

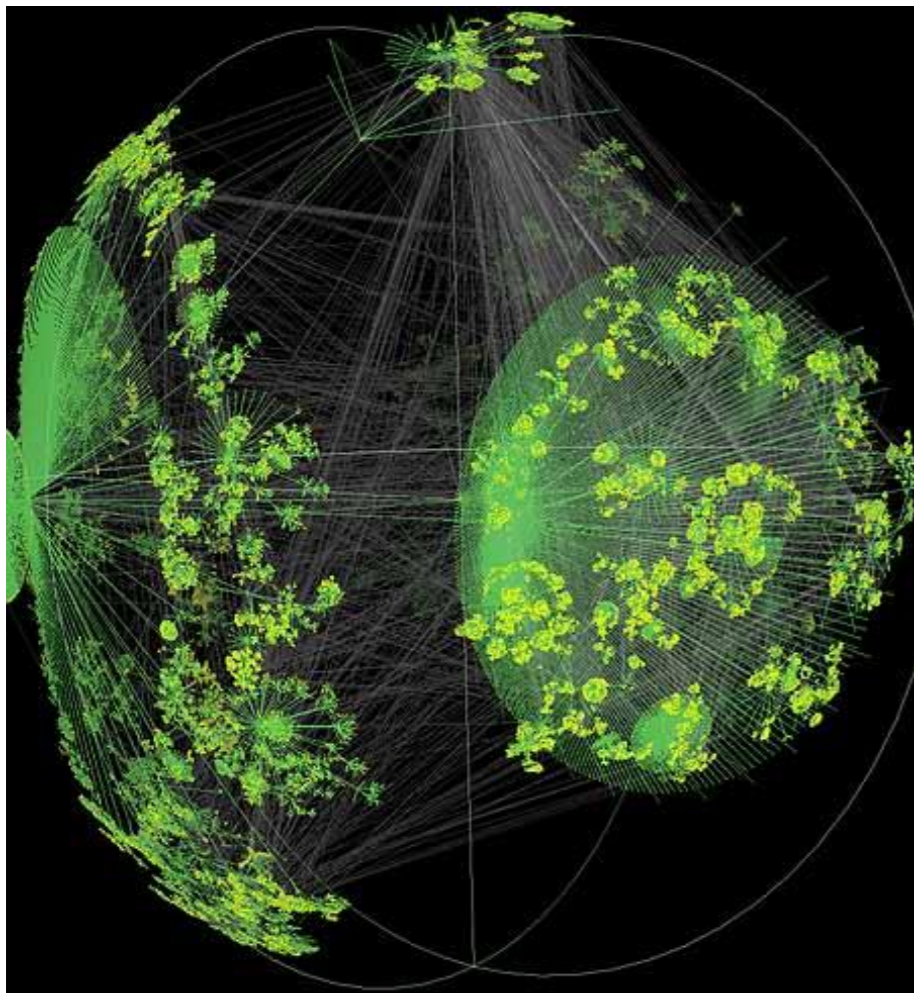


Building Blocks for Scalable Graph and Data Mining Software

Aydın Buluç, John R. Gilbert
University of California, Santa Barbara
SIAM PP'10

Sources of Massive Graphs

Graphs naturally arise from the internet and social interactions



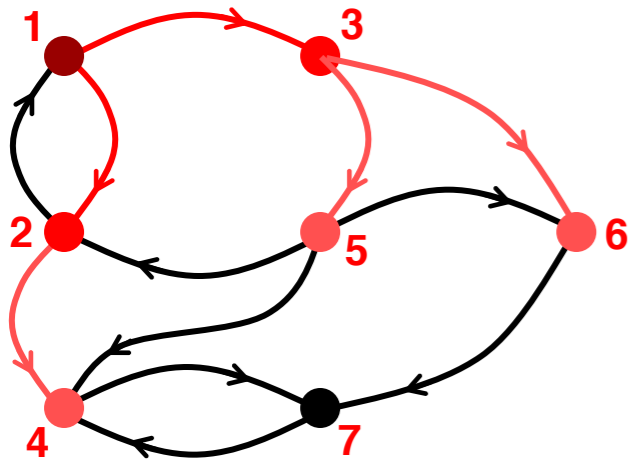
(WWW snapshot, courtesy Y. Hyun)

Many scientific (biological, chemical, cosmological, ecological, etc) datasets are modeled as graphs.



(Yeast protein interaction network, courtesy H. Jeong)

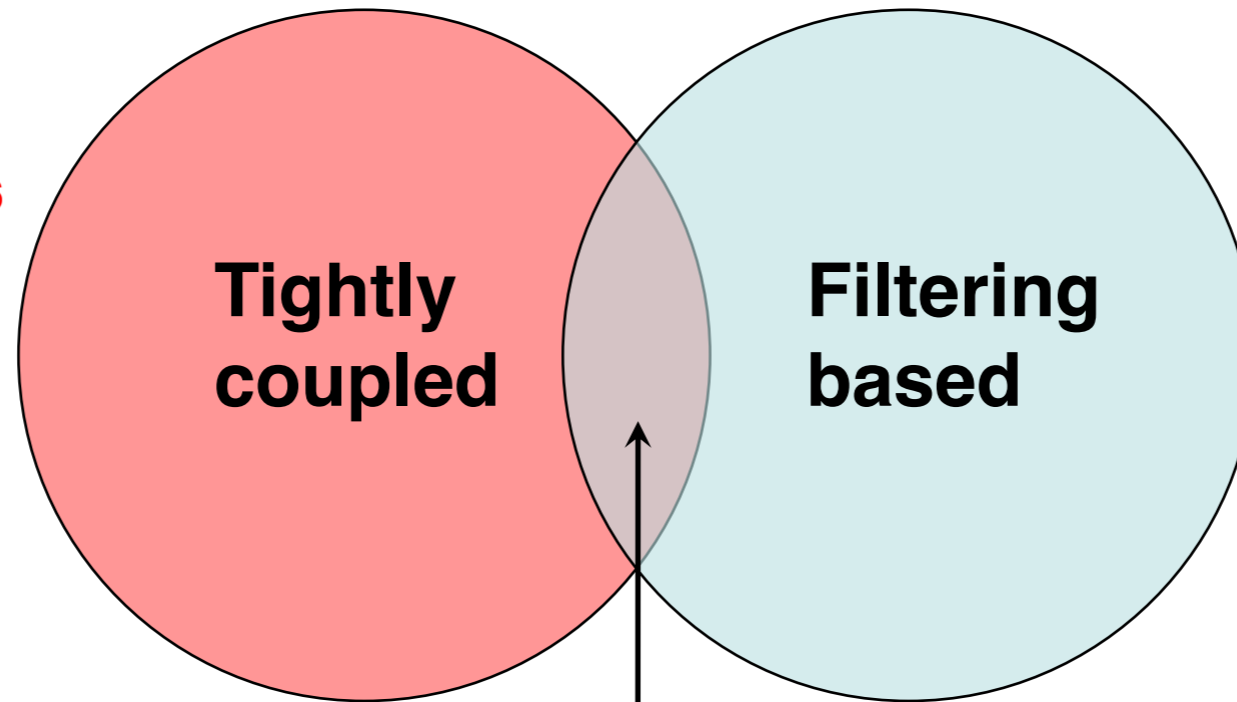
Types of Graph Computations



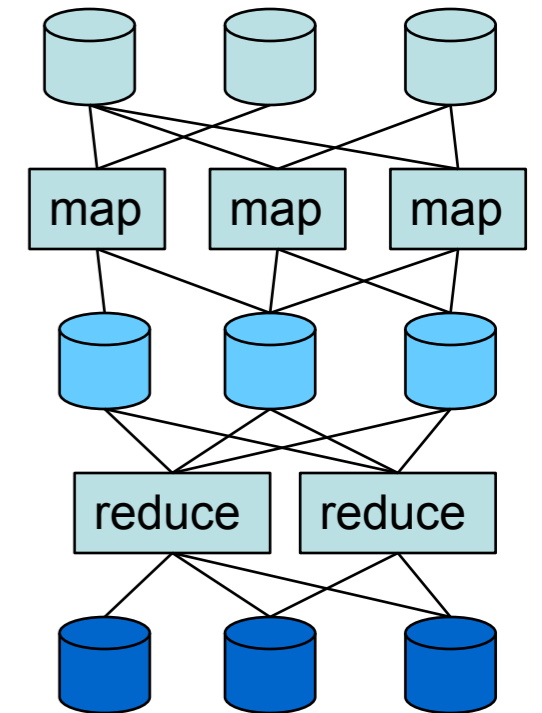
Tool: Graph Traversal

Examples:

- Centrality
- Shortest paths
- Network flows
- Strongly Connected Components



Fuzzy intersection
Examples: Clustering,
Algebraic Multigrid



Tool: Map/Reduce

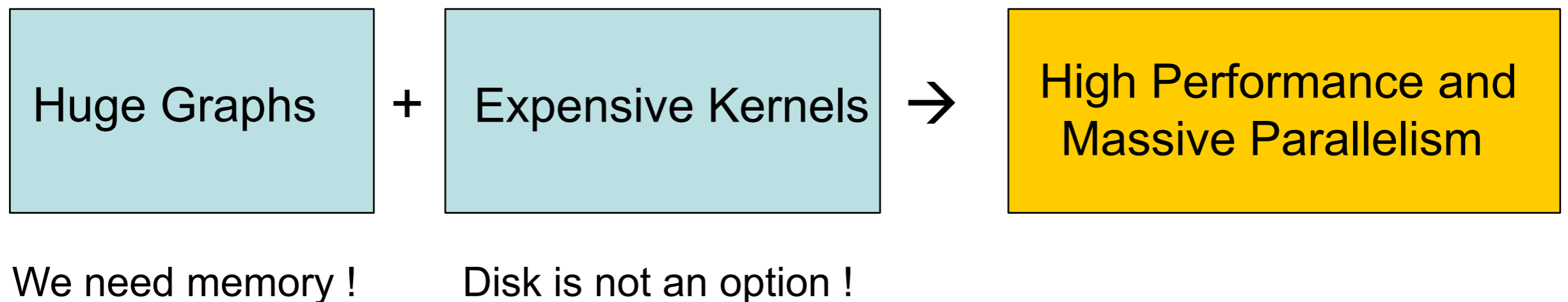
Examples:

- Loop and multi edge removal
- Triangle/Rectangle enumeration

Tightly Coupled Computations

- A. Computationally intensive graph/data mining algorithms.
(e.g. graph clustering, centrality, dimensionality reduction)
- B. Inherently latency-bound computations.
(e.g. finding s-t connectivity)

Let XMT handle (B),
but what can we do for (A) with commodity architectures?

