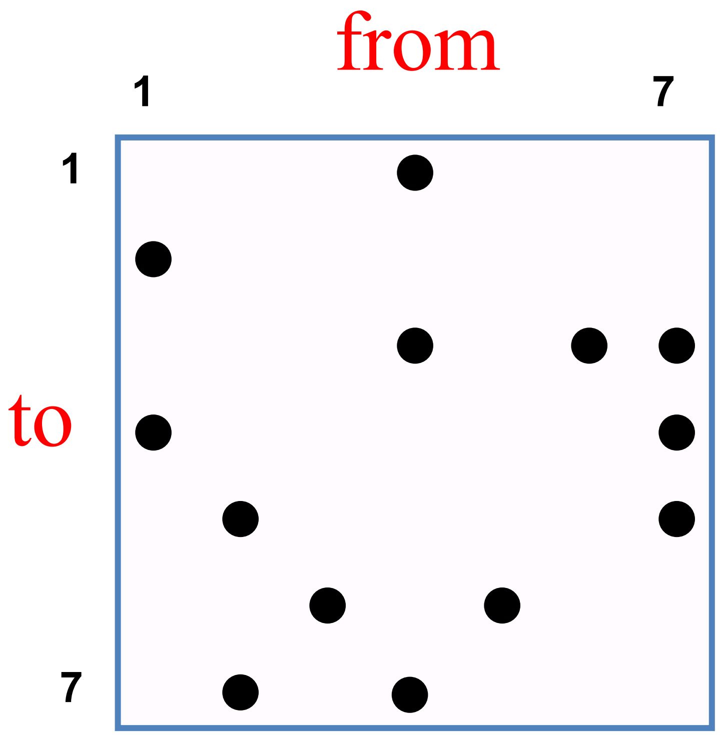
SOFTWARE • ENVIRONMENTS • TOOLS

siam

Graphs in the Language of Linear Algebra



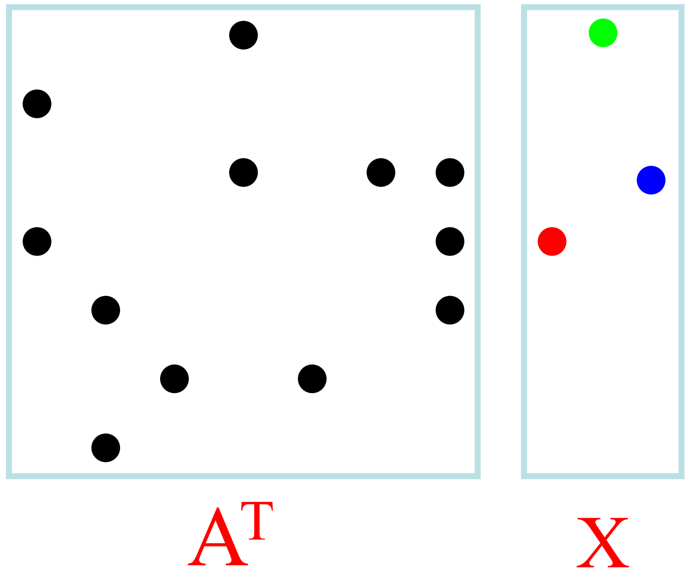
These two diagrams are equivalent representations of a graph.

