Communication avoiding for sparse matrices and graphs

Laura Grigori

ALPINES
INRIA and LJLL, UPMC
On sabbatical at UC Berkeley

March 2016

Plan

Introduction

Sparse Matrix Matrix multiplication
- Lower bounds for matrices with random sparsity
- Communication optimal algorithms

Sparse Cholesky factorization for SPD matrices
- Combinatorial tools: undirected graphs, elimination trees
- Parallel Cholesky factorization
- Lower bounds for model problems

Graphs: All pairs shortest path

Plan

Lower bounds on communication for sparse LA

- More difficult than the dense case
  - For example computing the product of two (block) diagonal matrices involves no communication in parallel
- Lower bound on communication from dense linear algebra is loose
- Very few existing results:
  - Lower bounds for parallel multiplication of sparse random matrices [Ballard et al., 2013]
  - Lower bounds for Cholesky factorization of model problems [Grigori et al., 2010]

Plan

Introduction

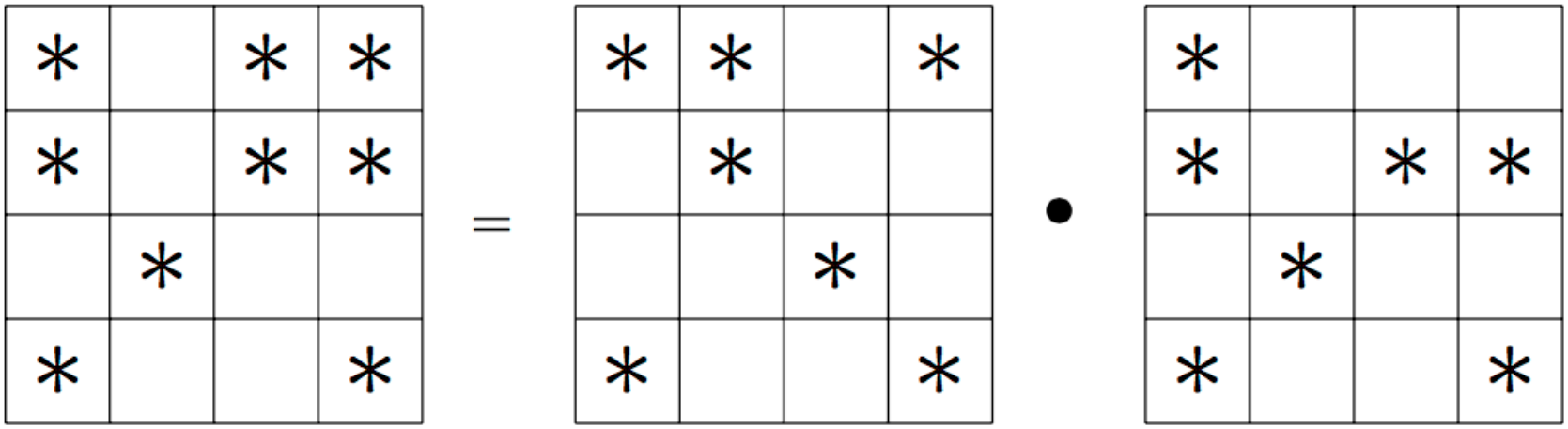
Sparse Matrix Matrix multiplication

- Lower bounds for matrices with random sparsity
- Communication optimal algorithms

Sparse Cholesky factorization for SPD matrices

Graphs: All pairs shortest path

Sparse matrix multiplication (SpGEMM)



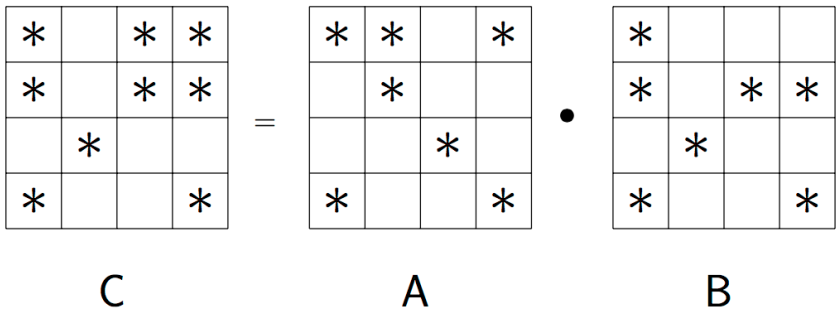
$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Slides with help from G. Ballard, A. Buluc, O. Schwartz. Results from B. Lipshitz thesis

Sequential sparse matrix multiplication

- Column-wise formulation by Gustavson, implemented in Matlab.
- Input matrices of size $n \times n$, stored in compressed sparse column.
- Complexity: $O(flops(A \cdot B) + nnz + n)$, optimal when $flops(A \cdot B) > nnz + n$.

Input: A, B, C
for $j = 1$ to n **do**
  for k where $b_{k,j} \neq 0$ **do**
    $C(:, j) := C(:, j) + A(:, k) \cdot b_{kj}$
  end for
end for



Sparse Matrix Multiplication

- Consider matrices with random sparsity: the adjacency matrices of Erdős - Rényi(n,d) graphs - ER(n,d).
- Let A and B be $n \times n$ ER(n,d) matrices. We assume $d \ll n$. Then:
  - Each entry in A and B is nonzero with probability d/n.
  - The expected number of nonzeros in A and B is dn.
  - The expected number of scalar multiplications in AB is $(d^2/n^2) \cdot n^3 = d^2 n$.
  - The expected number of nonzeros in C is $d^2 n(1 - o(1))$.

Results:

- Lower bounds on communication, improved (higher) with respect to ones derived from dense linear algebra
- Optimal algorithms

Communication bounds for matrix multiplication

Given: A,B of size $n \times n$, local memory of size M, P processors, the lower bound on volume of communication for computing $A \cdot B$ on P processors is:

Dense Classic (cubic flops)

Memory dependent

$$\Omega\left(\frac{n^3}{M^{3/2}} \cdot \frac{M}{P}\right)$$

Memory independent

$$\Omega\left(\frac{n^2}{P^{2/3}}\right)$$

Erdős - Rényi(n,d)

Extension of lower bound for dense matrices to sparse matrices

$$\Omega\left(\frac{\#flops}{M^{3/2}} \cdot \frac{M}{P}\right) = \Omega\left(\frac{d^2 n}{P\sqrt{M}}\right) \leq \Omega\left(\sqrt{\frac{d^2 n}{P}}\right)$$

No algorithm attains this bound.

Communication bounds for Erdős - Rényi(n,d)

Extension of lower bound for dense matrices to sparse matrices

$$\Omega\left(\frac{\#flops}{M^{3/2}} \cdot \frac{M}{P}\right) = \Omega\left(\frac{d^2 n}{P\sqrt{M}}\right) \leq \Omega\left(\sqrt{\frac{d^2 n}{P}}\right)$$

No algorithm attains this bound.

New bound from [Ballard et al., 2013]

$$\Omega\left(\min\left(\frac{dn}{\sqrt{P}}, \frac{d^2 n}{P}\right)\right) = \Omega\left(\frac{dn}{\sqrt{P}} \min\left(1, \frac{d}{\sqrt{P}}\right)\right)$$

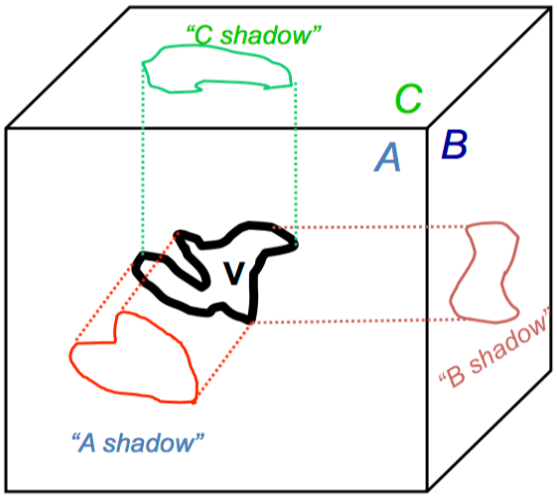
- With some assumptions. Which bound applies depends on ratio $d/\sqrt{P}$.
- Improvement factor of $\sqrt{M} \cdot \max\{1, \sqrt{P}/d\}$ with respect to previous bound
- Two algorithms attain this bound: recursive and 3D iterative.

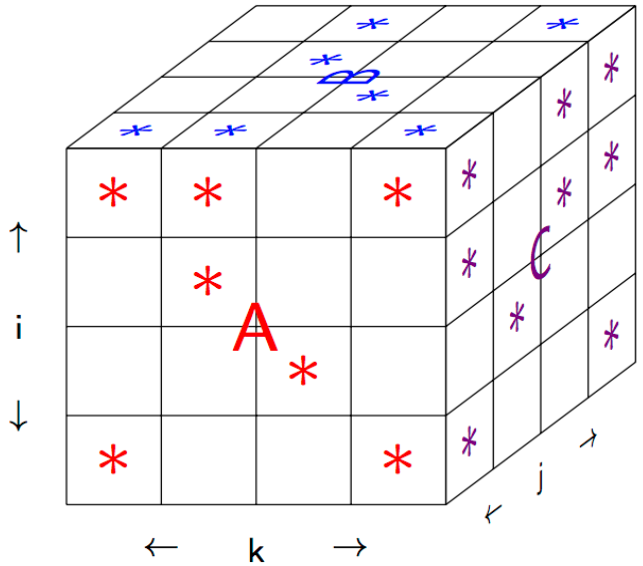
Geometric view of the computation

Computation cube for matrix multiply, with a specified subset of voxels:

- A face for each input/output matrix.
- Voxel (i, j, k) corresponds to the multiplication $a_{ik} \cdot b_{kj}$.
- Loomis & Whitney (1949): Volume of 3D set V satisfies:

$$V \leq (\text{area (A shadow)} \cdot \text{area(B shadow)} \cdot \text{area(C shadow)})^{1/2}$$





Source figure: G. Ballard

Communication bounds for Erdős-Rényi(n,d)

Assumptions:

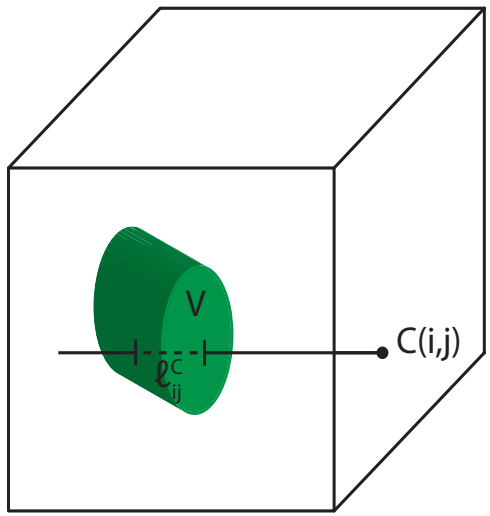
- Sparsity independent algorithms
- Input and output are sparse: $d \leq \sqrt{n}$
- The algorithm is load balanced

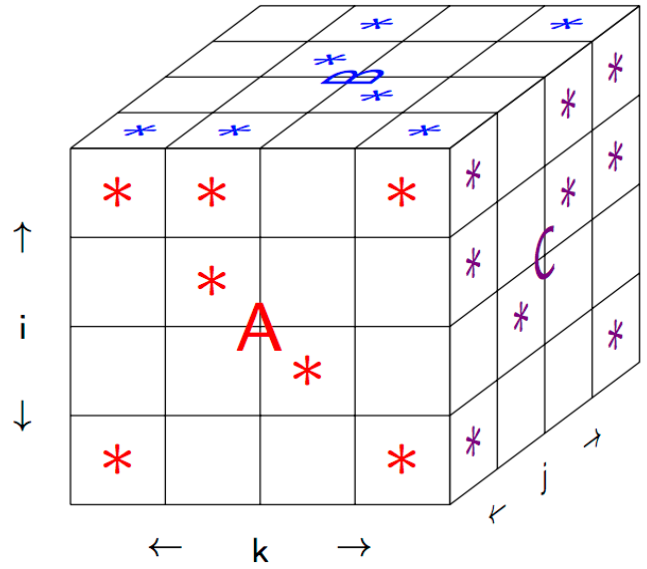
Sparsity independent algorithms:

- Assignement of entries of A, B, C to processors is independent of sparsity pattern of input/output matrices.
- Assignement of computation voxels to processors is independent of sparsity pattern of input/output matrices.
- All known algorithms are sparsity-independent

Lower bound - intuition of the proof

Idea: how many useful flops can be performed by using S inputs/outputs (similar to the dense case).





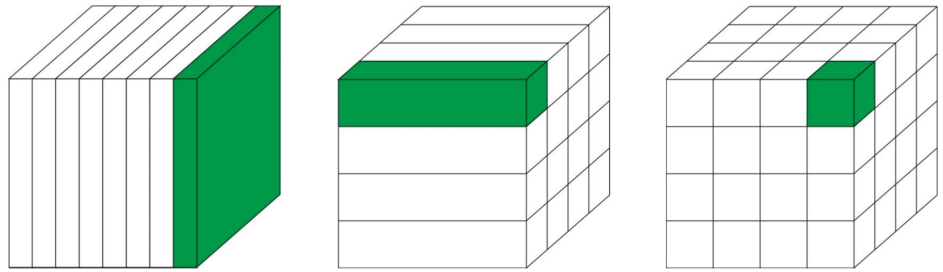
- Distinguish between input shadows and output shadow
- Given a shadow, is it stored on only one processor
- Given an internal grid point, does it correspond to a non-zero: use sparsity independence and randomness

Partitioning the work cube to processes

- Our bounds apply to all sparsity independent algorithms.
- We analyze algorithms that assign contiguous brick-shaped sets of voxels to each processor.

