UCSB

Sparse Matrices for High Performance Graph Analytics

John R. Gilbert
University of California, Santa Barbara

Purdue University
January 27, 2016

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

Thanks …

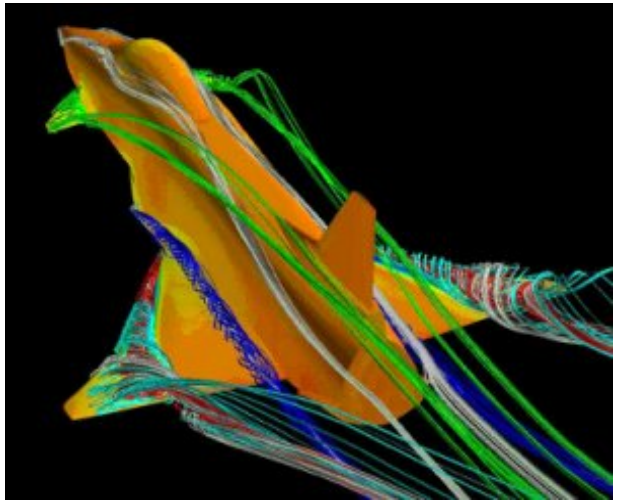
Ariful Azad, David Bader, Lucas Bang, Jon Berry, Eric Boman, Aydin Buluc, John Conroy, Kevin Deweese, Erika Duriakova, Armando Fox, Joey Gonzalez, Shoaib Kamil, Jeremy Kepner, Tristan Konolige, Manoj Kumar, Adam Lugowski, Tim Mattson, Brad McRae, Henning Meyerhenke, Dave Mizell, Jose Moreira, Lenny Oliker,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

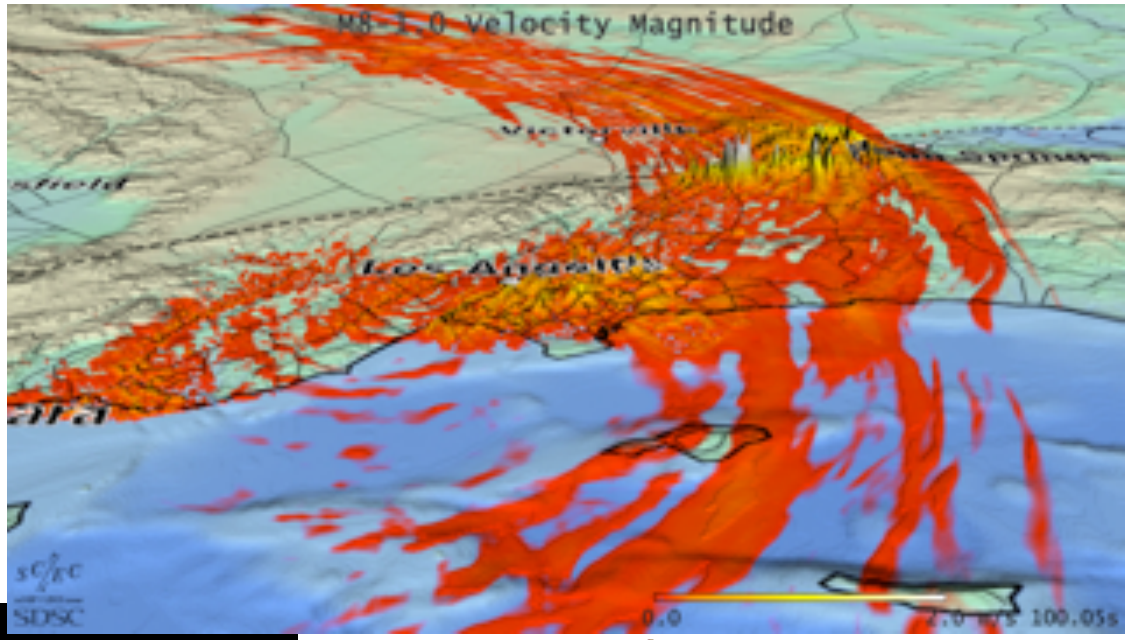
UCSB

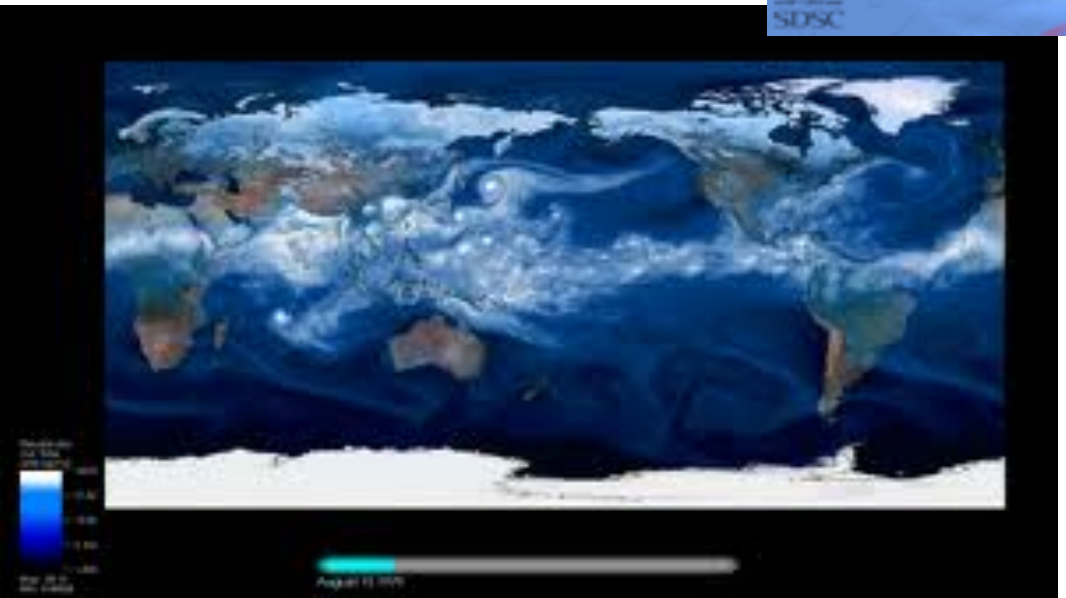
Outline

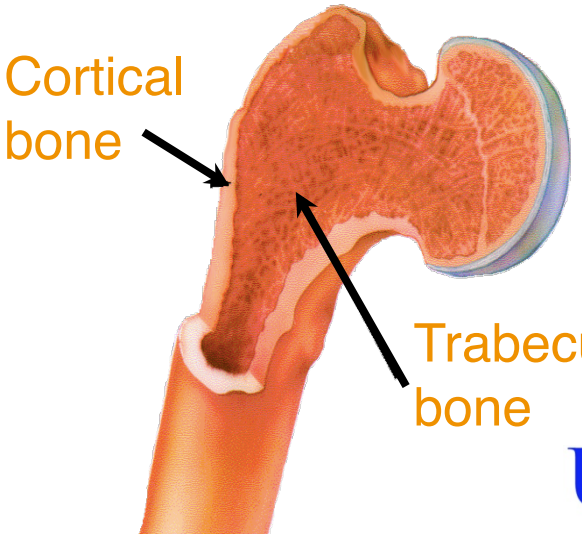
- Motivation: Graph applications
- Mathematics: Sparse matrices for graph algorithms
- Software: CombBLAS, KDT, QuadMat
- Standards: The Graph BLAS effort