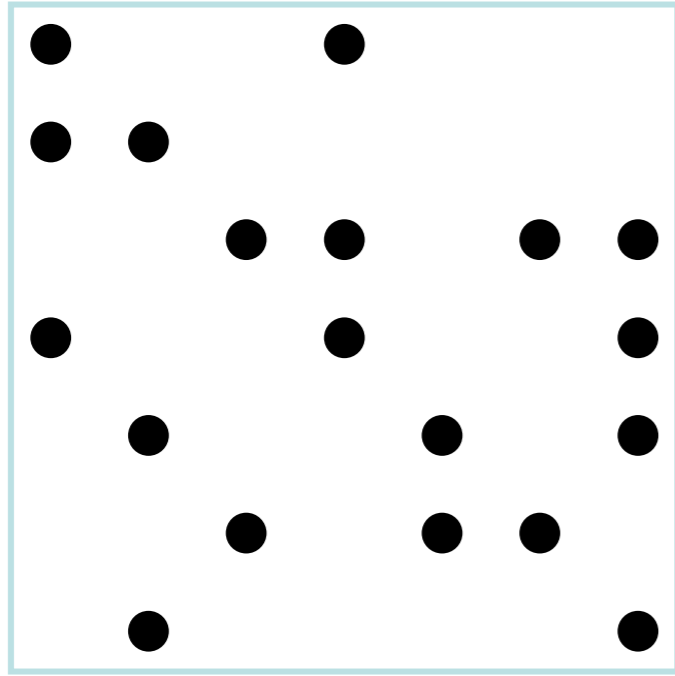


Parallel Graph Software

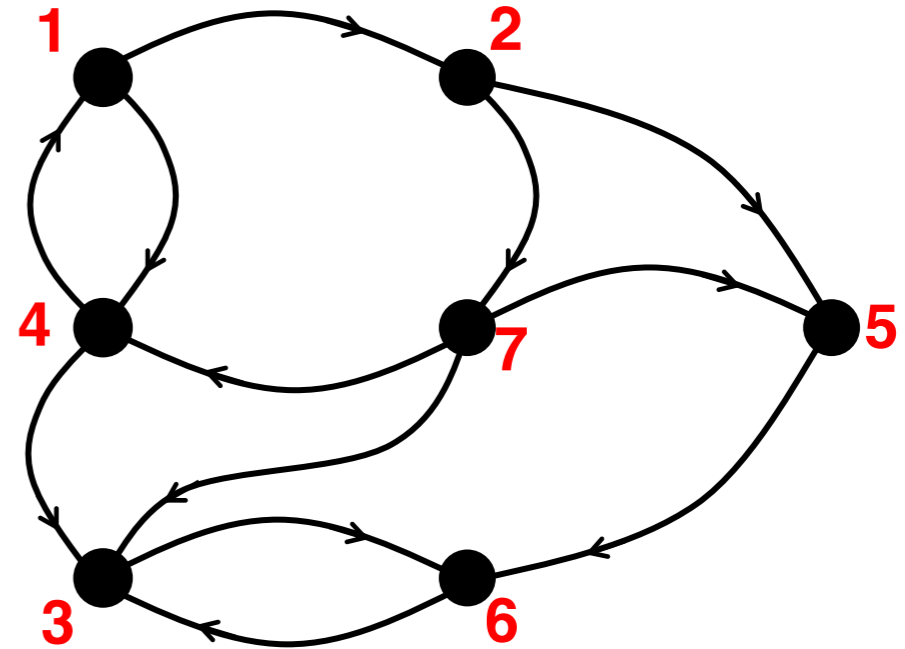
Packet	Parallelism	Abstraction	Offering	Scalability
PBGL	Distributed	Visitor	Algorithms	Limited
GAPDT	Distributed	Sparse Matrix	Both	Limited
MTGL	Shared	Visitor	Algorithms	Unknown
SNAP	Shared	Various	Both	Good
Combinatorial BLAS	Distributed	Sparse Matrix	Kernels	Good

- Shared memory scales, so it pays off to target productivity
- Distributed memory should be targeting $p > 100$ at least
- Shared and distributed memory complement each other.
- Coarse-grained parallelism for distributed memory.

Sparse Adjacency Matrix and Graph



A^T



- Every graph is a sparse matrix and vice-versa
- Adjacency matrix: sparse array w/ nonzeros for graph edges
- Storage-efficient implementation from sparse data structures

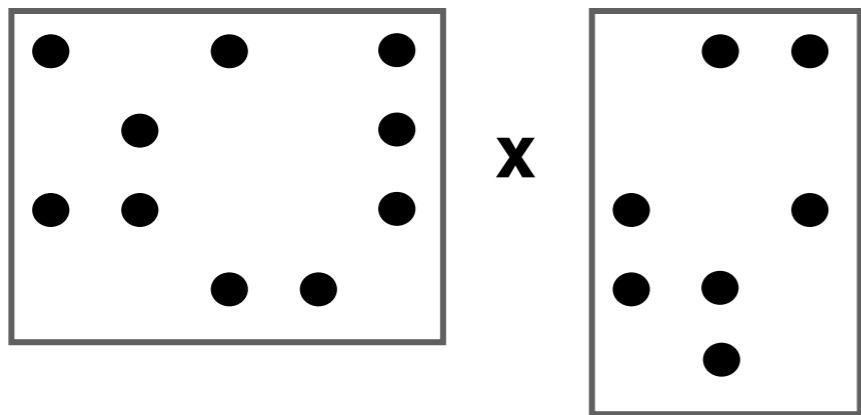
The Case for Sparse Matrices

- Many irregular applications contain sufficient coarse-grained parallelism that can ONLY be exploited using abstractions at proper level.

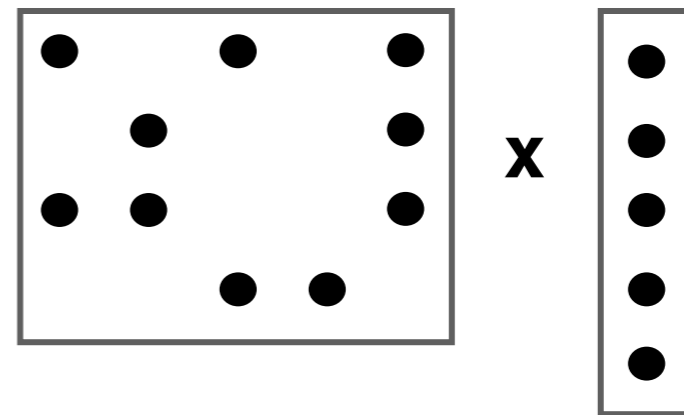
Traditional graph computations	Graphs in the language of linear algebra
Data driven. Unpredictable communication.	Fixed communication patterns.
Irregular and unstructured. Poor locality of reference	Operations on matrix blocks. Exploits memory hierarchy
Fine grained data accesses. Dominated by latency	Coarse grained parallelism. Bandwidth limited

Linear Algebraic Primitives

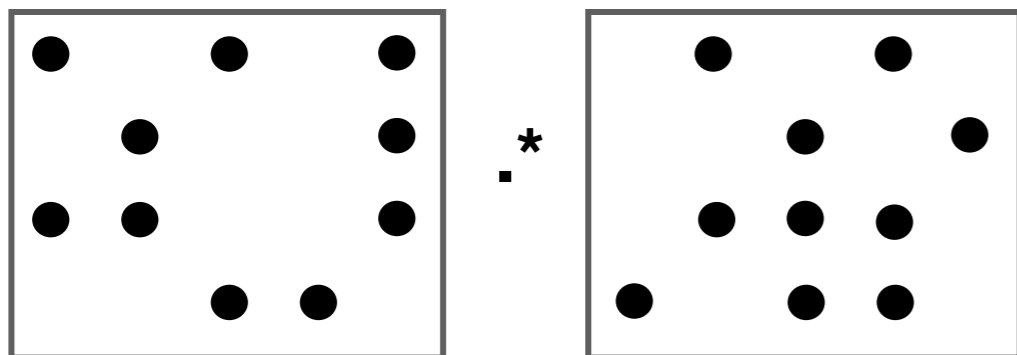
Sparse matrix-matrix
Multiplication (SpGEMM)