With correctly chosen data distributions:

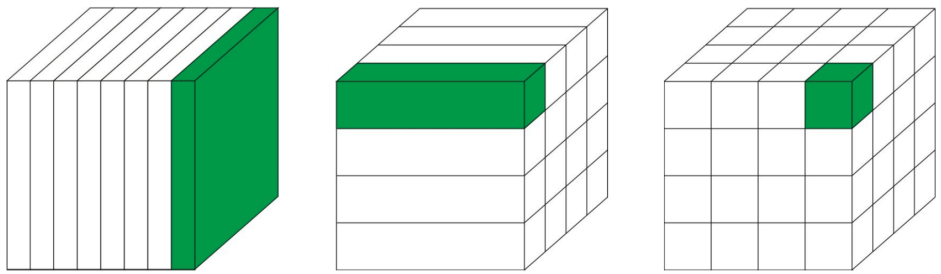
- 1D algorithms communicate entries of only one of the three matrices
- 2D algorithms communicate entries of two of the three matrices
- 3D algorithms communicate entries of all three matrices



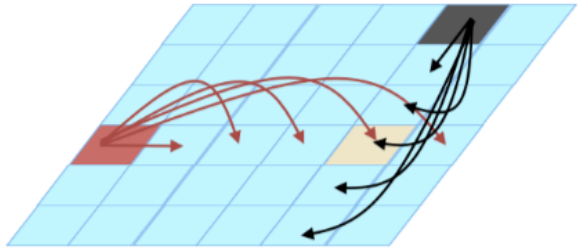
Partitioning the work cube to processes

Details:

- 1D algorithms communicate entries of only one of the three matrices:
  - Block Row: partition A, B, C on procs in a block row fashion. Shift block rows of B around a ring of processors. $W = dn, \quad S = P$
  - Improved Block Row: each proc gathers all required rows of B at once. Point to point communication: $W = d^2n/P, \min\{P, dn/P\}$
  - Outer product: Partition A in block cols, B in block rows, compute outer product, all-to-all to gather C. $W = d^2n/P, \quad S = \log P$
- 2D algorithms communicate entries of two of the three matrices: 2D Sparse SUMMA.
- 3D algorithms communicate entries of all three matrices: 3D Sparse SUMMA, 3D Recursive.



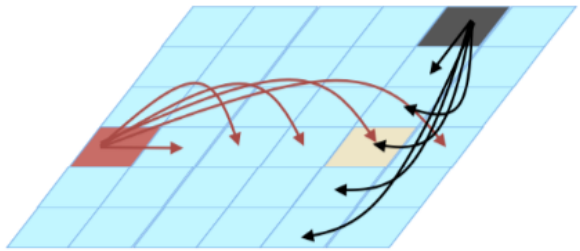
2D Summa



- Process grid $\sqrt{P} \times \sqrt{P}$ (in general does not have to be square)
- $C(i,j)$ is $n/\sqrt{P} \times n/\sqrt{P}$ submatrix of C on processor P_{ij}
- $A(i,k)$ is $n/\sqrt{P} \times b$ submatrix of A on processor P_{ik}
- $B(k,j)$ is $b \times n/\sqrt{P}$ submatrix of B on processor P_{kj}
- $C(i,j) = C(i,j) + \sum_k A(i,k) \cdot B(k,j)$
- To minimize communication, choose b close to $n/\sqrt{P}$ (as in the figure)

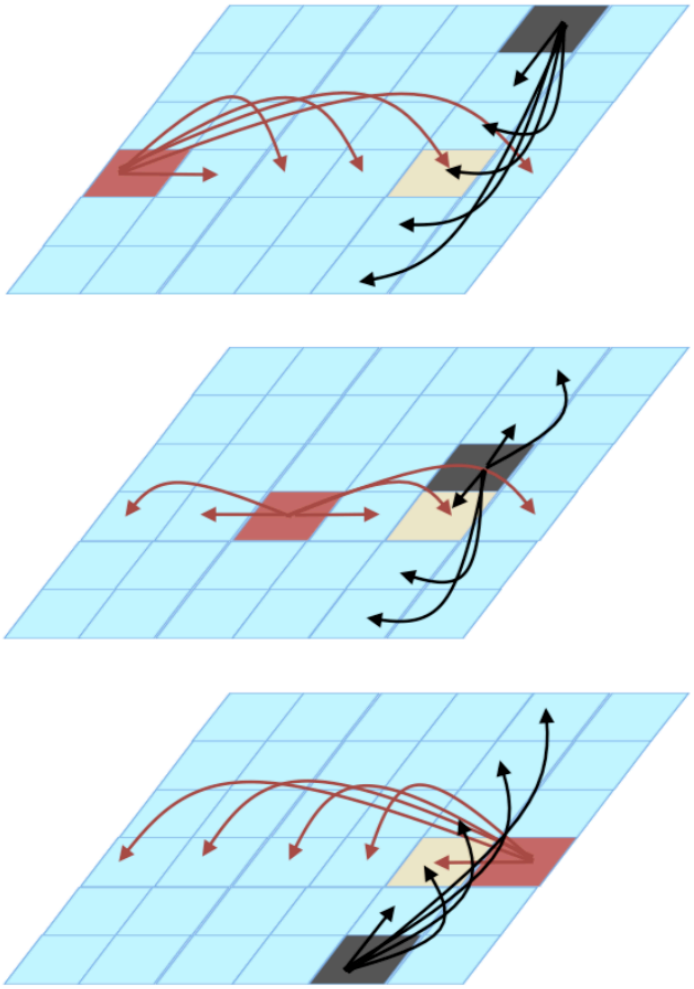
2D Summa (with $b = n/\sqrt{P}$)

- $C(i,j) = C(i,j) + \sum_k A(i,k) \cdot B(k,j)$



1: **for** k=1 to $\sqrt{P}$ **do**
2: **for** all $i,j = 1 \ldots \sqrt{P}$ **do**
3: P_{ik} broadcasts $A(i,k)$ along its row of processors $P_{i,:}$
4: P_{kj} broadcasts $B(k,j)$ along its column of processors $P_{:,j}$
5: $C(i,j) = C(i,j) + A(i,k) \cdot B(k,j)$
6: **end for**
7: **end for**

Dense 3D Summa

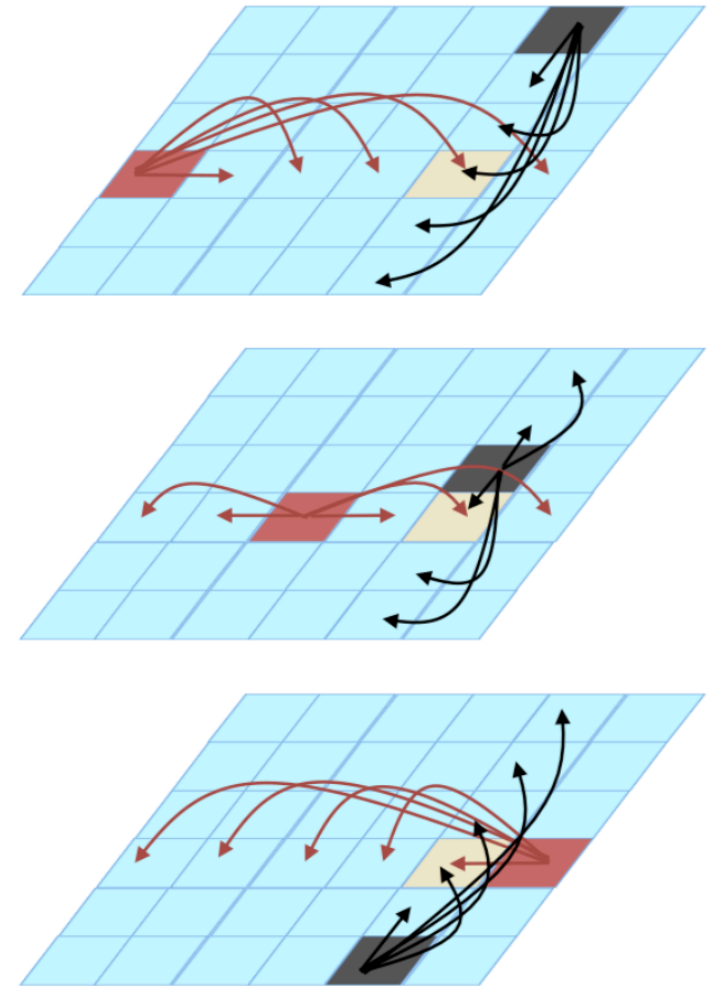


- Assume each processor can store cn^2/P data, $c > 1$
- Process grid: $\sqrt{P/c} \times \sqrt{P}/c \times c$

For $c = 3$

1. Layer 1 stores only $A(:, 1:2)$ and $B(1:2, :)$
2. Layer 2 stores only $A(:, 3:4)$ and $B(3:4, :)$
3. Layer 3 stores only $A(:, 5:6)$ and $B(5:6, :)$

Dense 3D Summa



Process grid: $\sqrt{P/c} \times \sqrt{P}/c \times c$

1. P_{ij0} broadcasts $A(i,j)$ and $B(i,j)$ to P_{ijt}
2. Processors at layer t perform $1/c$-th of SUMMA, i.e. 1/c-th of $\sum_k A(i,k) * B(k,j)$
3. Number of steps is $\sqrt{P/c^3}$
4. At each step, broadcast a block of A and a block of B along rows / columns of the face $\sqrt{P/c} \times \sqrt{P}/c$ process grid
5. Sum-reduce partial sums $\sum_k A(i,k) \cdot B(k,j)$ along t-axis so P_{ij0} owns $C(i,j)$

$$W = O(n^2/\sqrt{Pc}), \quad S = O(\sqrt{P/c^3} + \log c)$$

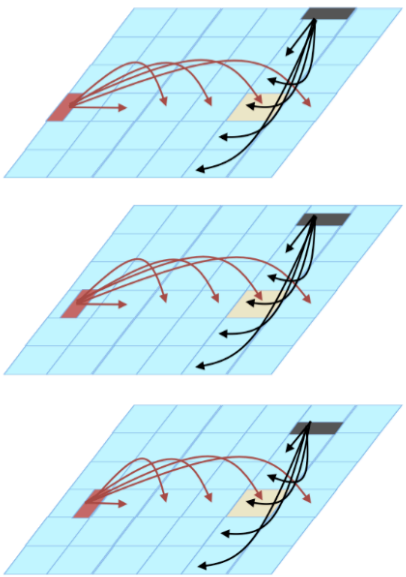
Sparse 3D Summa

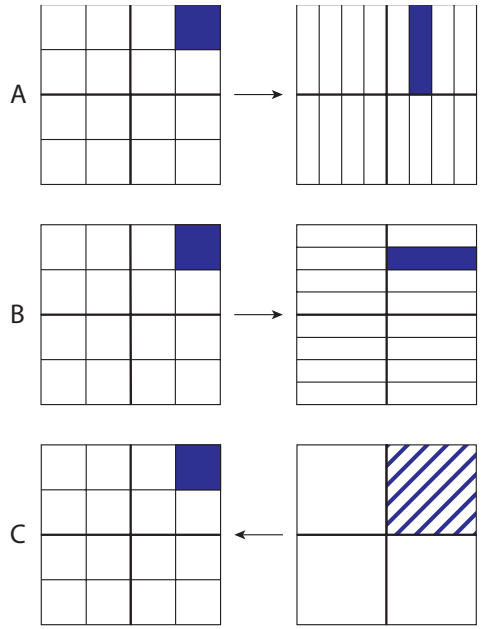
Process grid: $\sqrt{P/c} \times \sqrt{P/c} \times c$, A, B distributed over $\sqrt{P} \times \sqrt{P}$ procs.

- Distribute A and B on c layers: only $1/c$-th of columns of A and rows of B need to be distributed.
  E.g. for A, each proc owns a block of size $n/\sqrt{P/c} \times n/\sqrt{P/c^3}$.
  All-to-all operations performed by blocks of $\sqrt{c} \times \sqrt{c}$ procs.

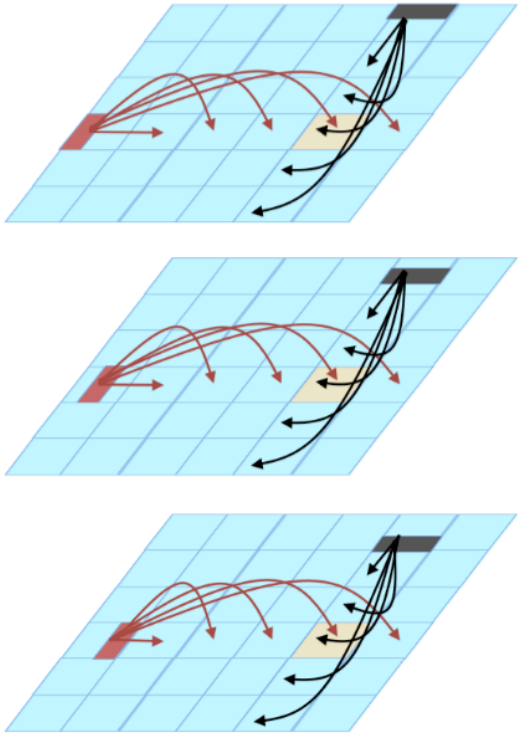
$$W = O(\frac{dn}{P} \cdot \log c), \quad S = O(\log c)$$

Example for $2 \times 2 \times 4$ grid, $c = 4$.





