Sparse Matrices and Graphs: There and Back Again

John R. Gilbert
University of California, Santa Barbara

Simons Institute Workshop on Parallel and Distributed Algorithms for Inference and Optimization

October 22, 2013

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

Combinatorial Scientific Computing

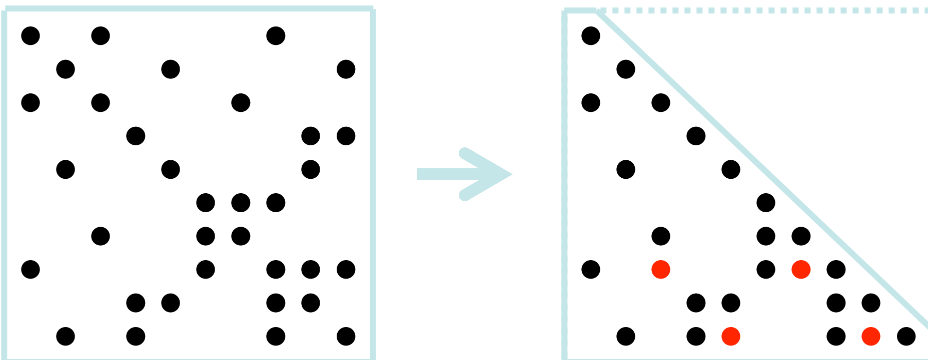
"I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were 'sparse' in the matrix, and that typically the triangular matrices associated with the forward and back solution provided by Gaussian elimination would remain sparse if pivot elements were chosen with care"

- Harry Markowitz, describing the 1950s work on portfolio theory that won the 1990 Nobel Prize for Economics

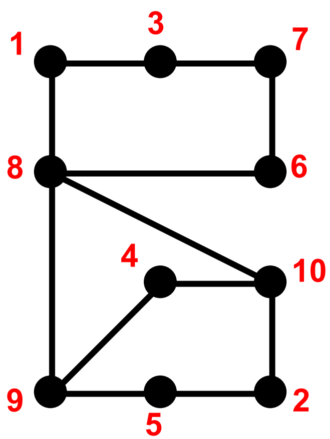
UCSB

Graphs and sparse matrices: Cholesky factorization

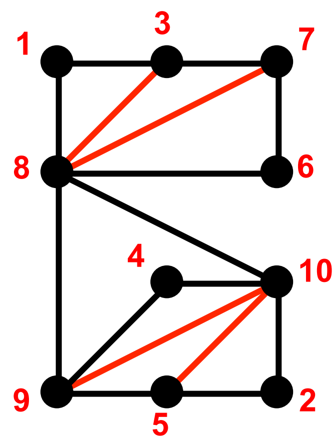
$A = LL^T$



Fill: new nonzeros in factor



G(A)



G(L)
[chordal]

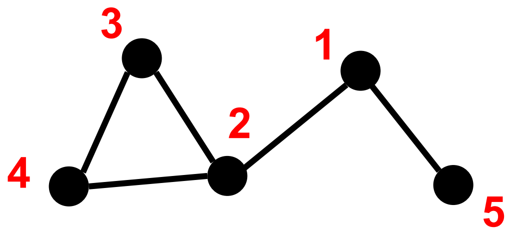
Symmetric Gaussian elimination:

for j = 1 to n
  add edges between j's
  higher-numbered neighbors

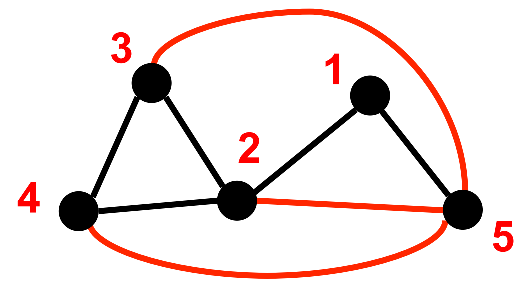
UCSB

Sparse Gaussian elimination and chordal completion

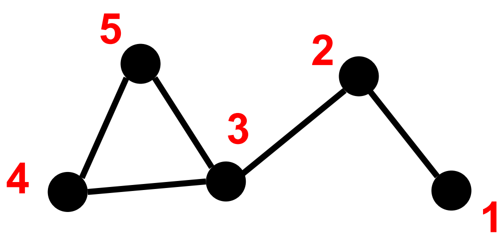
[Parter, Rose]



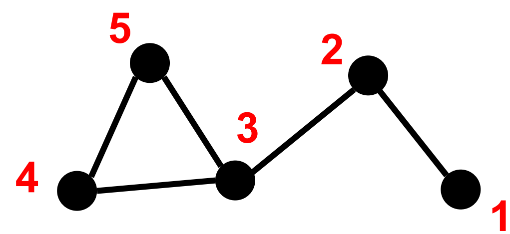
$$Ax = b$$



$$A = L_1 L_1^T$$



$$(PAP^T)\ (Px) = (Pb)$$



$$PAP^T = L_2 L_2^T$$

UCSB

Sparse Gaussian elimination and chordal completion

[Parter, Rose]

Repeat:

    Choose a vertex v and mark it;

    Add edges between unmarked neighbors of v;

Until: Every vertex is marked

Goal: End up with as few edges as possible.

Or, add fewest possible edges to make the graph chordal.

