



Challenges and Advances in Parallel Sparse Matrix-Matrix Multiplication

Aydin Buluc

John R. Gilbert

University of California, Santa Barbara

ICPP 2008

September 11, 2008

Reasons (pronounced “Outline”)

- Motivation?
 - Building block for many (graph) algorithms.
- Challenge?
 - Parallel scaling.
- Tools?
 - Novel work-scalable sequential kernel.
 - 2D parallel decomposition/algorithm
 - Overlapping communication with computation
- This work?
 - Comparative theoretical analysis of 1D and 2D algorithms.
 - Preliminary parallel implementation of the first 2D algorithm.
 - Challenges arising from sparsity, and ways to overcome them.

Matrices over Semirings

- Matrix multiplication $\mathbf{C} = \mathbf{AB}$ (or matrix/vector):

$$\mathbf{C}_{i,j} = \mathbf{A}_{i,1} \times \mathbf{B}_{1,j} + \mathbf{A}_{i,2} \times \mathbf{B}_{2,j} + \dots + \mathbf{A}_{i,n} \times \mathbf{B}_{n,j}$$

- Replace scalar operations \times and $+$ by

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

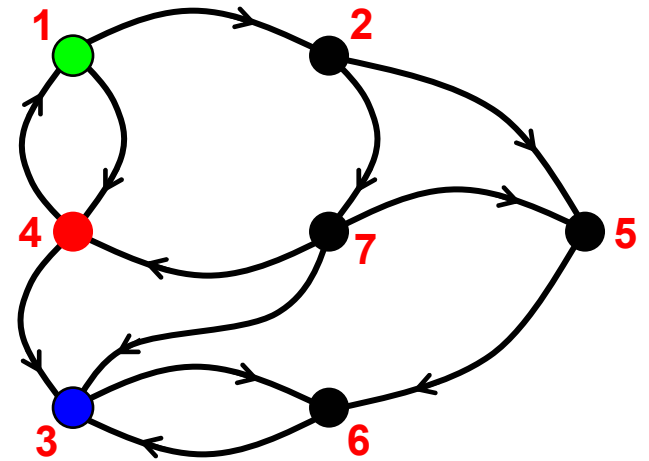
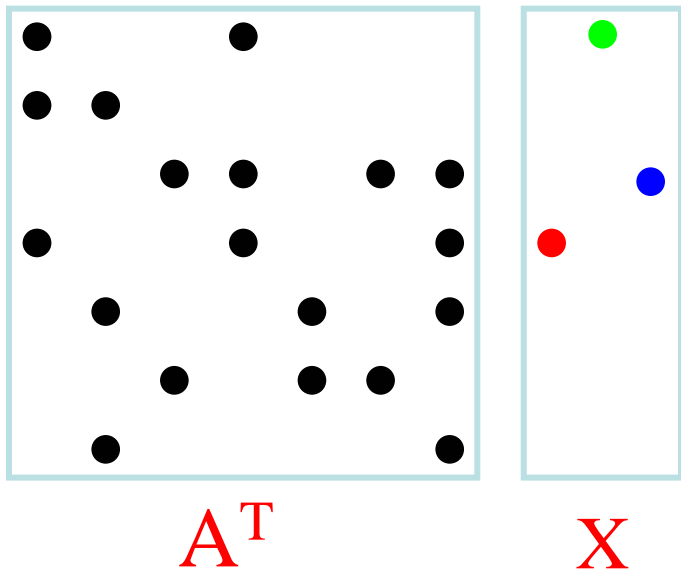
\oplus : associative, commutative, identity 0 annihilates under \otimes

- Then $\mathbf{C}_{i,j} = \mathbf{A}_{i,1} \otimes \mathbf{B}_{1,j} \oplus \mathbf{A}_{i,2} \otimes \mathbf{B}_{2,j} \oplus \dots \oplus \mathbf{A}_{i,n} \otimes \mathbf{B}_{n,j}$

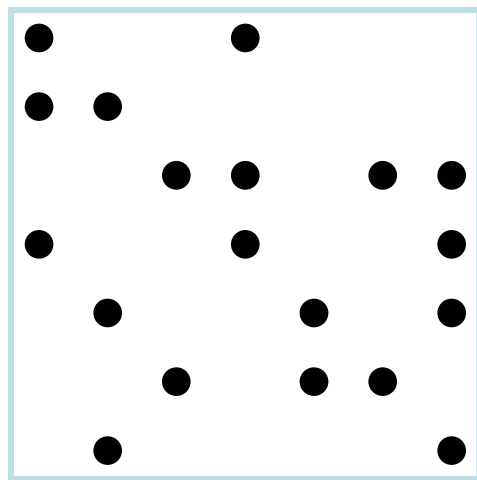
- Examples: $(\times, +)$; (and, or) ; $(+, \min)$; . . .

