Sparse direct linear solvers
Woudschoten conference on *Parallel numerical linear algebra*
6-7 Octobre 2010

Patrick Amestoy
INPT-IRIT (University of Toulouse)
http://amestoy.perso.enseeiht.fr/

in collaboration with members of MUMPS team
A. Buttari, A. Guermouche, J.Y. L'Excellent, B. Ucar, and F.H. Rouet
http://mumps.enseeiht.fr or
http://graal.ens-lyon.fr/MUMPS/

Outline

Sparse direct linear solvers (I)

Woudschoten conference 2010

Outline

Context and motivations

A selection of references

- Books
  - Duff, Erisman and Reid, Direct methods for Sparse Matrices, Clarenton Press, Oxford 1986.
  - George, Liu, and Ng, Computer Solution of Sparse Positive Definite Systems, book to appear (2004)
  - Davis, Direct methods for sparse linear systems, SIAM, 2006.
- Articles
  - Gilbert and Liu, Elimination structures for unsymmetric sparse LU factors, SIMAX, 1993.
  - Liu, The role of elimination trees in sparse factorization, SIMAX, 1990.
  - Heath and E. Ng and B. W. Peyton, Parallel Algorithms for Sparse Linear Systems, SIAM review, 1991.
- Lecture Notes
  - P. Amestoy and J.Y. L'Excellent, Lecture notes on *Linear algebra and sparse direct methods*, UNESCO (Tunis), Master lectures (ENS-Lyon and INPT-ENSEEIHT)

Motivations

- solution of linear systems of equations $\rightarrow$ key algorithmic kernel

Continuous problem

$\downarrow$

Discretization

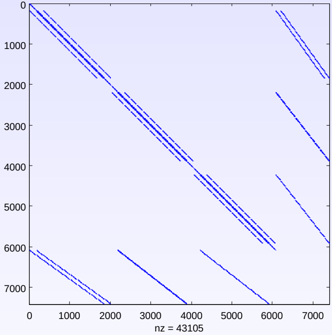
$\downarrow$

Solution of a linear system $Ax = b$

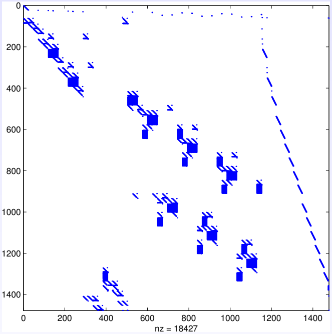
- Main parameters :
  - Numerical properties of the linear system (symmetry, pos. definite, conditioning, ...)
  - Size and structure :
    - Large ($> 10^7 \times 10^7$), square/rectangular
    - Dense or sparse (structured / unstructured)
    - Target computer (sequential/parallel/multicore/Cell/GPU)

Exemple of sparse matrices

Matrix from CFD
(Univ. Tel Aviv)