Sparse 3D Summa (contd)



- Processors collect all entries of A and B they need allgather operation among $\sqrt{P/c}$ procs.

$$W = O(\sqrt{P/c} \cdot \frac{dn}{P}) = O(dn/\sqrt{Pc}),$$
$$S = O(\log \sqrt{P/c})$$

- Reduce C on the first layer, and scatter it on all procs
  Sparse case: since each nonzero is contributed by a few flops, use instead gather + merge or all-to-all

$$W = O(\frac{d^2 n}{P} \cdot \log c), \quad S = O(\log c)$$

Optimizing c

Lower bound on communication:

$$\Omega\left(\min\left(\frac{dn}{\sqrt{P}}, \frac{d^2 n}{P}\right)\right)$$

If $d > \sqrt{P}$, then $d^2 n/P > dn/\sqrt{P}$.

Cost of sparse 3D Summa:

$$O\left(\frac{dn}{\sqrt{Pc}} + \frac{d^2 n}{P}\log c\right)$$

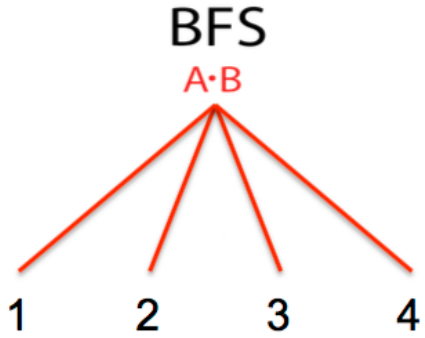
If $d > \sqrt{P}$, choose $c = 1$

If $d < \sqrt{P}$, choose $c = \Theta(P/d^2)$ to balance the two terms in the bandwidth cost.

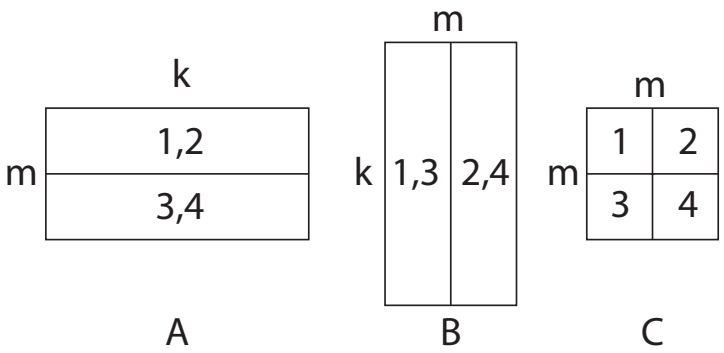
$\rightarrow$ Sparse Summa communication optimal by choosing $c = \min(1, P/d^2)$.
Remark: no increase in memory requirement, which remains $d^2 n/P$.

Recursive algorithm [Ballard et al., 2013]

Divides $A \cdot B$ into 4 sub-problems, each executed on $P/4$ processors.
While $P > 1$, pick the cheapest split

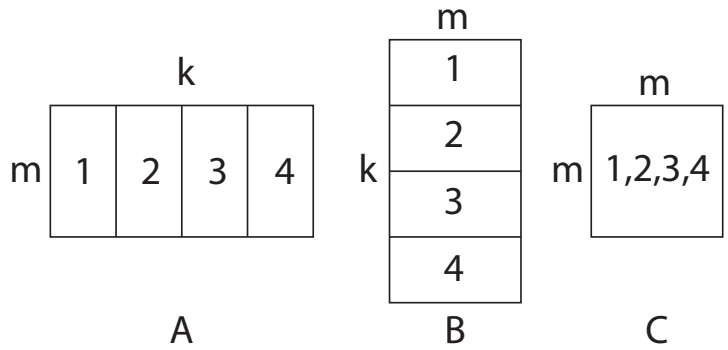


Split 1: Problem $m/2 \times k \times m/2$



Replicates A and B

Split 2: Problem $m \times k/4 \times m$

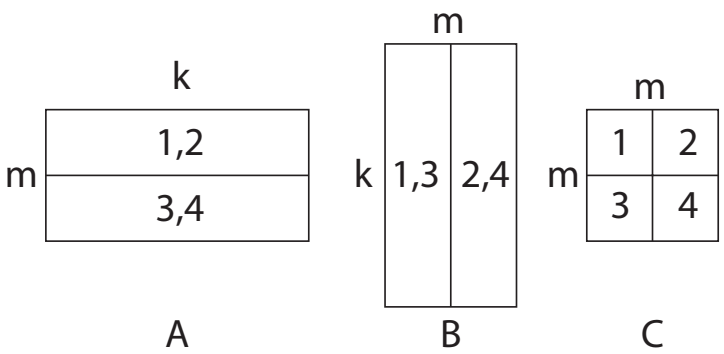


Redistributes and reduces C

Based on [Ballard, Demmel, Holtz, Lipshitz, Schwartz, SPAA'12

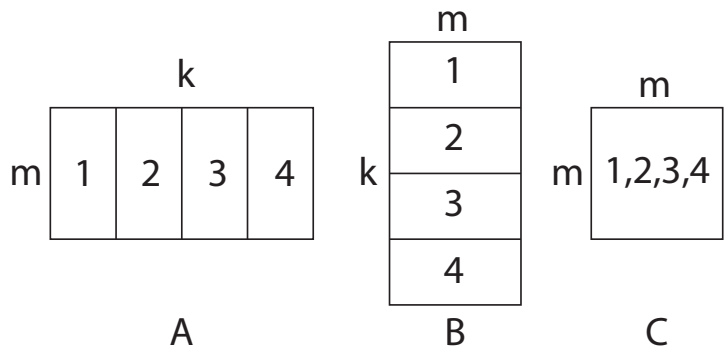
Recursive algorithm [Ballard et al., 2013]

Split 1: Problem $(m/2) \times k \times (m/2)$



Replicates A and B:
Allgather between disjoint pairs of procs
$W = O\left(\frac{dmk}{nP}\right), \quad S = O(1)$

Split 2: Problem $m \times k/4 \times m$



Redistributes and reduces C:
All-to-all between disjoint sets of 4 procs
$W = O\left(\frac{d^2m^2k}{n^2P}\right), \quad S = O(1)$

Recursive algorithm

- Algorithm: while $P > 1$, pick the cheapest split.
- Initially $m = k = n$, split 1 $O(dn/P)$ words is cheaper than split 2 $O(d^2 n/P)$.
- Split 1 cheaper for the first $\log_2 d$ steps.

Case 1: If $P \leq d^2$, Split 1 always cheapest:

$$W = \sum_{i=0}^{\log_4 P - 1} O\left(\frac{d(n/2^i)n}{nP/4^i}\right) = O\left(\frac{dn}{\sqrt{P}}\right), \quad S = O(\log P)$$

Recursive algorithm

Algorithm: while $P > 1$, pick the cheapest split
Case 2: If $P > d^2$, first $log_2 d$ steps use split 1, then use split 2.
After $log_2 d$ steps, subproblem has shape $n/d \times n \times n/d$ and P/d^2 procs.

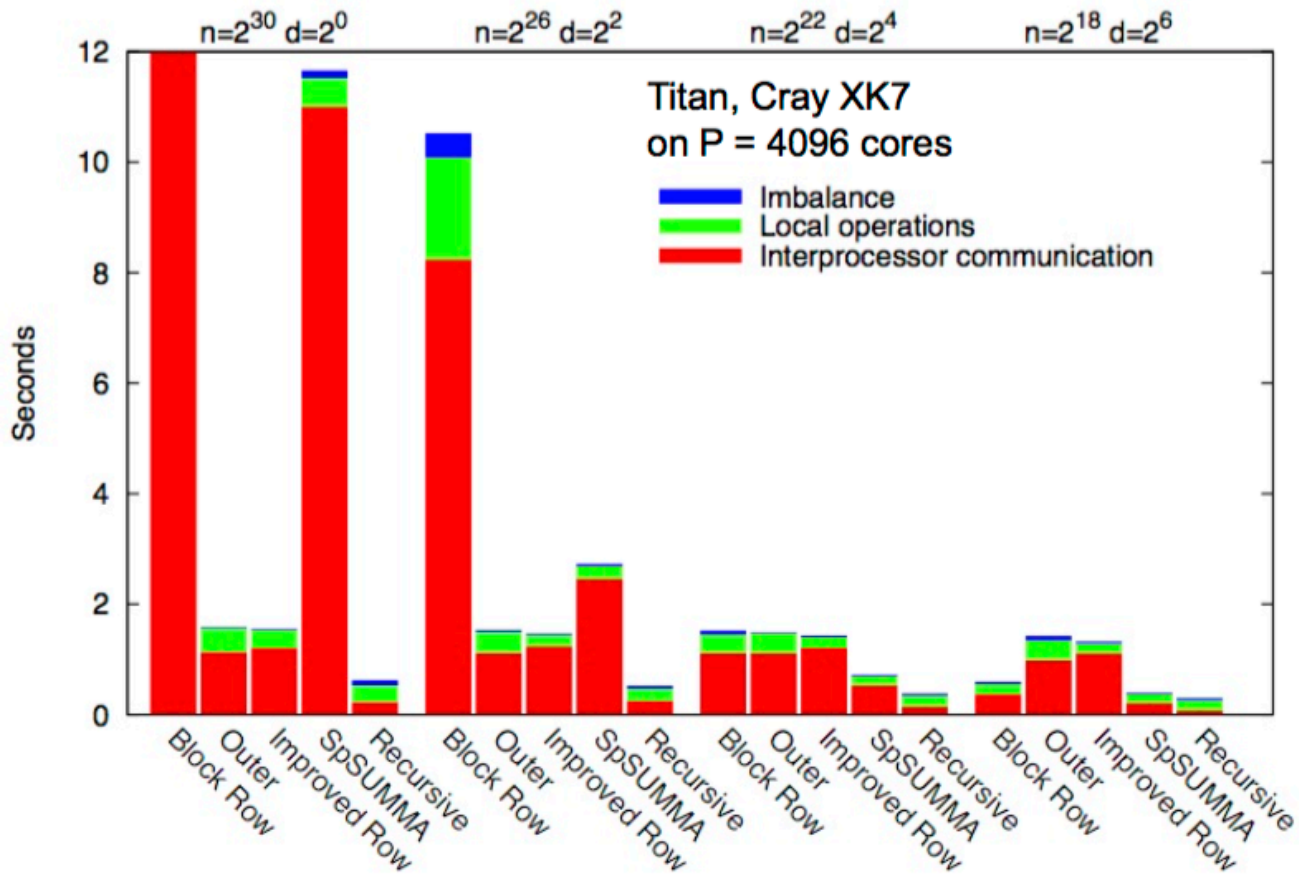
$$
\begin{aligned}
W &= \sum_{i=0}^{\log_2 d - 1} O\left(\frac{d(n/2^i)n}{nP/4^i}\right) + \sum_{i=\log_2 d}^{\log_4 P} O\left(\frac{d^2 n}{P}\right) = O\left(\frac{d^2 n}{P} \log \frac{P}{d^2}\right), \\
S &= O(\log P)
\end{aligned}
$$

- Matches the lower bound of $\Omega\left(\min\left(\frac{dn}{\sqrt{P}}, \frac{d^2 n}{P}\right)\right)$ up to log factor.
- Possible layout: A in block column layout, B in block-row layout, C has blocks of size $n/d \times n/d$, each distributed on a different $\lceil P/d^2 \rceil$ of the processors.
- No need to use DFS, since BFS uses only a constant factor extra memory.