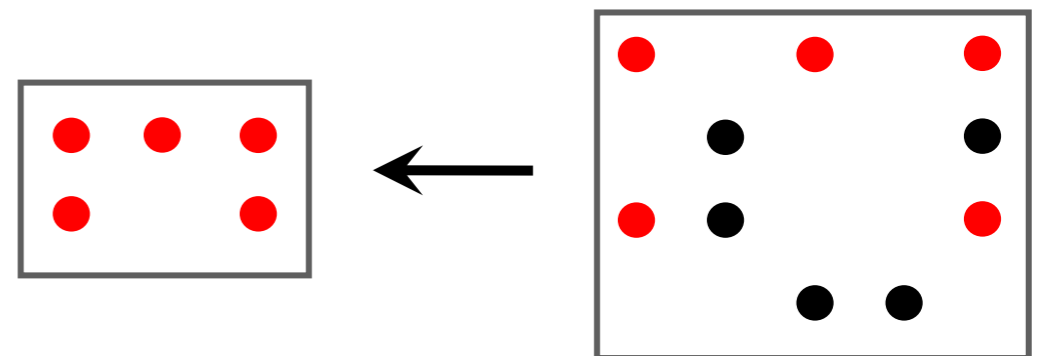
Sparse matrix-vector
multiplication



Element-wise operations



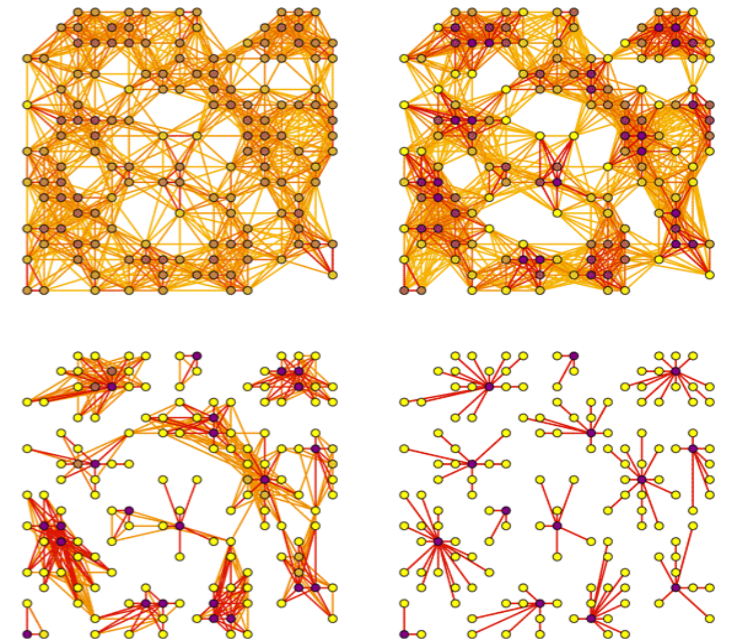
Sparse Matrix Indexing



Matrices on semirings, e.g. $(\cdot, +)$, (and, or), $(+, \min)$

Applications of Sparse GEMM

- *Graph clustering (MCL, 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 ...