- Same data reference pattern and control flow

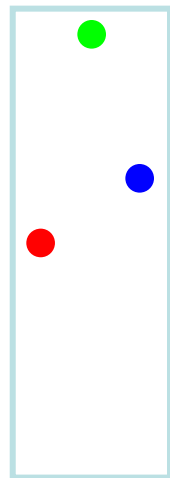
Multiple-source breadth-first search



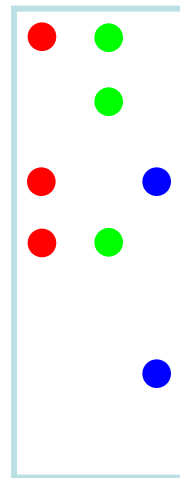
Multiple-source breadth-first search



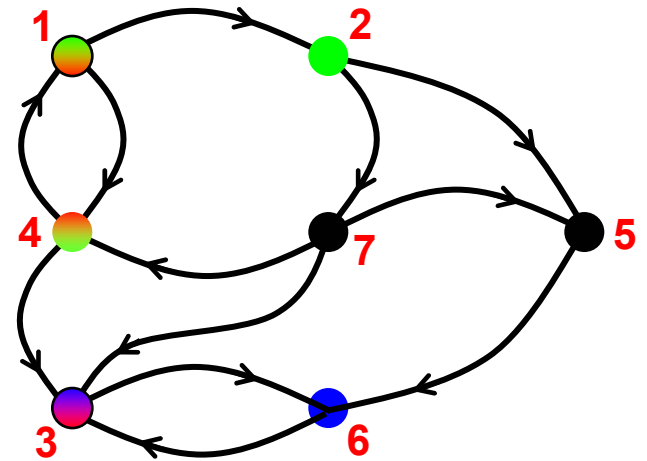
A^T



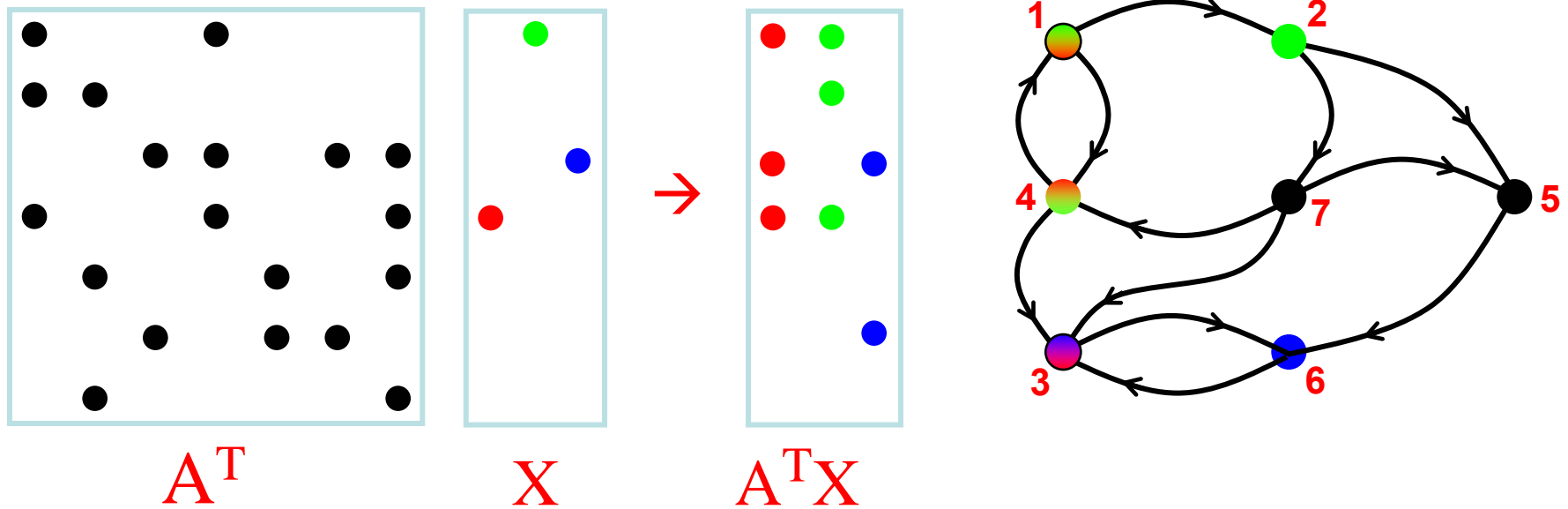
X



$A^T X$



Multiple-source breadth-first search



- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Load balance depends on SpGEMM implementation
- Not a panacea for the memory latency wall!

SpGEMM: Sparse Matrix \times Sparse Matrix

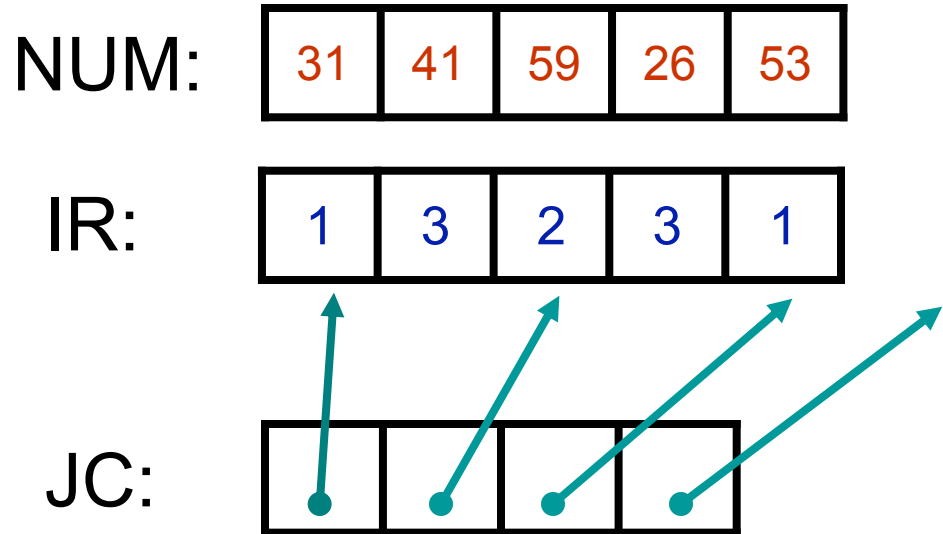
- Shortest path calculations (APSP)
- Betweenness centrality
- BFS from multiple source vertices
- Subgraph / submatrix indexing
- Graph contraction
- Cycle detection
- Multigrid interpolation & restriction
- Colored intersection searching
- Applying constraints in finite element computations
- Context-free parsing

Challenges of Parallel SpGEMM

- Scalable sequential kernel ($A_{ik} * B_{kj}$)
 - Solved [IPDPS'08]
- Load balancing
 - Especially for real world graphs
- Communication costs
 - Communication to computation ratio is much higher than dense GEMM
- Updates (additions)
 - scalar additions \neq scalar multiplications

