Building Blocks for Graph Algorithms in the Language of Linear Algebra

John R. Gilbert
University of California, Santa Barbara

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

CIMI Workshop on Innovative Clustering Methods
July 7, 2015

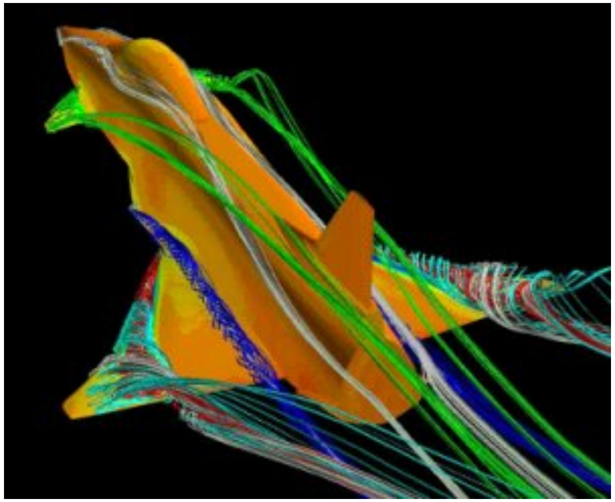
Support at UCSB: Intel, Microsoft, DOE Office of Science, NSF

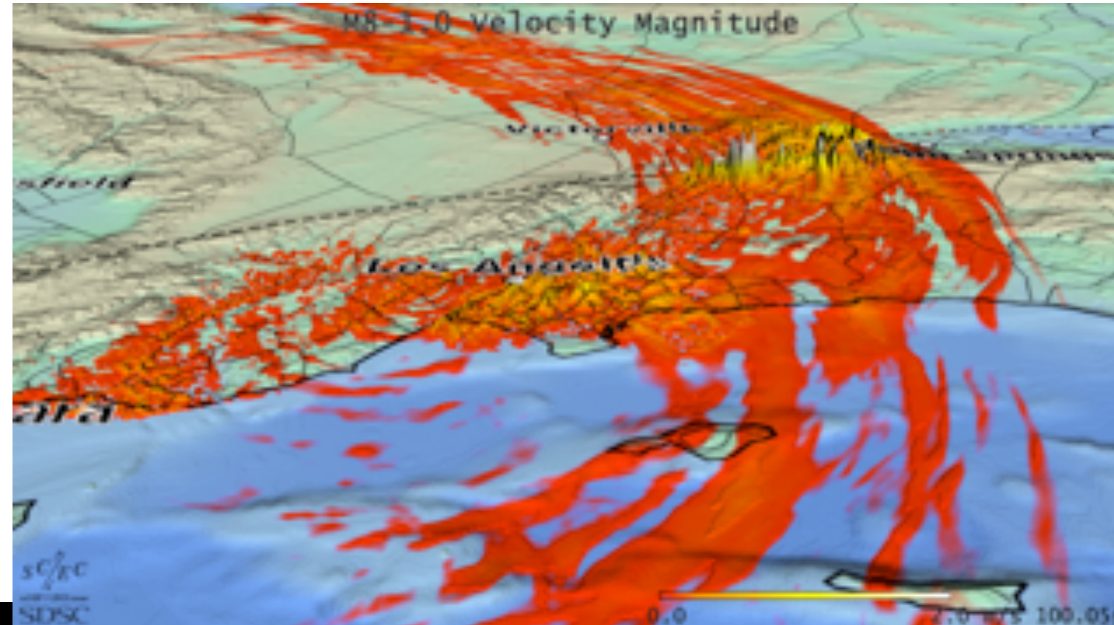
Outline

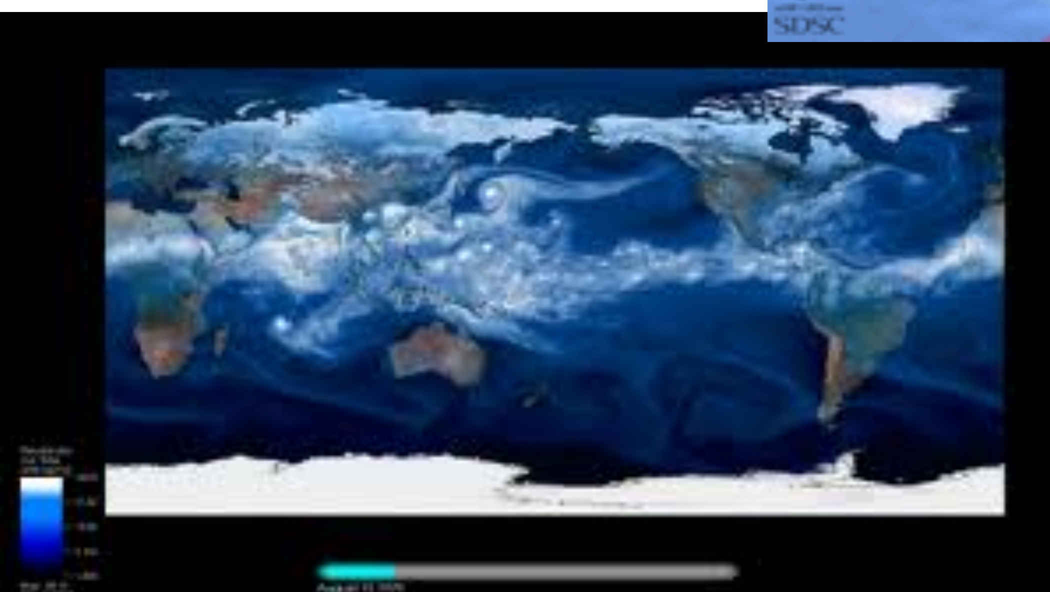
- Motivation: Graph applications
- Mathematics: Sparse matrices for graph algorithms
- Software: CombBLAS, KDT, QuadMat
- Standards: The Graph BLAS effort