Results for Erdős - Rényi Graph ER(n,d)

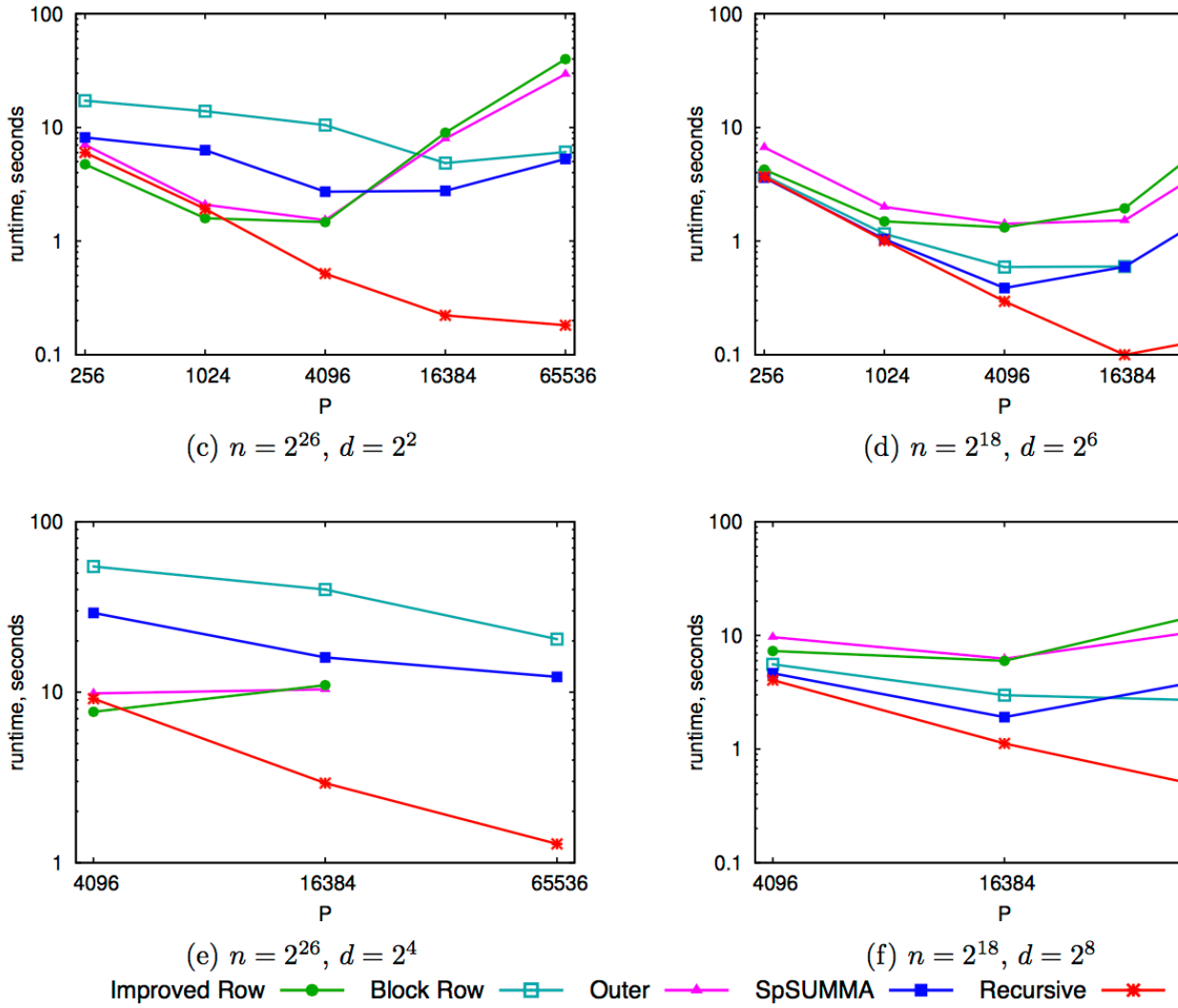
- Machine used: Titan, Cray XK7 from ORNL: 18,688 nodes, each node has 32GB of RAM, a 16 core AMD Opteron 6274 processor, and an Nvidia K20 GPU (not used).
- Experiments: use one core per process (16 MPI processes per node).

Results for Erdős - Rényi Graph



- For each case, expected number of nonzeros in the output is $d^2 n = 2^{30}$.
- For $n = 2^{30}, d = 1$, comm time = 39.3 secs, local operations = 11.9 secs and imbalance = 0.98 secs.

Results for Erdős - Rényi Graph - strong scaling



(c) $n = 2^{26}$, $d = 2^2$

(d) $n = 2^{18}$, $d = 2^6$

(e) $n = 2^{26}$, $d = 2^4$

(f) $n = 2^{18}$, $d = 2^8$

- In general, recursive algorithm outperforms all others
- Among the others: when $d < \sqrt{P}$, Outer product and Improved block row perform best when $d > \sqrt{P}$, sparse SUMMA performs best

Plan

Introduction

Sparse Matrix Matrix multiplication

Sparse Cholesky factorization for SPD matrices

- Combinatorial tools: undirected graphs, elimination trees
- Parallel Cholesky factorization
- Lower bounds for model problems

Graphs: All pairs shortest path

SPD matrices and Cholesky factorization

A is symmetric and positive definite (SPD) if

- $A = A^T$,
- all its eigenvalues are positive,
- or equivalently, A has a Cholesky factorization, $A = LL^T$.

Some properties of an SPD matrix A

- There is no need to pivot for accuracy (just performance) during the Cholesky factorization.
- For any permutation matrix P, PAP^T is also SPD.

Sparse Cholesky factorization

Algebra:

$$
\begin{aligned}
A &= \begin{pmatrix} a_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} \sqrt{a_{11}} & \\ A_{21}./\sqrt{a_{11}} & I \end{pmatrix} \cdot \begin{pmatrix} \sqrt{a_{11}} & A_{21}^T./\sqrt{a_{11}} \\ & A_{22}^s \end{pmatrix} \\
&= \begin{pmatrix} \sqrt{a_{11}} & \\ A_{21}./\sqrt{a_{11}} & L_{22} \end{pmatrix} \cdot \begin{pmatrix} \sqrt{a_{11}} & A_{21}^T./\sqrt{a_{11}} \\ & L_{22}^T \end{pmatrix}, \quad \text{where} \\
& A_{22}^s = A_{22} - (A_{21}./\sqrt{a_{11}}) \cdot (A_{21}^T./\sqrt{a_{11}})
\end{aligned}
$$

Algorithm:

for $k = 1 : n-1$ **do**
  $a_{kk} = \sqrt{a_{kk}}$
  /* $factor(k)$ */
  for $i = k+1 : n$ st $a_{ik} \neq 0$ **do**
    $a_{ik} = a_{ik}/a_{kk}$
  end for
  for $i = k+1 : n$ st $a_{ik} \neq 0$ **do**
    $update(k, i)$
    for $j = i : n$ st $a_{kj} \neq 0$ **do**
      $a_{ij} = a_{ij} - a_{ik}a_{jk}$
    end for
  end for
end for

$L + L^T =$

	1	2	3	4	5	6	7	8	9
1	x	x		x					
2	x	x	x	x	x				
3		x	x	x	x	x			
4	x	x	x	x	x		x		
5		x	x	x	x	x	x	x	
6			x		x	x	x	x	x
7				x	x	x	x	x	
8					x	x	x	x	x
9						x		x	x

Filled graph $G^+(A)$

- Given $G(A) = (V, E)$, $G^+(A) = (V, E^+)$ is defined as:
  there is an edge $(i, j) \in G^+(A)$ iff there is a path from i to j in $G(A)$ going through lower numbered vertices.
- Definition holds also for directed graphs (LU factorization).

- $G(L + L^T) = G^+(A)$, ignoring cancellations.
- $G^+(A)$ is chordal (every cycle of length at least four has a chord, an edge connecting two non-neighboring nodes).
- Conversely, if $G(A)$ is chordal, then there is a perfect elimination order, that is a permutation P such that $G(PAP^T) = G^+(PAP^T)$.

- References: [Parter, 1961, Rose, 1970, Rose and Tarjan, 1978]

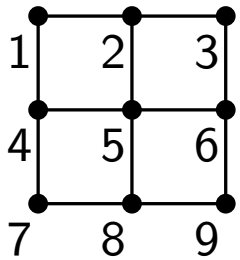
Filled graph $G^+(A)$

$A =$

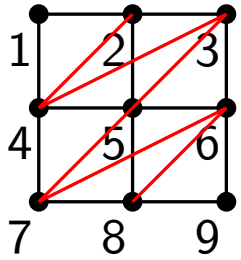
	1	2	3	4	5	6	7	8	9
1	x	x		x					
2	x	x	x		x				
3		x	x			x			
4	x			x	x		x		
5		x		x	x	x		x	
6			x		x	x			x
7				x			x	x	
8					x		x	x	x
9						x		x	x

$L + L^T =$

	1	2	3	4	5	6	7	8	9
1	x	x		x					
2	x	x	x	x	x				
3		x	x	x	x	x			
4	x	x	x	x	x		x		
5		x	x	x	x	x	x	x	
6			x		x	x	x	x	x
7				x	x	x	x	x	
8					x	x	x	x	x
9						x		x	x



$G(A)$



$G^+(A)$

Steps of sparse Cholesky factorization

1. Order rows and columns of A to reduce fill-in

2. Symbolic factorization: based on eliminaton trees
   - Compute the elimination tree (in nearly linear time in $nnz(A)$)
   - Allocate data structure for L
   - Compute the nonzero structure of the factor L, in $O(nnz(L)$

3. Numeric factorization
   - Exploit memory hierarchy
   - Exploit parallelism due to sparsity

4. Triangular solve

Order columns/rows of A

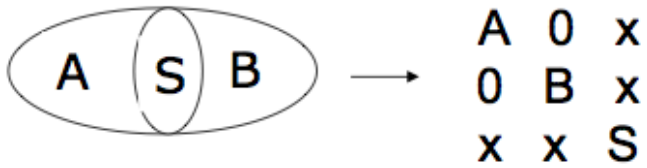
Strategies applied to the graph of A for Cholesky,
to the graph of $A^T A$ for LU with partial pivoting.

Local strategy: minimum degree [Tinney/Walker '67]

- Minimize locally the fill-in.
- Choose at each step (for 1 to n) the node of minimum degree.

Global strategy: graph partitioning approach

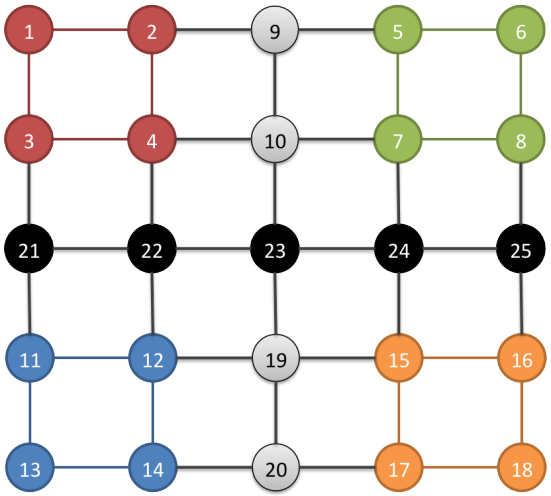
- Nested dissection [George, 1973]
  - First level: find the smallest possible separator S, order last
  - Recurse on A and B
- Multilevel schemes [Barnard/Simon '93, Hendrickson/Leland '95, Karypis/Kumar '95].

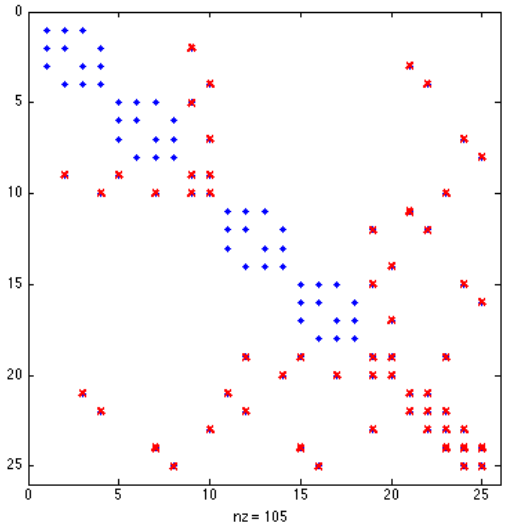


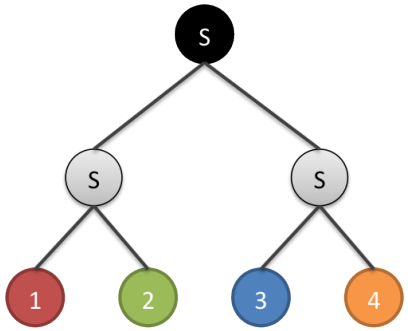
Nested dissection and separator tree

Separator tree:

- Combines together nodes belonging to a same separator, or to a same disjoint graph







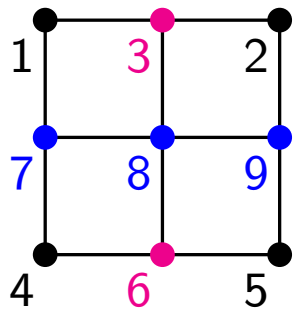
Some available packages:

- Metis, Parmetis (`http://glaros.dtc.umn.edu/gkhome/metis/metis/overview`)
- Scotch, Ptscotch (`www.labri.fr/perso/pelegrin/scotch/`)

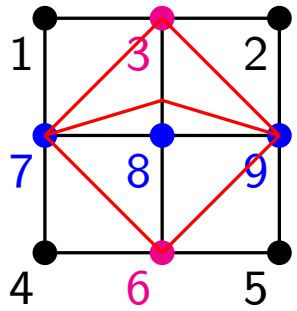
Nested dissection on our 9×9 structured matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \left(\begin{array}{ccc|ccc|ccc} x & & x & & & & x & & \\ & x & x & & & & & & x \\ \hline x & x & x & & & & & x & \\ \hline & & & x & & x & x & & \\ & & & & x & x & & & x \\ \hline & & & x & x & x & & x & \\ \hline x & & & x & & & x & x & \\ & & x & & & x & x & x & x \\ & x & & & x & & & x & x \end{array} \right) \end{array},$$

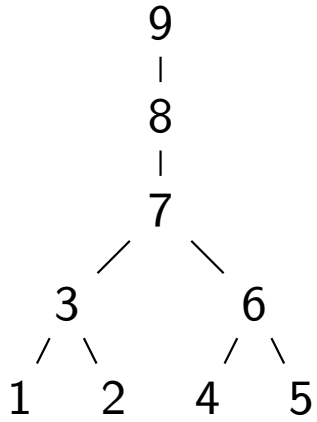
$$L + L^T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \left(\begin{array}{ccc|ccc|ccc} x & & x & & & & x & & \\ & x & x & & & & & & x \\ \hline x & x & x & & & & x & x & x \\ \hline & & & x & & x & x & & \\ & & & & x & x & & & x \\ \hline & & & x & x & x & x & x & x \\ \hline x & & x & x & & x & x & x & x \\ & & x & & & x & x & x & x \\ & x & x & & x & x & x & x & x \end{array} \right) \end{array}$$