Chemical proc. Simulation
lhr01

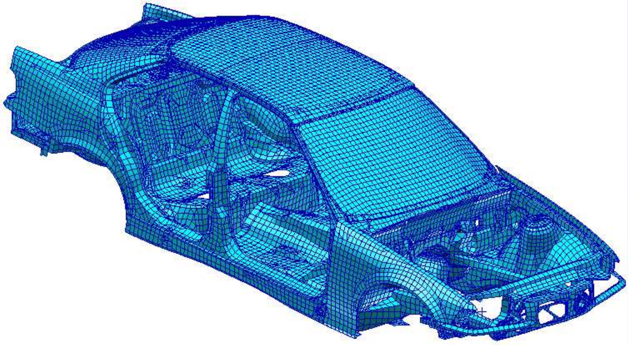


Matrix factorizations

Solution of $\mathbf{Ax} = \mathbf{b}$

- $\mathbf{A}$ is unsymmetric :
  - $\mathbf{A}$ is factorized as : $\mathbf{A} = \mathbf{LU}$, where
    $\mathbf{L}$ is a lower triangular matrix, and
    $\mathbf{U}$ is an upper triangular matrix.
  - Forward-backward substitution : $\mathbf{Ly} = \mathbf{b}$ then $\mathbf{Ux} = \mathbf{y}$
- $\mathbf{A}$ is symmetric :
  - $\mathbf{A} = \mathbf{LDL}^{\mathrm{T}}$ or $\mathbf{LL}^{\mathrm{T}}$
- $\mathbf{A}$ is rectangular $m \times n$ with $m \geq n$ and $\min_x \|\mathbf{Ax} - \mathbf{b}\|_2$:
  - $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q}$ is orthogonal ($\mathbf{Q}^{-1} = \mathbf{Q}^{\mathrm{T}}$) and $\mathbf{R}$ is triangular.
  - Solve : $\mathbf{y} = \mathbf{Q}^{\mathrm{T}}\mathbf{b}$ then $\mathbf{Rx} = \mathbf{y}$

Example in structural mechanics



BMW car body,
227,362 unknowns,
5,757,996 nonzeros,
MSC.Software

Size of factors : 51.1 million entries
Number of operations : 44.9×10^9

Solve $Ax = b$, A sparse

$\boxed{\text{Resolution with a 3 phase approach}}$

- Analysis phase
  - preprocess the matrix
  - prepare factorization
- Factorization phase
  - symmetric positive definite $\rightarrow$ $\mathbf{LL}^T$
  - symmetric indefinite $\rightarrow$ $\mathbf{LDL}^T$
  - unsymmetric $\rightarrow$ **LU**
- Solution phase exploiting factored matrices.
  - Postprocessing of the solution (iterative refinements and backward error analysis).

Sparse solver : only a black box ?

Default (often automatic/adaptive) setting of the options is often available ; However, a better knowledge of the options can help the user to further improve its solution.

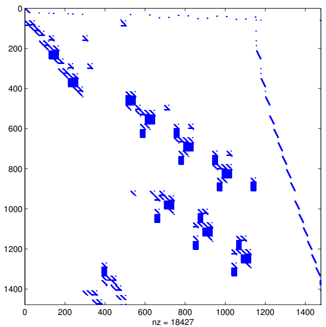
- Preprocessing may influence :
  - Operation cost and/or computational time
  - Size of factors and/or memory needed
  - Reliability of our estimations
  - Numerical accuracy.
- Describe preprocessing options and functionalities that are most critical to both performance and accuracy.

$Ax = b$?

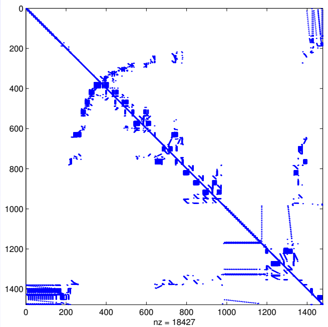
- Symmetric permutations to control inrease in the size of the factors : ($Ax = b \rightarrow PAP^tPx = b$)
- Numerical pivoting to preserve accuracy.
- Unsymmetric matrices ($A = LU$)
  - numerical equilibration (scaling rows/columns)
  - set large entries on the diagonal
  - modified problem : $A'x' = b'$ with $A' = P_n D_r PAQP^t D_c$
- Symmetric matrices ($A = LDL^t$) :
  Algorithms must also preserve symmetry (flops/memory divided by 2)
  - adapt equilibration and set large entries "on" diagonal while *preserving symmetry*
  - modified problem : $A' = P_N D_s PQ^t AQP^t D_s P_N^t$
- Preprocessing for parallelism (influence of task mapping on the performance)

Preprocessing - illustration

Original ($A =$lhr01)



Preprocessed matrix (A'(lhr01))



Outline

(Pre)Processing sparse matrices for efficiency and accuracy

- Fill-in and reordering
- Numerical threshold pivoting
- Preprocessing unsymmetric matrices
- Preprocessing symmetric matrices

(Pre)Processing sparse matrices for efficiency and accuracy

- Fill-in and reordering
- Numerical threshold pivoting
- Preprocessing unsymmetric matrices
- Preprocessing symmetric matrices