Standard data structure: CSC

31	0	53
0	59	0
41	26	0



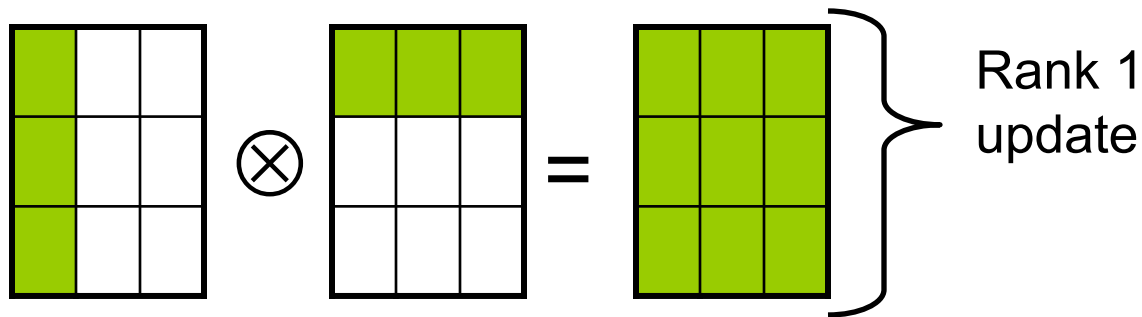
- JC holds column pointers – size: $n+1$
- IR holds row indices – size: nnz
- NUM holds numerical values – size: nnz

Organizations of Matrix Multiplication

1) outer product:

for $k = 1:n$

$$C = C + A(:, k) * B(k, :)$$

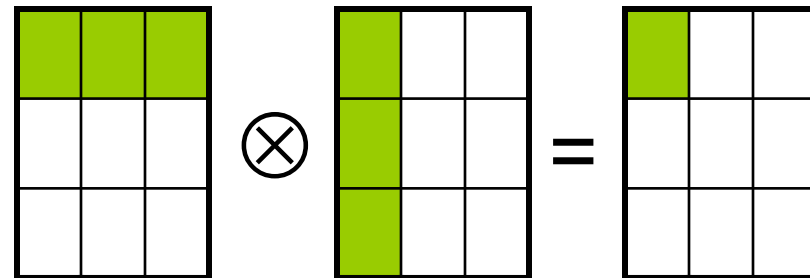


2) inner product:

for $i = 1:n$

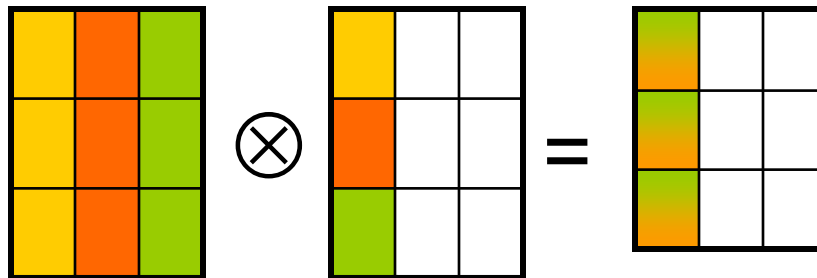
for $j = 1:n$

$$C(i, j) = A(i, :) * B(:, j)$$



Matlab's Algorithm

- 3) Column-by-column formulation:
for $j = 1:n$
 for all k s.t. $B(k,j) \neq 0$
 $C(:, j) = C(:, j) + A(:, k) * B(k, j)$

