Tools and Primitives for High Performance Graph Computation

John R. Gilbert
University of California, Santa Barbara

Aydin Buluç (LBNL)
Adam Lugowski (UCSB)

SIAM Minisymposium on
Analyzing Massive Real-World Graphs
July 12, 2010

Support: NSF, DARPA, DOE, Intel

An analogy?

Continuous physical modeling

↓

Linear algebra

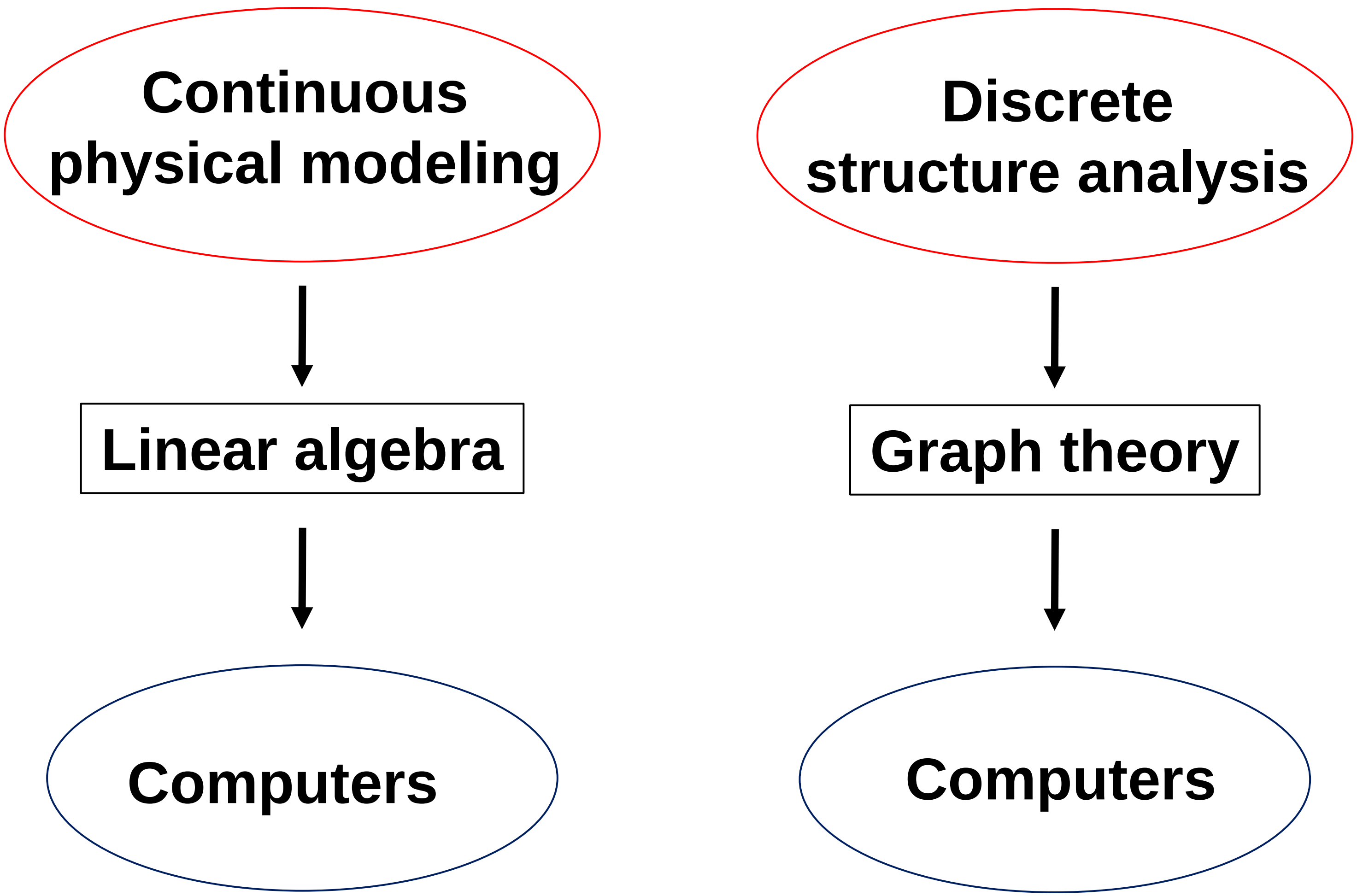
↓

Computers

As the "middleware" of scientific computing, linear algebra has supplied or enabled:

- Mathematical tools
- "Impedance match" to computer operations
- High-level primitives
- High-quality software libraries
- Ways to extract performance from computer architecture
- Interactive environments

An analogy?



UCSB

An analogy? Well, we're not there yet ….

- √ Mathematical tools
- ? "Impedance match" to computer operations
- ? High-level primitives
- ? High-quality software libs
- ? Ways to extract performance from computer architecture
- ? Interactive environments

Discrete structure analysis

↓

Graph theory

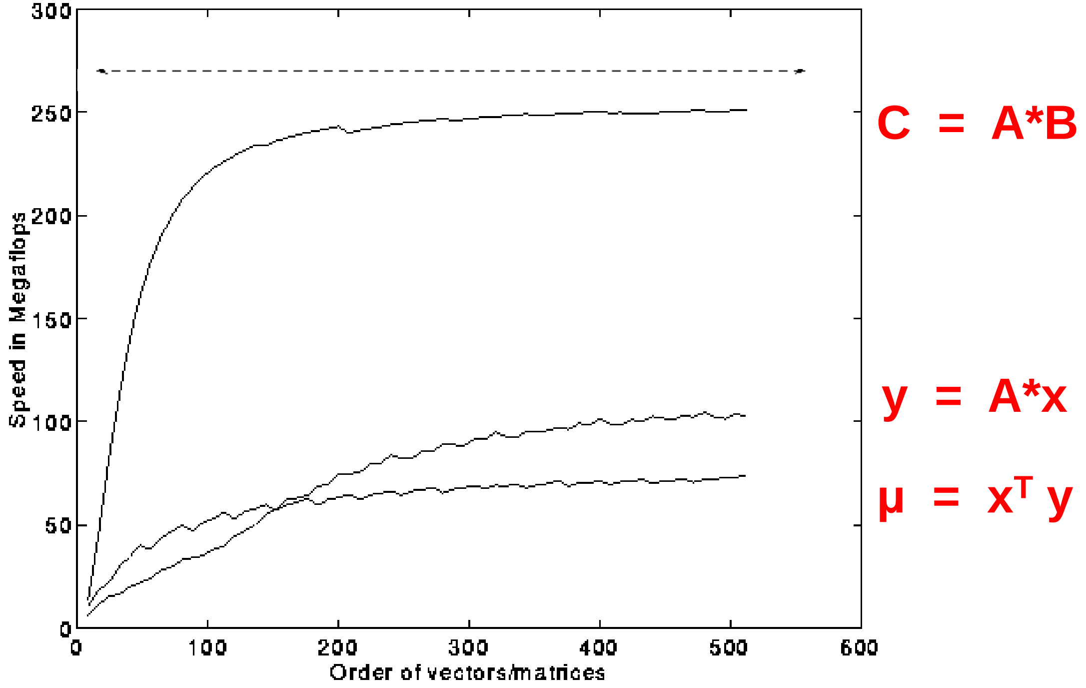
↓

Computers

The Primitives Challenge

- By analogy to numerical scientific computing. . .

Basic Linear Algebra Subroutines (BLAS): Speed (MFlops) vs. Matrix Size (n)



- What should the combinatorial BLAS look like?