Loop until convergence

$A = A * A;$

% expand

$A = A .^ 2;$

% inflate

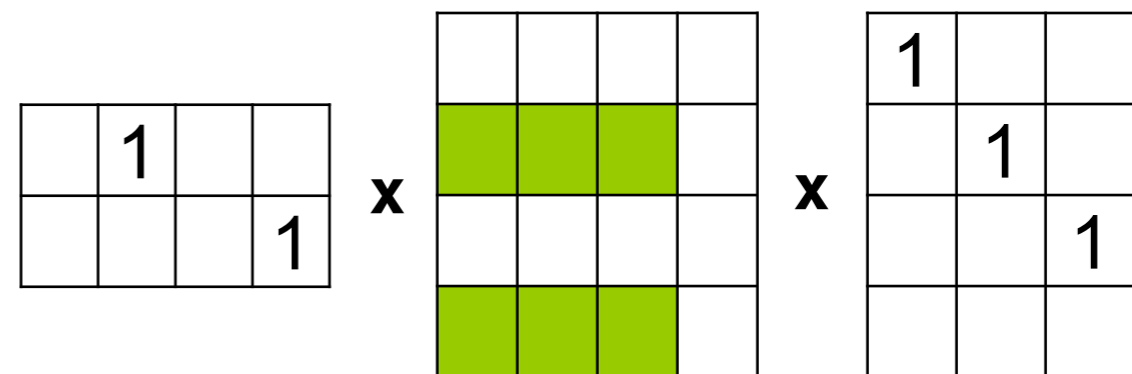
$sc = 1./sum(A, 1);$

$A = A * diag(sc);$

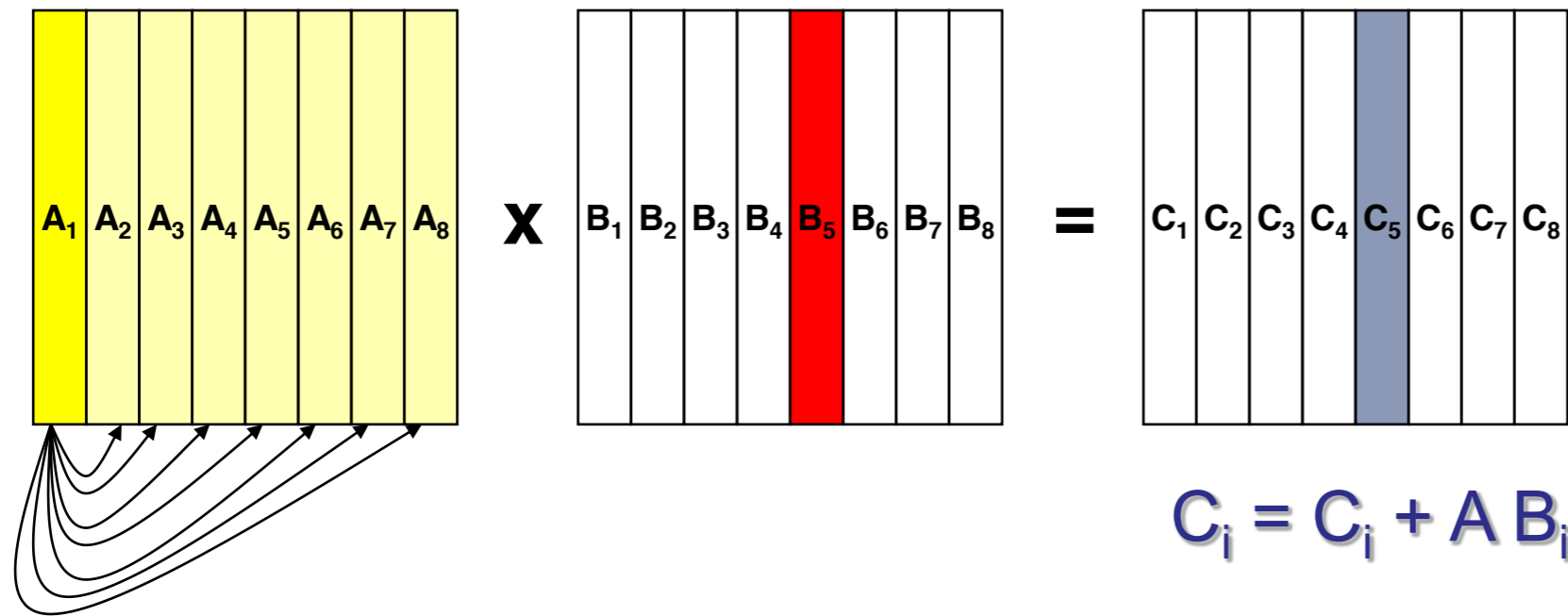
% renormalize

$A(A < 0.0001) = 0;$

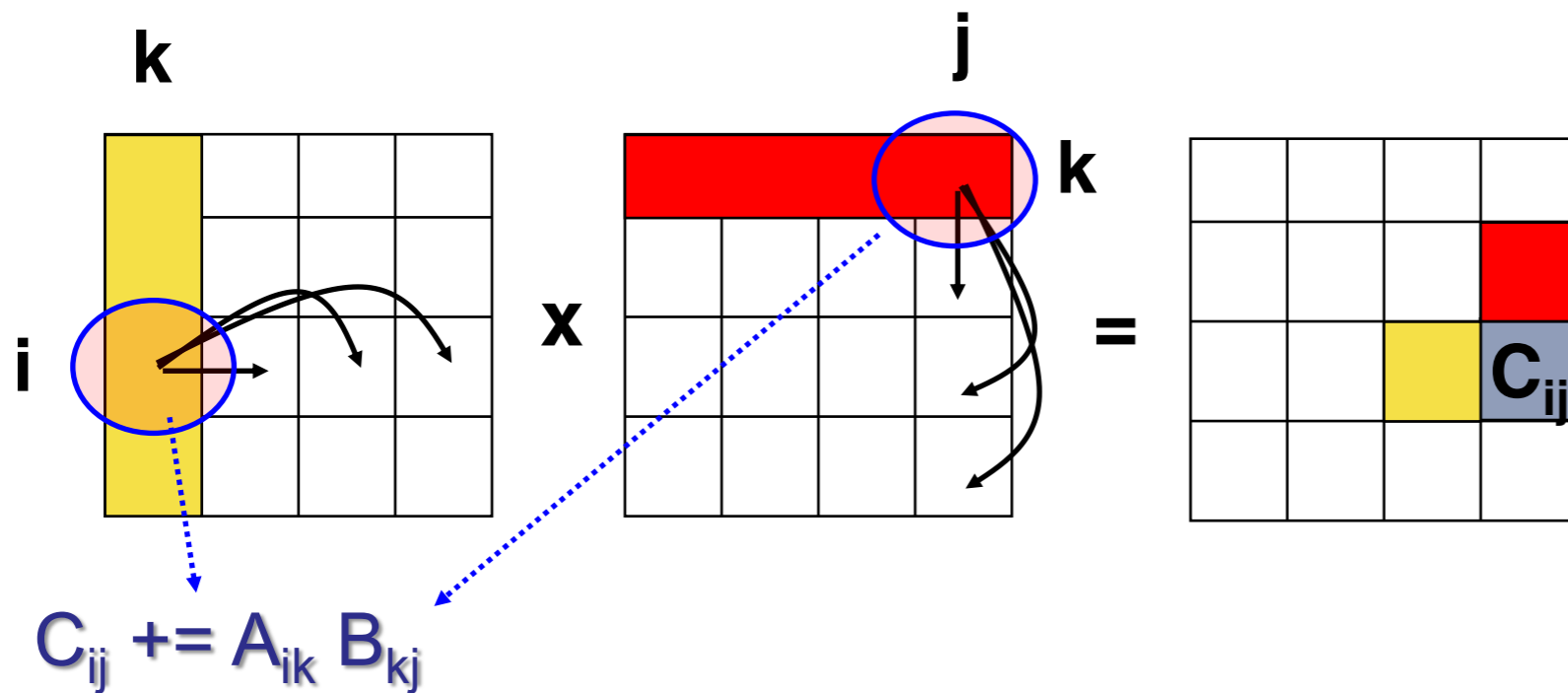
% prune entries



Two Versions of Sparse GEMM



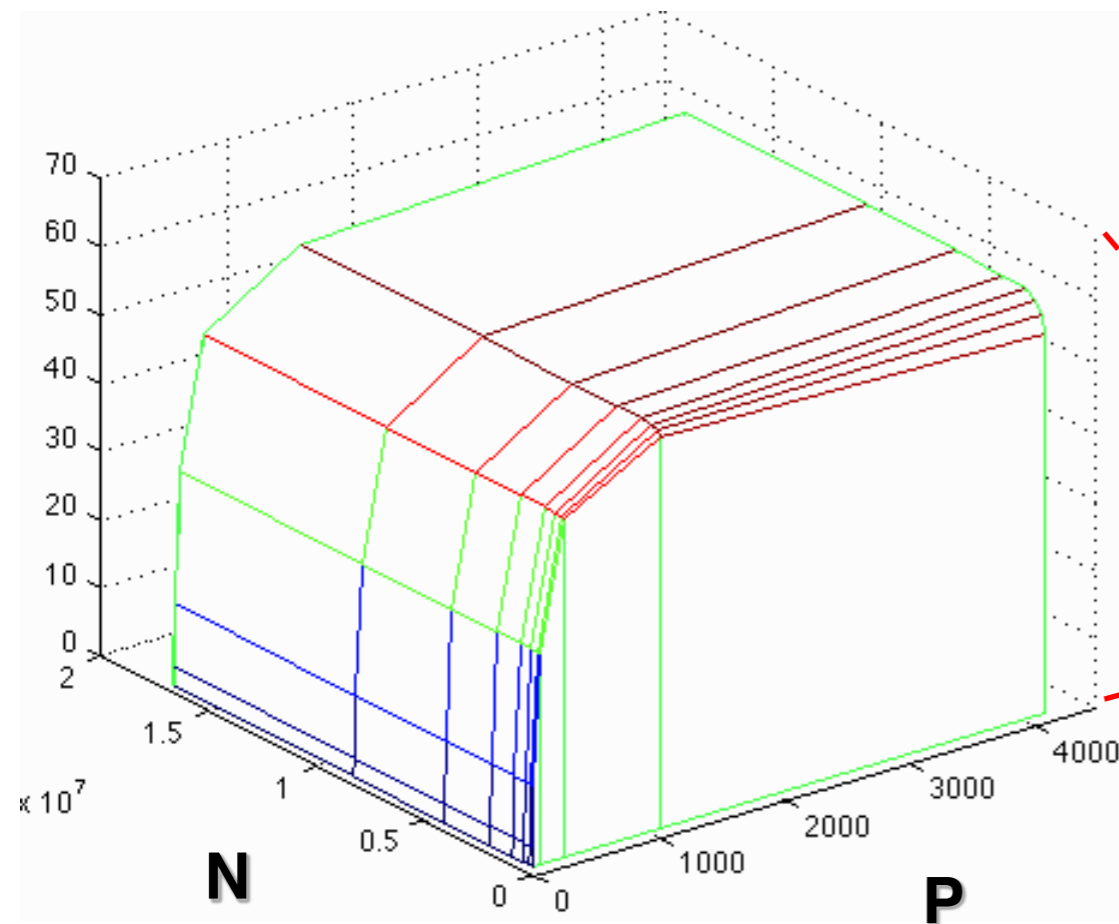
1D block-column distribution



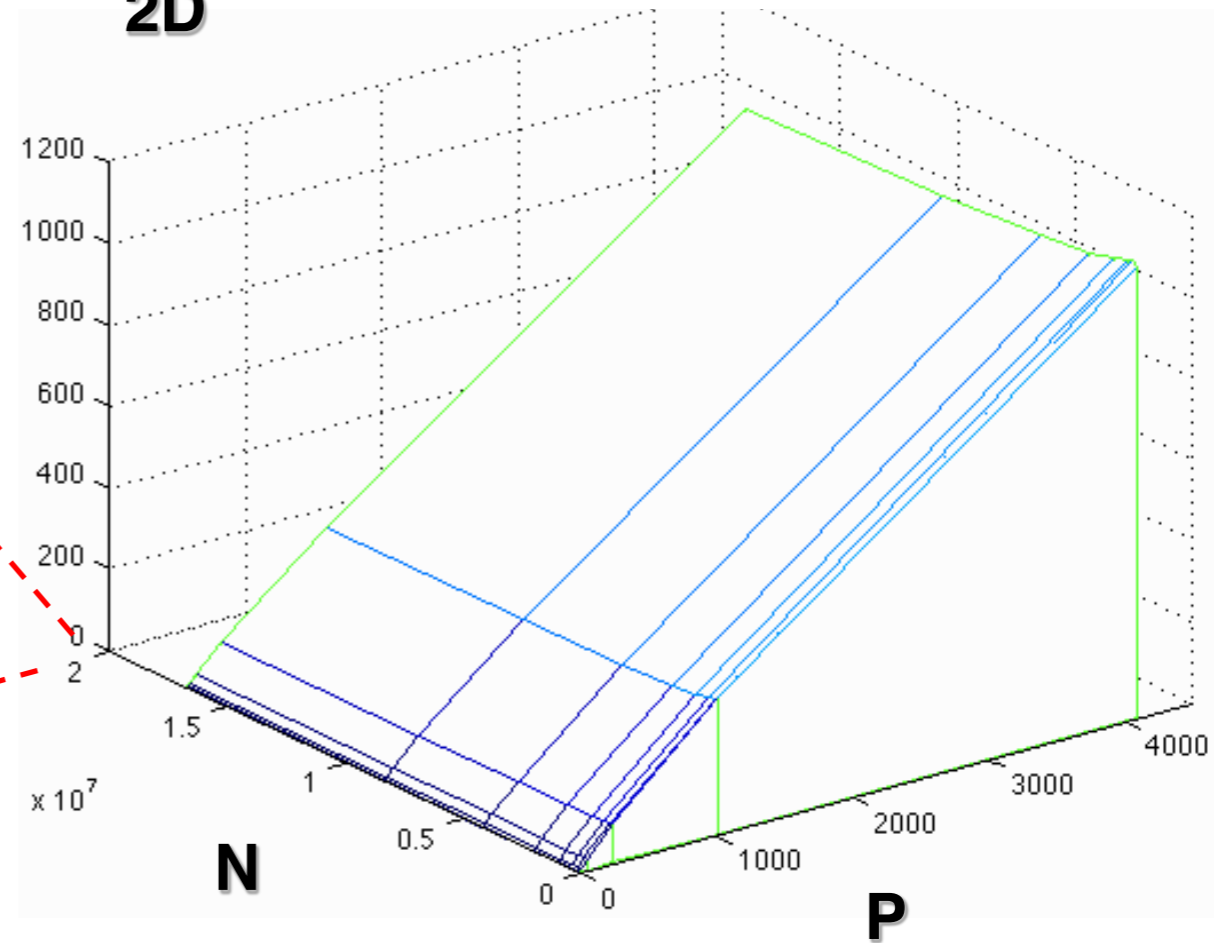