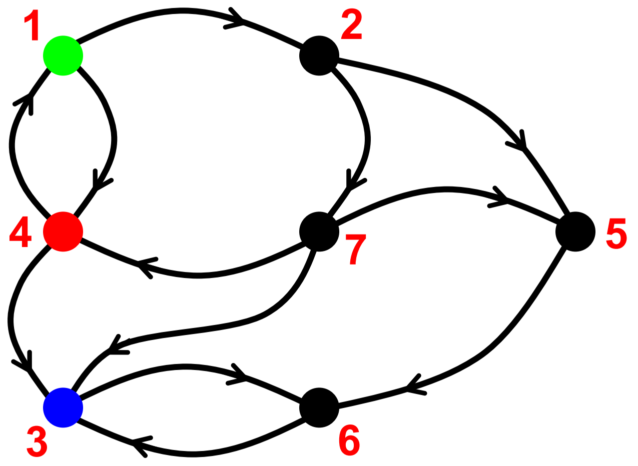


A^T

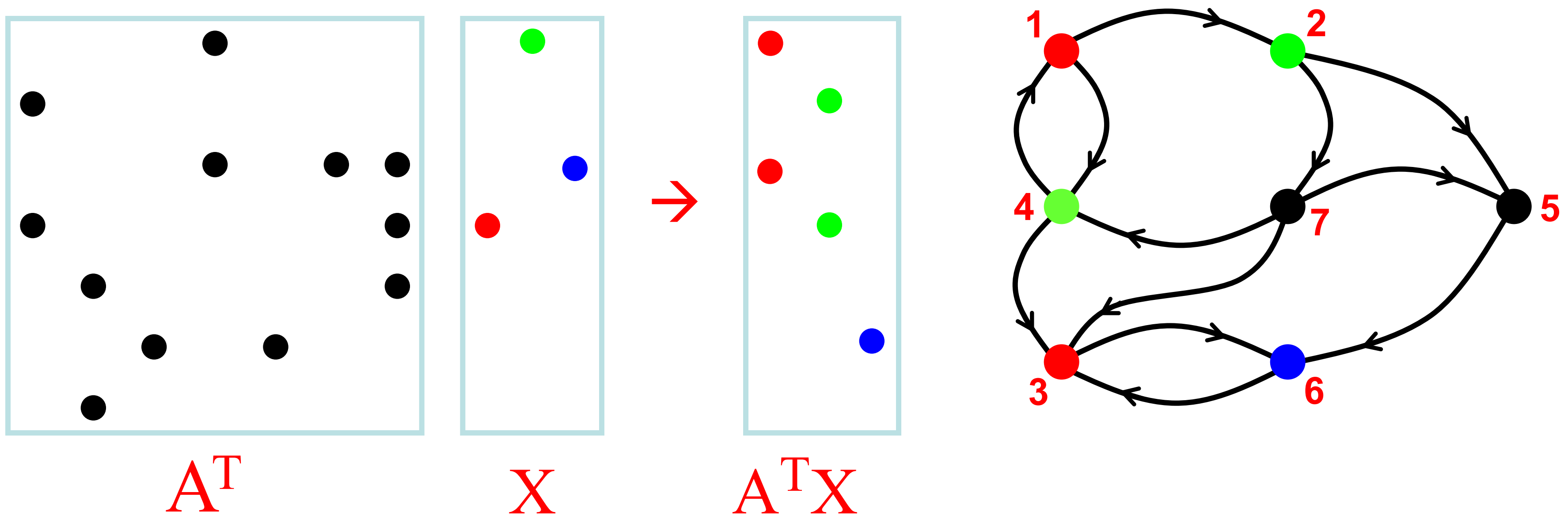
A = the adjacency matrix … Elements nonzero when vertices are adjacent

Multiple-source breadth-first search





Multiple-source breadth-first search



- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

Multiplication of sparse matrices captures Breadth first search and serves as the foundation of all algorithms based on BFS

UCSB

Moving beyond BFS with Algebraic Semirings

- A semiring generalizes the operations of traditional linear algebra by replacing (+,*) with binary operations (Op1, Op2)
  - Op1 and Op2 have identity elements sometimes called 0 and 1
  - Op1 and Op2 are associative.
  - Op1 is commutative,   Op2 distributes over op1 from both left and right
  - The Op1 identify is an Op2 annihilator.

Moving beyond BFS with Algebraic Semirings

- A semiring generalizes the operations of traditional linear algebra by replacing (+,*) with binary operations (Op1, Op2)
  - Op1 and Op2 have identity elements sometimes called 0 and 1
  - Op1 and Op2 are associative.
  - Op1 is commutative, Op2 distributes over op1 from both left and right
  - The Op1 identify is an Op2 annihilator.

(R, +, *, 0, 1) Real Field	Standard operations in linear algebra

Notation:	(R,	+,	*,	0,	1)
	Scalar type	Op1	Op2	Identity Op1	Identity Op2

Moving beyond BFS with Algebraic Semirings

- A semiring generalizes the operations of traditional linear algebra by replacing (+,*) with binary operations (Op1, Op2)
  - Op1 and Op2 have identity elements sometimes called 0 and 1
  - Op1 and Op2 are associative.
  - Op1 is commutative, Op2 distributes over op1 from both left and right
  - The Op1 identify is an Op2 annihilator.