Computational models of the physical world





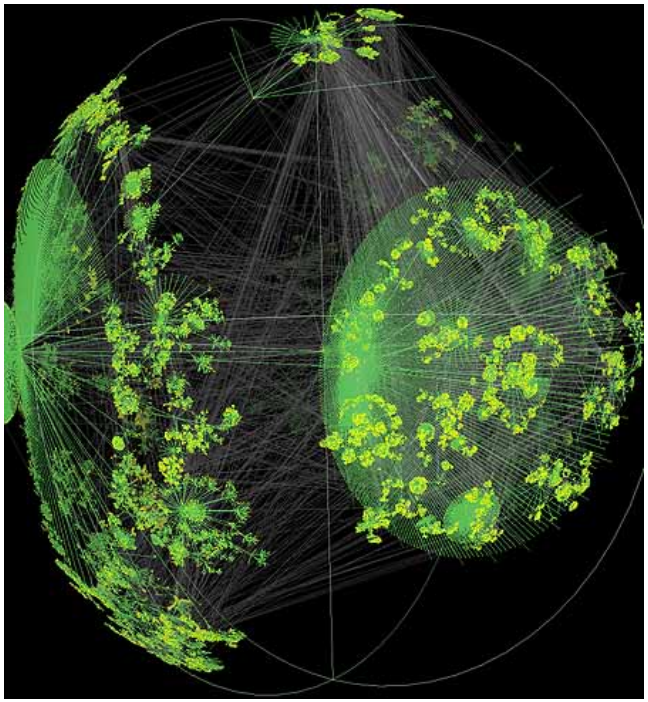




UCSB

Large graphs are everywhere…

- Internet structure
- Social interactions



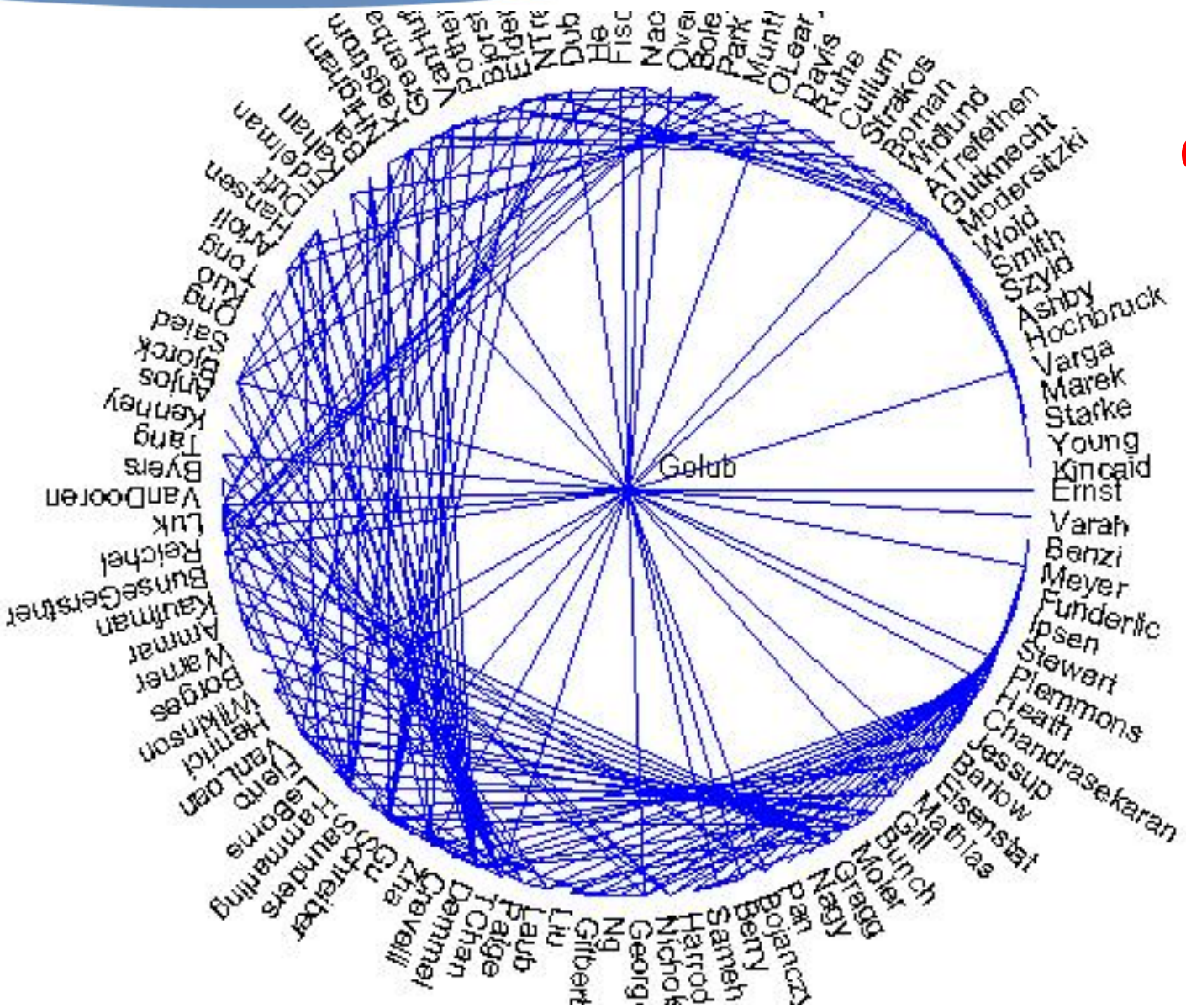
WWW snapshot, courtesy Y. Hyun

- Scientific datasets: biological, chemical, cosmological, ecological, …

Yeast protein interaction network, courtesy H. Jeong

UCSB

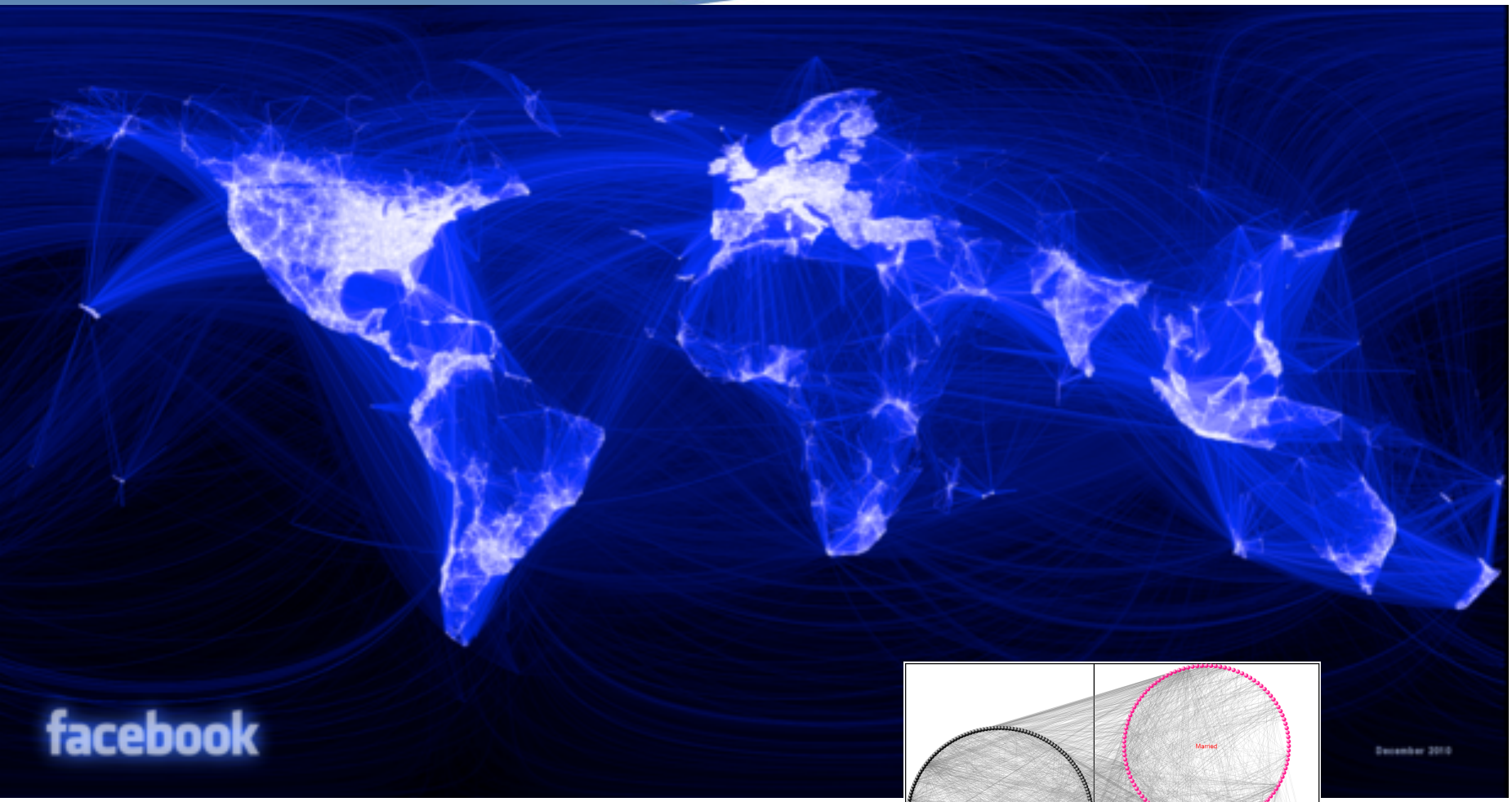
Social network analysis (1993)

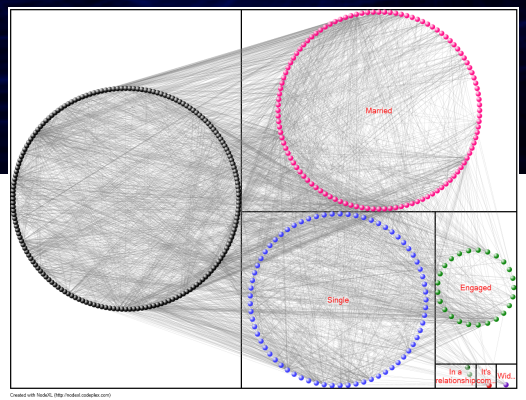


Co-author graph from 1993 Householder symposium

UCSB

Social network analysis (2016)