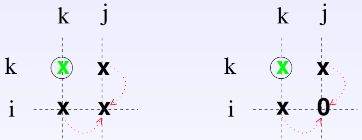
Fill-in and reordering

Step k of **LU** factorization (a_{kk} pivot) :

- For $i > k$ compute $l_{ik} = a_{ik}/a_{kk}$ $(= a'_{ik})$,
- For $i > k, j > k$

$$a'_{ij} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}} = a_{ij} - l_{ik} \times a_{kj}$$

- If $a_{ik} \neq 0$ and $a_{kj} \neq 0$ then $a'_{ij} \neq 0$
- If a_{ij} was zero $\rightarrow$ non-zero a'_{ij} must be stored : *fill-in*



Interest of permuting a matrix :

$$\begin{pmatrix} X & X & X & X & X \\ X & X & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 \\ X & 0 & 0 & X & 0 \\ X & 0 & 0 & 0 & X \end{pmatrix} \begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ X & X & X & X & X \end{pmatrix}$$

Symmetric matrices and graphs

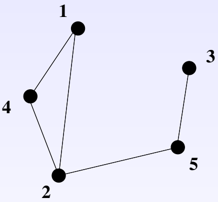
- Assumptions : **A** symmetric and pivots are chosen on the diagonal
- Structure of **A** symmetric represented by the graph $G = (V, E)$
  - Vertices are associated to columns : $V = \{1, \ldots, n\}$
  - Edges E are defined by : $(i, j) \in E \leftrightarrow a_{ij} \neq 0$
  - G undirected (symmetry of **A**)

Symmetric matrices and graphs

- Remarks :
  - Number of nonzeros in column j = $|\mathrm{adj}_G(j)|$
  - Symmetric permutation $\equiv$ renumbering the graph

	1	2	3	4	5
1	X	X		X	
2	X	X		X	X
3			X		X
4	X	X		X	
5		X	X		X

Symmetric matrix



Corresponding graph

The elimination graph model for symmetric matrices

- Let $\mathbf{A}$ be a symmetric positive define matrix of order n
- The $\mathbf{LL}^{\mathrm{T}}$ factorization can be described by the equation :

$$\mathbf{A} = \mathbf{A}_0 = \mathbf{H}_0 = \begin{pmatrix} d_1 & \mathbf{v}_1^{\mathrm{T}} \\ \mathbf{v}_1 & \overline{\mathbf{H}_1} \end{pmatrix}$$

$$= \begin{pmatrix} \sqrt{d_1} & 0 \\ \frac{\mathbf{v}_1}{\sqrt{d_1}} & \mathbf{I}_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{H}_1 \end{pmatrix} \begin{pmatrix} \sqrt{d_1} & \frac{\mathbf{v}_1^{\mathrm{T}}}{\sqrt{d_1}} \\ 0 & \mathbf{I}_{n-1} \end{pmatrix}$$

$= \mathbf{L}_1 \mathbf{A}_1 \mathbf{L}_1^{\mathrm{T}}$, where

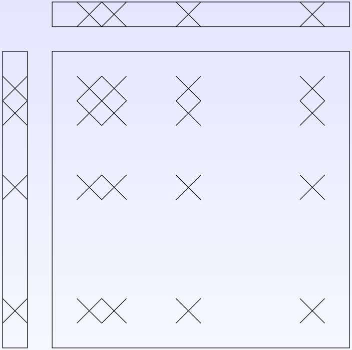
$$\mathbf{H}_1 = \overline{\mathbf{H}_1} - \frac{\mathbf{v}_1 \mathbf{v}_1^{\mathrm{T}}}{d_1}$$

- The basic step is applied on $\mathbf{H}_1 \mathbf{H}_2 \cdots$ to obtain :

$$\mathbf{A} = (\mathbf{L}_1 \mathbf{L}_2 \cdots \mathbf{L}_{n-1})\, \mathbf{I}_n \left(\mathbf{L}_{n-1}^T \ldots \mathbf{L}_2^T \mathbf{L}_1^T \right) \quad = \quad \mathbf{LL}^{\mathrm{T}}$$

The basic step : $\mathbf{H}_1 = \overline{\mathbf{H}_1} - \frac{\mathbf{v}_1\mathbf{v}_1^T}{d_1}$

What is $\mathbf{v}_1\mathbf{v}_1^T$ in terms of structure ?



