Parallel Combinatorial BLAS and Applications in Graph Computations

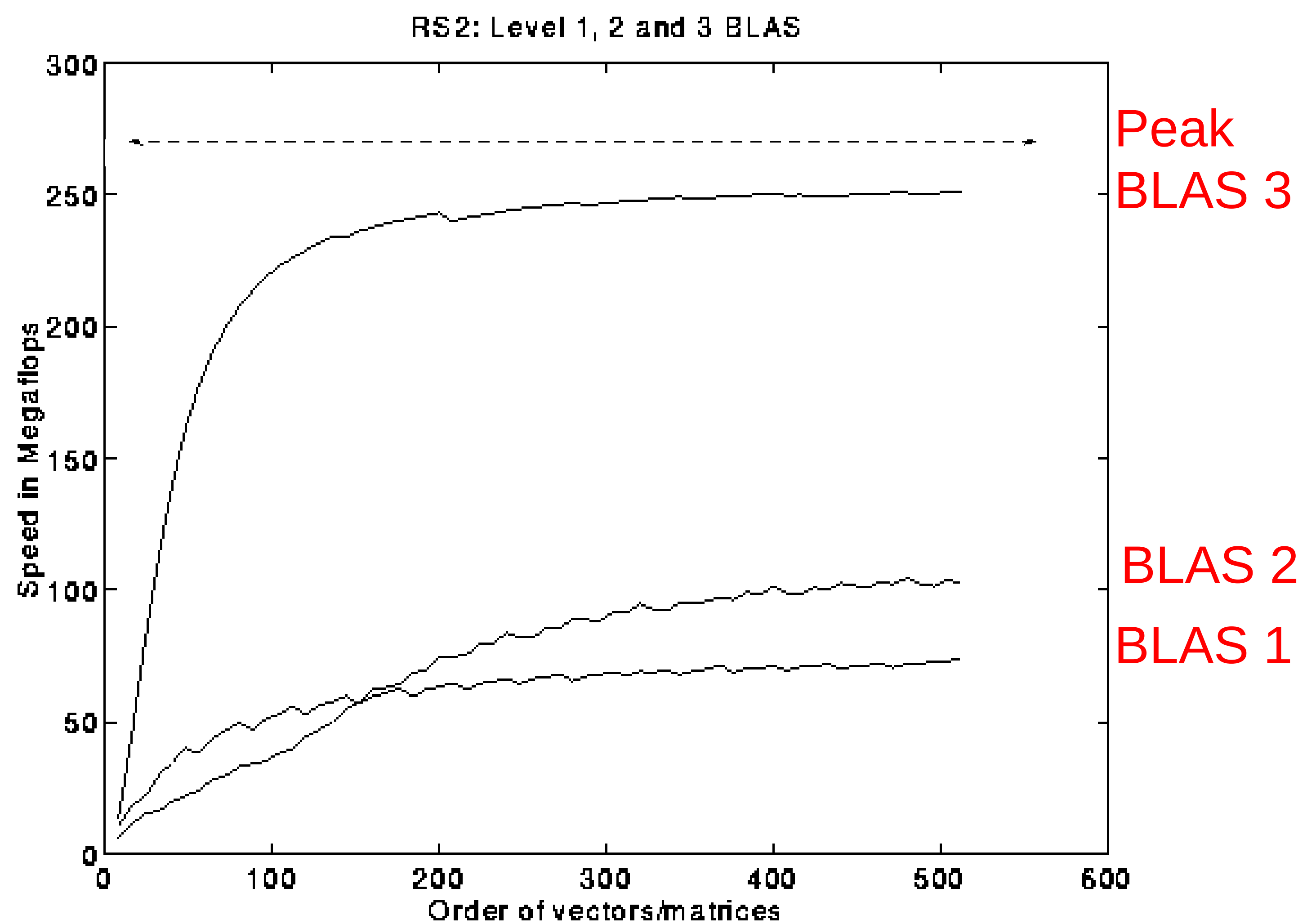
Aydın Buluç
John R. Gilbert
University of California, Santa Barbara

Adapted from talks at SIAM conferences

Primitives for Graph Computations

- By analogy to numerical linear algebra,

- What would the combinatorial BLAS look like?



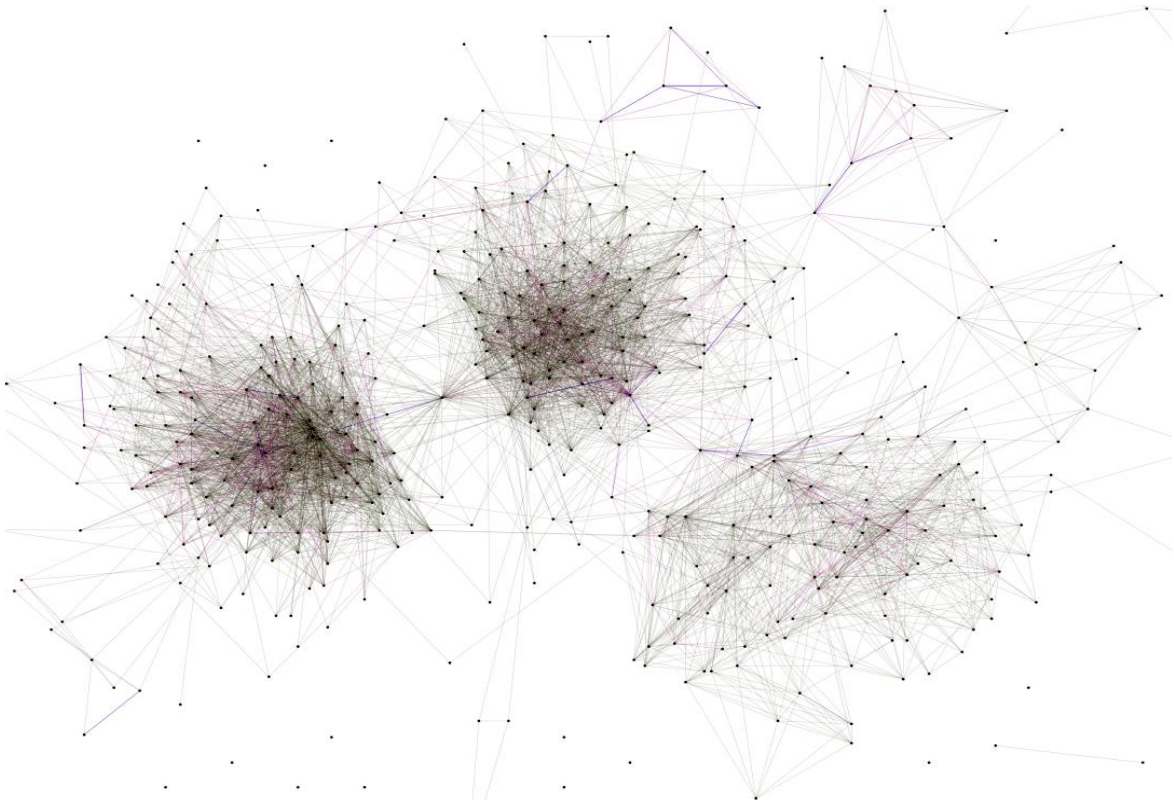
BLAS 3 (n-by-n matrix-matrix multiply)
BLAS 2 (n-by-n matrix-vector multiply)
BLAS 1 (sum of scaled n-vectors)

UCSB

Real-World Graphs

Properties:

- Huge (billions of vertices/edges)
- Very sparse (typically m = O(n))
- Scale-free [maybe]
- Community structure [maybe]



Examples:

- World-wide web
- Science citation graphs
- Online social networks

What Kinds of Computations?

- Some are inherently latency-bound.
  - → S-T connectivity
- Many graph mining algorithms are computationally intensive.
  - → Graph clustering
  - → Centrality computations

Huge Graphs	+	Expensive Kernels	→	Massive Parallelism
Very Sparse Graphs	→	Sparse Data Structures (Matrices)		

The Case for Sparse Matrices

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

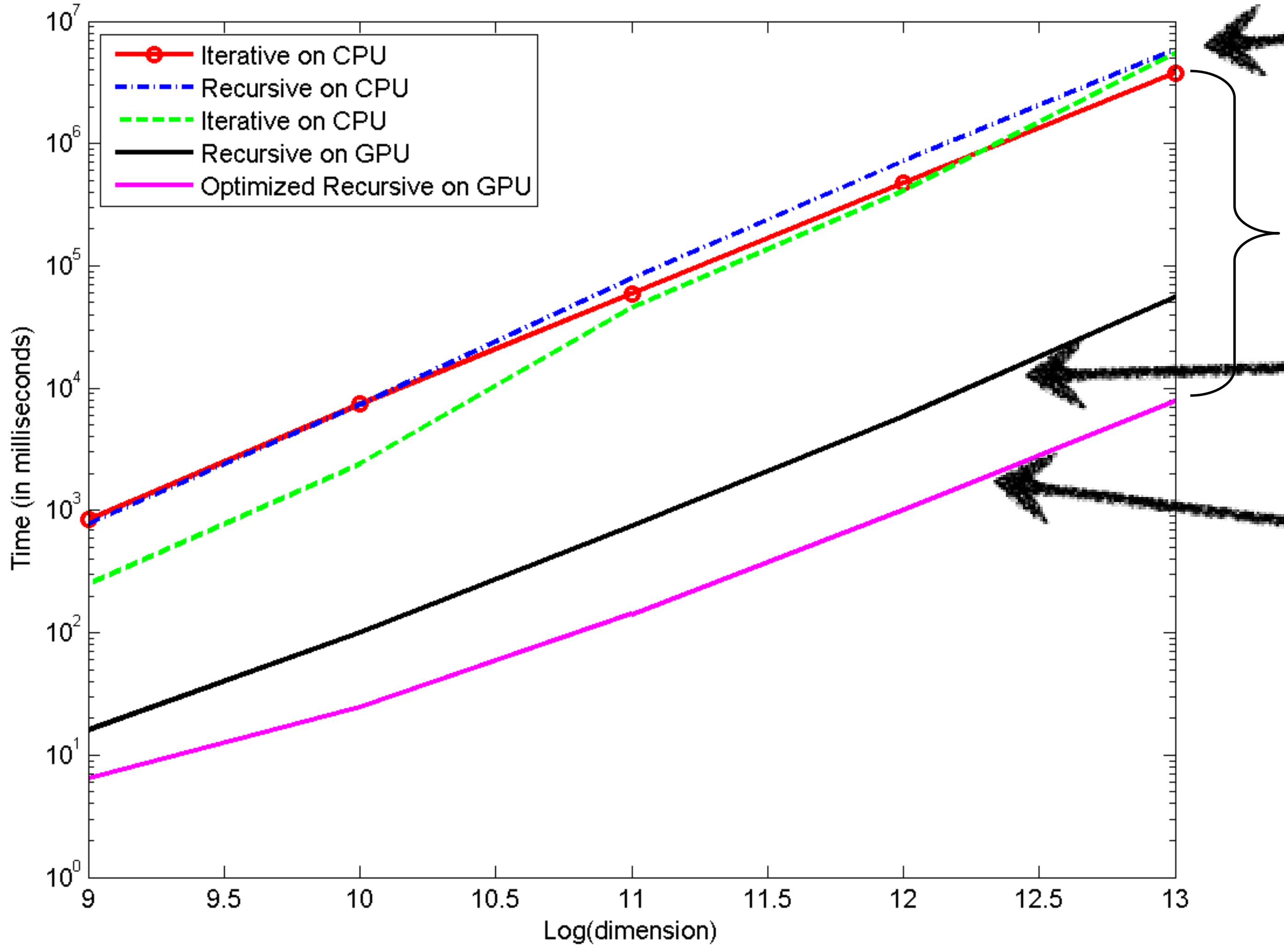
Traditional graph computations	Graphs in the language of linear algebra
Data driven. Unpredictable communication patterns	Fixed communication patterns. Overlapping opportunities
Irregular and unstructured. Poor locality of reference	Operations on matrix blocks. Exploits memory hierarchy
Fine grained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