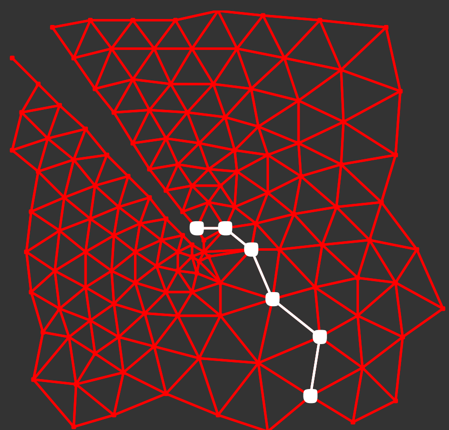
Space = edges + vertices = $\sum_{\text{vertices}}$ (1 + # higher neighbors)

Time = flops = $\sum_{\text{vertices}}$ (1+ # higher neighbors)^2

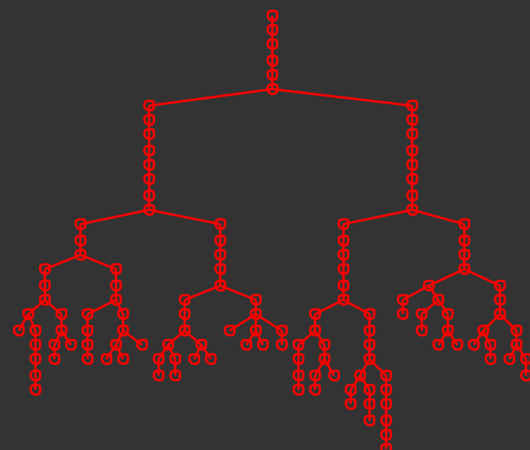
UCSB

Nested dissection and graph partitioning

[George 1973, many extensions]

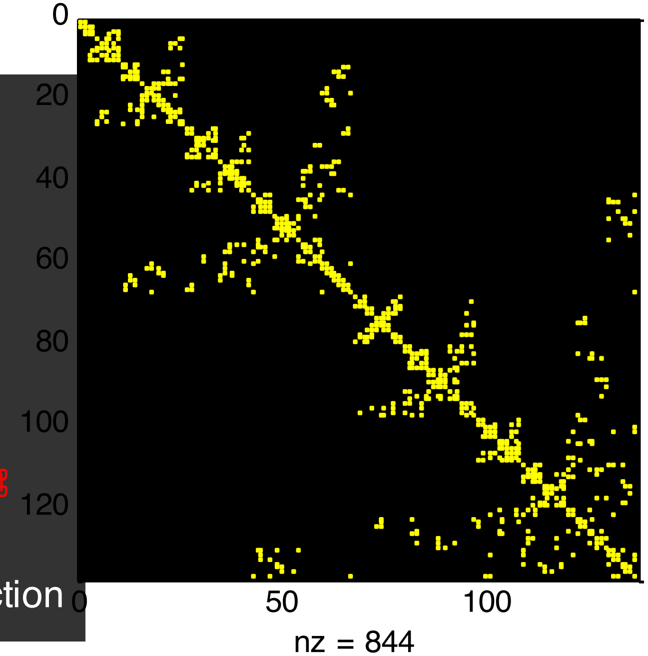


Vertex separator in graph of matrix



Elimination tree with nested dissection

Matrix reordered by nested dissection



- Find a small vertex separator, number it last, recurse on subgraphs
- Theory: approx optimal separators => approx optimal fill & flop count

Separators in theory

- Planar graphs have $O(n^{1/2})$ - separators.
- Well-shaped finite element meshes in 3 dimensions have $O(n^{2/3})$ - separators.
- Also some others – trees, bounded genus, chordal graphs, bounded-excluded-minor graphs, …
- Most of these theorems come with efficient algorithms, but they aren't used much – heuristics do okay.
- Random graphs don't have good separators.
  - e.g. Erdos-Renyi graphs have only $O(n)$ - separators.

Separators in practice

- Graph partitioning heuristics have been an active research area for many years, often motivated by partitioning for parallel computation.
- Some techniques:
  - Spectral partitioning (using Laplacian eigenvectors)
  - Geometric partitioning (meshes with vertex coordinates)
  - Iterative swapping (Kernighan-Lin, Fiduccia-Matheysses)
  - Breadth-first search (fast but low quality)
- Many popular modern codes (e.g. Zoltan, Metis) use multilevel iterative swapping

UCSB

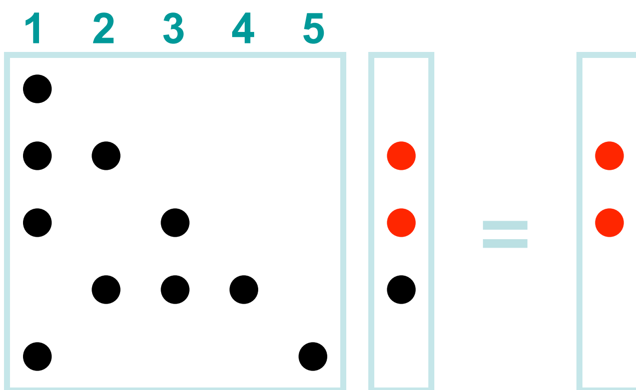
Graph algorithms in sparse matrix computation

Many, many graph algorithms have been used, invented, implemented at large scale for sparse matrix computation:

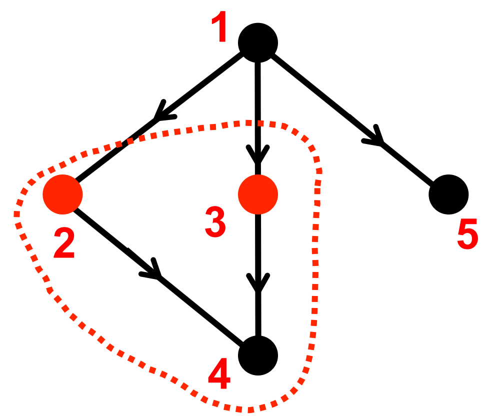
- Symmetric problems: elimination tree, nonzero structure prediction, sparse triangular solve, sparse matrix-matrix multiplication, min-height etree, …
- Nonsymmetric problems: sparse triangular solve, bipartite matching (weighted and unweighted), Dulmage-Mendelsohn decomposition / strong components, …
- Iterative methods: graph partitioning again, independent set, low-stretch spanning trees, …

UCSB

Sparse-sparse triangular solve



L x b



$G(L^T)$

Symbolic:

Predict structure of x by search from nonzeros of b

Numeric:

Compute values of x in topological order

$$\boxed{\textbf{Time} = \textbf{O(flops)}}$$

Sparse Cholesky factorization to solve $Ax = b$

1. Preorder: replace A by PAP^T and b by Pb
   - Independent of numerics

2. Symbolic Factorization: build static data structure
   - Elimination tree
   - Nonzero counts
   - Supernodes
   - Nonzero structure of L

3. Numeric Factorization: $A = LL^T$
   - Static data structure
   - Supernodes use BLAS3 to reduce memory traffic

4. Triangular Solves: solve $Ly = b$, then $L^Tx = y$

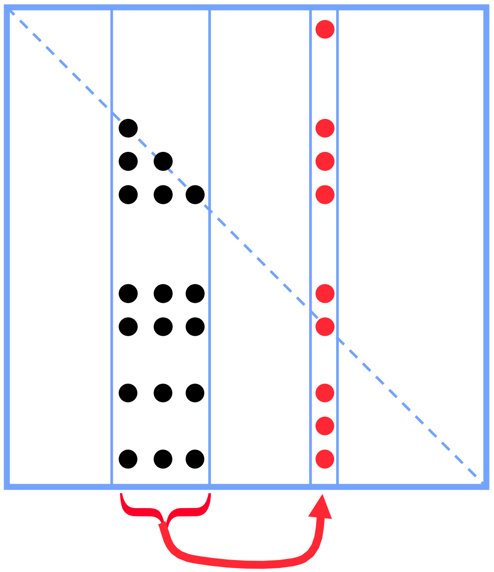
Chordal graphs, dense matrices, and communication

- A chordal graph can be compactly represented as a tree of overlapping cliques (complete subgraphs).
- A complete subgraph is a dense submatrix.
- Dense matrix ops do n^3 work for n^2 communication.
- Most of the ops in Gaussian elimination can be done within dense BLAS primitives, esp. DGEMM.

UCSB

Supernodes for Gaussian elimination

- Supernode = group of adjacent columns of L with same nonzero structure
- Related to clique structure of filled graph $G^+(A)$

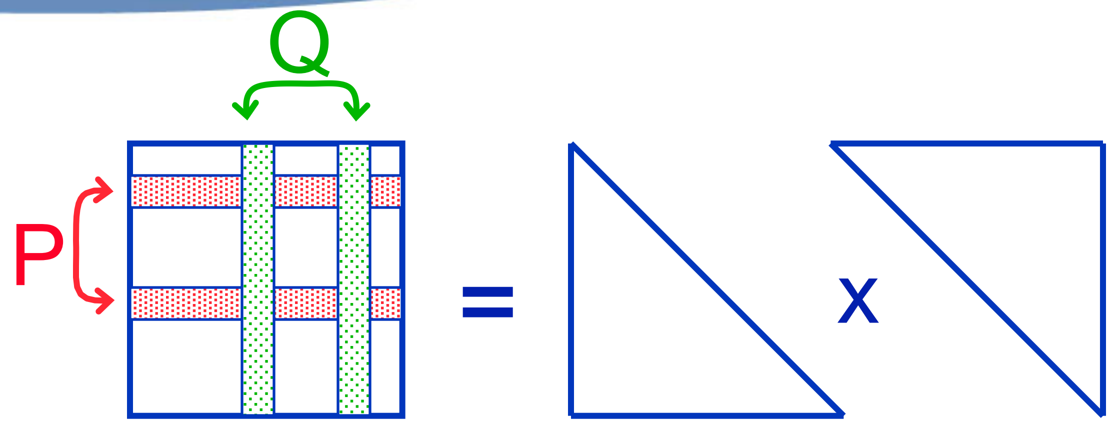


- Supernode-column update: k sparse vector ops become

  1 dense triangular solve
  + 1 dense matrix * vector
  + 1 sparse vector add

- Sparse BLAS 1 => Dense BLAS 2
- Supernode-panel or multifrontal updates => Dense BLAS 3

Aside: Nonsymmetric matrices and partial pivoting



- $PAQ^T = LU$: Q preorders columns for sparsity, P is row pivoting
- Column permutation of A ⇔ Symmetric permutation of A^TA
- Symmetric ordering: Nested dissection or approximate minimum degree
- But, forming A^TA is expensive (sometimes bigger than L+U).

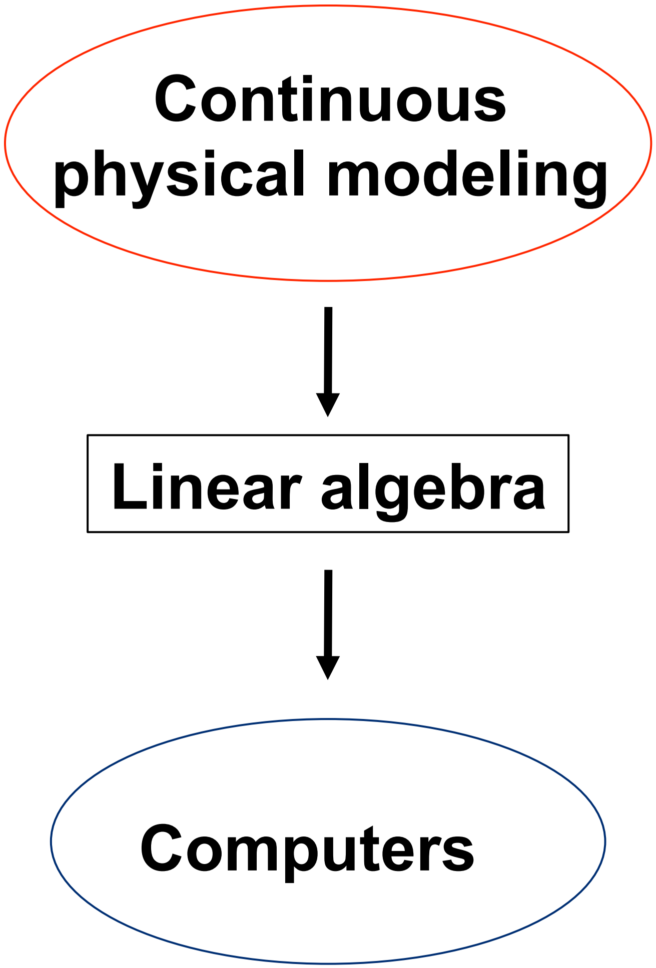
Aside: Nonsymmetric matrices and partial pivoting

Given the nonzero structure of (nonsymmetric) A,
one can find . . .