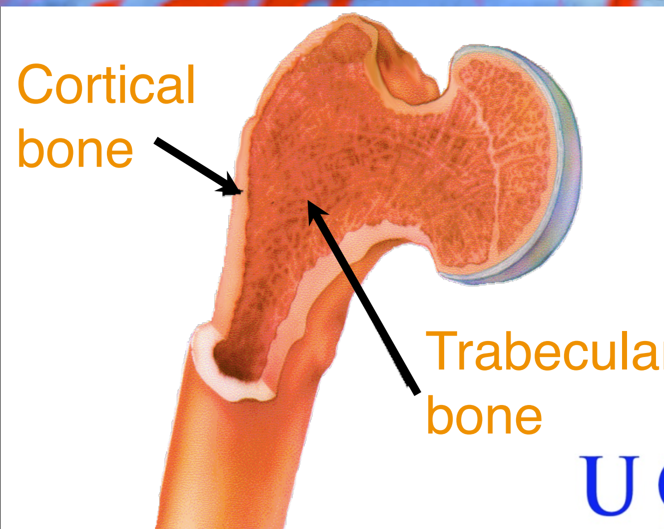
UCSB

Computational models of the physical world





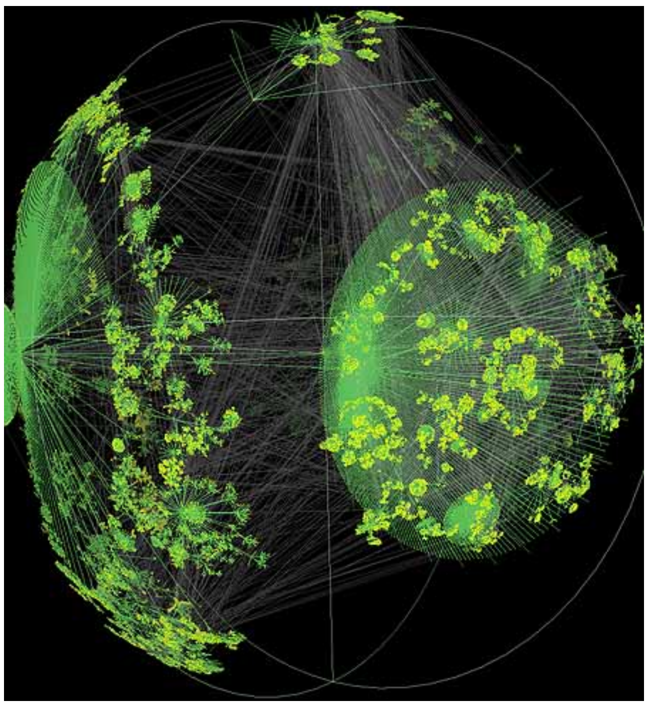




UCSB

Large graphs are everywhere…

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

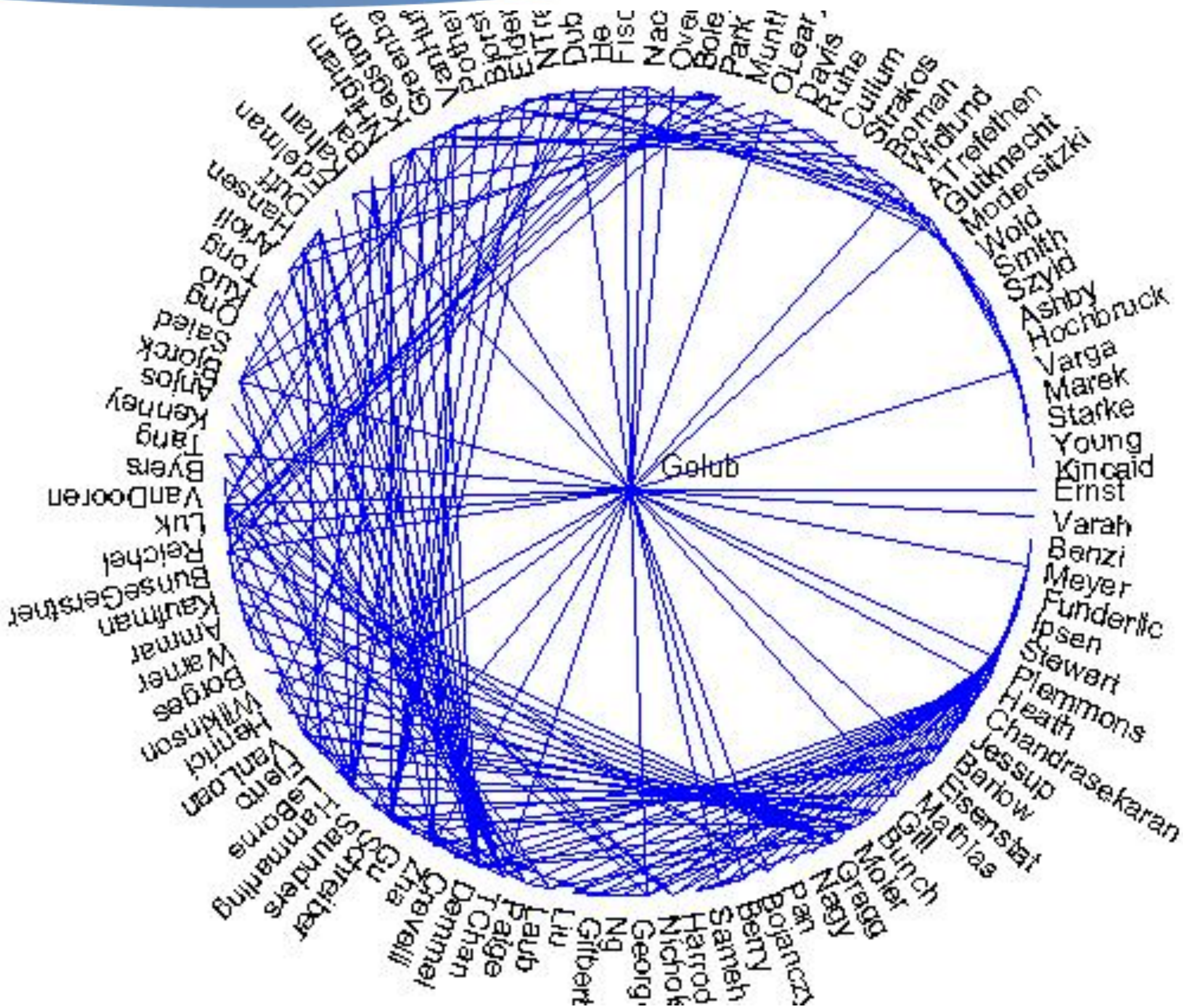


WWW snapshot, courtesy Y. Hyun

Yeast protein interaction network, courtesy H. Jeong

UCSB

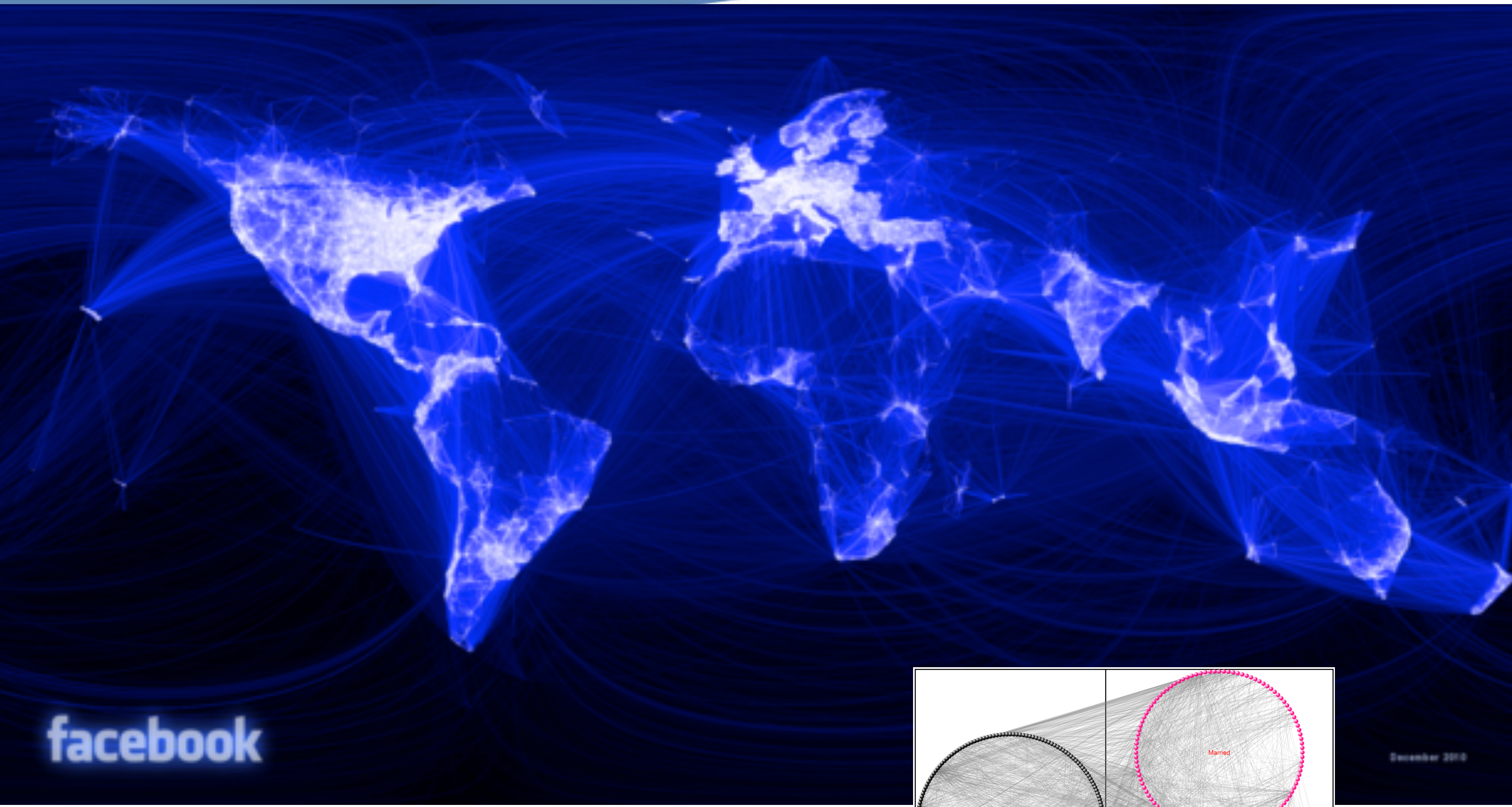
Social network analysis (1993)

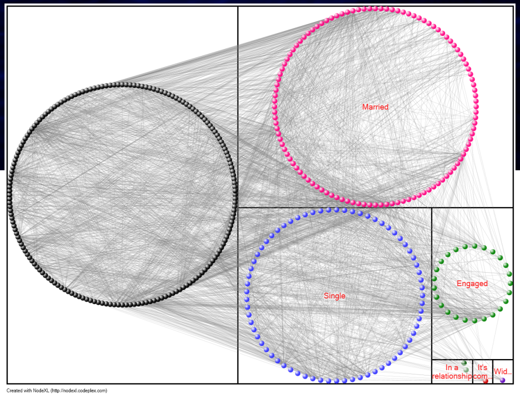


Co-author graph from 1993 Householder symposium

UCSB

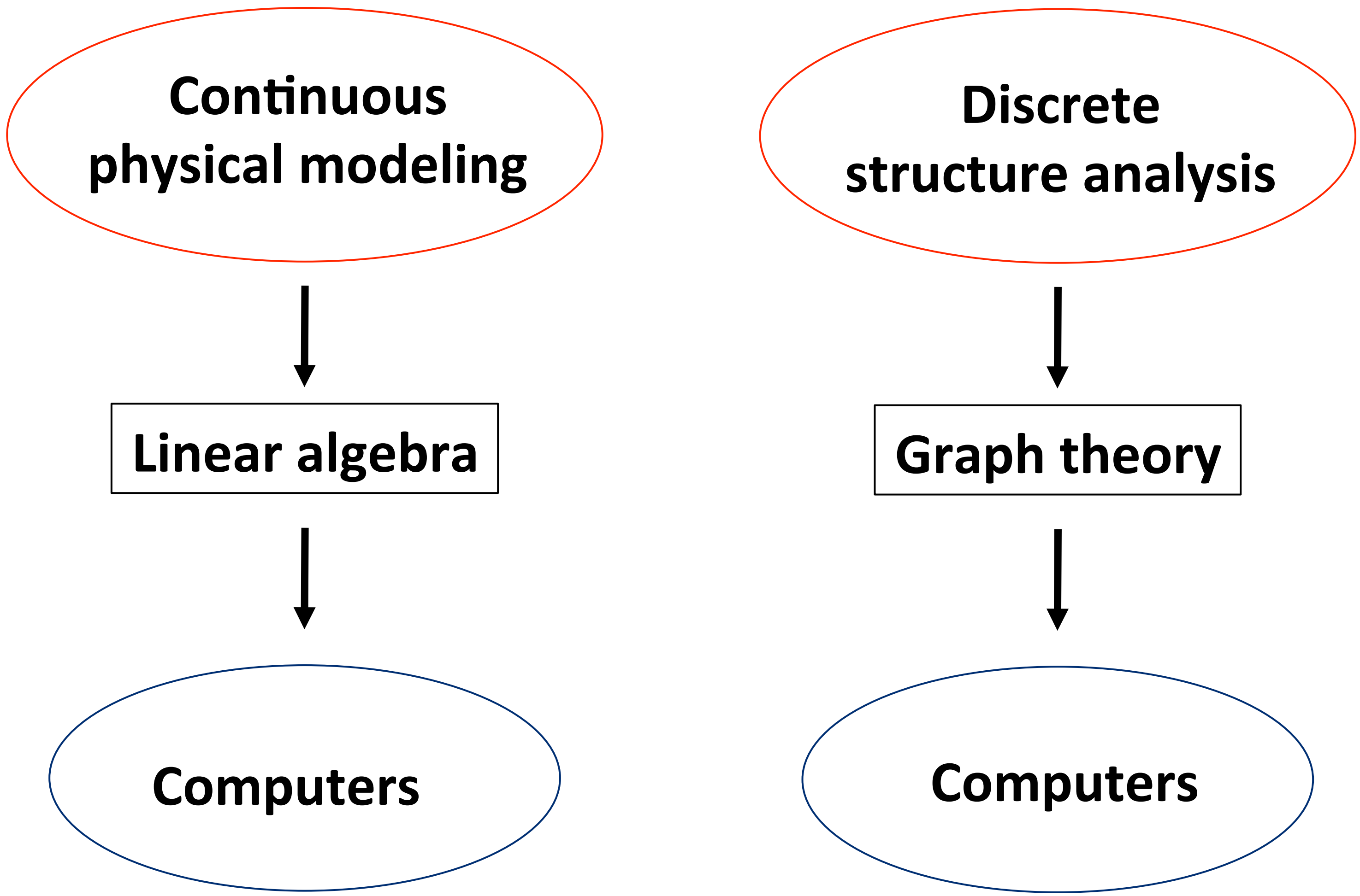
Social network analysis (2015)