Primitives should …

- Supply a common notation to express computations
- Have broad scope but fit into a concise framework
- Allow programming at the appropriate level of abstraction and granularity
- Scale seamlessly from desktop to supercomputer
- Hide architecture-specific details from users

The case for sparse matrices

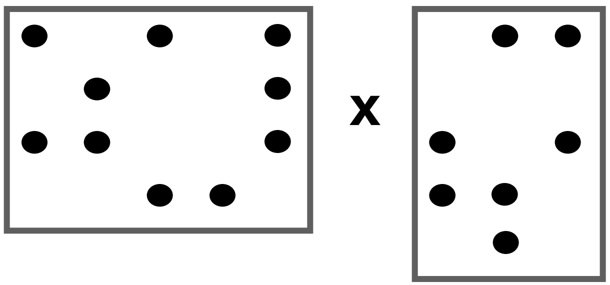
Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

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

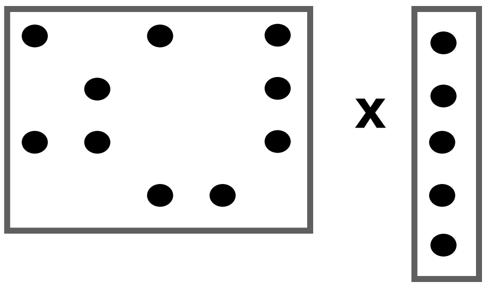
UCSB

Sparse array-based primitives

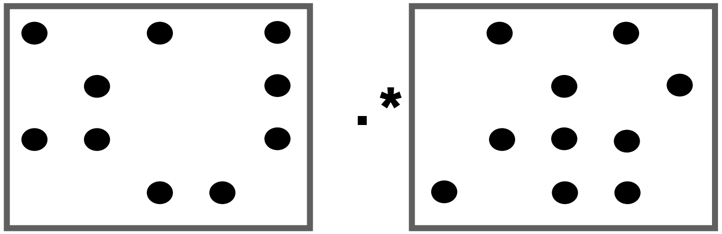
Sparse matrix-matrix multiplication (SpGEMM)



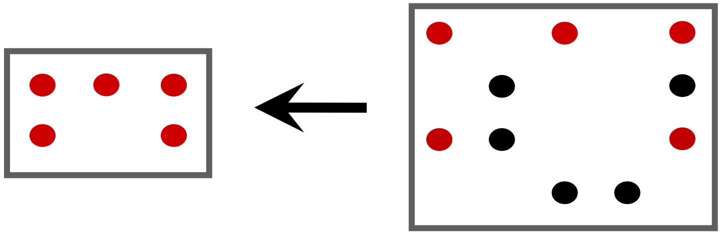
Sparse matrix-dense vector multiplication