- column nested dissection or min degree permutation
- column elimination tree $T(A^TA)$
- row and column counts for $G^+(A^TA)$
- supernodes of $G^+(A^TA)$
- nonzero structure of $G^+(A^TA)$

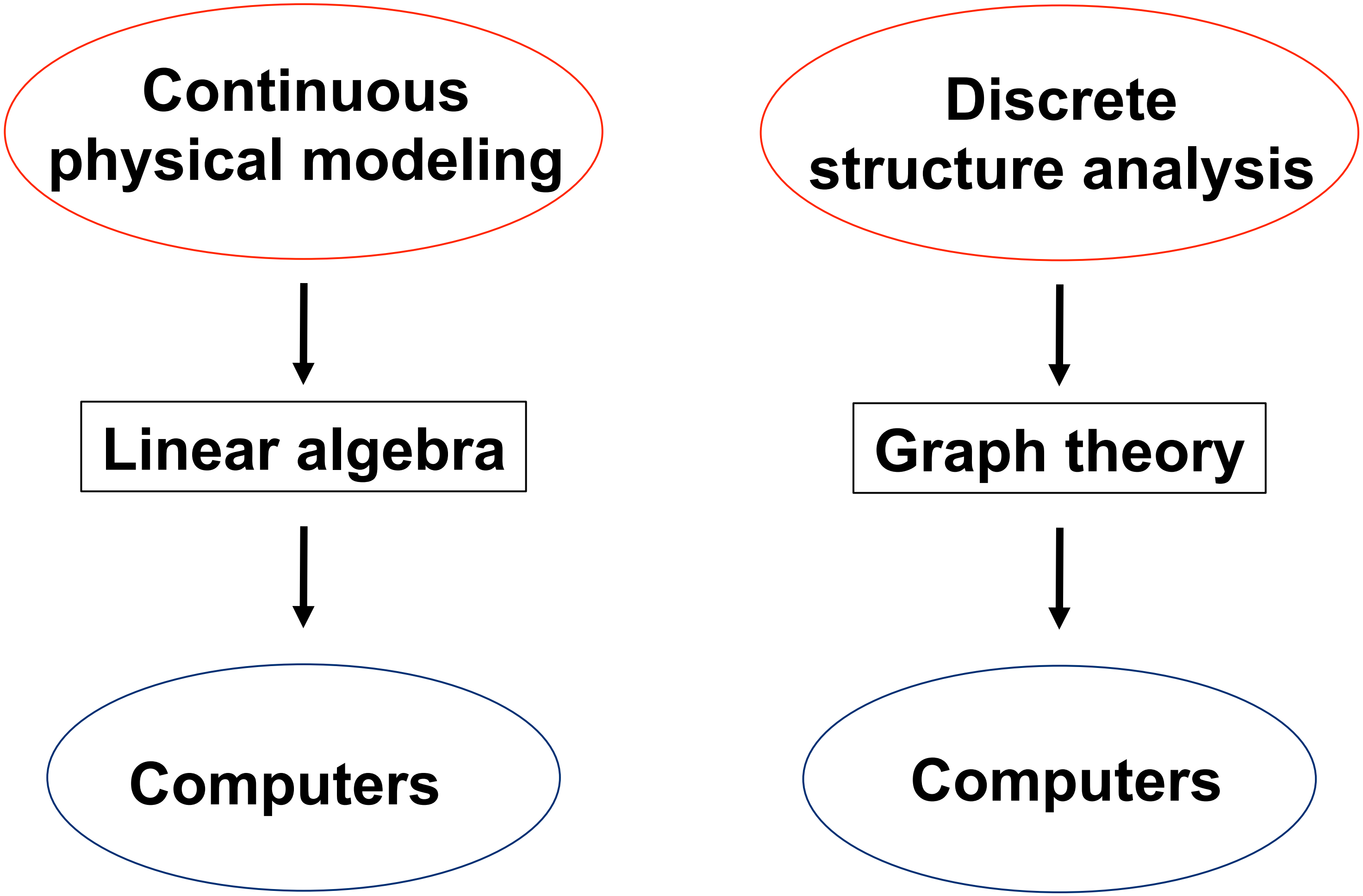
. . . without forming A^TA or $G_\cap(A)$.

The middleware of scientific computing



$$Ax = b$$

The middleware challenge for graph analysis



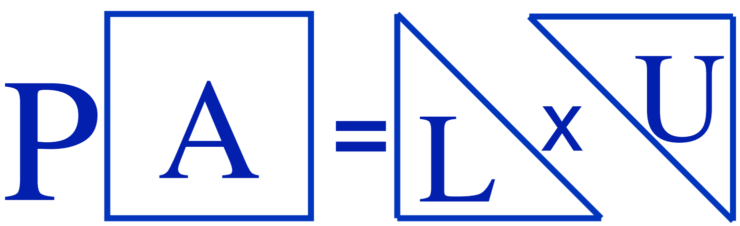
UCSB

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:

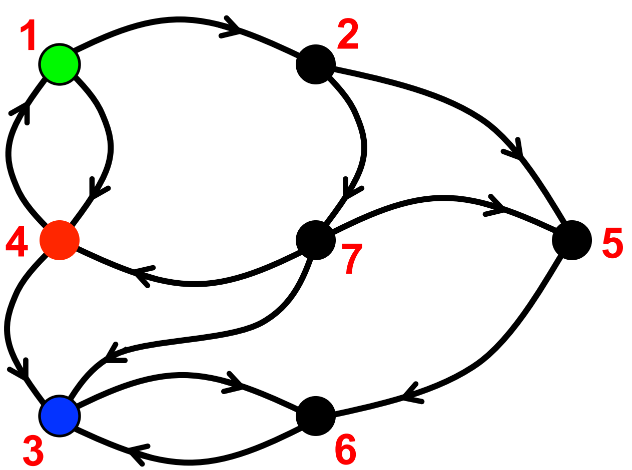
Solve a large system of linear equations by Gaussian elimination