UCSB

The Case for Primitives

It takes a "certain" level of expertise to get any kind of performance in this jungle of parallel computing

- I think you'll agree with me by the end of the talk :)



What's bandwidth anyway?

480x

I can just implement it (w/ enough coffee)

The right primitive !

All pairs shortest paths on the GPU

UCSB

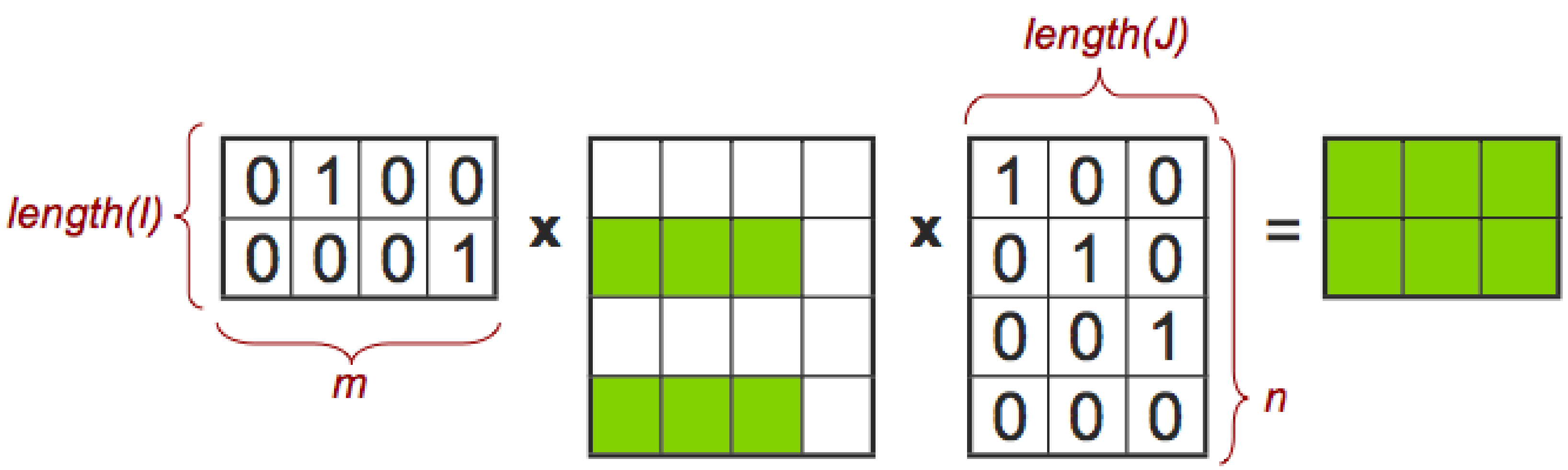
Identification of Primitives

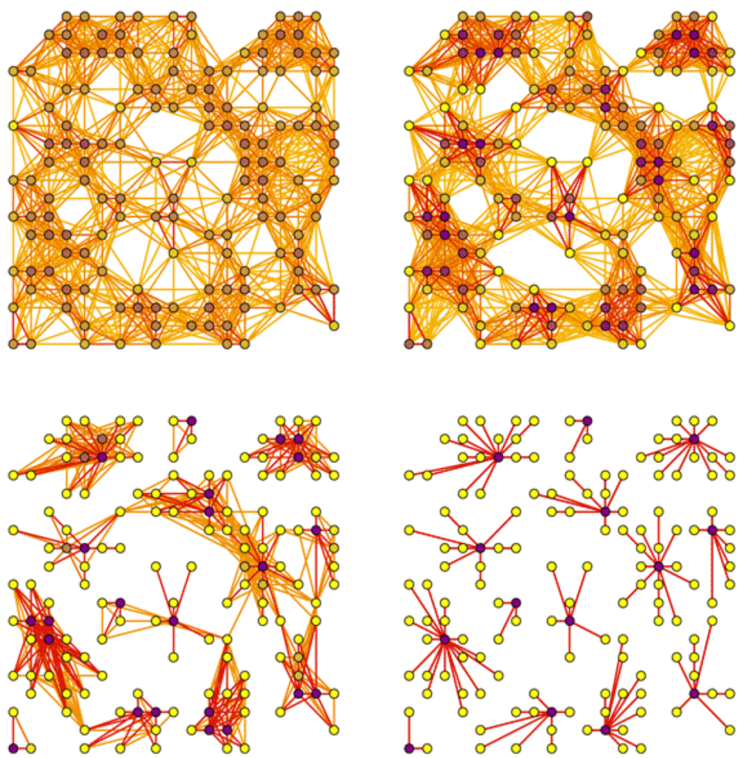
- Sparse matrix-matrix multiplication (SpGEMM)
  - Most general and challenging parallel primitive.
- Sparse matrix-vector multiplication (SpMV)
- Sparse matrix-transpose-vector multiplication (SpMVT)
  - Equivalently, multiplication from the left
- Addition and other point-wise operations (SpAdd)
  - Included in SpGEMM, “proudly” parallel
- Indexing and assignment (SpRef, SpAsgn)
  - A(I,J) where I and J are arrays of indices
  - Reduces to SpGEMM

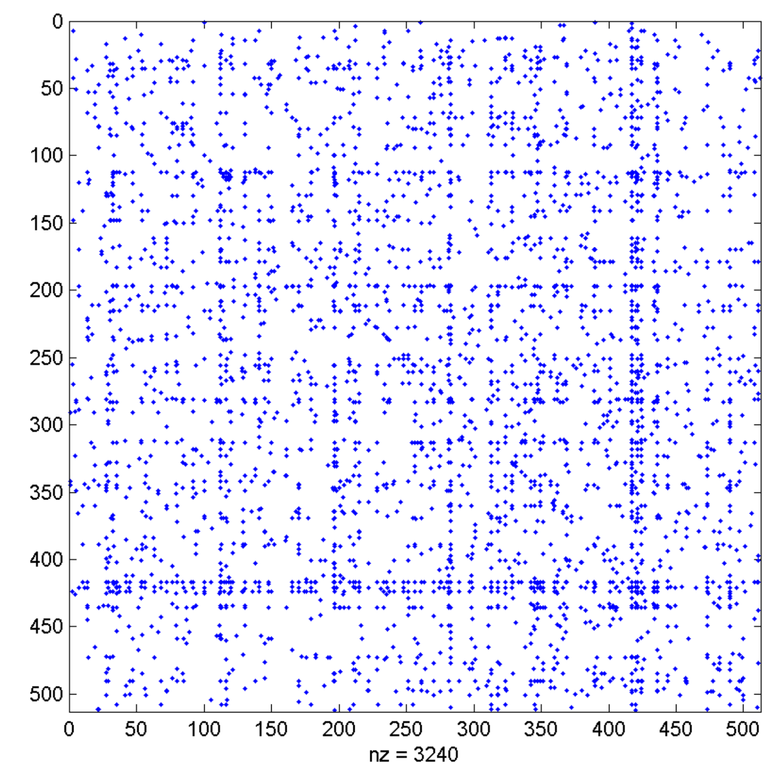
Matrices on semirings, e.g. (·, +), (and, or), (+, min)

UCSB

Why focus on SpGEMM?