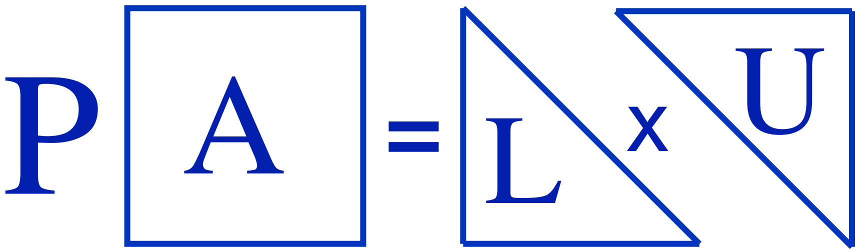
Facebook graph:
> 1,000,000,000 vertices

The middleware challenge for graph analysis

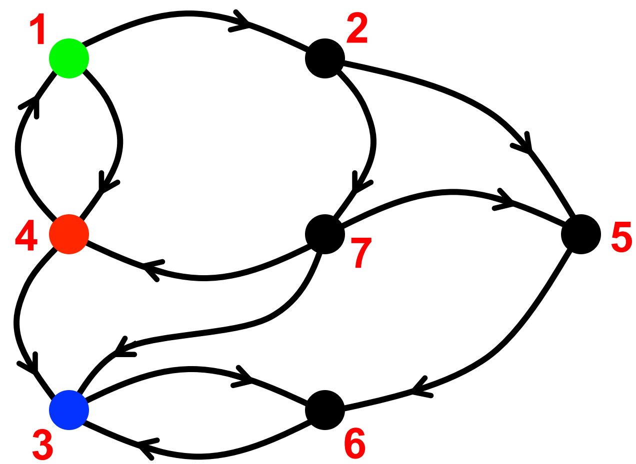


Dense linear algebra vs. graphs, November 2014

34 Petaflops (Top500)



24 Terateps (Graph500)

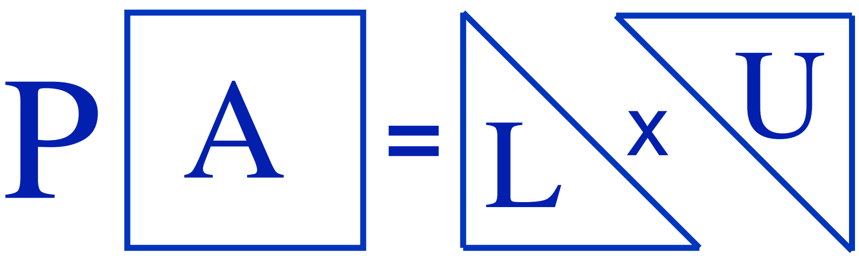


34 Peta / 24 Tera is about 1,400

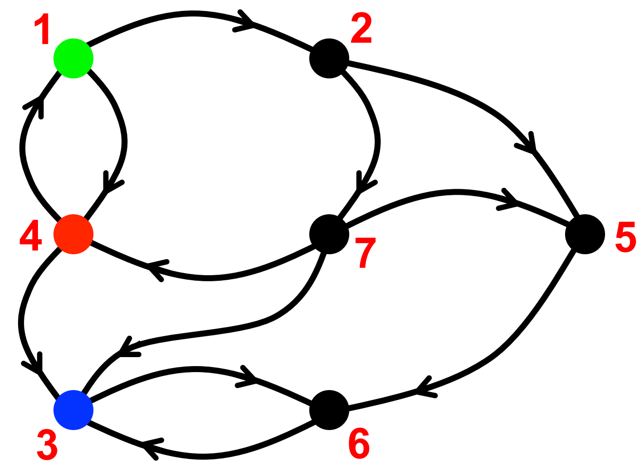
UCSB

Dense linear algebra vs. graphs, November 2014

34 Petaflops (Top500)



24 Terateps (Graph500)



Nov 2014: **34 Peta / 24 Tera** **~ 1,400**

Nov 2010: **2.5 Peta / 6.6 Giga** **~ 380,000**

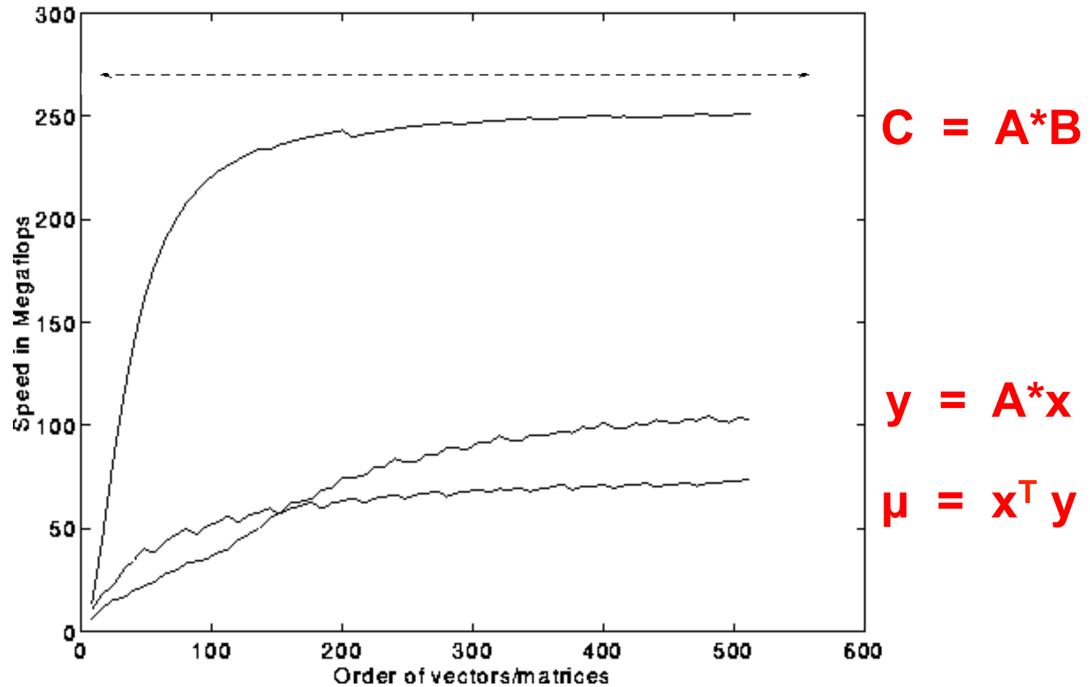
UCSB

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



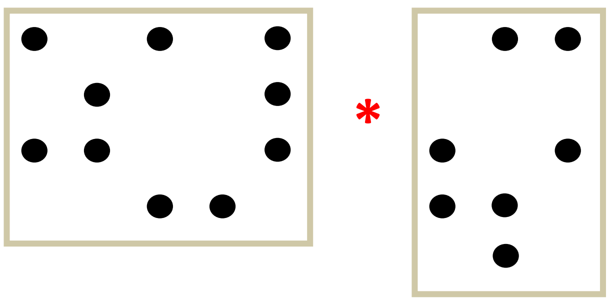
The case for sparse matrices

Coarse-grained parallelism can be exploited by abstractions at the right level.