$G(A)$



$G^+(A)$



$T(A)$

Elimination tree (etree)

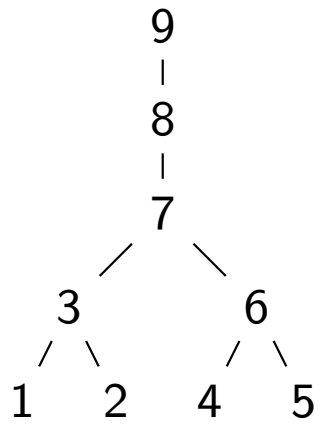
Definition ([Schreiber, 1982] and also [Duff, 1982])

Given $A = LL^T$, the etree $T(A)$ has the same node set as $G(A)$, and k is the parent of j iff

$$k = \min\{i > j : l_{ij} \neq 0\}$$

$L + L^T =$

	1	2	3	4	5	6	7	8	9
1	x		x				x		
2		x	x						x
3	x	x	x				x	x	x
4				x		x	x		
5					x	x			x
6				x	x	x	x	x	x
7	x		x	x		x	x	x	x
8			x			x	x	x	x
9		x	x		x	x	x	x	x



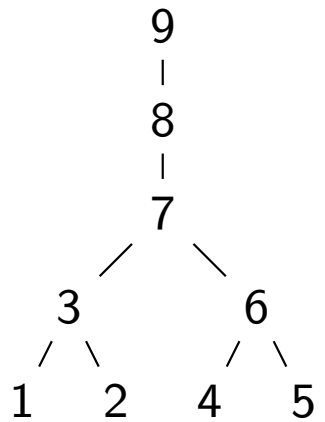
$T(A)$

Column dependencies and the elimination tree

- If $l_{jk} \neq 0$, then
  - $Factor(k)$ needs to be computed before $Factor(j)$.
  - k is an ancestor of j in $T(A)$.
- Columns belonging to disjoint subtrees can be factored independently.
- Topological orderings of $T(A)$ (that number children before their parent)
  - preserve the amount of fill, the flops of the factorization, the structure of $T(A)$

$L + L^T =$

	1	2	3	4	5	6	7	8	9
1	x		x				x		
2		x	x						x
3	x	x	x				x	x	x
4				x		x	x		
5					x	x			x
6				x	x	x	x	x	x
7	x		x	x		x	x	x	x
8			x			x	x	x	x
9		x	x		x	x	x	x	x



$T(A)$

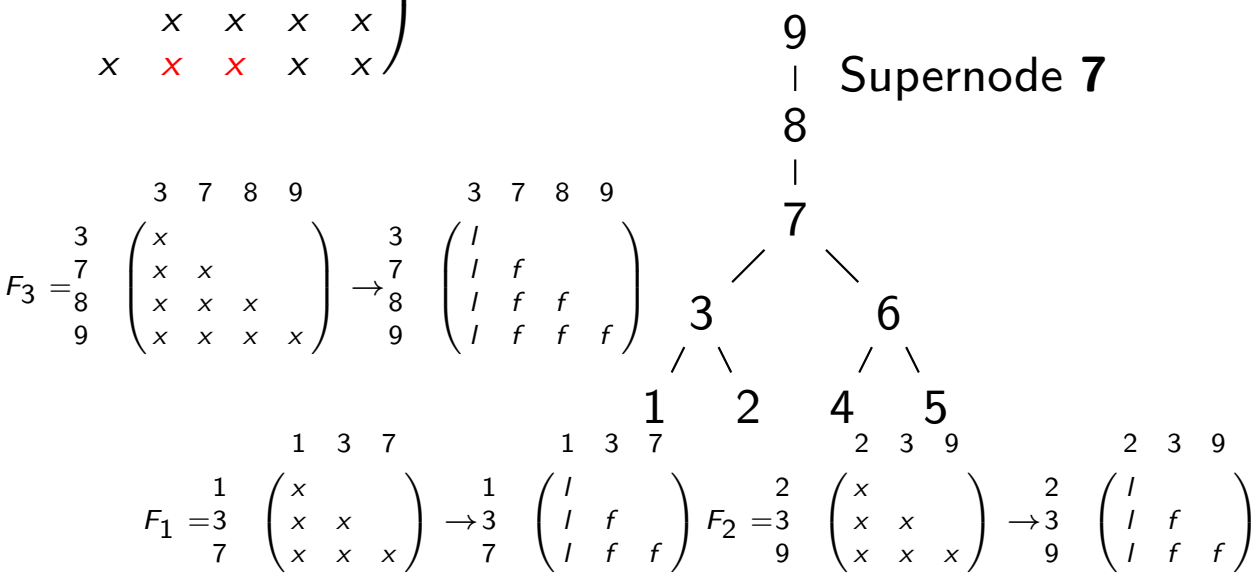
Numeric factorization - multifrontal approach

- Driven by the separator tree of A, a supernodal elimination tree.
- The Cholesky factorization is performed during a postorder traversal of the separator tree.
- At each node k of the separator tree:
  - A frontal matrix F_k is formed by rows and columns involved at step k of factorization:
    - rows that have their first nonzero in column k of A,
    - contribution blocks (part of frontal matrices) from children in $T(A)$.
  - The new frontal matrix is obtained by an extend-add operation.
  - The first rows/columns of F_k corresponding to supernode k are factored.

Numeric factorization - an example

$$L + L^T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} x & & x & & & & x & & \\ & x & x & & & & & & x \\ x & x & x & & & & x & x & x \\ & & & x & & x & x & & \\ & & & & x & x & & & x \\ & & & x & x & x & x & x & x \\ x & & x & x & & x & x & x & x \\ & & x & & & x & x & x & x \\ & x & x & & x & x & x & x & x \end{pmatrix} \end{array}$$

$$F_{\{7,8,9\}} = \begin{array}{c} \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccc} 7 & 8 & 9 \end{array} \\ \begin{pmatrix} x & & \\ x & x & \\ x & x & x \end{pmatrix} \end{array} \rightarrow \begin{array}{c} \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccc} 7 & 8 & 9 \end{array} \\ \begin{pmatrix} l & & \\ l & l & \\ l & l & l \end{pmatrix} \end{array}$$



$$F_3 = \begin{array}{c} \\ 3 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{cccc} 3 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} x & & & \\ x & x & & \\ x & x & x & \\ x & x & x & x \end{pmatrix} \end{array} \rightarrow \begin{array}{c} \\ 3 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{cccc} 3 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} l & & & \\ l & f & & \\ l & f & f & \\ l & f & f & f \end{pmatrix} \end{array}$$

$$F_1 = \begin{array}{c} \\ 1 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 3 & 7 \end{array} \\ \begin{pmatrix} x & & \\ x & x & \\ x & x & x \end{pmatrix} \end{array} \rightarrow \begin{array}{c} \\ 1 \\ 3 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 3 & 7 \end{array} \\ \begin{pmatrix} l & & \\ l & f & \\ l & f & f \end{pmatrix} \end{array} \qquad F_2 = \begin{array}{c} \\ 2 \\ 3 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 9 \end{array} \\ \begin{pmatrix} x & & \\ x & x & \\ x & x & x \end{pmatrix} \end{array} \rightarrow \begin{array}{c} \\ 2 \\ 3 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 3 & 9 \end{array} \\ \begin{pmatrix} l & & \\ l & f & \\ l & f & f \end{pmatrix} \end{array}$$

Notation used for frontal matrices F_k:

- x - elements obtained by the extend-add operation,
- l - elements of L computed at node k, f - elements of frontal matrix that will be passed to parent of node k.

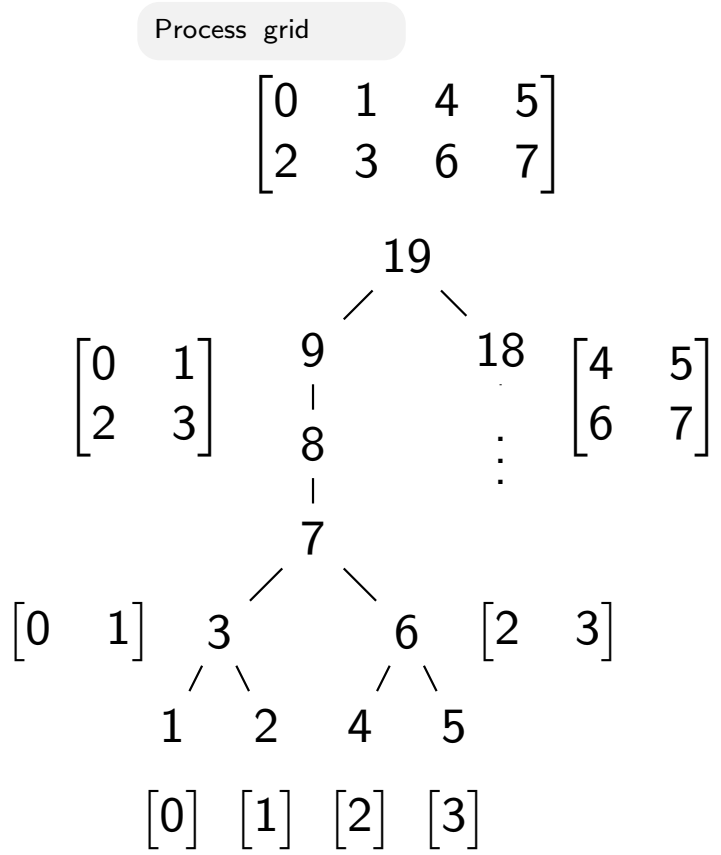
Numeric factorization - PSPASES [Gupta et al., 1995]

- Based on subtree to subcube mapping [George et al., 1989] applied on the separator tree

Subtree to subcube mapping

1. Assign all the processors to the root.
2. Assign to each subtree half of the processors.
3. Go to Step 1 for each subtree which is assigned more than one processor.

The figure displays the process grid used by PSPASES.



Numeric factorization - PSPASES [Gupta et al., 1995]

- Subtree to subcube mapping and bitmask based cyclic distribution:

  Starting at the last level of the separator tree (bottom up traversal), let $i = 1$

  for each two consecutive levels $k, k-1$, based on value of i-th LSB of column/row indices

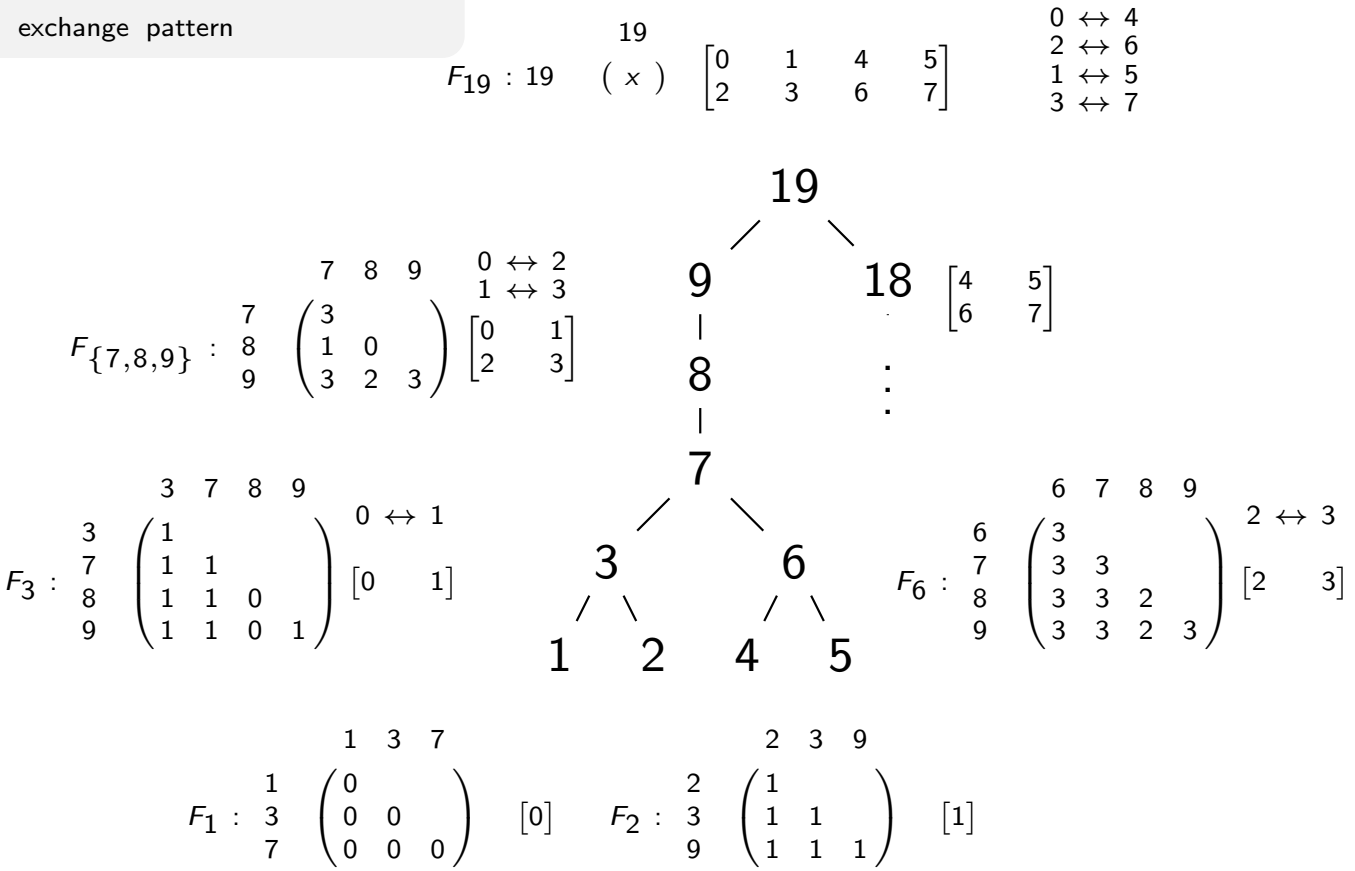
  - For level k:
    Map all even columns to subcube with lower processor numbers
    Map all odd columns to subcube with higher processor numbers
  - For level $k-1$:
    Map all even rows to subcube with lower processor numbers
    Map all odd rows to subcube with higher processor numbers
  - Let $i = i + 1$

PSPASES uses a bitmask based block-cyclic distribution.