$\mathbf{v}_1$ is a column of $\mathbf{A}$, hence the neighbors of the corresponding vertex.

$\mathbf{v}_1\mathbf{v}_1^T$ results in a dense sub-block in $\mathbf{H}_1$.

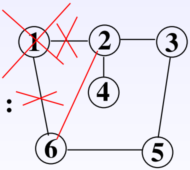
If any of the nonzeros in dense submatrix are not in $\mathbf{A}$, then we have fill-ins.

The elimination process in the graphs

$G_U(V,E) \leftarrow$ undirected graph of $\mathbf{A}$
for $k = 1 : n-1$ **do**
  $V \leftarrow V - \{k\}$ {remove vertex k}
  $E \leftarrow E - \{(k,\ell) : \ell \in \mathrm{adj}(k)\} \cup \{(x,y) : x \in \mathrm{adj}(k) \text{ and } y \in \mathrm{adj}(k)\}$
  $G_k \leftarrow (V,E)$ {for definition}
end for

G_k are the so-called elimination graphs (Parter,'61).

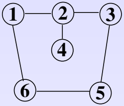
G0 :



$$\mathbf{H0} = \begin{bmatrix} \mathbf{1} & \times & & & & \times \\ \times & \mathbf{2} & \times & \times & & \times \\ & \times & \mathbf{3} & & \times & \\ & \times & & \mathbf{4} & & \\ & & \times & & \mathbf{5} & \times \\ \times & \times & & & \times & \mathbf{6} \end{bmatrix}$$

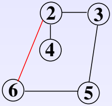
A sequence of elimination graphs

G0 :



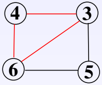
$$H0 = \begin{bmatrix} 1 & \times & & & & \times \\ \times & 2 & \times & \times & & \\ & \times & 3 & & \times & \\ & \times & & 4 & & \\ & & \times & & 5 & \times \\ \times & & & & \times & 6 \end{bmatrix}$$

G1 :



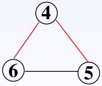
$$H1 = \begin{bmatrix} 2 & \times & \times & & + \\ \times & 3 & & \times & \\ \times & & 4 & & \\ & \times & & 5 & \times \\ + & & & \times & 6 \end{bmatrix}$$

G2 :



$$H2 = \begin{bmatrix} 3 & + & \times & + \\ + & 4 & & + \\ \times & & 5 & \times \\ + & + & \times & 6 \end{bmatrix}$$

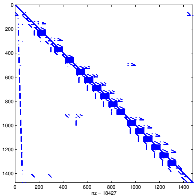
G3 :



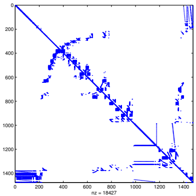
$$H3 = \begin{bmatrix} 4 & + & + \\ + & 5 & \times \\ + & \times & 6 \end{bmatrix}$$

Fill-in and reordering

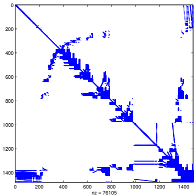
"Before permutation" (A''(lhr01))



Permuted matrix (A'(lhr01))



Factored matrix ($LU(A')$)

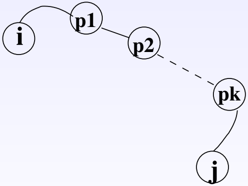


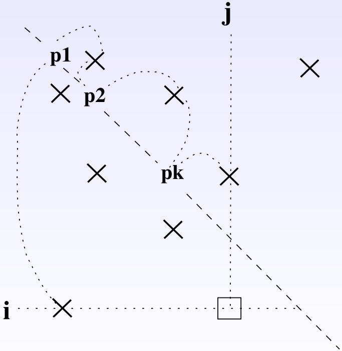
Fill-in characterization

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^t$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.