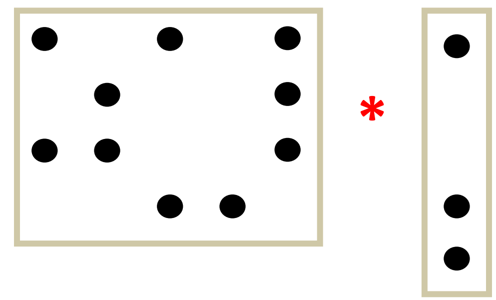
Vertex/edge graph computations	Graphs in the language of linear algebra
Unpredictable, data-driven communication patterns	Fixed communication patterns
Irregular data accesses, with poor locality	Matrix block operations exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse grained parallelism, limited by bandwidth not latency

Sparse array primitives for graphs

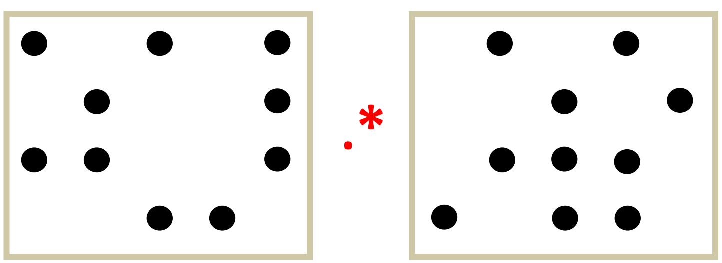
Sparse matrix-sparse matrix multiplication



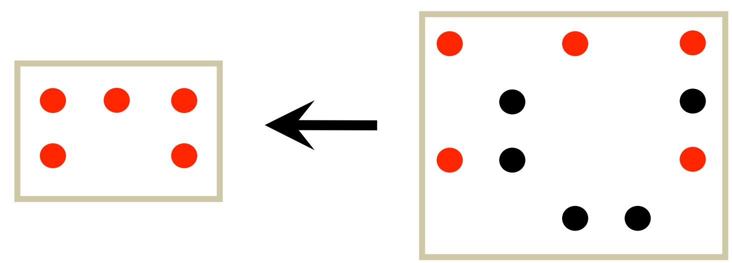
Sparse matrix-sparse vector multiplication



Element-wise operations

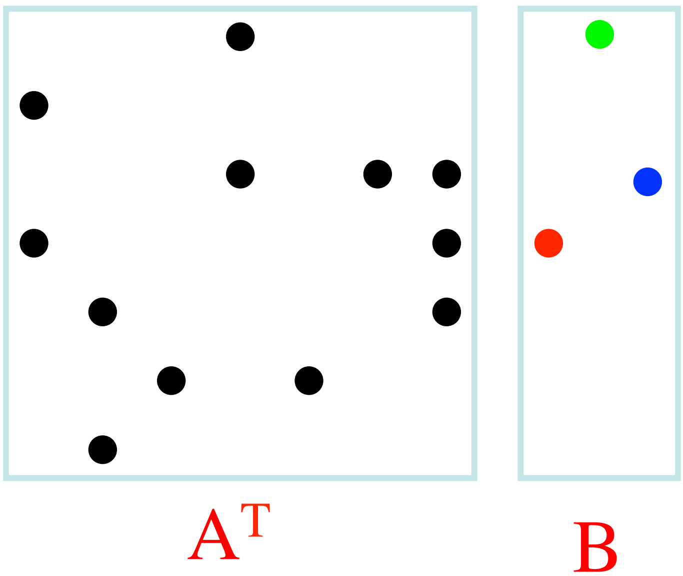


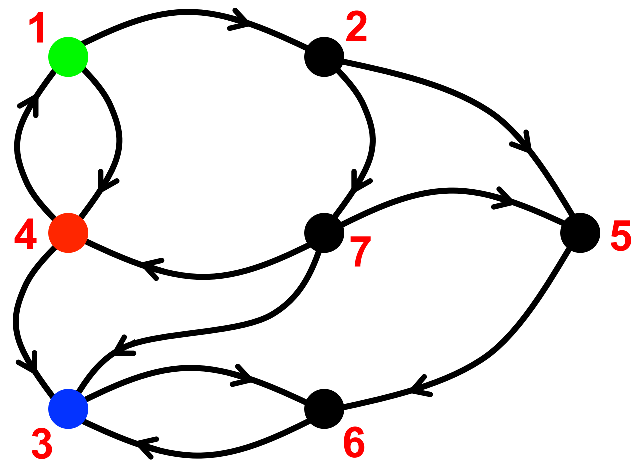
Sparse matrix indexing



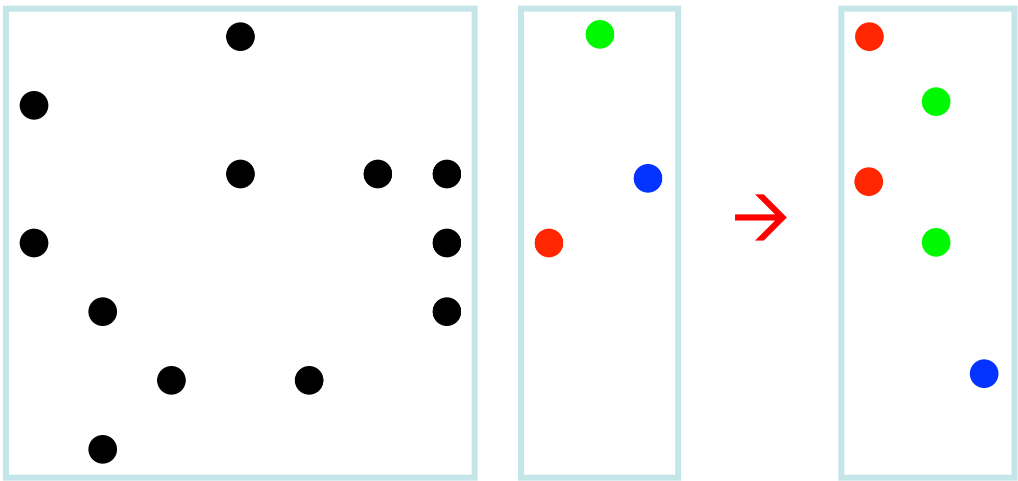
Matrices over various semirings: (+, ×), (and, or), (min, +), …

Multiple-source breadth-first search

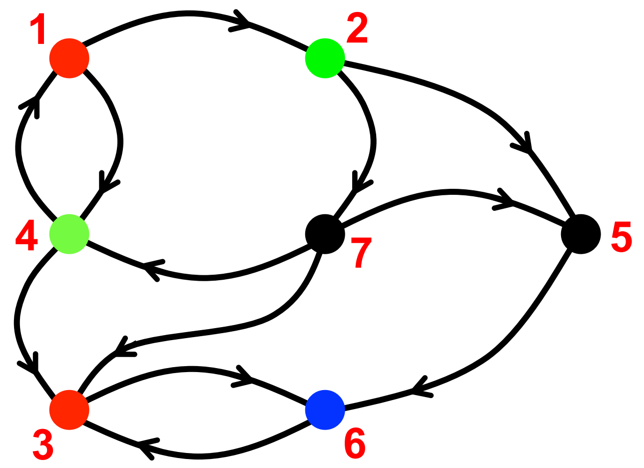




Multiple-source breadth-first search



A^T B $A^T B$



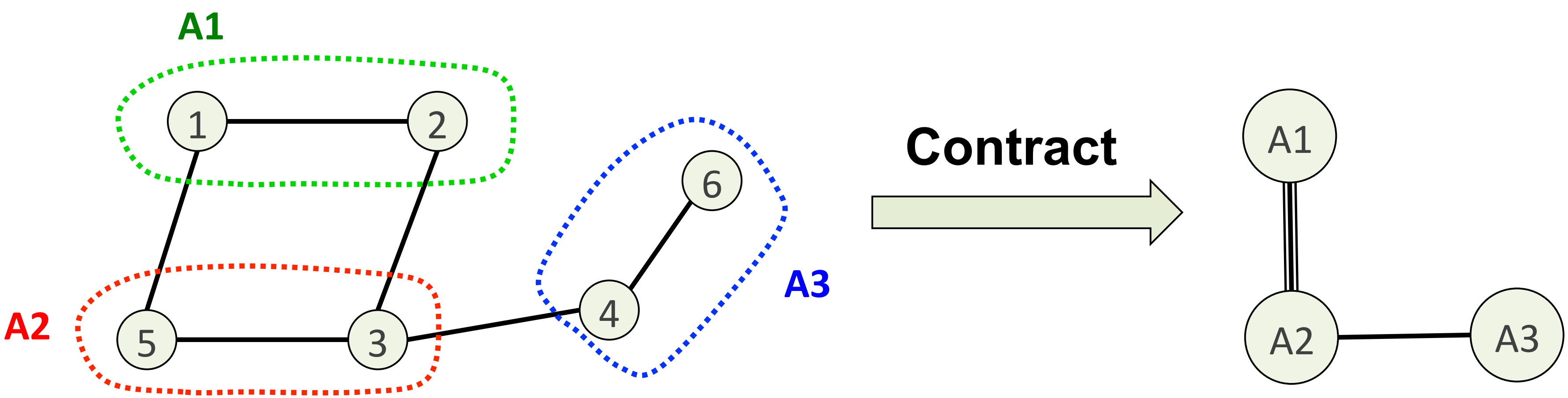
- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

Examples of semirings in graph algorithms

("values": edge/vertex attributes, "add": vertex data aggregation, "multiply": edge data processing)	General schema for user-specified computation at vertices and edges
Real field: (R, +, *)	Numerical linear algebra
Boolean algebra: ({0 1}, |, &)	Graph traversal
Tropical semiring: (R U {∞}, min, +)	Shortest paths
(S, select, select)	Select subgraph, or contract nodes to form quotient graph