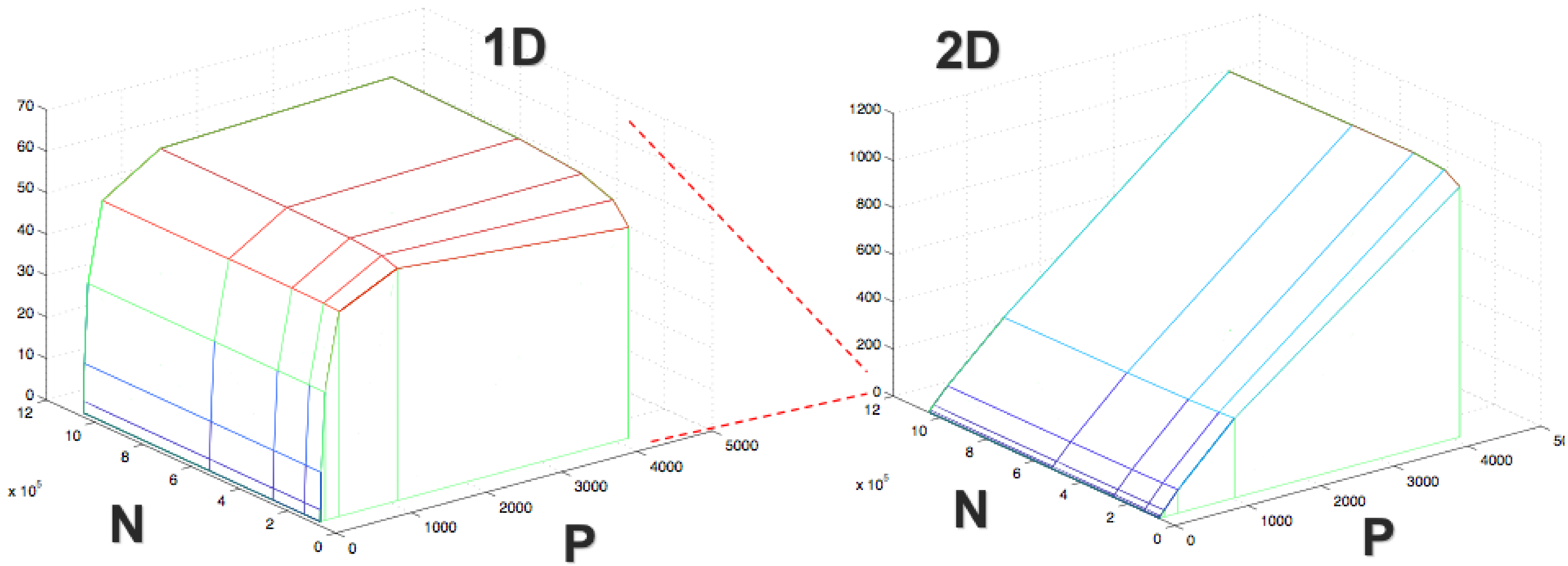






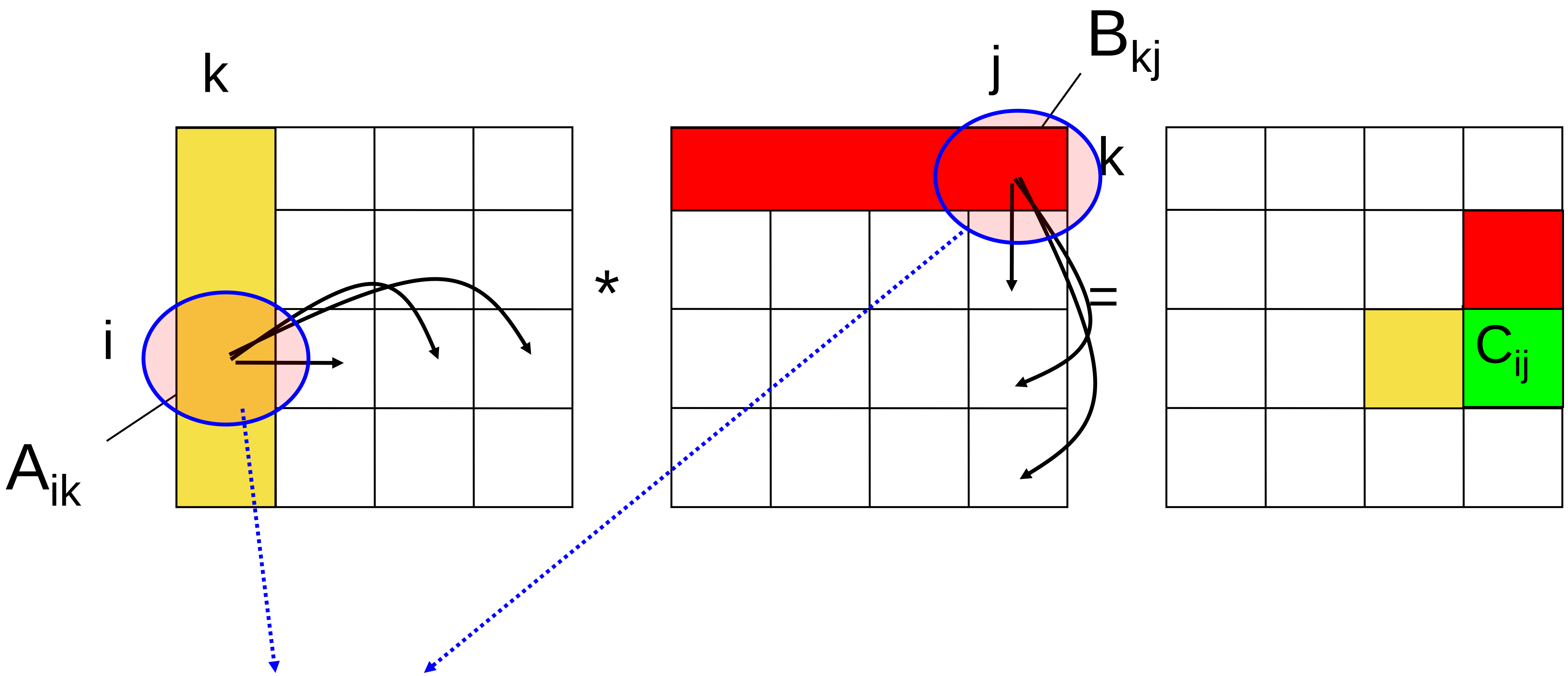
- Graph clustering (Markov, peer pressure)
- Shortest path calculations
- Betweenness centrality
- Subgraph / submatrix indexing
- Graph contraction
- Cycle detection
- Multigrid interpolation & restriction
- Colored intersection searching
- Applying constraints in finite element computations
- Context-free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

Comparative Speedup of Sparse 1D & 2D



In practice, 2D algorithms have <u>the potential</u> to scale, if implemented correctly. Overlapping communication, and maintaining load balance are crucial.

2-D example: Sparse SUMMA



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

- Based on dense SUMMA
- Generalizes to nonsquare matrices, etc.

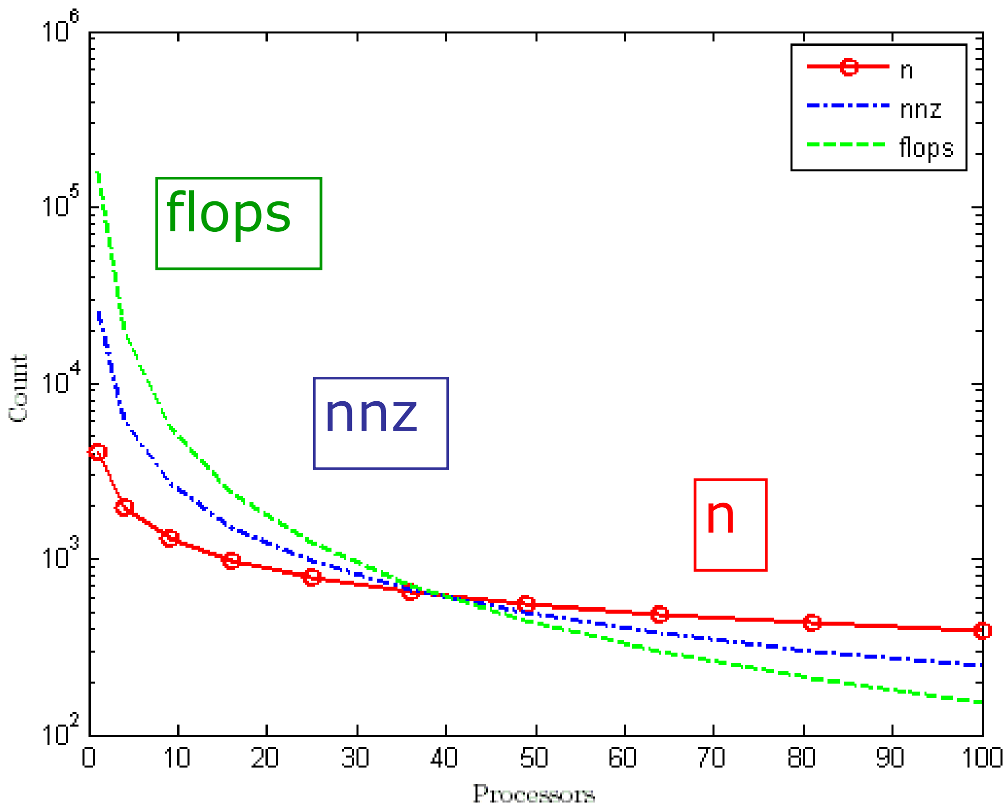
Sequential Kernel

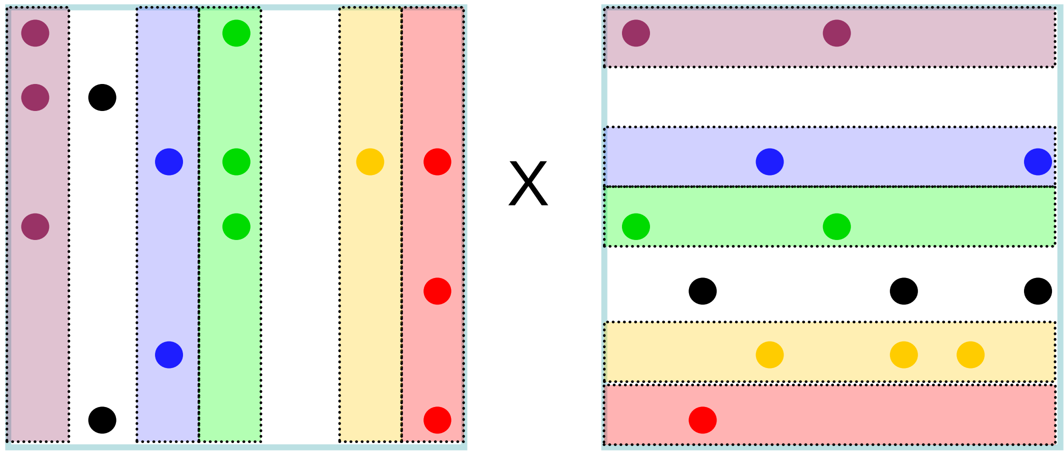
Standard algorithm is O(nnz+ flops+n)