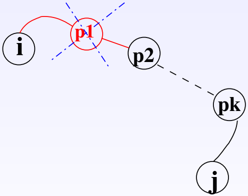


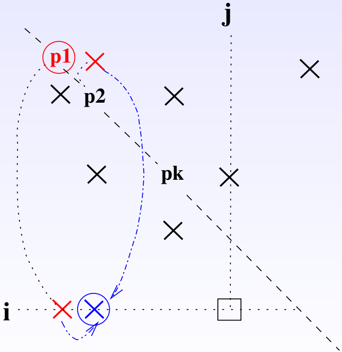
Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^t$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.



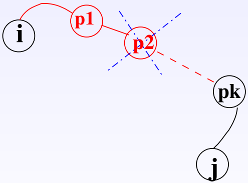


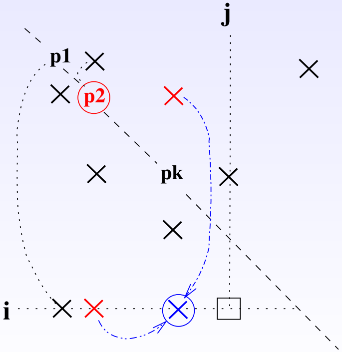
Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^t$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.



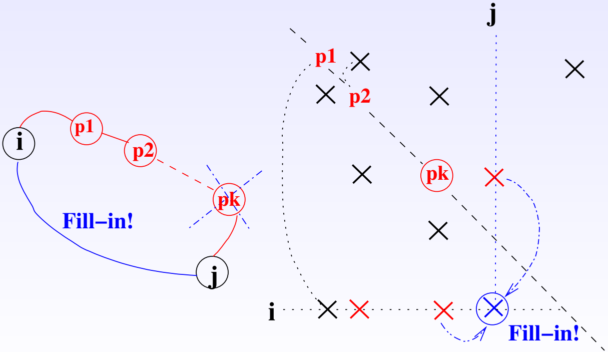


Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^t$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.

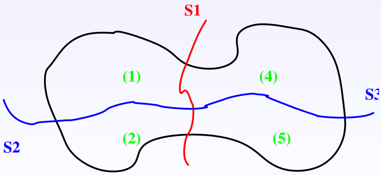


Fill-reducing heuristics

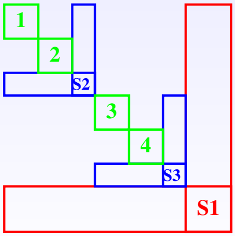
Three main classes of methods for minimizing fill-in during factorization

- Global approach : The matrix is permuted into a matrix with a given pattern
    - Fill-in is restricted to occur within that structure
    - Cuthill-McKee (block tridiagonal matrix)
    - Nested dissections (“block bordered” matrix)
      (Remark : interpretation using the fill-path theorem)

Graph partitioning



Permuted matrix



Fill-reducing heuristics

- Local heuristics : At each step of the factorization, selection of the pivot that is likely to minimize fill-in.
    - Method is characterized by the way pivots are selected.
    - Markowitz criterion (for a general matrix).
    - Minimum degree or Minimum fill-in (for symmetric matrices).
- Hybrid approaches : Once the matrix is permuted to block structure, local heuristics are used within the blocks.

Local heuristics to reduce fill-in during factorization

Let $G(A)$ be the graph associated to a matrix A that we want to order using local heuristics.
Let *Metric* such that $Metric(v_i) < Metric(v_j)$ implies v_i is a better than v_j

Generic algorithm

Loop until all nodes are selected

Step1 : select current node p (so called pivot) with minimum metric value,

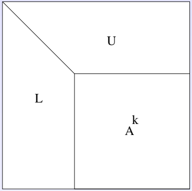
Step2 : update elimination graph,

Step3 : update $Metric(v_j)$ for all non-selected nodes v_j.

Step3 should only be applied to nodes for which the Metric value might have changed.