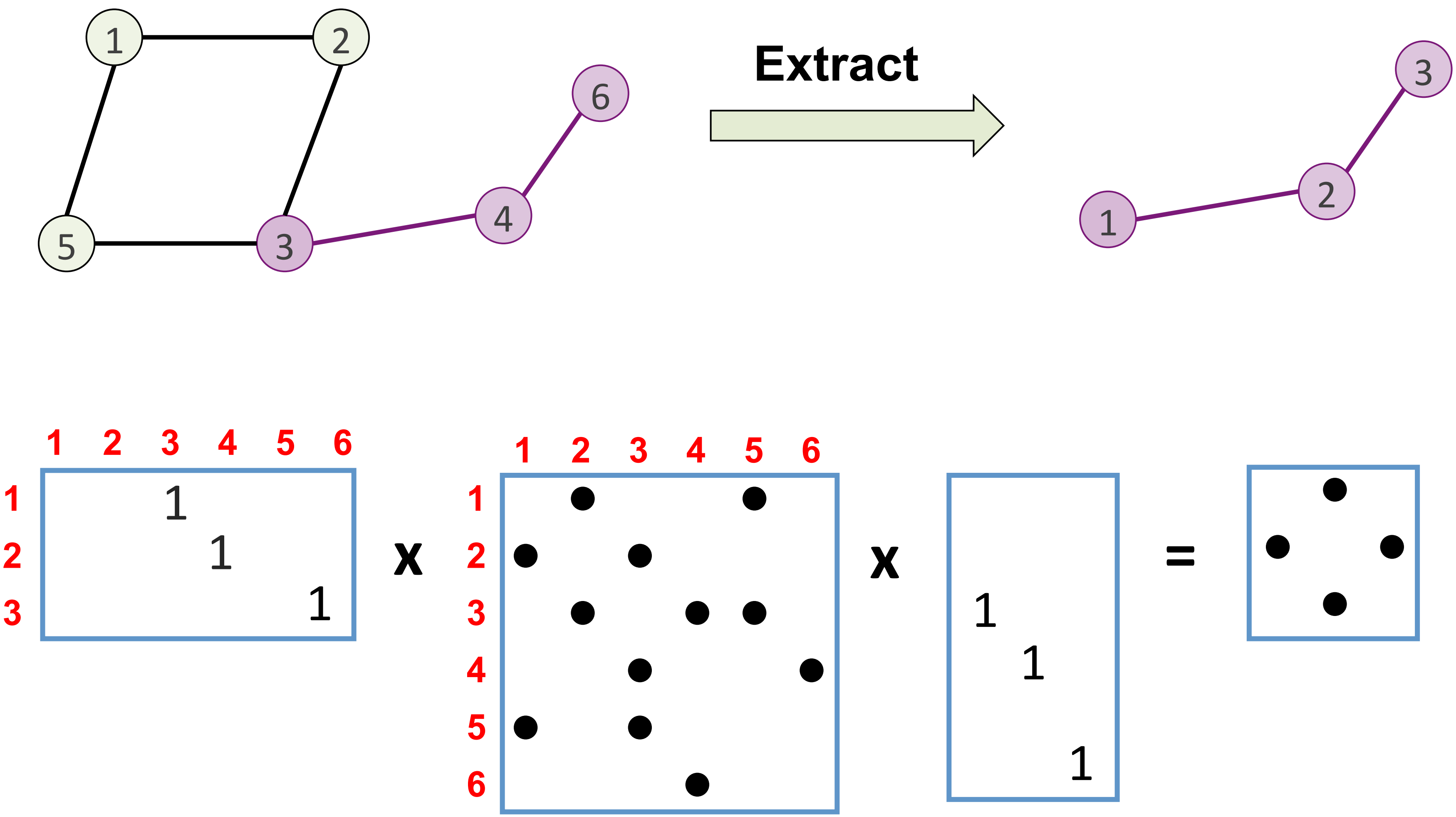
UCSB

Graph contraction via sparse triple product

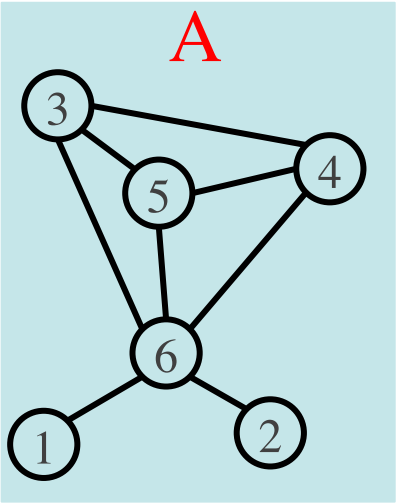


$$
\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 1 & 1 & & & & \\
2 & & & 1 & & 1 & \\
3 & & & & 1 & & 1
\end{array}
\;\times\;
\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & & \bullet & & & \bullet & \\
2 & \bullet & & \bullet & & & \\
3 & & \bullet & & \bullet & \bullet & \\
4 & & & \bullet & & & \bullet \\
5 & \bullet & & \bullet & & & \\
6 & & & & \bullet & &
\end{array}
\;\times\;
\begin{bmatrix}
1 & & \\
1 & & \\
 & 1 & \\
 & & 1 \\
 & 1 & \\
 & & 1
\end{bmatrix}
\;=\;
\begin{bmatrix}
 & \bullet & \\
\bullet & & \bullet \\
 & \bullet &
\end{bmatrix}
$$

Subgraph extraction via sparse triple product



Counting triangles (clustering coefficient)

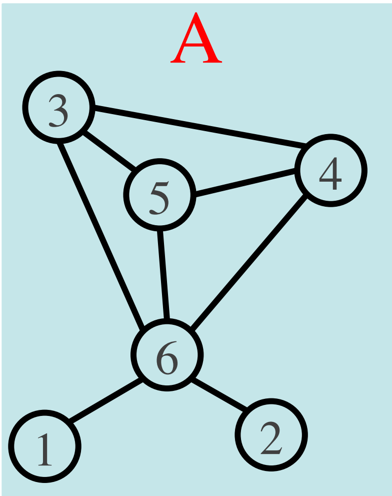


Clustering coefficient:

- Pr (wedge i-j-k makes a triangle with edge i-k)
- 3 * # triangles / # wedges
- 3 * **4** / **19** = **0.63** in example
- may want to compute for each vertex j

UCSB

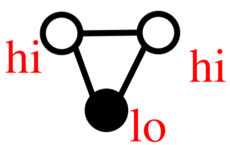
Counting triangles (clustering coefficient)



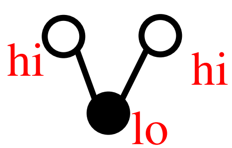
Clustering coefficient:

- Pr (wedge i-j-k makes a triangle with edge i-k)
- 3 * # triangles / # wedges
- 3 * **4** / **19** = **0.63** in example
- may want to compute for each vertex j

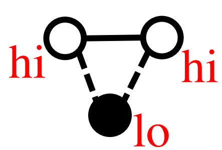
"Cohen's" algorithm to count triangles:



- Count triangles by lowest-degree vertex.

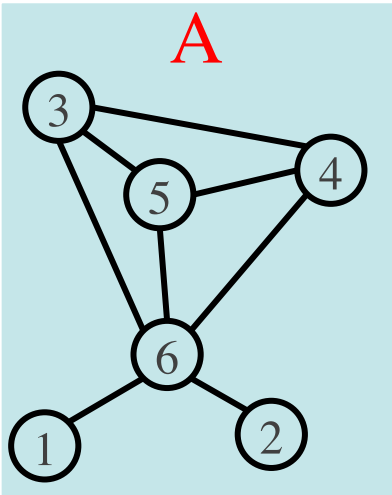


- Enumerate "low-hinged" wedges.



- Keep wedges that close.

Counting triangles (clustering coefficient)

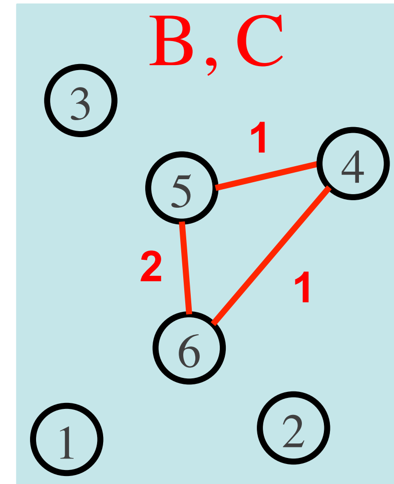


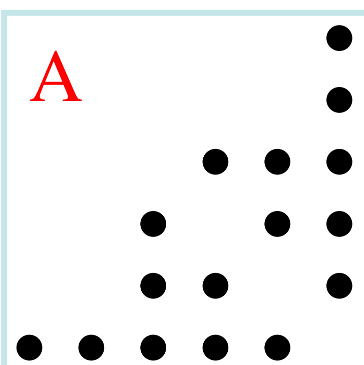
$A = L + U$ (hi->lo + lo->hi)

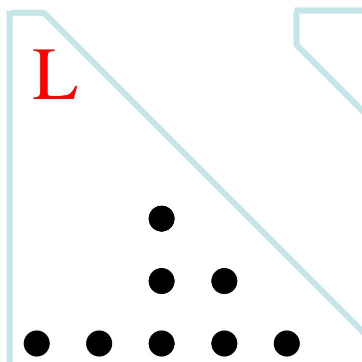
$L \times U = B$ (wedge, low hinge)

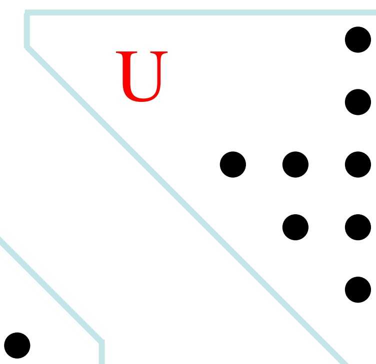
$A \wedge B = C$ (closed wedge)