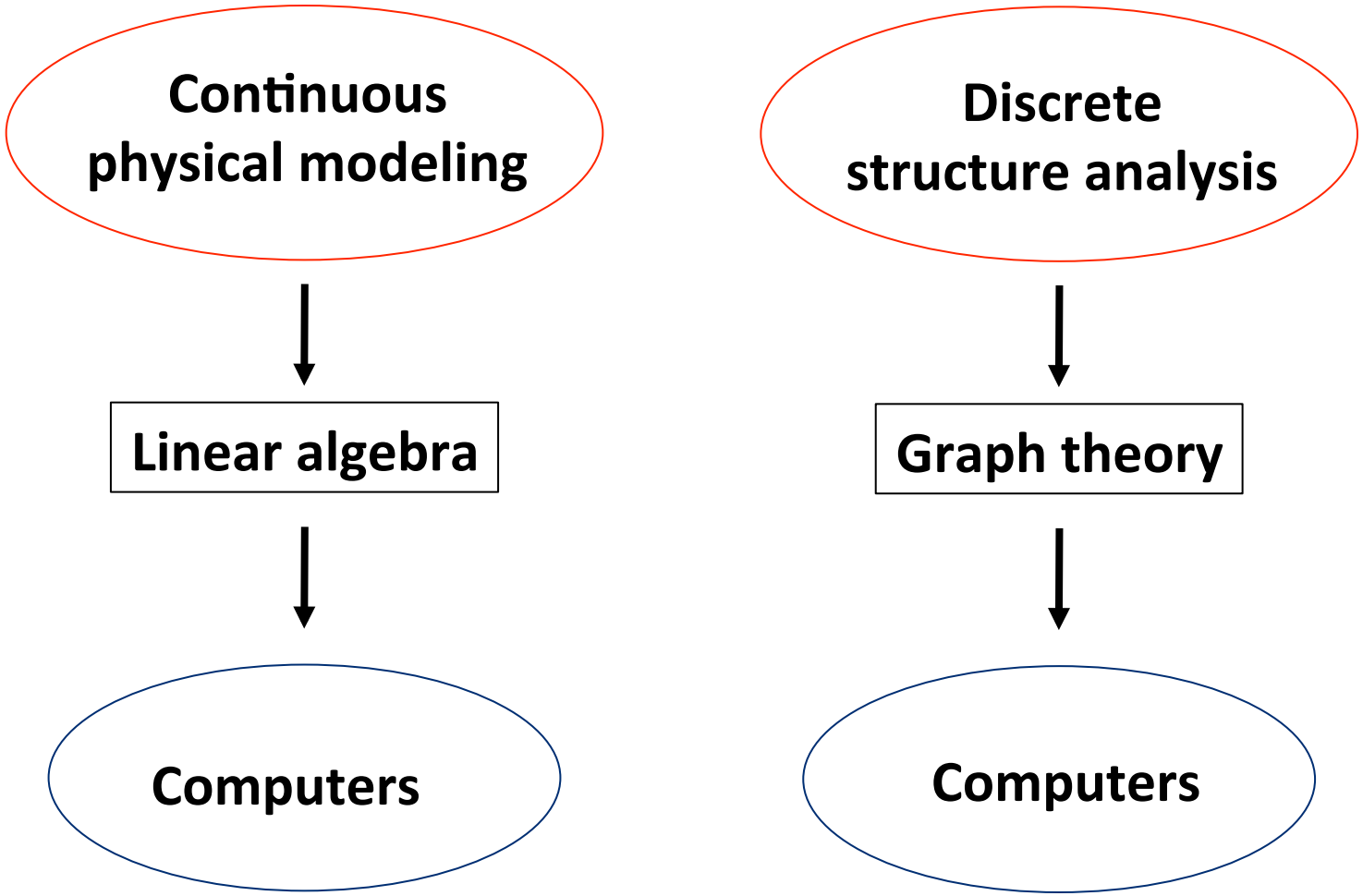




Facebook graph:
> 1,200,000,000 vertices

UCSB

The middleware challenge for graph analysis

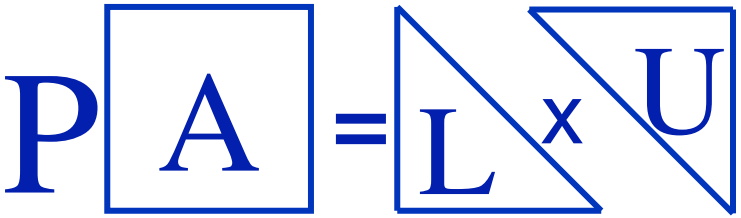


UCSB

Top 500 List (November 2015)

Top500 Benchmark:

Solve a large system
of linear equations
by Gaussian elimination

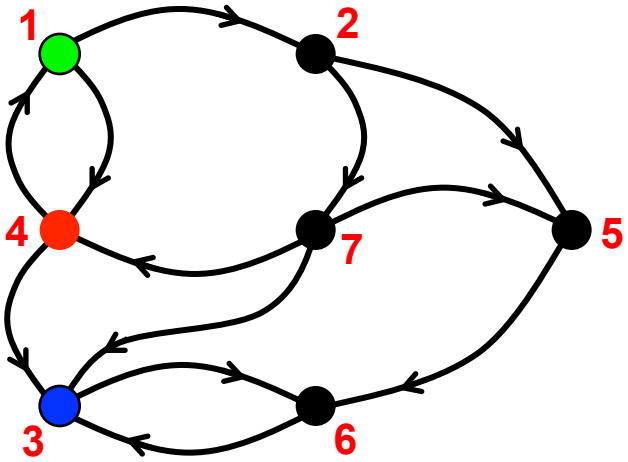


RANK	SITE	SYSTEM	CORES	RMAX (TFLOP/S)
1	National Super Computer Center in Guangzhou China	**Tianhe-2 (MilkyWay-2)** - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT	3,120,000	33,862.7
2	DOE/SC/Oak Ridge National Laboratory United States	**Titan** - Cray XK7 , Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.	560,640	17,590.0
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6
6	DOE/NNSA/LANL/SNL United States	**Trinity** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	301,056	8,100.9
7	Swiss National Supercomputing Centre (CSCS) Switzerland	**Piz Daint** - Cray XC30, Xeon E5-2670 8C 2.600GHz, Aries interconnect , NVIDIA K20x Cray Inc.	115,984	6,271.0
8	HLRS - Höchstleistungsrechenzentrum Stuttgart Germany	**Hazel Hen** - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect Cray Inc.	185,088	5,640.2
9	King Abdullah University of Science and Technology Saudi Arabia	**Shaheen II** - Cray XC40, Xeon E5-2698v3 16C 2.3GHz, Aries interconnect Cray Inc.	196,608	5,537.0
10	Texas Advanced Computing Center/Univ. of Texas	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P	462,462	5,168.1

Graph 500 List (November 2015)

Graph500 Benchmark:

Breadth-first search in a large power-law graph

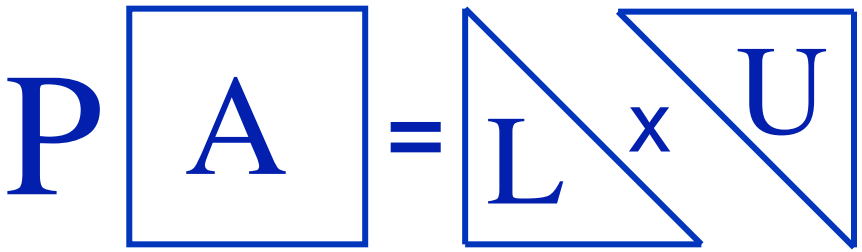


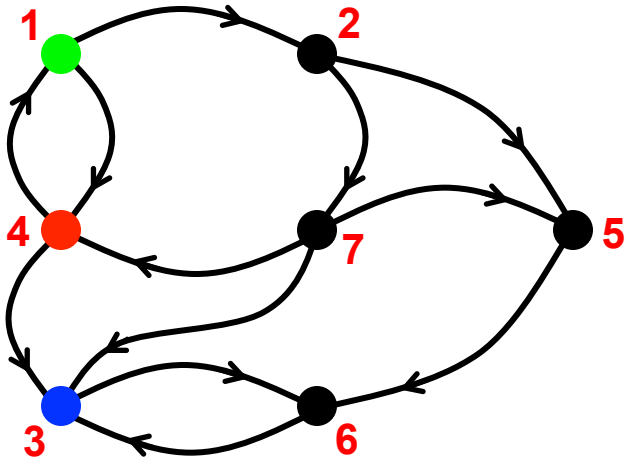
No.	Rank	Machine	Installation Site	Number of nodes	Number of cores	Problem scale	GTEPS
1	1	K computer (Fujitsu - Custom)	RIKEN Advanced Institute for Computational Science (AICS)	82944	663552	40	38621.4
2	2	DOE/NNSA/LLNL Sequoia (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Lawrence Livermore National Laboratory	98304	1572864	41	23751
3	3	DOE/SC/Argonne National Laboratory Mira (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Argonne National Laboratory	49152	786432	40	14982
4	4	JUQUEEN (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Forschungszentrum Juelich (FZJ)	16384	262144	38	5848
5	5	Fermi (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	CINECA	8192	131072	37	2567
6	6	Tianhe-2 (MilkyWay-2) (National University of Defense Technology - MPP)	Changsha, China	8192	196608	36	2061.48
7	7	Turing (IBM - BlueGene/Q, Power BQC 16C 1.60GHz)	CNRS/IDRIS-GENCI	4096	65536	36	1427
8	7	Blue Joule (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Science and Technology Facilities Council - Daresbury Laboratory	4096	65536	36	1427

Floating-Point vs. Graphs, November 2015

34 Petaflops

38 Terateps





34 Peta / 38 Tera is about 900.

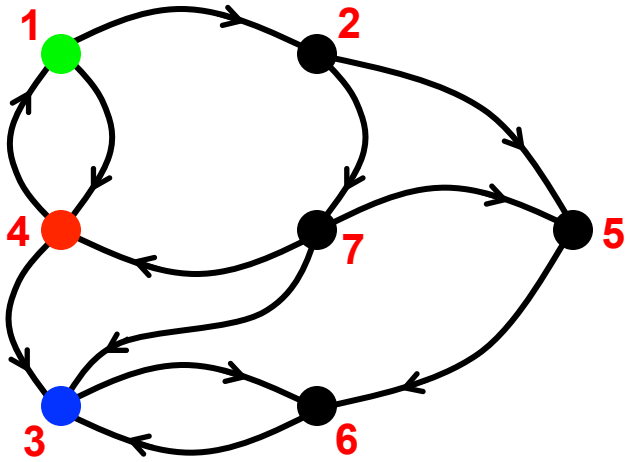
UCSB

Floating-Point vs. Graphs, November 2015

34 Petaflops

$$P\boxed{A} = L \times U$$

38 Terateps



Nov 2015:	**34 Peta / 38 Tera**	**~ 900**
Nov 2010:	**2.5 Peta / 6.6 Giga**	**~ 380,000**