Reordering unsymmetric matrices : Markowitz criterion

- At step k of Gaussian elimination :

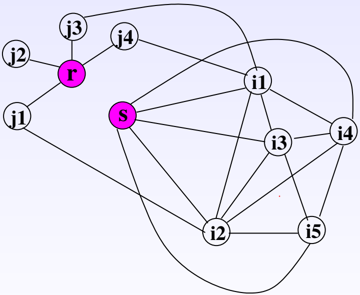


  - r_i^k = number of non-zeros in row i of $\mathbf{A}^k$
  - c_j^k = number of non-zeros in column j of $\mathbf{A}^k$
  - a_{ij} must be large enough and should minimize $(r_i^k - 1) \times (c_j^k - 1) \quad \forall i, j > k$

- $\boxed{\text{Minimum degree}}$: Markowitz criterion for symmetric diagonally dominant matrices

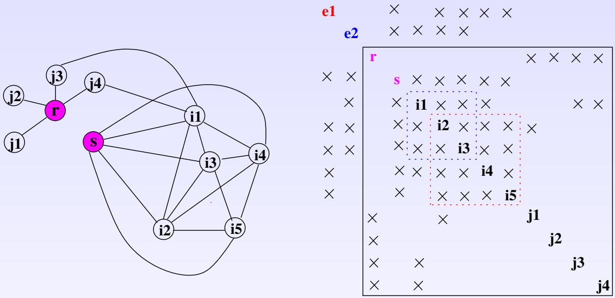
Minimum fill based algorithm

- *Metric*(v_i) is the amount of fill-in that v_i would introduce if it were selected as a pivot.
- Illustration : r has a degree $d = 4$ and a fill-in metric of $d \times (d-1)/2 = 6$ whereas s has degree $d = 5$ but a fill-in metric of $d \times (d-1)/2 - 9 = 1$.



Minimum fill-in properties

- The situation typically occurs when $\{i_1, i_2, i_3\}$ and $\{i_2, i_3, i_4, i_5\}$ were adjacent to two already selected nodes (here e_2 and e_1)



e1 and e2 are previously selected nodes

- The elimination of a node v_k affects the degree of nodes adjacent to v_k. *The fill-in metric of $Adj(Adj(v_k))$ is also affected.*
- Illustration : selecting r affects the fill-in of i_1 (fill edge (j_3, j_4) should be deduced).

Impact of fill-reducing heuristics

Number of operations (millions)

	METIS	SCOTCH	PORD	AMF	AMD
gupta2	2757.8	4510.7	*4993.3*	2790.3	**2663.9**
ship_003	**83828.2**	92614.0	112519.6	96445.2	*155725.5*
twotone	29120.3	**27764.7**	*37167.4*	29847.5	29552.9
wang3	**4313.1**	5801.7	5009.9	6318.0	*10492.2*
xenon2	**99273.1**	112213.4	126349.7	237451.3	*298363.5*

- METIS (Karypis and Kumar) and SCOTCH (Pellegrini) are global strategies (recursive nested dissection based orderings).
- PORD (Schulze, Paderborn Univ.) recursive dissection based on a bottom up strategy to build the separator
- AMD (Amestoy, Davis and Duff) is a local strategy based on Approximate Minimum Degree.
- AMF (Amestoy) is a local strategy based on Approx. Minimum Fill.

Impact of fill-reducing heuristics

Time for factorization (seconds)

		1p	16p	32p	64p	128p
coneshl	METIS	970	60	41	27	14
	PORD	1264	104	67	41	26
audi	METIS	2640	198	108	70	42
	PORD	1599	186	146	83	54

Matrices with quasi dense rows :
Impact on the analysis time (seconds) of gupta2 matrix

	AMD	METIS	QAMD
Analysis	361	52	23
Total	379	76	59

- QAMD (Amestoy) Approximate Minimum Degree (local) strategy designed for matrices with *quasi dense rows*.

(Pre)Processing sparse matrices for efficiency and accuracy

- Fill-in and reordering
- Numerical threshold pivoting
- Preprocessing unsymmetric matrices
- Preprocessing symmetric matrices

Numerical pivoting during LU factorization

$$\text{Let } A = \begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\epsilon} & 1 \end{bmatrix} \times \begin{bmatrix} \epsilon & 1 \\ 0 & 1 - \frac{1}{\epsilon} \end{bmatrix}$$

$\kappa_2(A) = 1 + O(\epsilon)$.