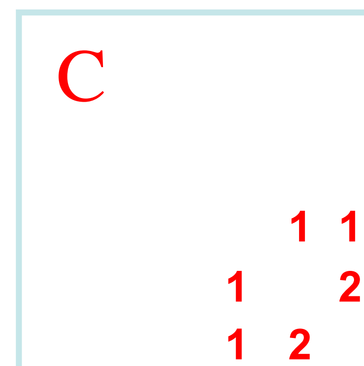
sum(C)/2 = **4 triangles**





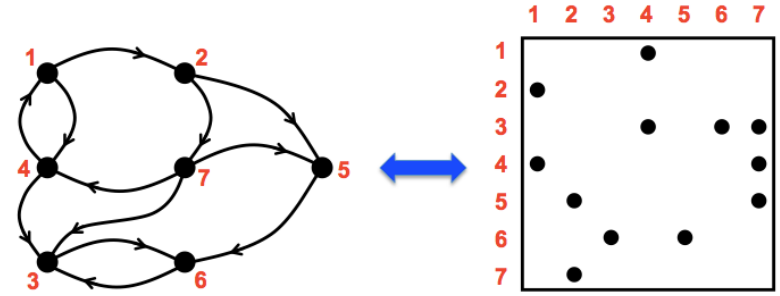






Combinatorial BLAS

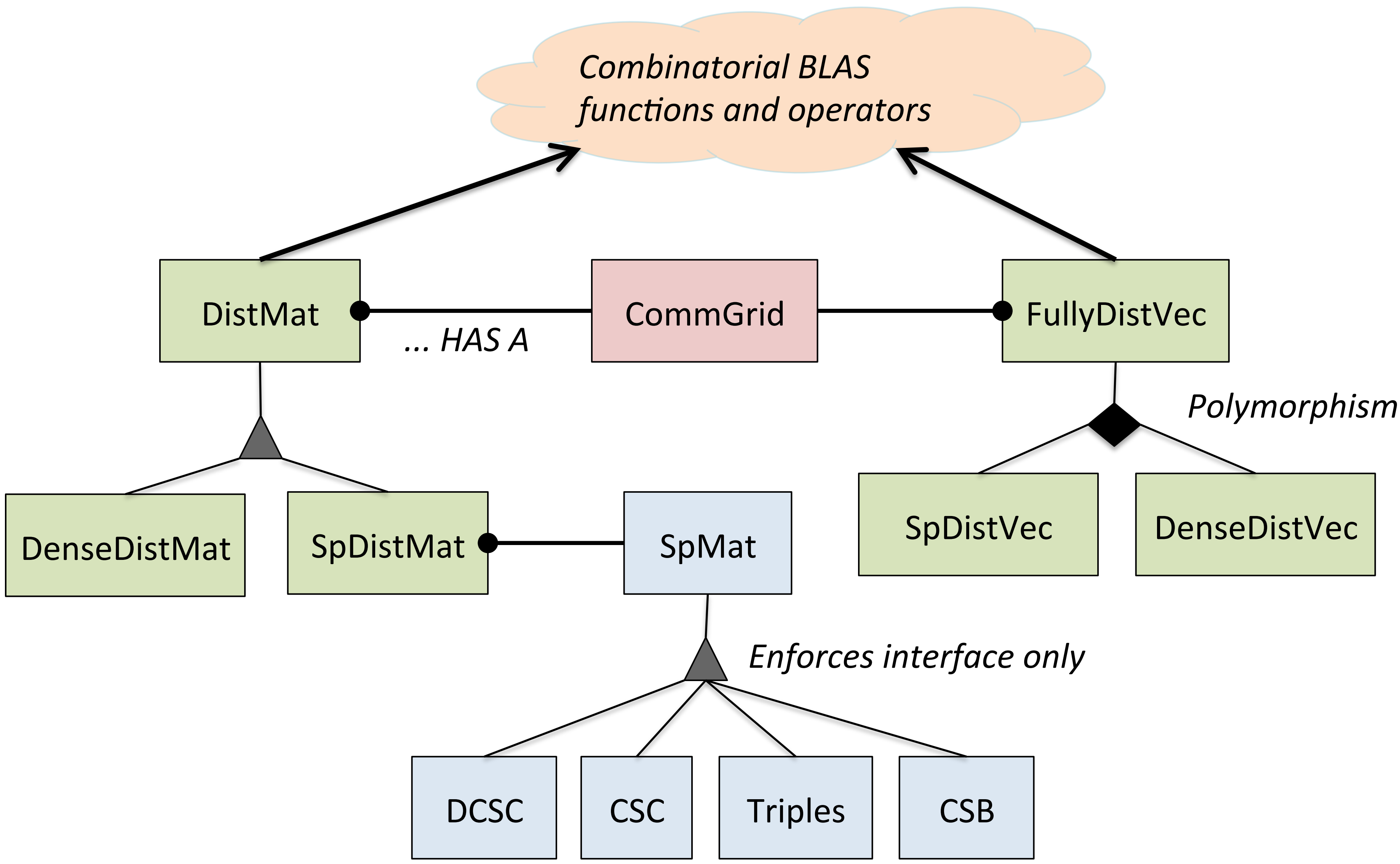
http://gauss.cs.ucsb.edu/~aydin/CombBLAS



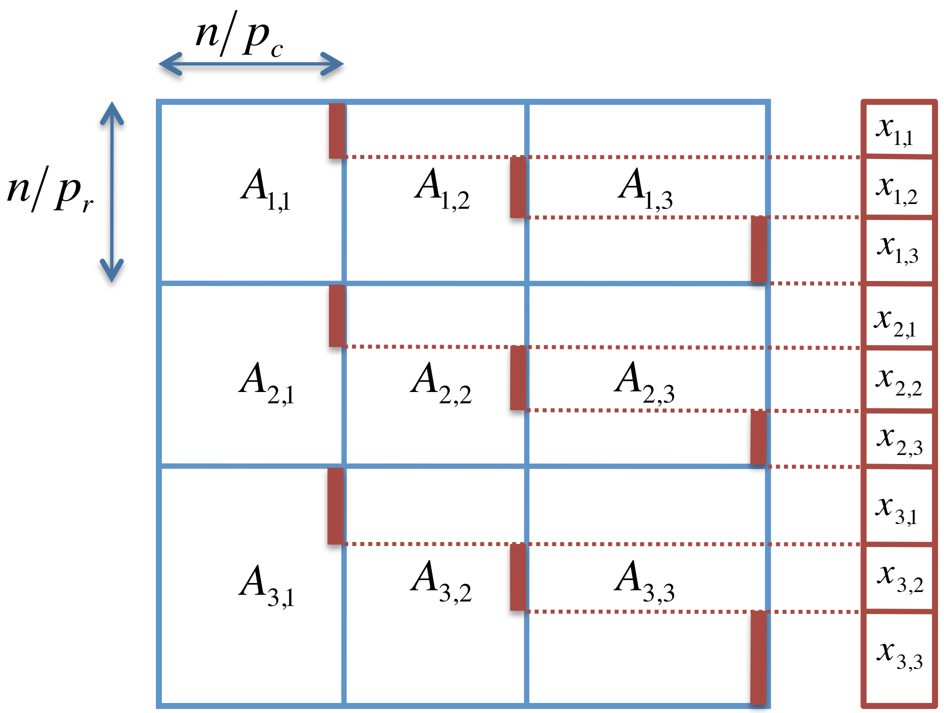
An extensible distributed-memory library offering a small but powerful set of linear algebraic operations specifically targeting graph analytics.

- 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



2D Layout for Sparse Matrices & Vectors



Matrix/vector distributions, interleaved on each other.

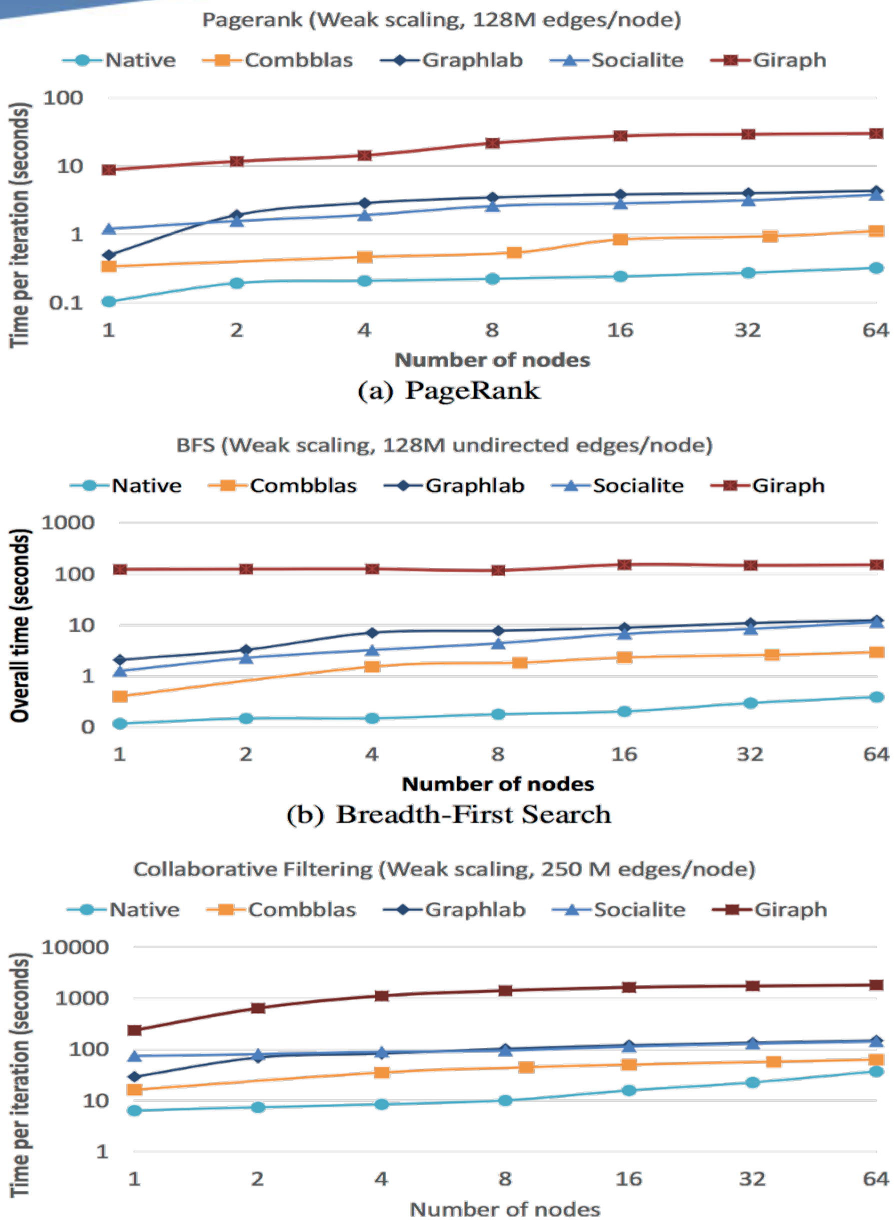
Default distribution in **Combinatorial BLAS**.