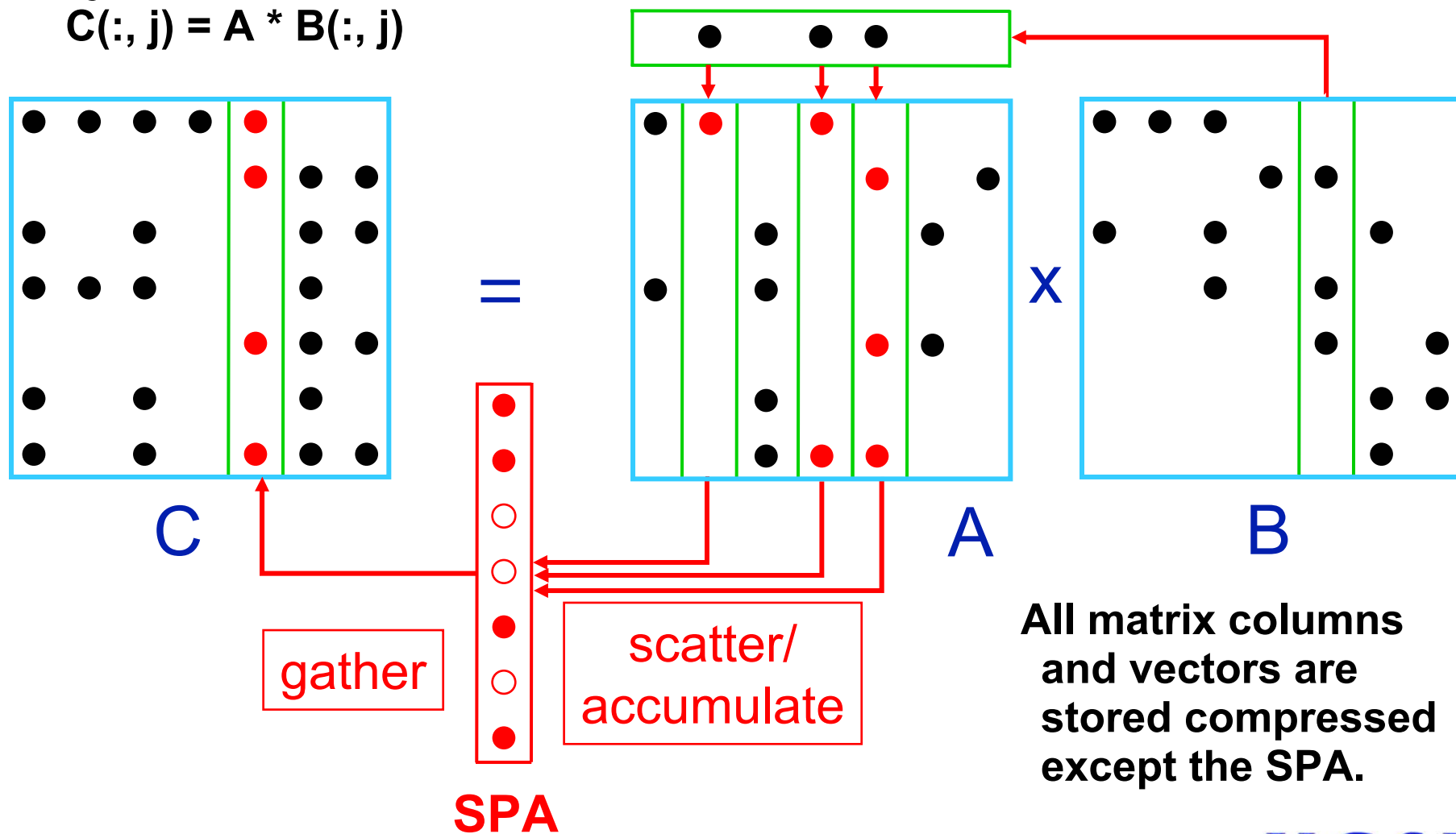


- Scanning $B(:,j)$ to find the nonzero elements is sequential.
- Scanning $A(:, k)$ for scalar*vector is sequential for each k .
- But the size and the structure of $C(:,j)$ is not known in advance and updates may be costly.

CSC Sparse Matrix Multiplication with SPA

for $j = 1:n$

$$C(:, j) = A * B(:, j)$$

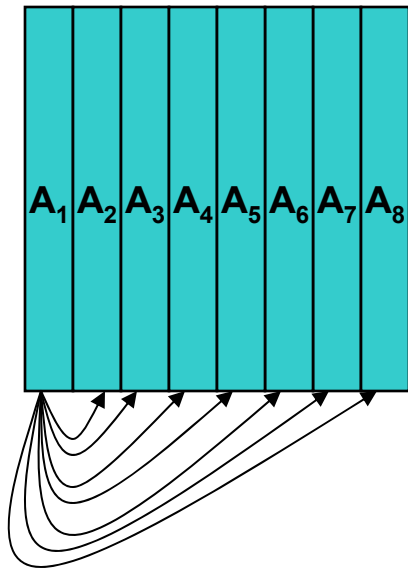


All matrix columns and vectors are stored compressed except the SPA.

SPA (Sparse Accumulator)

- Supports efficient vector addition by allowing random access to the currently active column.
 - Dense *Boolean array* (BA) to determine efficiently whether an element was previously inscribed or not.
 - Dense vector of real values (RE)
 - Unordered list of indices who were previously inscribed.
- Complexity:
 - Initialize SPA, set $x = 0$: $O(n)$ // Only once
 - $x = x + \alpha*y$: $O(\text{nnz}(y))$
 - Output x , reset $x = 0$: $O(\text{nnz}(x))$

Parallel 1D Algorithm



- Distribution by columns: only A needs to be communicated. (only B for distribution by rows)
- A_i refers to the n by n/p block column that processor i owns.
- Algorithm uses the formula:

$$C_i = C_i + A * B_i$$

- Processor i likely to need columns from all processors to fully form C_i
 - ✗ All-to-all broadcast at once. [we don't have space for that]
 - ✓ Multiple iterations to fully form C_i

Sparse 1D Algorithm

```
for all processors  $P_i$  in parallel
for  $j \leftarrow 1$  to  $p$  do
  Broadcast( $A_j$ )
  for  $k \leftarrow 1$  to  $N/p$  do
    SPA  $\leftarrow$  Load( $C_i(:,k)$ )
    SPA  $\leftarrow$  SPA +  $A_j * B_{ij}(:,k)$ 
     $C_i(:,k) \leftarrow$  UnLoad(SPA)
  end
end
end
end
```

Time spent on
load/unloads of SPA are
not amortized by flops

The sparsity parameter:
average number of
nonzeros per column/row

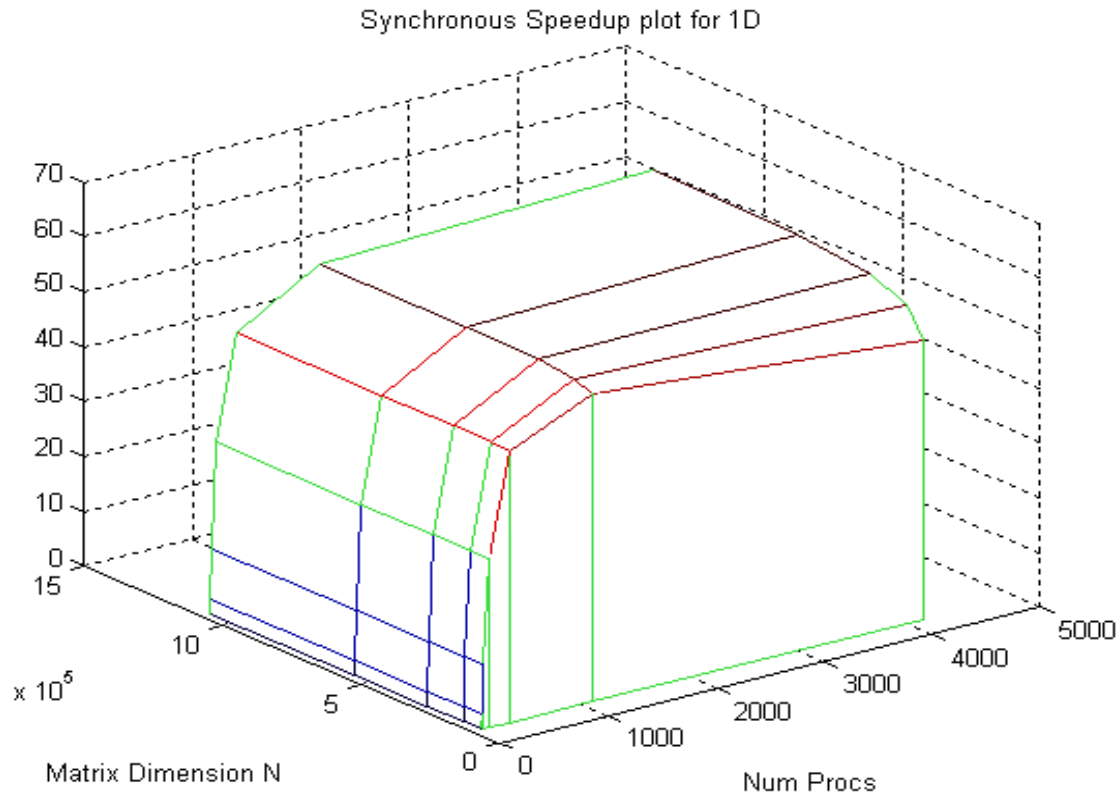
- Computation per processor is independent of p
- No speed up at all

$$T_{comp} = \gamma \left(N \left(1 + c^2 \right) \right)$$

The cost of one
arithmetic operation

Utopian 1D Algorithm

- Assume we found a way to amortize the cost of loads/unloads of SPA.
- It would still be unscalable due to communication.