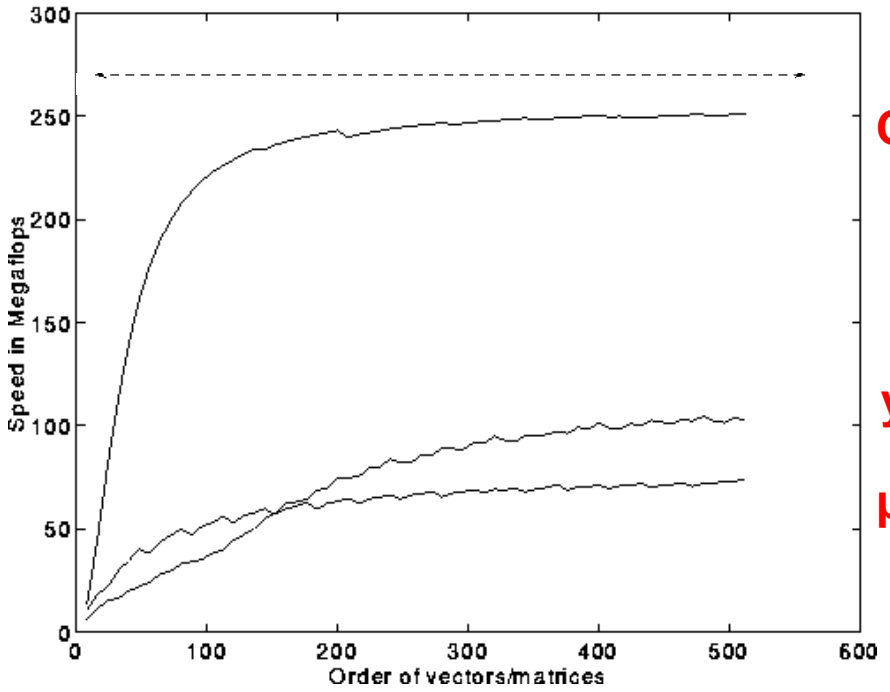
UCSB

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



C = A*B

y = A*x

μ = x^T y

UCSB

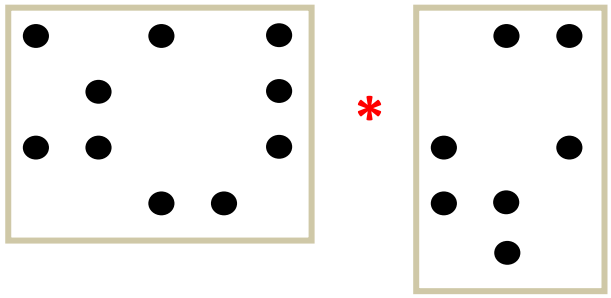
The case for sparse matrices

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

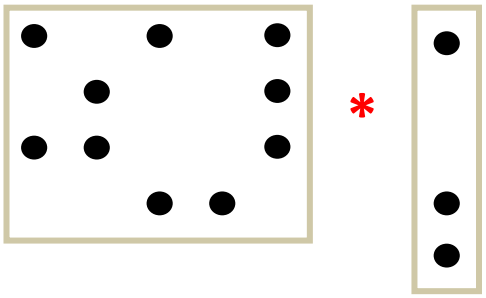
Vertex/edge graph computations	Graphs in the language of linear algebra
Unpredictable, data-driven communication patterns	Fixed communication patterns
Irregular data accesses, with poor locality	Matrix block operations exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse grained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

Sparse array primitives for graphs

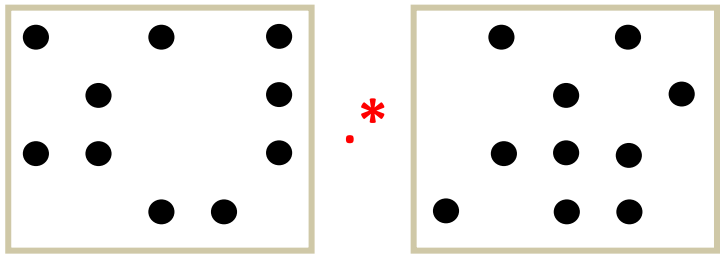
Sparse matrix-sparse matrix multiplication



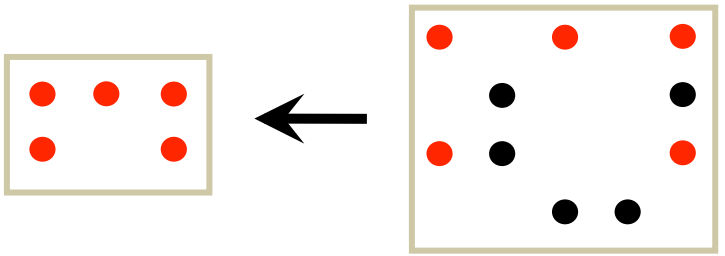
Sparse matrix-sparse vector multiplication