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

Benchmarking graph analytics frameworks

Combinatorial BLAS was fastest among all tested graph processing frameworks on 3 out of 4 benchmarks in an independent study by Intel.



(a) PageRank

(b) Breadth-First Search

(c) Collaborative Filtering

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

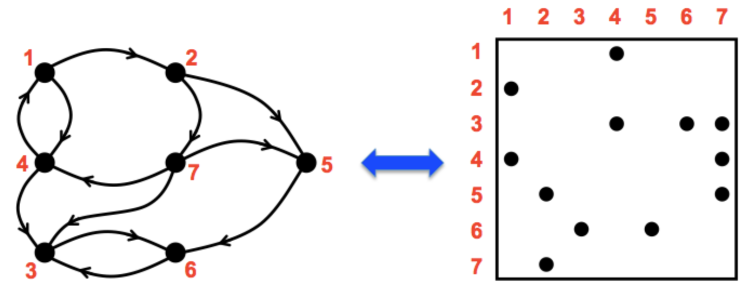
UCSB

Knowledge

Discovery

Toolbox

http://kdt.sourceforge.net/



A general graph library with operations based on linear algebraic primitives

- 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)

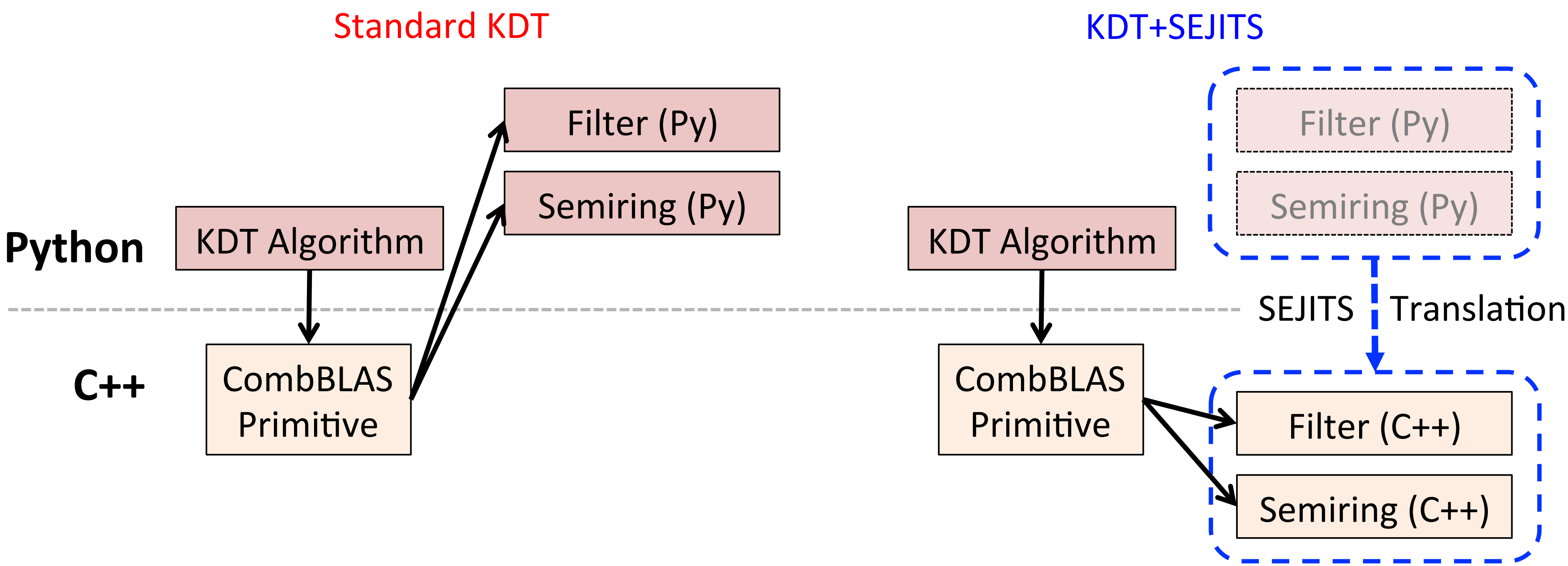
Attributed semantic graphs and filters

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

```
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')
```

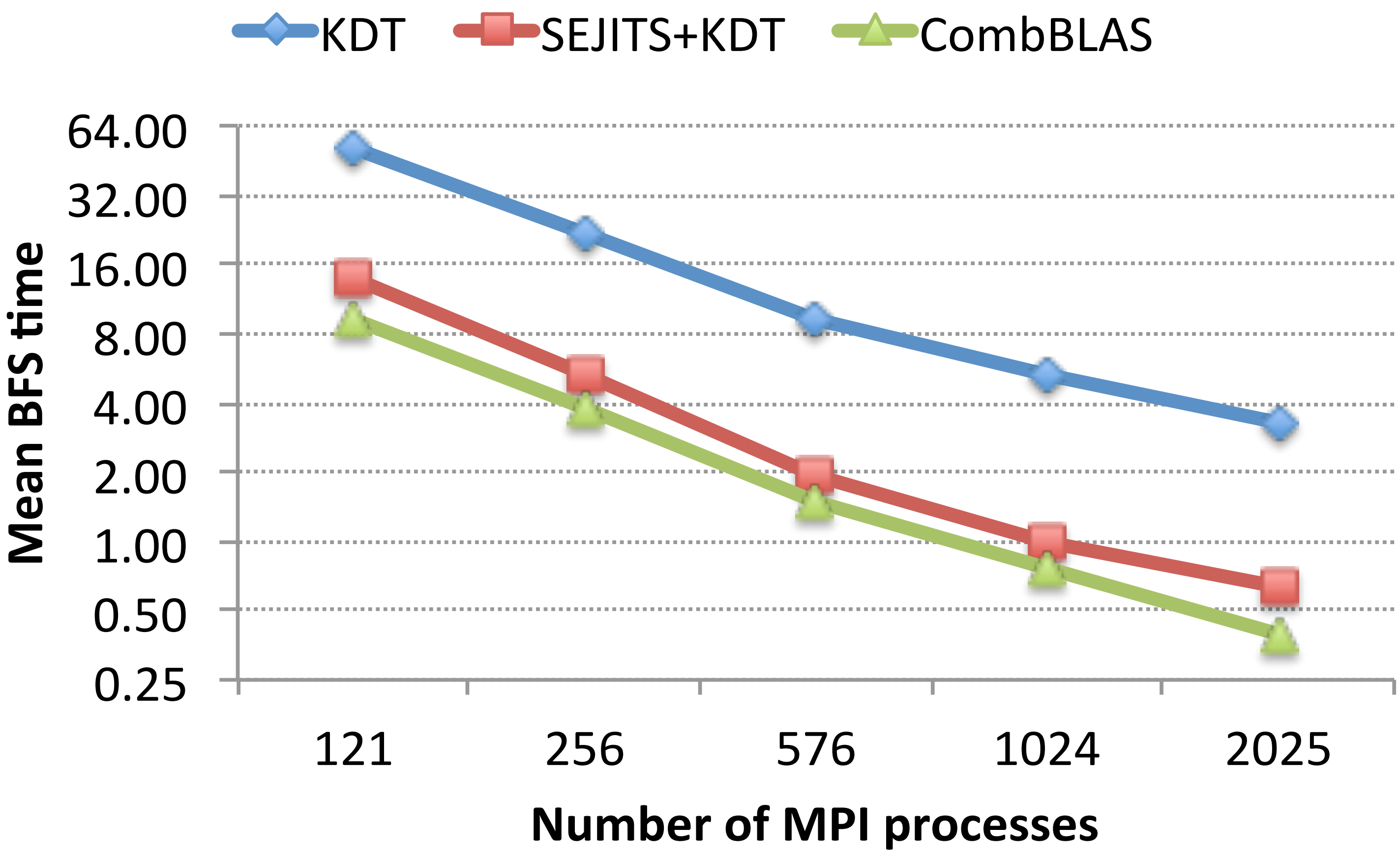
SEJITS for filter/semiring acceleration



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.

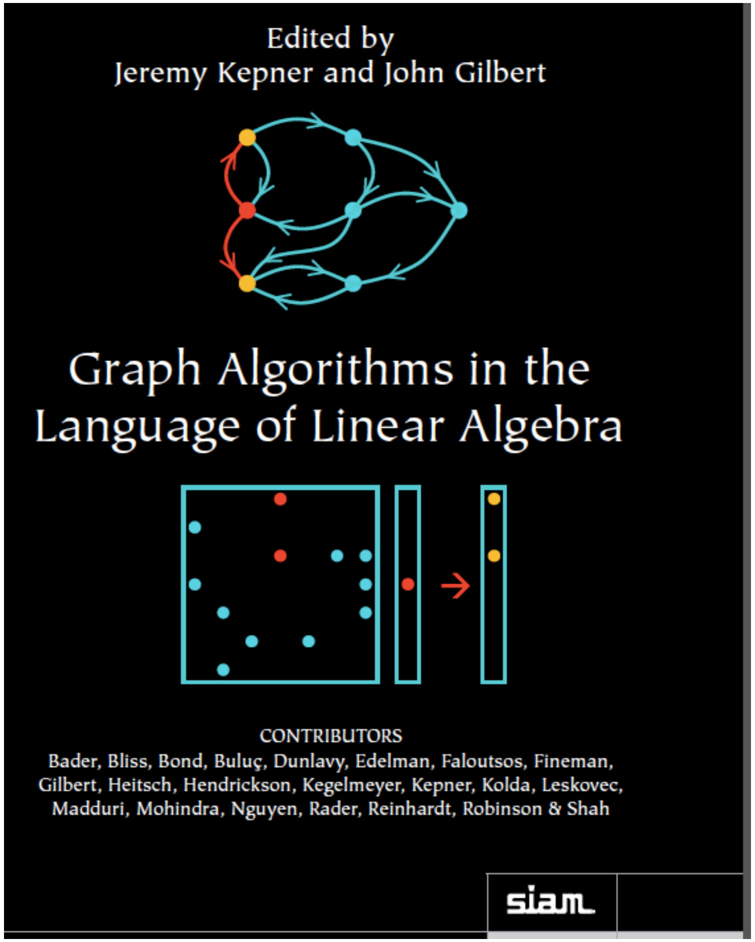
A few other graph algorithms we've implemented in linear algebraic style

- 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]

UCSB

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



UCSB

The (original) BLAS

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

BLAS 1	vector ops	Lawson, Hanson, Kincaid, Krogh, 1979	LINPACK
BLAS 2	matrix-vector ops	Dongarra, Du Croz, Hammarling, Hanson, 1988	LINPACK on vector machines
BLAS 3	matrix-matrix ops	Dongarra, Du Croz, Duff, Hammarling, 1990	LAPACK on cache based machines

- 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.