(R, +, *, 0, 1) Real Field	Standard operations in linear algebra
({0,1}, |, &, 0, 1) Boolean Semiring	Graph traversal algorithms
(R U {∞},min, +, ∞, 0) Tropical semiring	Shortest path algorithms
(R U {∞}, min, *, ∞, 1)	Selecting a subgraph or contracting nodes to form a quotient graph.

The case for graph primitives based on sparse matrices

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

Traditional graph computations
Data driven, unpredictable communication.
Irregular and unstructured, poor locality of reference
Fine grained data accesses, dominated by latency

The case for graph primitives based on sparse matrices

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

Traditional graph computations	Graphs in the language of linear algebra
Data driven, unpredictable communication.	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse grained parallelism, bandwidth limited

GraphBLAS launched at HPEC'13 … co-authors of the GraphBLAS position paper

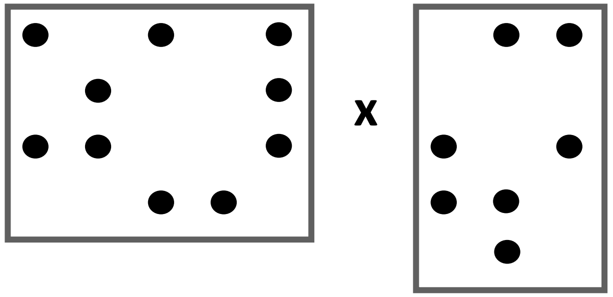
Tim Mattson	Intel Corporation	David Bader	Georgia Tech
Jon Berry	Sandia National Laboratory	Aydın Buluç	Lawrence Berkeley National Laboratory
Jack Dongarra	University of Tennessee	Christos Faloutsos	(Carnegie Melon University
John Feo	Pacific Northwest National Laboratory	John Gilbert	UC Santa Barbara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

Outline

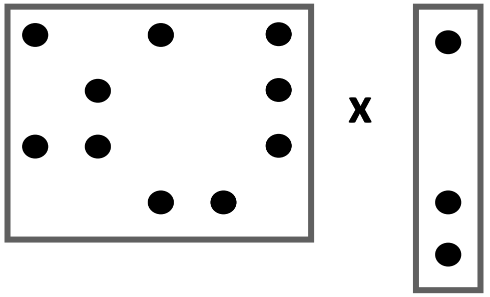
- Introduction: Graphs and Linear Algebra
- The Draft GraphBLAS primitives
- Conclusion/Summary

Linear-algebraic primitives

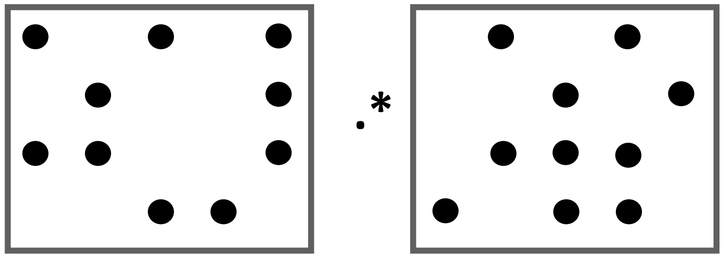
Sparse matrix-sparse matrix multiplication



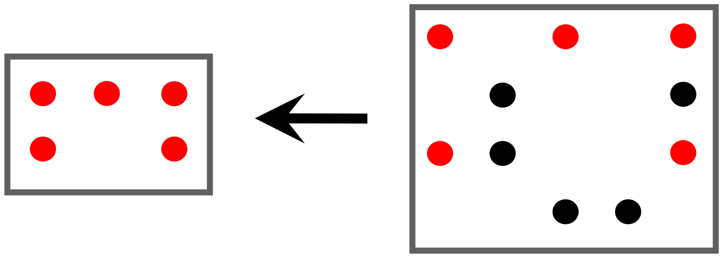
Sparse matrix-sparse vector multiplication