Checkerboard (2D block) distribution

Projected performances of Sparse 1D & 2D

1D



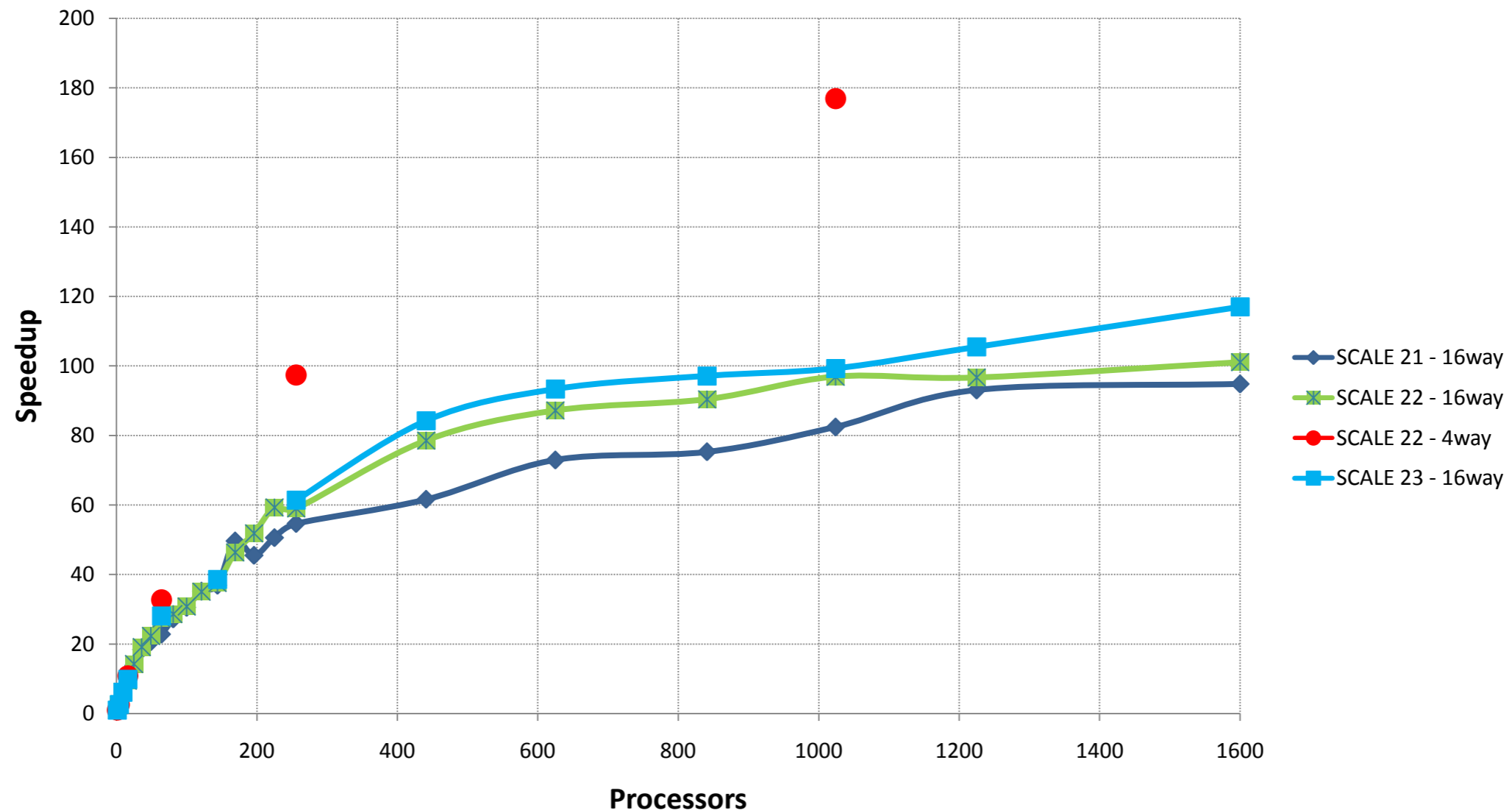
2D



In practice, 2D algorithms have the potential to scale, but not linearly

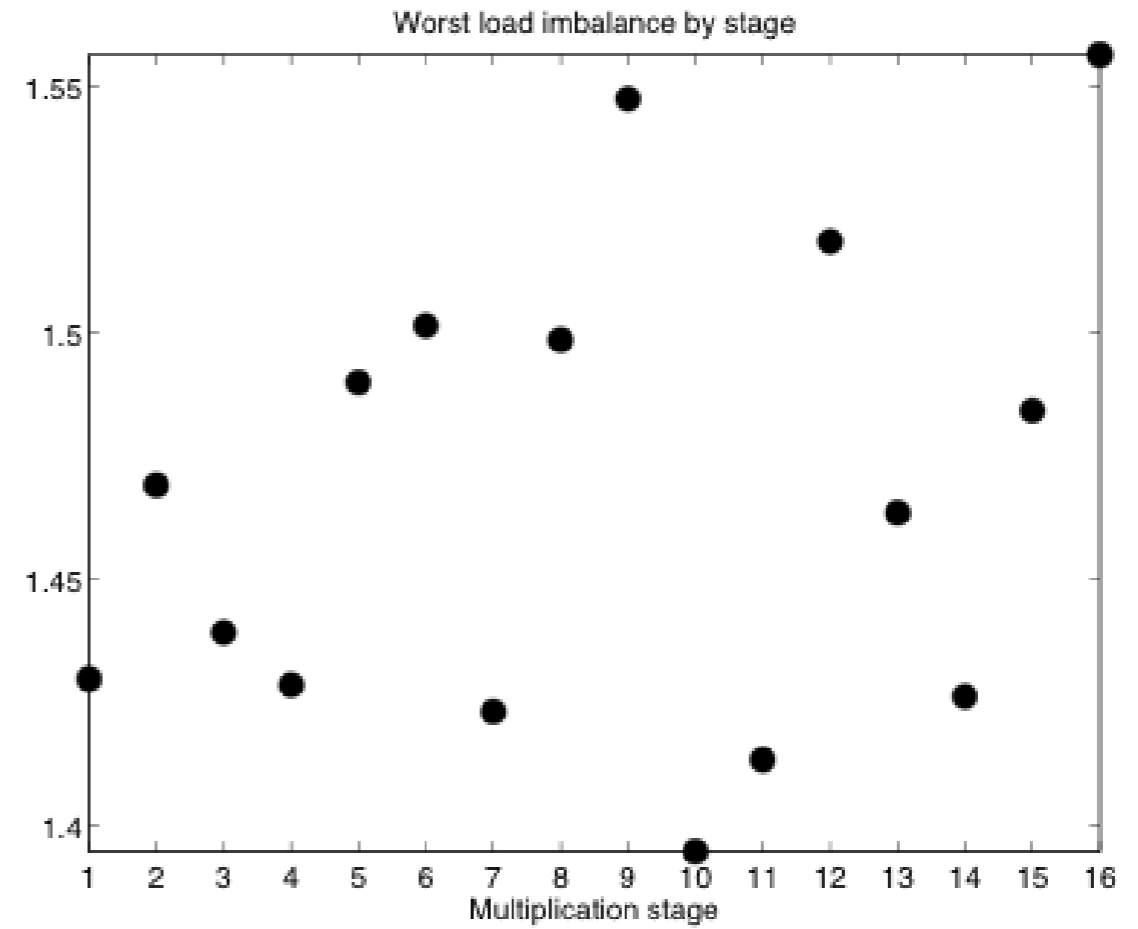
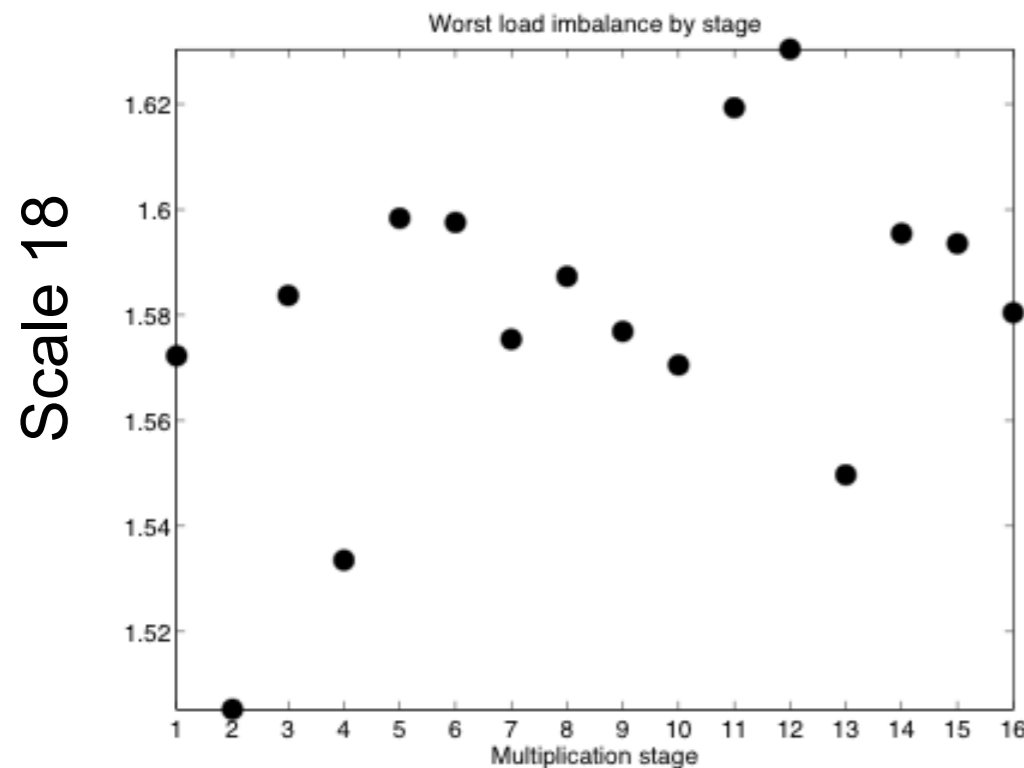
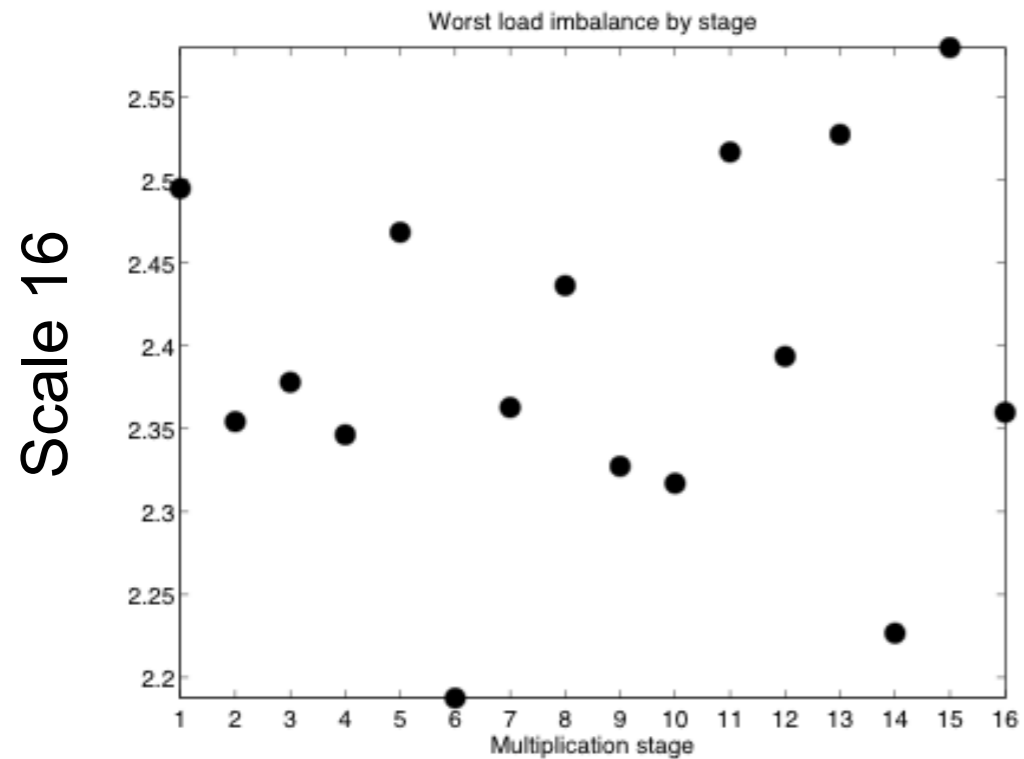
$$E = \frac{W}{p(T_{comp} + T_{comm})} = \frac{\gamma c^2 n}{(\gamma + \beta) cn\sqrt{p} + \gamma c^2 n \lg(c^2 n/p) + \alpha p\sqrt{p}}$$