Element-wise operations

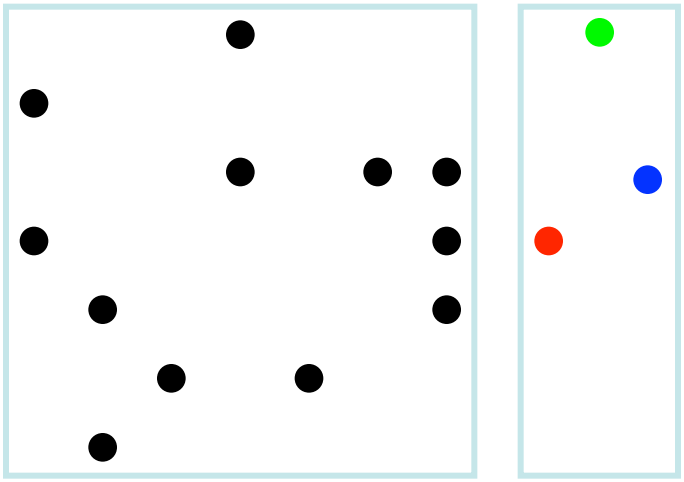


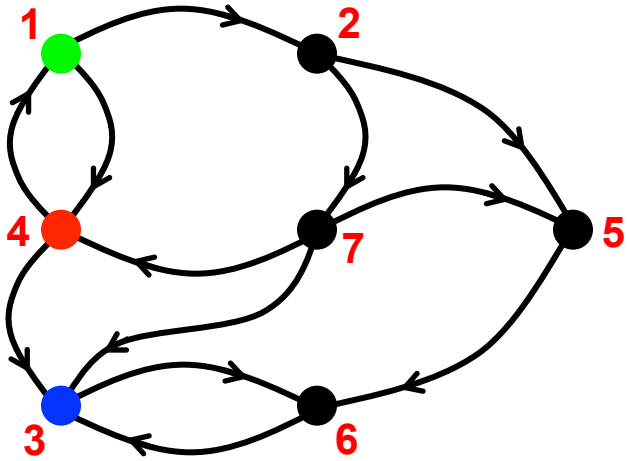
Sparse matrix indexing



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

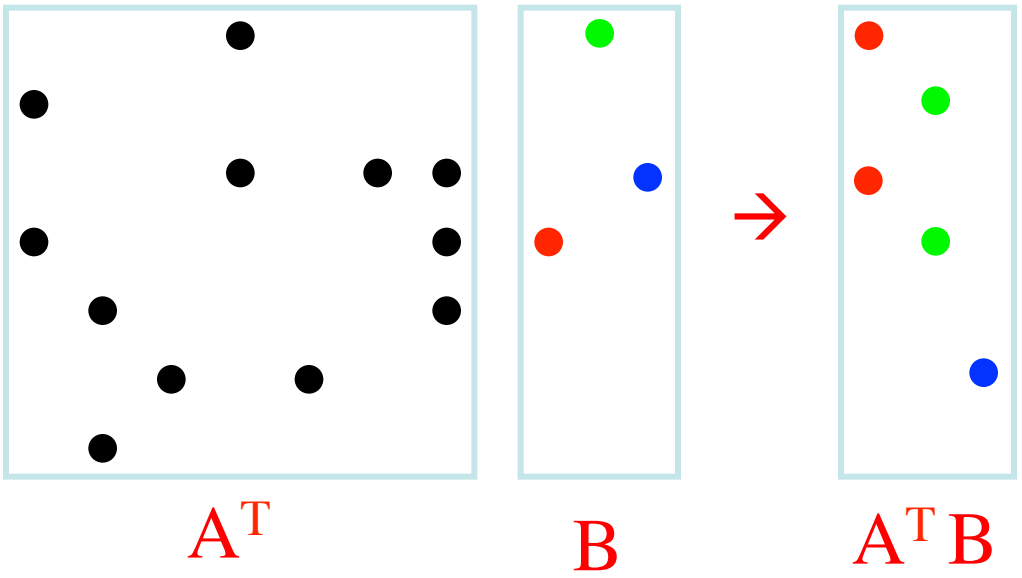
Multiple-source breadth-first search

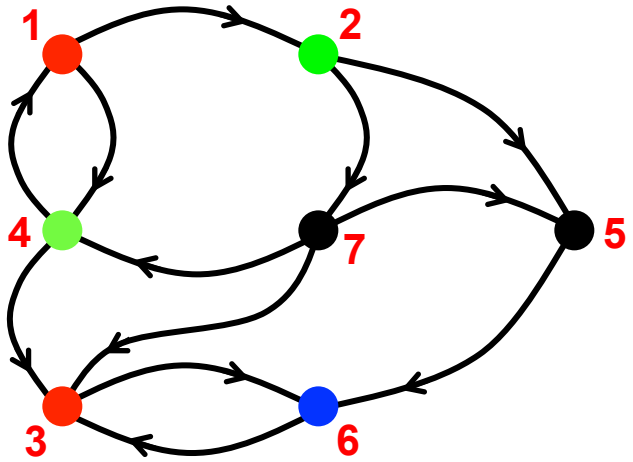




UCSB

Multiple-source breadth-first search





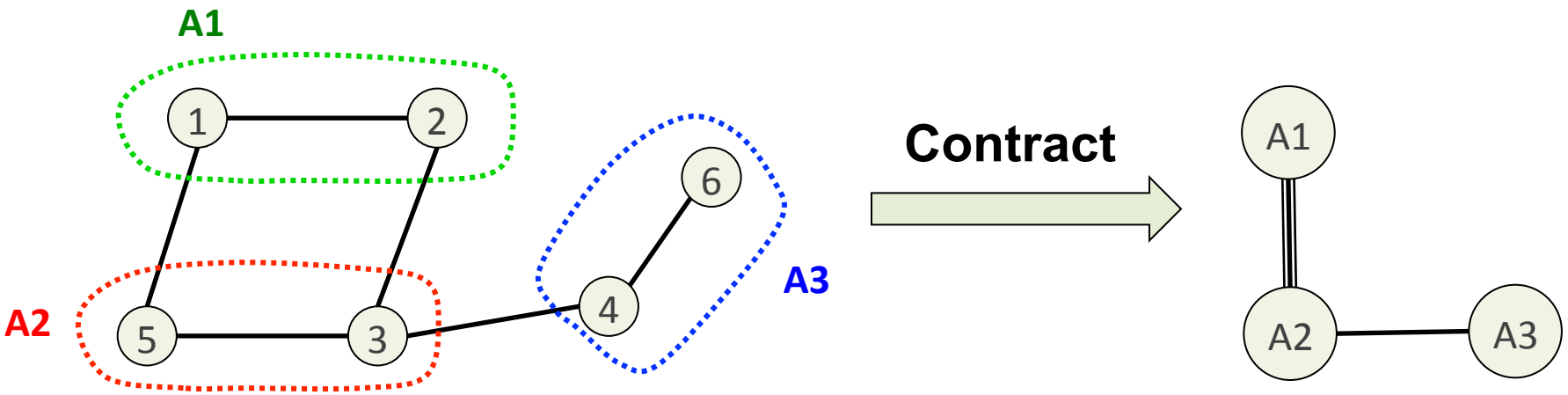
- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

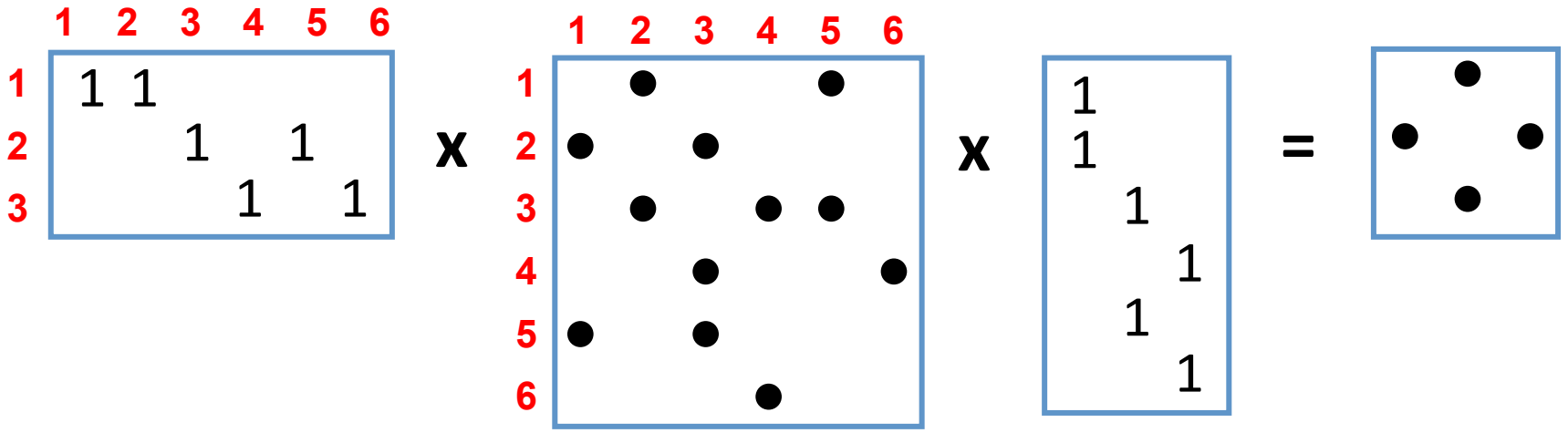
UCSB

Examples of semirings in graph algorithms

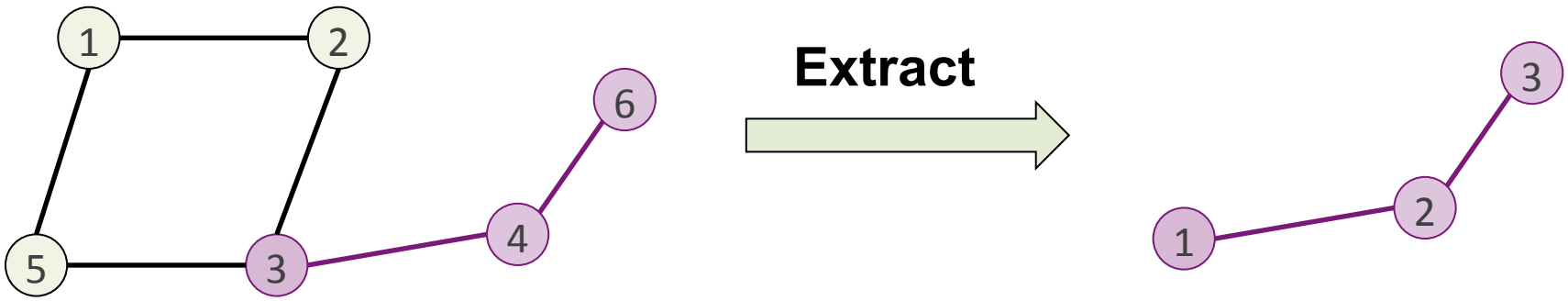
(“values”: edge/vertex attributes, “add”: vertex data aggregation, “multiply”: edge data processing)	General schema for user-specified computation at vertices and edges
Real field: (R, +, *)	Numerical linear algebra
Boolean algebra: ({0 1}, |, &)	Graph traversal
Tropical semiring: (R U {∞}, min, +)	Shortest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

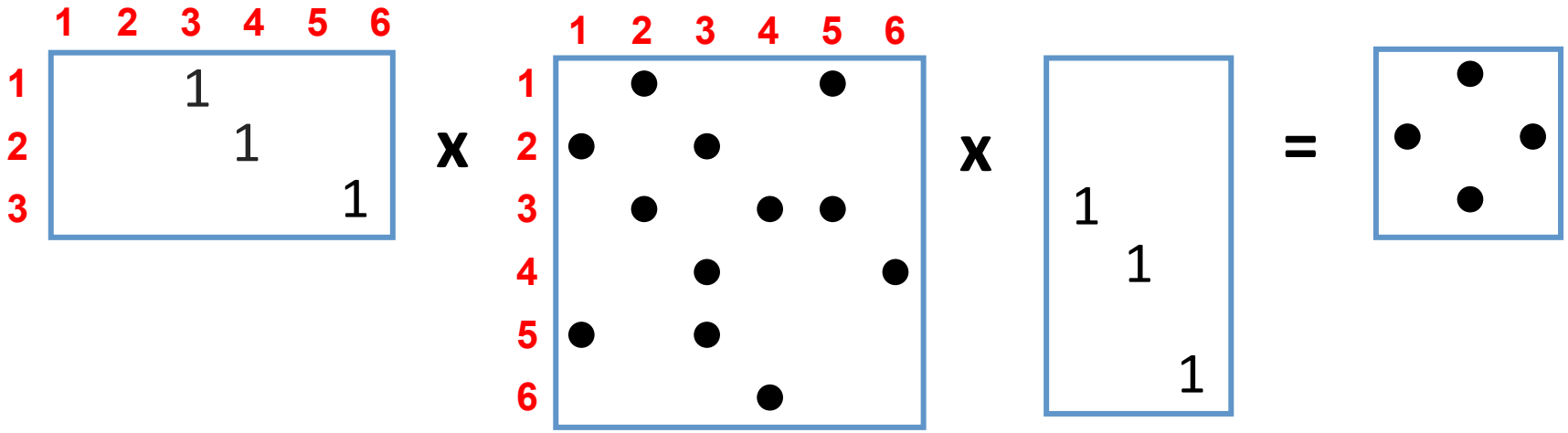
Graph contraction via sparse triple product



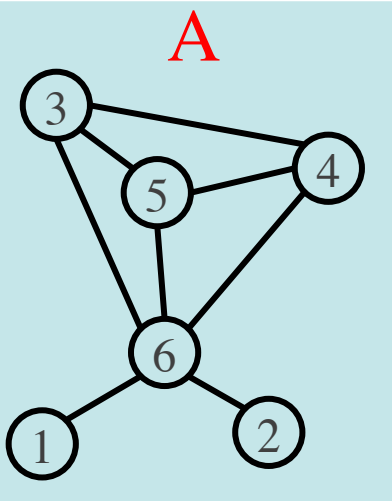


Subgraph extraction via sparse triple product





Counting triangles (clustering coefficient)

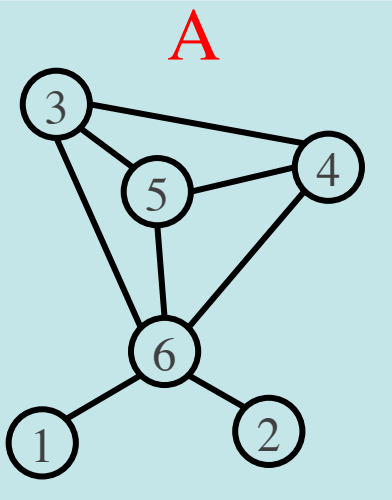


Clustering coefficient:

- 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

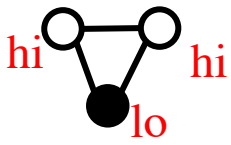
Counting triangles (clustering coefficient)



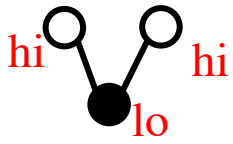
Clustering coefficient:

- 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

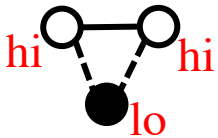
"Cohen's" algorithm to count triangles:



- Count triangles by lowest-degree vertex.



