Sparse Matrices for High-Performance Graph Analytics

John R. Gilbert
University of California, Santa Barbara

Chesapeake Large-Scale Analytics Conference
October 17, 2013

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

Thanks …

Aydin Buluc (LBL), Kevin Deweese (UCSB), Erika Duriakova (Dublin), Armando Fox (UCB), Shoaib Kamil (MIT), Jeremy Kepner (MIT), Adam Lugowski (UCSB), Tim Mattson (Intel), Lenny Oliker (LBL), Carey Priebe (JHU), Steve Reinhardt (YarcData), Lijie Ren (Google), Eric Robinson (Lincoln), Viral Shah (UIDAI), Veronika Strnadova (UCSB), Yun Teng (UCSB), Joshua Vogelstein (Duke), Drew Waranis (UCSB), Sam Williams (LBL)

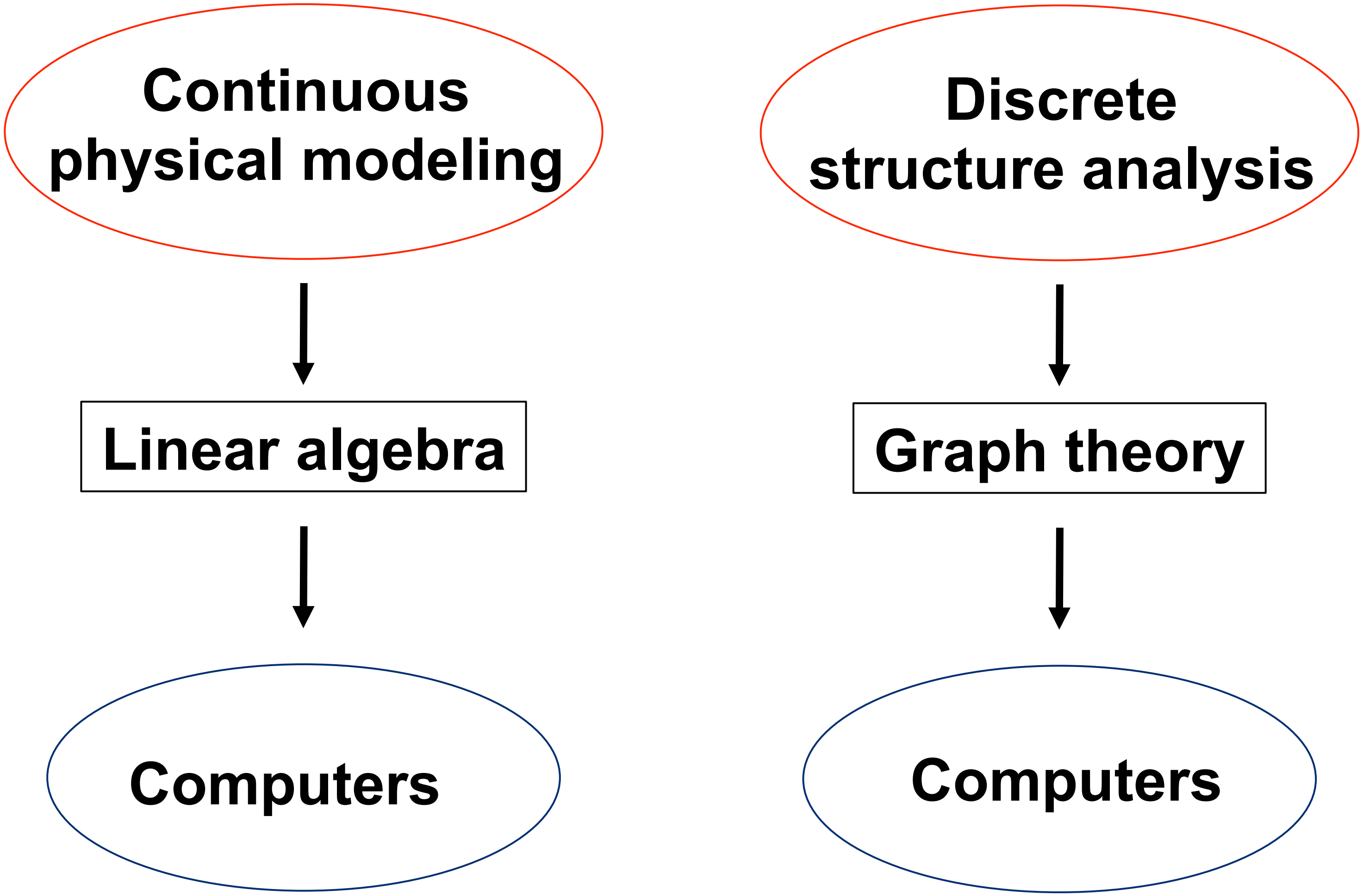
Outline

- Motivation
- Sparse matrices for graph algorithms
- CombBLAS: sparse arrays and graphs on parallel machines
- KDT: attributed semantic graphs in a high-level language
- Standards for graph algorithm primitives

A few biological graph analysis problems

- Connective abnormalities in schizophrenia [van den Heuvel et al.]
  - Problem: understand disease from anatomical brain imaging
  - Tools: betweenness centrality, shortest path length
  - Results: global statistics on connection graph correlate w/ diagnosis
- Genomics for biofuels [Strnadova et al.]
  - Problem: scale to millions of markers times thousands of individuals
  - Tools: min spanning tree, customized clustering
  - Results: using much more data leads to much better genomic maps
- Alignment and matching of brain scans [Vogelstein et al.]
  - Problem: match corresponding functional regions across individuals
  - Tools: graph partitioning, clustering, and more. . .
  - Results: in progress

The middleware of scientific computing

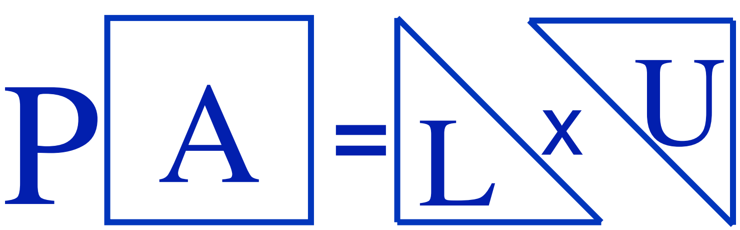


Top 500 List (June 2013)

Rank	Site	System	Cores	Rmax (TFlop/s)	Rpeak (TFlop/s)	Power (kW)
1	National University of Defense Technology 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	54,902.4	17,808
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	27,112.5	8,209
3	DOE/NNSA/LLNL United States	**Sequoia** - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM	1,572,864	17,173.2	20,132.7	7,890
4	RIKEN Advanced Institute for Computational Science (AICS) Japan	K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu	705,024	10,510.0	11,280.4	12,659.9
5	DOE/SC/Argonne National Laboratory United States	**Mira** - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM	786,432	8,586.6	10,066.3	3,945
6	Texas Advanced Computing Center/Univ. of Texas United States	**Stampede** - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell	462,462	5,168.1	8,520.1	4,510
7	Forschungszentrum Juelich (FZJ) Germany	**JUQUEEN** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	458,752	5,008.9	5,872.0	2,301
8	DOE/NNSA/LLNL United States	**Vulcan** - BlueGene/Q, Power BQC 16C 1.600GHz, Custom Interconnect IBM	393,216	4,293.3	5,033.2	1,972
9	Leibniz Rechenzentrum Germany	**SuperMUC** - iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR IBM	147,456	2,897.0	3,185.1	3,422.7
10	National Supercomputing Center in Tianjin China	**Tianhe-1A** - NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050 NUDT	186,368	2,566.0	4,701.0	4,040
11	Total Exploration Production France	**Pangea** - SGI ICE X, Xeon E5-2670 8C 2.600GHz, Infiniband FDR SGI	110,400	2,098.1	2,296.3	2,118

Top500 Benchmark:

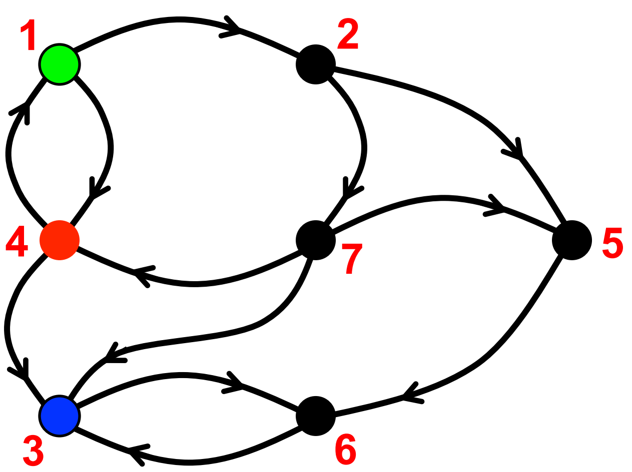
Solve a large system of linear equations by Gaussian elimination



Graph 500 List (June 2013)

Graph500 Benchmark:

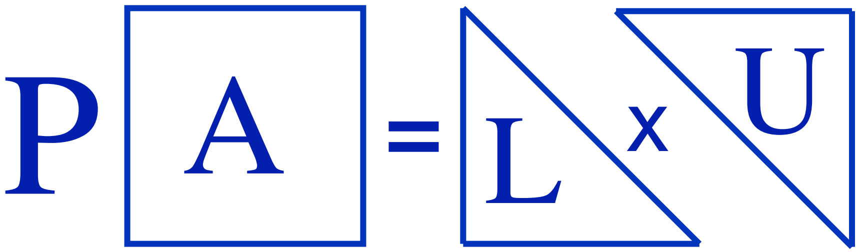
Breadth-first search in a large power-law graph