Element-wise operations

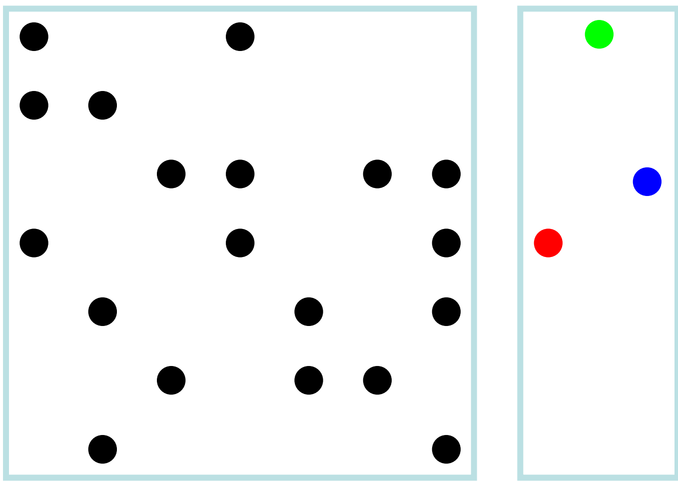


Sparse matrix indexing

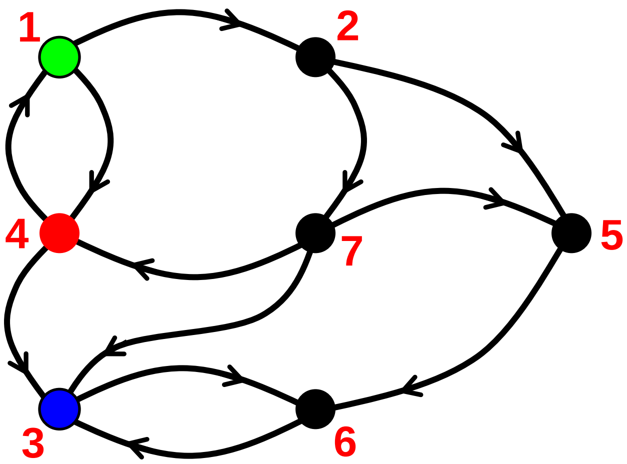


Matrices on various semirings: (x, +) , (and, or) , (+, min) , …

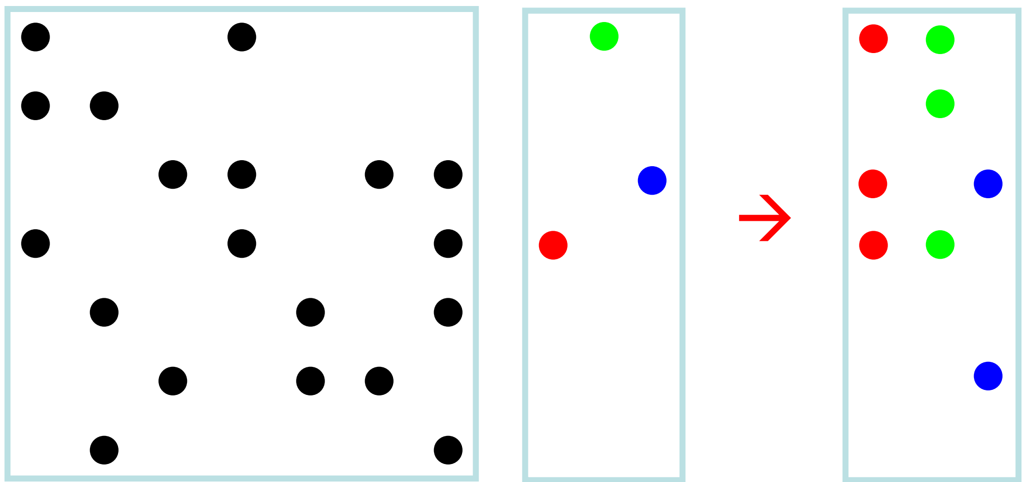
Multiple-source breadth-first search



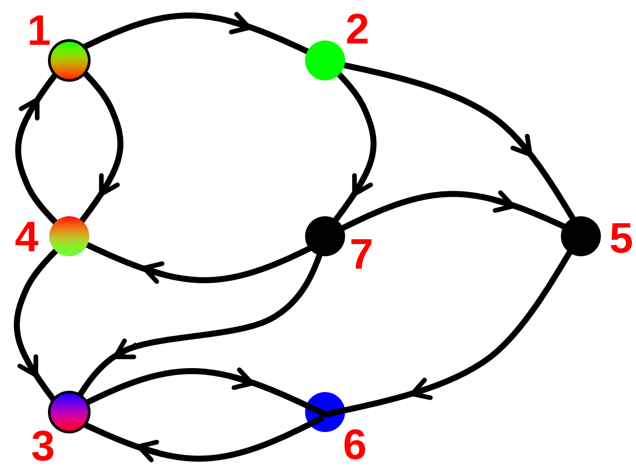
A^T X



Multiple-source breadth-first search

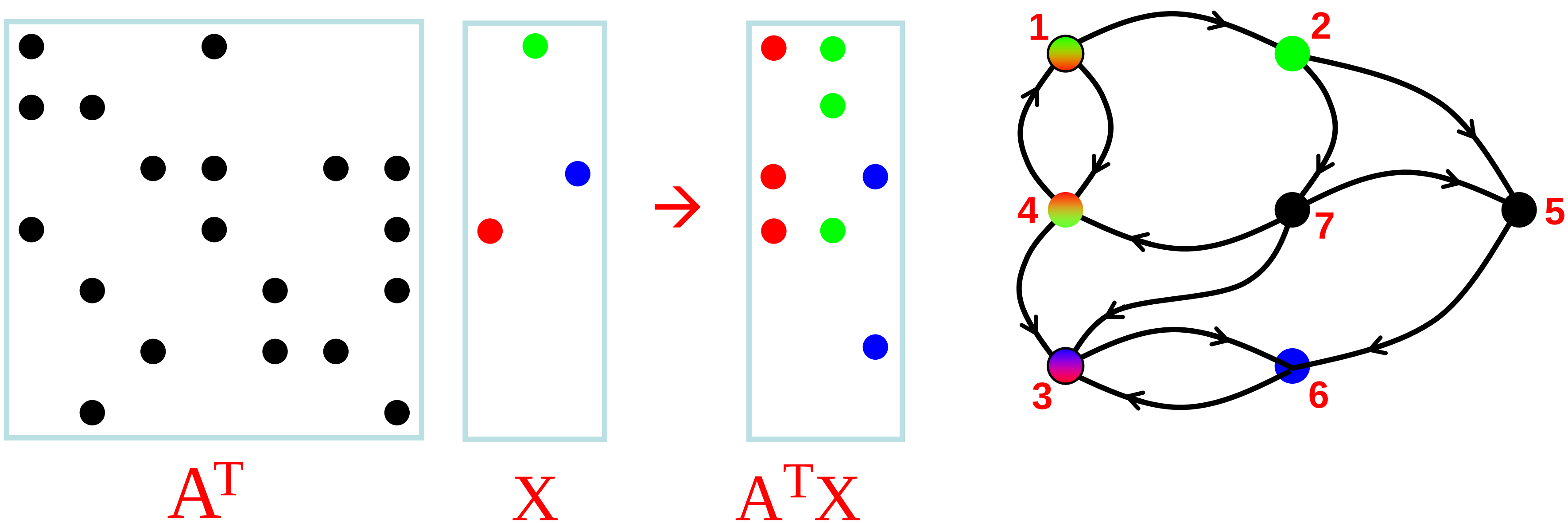


A^T $\quad X \quad \rightarrow \quad A^TX$



UCSB

Multiple-source breadth-first search



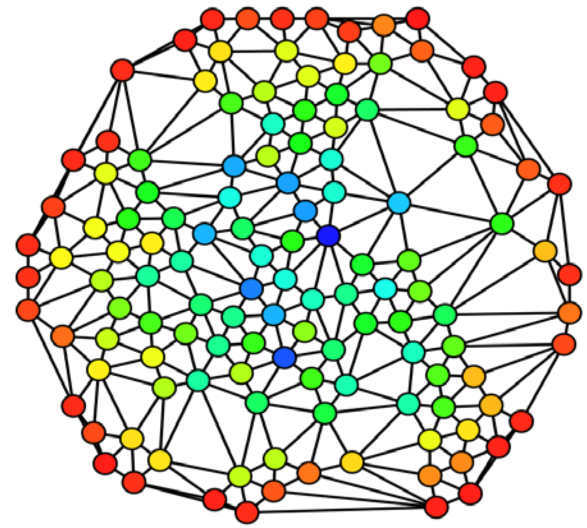
- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

UCSB

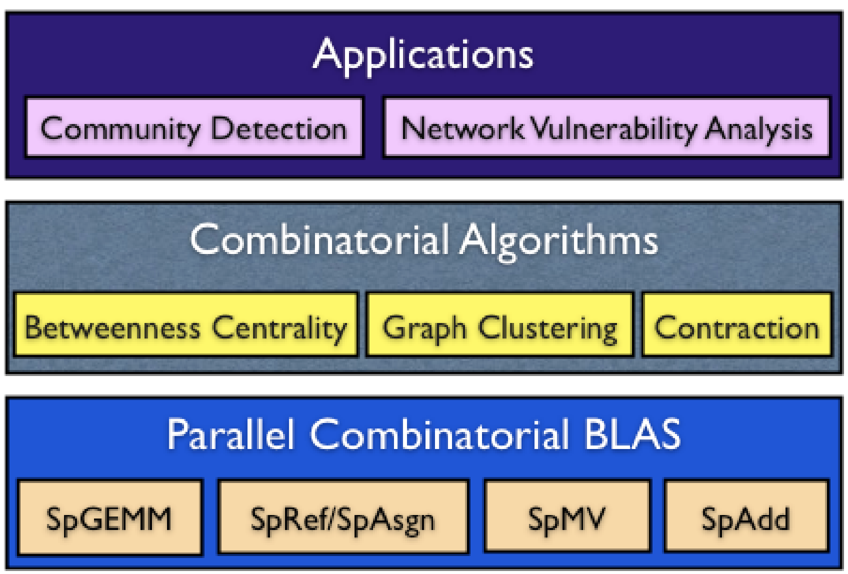
A Few Examples

Combinatorial BLAS
[Buluc, G]

A parallel graph library based on distributed-memory sparse arrays and algebraic graph primitives



Typical software stack



Betweenness Centrality (BC)

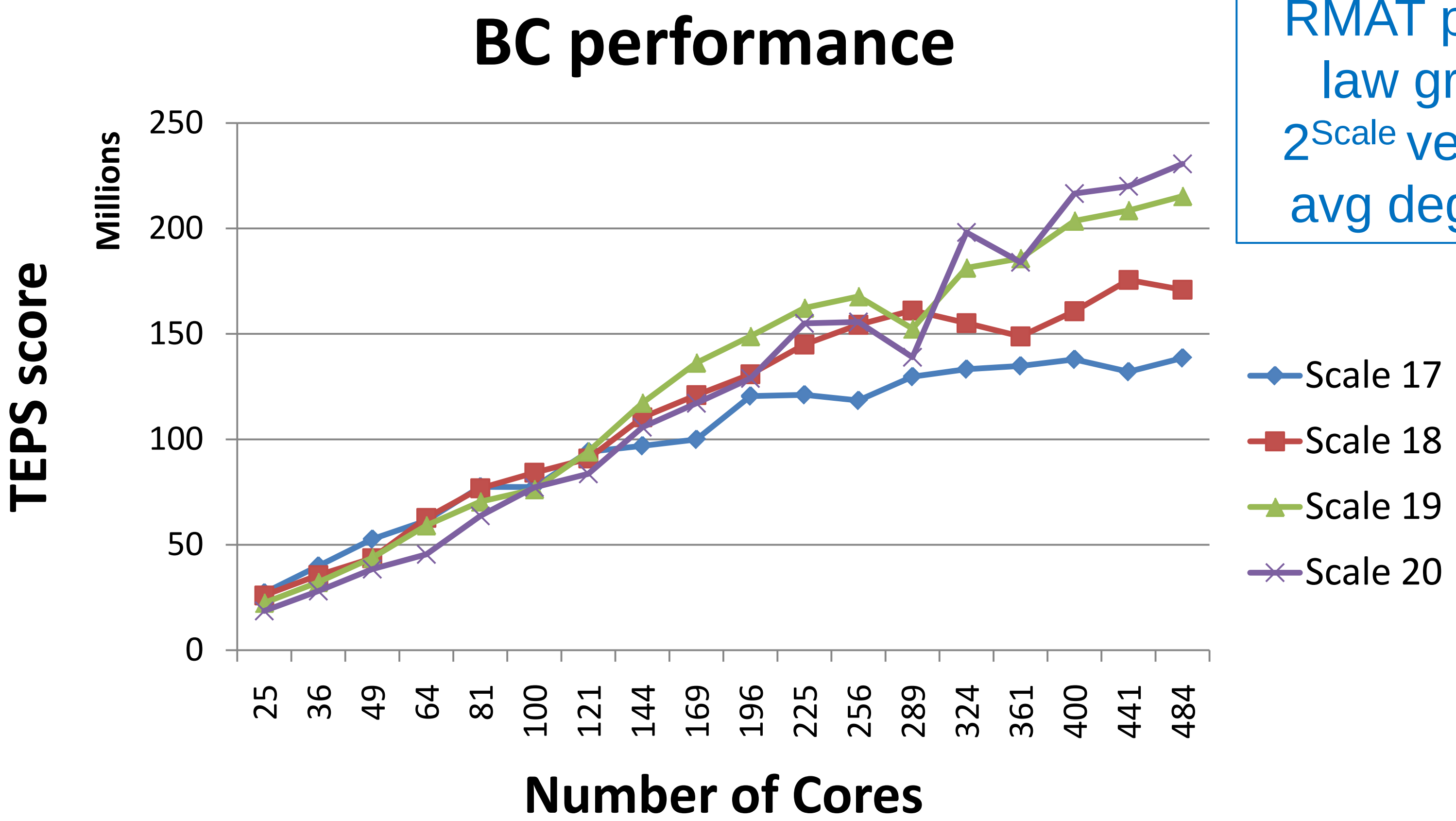
What fraction of shortest paths pass through this node?