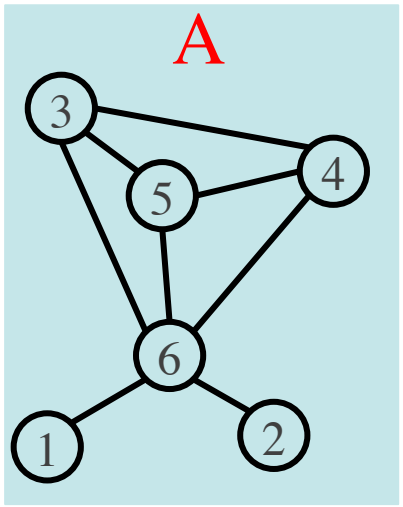
- Enumerate "low-hinged" wedges.



- Keep wedges that close.

UCSB

Counting triangles (clustering coefficient)

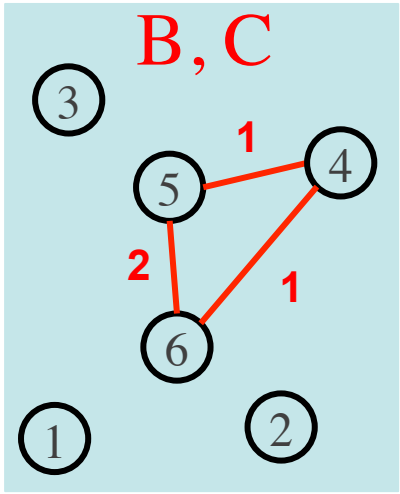


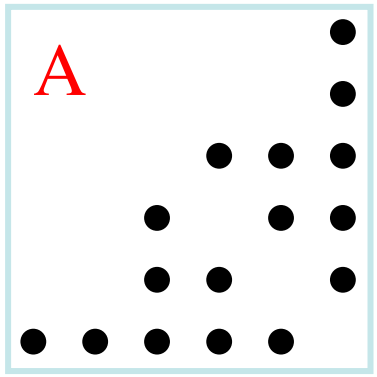
$A = L + U$ (hi->lo + lo->hi)

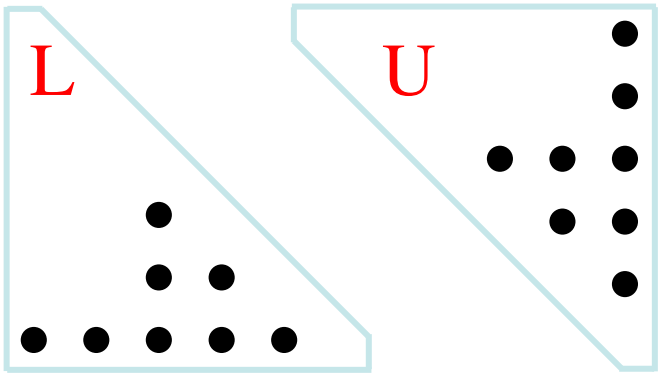
$L \times U = B$ (wedge, low hinge)

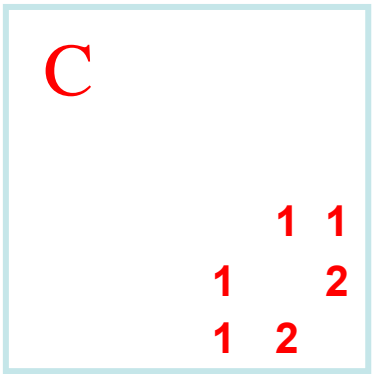
$A \wedge B = C$ (closed wedge)

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





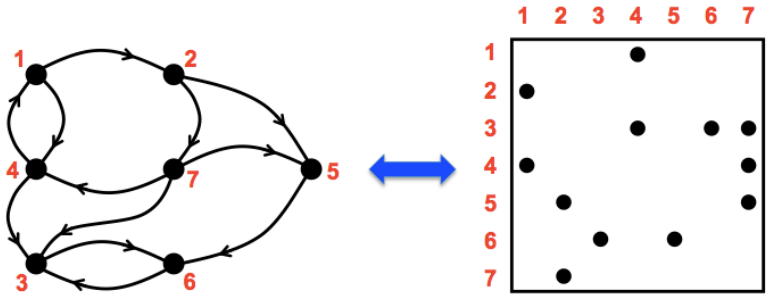




UCSB

Combinatorial BLAS

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

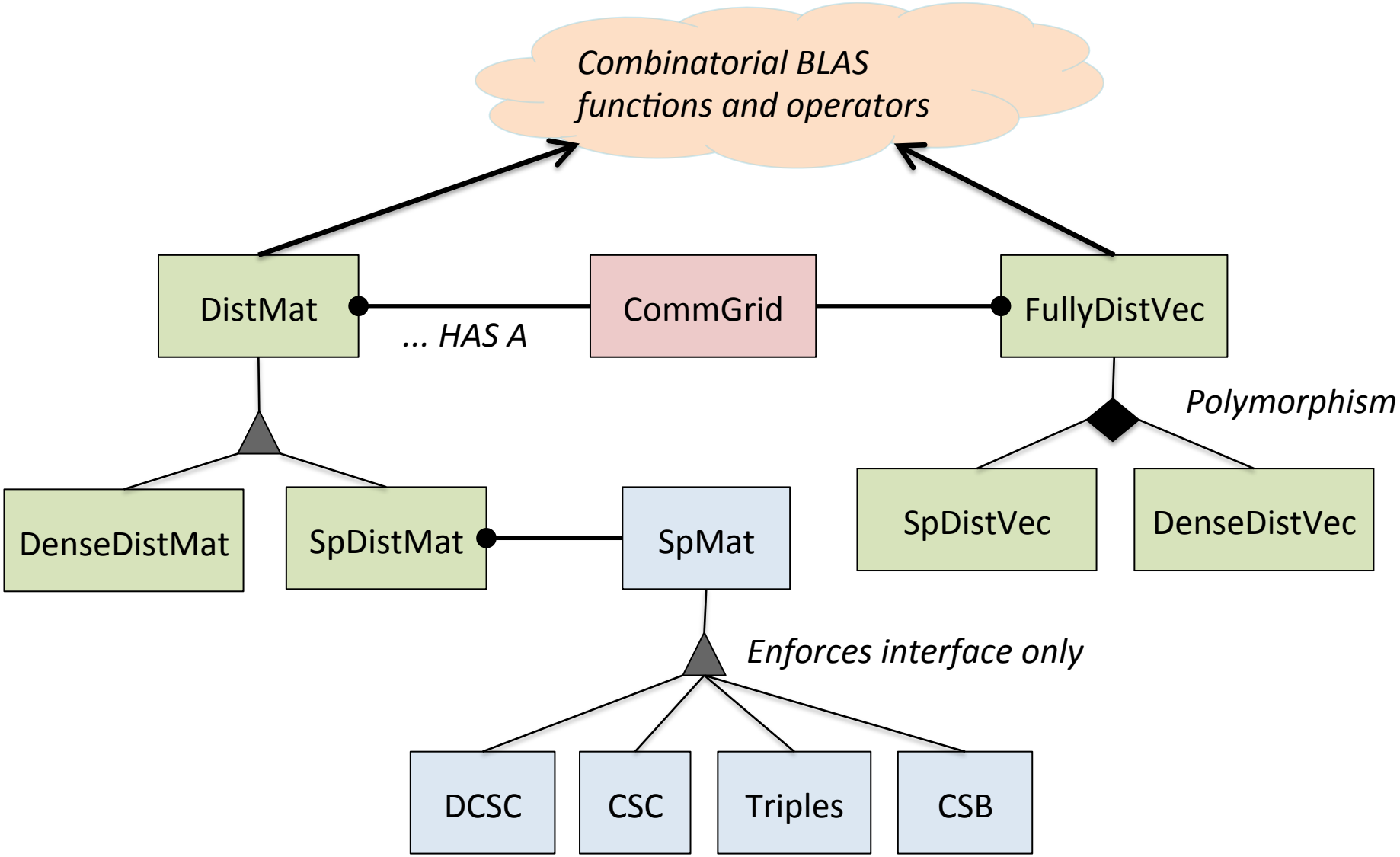


An extensible distributed-memory library offering a small but powerful set of linear algebraic operations specifically targeting graph analytics.