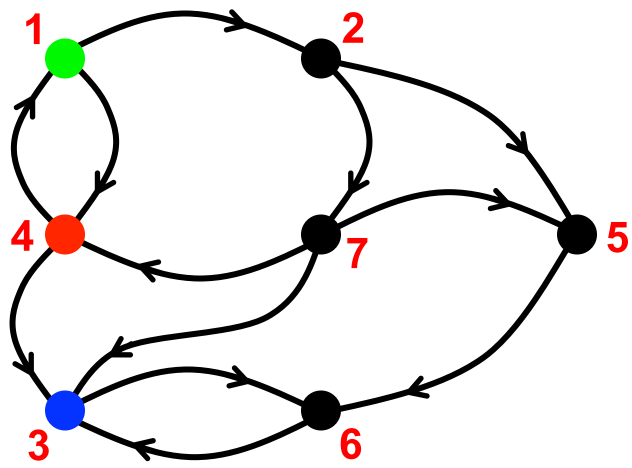
No.	Rank	Machine	Installation Site	Number of nodes	Number of cores	Problem scale	GTEPS
1	1	DOE/NNSA/LLNL Sequoia (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Lawrence Livermore National Laboratory	65536	1048576	40	15363
2	2	DOE/SC/Argonne National Laboratory Mira (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Argonne National Laboratory	49152	786432	40	14328
3	3	JUQUEEN (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Forschungszentrum Juelich (FZJ)	16384	262144	38	5848
4	4	K computer (Fujitsu - Custom supercomputer)	RIKEN Advanced Institute for Computational Science (AICS)	65536	524288	40	5524.12
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
9	7	DIRAC (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	University of Edinburgh	4096	65536	36	1427

Floating-Point vs. Graphs, June 2013

33.8 Petaflops



15.3 Terateps

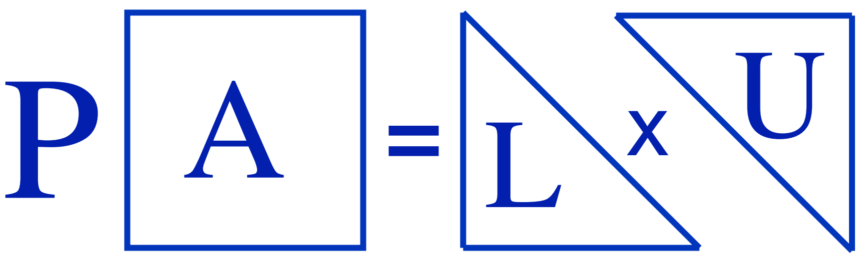


33.8 Peta / 15.3 Tera is about 2200.

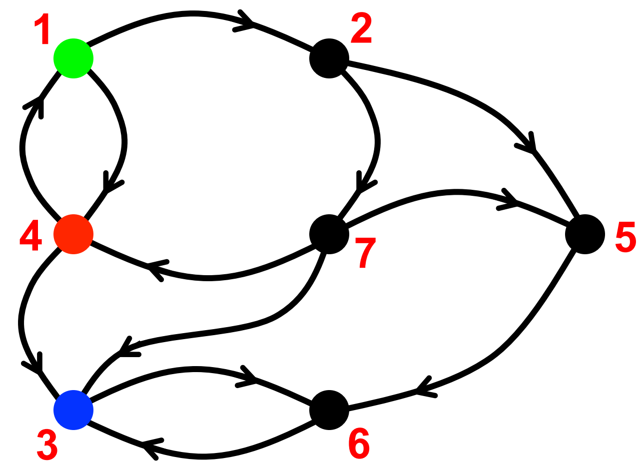
UCSB

Floating-Point vs. Graphs, June 2013

33.8 Petaflops



15.3 Terateps



Jun 2013: **33.8 Peta / 15.3 Tera ~ 2,200**

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

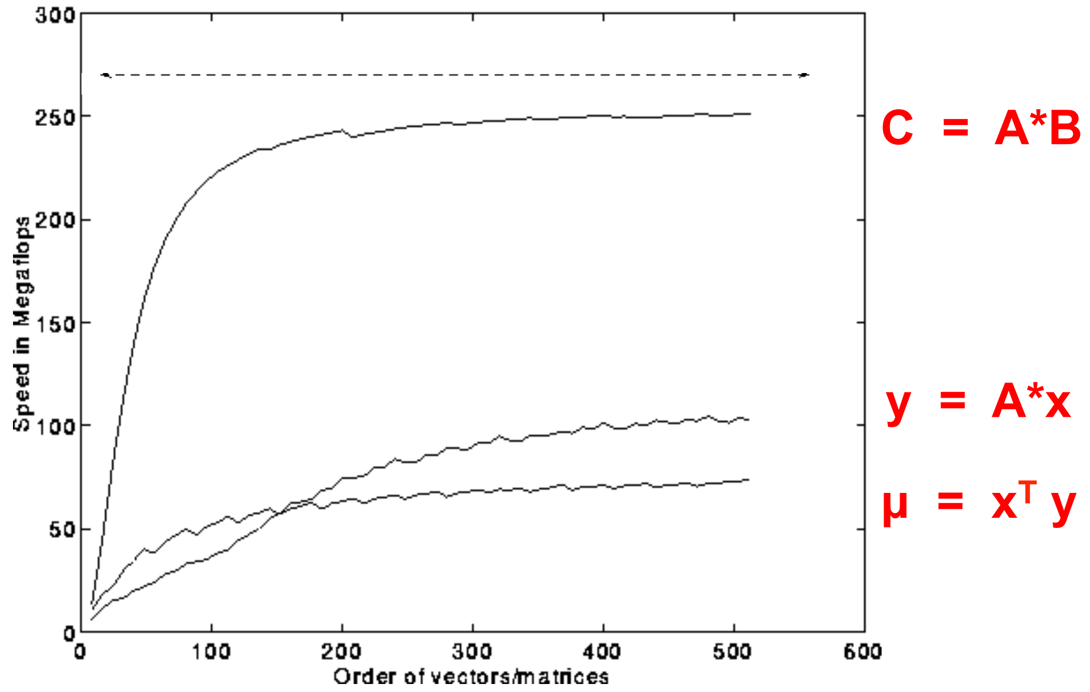
UCSB

The challenge of the software stack

- By analogy to numerical scientific computing. . .

- What should the combinatorial BLAS look like?

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

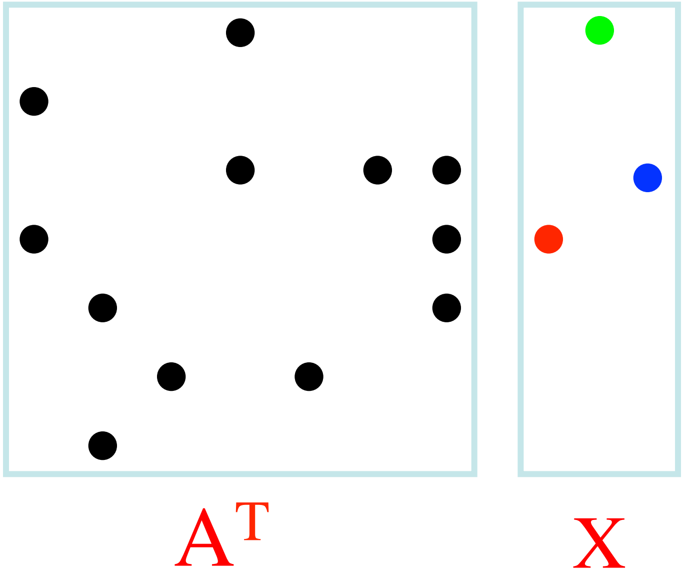


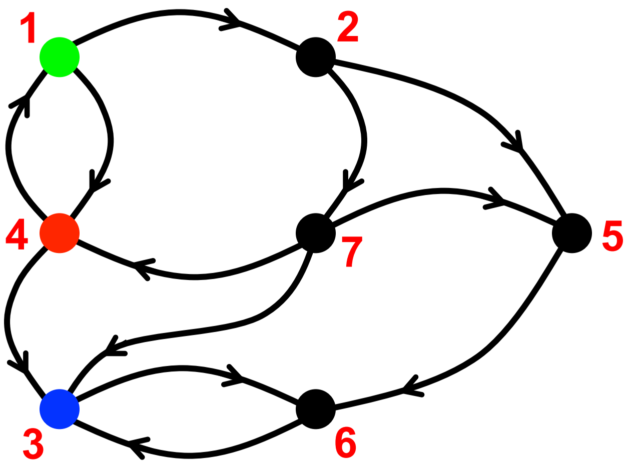
Outline

- Motivation
- Sparse matrices for graph algorithms
- CombBLAS: sparse arrays and graphs on parallel machines
- KDT: attributed semantic graphs in a high-level language
- Standards for graph algorithm primitives