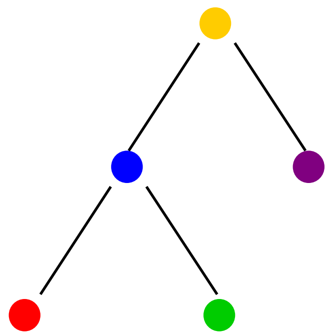
$$n'(\text{dimension}) \approx \frac{n}{\sqrt{p}}$$

$$nnz'(\text{data size}) \approx \frac{nnz}{p}$$

$$flops'(\text{work}) \approx \frac{flops}{p\sqrt{p}}$$



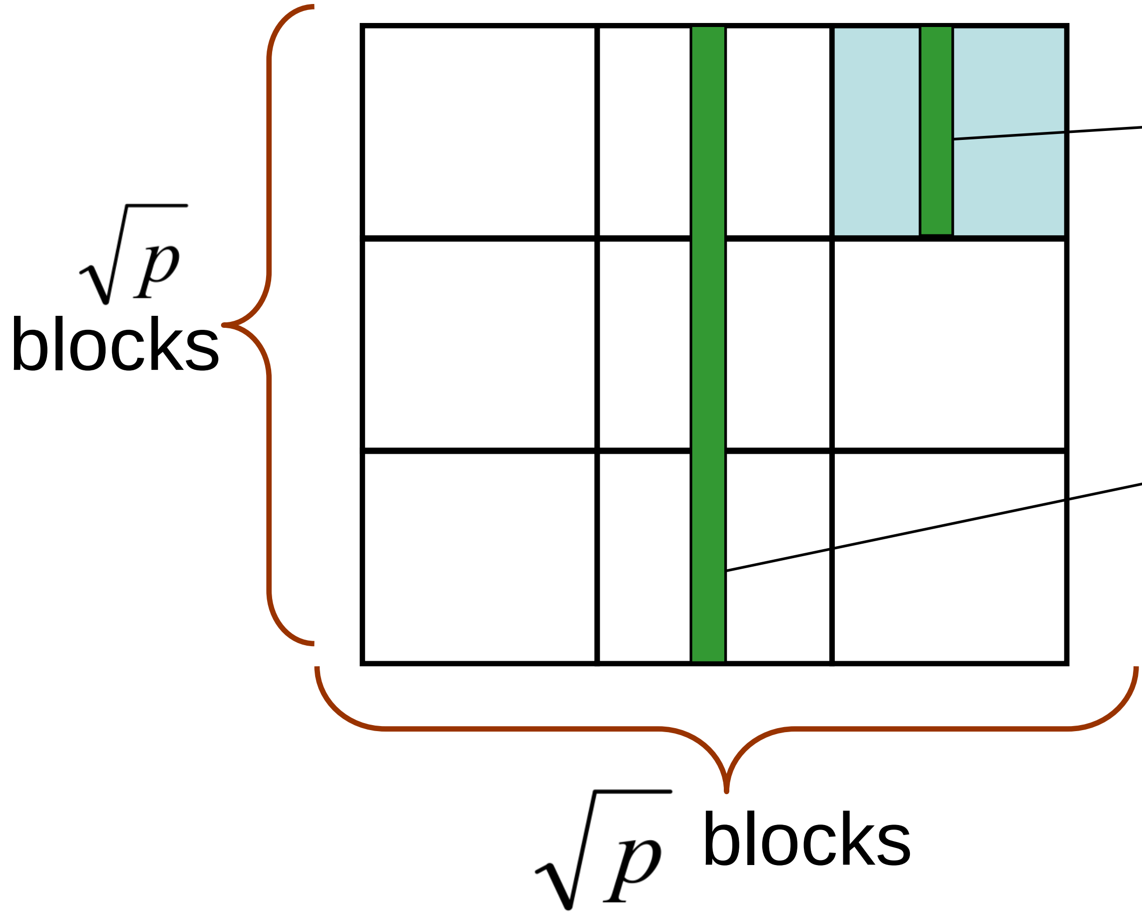




- Strictly O(nnz) data structure
- Outer-product formulation
- Work-efficient

Node Level Considerations

Submatrices are hypersparse (i.e. nnz << n)



$$nnz' = \frac{c}{\sqrt{p}} \rightarrow 0$$

Average of c nonzeros per column

<u>Total Storage</u>:

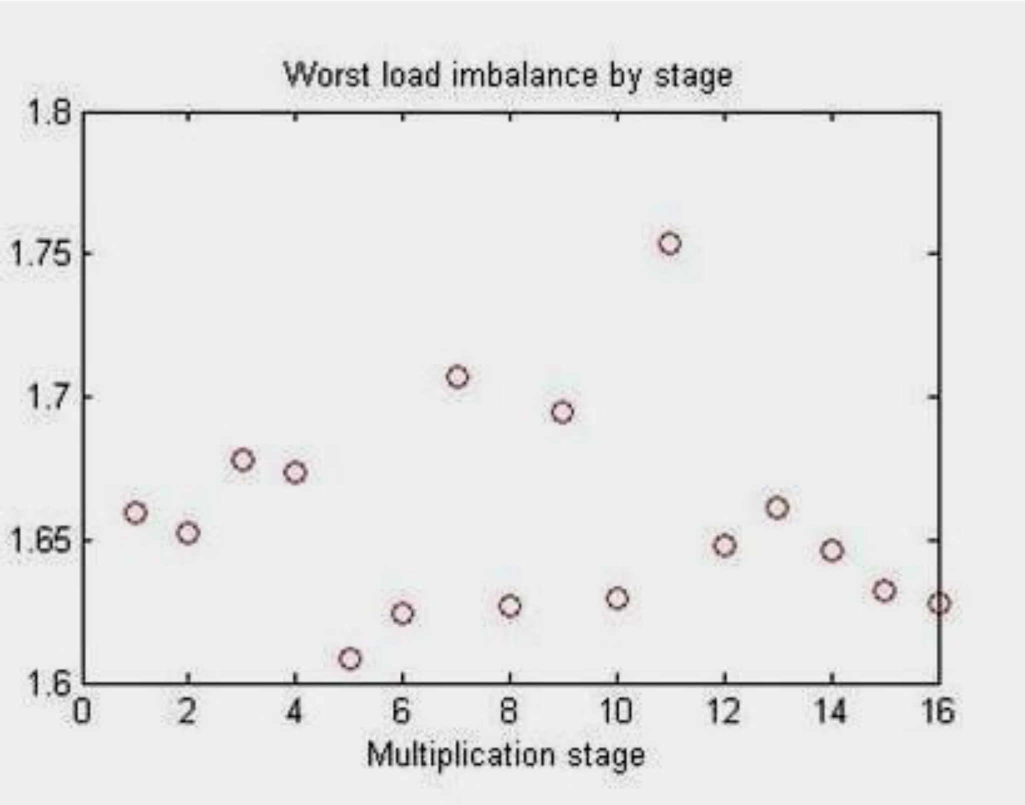
$$O(n + nnz) \Rightarrow O(n\sqrt{p} + nnz)$$

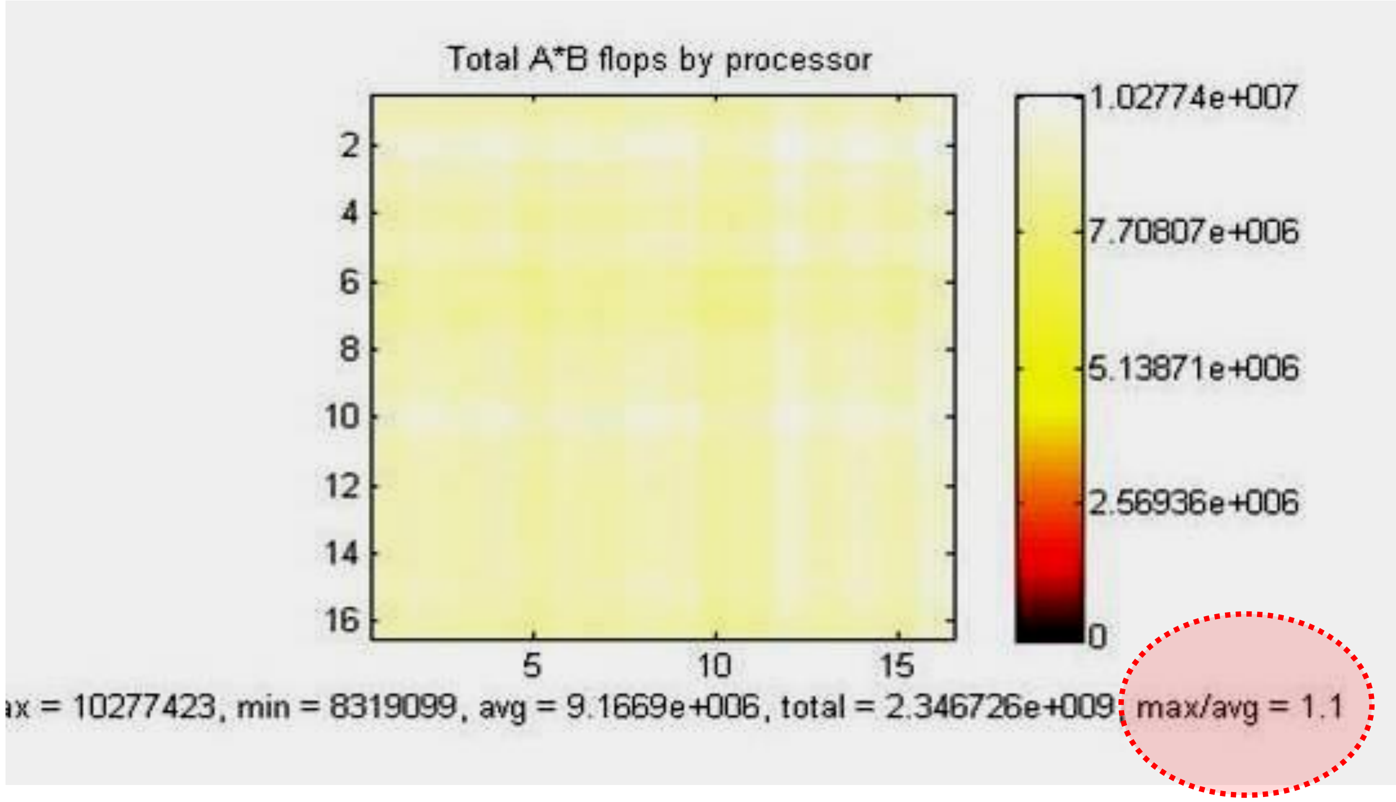
- A data structure or algorithm that depends on the matrix dimension n (e.g. CSR or CSC) is asymptotically too wasteful for submatrices