Graph 500 List (June 2013)

Graph500 Benchmark:

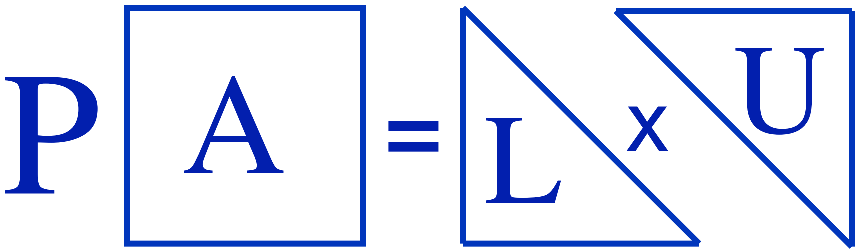
Breadth-first search in a large power-law graph



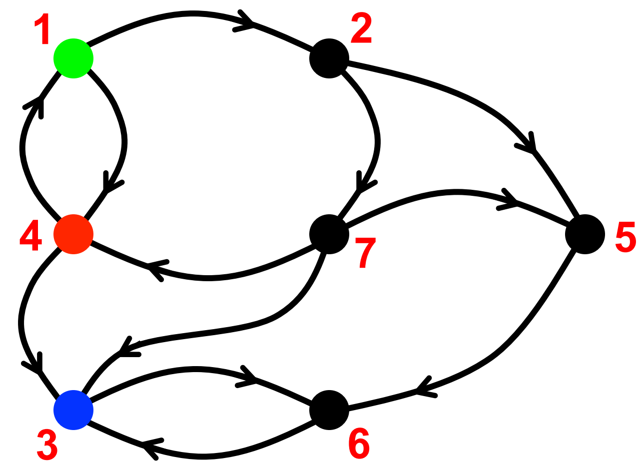
No.	Rank	Machine	Installation Site	Number of nodes	Number of cores	Problem scale	GTEPS
1	1	DOE/NNSA/LLNL Sequoia (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	Lawrence Livermore National Laboratory	65536	1048576	40	15363
2	2	DOE/SC/Argonne National Laboratory Mira (IBM - BlueGene/Q, Power BQC 16C 1.60 GHz)	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

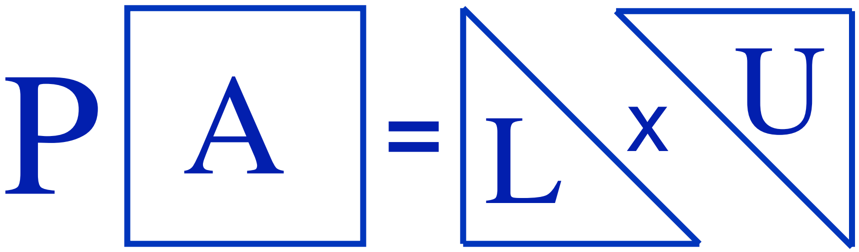


33.8 Peta / 15.3 Tera is about 2200.

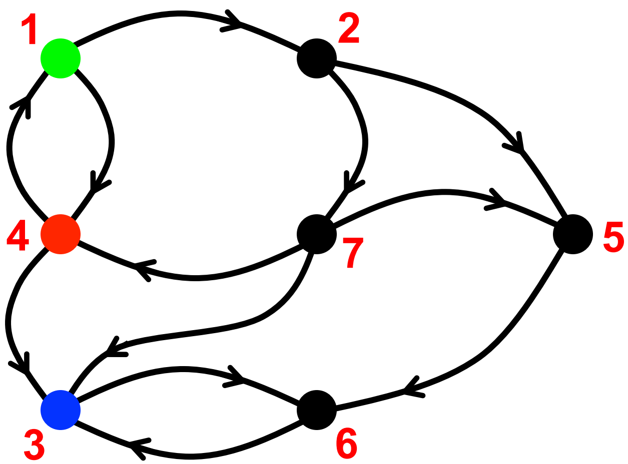
UCSB

Floating-point vs. graphs, June 2013

33.8 Petaflops



15.3 Terateps



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

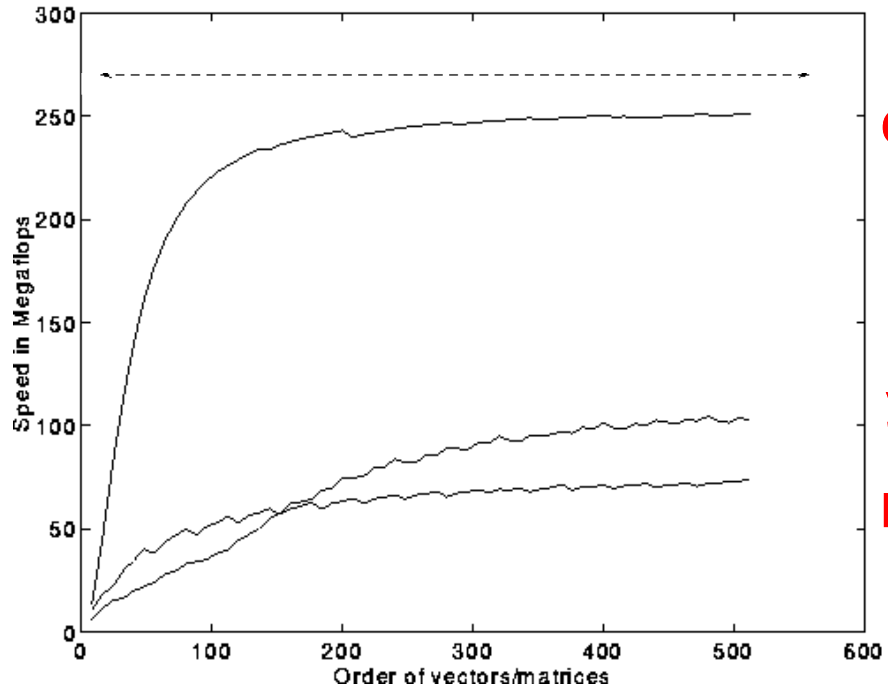
Nov 2010: **2.5 Peta / 6.6 Giga ~ 380,000**