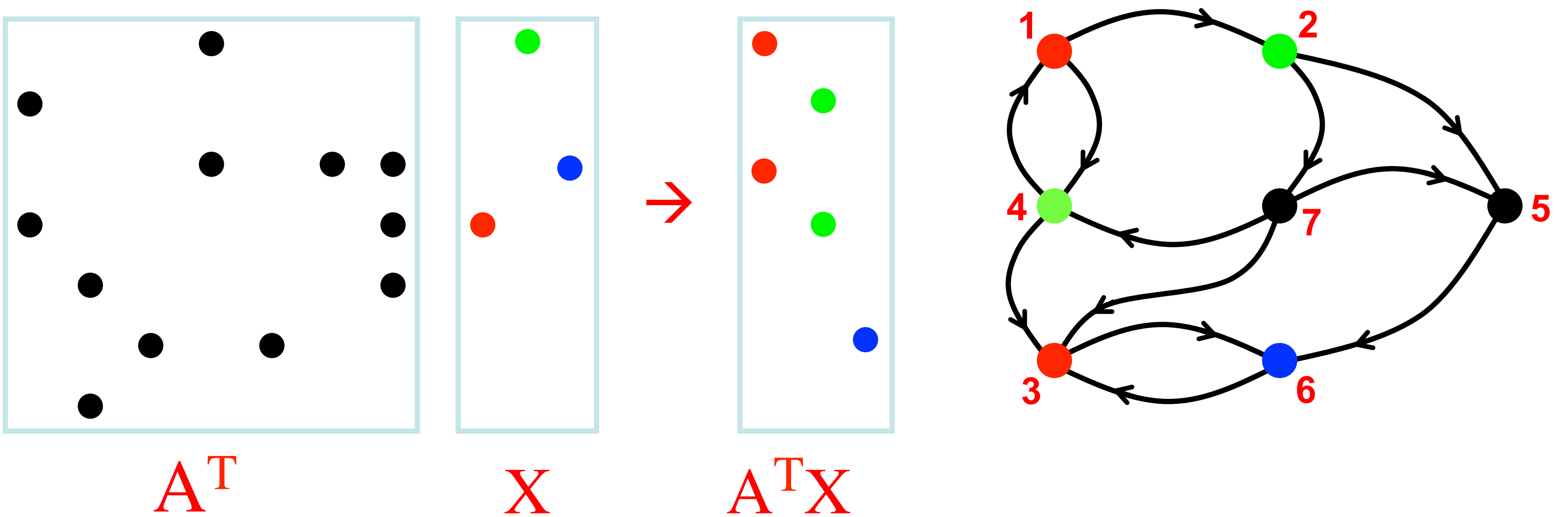
UCSB

Multiple-source breadth-first search





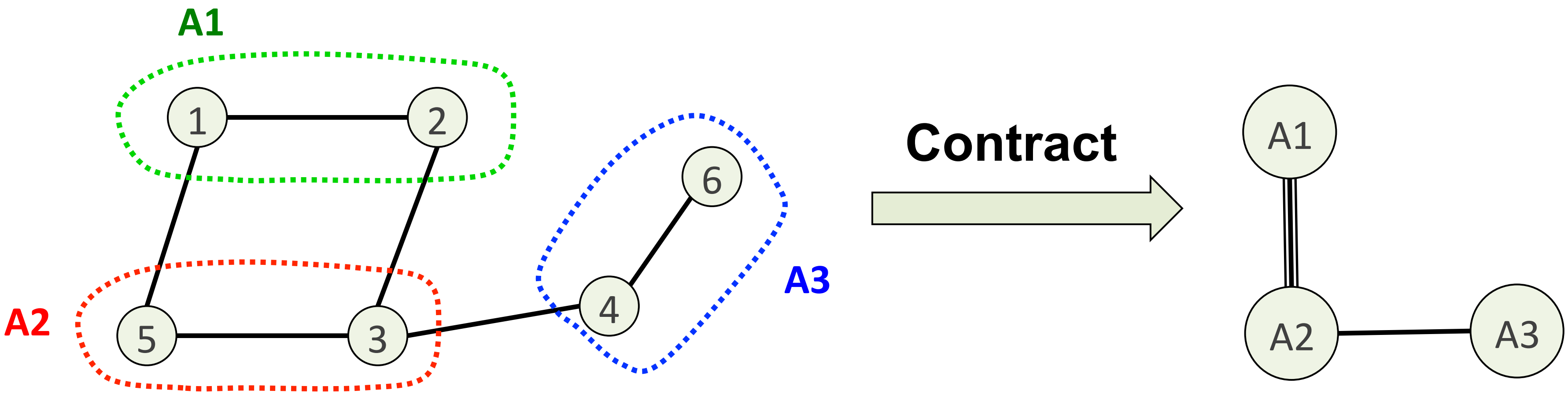
Multiple-source breadth-first search

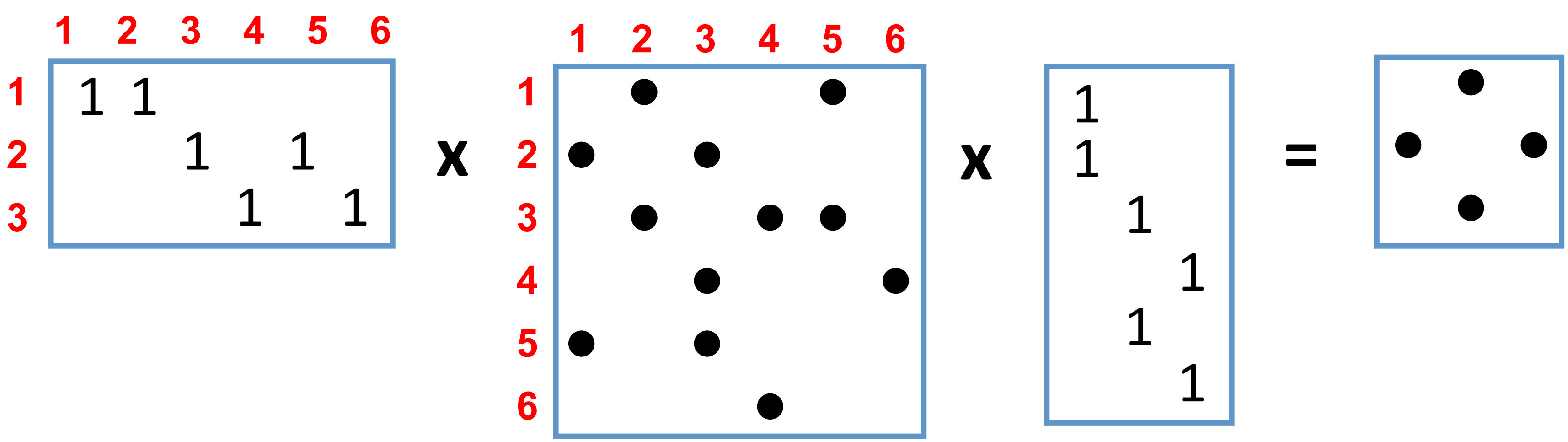


- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

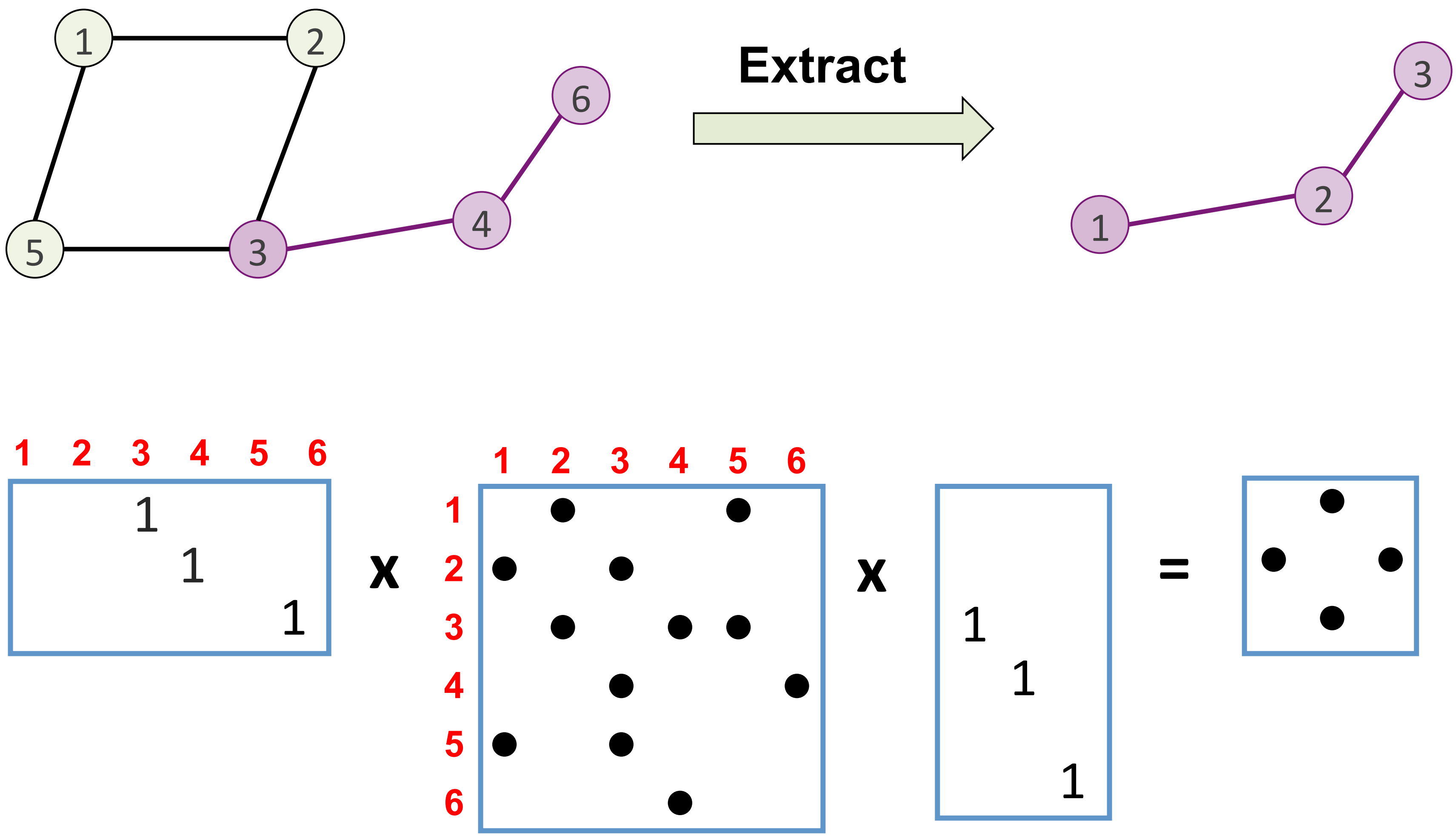
UCSB

Graph contraction via sparse triple product





Subgraph extraction via sparse triple product



Betweenness centrality *[Robinson 2008]*

$b = \text{BetweennessCentrality}(G = A : \mathbb{B}^{N_V \times N_V})$

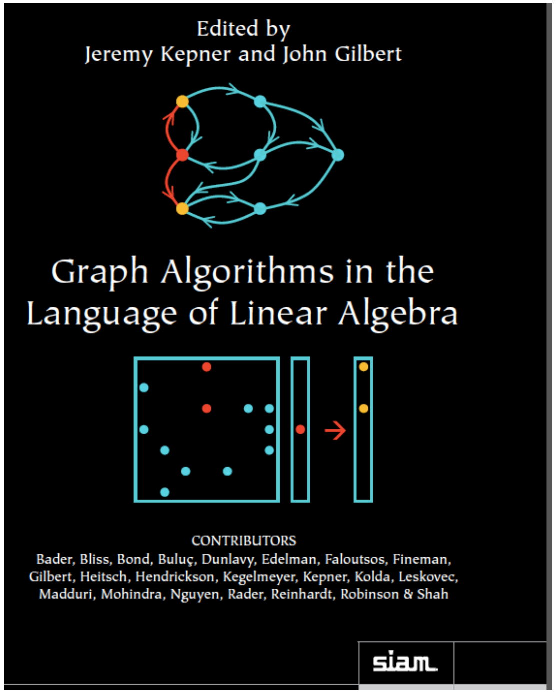
$\mathbf{b} = 0$
for $1 \le r \le N_V$
  do
    $d = 0$
    $\mathbf{S} = 0$
    $\mathbf{p} = 0,\ p_r = 1$
    $\mathbf{f} = \mathbf{a}_{r,:}$
    while $\mathbf{f} \neq 0$
      do
        $d = d + 1$
        $\mathbf{p} = \mathbf{p} + \mathbf{f}$
        $\mathbf{s}_{d,:} = \mathbf{f}$
        $\mathbf{f} = \mathbf{f}\mathbf{A} \times \neg\mathbf{p}$
    while $d \ge 2$
      do
        $\mathbf{w} = \mathbf{s}_{d,:} \times (1 + \mathbf{u}) \div \mathbf{p}$
        $\mathbf{w} = \mathbf{A}\mathbf{w}$
        $\mathbf{w} = \mathbf{w} \times \mathbf{s}_{d-1,:} \times \mathbf{p}$
        $\mathbf{u} = \mathbf{u} + \mathbf{w}$
        $d = d - 1$
    $\mathbf{b} = \mathbf{b} + \mathbf{u}$