Element-wise operations



Sparse matrix indexing



The Combinatorial BLAS implements these, and more, on arbitrary semirings, e.g. (·, +), (and, or), (+, min)

Draft GraphBLAS functions*

Function	Parameters	Returns	Math Notation
SpGEMM	- sparse matrices **A, B** and **C** - unary functors (op)	sparse matrix	**C** += op(**A**) * op(**B**)
SpM{Sp}V **(Sp: sparse)**	- sparse matrix **A** - sparse/dense vector **x**	sparse/dense vector	**y** = **A** * **x**
SpEWiseX	- sparse matrices or vectors - binary functor and predicate	in place or sparse matrix/vector	**C** = **A** .* **B**
Reduce	- sparse matrix **A** and functors	dense vector	**y** = sum(**A**, op)
SpRef	- sparse matrix **A** - index vectors **p** and **q**	sparse matrix	**B** = **A**(**p**,**q**)
SpAsgn	- sparse matrices **A** and **B** - index vectors **p** and **q**	none	**A**(**p**,**q**) = **B**
Scale	- sparse matrix **A** - dense matrix **B** or vector **X**	none	∀ **A**(i,j)!=0: **A**(i,j) *=**B**(i,j) + related forms for **X**
Apply	- any matrix or vector **X** - unary functor (op)	none	op(**X**)

*based on the Combinatorail BLAS from Buluç and Gilbert

Matrix times Matrix over semiring

Inputs

matrix **A**: $\mathbb{S}^{MxN}$ (sparse or dense)

matrix **B**: $\mathbb{S}^{NxL}$ (sparse or dense)

Optional Inputs

matrix **C**: $\mathbb{S}^{MxL}$ (sparse or dense)

scalar "add" function ⊕

scalar "multiply" function ⊗

transpose flags for **A**, **B**, **C**

Outputs

matrix **C**: $\mathbb{S}^{MxL}$ (sparse or dense)

Implements **C** ⊕= **A** ⊕.⊗ **B**

```
for j = 1 : N
    C(i,k) = C(i,k) ⊕ (A(i,j) ⊗ B(j,k))
```

If input **C** is omitted, implements

C = **A** ⊕.⊗ **B**

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

Notes

$\mathbb{S}$ is the set of scalars, user-specified

$\mathbb{S}$ defaults to IEEE double float

⊕ defaults to floating-point +

⊗ 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 **A**: $\mathbb{S}^{M \mathrm{x} N}$ (sparse)

matrix **B**: $\mathbb{S}^{|p| \mathrm{x} |q|}$ (sparse)

vector $p \subseteq \{1, \ldots, M\}$

vector $q \subseteq \{1, \ldots, N\}$

Optional Inputs

none

Outputs

matrix **A**: $\mathbb{S}^{M \mathrm{x} N}$ (sparse)

matrix **B**: $\mathbb{S}^{|p| \mathrm{x} |q|}$ (sparse)

SpRef Implements **B** = **A**(p,q)

```
for i = 1 : |p|
  for j = 1 : |q|
    B(i,j) = A(p(i),q(j))
```

SpAsgn Implements **A**(p,q) = **B**

```
for i = 1 : |p|
  for j = 1 : |q|
    A(p(i),q(j)) = 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**: $\mathbb{S}^{MxN}$ (sparse or dense)

matrix **B**: $\mathbb{S}^{MxN}$ (sparse or dense)

Optional Inputs

matrix **C**: $\mathbb{S}^{MxN}$ (sparse or dense)

scalar "add" function $\oplus$

scalar "multiply" function $\otimes$

Outputs

matrix **C**: $\mathbb{S}^{MxN}$ (sparse or dense)

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

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

If input **C** is omitted, implements

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

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 *

Specific cases and function names:

SpEWiseX: matrix elementwise

M=1 or N=1: vector elementwise

Scale: when **A** or **B** is a scalar

Apply/Update

<u>Inputs</u>

matrix **A**: $\mathbb{S}^{MxN}$ (sparse or dense)

<u>Optional Inputs</u>

matrix **C**: $\mathbb{S}^{MxN}$ (sparse or dense)

scalar "add" function $\oplus$

unary function f()

<u>Outputs</u>

matrix **C**: $\mathbb{S}^{MxN}$ (sparse or dense)

<u>Implements</u> **C** $\oplus$= f(**A**)

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

If input **C** is omitted, implements

C = f(**A**)

<u>Notes</u>

$\mathbb{S}$ is the set of scalars, user-specified

$\mathbb{S}$ defaults to IEEE double float

$\oplus$ defaults to floating-point +

<u>Specific cases and function names:</u>

Apply: matrix apply

M=1 or N=1: vector apply

Matrix/Vector Reductions

Inputs

matrix **A**: $\mathbb{S}^{MxN}$ (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**: $\mathbb{S}^{MxN}$ (sparse or dense)

Implements $\mathbf{c}(i) \oplus= \oplus_j \mathbf{A}(i,j)$