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



$C = A*B$

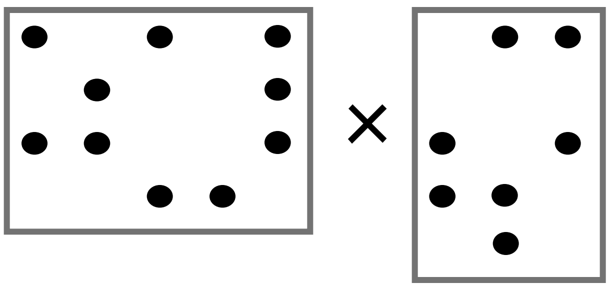
$y = A*x$

$\mu = x^T y$

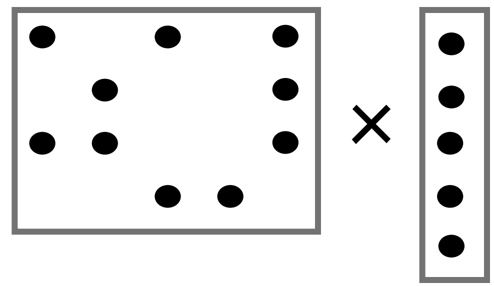
UCSB

Sparse array primitives for graph manipulation

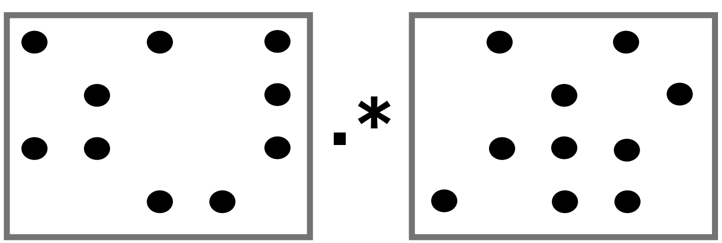
Sparse matrix-matrix multiplication (SpGEMM)



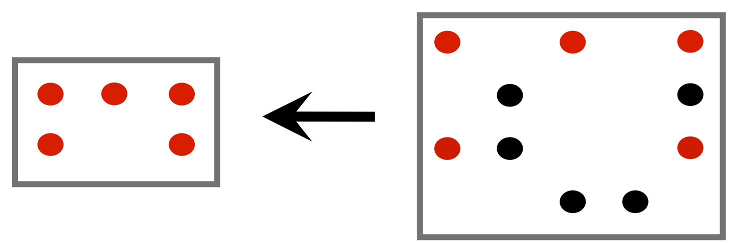
Sparse matrix-dense vector multiplication