Variables:			**Storage:**
A :	sparse adjacency matrix	$B^{S(N\times N)}$	O(M+N)
f :	sparse fringe vector	$Z^{S(N)}$	O(N)
p :	shortest path vector	Z^{N}	O(N)
S :	sparse depth matrix	$B^{S(N\times N)}$	O(N)
u :	centrality update vector	R^{N}	O(N)

Storage: O(M+N)
Time: $O(MN + N^2)$

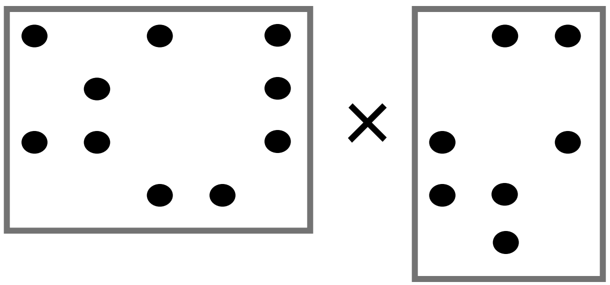
Graph algorithms in the language of linear algebra

- 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, using CombBLAS

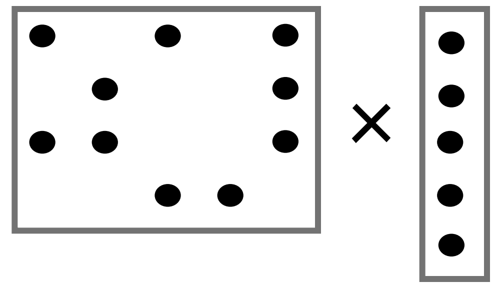


Sparse array-based primitives

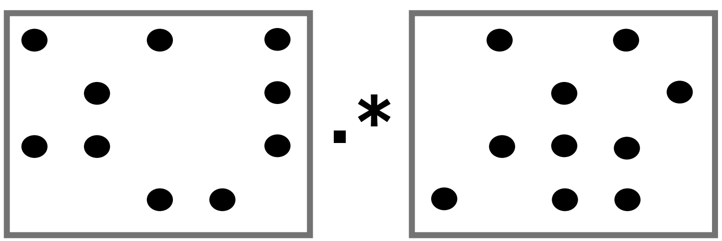
Sparse matrix-matrix multiplication (SpGEMM)



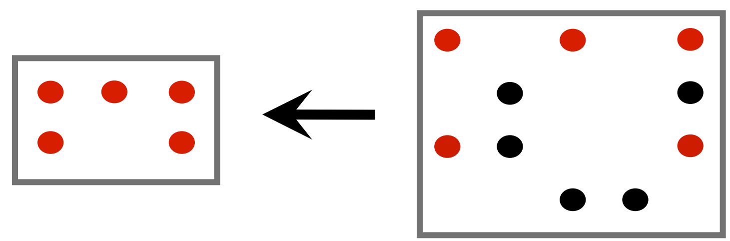
Sparse matrix-dense vector multiplication



Element-wise operations



Sparse matrix indexing



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

UCSB

The case for sparse matrix graph primitives

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 sparse matrix graph primitives

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

Matrices over semirings

- E.g. matrix multiplication **C = AB** (or matrix/vector):

$$C_{i,j} = A_{i,1} \times B_{1,j} + A_{i,2} \times B_{2,j} + \cdots + A_{i,n} \times B_{n,j}$$

- Replace scalar operations **×** and **+** by

  $\otimes$: associative, distributes over $\oplus$

  $\oplus$: associative, commutative

- Then $C_{i,j} = A_{i,1} \otimes B_{1,j} \oplus A_{i,2} \otimes B_{2,j} \oplus \cdots \oplus A_{i,n} \otimes B_{n,j}$

- Examples: **×.+** ; and.or ; +.min ; . . .

- Same data reference pattern and control flow

UCSB

Examples of semirings in graph algorithms

(R, +, x) Real Field	Standard numerical linear algebra
({0,1}, |, &) Boolean Semiring	Graph traversal
(R U {∞}, min, +) Tropical Semiring	Shortest paths
(R U {∞}, min, x)	Select subgraph, or contract nodes to form quotient graph
(edge/vertex attributes, vertex data aggregation, edge data processing)	Schema for user-specified computation at vertices and edges

UCSB

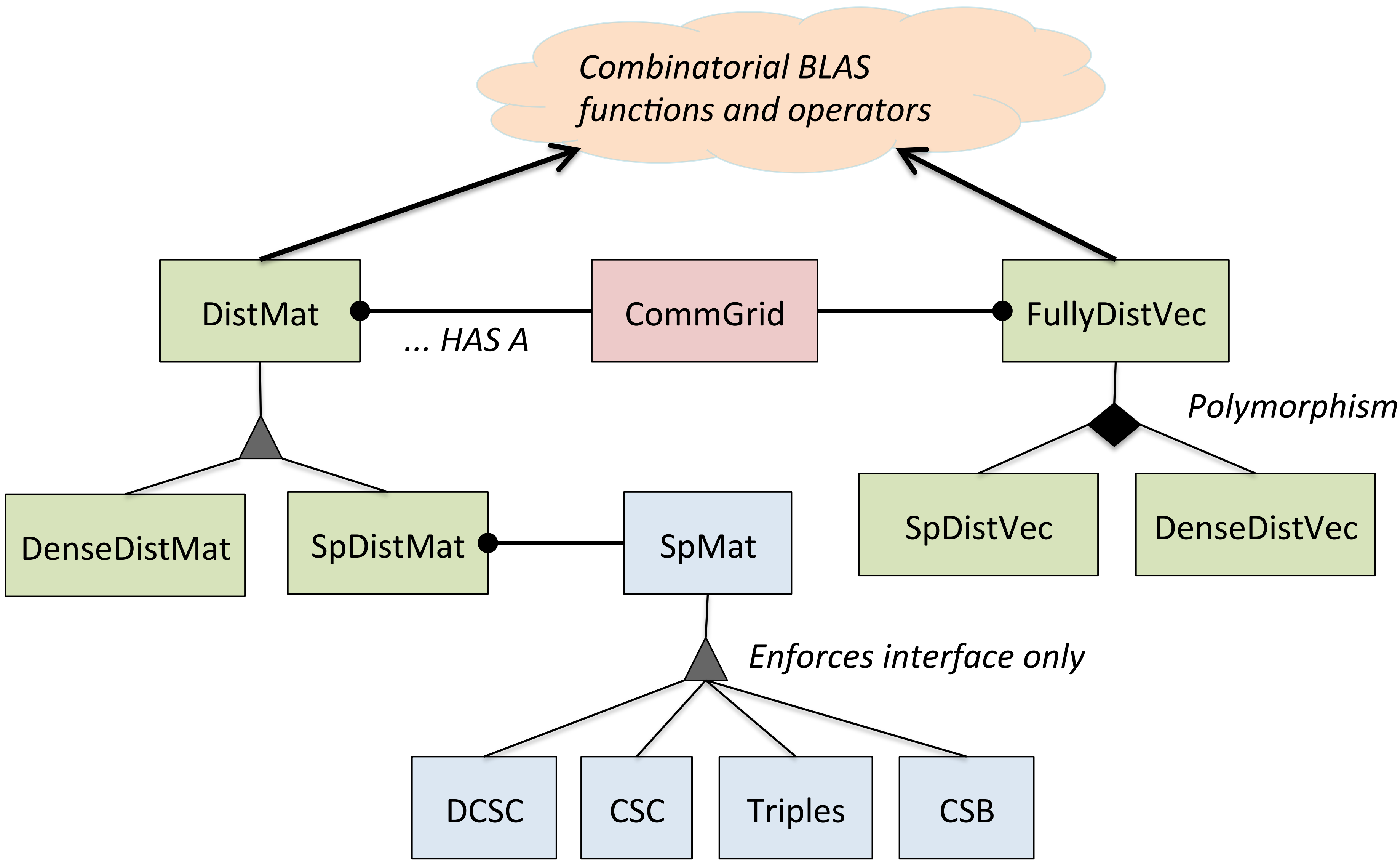
Outline

- Motivation
- Sparse matrices for graph algorithms
- CombBLAS: sparse arrays and graphs on parallel machines
- KDT: attributed semantic graphs in a high-level language
- Standards for graph algorithm primitives