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

Brandes' algorithm

UCSB

BC performance in distributed memory



RMAT power-law graph, 2^{Scale} vertices, avg degree 8

- TEPS = Traversed Edges Per Second
- One page of code using C-BLAS

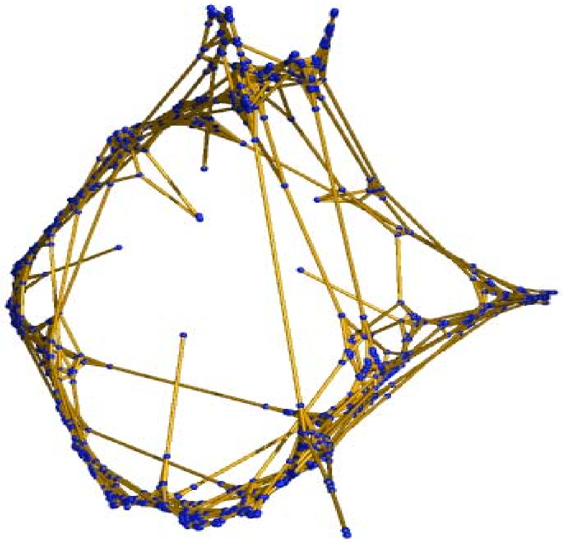
UCSB

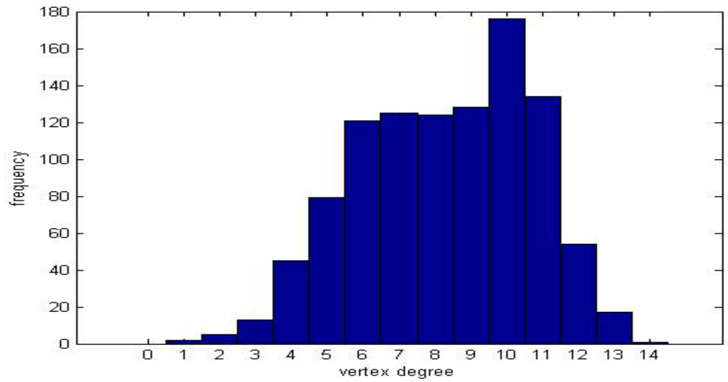
KDT: A toolbox for graph analysis and pattern discovery

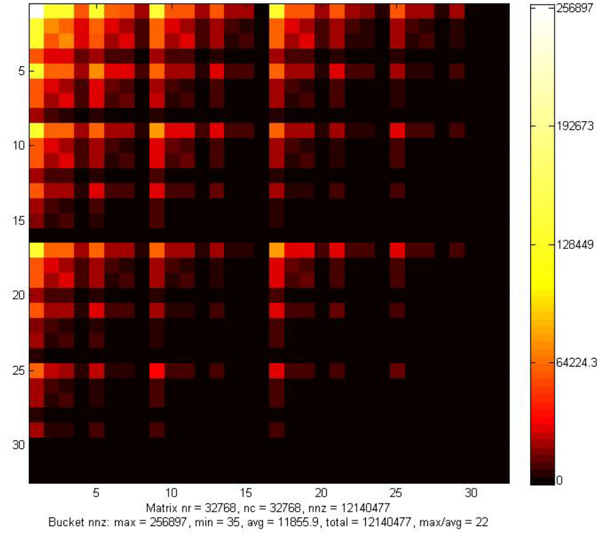
[G, Reinhardt, Shah]

Layer 1: Graph Theoretic Tools

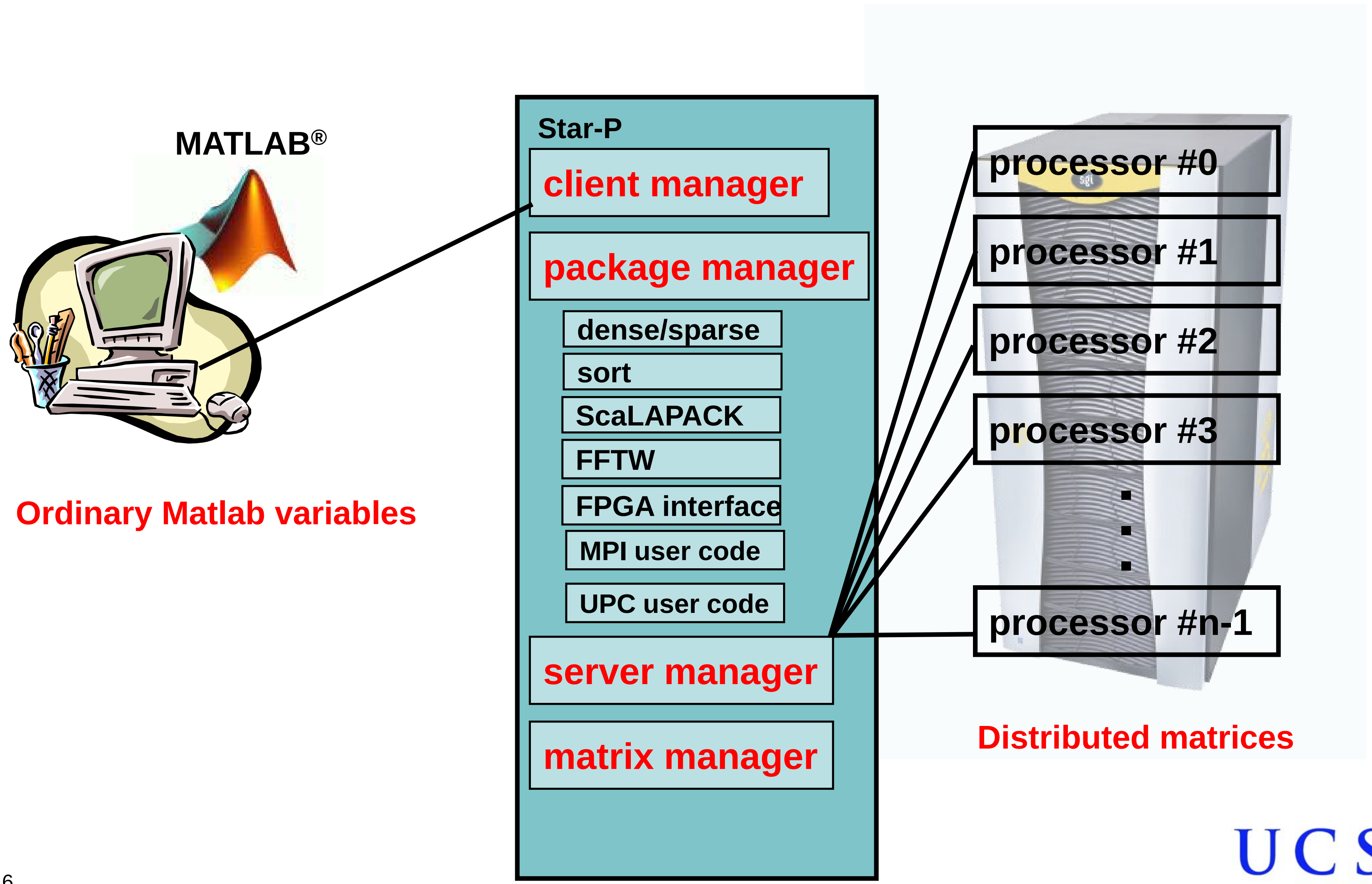
- Graph operations
- Global structure of graphs
- Graph partitioning and clustering
- Graph generators
- Visualization and graphics
- Scan and combining operations
- Utilities