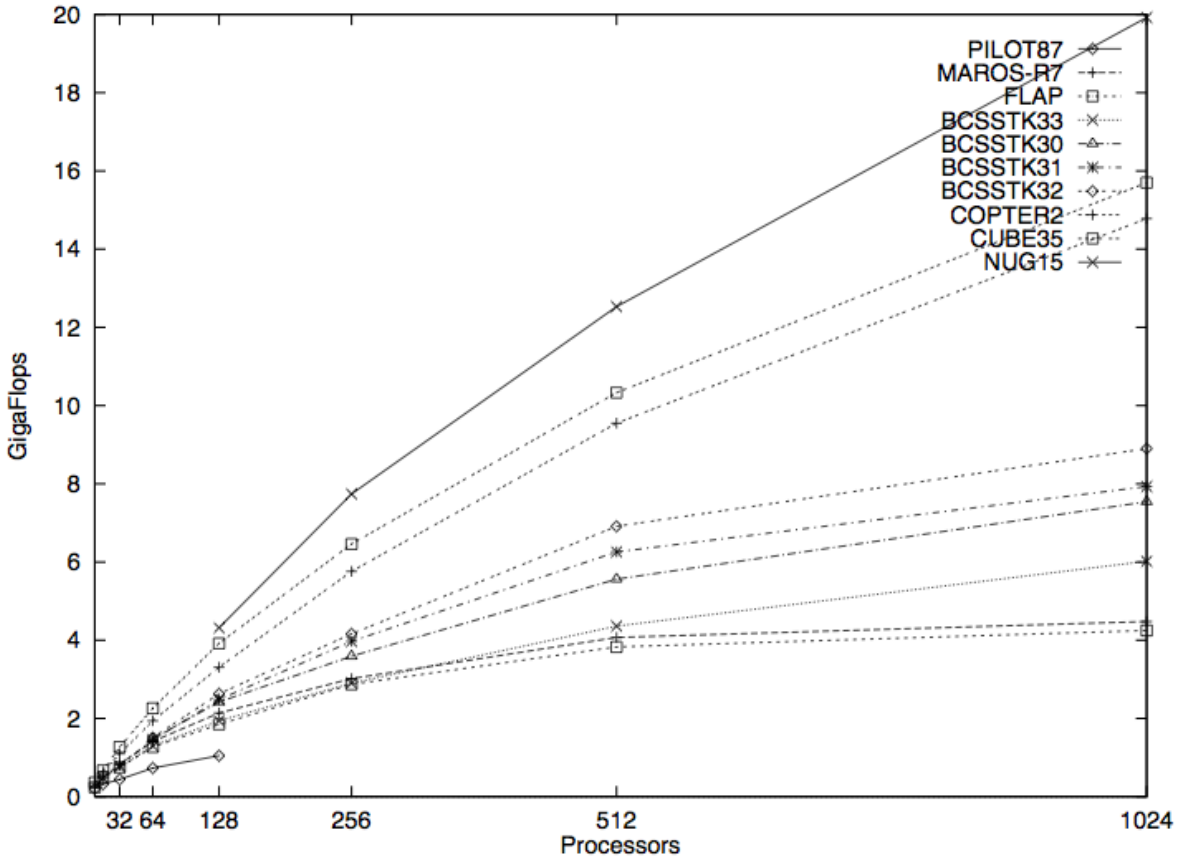
Numeric factorization - PSPASES [Gupta et al., 1995]

- Based on subtree to subcube mapping [George et al., 1989].
- Extend-add operation requires each processor to exchange half of its data with a corresponding processor from the other half of the grid.

Data distribution, process grid and data exchange pattern



Performance results on Cray T3D

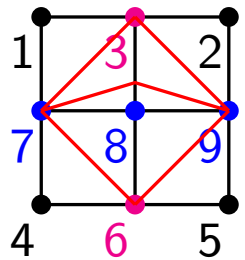


Results from [Gupta et al., 1995]

Lower bounds on communication for Cholesky

- Consider A of size $k^s \times k^s$ results from a finite difference operator on a regular grid of dimension $s \geq 2$ with k^s nodes.
- Its Cholesky L factor contains a dense lower triangular matrix of size $k^{s-1} \times k^{s-1}$.

$$L + L^T = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \end{array} \begin{array}{c} \begin{array}{ccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{array} \\ \begin{pmatrix} x & & x & & & & x & & \\ & x & x & & & & & & x \\ x & x & x & & & & x & x & x \\ & & & x & & x & x & & \\ & & & & x & x & & & x \\ & & & x & x & x & x & x & x \\ x & & x & x & & x & x & x & x \\ & & x & & & x & x & x & x \\ & x & x & & x & x & x & x & x \end{pmatrix} \end{array}$$



$G^+(A)$

- Computing the Cholesky factorization of the $k^{s-1} \times k^{s-1}$ matrix dominates the computation.

Lower bounds on communication

- This result applies more generally to matrix A whose graph $G = (V, E)$, $|V| = n$ has the following property for some l:
  - if every set of vertices $W \subset V$ with $n/3 \leq |W| \leq 2n/3$ is adjacent to at least l vertices in $V - W$,
  - then the Cholesky factor of A contains a dense $l \times l$ submatrix.

Lower bounds on communication

For the Cholesky factorization of a $k^s \times k^s$ matrix resulting from a finite difference operator on a regular grid of dimension $s \geq 2$ with k^s nodes:

$$\#words \geq \Omega\left(\frac{W}{\sqrt{M}}\right), \qquad \#messages \geq \Omega\left(\frac{W}{M^{3/2}}\right)$$

- Sequential algorithm
  - $W = k^{3(s-1)}/3$ and M is the fast memory size
- Work balanced parallel algorithm executed on P processors
  - $W = \frac{k^{3(s-1)}}{3P}$ and $M \approx nnz(L)/P$

Why / how PSPASES attains optimality

- For each node in the separator tree, the communication in the Cholesky factorization dominates the communication in the extend-add step.
- Optimal dense Cholesky factorization needs to be used for each multifrontal matrix ($n \times n$, P procs).
  - optimal block size - minimize communication while increasing flops by a lower order term

$$b = \frac{n}{\sqrt{P}} \log_2^{-2} \sqrt{P}$$

Optimal sparse Cholesky factorization

- Results for $n \times n$ matrix resulting from 2D and 3D regular grids.
- Analysis assumes local memory per processor is $M = O(n \log n/P)$- 2D case and $M = O(n^{4/3}/P)$- 3D case.

	PSPASES	PSPASES with optimal layout	Lower bound
2D grids			
# flops	$O\left(\frac{n^{3/2}}{P}\right)$	$O\left(\frac{n^{3/2}}{P}\right)$	$\Omega\left(\frac{n^{3/2}}{P}\right)$
# words	$O(\frac{n}{\sqrt{P}})$	$O\left(\frac{n}{\sqrt{P}} \log P\right)$	$\Omega\left(\frac{n}{\sqrt{P \log n}}\right)$
# messages	$O(\sqrt{n})$	$O\left(\sqrt{P} \log^3 P\right)$	$\Omega\left(\frac{\sqrt{P}}{(\log n)^{3/2}}\right)$
3D grids			
# flops	$O\left(\frac{n^2}{P}\right)$	$O\left(\frac{n^2}{P}\right)$	$\Omega\left(\frac{n^2}{P}\right)$
# words	$O(\frac{n^{4/3}}{\sqrt{P}})$	$O\left(\frac{n^{4/3}}{\sqrt{P}} \log P\right)$	$\Omega\left(\frac{n^{4/3}}{\sqrt{P}}\right)$
# messages	$O(n^{2/3})$	$O\left(\sqrt{P} \log^3 P\right)$	$\Omega\left(\sqrt{P}\right)$

Optimal sparse Cholesky factorization: summary

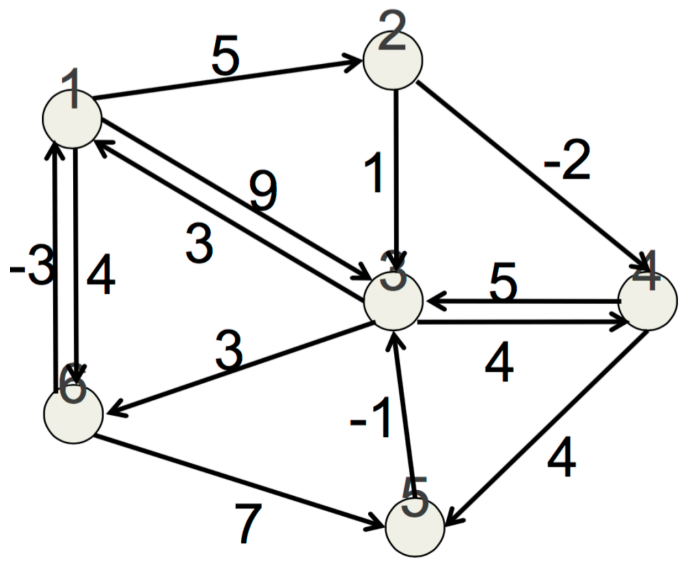
- PSPASES with an optimal layout attains the lower bound in parallel for 2D/3D regular grids:
  - Uses nested dissection to reorder the matrix
  - Distributes the matrix using the subtree to subcube algorithm
  - The factorization of every dense multifrontal matrix is performed using an optimal dense Cholesky factorization
- Sequential multifrontal algorithm attains the lower bound
  - The factorization of every dense multifrontal matrix is performed using an optimal dense Cholesky factorization

Plan

Preliminaries

Graph $G = (V, E)$ is formed by:

- a set of vertices V,
- a set of edges E.



- Edges can be directed or not, can have weights or not.
- A path from v_1 to v_n is formed by a sequence of edges $(v_1, v_2), \ldots (v_{n-1}, v_n)$. Its length is the sum of its weights.

Parallel graph algorithms

- Graph traversals: breadth-first search
- Single Source Shortest Path: Delta-stepping (Meyer and Sanders), randomized approach (Ullman and Yannakakis)
- All Pairs Shortest Path (APSP): Floyd-Warshall, Johnson (based on Dijkstra).
- Graph partitioning

Applications

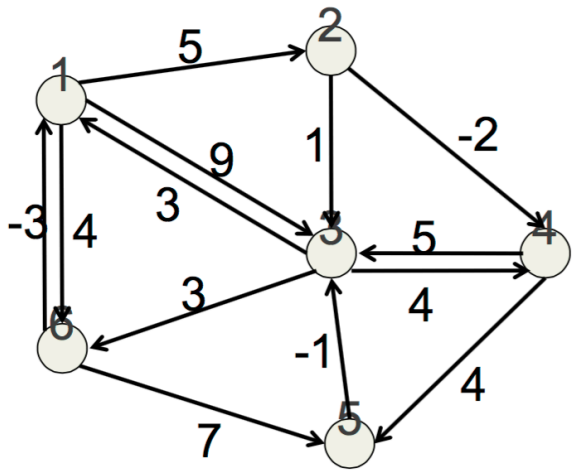
- Routing in transportation networks: compute point to point shortest paths
- Internet and WWW: web search, page rank, document classification and clustering
- Scientific computing: reorderings, graph partitioning, maximum matchings
- APSP: urban planning and simulation, datacenter network design, traffic routing, subroutine in Ullman and Yannakaki's BFS algorithm

All-pairs shortest paths

- Input: directed graphs with weights on edges
- APSP: find shortest paths between all reachable vertex pairs

Floyd-Warshall

for $i,j = 1 : n$, $d(i \to i) := 0$, $d(i \to j) := \infty$
for each edge (i,j)
  $d(i \to j) := w(i \to j)$, $\Pi(i,j) := i$
for k=1 to n **do**
  for $i = 1$ to n **do**
    for $j = 1$ to n **do**
      If $d(i \to k) + d(k \to j) < d(i \to j)$
        $d(i \to j) := d(i \to k) + d(k \to j)$
        $\Pi(i,j) := \Pi(k,j)$
    end for
  end for
end for



$$\begin{bmatrix} 0 & 5 & 9 & \infty & \infty & 4 \\ \infty & 0 & 1 & -2 & \infty & \infty \\ 3 & \infty & 0 & 4 & \infty & 3 \\ \infty & \infty & 5 & 0 & 4 & \infty \\ \infty & \infty & -1 & \infty & 0 & \infty \\ -3 & \infty & \infty & \infty & 7 & 0 \end{bmatrix}$$

- First step: computes the lengths of the paths between all pairs of vertices
- Second step: if required, path reconstruction
- Assume there is no negative cycle

All-pairs shortest paths

APSP problem corresponds to finding the matrix closure on the tropical $(min, +)$ semiring. In the semiring matrix multiplication (distance product)

- replace each multiply with an addition: compute length of a larger path from smaller paths or edges
- replace each add with a minimum operation: get the minimum path if there are multiple paths

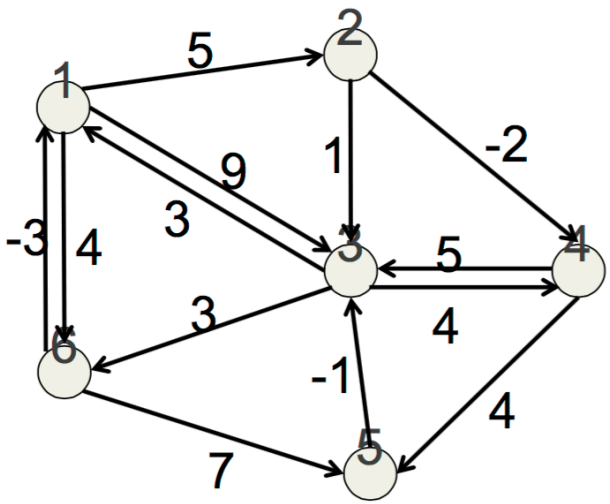
Assume for simplicity adjacency matrices of power of two dimension.