2D Algorithms: The Dense Case

$p(0,0)$	$p(0,1)$	$p(0,2)$
$p(1,0)$	$p(1,1)$	$p(1,2)$
$p(2,0)$	$p(2,1)$	$p(2,2)$

- For the dense case, 2D scales better with the number of processors
- Likely to be same for the sparse case.

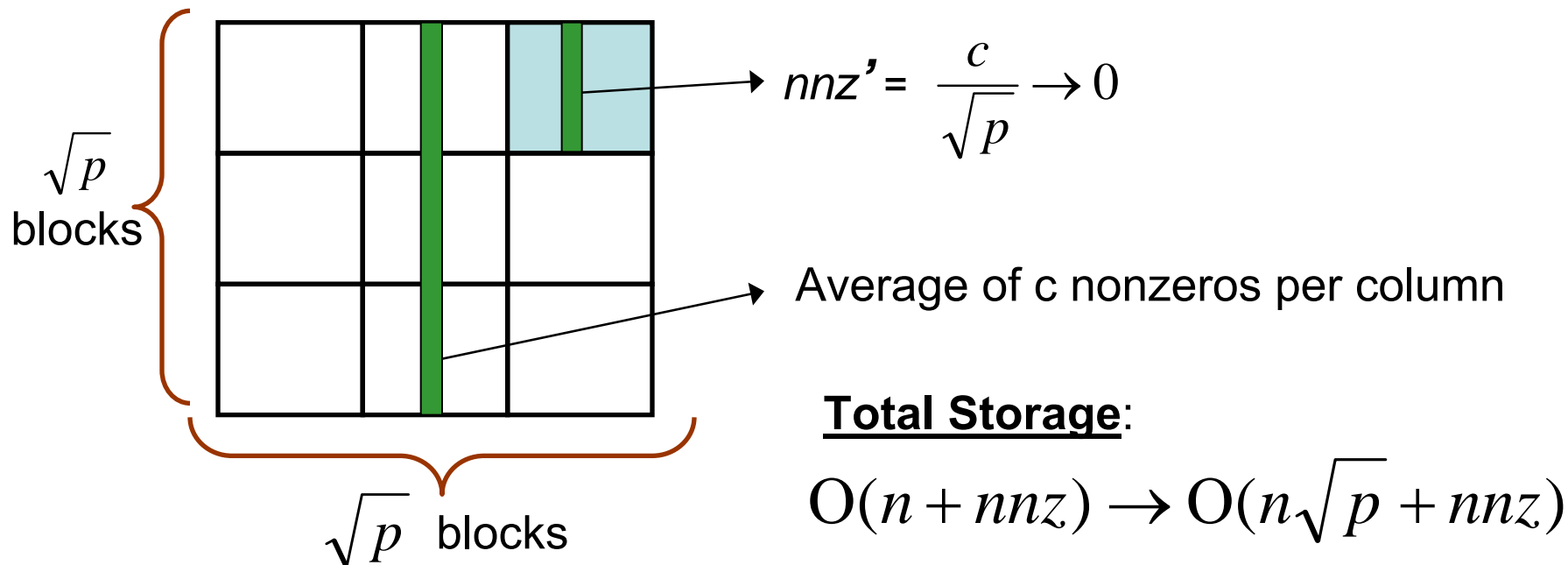
Parallel Efficiency:

$$\text{1D Layout: } \frac{1}{1 + O\left(\frac{p}{n}\right)}$$

$$\text{2D Layout: } \frac{1}{1 + O\left(\frac{\sqrt{p}}{n}\right)}$$

Should be zero for perfect efficiency

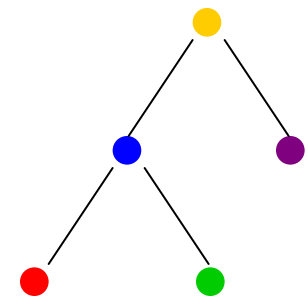
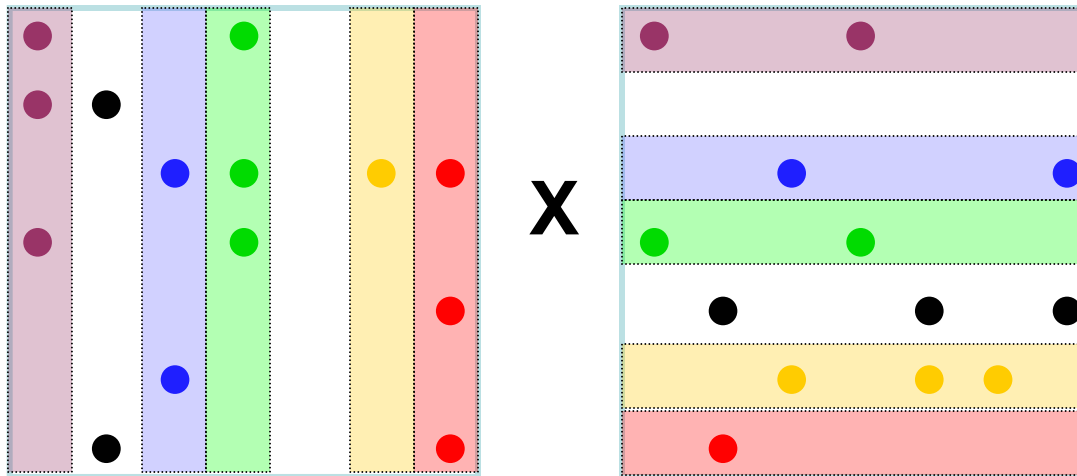
Submatrices are *hypersparse* (i.e. $nnz \ll n$)