Star-P architecture

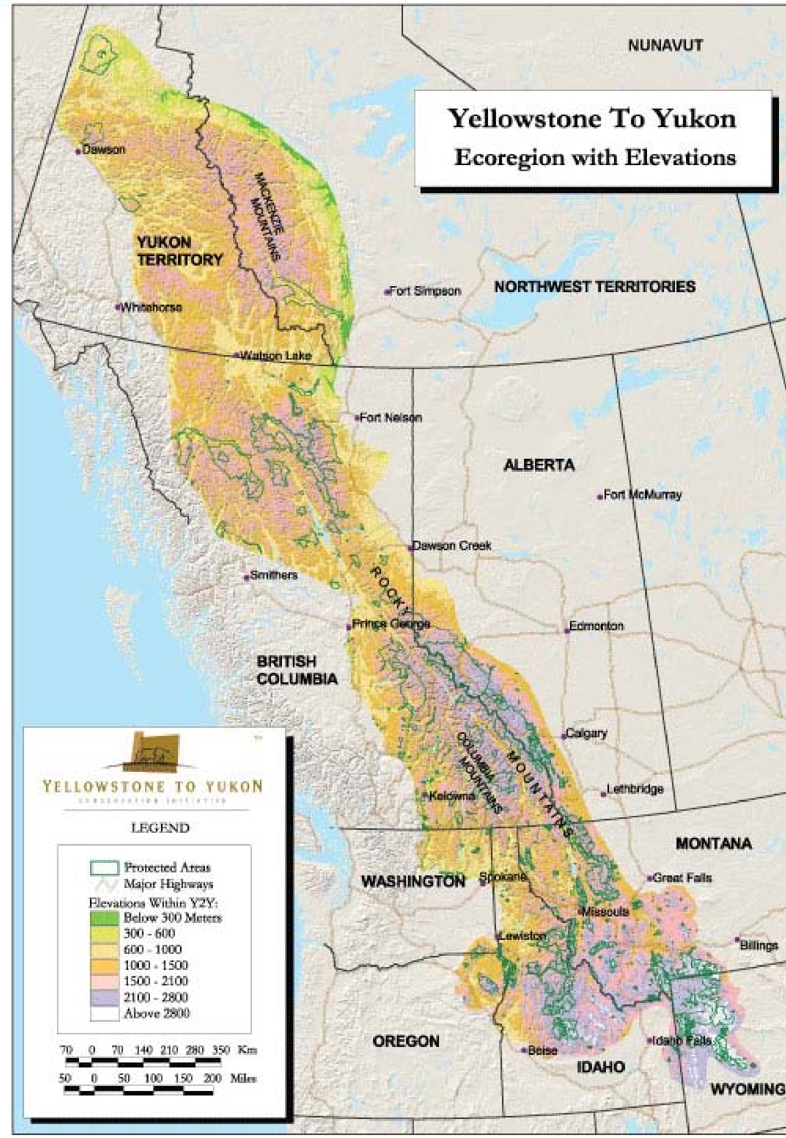


Landscape connectivity modeling

- Habitat quality, gene flow, corridor identification, conservation planning

- Pumas in southern California: 12 million nodes, < 1 hour

- Targeting larger problems: Yellowstone-to-Yukon corridor



Figures courtesy of Brad McRae

UCSB

Circuitscape [McRae, Shah]

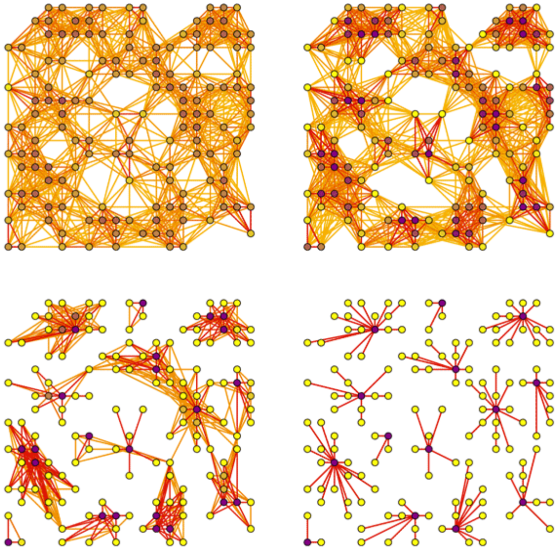
- Predicting gene flow with resistive networks
- Matlab, Python, and Star-P (parallel) implementations
- Combinatorics:
  - Initial discrete grid: ideally 100m resolution (for pumas)
  - Partition landscape into connected components
  - Graph contraction: habitats become nodes in resistive network
- Numerics:
  - Resistance computations for pairs of habitats in the landscape
  - Iterative linear solvers invoked via Star-P: Hypre (PCG+AMG)

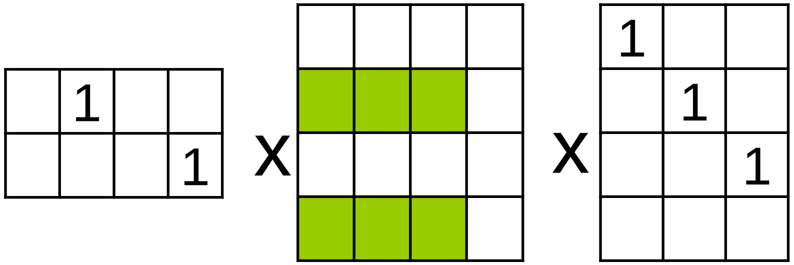
UCSB

A Few Nuts & Bolts

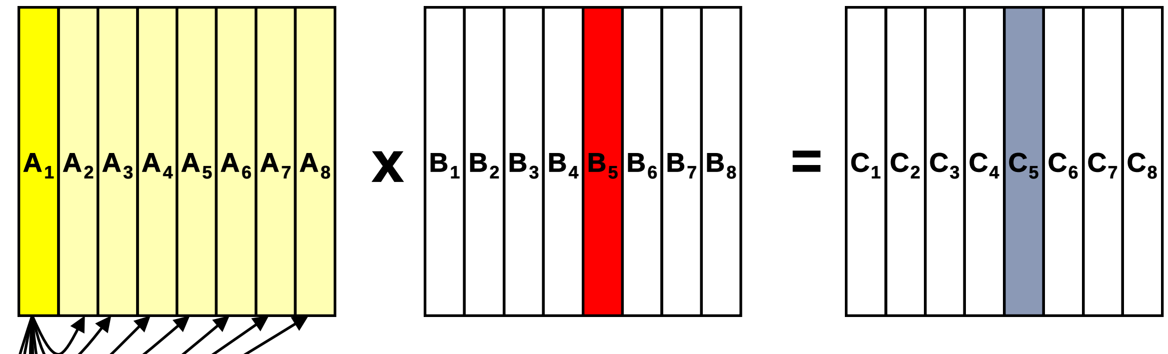
SpGEMM: sparse matrix x sparse matrix

- Graph clustering (Markov, peer pressure)
- Subgraph / submatrix indexing
- Shortest path calculations
- Betweenness centrality
- Graph contraction
- Cycle detection
- Multigrid interpolation & restriction
- Colored intersection searching
- Applying constraints in finite element computations
- Context-free parsing ...