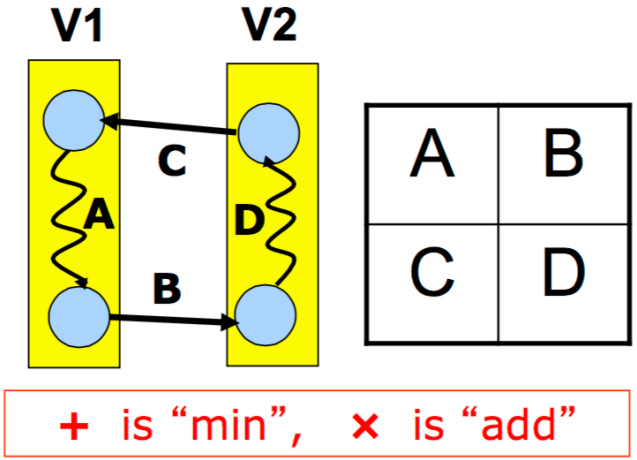
Cost first step: $O(n^3)$ additions and $O(n^2)$ min operations.
Cost path reconstruction: the Shortest-path tree can be calculated for each node in $O(|E|)$ time using $O(n)$ memory to store each tree.

All-pairs shortest paths

- **Floyd-Warshall** more suitable for denser graphs
- In parallel, Floyd-Warshall can be competitive even for sparser graphs, as for example on GPUs [Buluc et al., 2010].
- **Johnson's algorithm**, using for each vertex Dijkstra's single-source shortest path algorithm, requires less flops than Floyd-Warshall for sparse graphs: $O(|E| + |V| \log |V|)$ for each vertex.
- **Divide and Conquer APSP (DC-APSP)**
  - Idea presented in a proof by Aho et al showing equivalence between semiring matrix multiplication and APSP, later presented in papers by Tiskin, Park et al.
  - Faster than the 3 nested loops on GPUs [Buluc et al., 2010].

Divide and conquer APSP



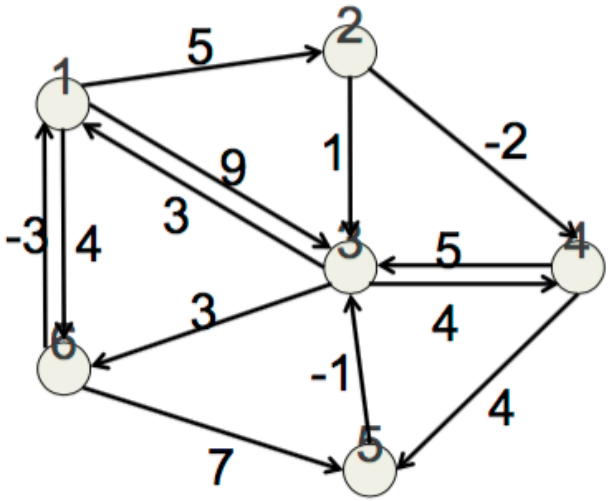


$$\begin{bmatrix} 0 & 5 & 9 & \infty & \infty & 4 \\ \infty & 0 & 1 & -2 & \infty & \infty \\ 3 & \infty & 0 & 4 & \infty & 3 \\ \infty & \infty & 5 & 0 & 4 & \infty \\ \infty & \infty & -1 & \infty & 0 & \infty \\ -3 & \infty & \infty & \infty & 7 & 0 \end{bmatrix}$$

DC-APSP(A,n)
A = DC-APSP(A,n/2);
B = AB; C = CA;
D = D + CB;
D = DC-APSP(D,n/2)
B = BD; C = DC;
A = A + BC;

Source slide: A. Buluç

Divide and conquer APSP



$$\left[\begin{array}{c|cc|ccc} 0 & 5 & 9 & \infty & \infty & 4 \\ \hline \infty & 0 & 1 & -2 & \infty & \infty \\ 3 & \infty & 0 & 4 & \infty & 3 \\ \hline \infty & \infty & 5 & 0 & 4 & \infty \\ \infty & \infty & -1 & \infty & 0 & \infty \\ -3 & \infty & \infty & \infty & 7 & 0 \end{array}\right]$$

DC-APSP(A,n)

find APSP in V_1

A = DC-APSP(A,n/2);

propagate paths from V_1 to V_2

B = AB;

propagate paths from V_2 to V_1

C = CA;

update paths in V_2

D = D + CB;

find APSP in V_2

D = DC-APSP(D,n/2)

find SP from V_1 to V_2, V_2 to V_1

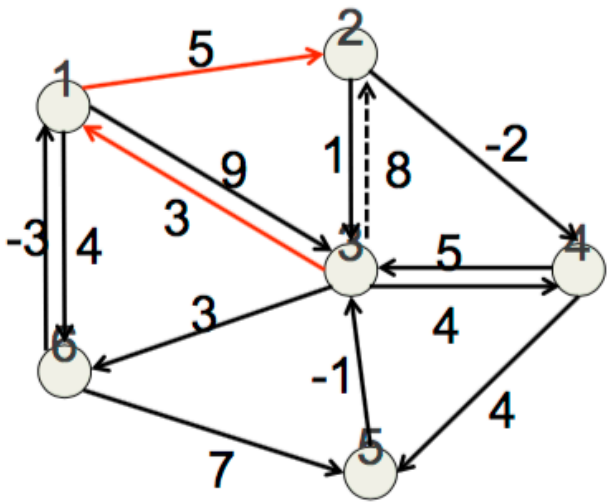
B = BD; C = DC;

find APSP in V_1

A = A + BC;

Source slide: A. Buluç

Divide and conquer APSP



$$\begin{bmatrix} \infty \\ 3 \end{bmatrix} \quad \begin{bmatrix} 5 & 9 \end{bmatrix} = \begin{bmatrix} \infty & \infty \\ 8 & 12 \end{bmatrix}$$

C **B**

The cost of 3-1-2 path

$$d(3,2) = d(3,1) + d(1,2) \xrightarrow{then} \Pi(3,2) = \Pi(1,2)$$

$$\begin{bmatrix} 0 & 5 & 9 & \infty & \infty & 4 \\ \infty & 0 & 1 & -2 & \infty & \infty \\ 3 & \mathbf{8} & 0 & 4 & \infty & 3 \\ \infty & \infty & 5 & 0 & 4 & \infty \\ \infty & \infty & -1 & \infty & 0 & \infty \\ -3 & \infty & \infty & \infty & 7 & 0 \end{bmatrix}$$

Distances

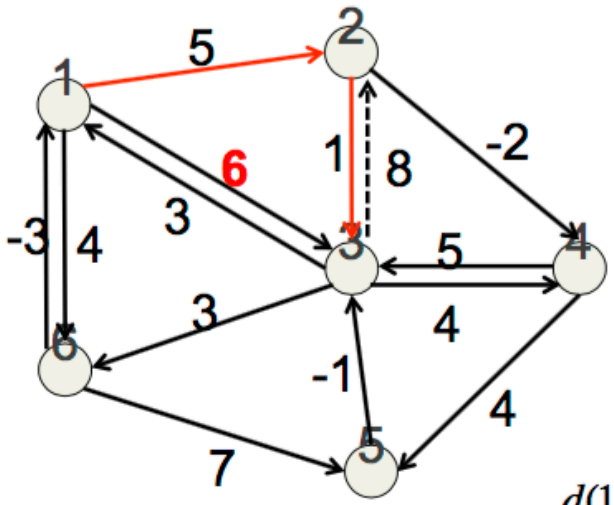
$$\Pi = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & \mathbf{1} & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 & 6 \end{bmatrix}$$

Parents

CB: update paths in V_2.

Source slide: A. Buluç.

Divide and conquer APSP



D = DC-APSP(D): no change

$$\begin{bmatrix} 5 & 9 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 8 & 0 \end{bmatrix} = \begin{bmatrix} 5 & 6 \end{bmatrix}$$

B **D**

Path: 1-2-3

$$d(1,3) = d(1,2) + d(2,3) \xrightarrow{then} \Pi(1,3) = \Pi(2,3)$$

$$\begin{bmatrix} 0 & 5 & \mathbf{6} & \infty & \infty & 4 \\ \infty & 0 & 1 & -2 & \infty & \infty \\ 3 & 8 & 0 & 4 & \infty & 3 \\ \infty & \infty & 5 & 0 & 4 & \infty \\ \infty & \infty & -1 & \infty & 0 & \infty \\ -3 & \infty & \infty & \infty & 7 & 0 \end{bmatrix}$$

Distances

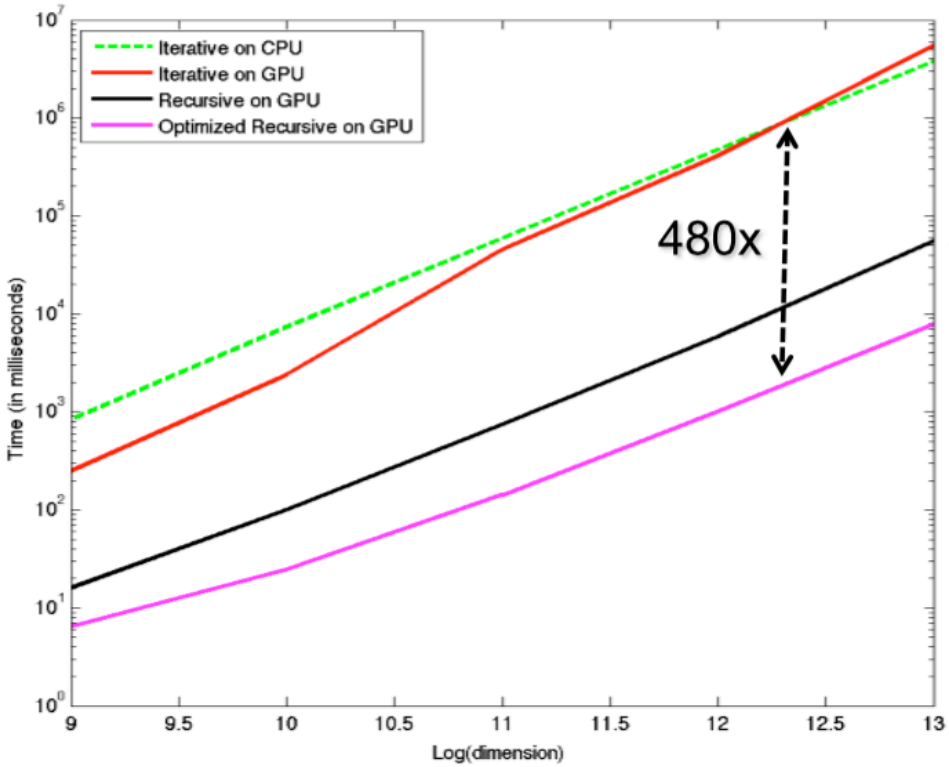
$$\Pi = \begin{bmatrix} 1 & 1 & \mathbf{2} & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 1 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 & 6 \end{bmatrix}$$

Parents

BD: find SP from V_1 to V_2.

Source slide: A. Buluç.