Scaling Results for SpGEMM



- RMat X RMat product (graphs with high variance on degrees)
- Random permutations useful for the overall computation (<10% imbalance).
- Bulk synchronous algorithms may still suffer due to imbalance **within the stages.** *But this goes away as matrices get larger*

Load Imbalances



(f) RMAT Scale-20

Load imbalances per stage for multiplying two RMAT matrices on $p=256$, using the 2D algorithm

The Combinatorial BLAS

Extendable interface for the right **primitives/abstractions**.

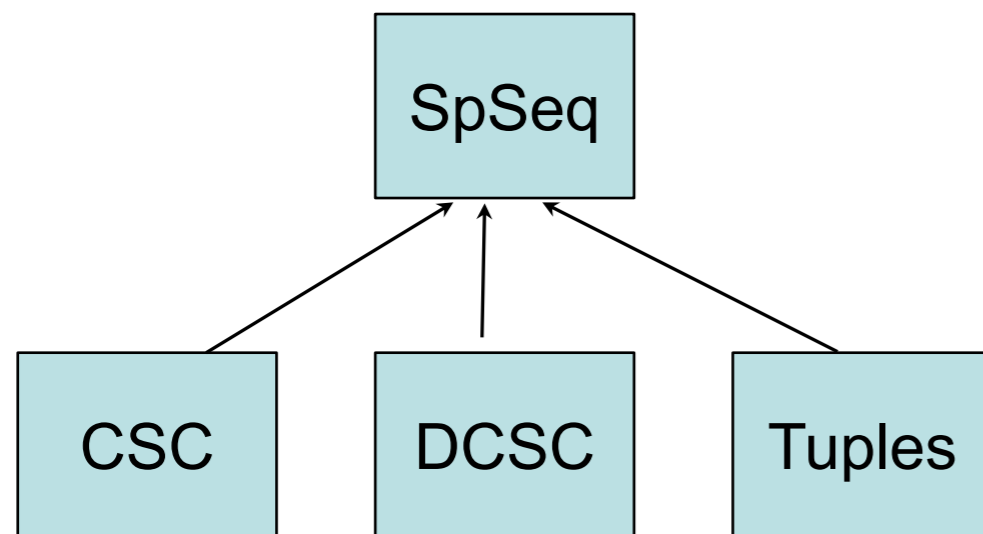
Maintaining compatibility and seamless upgrades.

➔ Decouple parallel logic from the sequential part.

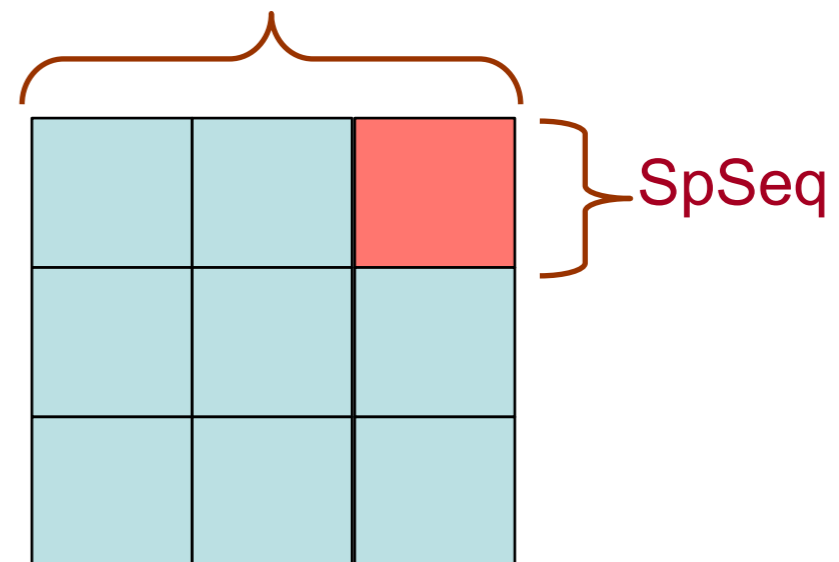
Commonalities:

- Support the sequential API
- Composed of a number of arrays

Any parallel logic:
asynchronous, bulk synchronous, etc

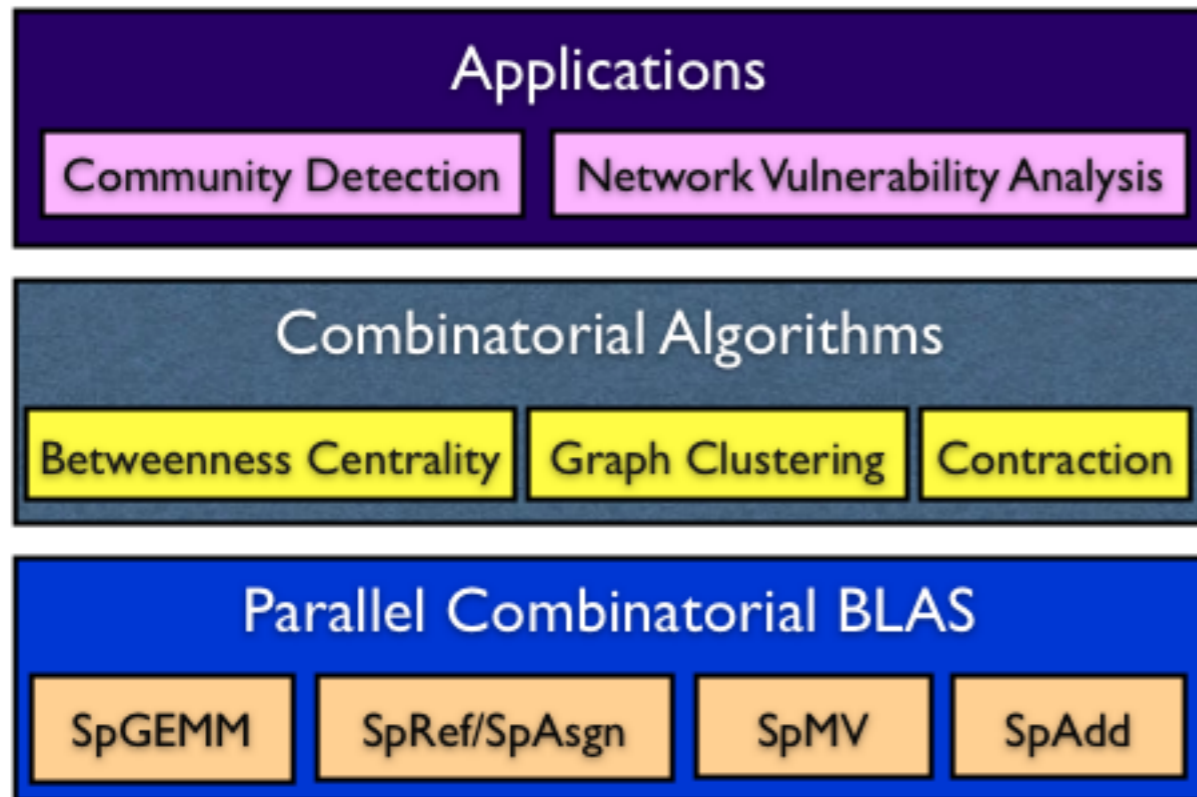


$\text{SpPar}\langle \text{Comm}, \text{SpSeq} \rangle$

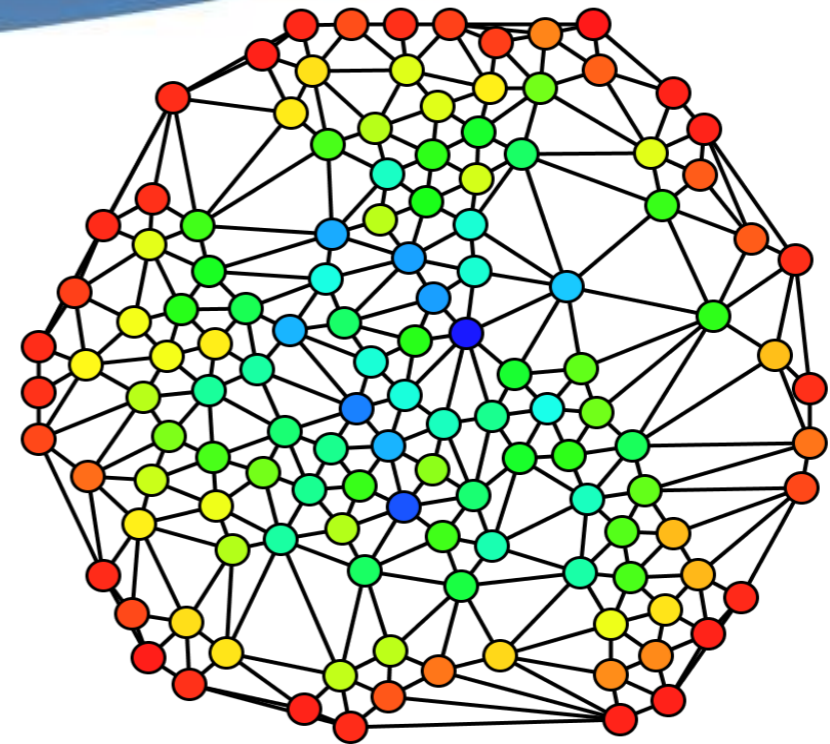


Each **may** support an iterator concept

Social Network Analysis



A typical software stack for an application enabled with the Combinatorial BLAS



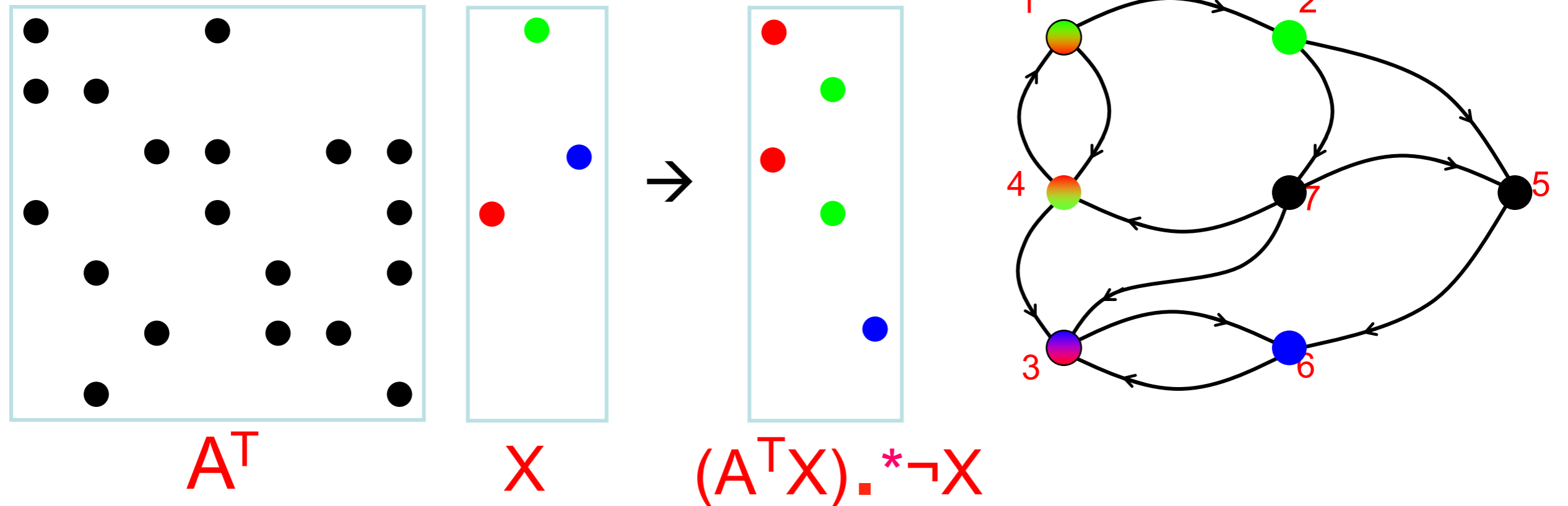
Betweenness Centrality (BC)

$C_B(v)$: Among all the shortest paths, what fraction of them pass through the node of interest?

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

Brandes' algorithm

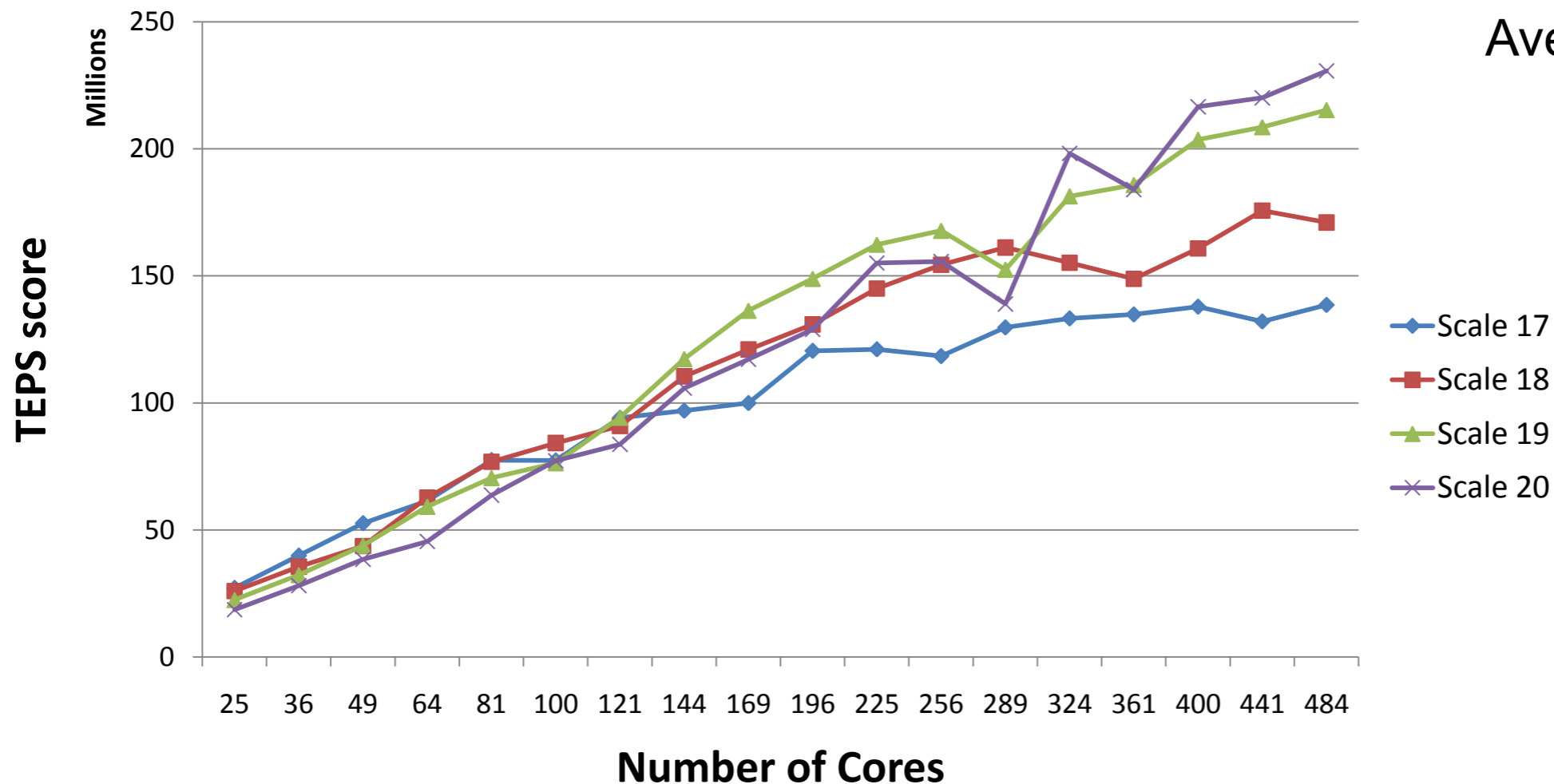
Betweenness Centrality using Sparse GEMM



- Parallel breadth-first search is implemented with sparse matrix-matrix multiplication
- Work efficient algorithm for BC