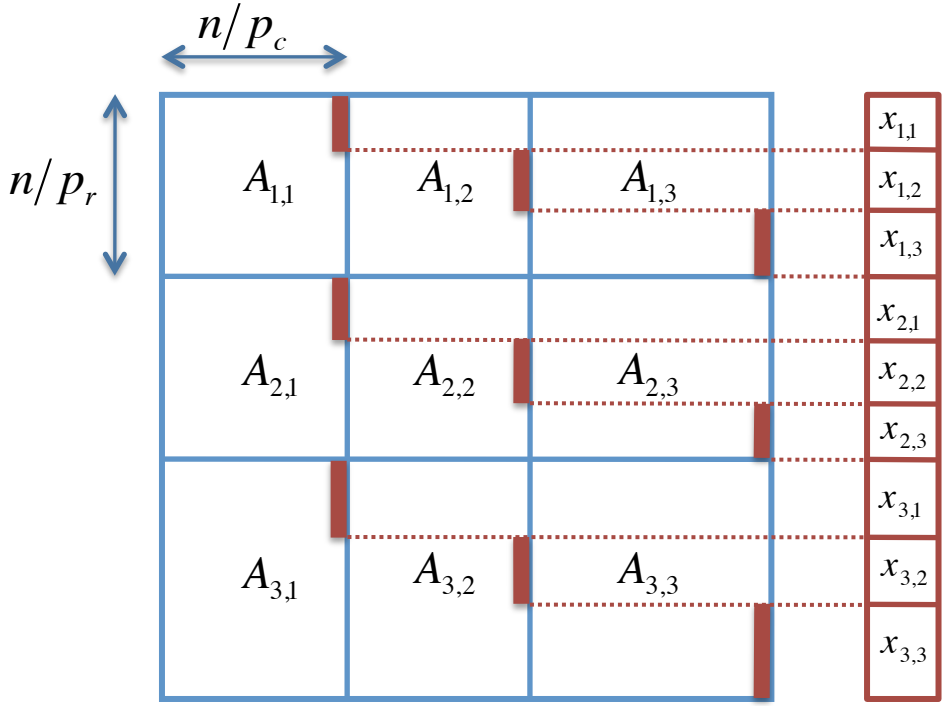
- Aimed at graph algorithm designers/programmers who are not expert in mapping algorithms to parallel hardware.
- Flexible, templated C++ interface.
- Scalable performance from laptop to 100,000-processor HPC.

- Open source software, version 1.5.0 released January 2016.

Combinatorial BLAS in distributed memory



2D Layout for Sparse Matrices & Vectors



Matrix/vector distributions, interleaved on each other.

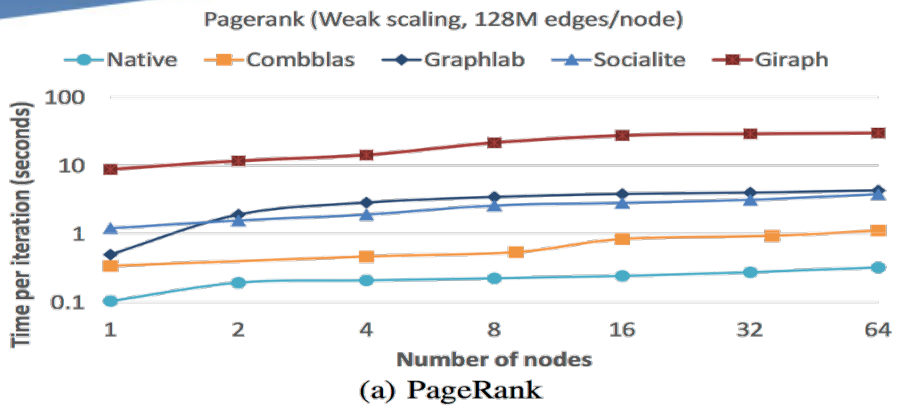
Default distribution in **Combinatorial BLAS**.

Scalable with increasing number of processes

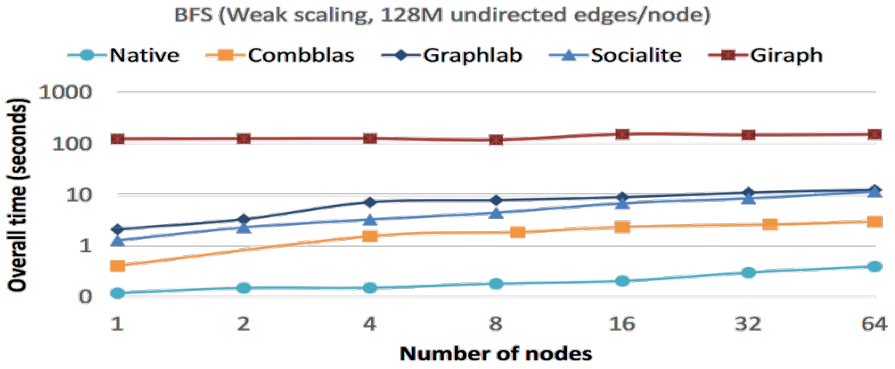
- 2D matrix layout wins over 1D with large core counts and with limited bandwidth/compute
- 2D vector layout sometimes important for load balance

Benchmarking graph analytics frameworks

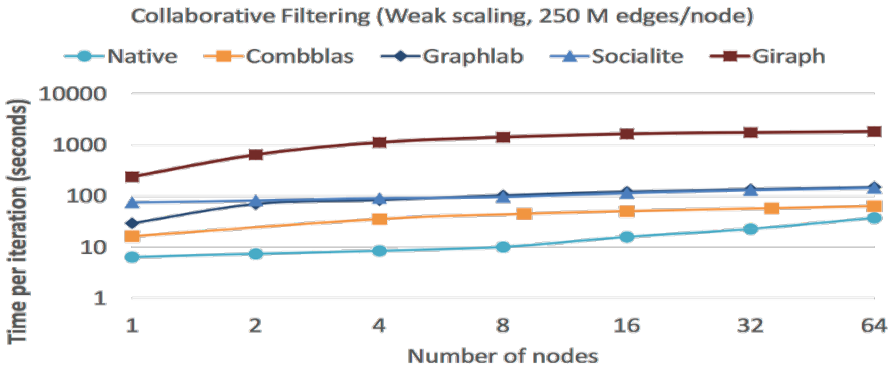
Combinatorial BLAS was fastest among all tested graph processing frameworks on 3 out of 4 benchmarks in an independent study by Intel.



(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" (Jan 2016)

- IBM (T.J.Watson, Zurich, & Tokyo)
- Intel
- Cray
- Microsoft
- Stanford
- MIT
- UC Berkeley
- Carnegie-Mellon
- Georgia Tech
- Purdue
- Ohio State
- U Texas Austin
- NC State
- UC San Diego
- UC Merced
- UC Santa Barbara
- Berkeley Lab
- Sandia Labs
- Columbia
- U Minnesota
- Duke
- Indiana U
- Mississippi State
- SEI
- Paradigm4
- Mellanox
- IHPC (Singapore)
- Tokyo Inst of Technology
- Chinese Academy of Sciences
- U Canterbury (New Zealand)
- King Fahd U (Saudi Arabia)
- Bilkent U (Turkey)
- U Ghent (Belgium)

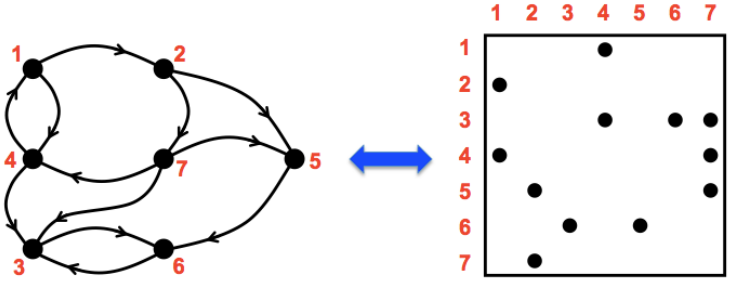
UCSB

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

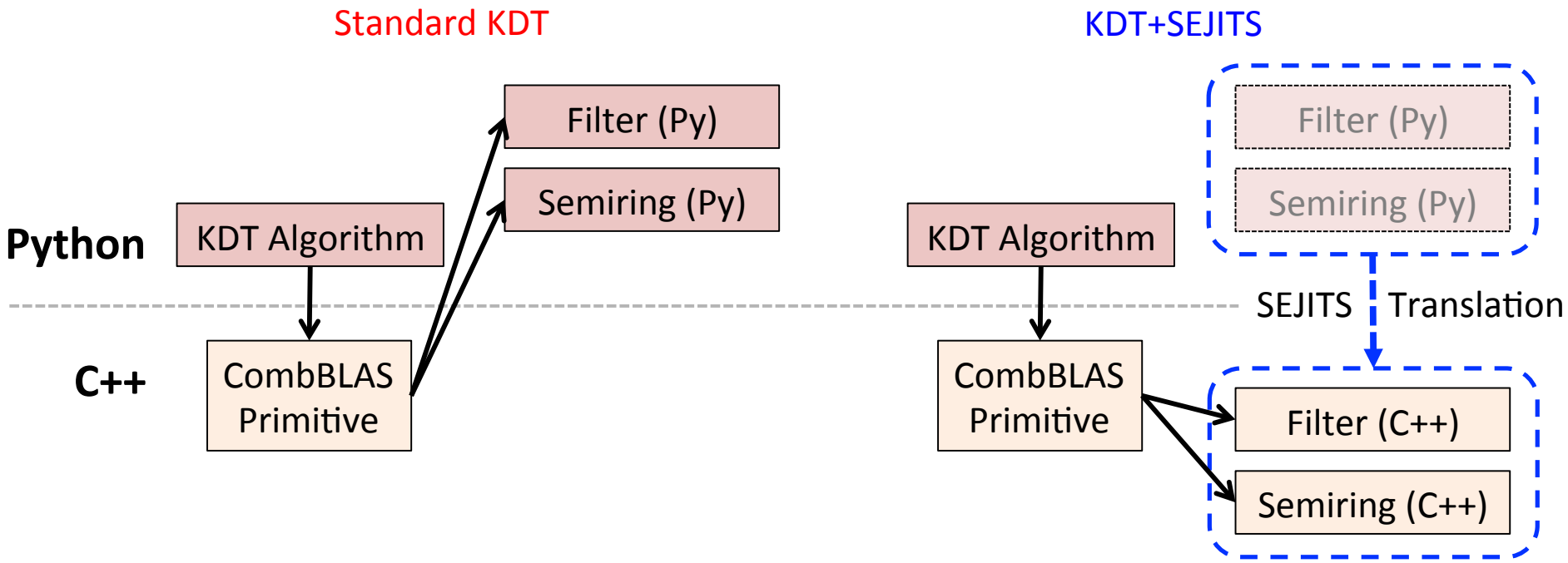
Example:

- 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')
```

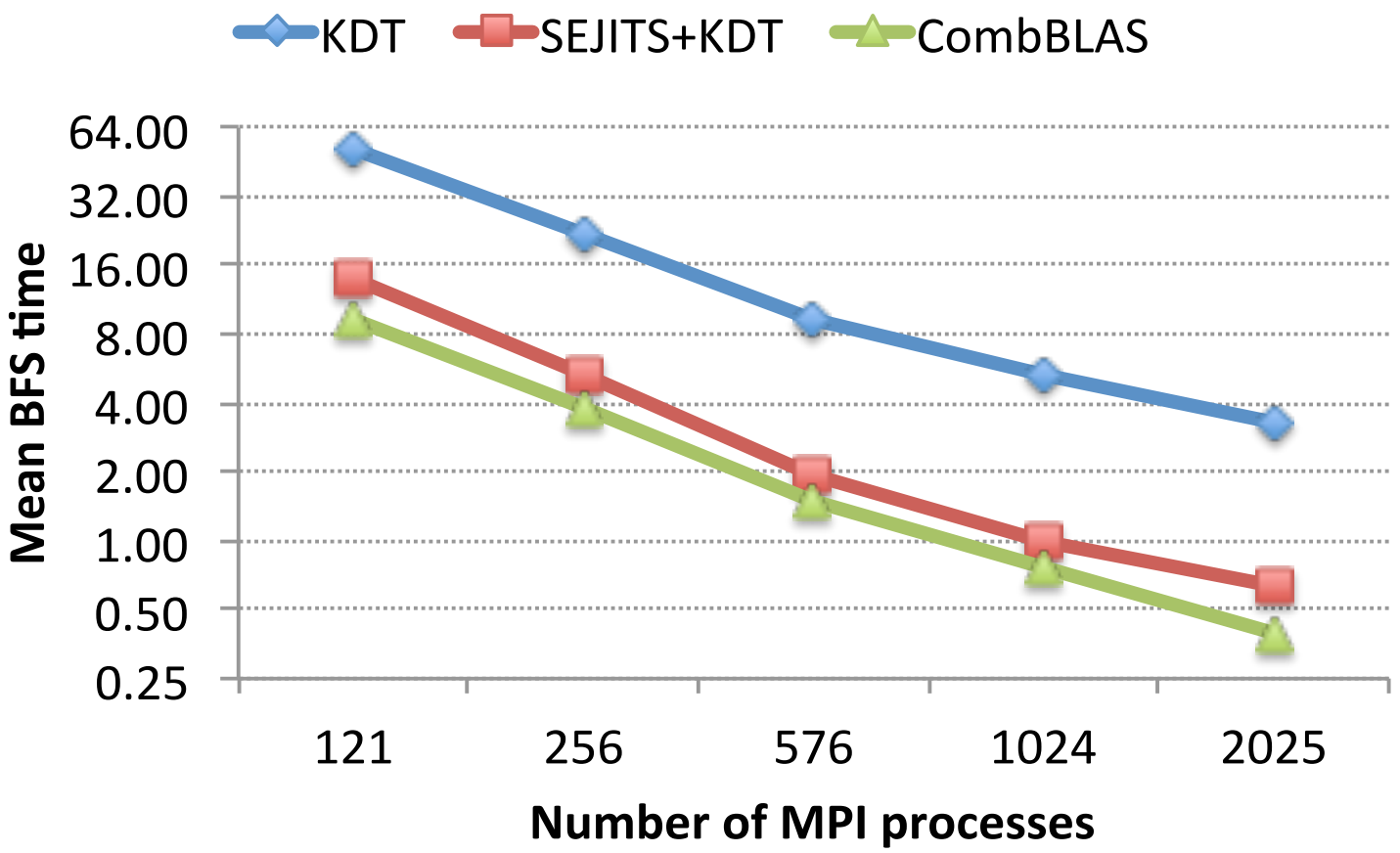
SEJITS for filter/semiring acceleration



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]

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 and HPEC – next in May 2016
- Periodic working group telecons and meetings
- Graph BLAS Forum: http://graphblas.org

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

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 $\oplus$

scalar "multiply" function $\otimes$

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

Outputs

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

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

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

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:

SpGEMM: sparse matrix times sparse matrix

SpMSpV: sparse matrix times sparse vector

SpMV: sparse matrix times dense vector

SpMM: sparse matrix times dense matrix

Breadth-first search with (draft) Graph BLAS

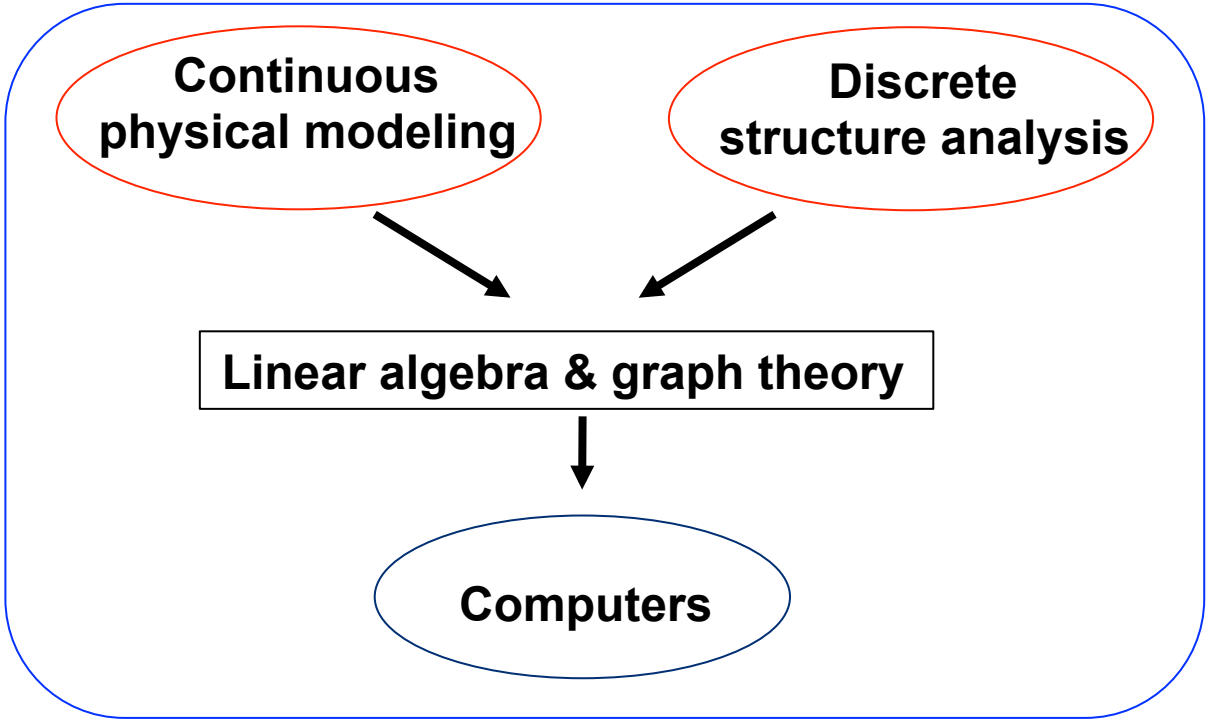
```
7  GrB_info BFS(GrB_Vector *v, GrB_Matrix A, GrB_index s)
8  /*
9   * Given a boolean n x n adjacency matrix A and a source vertex s, performs a BFS traversal
10  * of the graph and sets v[i] to the level in which vertex i is visited (v[s] == 1).
11  * If i is not reacheable from s, then v[i] = 0. (Vector v should be empty on input.)
12  */
13 {
14   GrB_index n;
15   GrB_Matrix_nrows(&n,A);                    // n = # of rows of A
16
17   GrB_Vector_new(v,GrB_INT32,n);             // Vector<int32_t> v(n)
18   GrB_assign(v,0);                           // v = 0
19
20   GrB_Vector q;                              // vertices visited in each level
21   GrB_Vector_new(&q,GrB_BOOL,n);             // Vector<bool> q(n)
22   GrB_assign(&q,false);
23   GrB_assign(&q,true,s);                     // q[s] = true, false everywhere else
24
25   GrB_Space Boolean;                         // Boolean space <bool,bool,bool,||,&&,false,true>
26   GrB_Space_new(&Boolean,GrB_BOOL,GrB_BOOL,GrB_BOOL,GrB_LOR,GrB_LAND,false,true);
27
28   GrB_Descriptor desc;                       // Descriptor for vxm
29   GrB_Descriptor_new(&desc);
30   GrB_Descriptor_add(desc,GrB_ARG1,GrB_NOP); // no operation on the vector
31   GrB_Descriptor_add(desc,GrB_ARG2,GrB_NOP); // no operation on the matrix
32   GrB_Descriptor_add(desc,GrB_MASK,GrB_LNOT);// invert the mask
33
34   /*
35    * BFS traversal and label the vertices.
36    */
37   int32_t d = 1;                             // d = level in BFS traversal
38   bool succ = false;                         // succ == true when some successor found
39   do {
40     GrB_assign(v,d,q);                       // v[q] = d
41     GrB_vxm(&q,Boolean,q,A,*v,desc);         // q[!v] = q ||.&& A ; finds all the unvisited
42                                              // successors from current q
43     GrB_reduce(&succ,q,GrB_LOR);             // succ = ||(q)
44     d++;                                     // next level
45   } while (succ);                            // if there is no successor in q, we are done.
46
47   GrB_free(q);                               // q vector no longer needed
48   GrB_free(Boolean);                         // Boolean semiring no longer needed
49   GrB_free(desc);                            // descriptor no longer needed
50
51   return GrB_SUCCESS;
52 }
```

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)

Thank You!



http://graphblas.org