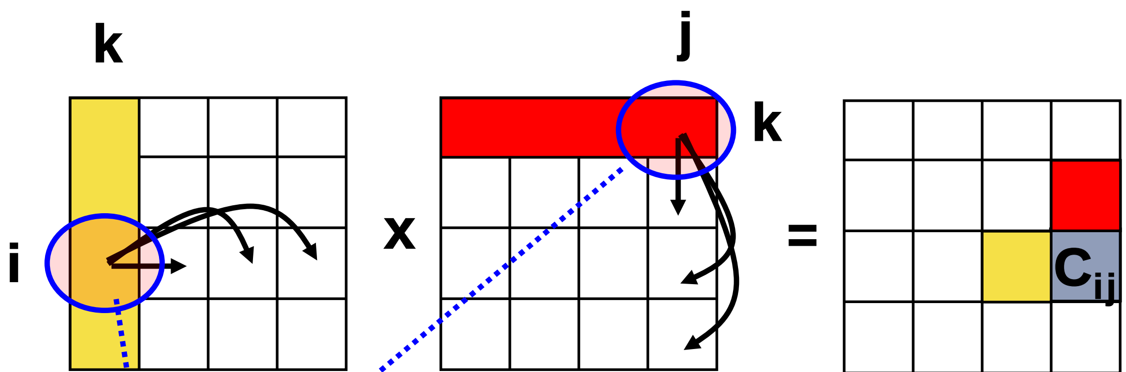


Two Versions of Sparse GEMM



$C_i = C_i + A\ B_i$

1D block-column distribution

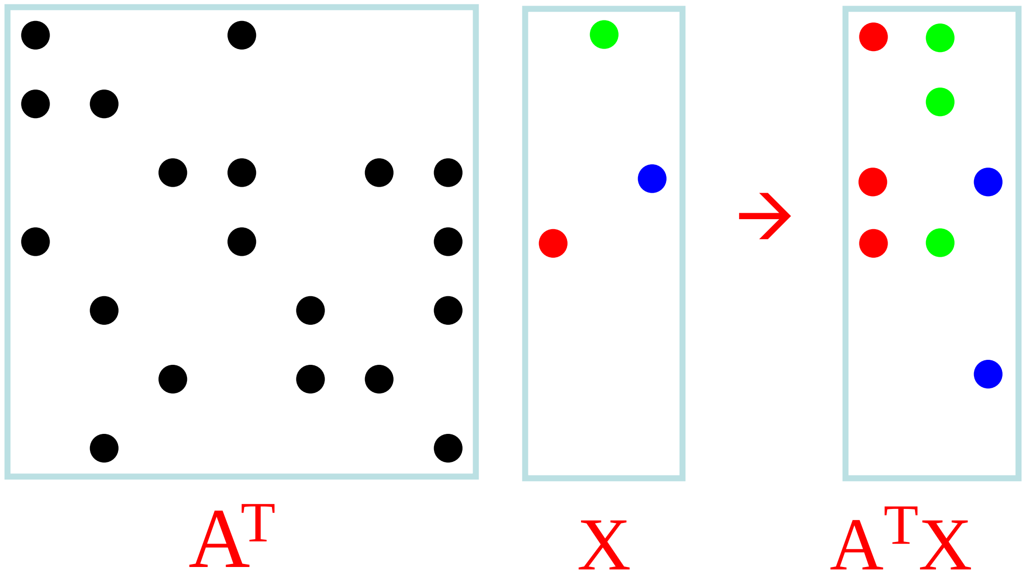


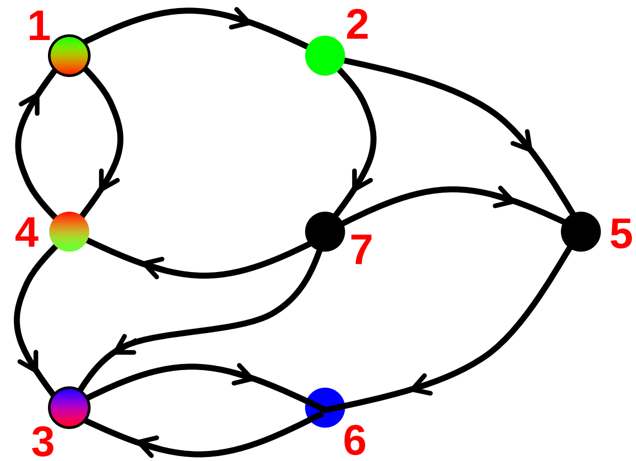
$C_{ij} \mathrel{+}= A_{ik}\ B_{kj}$

2D block distribution

UCSB

Parallelism in multiple-source BFS

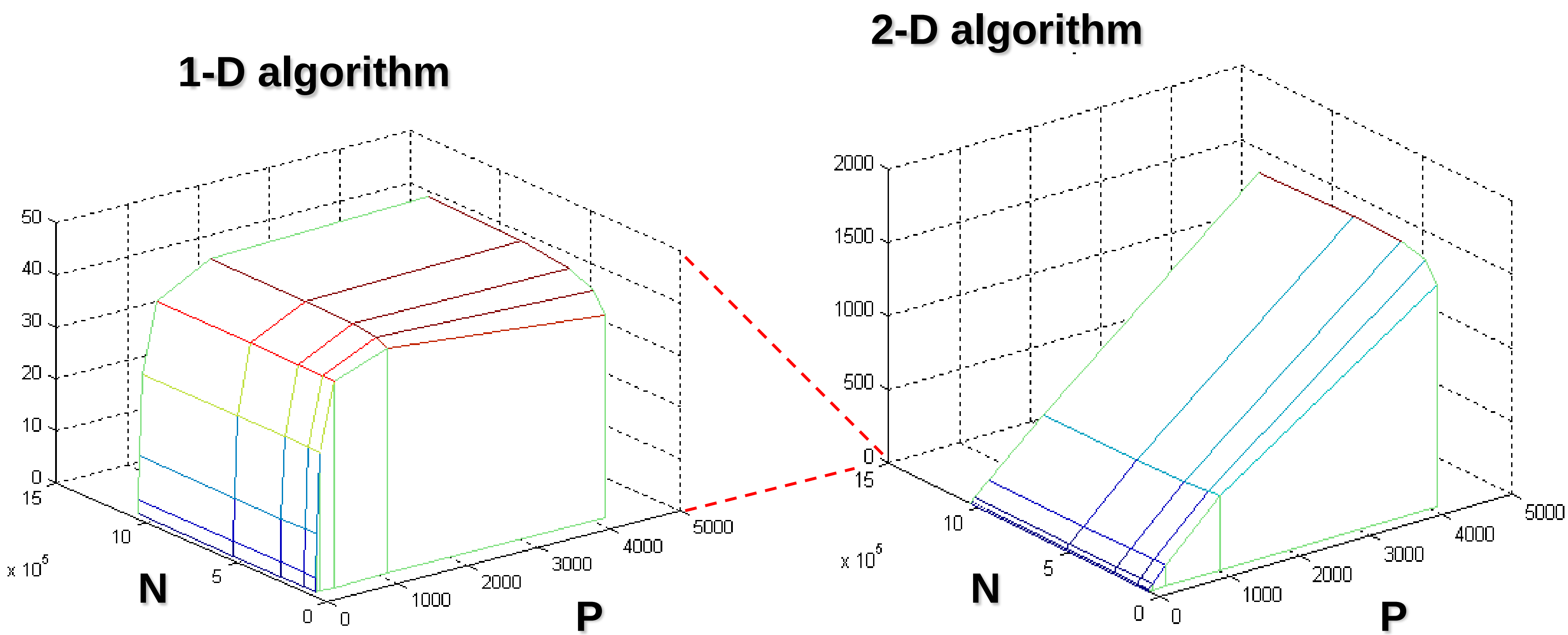




Three levels of parallelism from 2-D data decomposition:

- columns of X : over multiple simultaneous searches
- rows of X & columns of A^T: over multiple frontier nodes
- rows of A^T: over edges incident on high-degree frontier nodes

Modeled limits on speedup, sparse 1-D & 2-D



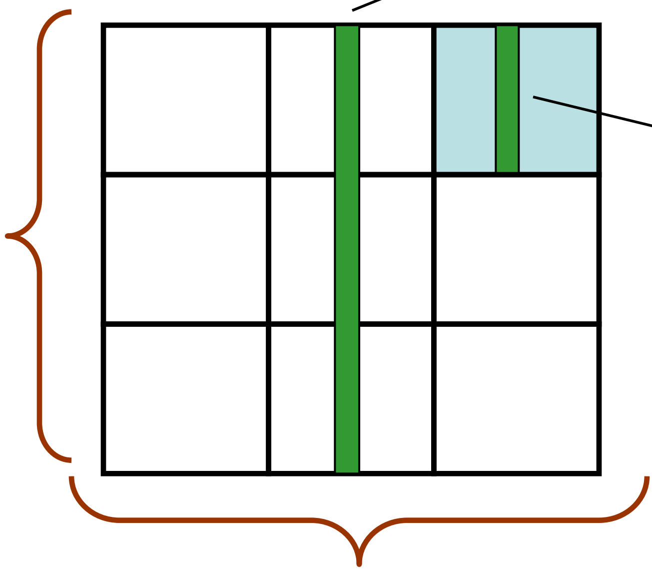
- 1-D algorithms do not scale beyond 40x
- Break-even point is around 50 processors