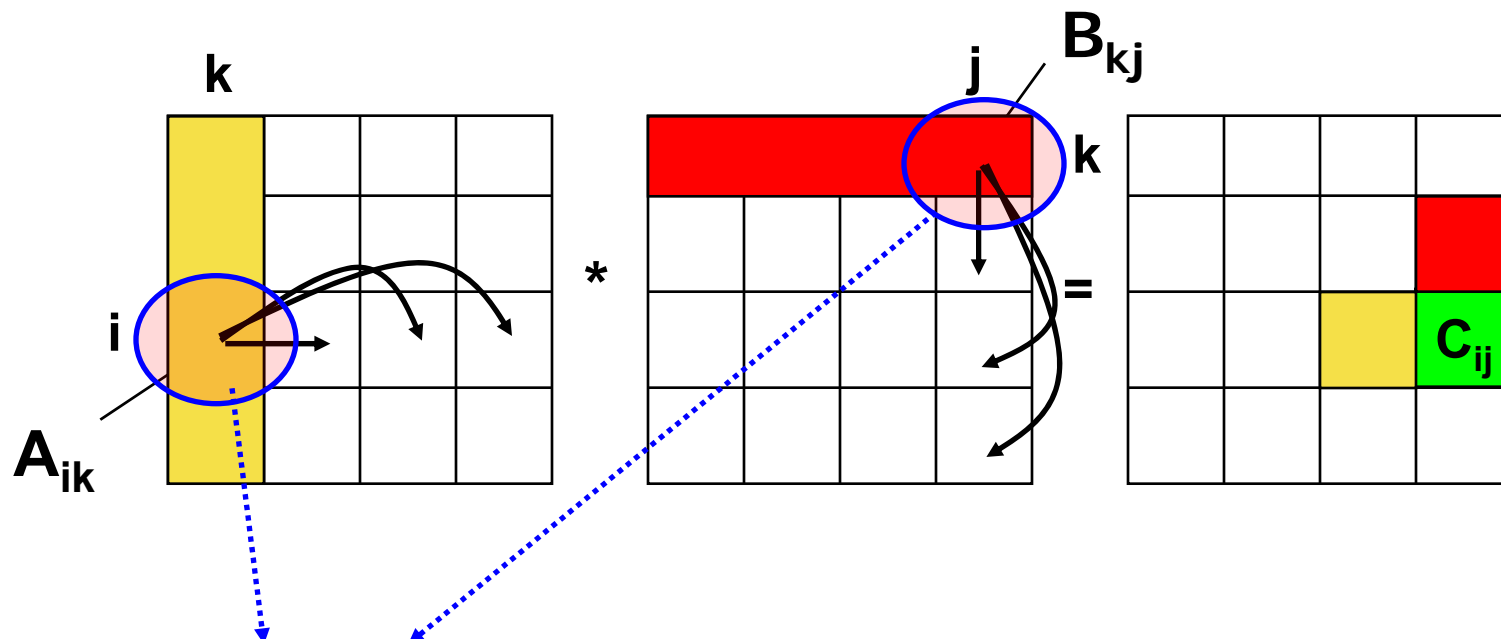
- A data structure or algorithm that depends on the matrix dimension n (e.g. CSR or CSC) is asymptotically too wasteful for submatrices

Sequential Kernel [IPDPS 2008]

- Strictly $O(nnz)$ data structure
- Complexity independent of matrix dimension
- Outer-product formulation with multi-way merging
- Scalable in terms of the amount of work it performs.



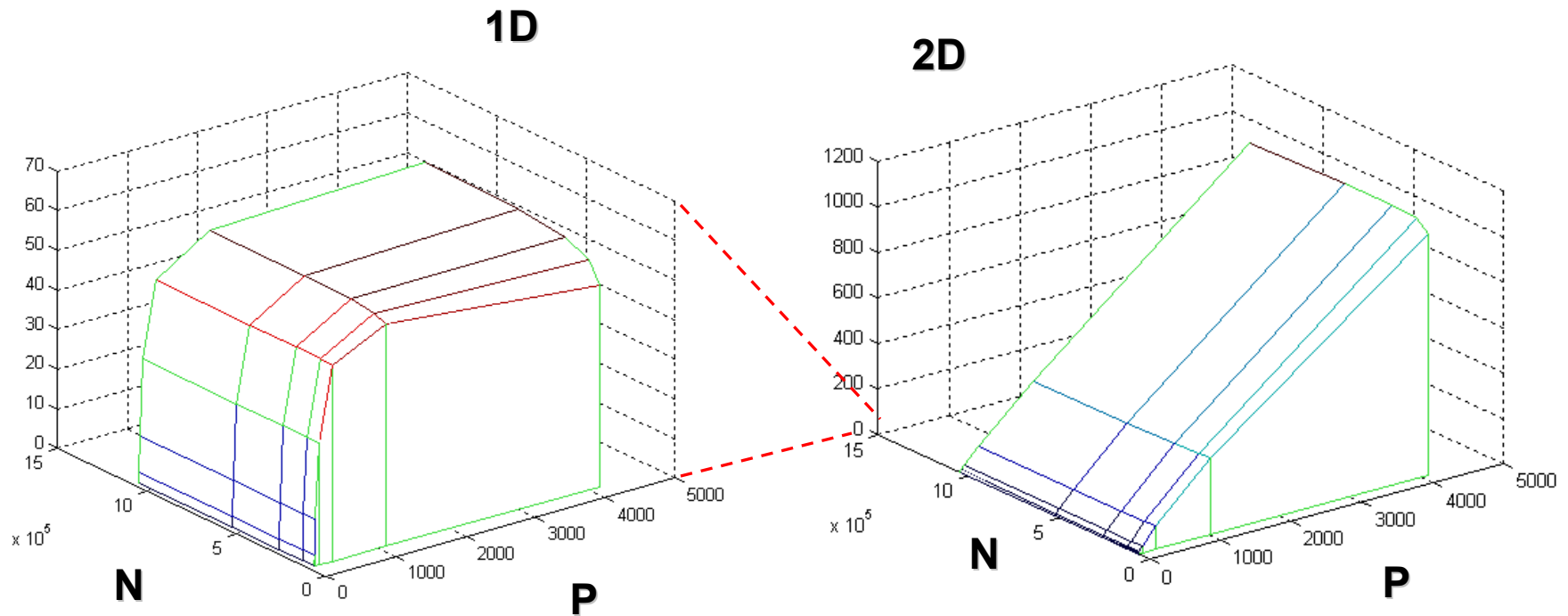
2D Example: Sparse SUMMA



- $C_{ij} += A_{ik} * B_{kj}$
- At worst doubles local storage

- Based on SUMMA (block size = $n/\text{sqrt}(p)$)
- Easy to generalize nonsquare matrices, etc.