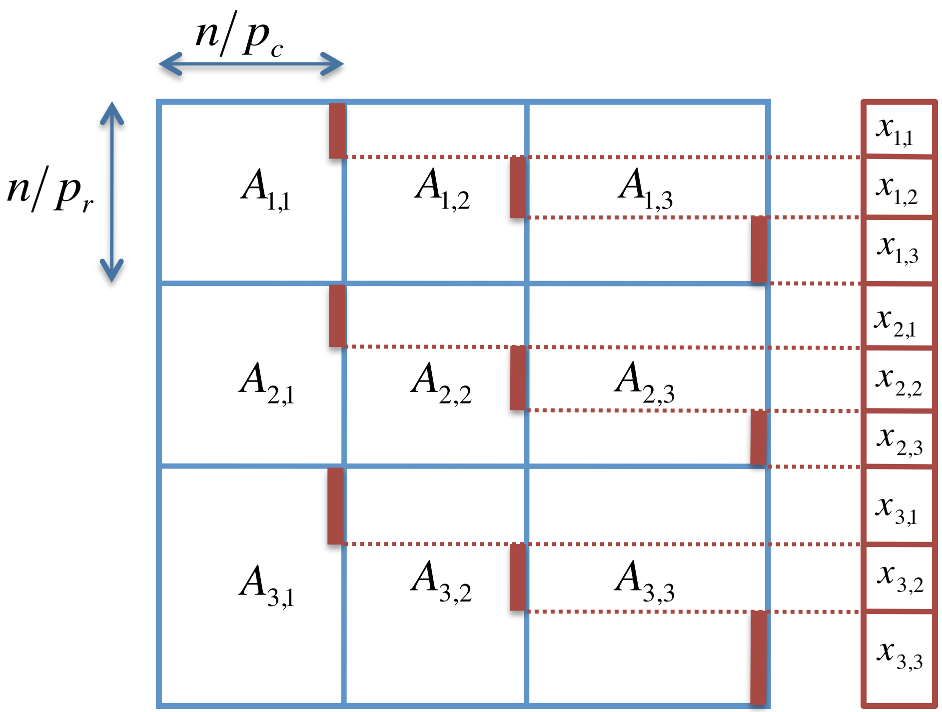
Combinatorial BLAS: Functions

Function	Applies to	Parameters		Returns	Matlab Phrasing
SpGEMM	Sparse Matrix (as friend)	**A**, **B**: trA: trB:	sparse matrices transpose **A** if true transpose **B** if true	Sparse Matrix	$\mathbf{C} = \mathbf{A} * \mathbf{B}$
SpMV	Sparse Matrix (as friend)	**A**: **x**: trA:	sparse matrices sparse or dense vector(s) transpose **A** if true	Sparse or Dense Vector(s)	$\mathbf{y} = \mathbf{A} * \mathbf{x}$
SpEWiseX	Sparse Matrices (as friend)	**A**, **B**: notA: notB:	sparse matrices negate **A** if true negate **B** if true	Sparse Matrix	$\mathbf{C} = \mathbf{A} * \mathbf{B}$
Reduce	Any Matrix (as method)	dim: *binop*:	dimension to reduce reduction operator	Dense Vector	sum(**A**)
SpRef	Sparse Matrix (as method)	**p**: **q**:	row indices vector column indices vector	Sparse Matrix	$\mathbf{B} = \mathbf{A}(\mathbf{p}, \mathbf{q})$
SpAsgn	Sparse Matrix (as method)	**p**: **q**: **B**:	row indices vector column indices vector matrix to assign	none	$\mathbf{A}(\mathbf{p}, \mathbf{q}) = \mathbf{B}$
Scale	Any Matrix (as method)	**rhs**:	any object (except a sparse matrix)	none	Check guiding principles 3 and 4
Scale	Any Vector (as method)	**rhs**:	any vector	none	none
Apply	Any Object (as method)	*unop*:	unary operator (applied to non-zeros)	None	none

Combinatorial BLAS: Distributed-memory reference implementation



2D layout for sparse matrices & vectors



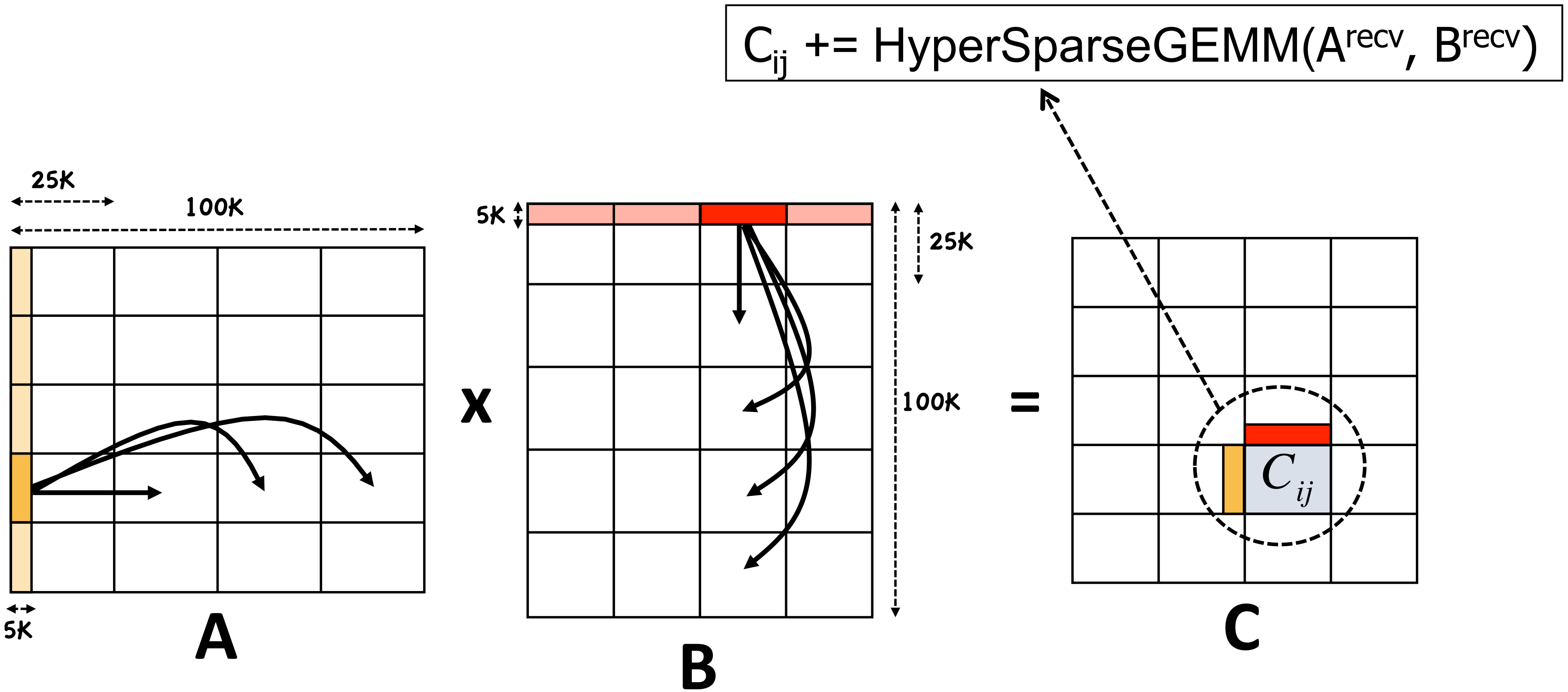
Matrix/vector distributions, interleaved on each other.

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

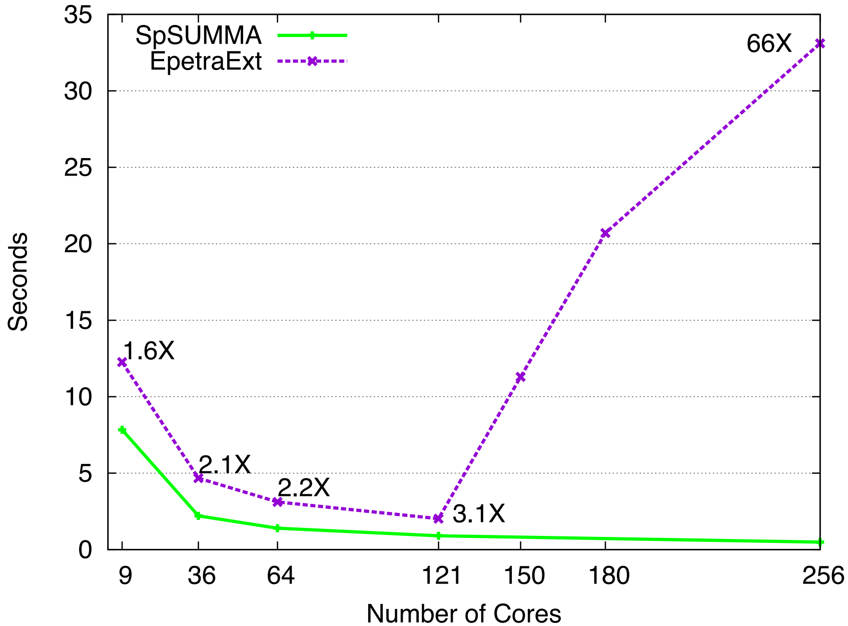
Parallel sparse matrix-matrix multiplication algorithm



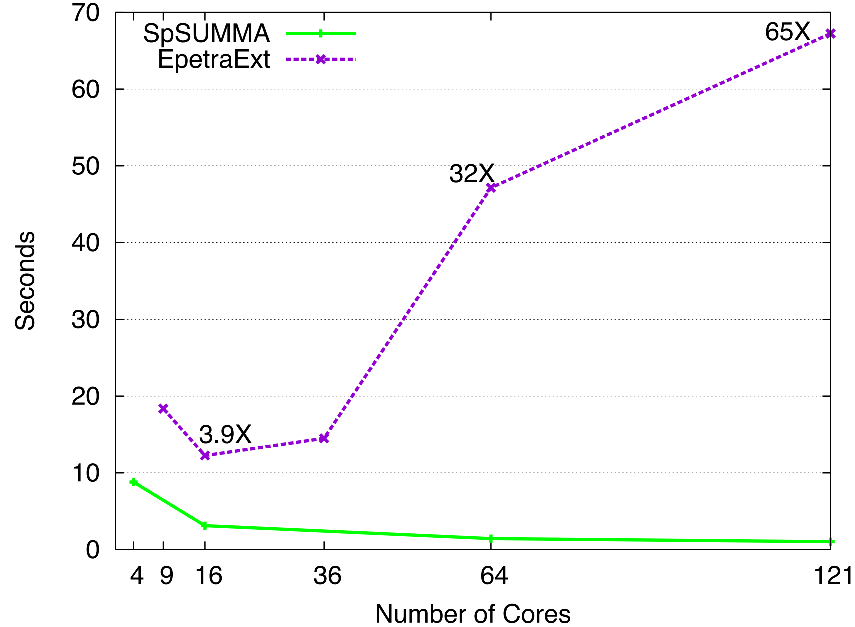
2D algorithm: Sparse SUMMA (based on dense SUMMA)

General implementation that handles rectangular matrices

1D vs. 2D scaling for sparse matrix-matrix multiplication



(a) R-MAT × R-MAT product (scale 21).