UCSB

Submatrices are ***hypersparse*** *(nnz << n)*

Average nonzeros per column $= c$

$\sqrt{p}$ blocks



$\sqrt{p}$ blocks

Average nonzeros per column within block $= \dfrac{c}{\sqrt{p}} \rightarrow 0$

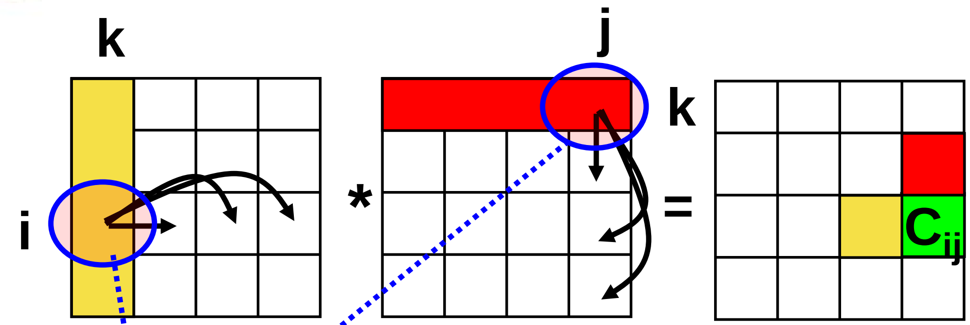
Total memory using compressed sparse columns $= O(n\sqrt{p} + nnz)$

Any algorithm whose complexity depends on matrix dimension *n* is asymptotically too wasteful.

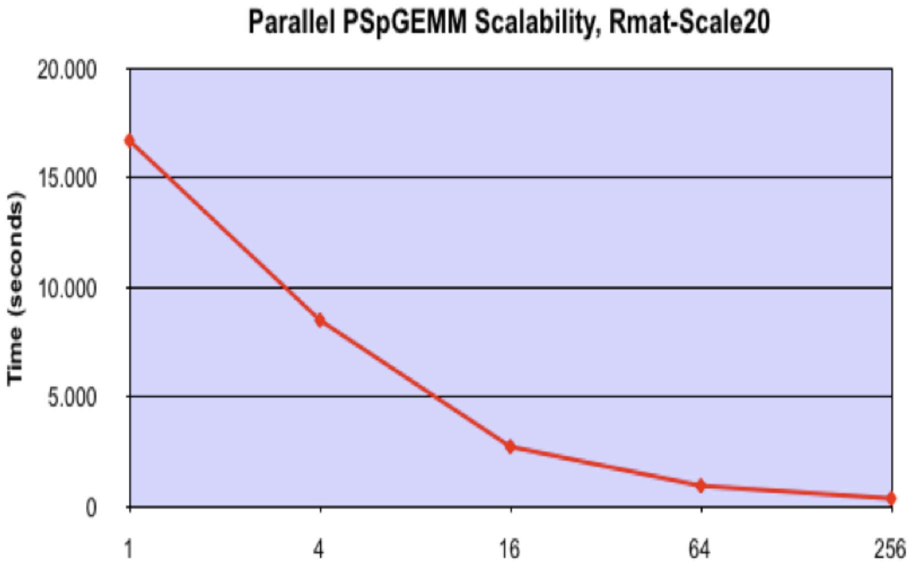
UCSB

Distributed-memory sparse matrix-matrix multiplication

- 2D block layout
- Outer product formulation
- Sequential "hypersparse" kernel



C_{ij} += A_{ik} * B_{kj}

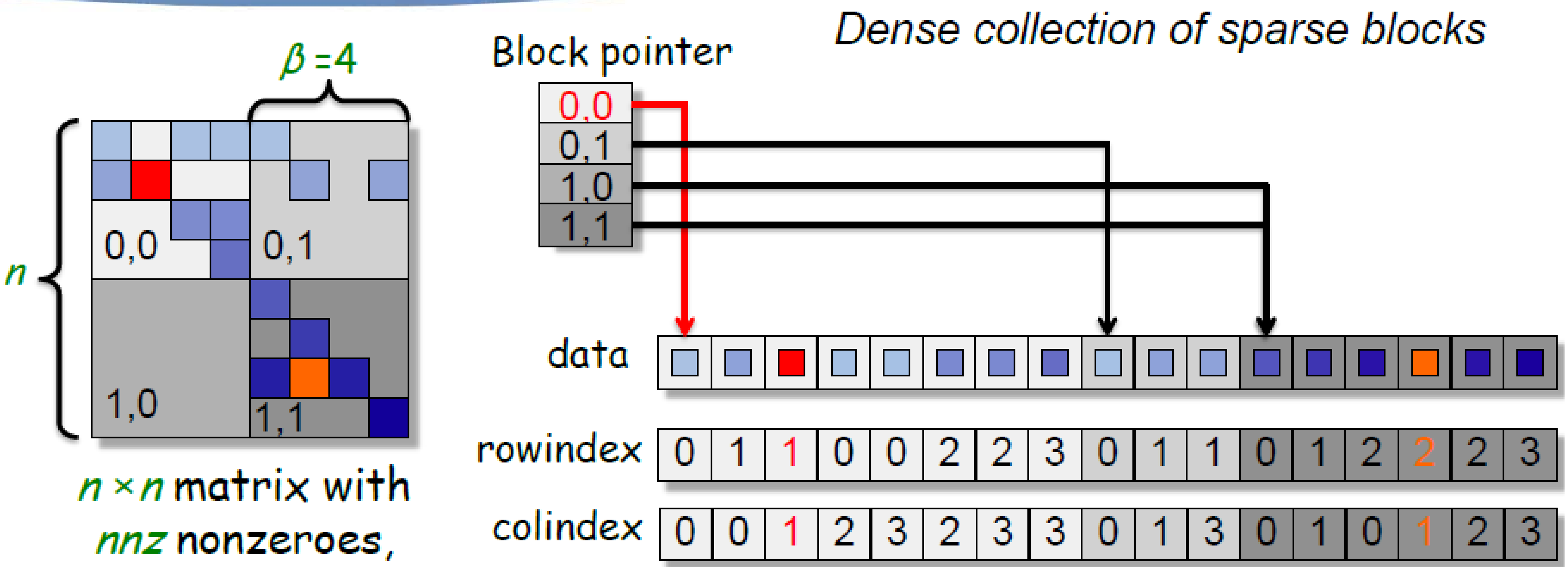


- Scales well to hundreds of processors
- Betweenness centrality benchmark: over 200 MTEPS
- Experiments: TACC Lonestar cluster

Time vs Number of cores -- 1M-vertex RMAT

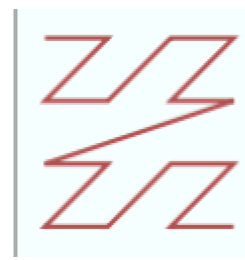
CSB: Compressed sparse block storage

[Buluc, Fineman, Frigo, G, Leiserson]



n × *n* matrix with *nnz* nonzeroes, in *β* × *β* blocks

- Dense 2-D array of sparse blocks
- Blocks stored in row-major order
- Entries within blocks stored in Z-morton order
- Storage = 1 word / edge + 1 word / vertex