Comparative Speedup of Sparse 1D & 2D



- Even utopian 1D algorithms can not scale beyond 40x
- Break-even point is around 50 processors.

Parallel efficiency of 2D algorithm?

- Parallel efficiency due to computation: $E_{comp} = \frac{cN}{p + cN \log\left(\frac{c^2 N}{p}\right)}$
 - ✓ As long as N grows linearly with p (weak scaling), this is constant at $1/\log(c^2 N/p)$
 - ✓ Constant but not 1
- Parallel efficiency due to latency: $E_{latency} = \frac{\gamma \cdot c^2 N}{\alpha \cdot p \sqrt{p}}$
 - ✓ N should grow on the order of $p^{1.5}$
- Parallel efficiency due to bandwidth: $E_{bw} = \frac{\gamma \cdot c}{\beta \cdot \sqrt{p}}$
 - ✓ Better than 1D
 - ✓ Yet, still not scalable.

Hiding Communication Costs

- SpGEMM is much harder to scale than dense GEMM.
 - × Both communication and computation costs increase at the same rate with increasing problem size !
- Overlap communication with computation.
 - ✓ As much as possible. Possible for $p < 4000$
- Asynchronous, one sided communication

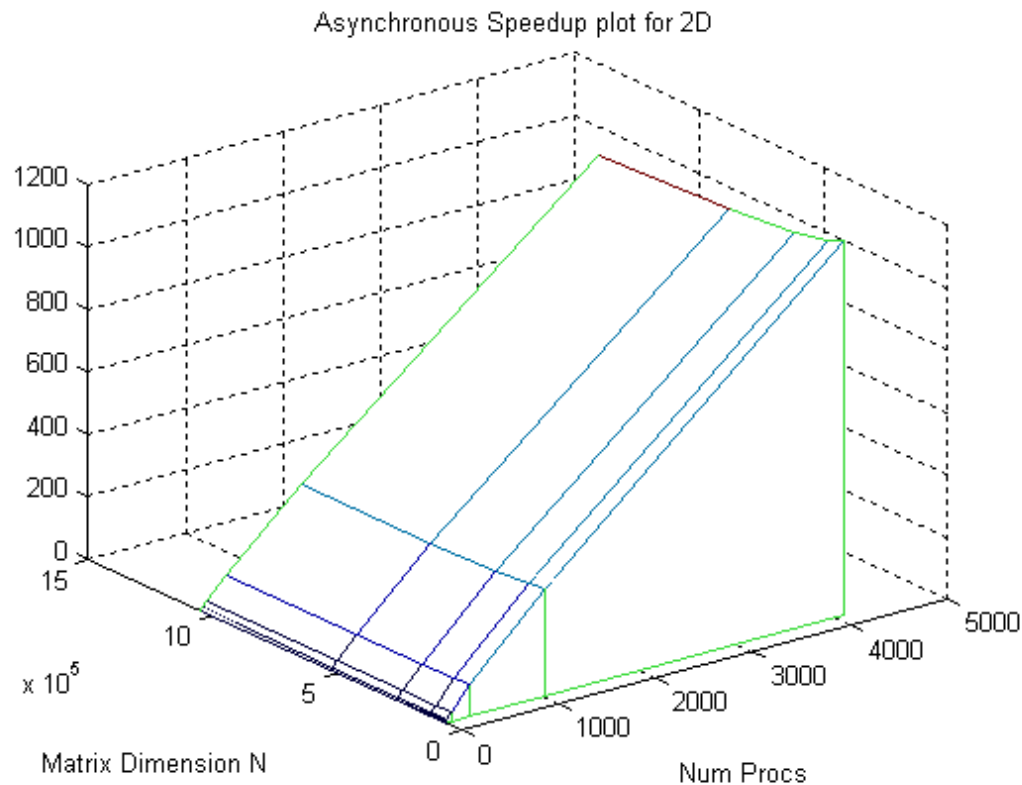
GASNET, ARMCI

(Truly one-sided) Communication layers

Myrinet, Infiniband, etc

Hardware supporting zero copy RDMA