Element-wise operations

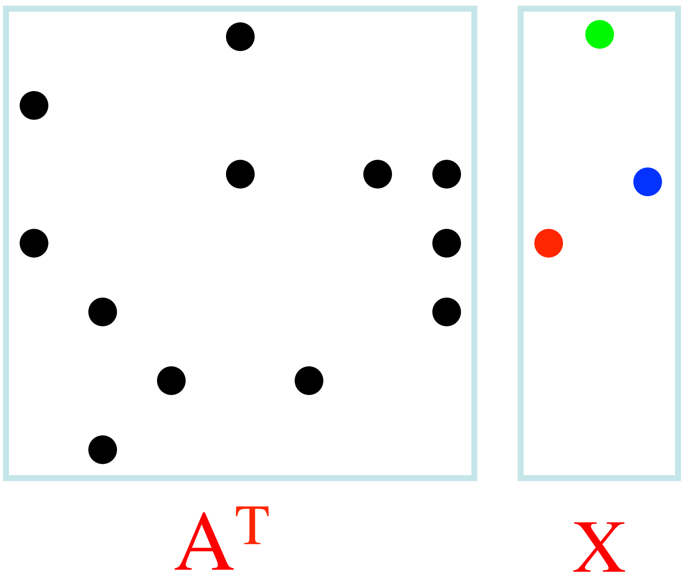


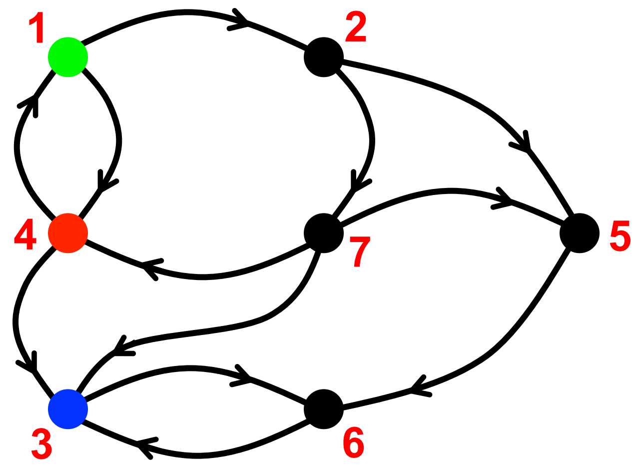
Sparse matrix indexing



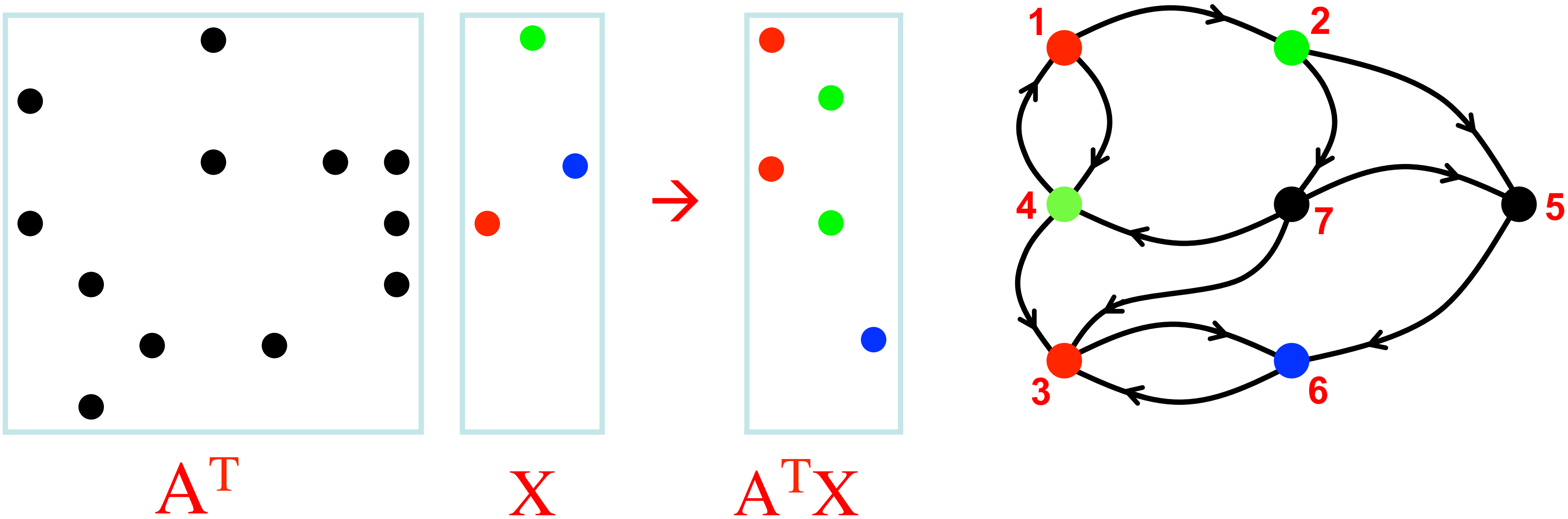
Matrices over various semirings: (+ . x), (min . +), (or . and), …