If we solve :

$$\begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1+\epsilon \\ 2 \end{bmatrix}$$

Exact solution :$x^* = (1, 1)$.

ϵ	$\frac{\|x^* - x\|}{\|x^*\|}$
10^{-3}	6×10^{-6}
10^{-9}	9×10^{-8}
10^{-15}	7×10^{-2}

Table: Relative error as a function of ϵ.

Numerical pivoting during LU factorization (II)

- Even if A well-conditioned then Gaussian elimination might introduce errors.
- Explanation : pivot ϵ is too small (relative)
- Solution : interchange rows 1 and 2 of A.

$$\begin{bmatrix} 1 & 1 \\ \epsilon & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1+\epsilon \end{bmatrix}$$

$\rightarrow$ No more error.

Threshold pivoting for sparse matrices

- $\boxed{\text{Sparse } LU \text{ factorization}}$
  - Threshold u : Set of eligible pivots = $\{r \mid |a_{rk}^{(k)}| \geq u \times \max_i |a_{ik}^{(k)}|\}$, where $0 < u \leq 1$.
  - Among eligible pivots select one preserving sparsity.
- $\boxed{\text{Sparse } LDL^T \text{ factorization}}$
  - Symmetric indefinite case : requires 2 by 2 pivots, e.g. $\begin{pmatrix} \epsilon & x \\ x & \epsilon \end{pmatrix}$
  - 2×2 pivot $P = \begin{pmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{pmatrix}$:

$$|P^{-1}| \begin{pmatrix} \max_i |a_{ki}| \\ \max_j |a_{lj}| \end{pmatrix} \leq \begin{pmatrix} 1/u \\ 1/u \end{pmatrix}$$

- $\boxed{\text{Static pivoting}}$: Add small perturbations to the matrix of factors to reduce the amount of numerical pivoting.

Threshold pivoting for sparse matrices

- $\boxed{\text{Sparse } LU \text{ factorization}}$
  - Threshold u : Set of eligible pivots = $\{r \mid |a_{rk}^{(k)}| \geq u \times \max_i |a_{ik}^{(k)}|\}$, where $0 < u \leq 1$.
  - Among eligible pivots select one preserving sparsity.
- $\boxed{\text{Sparse } LDL^T \text{ factorization}}$
  - Symmetric indefinite case : requires 2 by 2 pivots, e.g. $\begin{pmatrix} \epsilon & x \\ x & \epsilon \end{pmatrix}$
  - 2×2 pivot $P = \begin{pmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{pmatrix}$:

$$|P^{-1}| \begin{pmatrix} \max_i |a_{ki}| \\ \max_j |a_{lj}| \end{pmatrix} \leq \begin{pmatrix} 1/u \\ 1/u \end{pmatrix}$$

- $\boxed{\text{Static pivoting}}$: Add small perturbations to the matrix of factors to reduce the amount of numerical pivoting.

Threshold pivoting for sparse matrices

- $\boxed{\text{Sparse } LU \text{ factorization}}$
  - Threshold u : Set of eligible pivots = $\{r \mid |a_{rk}^{(k)}| \geq u \times \max_i |a_{ik}^{(k)}|\}$, where $0 < u \leq 1$.
  - Among eligible pivots select one preserving sparsity.
- $\boxed{\text{Sparse } LDL^T \text{ factorization}}$
  - Symmetric indefinite case : requires 2 by 2 pivots, e.g. $\begin{pmatrix} \epsilon & x \\ x & \epsilon \end{pmatrix}$
  - 2×2 pivot $P = \begin{pmatrix} a_{kk} & a_{kl} \\ a_{lk} & a_{ll} \end{pmatrix}$:

$$|P^{-1}| \begin{pmatrix} \max_i |a_{ki}| \\ \max_j |a_{lj}| \end{pmatrix} \leq \begin{pmatrix} 1/u \\ 1/u \end{pmatrix}$$

- $\boxed{\text{Static pivoting}}$: Add small perturbations to the matrix of factors to reduce the amount of numerical pivoting.