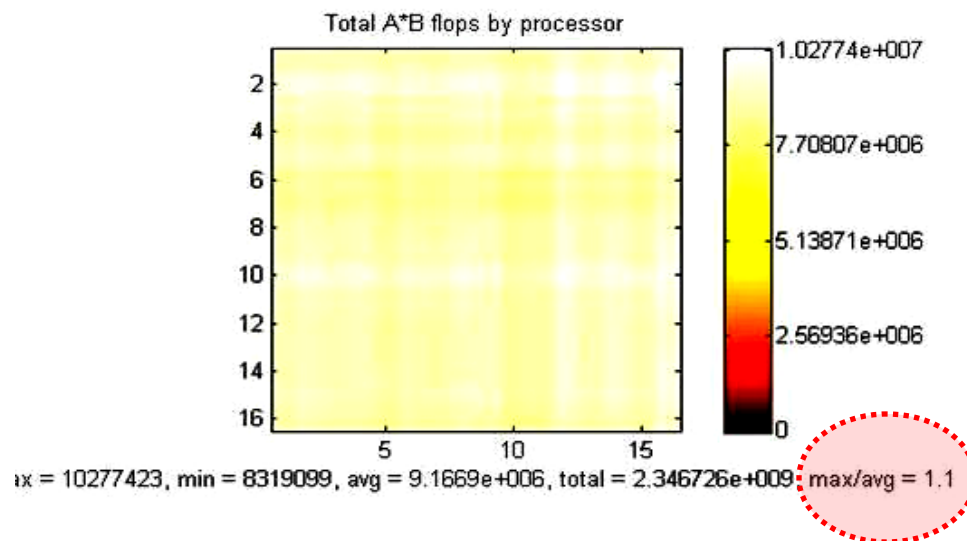
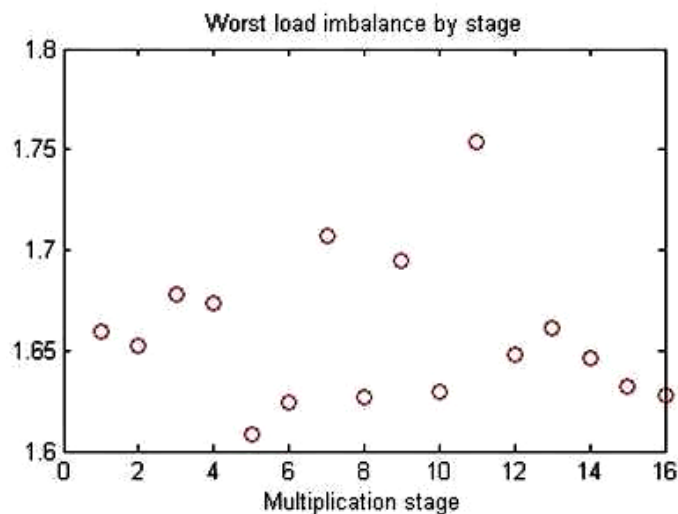
Speedup of Asynchronous 2D



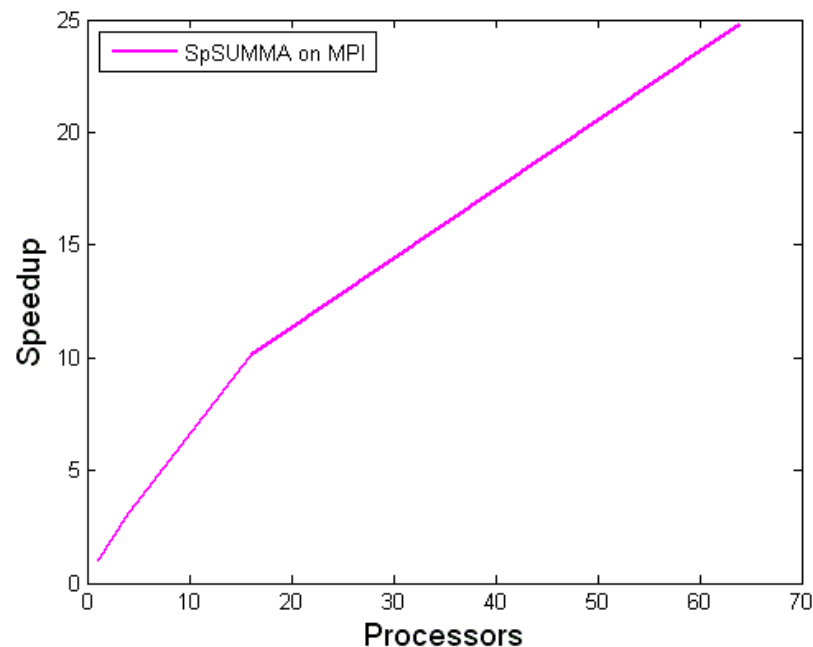
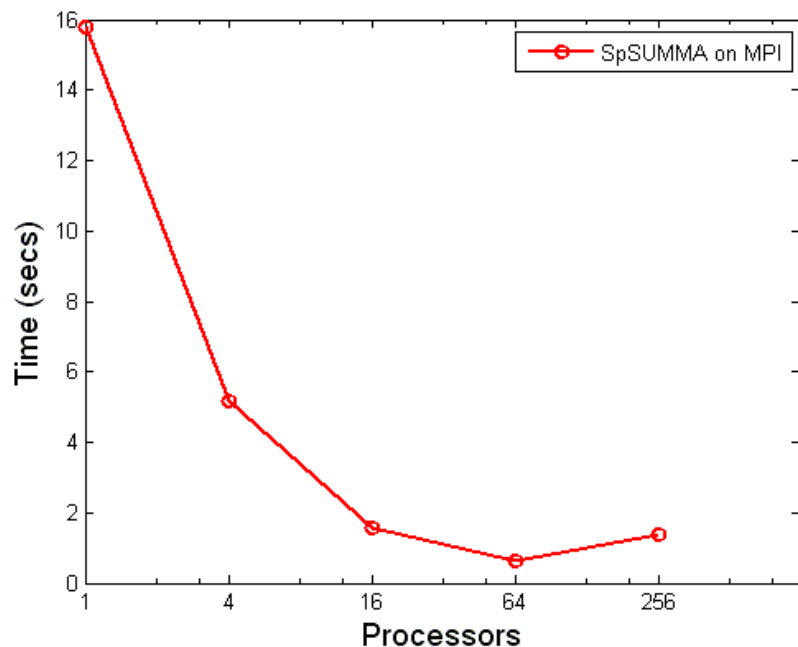
- ✓ Slightly better performance for smaller matrices
- ✓ Remember, this assumes random matrices (Erdos-Renyi)

Addressing the Load Balance (in Real-World Matrices)

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



Strong Scaling of Real Code



- Synchronous implementation with C++ and MPI
- Runs on TACC's Lonestar cluster (2.66Ghz dual-core nodes)
- Good up to $p = 64$, flattens out at $p = 256$

Barriers to Scalability (pronounced “Future Work”)

- Submatrix additions ($C_{ij} += A_{ik} * B_{kj}$)
 - o We currently use scanning based merging
 - o Fast unbalanced merging is required.
- Load-balance problem
 - o We used real world matrices for testing the implementation.
 - o One-sided communication through RDMA is required
- Extra $\log(p)$ factor in Sparse SUMMA
 - o Creates extra communication costs
 - o Better overlapping communication with computation
 - o Again, solution is asynchronous implementation.

Thank You !

Questions?