Divide and conquer APSP - results



Floyd-Warshall ported to GPU

Naïve recursive implementation

The right primitive (Matrix multiply)

A. Buluç, J. R. Gilbert, and C. Budak. Solving path problems on the GPU. Parallel Computing, 36(5-6):241 - 253, 2010.

- GPU: Nvidia GeForce 8800 Ultra
- Dense graph of at most 8192 vertices
- Matrix multiply optimized by modifying Volkov's code

Lower bounds for APSP

Semiring matrix multiplication has same computational dependency as classic matrix multiplication, and the bounds of [Hong and Kung, 1981], [Irony et al., 2004] apply [Solomonik et al., 2013]:

Memory dependent

$$W = \Omega\left(\frac{n^3}{M^{3/2}} \cdot \frac{M}{P}\right)$$
$$S(M) = \Omega\left(\frac{n^3}{P \cdot M^{3/2}}\right)$$

Memory independent

$$W = \Omega\left(\frac{n^2}{P^{2/3}}\right)$$
$$S = \Omega(\log P)$$

Latency bandwidth trade-off for DC-APSP

Divide and conquer APSP has dependencies similar to 2.5D LU factorization. Hence the same latency-bandwidth trade-off exists [Solomonik et al., 2013]:

If each processor stores $M = cn^2/P$ copies of data, then:

$$S \cdot W = \Omega(\sqrt{cP})$$

and if we want to decrease the bandwidth cost by a factor of $\sqrt{c}$ we obtain:

$$W = \Omega\left(\frac{n^2}{\sqrt{cP}}\right), \quad S = \Omega(\sqrt{cP})$$

Parallelizing APSP

Grid of processors: $\sqrt{P/c} \times \sqrt{P/c} \times c$

Floyd-Warshall: a communication optimal algorithm can be obtained by using the same idea as 2.5D dense matrix multiply $\rightarrow$ 2.5D-SMMM.

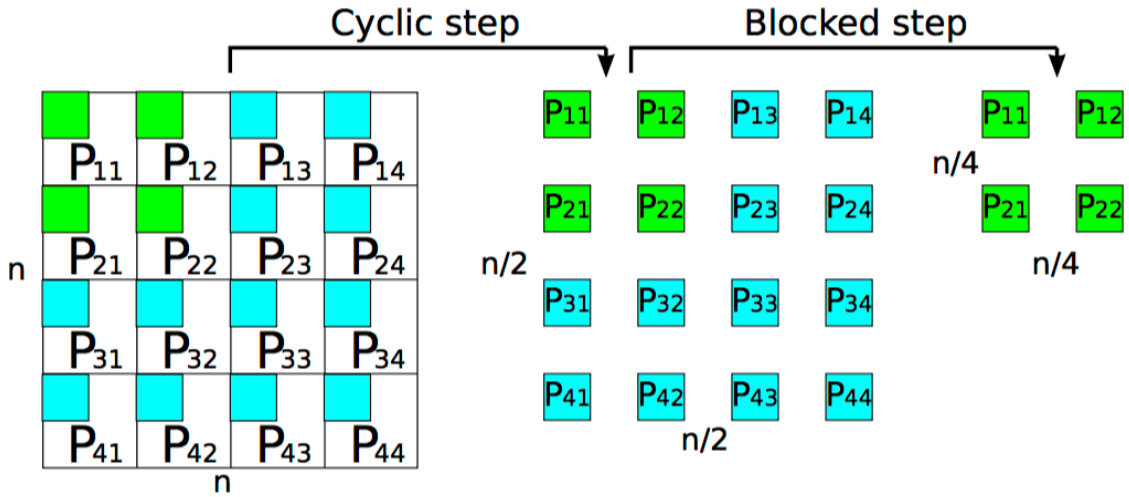
DC-APSP

- To minimize latency, 1/8-th of the processors should be assigned to solving a sub-problem
  $\rightarrow$ but then only 1/8-th of the processors are active

Solution: 2.5D block cyclic DC-APSP [Solomonik et al., 2013]

2.5D block cylic DC-APSP

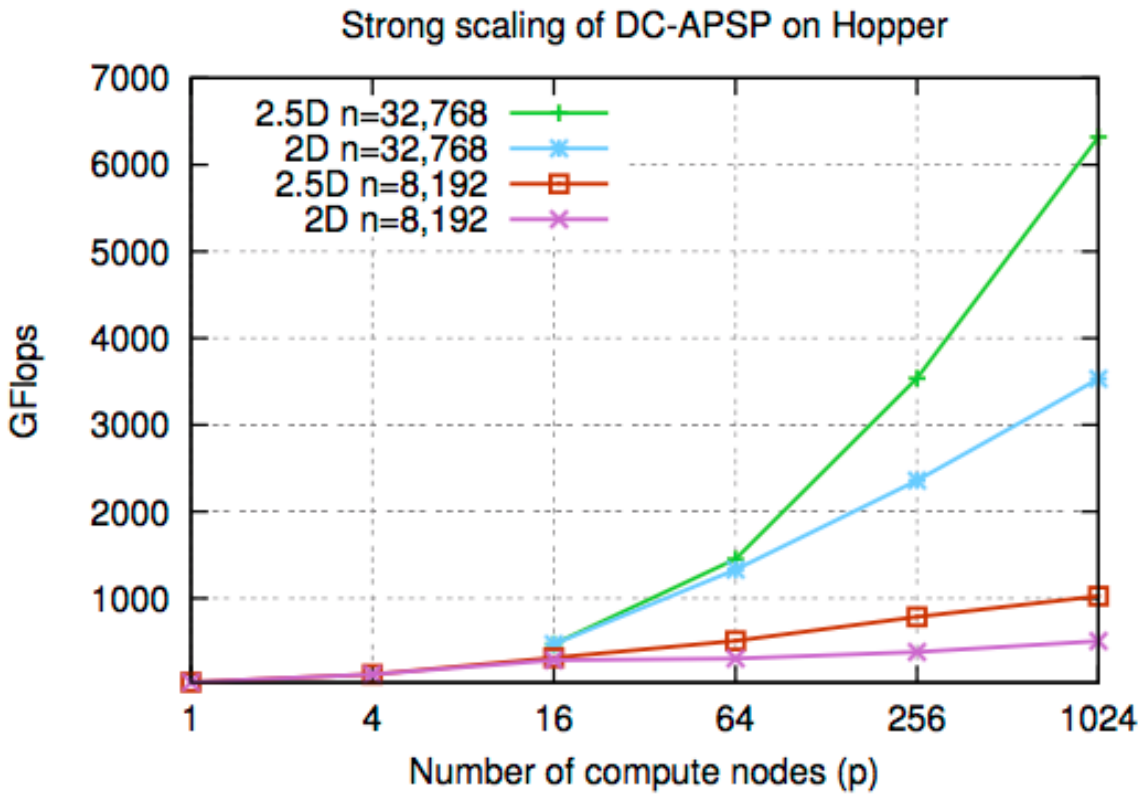
- Use 2.5D block cyclic DC-APSP until $c = 1$
  For block size $b = O(n/c)$, there are $O(\log c)$ recursive steps
- When $c = 1$, $P \geq 1$, switch to 2.5D-SMMM.



Communication optimal:

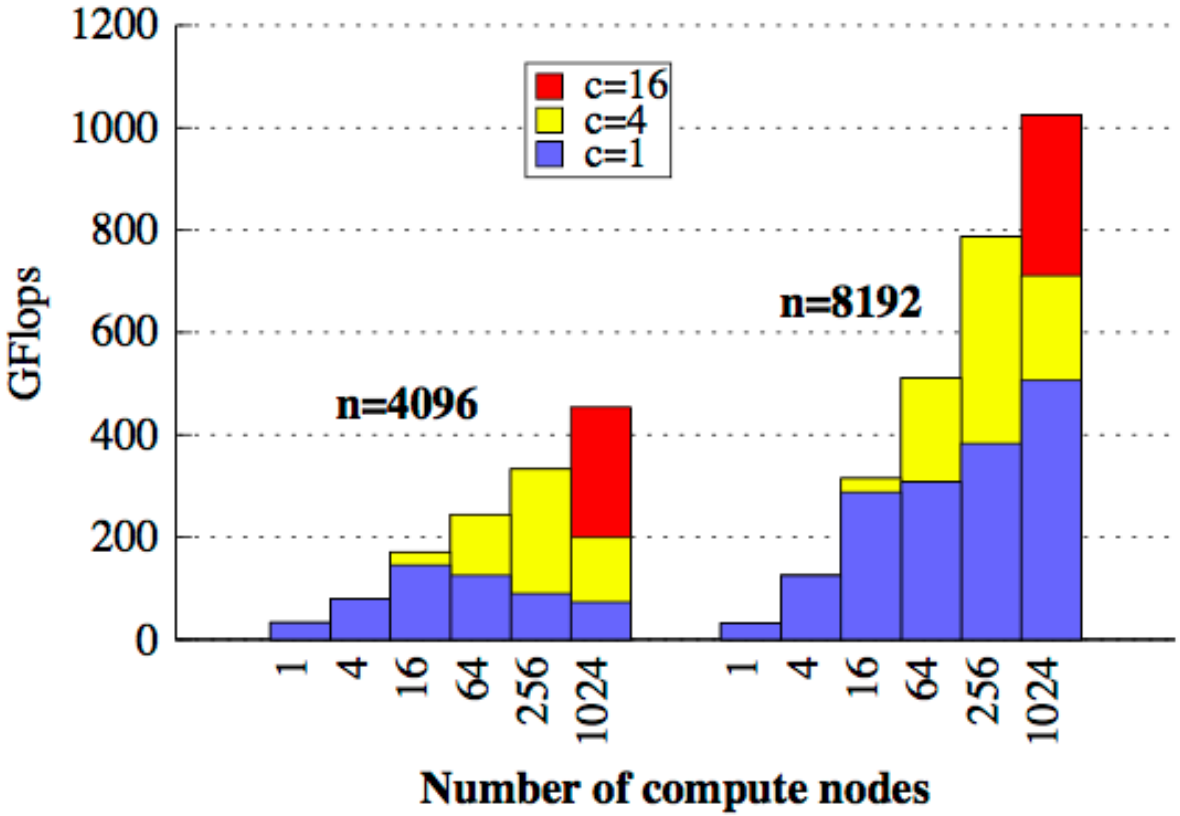
$$W = O(n^2/\sqrt{cP}), \quad S = O(\sqrt{cP}\log^2 P)$$

Experimental results [Solomonik et al., 2013]



- Hopper, Cray XE6, each node is a dual-socket 12-core Magny-Cours Opteron.
- Threaded Semiring-Matrix-Matrix-Multiply achieves 25% of peak performance (13.6 GFlops) on 6 cores (no fused multiply-add operation for the semiring).
- Strong scaling data: best performance for any replication factor c (often $c = 4$).
- On 24,276 cores, 2.5D faster by 1.8x for $n = 8,192$ and 2.0x for $n = 32,768$.

Experimental results [Solomonik et al., 2013]



- In the figure, bars stacked such that $c = 4$ case shows the benefit over $c = 1$ case.
- For $n = 4096$, $c = 16$ leads to a speed-up of 6.2x with respect to $c = 1$.

Conclusions

- Open problems:
  - Identify lower bounds on communication for other operations: LU, QR, etc.
  - Study other graph algorithms

References (1)

Ballard, G., Buluc, A., Demmel, J., Grigori, L., Schwartz, O., and Toledo, S. (2013).

Communication optimal parallel multiplication of sparse random matrices.
In *In Proceedings of ACM SPAA, Symposium on Parallelism in Algorithms and Architectures.*

Buluc, A., Gilbert, J. R., and Budak, C. (2010).

Solving path problems on the GPU.
Parallel Computing, 36:241–253.

Duff, I. S. (1982).

Full matrix techniques in sparse gaussian elimination.
In Springer-Verlag, editor, *Lecture Notes in Mathematics (912)*, pages 71–84.

George, A. (1973).

Nested dissection of a regular finite element mesh.
SIAM Journal on Numerical Analysis, 10:345–363.

George, A., Liu, J. W.-H., and Ng, E. G. (1989).

Communication results for parallel sparse Cholesky factorization on a hypercube.
Parallel Computing, 10(3):287–298.

Grigori, L., David, P.-Y., Demmel, J., and Peyronnet, S. (2010).

Brief announcement: Lower bounds on communication for direct methods in sparse linear algebra.
Proceedings of ACM SPAA.

Gupta, A., Karypis, G., and Kumar, V. (1995).

Highly scalable parallel algorithms for sparse matrix factorization.
IEEE Transactions on Parallel and Distributed Systems, 8(5).

References (2)

Hong, J.-W. and Kung, H. T. (1981).

I/O complexity: The Red-Blue Pebble Game.
In *STOC '81: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing*, pages 326–333, New York, NY, USA. ACM.

Irony, D., Toledo, S., and Tiskin, A. (2004).

Communication lower bounds for distributed-memory matrix multiplication.
J. Parallel Distrib. Comput., 64(9):1017–1026.

Liu, J. W. H. (1990).

The role of elimination trees in sparse factorization.
SIAM. J. Matrix Anal. & Appl., 11(1):134 – 172.

Parter, S. (1961).

The use of linear graphs in gaussian elimination.
SIAM Review, pages 364–369.

Rose, D. J. (1970).

Triangulated graphs and the elimination process.
Journal of Mathematical Analysis and Applications, pages 597–609.

Rose, D. J. and Tarjan, R. E. (1978).

Algorithmic aspects of vertex elimination on directed graphs.
SIAM J. Appl. Math., 34(1):176–197.

Schreiber, R. (1982).

A new implementation of sparse gaussian elimination.
ACM Trans. Math. Software, 8:256–276.

References (3)

Solomonik, E., Buluc, A., and Demmel, J. (2013).

Minimizing communication in all-pairs shortest-paths.

In *27th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*.