```
for i = 1 : M
  for j = 1 : N
    $\mathbf{c}(i) = \mathbf{c}(i) \oplus \mathbf{A}(i,j)$
```

If input **C** is omitted, implements

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

Notes

$\mathbb{S}$ is the set of scalars, user-specified

$\mathbb{S}$ defaults to IEEE double float

$\oplus$ defaults to floating-point +

d defaults to 2

Specific cases and function names:

Reduce (d = 1): reduce matrix to row vector

Reduce (d = 2): reduce matrix to col vector

Outline

- Introduction: Graphs and Linear Algebra
- The Draft GraphBLAS primitives
- Conclusion/Summary

Conclusion/Summary

- The time is right to define a Standard to support “Graphs in the Language of Linear Algebra”.

- Agreeing on a standard could have a transformative impact on Graph Algorithms research … much as the original BLAS did on computational Linear Algebra.

- Starting from the CombBLAS (Buluç and Gilbert), we have produced an initial Draft set of Primitives.

- Join with us to turn this into a final spec
  - Follow our work at: http://istc-bigdata.org/GraphBlas/
  - Send email to timothy.g.mattson@intel.com if you want to be added to the GraphBLAS Google Group and join the effort

Bonus Slides

Sparse array attribute survey

Function	Graph BLAS	Comb BLAS	Sparse BLAS	STINGER	D4M	SciDB	Tensor Toolbox	Julia	GraphLab
Version		1.3.0	2006	r633	2.5	13.9	2.5	0.2.0	2.2
Language	any	C++	F,C,C++	C	Matlab	C++	Matlab, C++	Julia	C++
Dimension	2	1, 2	2	1, 2, 3	2	1 to 100	2, 3	1,2	2
Index Base	0 or 1	0	0 or 1	0	1	±N	1	1	0
Index Type	uint64	uint64	int	int64	double, string	int64	double	any int	uint64
Value Type	?	user	single, double, complex	int64	logical, double, complex, string	user	logical, double, complex	user	user
Null	0	user	0	0	≤0	null	0	0	int64(-1)
Sparse Format	?	tuple	undef	linked list	dense, csc, tuple	RLE	dense, csc	csc	csr/csc
Parallel	?	2D block	none	block	arbitrary	N-D block, cyclic w/overlap	none	N-D block, cyclic w/overlap	Edge based w/ vertex split
+ operations *** operations**	user? user?	user user	+ *	user user	+,*,max,min, ∩,∪	user user	+ *	user user	user user

Sparse Matrix Products (General Form)

$$\mathbf{C} = \mathrm{op}(\mathbf{A}) * \mathrm{op}(\mathbf{B})$$

$\mathrm{op}(\mathbf{A})$	: $\mathbb{S}^{NxM}$	(sparse matrix)
$\mathrm{op}(\mathbf{B})$	: $\mathbb{S}^{MxK}$	(sparse or dense)
$\mathbf{C}$	: $\mathbb{S}^{NxK}$	(sparse or dense)
$\mathbb{S}$	: base type (int, float, double,…)	
op()	: transpose or no-op	
*	: f().g() matrix multiply operation	
f()	: binary function (e.g., addition)	
g()	: binary function (e.g., multiply)	

Example, let op() be a no-op, then **C**(i,k) can be computed as follows:

```
for j=1:M
        C(i,k) = f( C(i,k) ,  g(A(i,j),B(j,k)) )
```

Generalizes:
- SpGEMM
- SpMM

(special case K=1)
- SpMV
- SpMSpV

Sparse Matrix Products (Triple Store & BFS View)

Extends associative arrays to 2D and mixed data types

```
A('alice ','bob ') = 'cited '
```

or

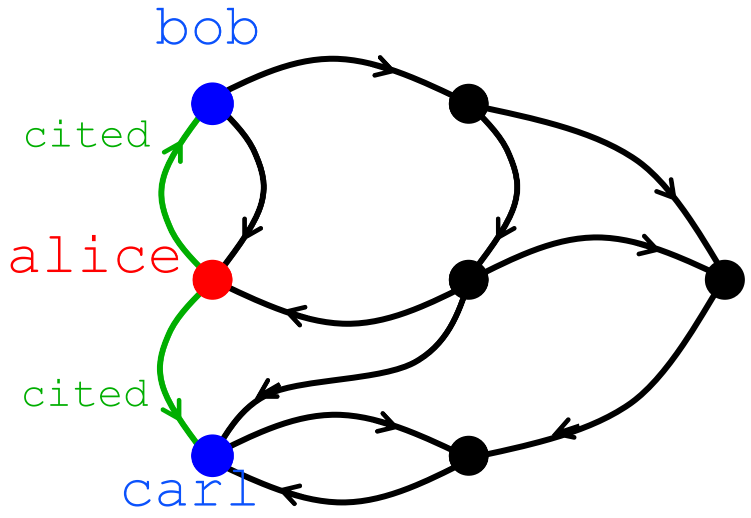
```
A('alice ','bob ') = 47.0
```

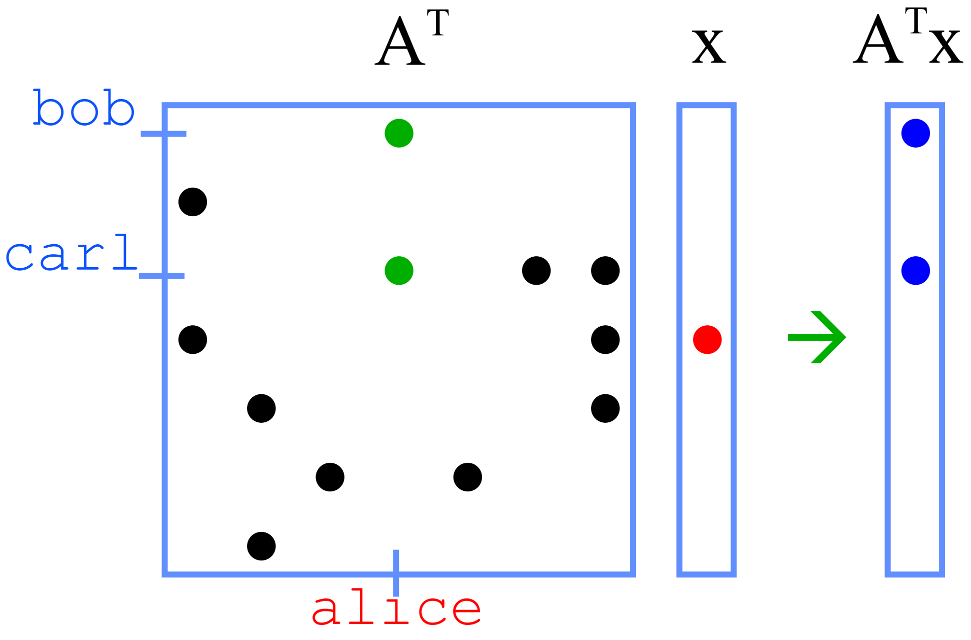
Key innovation: 2D is 1-to-1 with triple store