Can we standardize a "Graph BLAS"?

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:

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)

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

ND* : N-D block, cyclic w/overlap
Edge* : Edge based w/ vertex split

Function	Graph BLAS	Comb BLAS	Sparse BLAS	STINGER	D4M	SciDB	Tensor Toolbox	Julia	Graph Lab	PBGL	Networ Kit
Version		1.4.0	2006	r633	2.5	13.9	2.5	0.2.0	2.2	2-1.0	3.3
Language	any	C++	F,C,C++	C	Matlab	C++	Matlab, C++	Julia	C++	C++	C++, Python
Dim.	2	1, 2	2	1-3	2	1-100	2, 3	1,2	2	2	2
Index Base	0 or 1	0	0 or 1	0	1	±N	1	1	0	0	0
Index Type	uint64	any int	int	int64	double, string	int64	double	any int	uint64	any int	uint64
Value Type	?	user	single, double, complex	int64	bool, string, double, complex	user	bool, double, complex	user	user	user	double
Null	0	user	0	0	≤0	null	0	0	int64(-1)	user	uint64(-1)
Sparse Format	?	tuple	undef	linked list	dense, csc, tuple	RLE	dense, csc	csc	csr/csc	linked list,csr, map	csr
Parallel	?	2D block	none	block	arbitrary	ND*	none	ND*	Edge*	1D (vertex)	Iterator and loop based
+ op *** op**	user? user?	user user	+ *	user user	+,*,max,min,∩,∪	user user	+ *	user user	user user	user user	+ *

Some Graph BLAS basic functions
(names not final)

Function (CombBLAS equiv)	Parameters	Returns	Matlab notation
matmul (SpGEMM)	- sparse matrices **A** and **B** - optional unary functs	sparse matrix	**C** = **A** * **B**
matvec (SpM{Sp}V)	- sparse matrix **A** - sparse/dense vector **x**	sparse/dense vector	**y** = **A** * **x**
ewisemult, add, … (SpEWiseX)	- sparse matrices or vectors - binary funct, optional unarys	in place or sparse matrix/vector	**C** = **A** .* **B** **C** = **A** + **B**
reduce (Reduce)	- sparse matrix **A** and funct	dense vector	**y** = sum(**A**, op)
extract (SpRef)	- sparse matrix **A** - index vectors **p** and **q**	sparse matrix	**B** = **A**(**p**, **q**)
assign (SpAsgn)	- sparse matrices **A** and **B** - index vectors **p** and **q**	none	**A**(**p**, **q**) = **B**
buildMatrix (Sparse)	- list of edges/triples **(i, j, v)**	sparse matrix	**A = sparse(i, j, v, m, n)**
getTuples (Find)	- sparse matrix **A**	edge list	**[i, j, v] = find(A)**

Matrix times matrix over semiring

<u>Inputs</u>

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

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

<u>Optional Inputs</u>

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

scalar "add" function $\oplus$

scalar "multiply" function $\otimes$

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

<u>Outputs</u>

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

<u>Implements</u> $\mathbf{C} \oplus= \mathbf{A} \oplus.\otimes \mathbf{B}$

for j = 1 : N

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

If input **C** is omitted, implements

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

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

<u>Notes</u>

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

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

$\oplus$ defaults to floating-point +

$\otimes$ defaults to floating-point *

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

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 $\mathbf{A}$: $\mathbb{S}^{M\text{x}N}$ (sparse)

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

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

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

Optional Inputs

none

Outputs

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

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

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

for i = 1 : $|p|$
  for j = 1 : $|q|$
    $\mathbf{B}(\mathrm{i,j}) = \mathbf{A}(p(\mathrm{i}), q(\mathrm{j}))$

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

for i = 1 : $|p|$
  for j = 1 : $|q|$
    $\mathbf{A}(p(\mathrm{i}), q(\mathrm{j})) = \mathbf{B}(\mathrm{i,j})$

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

<u>Inputs</u>

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

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

<u>Optional Inputs</u>

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

scalar "add" function $\oplus$

scalar "multiply" function $\otimes$

<u>Outputs</u>

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

<u>Implements</u> $\mathbf{C} \oplus= \mathbf{A} \otimes \mathbf{B}$

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

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

<u>Notes</u>

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

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

$\oplus$ defaults to floating-point +

$\otimes$ defaults to floating-point *

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

SpEWiseX: matrix elementwise

M=1 or N=1: vector elementwise

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

Apply / update

Inputs

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

Optional Inputs

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

scalar "add" function $\oplus$

unary function f()

Outputs

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

Implements $\mathbf{C} \oplus= f(\mathbf{A})$

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 $\mathbf{C}$ is omitted, implements

$\mathbf{C} = f(\mathbf{A})$

Notes

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

$\mathbb{S}$ 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**: $\mathbb{S}^{MxN}$ (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 $\mathbf{c}(i) \oplus= \bigoplus_j \mathbf{A}(i,j)$

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

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

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

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”

UCSB

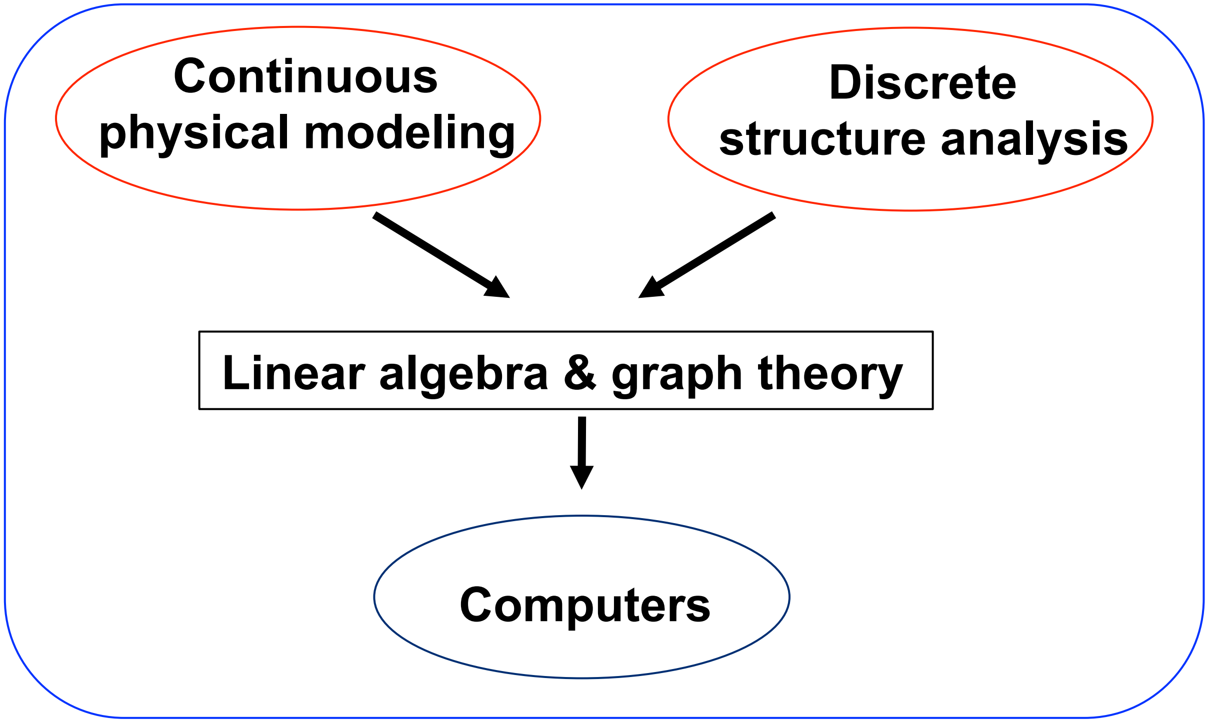
Conclusions

- 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 …

Ariful Azad, David Bader, Lucas Bang, Jon Berry, Eric Boman, Aydin Buluc, John Conroy, Kevin Deweese, Erika Duriakova, Armando Fox, Joey Gonzalez, Shoaib Kamil, Jeremy Kepner, Tristan Konolige, Adam Lugowski, Tim Mattson, Brad McRae, Henning Meyerhenke, Dave Mizell, Jose Moreira, Lenny Oliker, Carey Priebe, Steve Reinhardt, Lijie Ren, Eric Robinson, Viral Shah, Veronika Strnadova-Neely, Yun Teng, Joshua Vogelstein, Drew Waranis, Sam Williams

Thank You!



http://graphblas.org