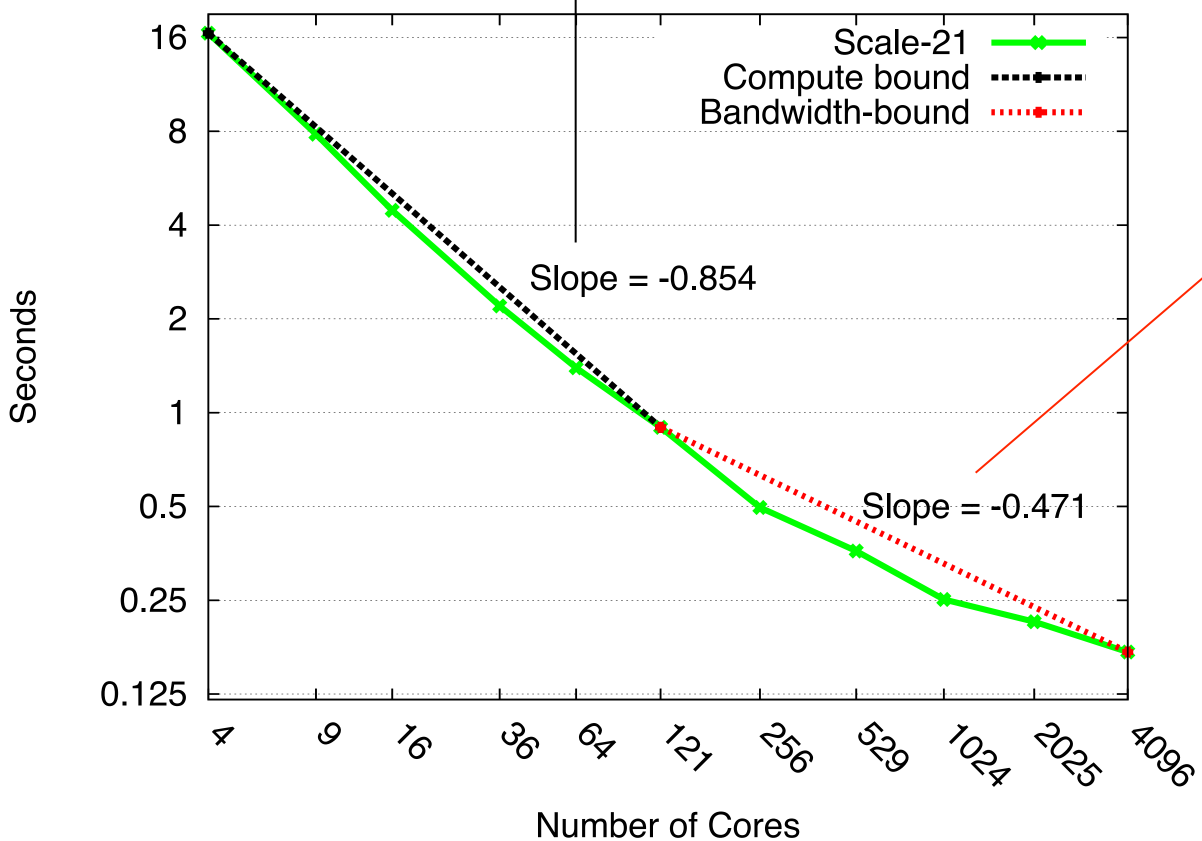


(b) Multiplication of an R-MAT matrix of scale 23 with the restriction operator of order 8.

- 1-D data layout
- 2-D data layout (Combinatorial BLAS)

Scaling to more processors…

Almost linear scaling until bandwidth costs starts to dominate



Scaling proportional to √p afterwards

$$T_{comp} = O(n)$$

$$T_{comm} = O(n\sqrt{p})$$

Combinatorial BLAS users (Sep 2013)

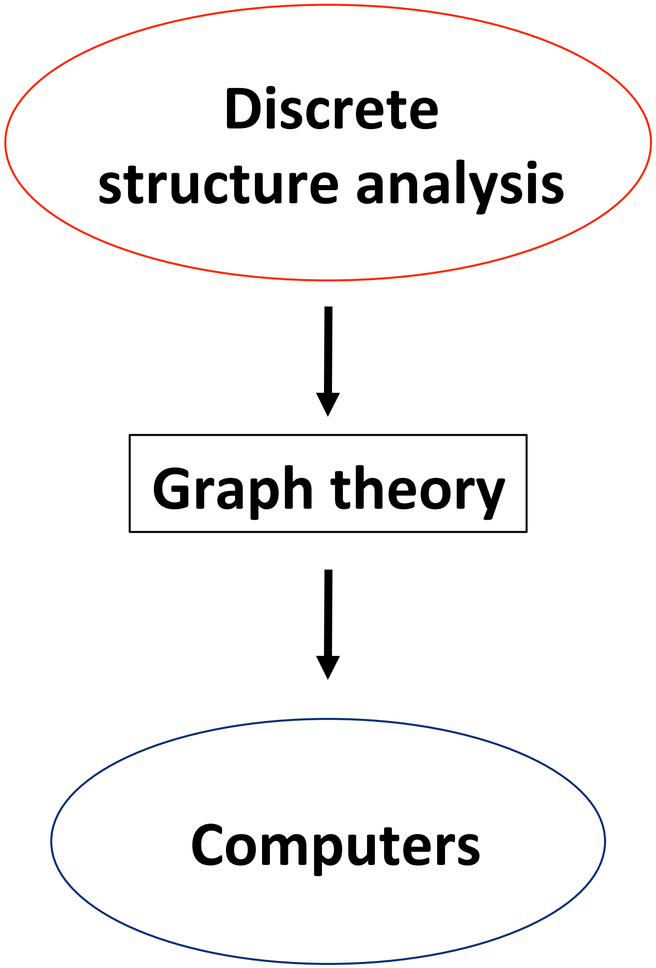
- IBM (T.J. Watson, Zurich, & Tokyo)
- Microsoft
- Intel
- Cray
- Stanford
- UC Berkeley
- Carnegie-Mellon
- Georgia Tech
- Ohio State
- Columbia
- U Minnesota
- King Fahd U
- Tokyo Inst of Technology
- Chinese Academy of Sciences
- U Ghent (Belgium)
- Bilkent U (Turkey)
- U Canterbury (New Zealand)
- Purdue
- Indiana U
- Mississippi State
- UC Merced

UCSB

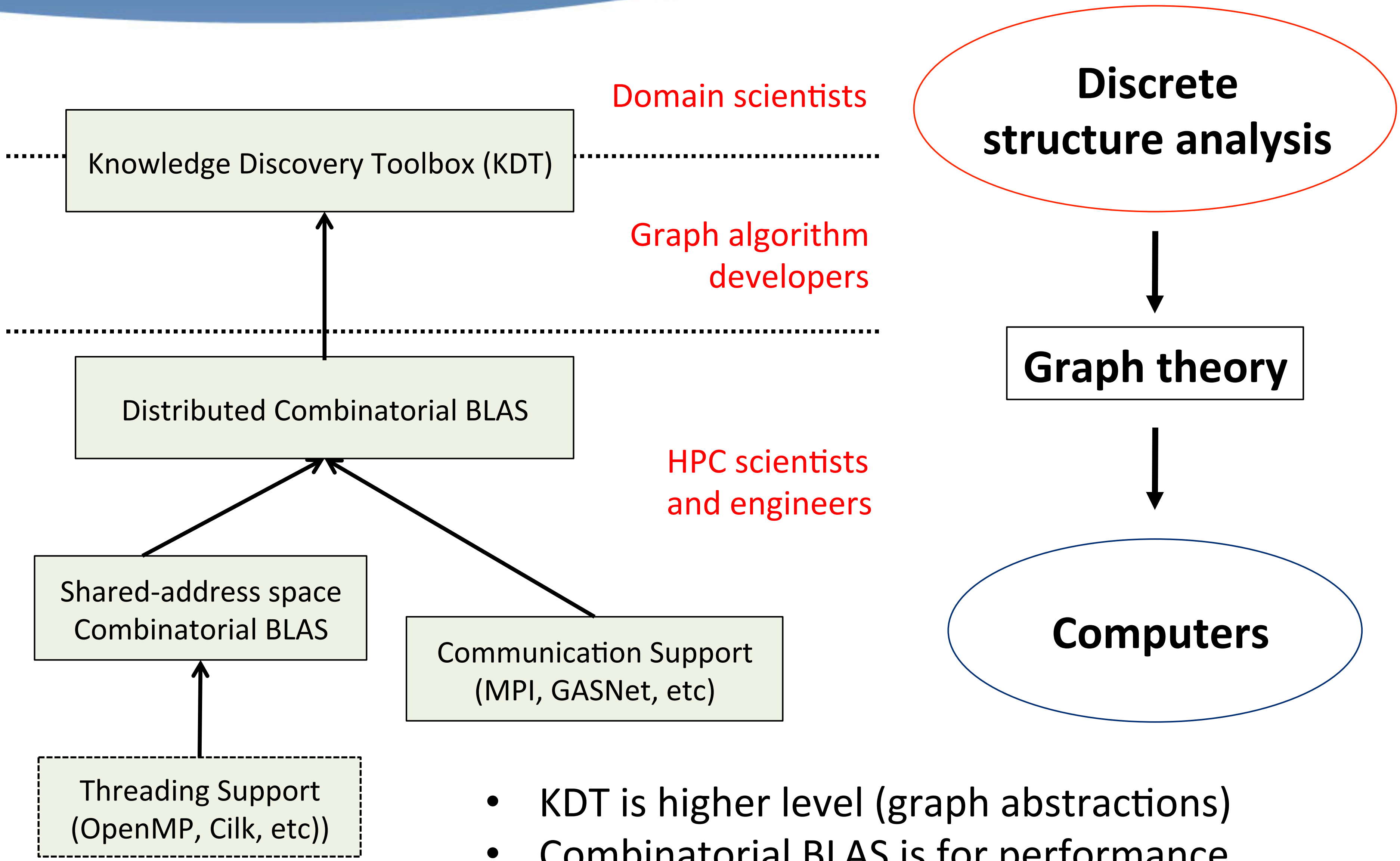
Outline

- Motivation
- Sparse matrices for graph algorithms
- CombBLAS: sparse arrays and graphs on parallel machines
- KDT: attributed semantic graphs in a high-level language
- Standards for graph algorithm primitives