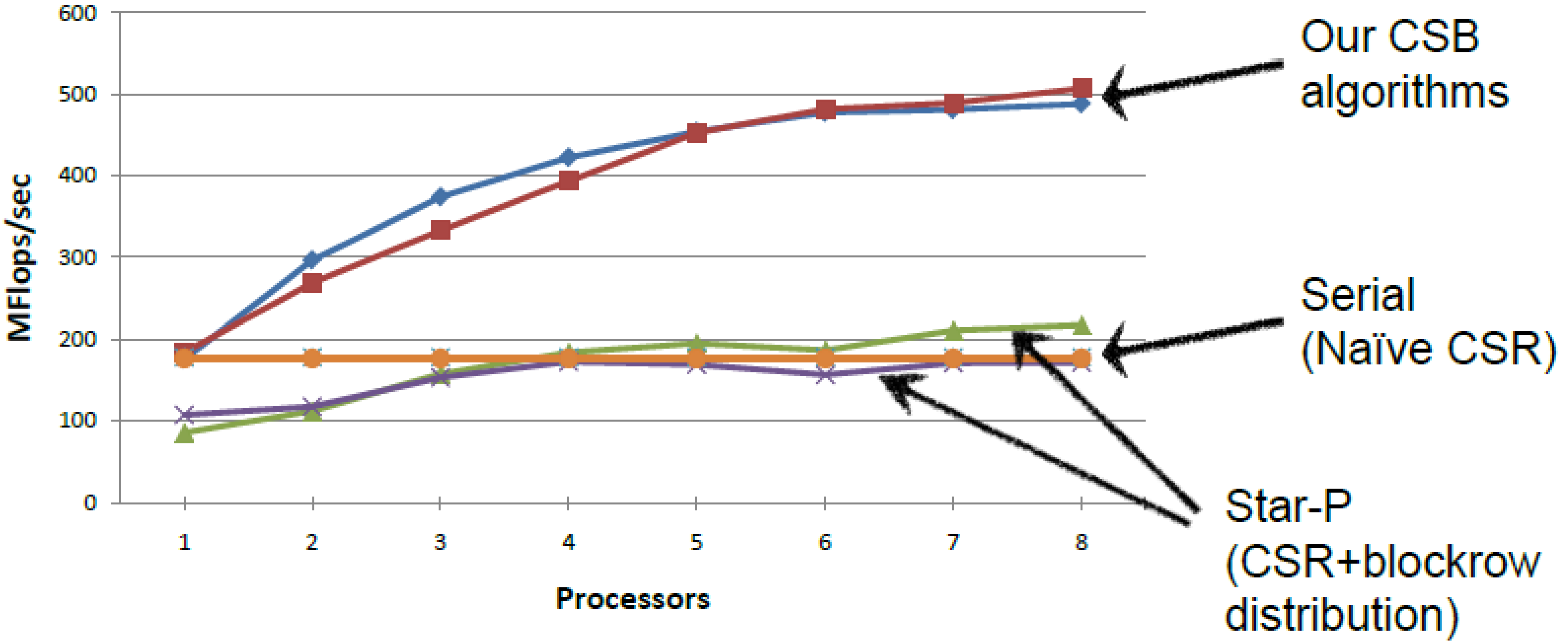


UCSB

CSB for parallel Ax and $A^T x$

[Buluc, Fineman, Frigo, G, Leiserson]

- Efficient multiplication of a sparse matrix *and its transpose* by a vector
- Compressed sparse block storage
- Critical path never more than ~ sqrt(n)*log(n)
- Multicore / multisocket architectures



From Semirings to Computational Patterns

Matrices over semirings

- Matrix multiplication **C = AB** (or matrix/vector):

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

- Replace scalar operations $\times$ and $+$ by

  $\otimes$: associative, distributes over $\oplus$, identity 1

  $\oplus$: associative, commutative, identity 0 annihilates under $\otimes$

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

- Examples: $(\times,+)$; (and,or) ; $(+,\min)$; . . .

- No change to data reference pattern or control flow

From semirings to computational patterns

Sparse matrix times vector as a semiring operation:

- Given vertex data $\mathbf{x_i}$ and edge data $\mathbf{a_{i,j}}$
- For each vertex j of interest, compute

$$\mathbf{y_j = a_{i_1,j} \otimes x_{i_1} \oplus a_{i_2,j} \otimes x_{i_2} \oplus \cdots \oplus a_{i_k,j} \otimes x_{i_k}}$$

- User specifies: definition of operations $\otimes$ and $\oplus$

UCSB

From semirings to computational patterns

<u>Sparse matrix times vector as a computational pattern:</u>

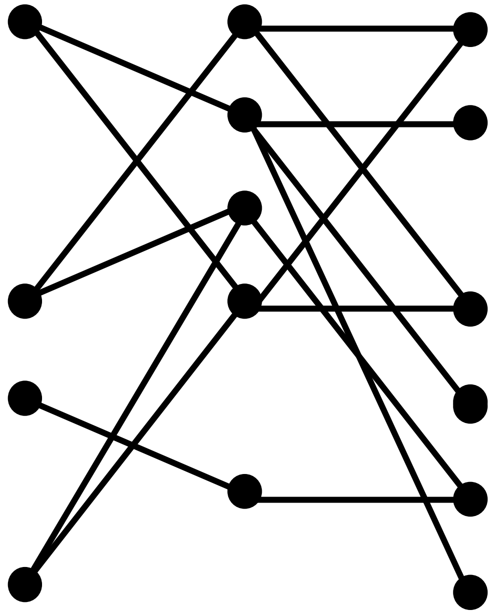
– Given vertex data and edge data

– For each vertex of interest, combine data from neighboring vertices and edges

– User specifies: desired computation on data from neighbors

SpGEMM as a computational pattern

- Explore length-two paths that use specified vertices
- Possibly do some filtering, accumulation, or other computation with vertex and edge attributes
- E.g. “friends of friends” (per Lars Backstrom)
- May or may not want to form the product graph explicitly
- Formulation as semiring matrix multiplication is often possible but sometimes clumsy
- Same data flow and communication patterns as in SpGEMM



UCSB

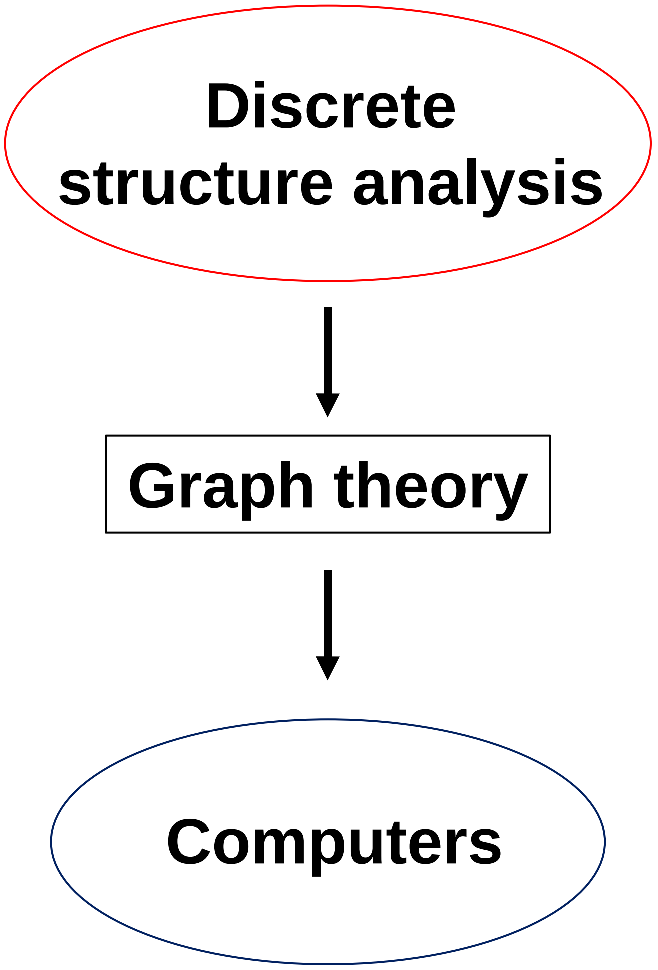
Graph BLAS: A pattern-based library

- User-specified operations and attributes give the performance benefits of algebraic primitives with a more intuitive and flexible interface.
- Common framework integrates algebraic (edge-based), visitor (traversal-based), and map-reduce patterns.
- 2D compressed sparse block structure supports user-defined edge/vertex/attribute types and operations.
- “Hypersparse” kernels tuned to reduce data movement.
- Initial target: manycore and multisocket shared memory.

UCSB

Challenge: Complete the analogy . . .

- √ Mathematical tools
- ? "Impedance match" to computer operations
- ? High-level primitives
- ? High-quality software libs
- ? Ways to extract performance from computer architecture
- ? Interactive environments



UCSB

Some challenges

- Fault tolerance
- Uncertainty and probabilistic attributes
- Fine-grained dynamic updates
- Interactive environments for exploration & productivity
- Hardware architecture

UCSB