Addressing the Load Balance

RMat: Model for graphs with high variance on degrees

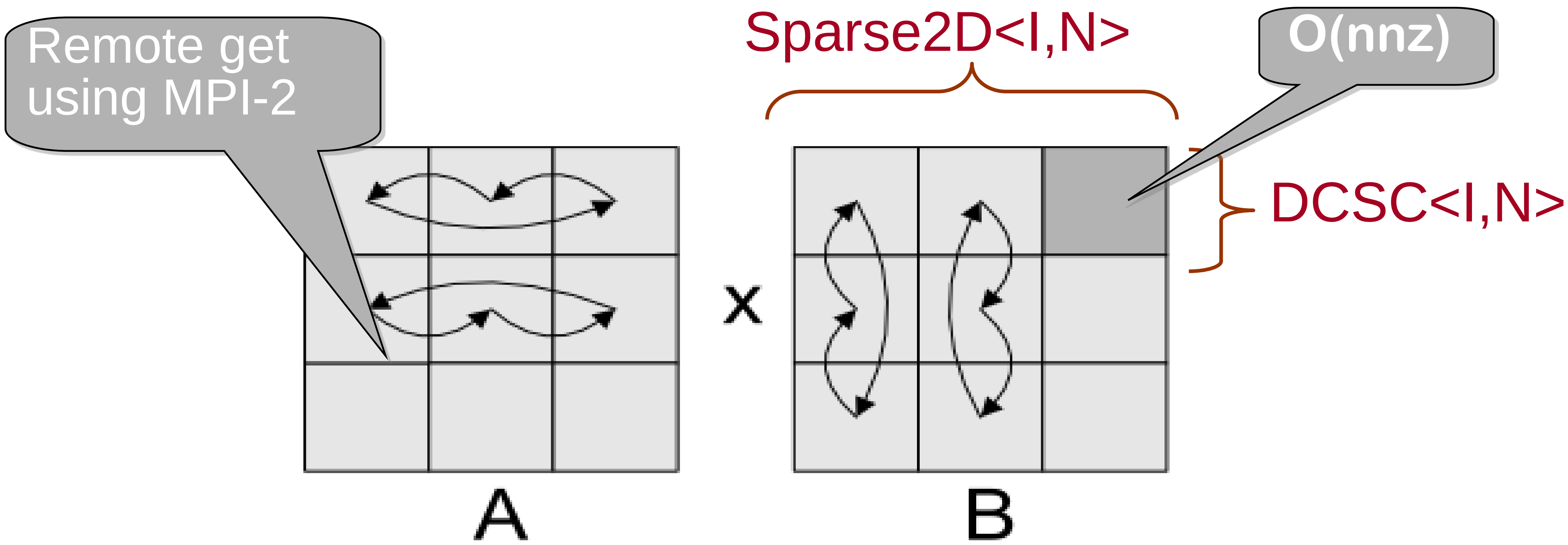
- Random permutations are useful. But...
- Bulk synchronous algorithms may still suffer:
- <u>Asynchronous</u> algorithms have <u>no notion of stages</u>.
- Overall, no significant imbalance.





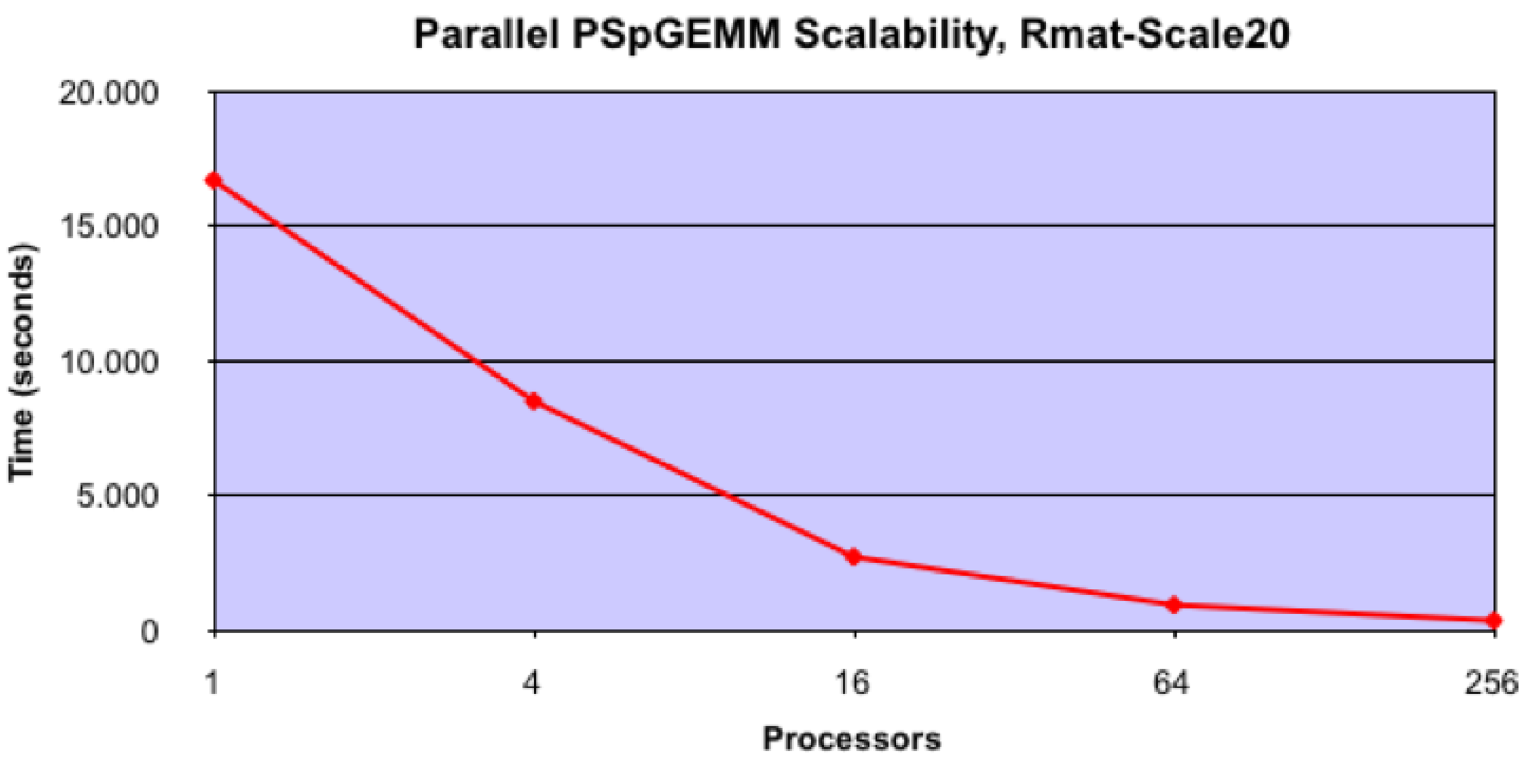
UCSB

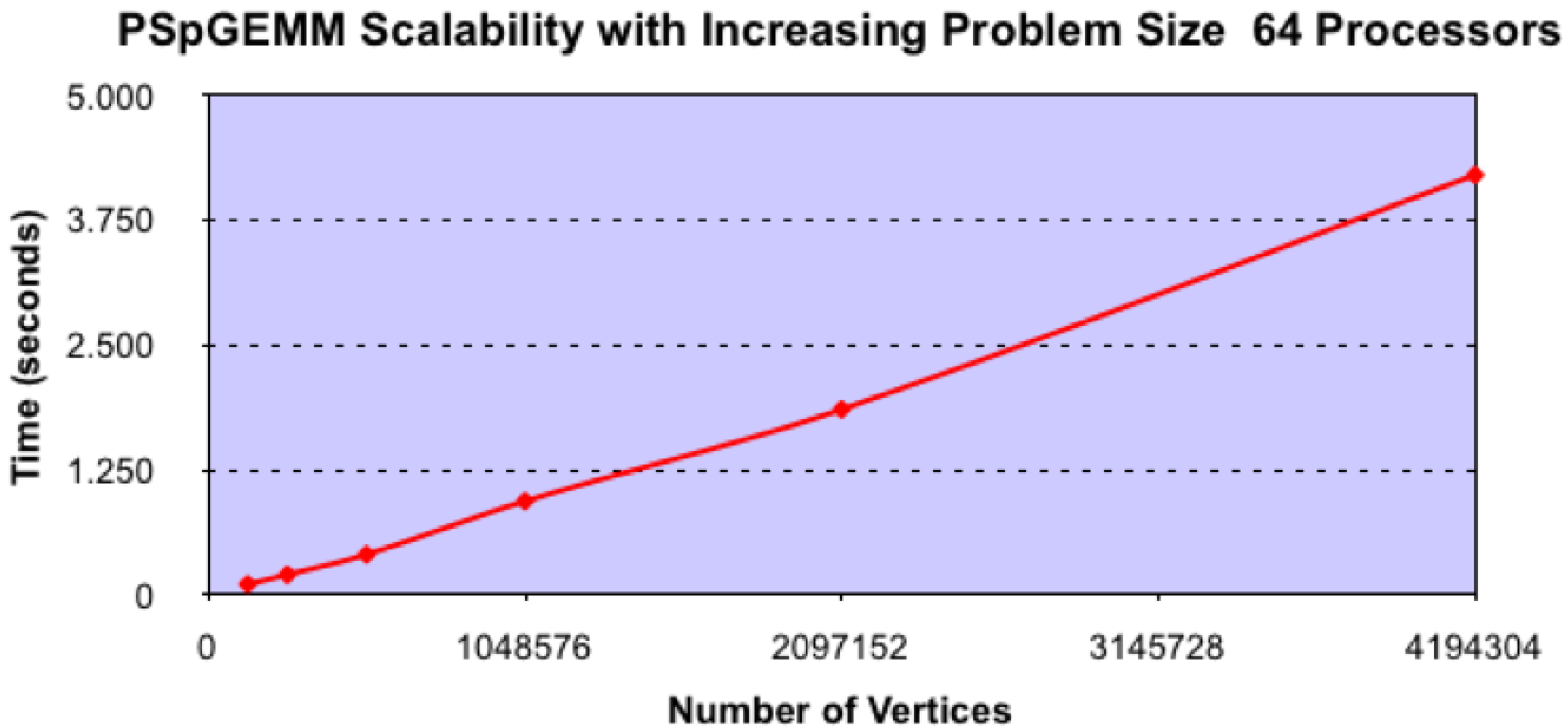
Asynchronous Implementation



- Two-dimensional block layout
- (Passive target) remote-memory access
- Avoids hot spots
- With very high probability, a block is accessed at most by a single remote get operation at any given time