Parallel graph analysis software



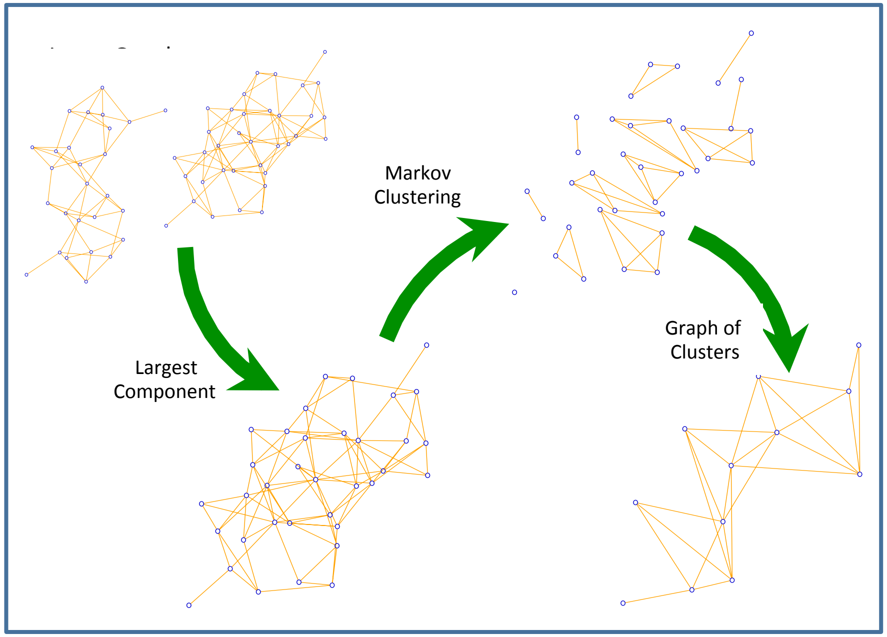
Parallel graph analysis software



- KDT is higher level (graph abstractions)
- Combinatorial BLAS is for performance

Domain expert vs. graph expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank

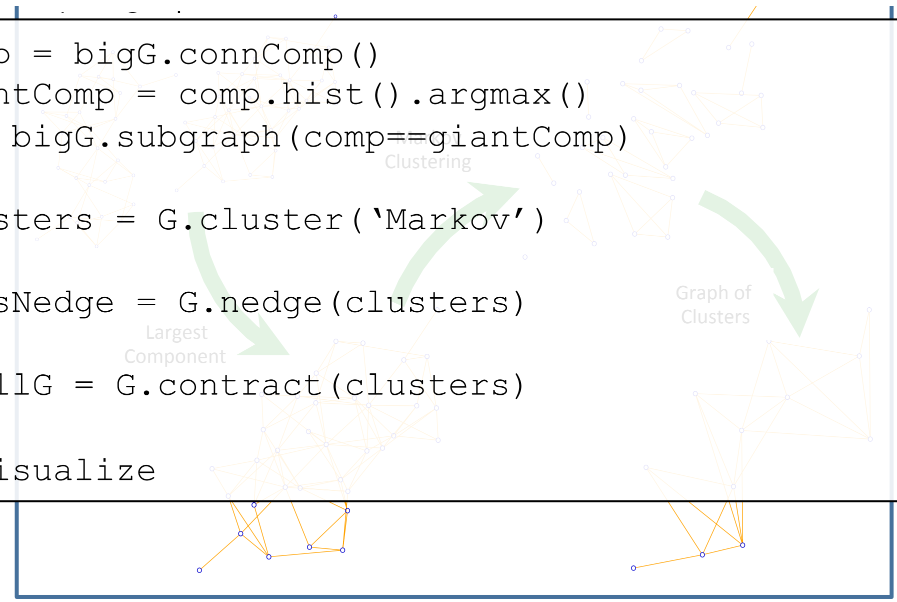


- Hypergraphs and sparse matrices
- Graph primitives (*e.g.*, `bfsTree()`)
- SpMV / SpGEMM on semirings

Domain expert vs. graph expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank

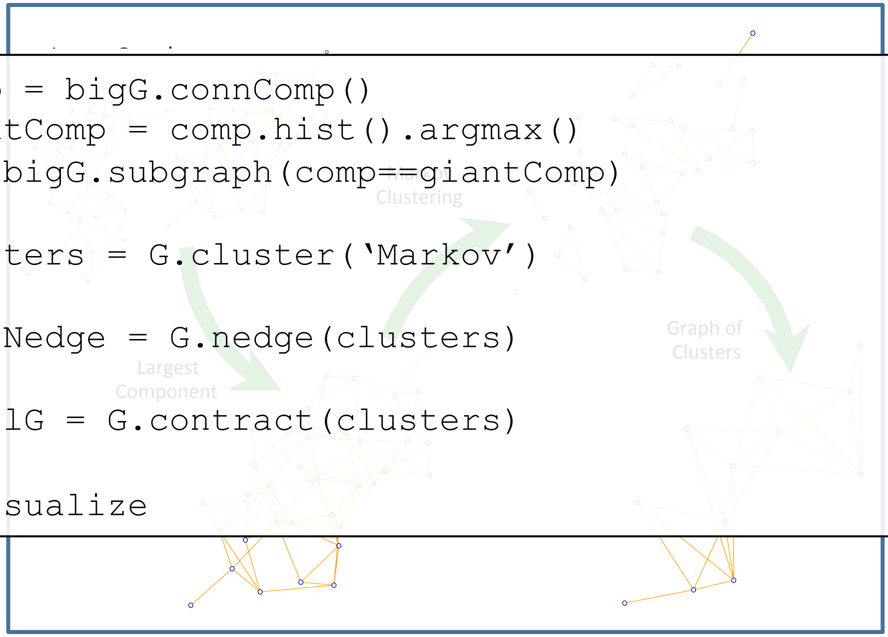
```
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)

clusters = G.cluster('Markov')

clusNedge = G.nedge(clusters)

smallG = G.contract(clusters)

# visualize
```



- Hypergraphs and sparse matrices
- Graph primitives (*e.g.*, `bfsTree()`)
- SpMV / SpGEMM on semirings

Domain expert vs. graph expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority: betweenness centrality, PageRank




```
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)

clusters = G.cluster('Markov')

clusNedge = G.nedge(clusters)

smallG = G.contract(clusters)

# visualize
```

- Hypergraphs and sparse matrices
- Graph primitives (*e.g.*, `bfsTree()`)
- SpMV / SpGEMM on semirings

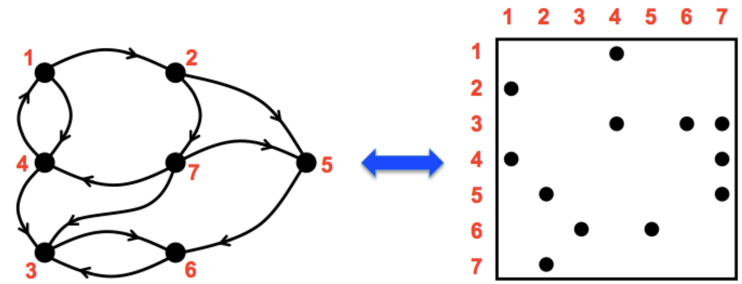
```
            [...]
L = G.toSpParMat()
d = L.sum(kdt.SpParMat.Column)
L = -L
L.setDiag(d)
M = kdt.SpParMat.eye(G.nvert()) - mu*L
pos = kdt.ParVec.rand(G.nvert())
for i in range(nsteps):
    pos = M.SpMV(pos)
```

Knowledge

Discovery

Toolbox

http://kdt.sourceforge.net/



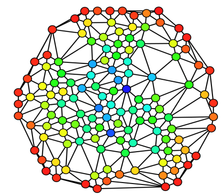
A general graph library with operations based on linear algebraic primitives

- 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)
- V0.3 release April 2013

A few KDT applications

Betweenness Centrality



$$C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

image courtesy Claudio Rocchini

Betweenness Centrality says that a vertex is important if it appears on many shortest paths between other vertices. An exact computation requires a BFS for every vertex. A good approximation can be achieved by sampling starting vertices.

Markov Clustering

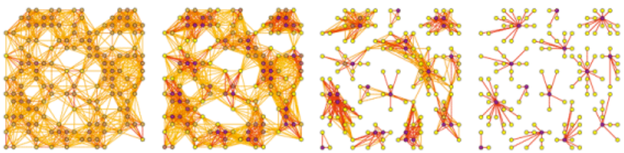


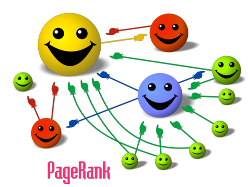
image courtesy Stijn van Dongen

Markov Clustering (MCL) finds clusters by postulating that a random walk that visits a dense cluster will probably visit many of its vertices before leaving.

We use a Markov chain for the random walk. This process is reinforced by adding an inflation step that uses the Hadamard product and rescaling.

PageRank

PageRank says a vertex is important if other important vertices link to it.

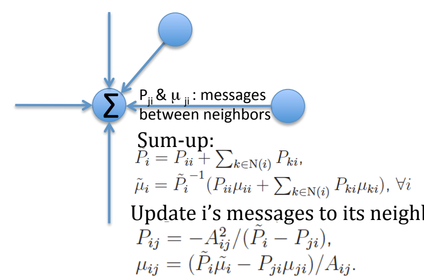


courtesy Felipe Micaroni Lalli

Each vertex (webpage) votes by splitting its PageRank score evenly among its out edges (links). This broadcast (an SpMV) is followed by a normalization step (ColWise). Repeat until convergence.

PageRank is the stationary distribution of a Markov Chain that simulates a "random surfer".