```
('alice ','bob ','cited ')
```

or

```
('alice ','bob ',47.0)
```





Sparse Matrix Products (Functional Interpretation)

$$\mathbf{C} = \mathrm{op}(\mathbf{A}) * \mathrm{op}(\mathbf{B})$$

E: The edges of the graph
V: The vertices of the graph
B= X<V> is the set of active vertices
A=G<E,V> represents the graph G
C=X'<V> is the new set of active vertices

Examples for X<V>:

- Breadth-first search frontier
- The candidate set in Luby's maximal independent set alg.
- The active vertices (w/ values not converged) in PageRank

$$\mathit{prod}(G\langle E,V\rangle, X\langle V\rangle, g(), f()) \rightarrow X'\langle V\rangle$$
$$g: E,V \rightarrow V$$
$$f: V,V \rightarrow V$$

$$X\langle V\rangle, X'\langle V\rangle \subseteq V$$

Sparse Matrix Indexing & Assignment

$$\begin{array}{ll} \text{spref:} & B = A(p,q) \\ \text{spasgn:} & A(p,q) = B \end{array}$$

$A \in S^{n \times m}$

$B \in S^{length(p) \times length(q)}$

A & B are sparse

p & q are integer vectors

$p \subseteq \{1, 2, \ldots, n\}$

$q \subseteq \{1, 2, \ldots, m\}$

A functional interpretation of SpAsgn on Graphs:

$$subscript(G\langle E,V\rangle, G'\langle E',V\rangle, I) \rightarrow G''\langle E \cup E', V\rangle$$

$$G''_{i,j} = \left| \begin{array}{ll} G'_{i,j} & \text{if } (i,j) \in I \\ G_{i,j} & \text{otherwise} \end{array} \right.$$

I : vertex pairs

$I \square \{1, \ldots, n\}^2$

Element-Wise Operations

$$\text{Matrix: } C = A \;.^{\square}\; B$$
$$\text{Vector: } z = x \;.^{\square}\; \mathrm{y}$$

- All operands are sparse
- Zeros are not stored

$$C'_{i,j} = \begin{cases} id(A) \square B_{i,j} & \text{if } A_{i,j} = 0, B_{i,j} \square 0 \\ A_{i,j} \square id(B) & \text{if } A_{i,j} \square 0, B_{i,j} = 0 \\ A_{i,j} \square B_{i,j} & \text{if } A_{i,j} \square 0, B_{i,j} \square 0 \end{cases}$$

$$C_{i,j} = \begin{cases} C'_{i,j} & \text{if } C'_{i,j} \square id(C) \\ 0 & \text{otherwise} \end{cases}$$

- Identities id(A),id(B),id(C) defined **per operation**, not operand
- sparsity is defined on operand

Apply/Update

$$B = apply(A, f)$$

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

$$sparsity(A) = sparsity(B)$$

f is a unary operation

$$f(a_{i,j}) = \begin{cases} f(a_{i,j}) & \text{if } a_{i,j} \neq 0 \\ no-op & \text{otherwise} \end{cases}$$

A functional interpretation of Apply:

$$map(G\langle E,V\rangle, f) \rightarrow G'\langle E',V\rangle$$

$$f(i,j,G_{i,j}) \rightarrow G'_{i,j}$$

Matrix/Vector Reductions

$$\mathrm{v} = reduce(A, \dim)$$

Graph in/out degrees

$$A \in S^{n \times m}$$

$$\dim = 0(row), 1(column)$$

v is dense

$$v \in \begin{cases} S^n & \text{if dim=0} \\ S^m & \text{if dim=1} \end{cases}$$

References

- Graph Algorithms in the Language of Linear Algebra, Edited by J. Kepner and J. Gilbert, SIAM, 2011

- The Combinatorial BLAS: Design, Implementation and Applications, A. Buluc and J. Gilbert
  http://www.cs.ucsb.edu/research/tech_reports/reports/2010-18.pdf

- Standards for Graph Algorithm Primitives, Proceedings of the IEEE High Performance Extreme Computing Conference 2013, T. G. Mattson, D. Bader, J. Berry, A. Buluc, J. Dongarra, C. Jaloutsos, J. Feo, J. Gilbert, J. Gonzalez, B. Hendrickson, J. Kepner, C. Leiserson, A. Lumsdaine, D. Padua, S. Poole, S. Reinhardt, M. Stonebraker, S. Wallach,and A. Yoo.
  http://www.netlib.org/utk/people/JackDongarra/PAPERS/GraphPrimitives-HPEC.pdf