Scaling Results for SpGEMM





- Asynchronous implementation

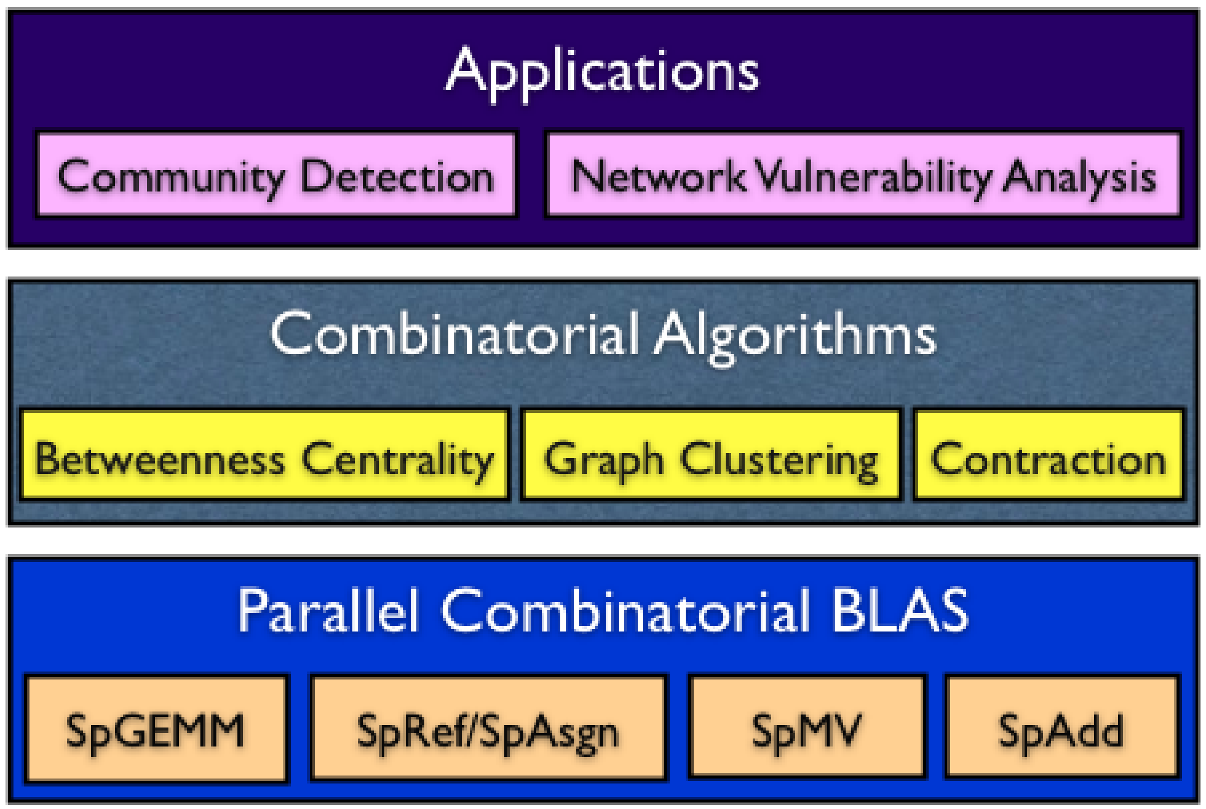
  One-sided MPI-2
- Runs on TACC's Lonestar cluster
- Dual-core dual-socket

  Intel Xeon 2.66 Ghz
- RMat X RMat product

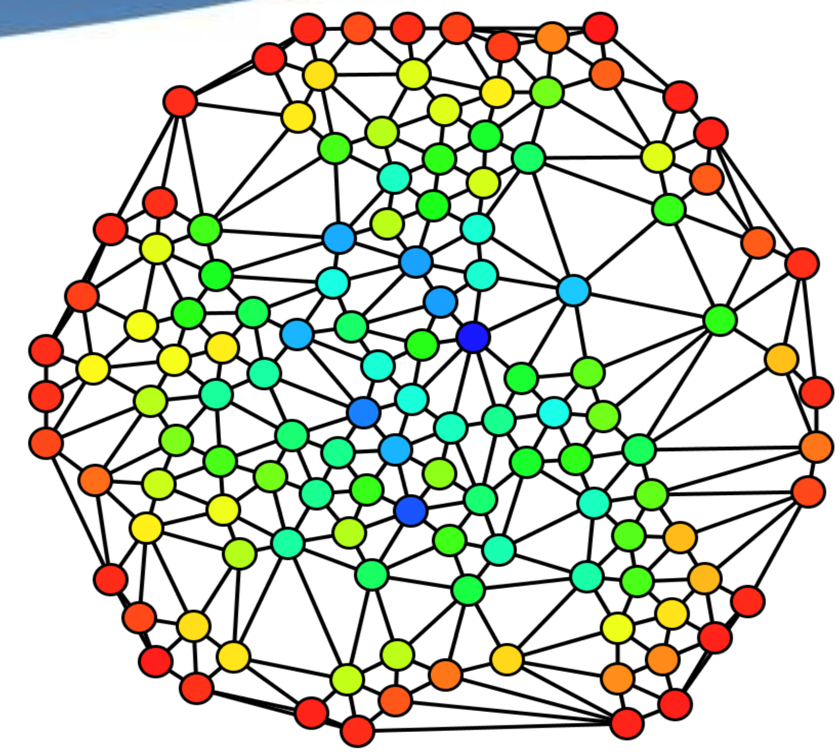
  Average degree (nnz/n) ≈ 8

UCSB

Applications and Algorithms



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



Betweenness Centrality

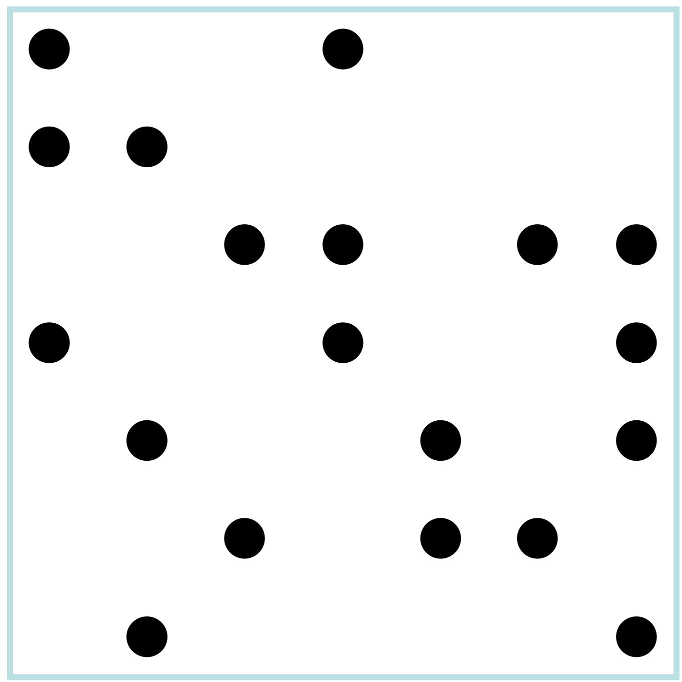
$C_B(v)$: Among all the shortest paths, what fraction of them pass through the node of interest?

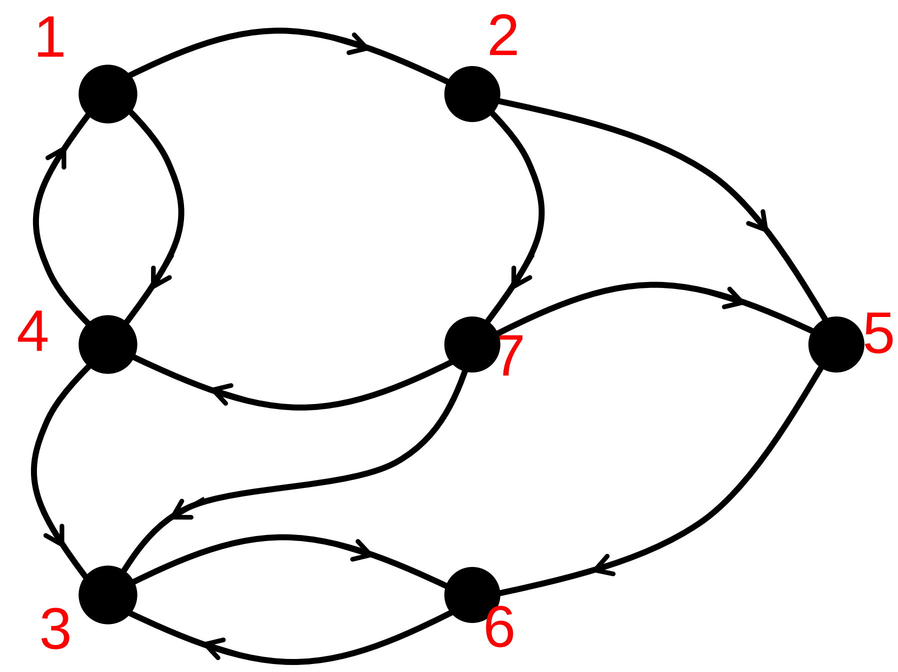
$$C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Brandes' algorithm

Betweenness Centrality using Sparse Matrices [Robinson, Kepner]