BC Performance on Distributed-memory

BC performance

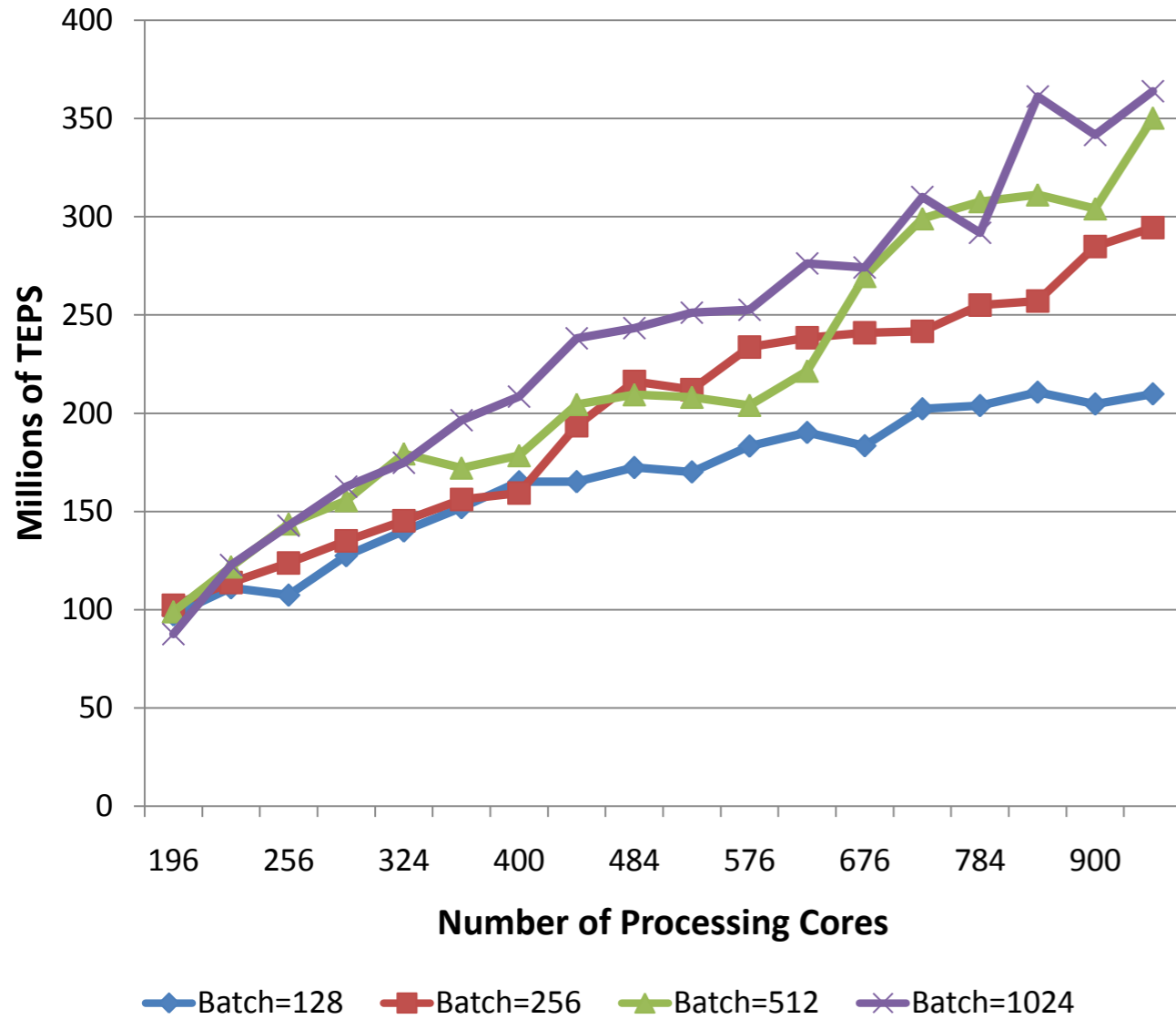


- TEPS: Traversed Edges Per Second
- Batch of 512 vertices at each iteration
- Code only a few lines longer than Matlab version

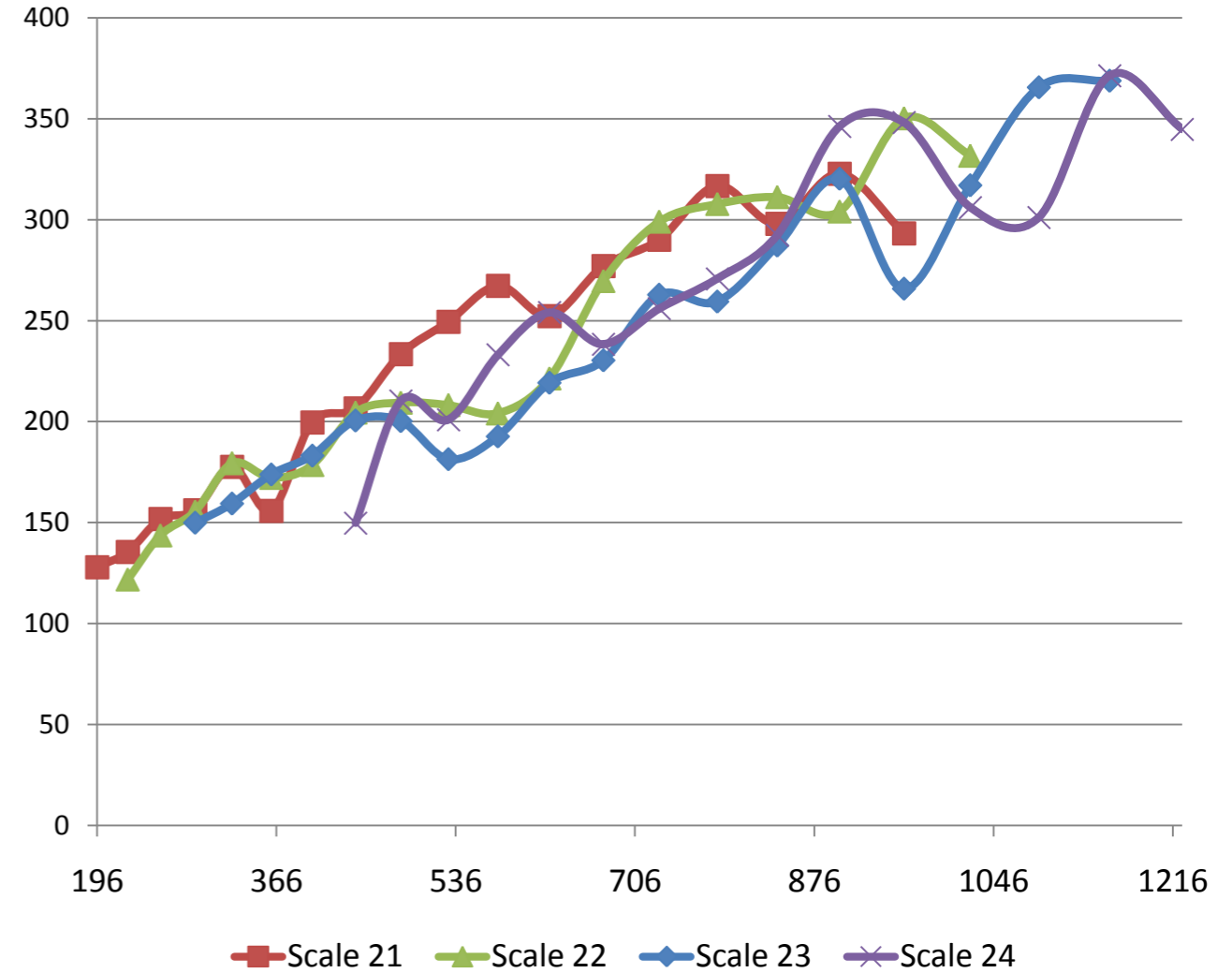
Pure MPI-1 version.
No reliance on any
particular hardware.

More scaling and sensitivity (Betweenness Centrality)

Sensitivity to batch processing



Scaling beyond hundreds



Experiment with 512 batch nodes

Input is an RMat graph of scale 22

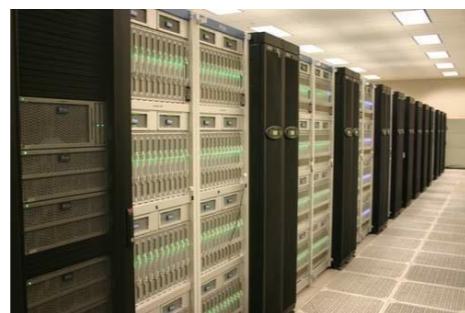
Future Directions

- ▶ Novel scalable algorithms
- ▶ Static graphs are just the beginning.
Dynamic graphs, Hypergraphs, Tensors
- ▶ Architectures (mainly nodes) are evolving
 - Heterogeneous multicores
 - Homogenous multicores with more cores per node

Hierarchical
parallelism



TACC Lonestar (2006)
4 cores / node



TACC Ranger (2008)
16 cores / node



SDSC Triton (2009)
32 cores / node



XYZ Resource (2020)

So long, grad school...

- ▶ What did I learn about parallel graph computations?
- ▶ No silver bullets.
 - ▶ Graph computations are not homogenous.
- ▶ No free lunch.
 - ▶ If communication \gg computation for your algorithm, overlapping them will NOT make it scale.
 - ▶ Scale your algorithm, not your implementation.
- ▶ Any sustainable speedup is a success !
 - ▶ Don't get disappointed with poor parallel efficiency.