Belief Propagation



Gaussian belief propagation (GaBP) is an iterative algorithm for solving the linear system of equations Ax = b, where A is symmetric positive definite.
GaBP assumes each variable follows a normal distribution. It iteratively calculates the precision P and mean value μ of each variable; the converged mean-value vector approximates the actual solution.

Attributed semantic graphs and filters

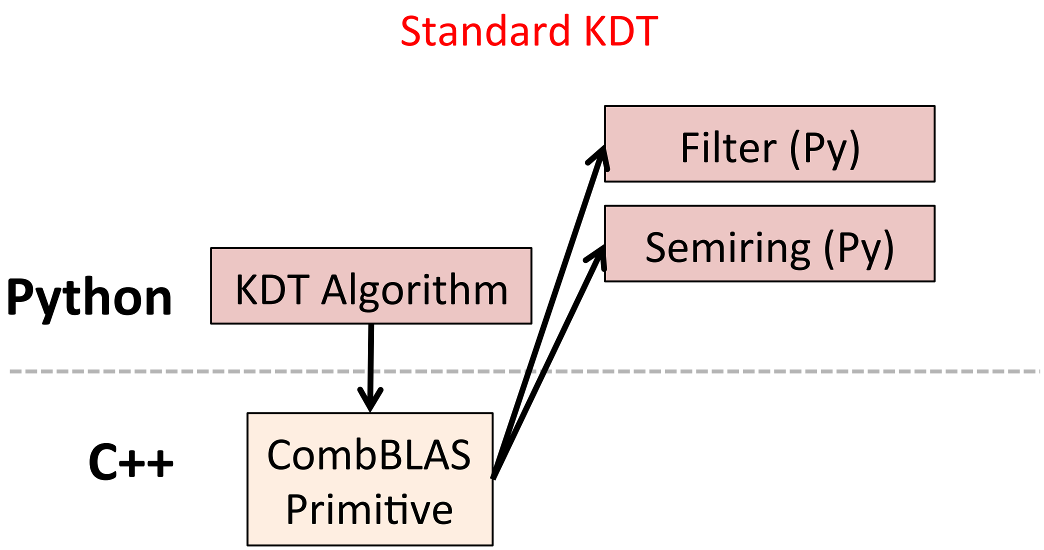
Example:

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

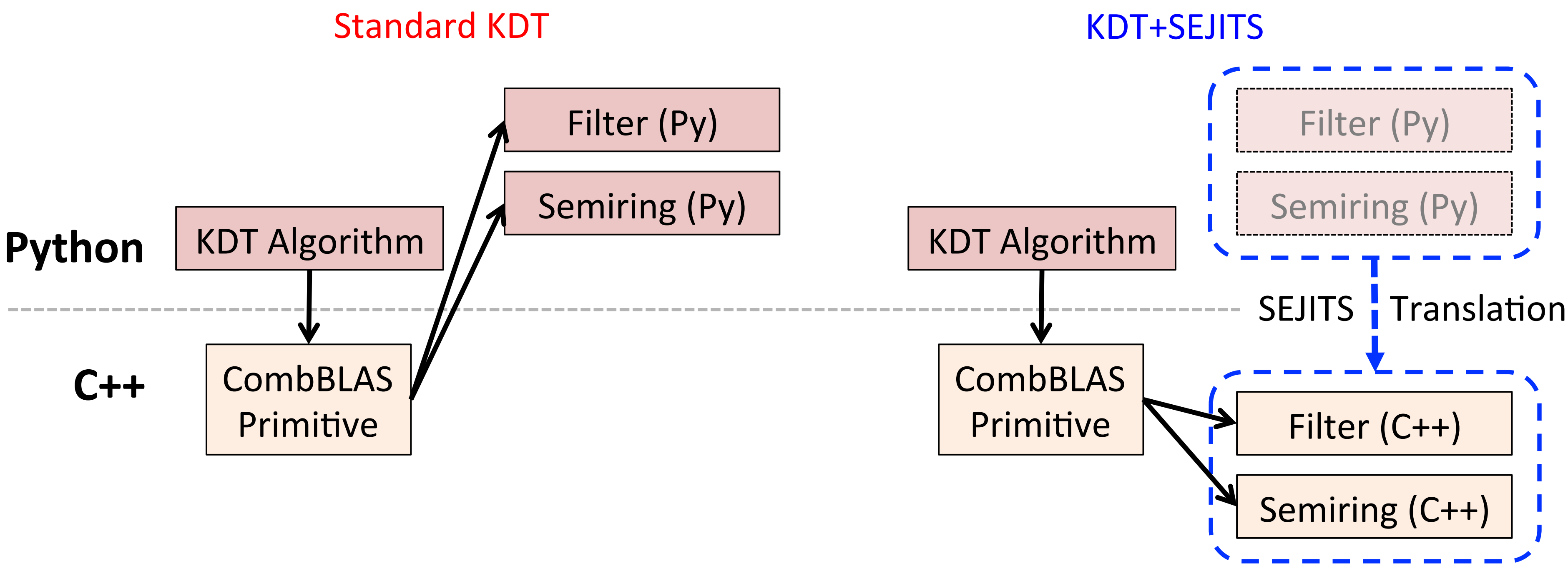
- Calculate centrality just for emails among engineers sent between given start and end times

```
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



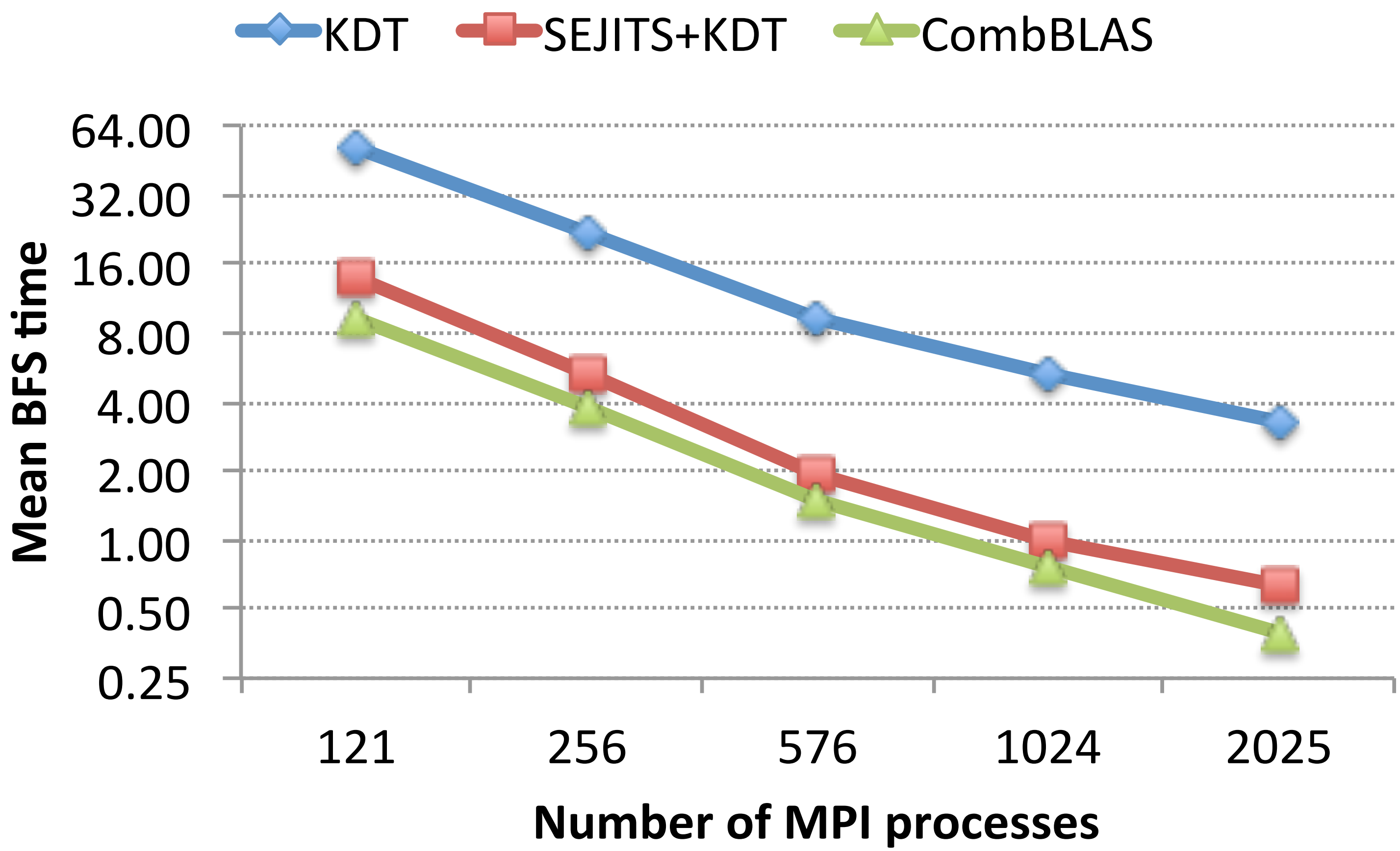
SEJITS for filter/semiring acceleration



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.

Outline

- Motivation
- Sparse matrices for graph algorithms
- CombBLAS: sparse arrays and graphs on parallel machines
- KDT: attributed semantic graphs in a high-level language
- Standards for graph algorithm primitives

History of 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, Hammarling, Hanson, 1990	LAPACK on cache based machines

- Separation of concerns:
  - Experts in mapping algorithms onto hardware tuned BLAS to specific platforms.
  - Experts in linear algebra built software on top of the BLAS to obtain high performance “for free”.
- Today every computer, phone, etc. comes with `/usr/lib/libblas`

UCSB

Can we define and standardize the "Graph BLAS"?

- No, it is not reasonable to define a universal set of graph algorithm building blocks:
  - Huge diversity in matching algorithms to hardware platforms.
  - No consensus on data structures and linguistic primitives.
  - Lots of graph algorithms remain to be discovered.
  - Early standardization can inhibit innovation.
- Yes, it is reasonable to define a common set of graph algorithm 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.

UCSB

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.

Conclusion

- It helps to look at things from two directions.
- Sparse arrays and matrices yield useful primitives and algorithms for high-performance graph computation.
- Graphs in the language of linear algebra are sufficiently mature to support a standard set of BLAS.