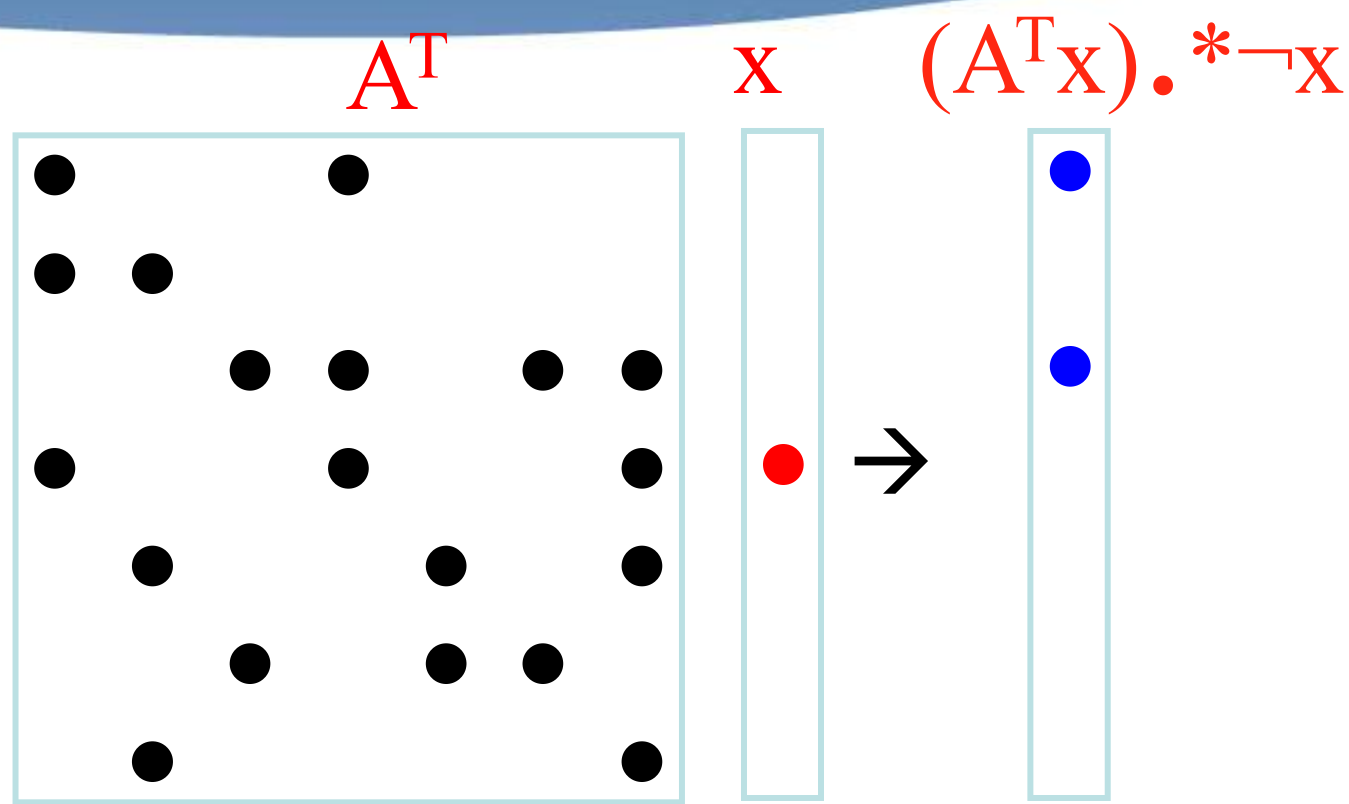
A^T

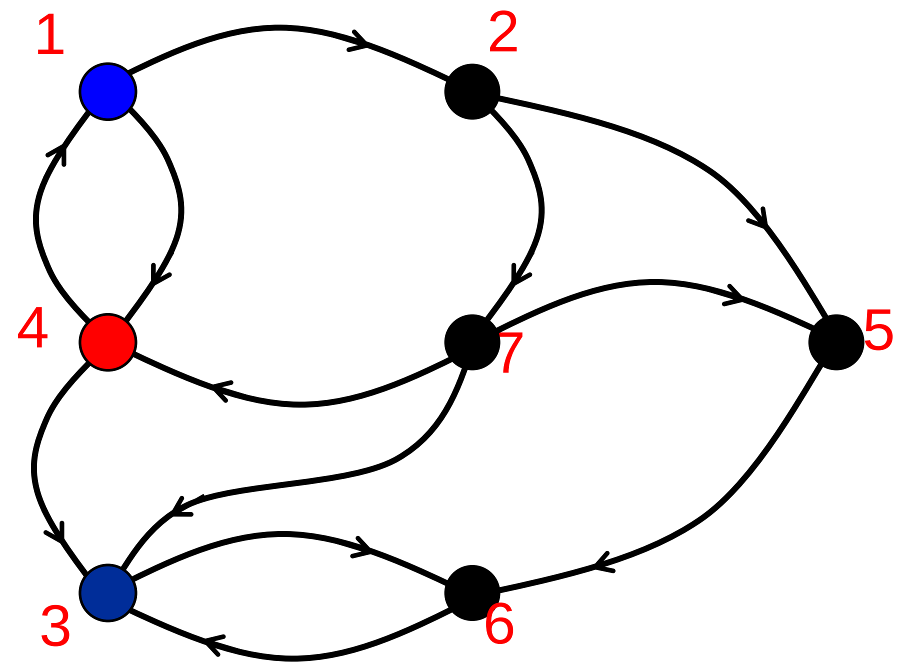


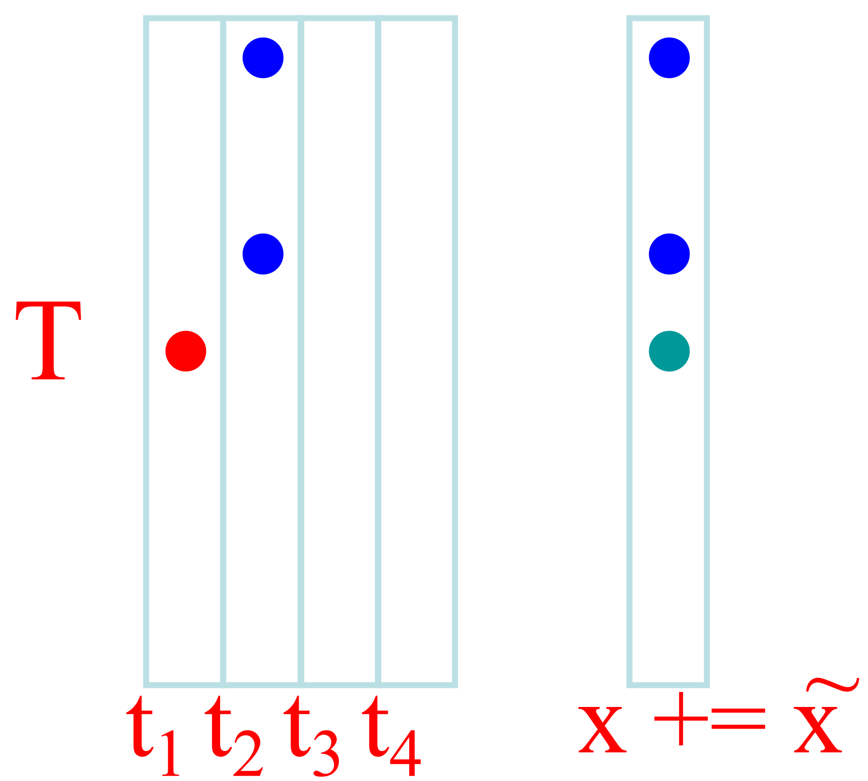


- Adjacency matrix: sparse array w/ nonzeros for graph edges
- Storage-efficient implementation from sparse data structures
- Betweenness Centrality Algorithm:
  1. Pick a starting vertex, v
  2. Compute shortest paths from v to all other nodes
  3. Starting with most distant nodes, roll back and tally paths