Multiple-source breadth-first search



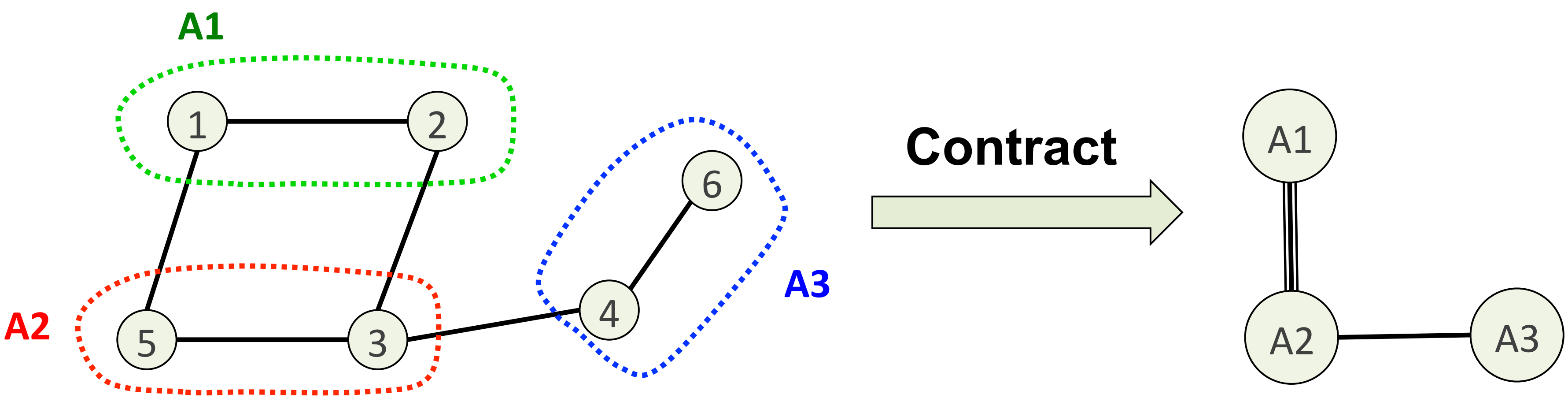


Multiple-source breadth-first search



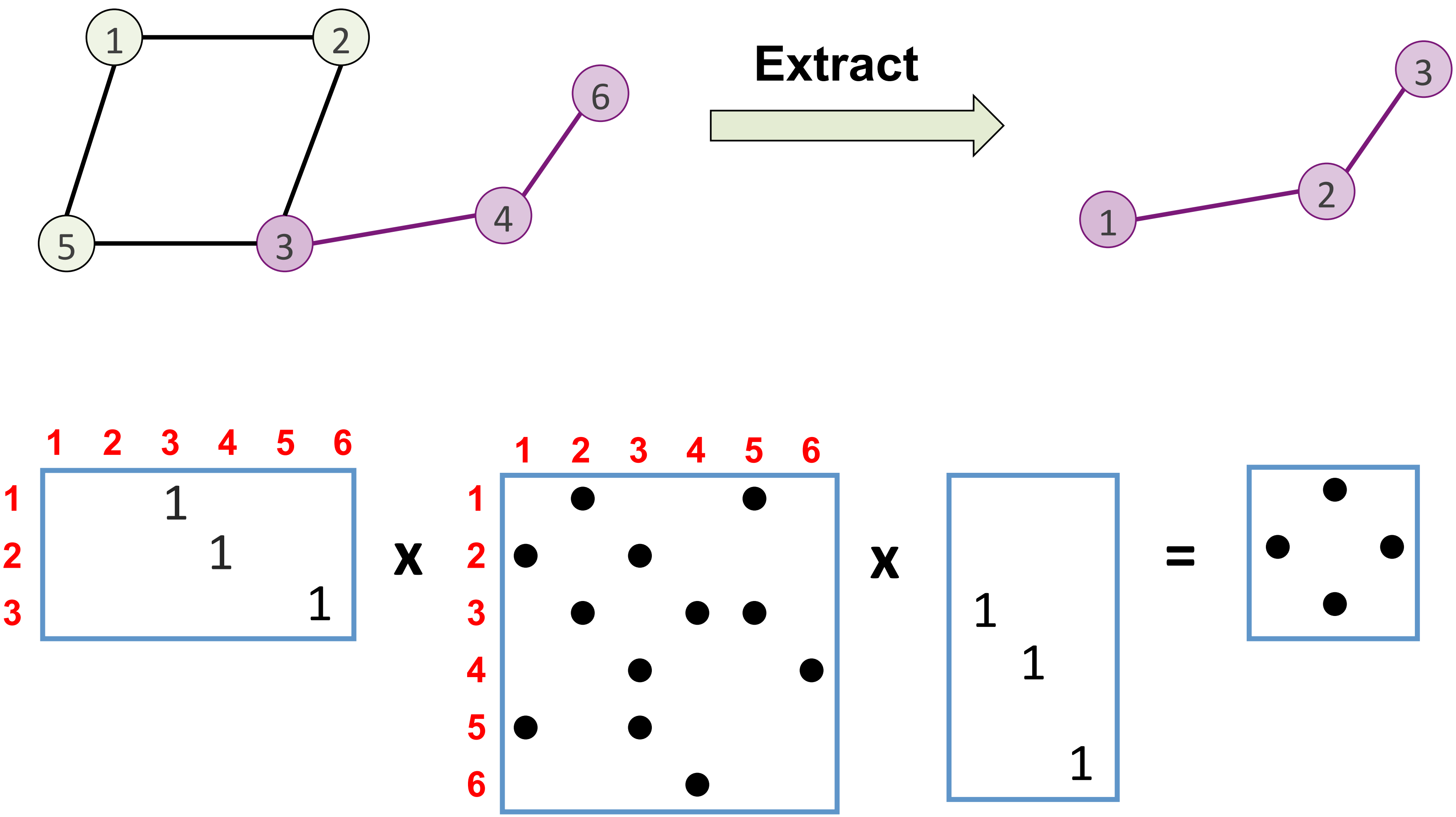
- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

Graph contraction via sparse triple product



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

Subgraph extraction via sparse triple product



Betweenness centrality *[Robinson 2008]*

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

$\mathbf{b} = 0$

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

        $\mathbf{w} = \mathbf{s}_{d,:} \times (1 + \mathbf{u}) \div \mathbf{p}$

        $\mathbf{w} = \mathbf{A}\mathbf{w}$

        $\mathbf{w} = \mathbf{w} \times \mathbf{s}_{d-1,:} \times \mathbf{p}$

        $\mathbf{u} = \mathbf{u} + \mathbf{w}$

        $d = d - 1$

    $\mathbf{b} = \mathbf{b} + \mathbf{u}$

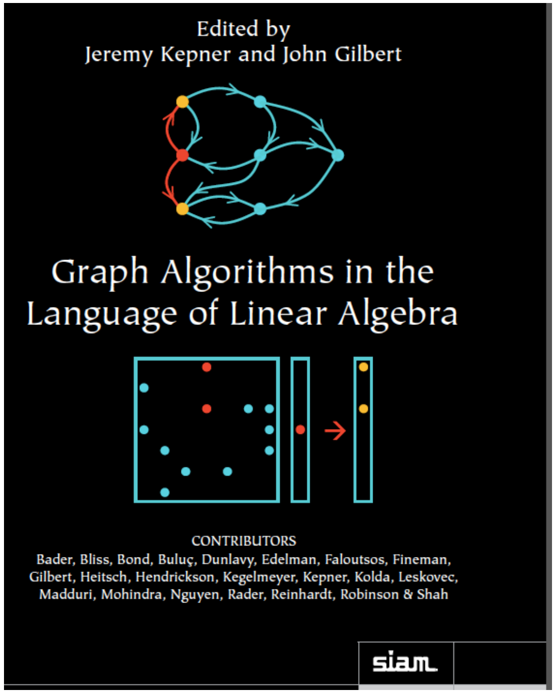
Variables:			Storage:
A :	sparse adjacency matrix	$B^{S(NxN)}$	O(M+N)
f :	sparse fringe vector	$Z^{S(N)}$	O(N)
p :	shortest path vector	Z^{N}	O(N)
S :	sparse depth matrix	$B^{S(NxN)}$	O(N)
u :	centrality update vector	R^{N}	O(N)

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, implemented with CombBLAS



UCSB

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 $\times$ 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

Question: Berry challenge problems

- Clustering coefficient (triangle counting)
- Connected components (bully algorithm)
- Maximum independent set (NP-hard)
- Maximal independent set (Luby algorithm)
- Single-source shortest paths
- Special betweenness (for subgraph isomorphism)

Question: Not materializing big matrix products

Recall: Given nonsymmetric A, one can find **. . .**

- column nested dissection or min degree permutation
- column elimination tree $T(A^TA)$
- row and column counts for $G^+(A^TA)$
- supernodes of $G^+(A^TA)$
- nonzero structure of $G^+(A^TA)$

. . . without forming A^TA.

- How generally can we do graph algorithms in linear algebra without storing intermediate results?
- Maybe related to Joey Gonzalez’s scheduling of vertex and edge operations in GraphLab.
- Maybe related to techniques for avoiding “boil the ocean” database queries.

UCSB

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`

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.

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

- Matrix computation is beginning to repay a 50-year debt to graph algorithms.
- Graphs in the language of linear algebra are sufficiently mature to support a standard set of BLAS.
- It helps to look at things from two directions.