Betweenness Centrality using BFS

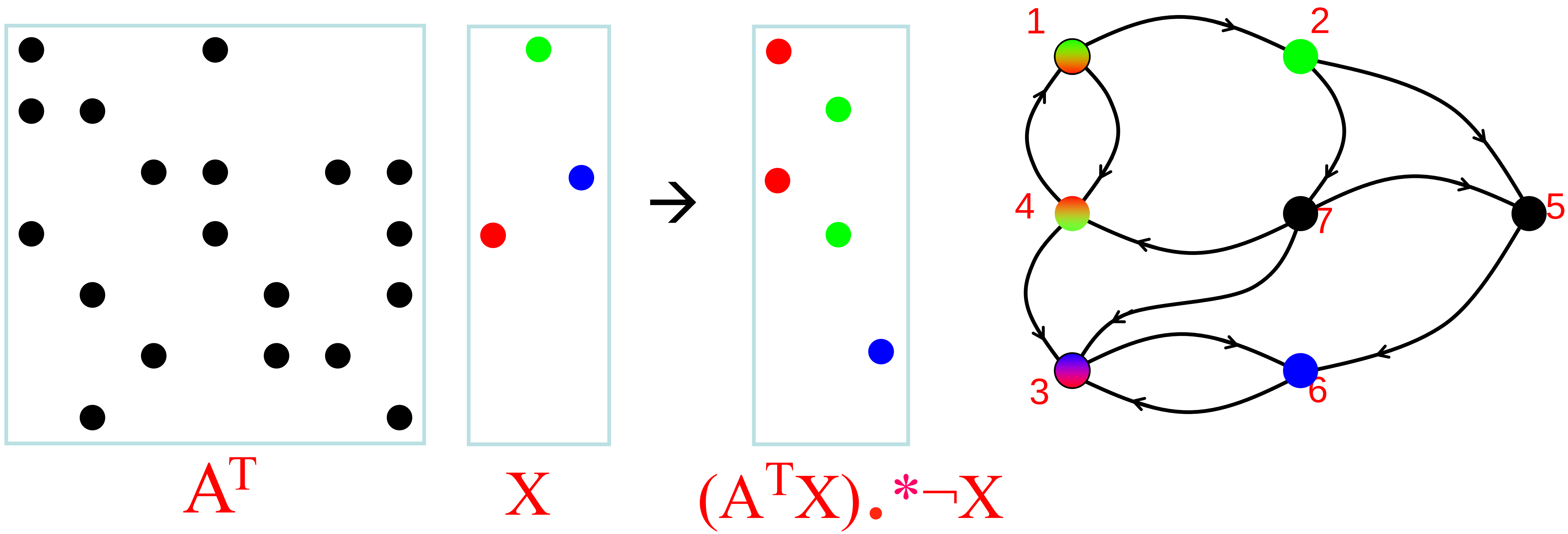






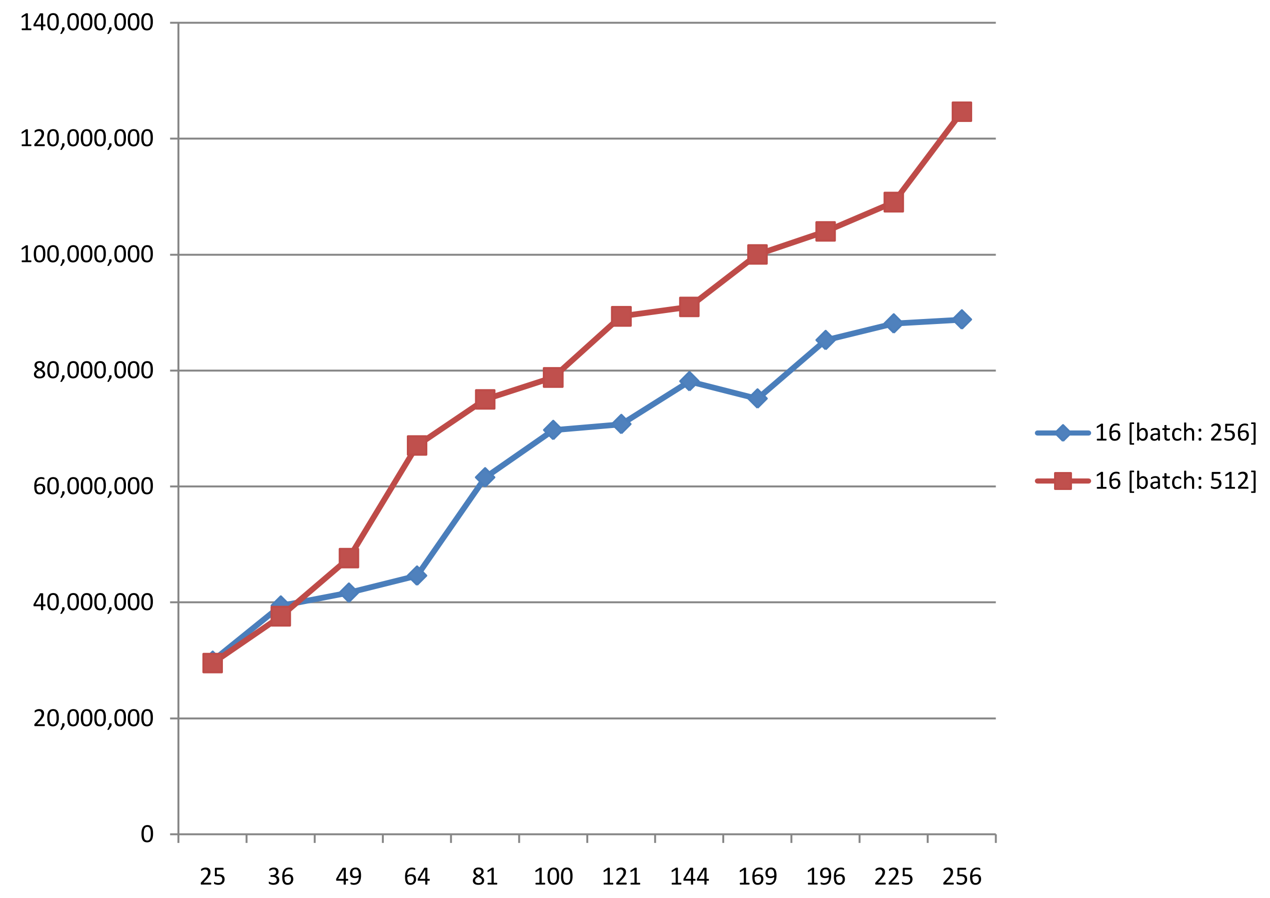
- Every iteration, another level of the BFS is discovered.
- Sparsity is preserved, but sparse matrix times sparse vector has <u>very little potential parallelism</u> (has o(nnz) work)

Parallelism: Multiple-source BFS

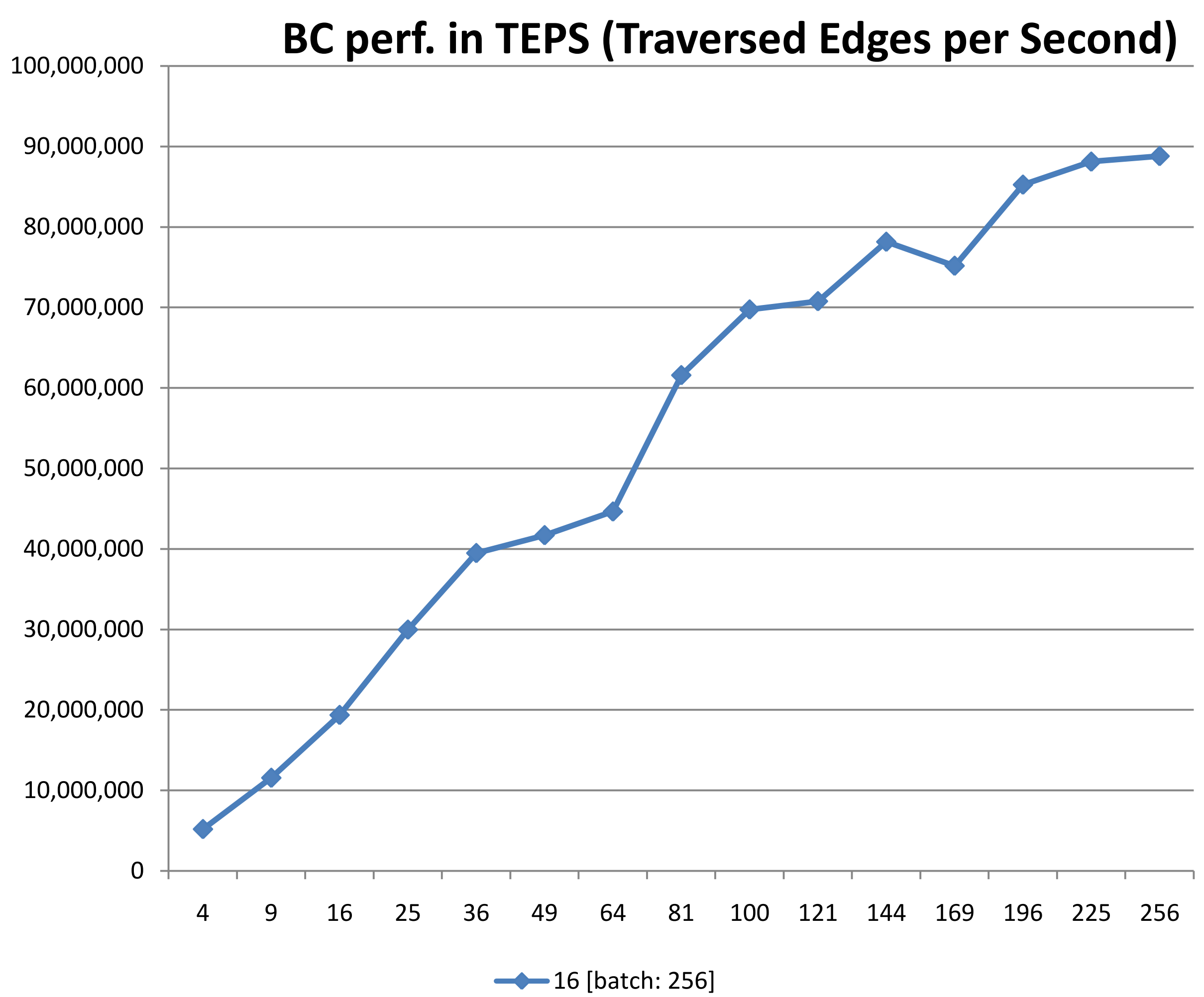


- Batch processing of multiple source vertices
- Sparse matrix-matrix multiplication => work efficient
- Potential parallelism is much higher
- Same applies to the tallying phase

Betweenness Centrality on Combinatorial BLAS



Batch processing greatly helps for large p



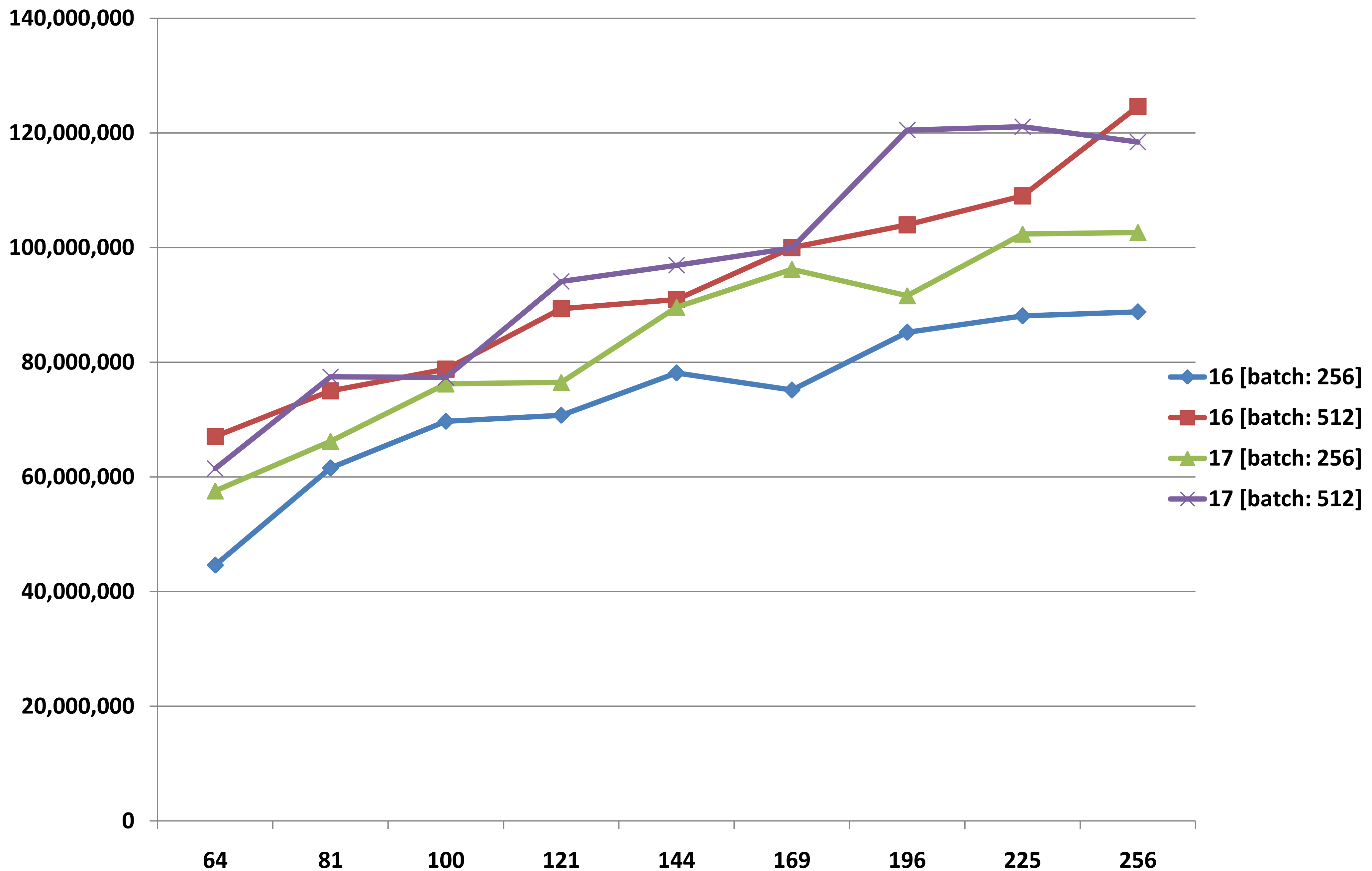
RMAT scale N has 2^N vertices and $8*2^N$ edges

- Likely to perform better on large inputs
- Code only a few lines longer than Matlab version

UCSB

Betweenness Centrality on Combinatorial BLAS

Fundamental trade-off:
Parallelism vs memory usage



Thank You !

Questions?

UCSB