(Pre)Processing sparse matrices for efficiency and accuracy

- Fill-in and reordering
- Numerical threshold pivoting
- Preprocessing unsymmetric matrices
- Preprocessing symmetric matrices

Preprocessing unsymmetric matrices - scaling

- *Objective* : Matrix equilibration to help threshold pivoting.
- Row and column scaling : $B = D_r A D_c$ where D_r, D_c are diagonal matrices to respectively scale rows and columns of A
  - reduce the amount of numerical problems

$$\text{Let } A = \begin{bmatrix} 1 & 2 \\ 10^{16} & 10^{16} \end{bmatrix} \rightarrow \text{Let } B = D_r A = \begin{bmatrix} 1 & 2 \\ 1 & 1 \end{bmatrix}$$

  - better detect real problems.

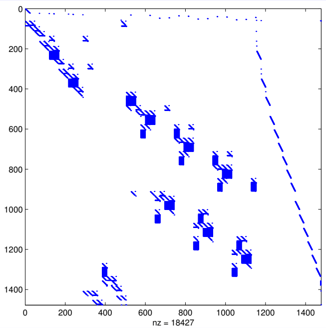
$$\text{Let } A = \begin{bmatrix} 1 & 10^{16} \\ 1 & 1 \end{bmatrix} \rightarrow \text{Let } B = D_r A = \begin{bmatrix} 10^{-16} & 1 \\ 1 & 1 \end{bmatrix}$$

- Influence quality of fill-in estimations and accuracy.
- Should be activated when the number of uneliminated variables is large.

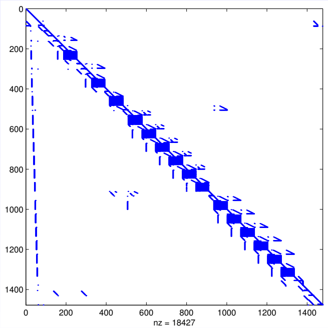
Preprocessing - Maximum weighted matching (I)

- *Objective* : Set large entries on the diagonal
  - Unsymmetric permutation and scaling
  - Preprocessed matrix $\mathbf{B} = \mathbf{D}_1\mathbf{AQD}_2$ is such that $|b_{ii}| = 1$ and $|b_{ij}| \leq 1$

Original ($A =$lhr01)



Permuted ($A' = AQ$)



Combine maximum transversal and fill-in reduction

- Consider the $\mathbf{LU}$ factorization $\mathbf{A} = \mathbf{LU}$ of an unsymmetric matrix.
- Compute the column permutation $\mathbf{Q}$ leading to a maximum numerical transversal of A. $\mathbf{AQ}$ has large (in some sense) numerical entries on the diagonal.
- Find best ordering of $\mathbf{AQ}$ preserving the diagonal entries. Equivalent to finding symmetric permutation $\mathbf{P}$ such that the factorization of $\mathbf{PAQP}^{\mathrm{T}}$ has reduced fill-in.

Preprocessing - Maximum weighted matching

- *Influence of maximum weighted matching* (Duff and Koster (99,01) on the performance

Matrix		Symmetry	$\|LU\|$ (10^6)	Flops (10^9)	Backwd Error
twotone	**OFF**	28	235	1221	
	ON	43	22	29	
fidapm11	**OFF**	100	16	10	
	ON	46	28	29	

- On very unsymmetric matrices : reduce flops, factor size and memory used.
- In general improve accuracy, and reduce number of iterative refinements.
- Improve reliability of memory estimates.

Preprocessing - Maximum weighted matching

- *Influence of maximum weighted matching* (Duff and Koster (99,01) on the performance

Matrix		Symmetry	$\|LU\|$ (10^6)	Flops (10^9)	Backwd Error
twotone	**OFF**	28	235	1221	10^{-6}
	ON	43	22	29	10^{-12}
fidapm11	**OFF**	100	16	10	10^{-10}
	ON	46	28	29	10^{-11}

- On very unsymmetric matrices : reduce flops, factor size and memory used.
- In general improve accuracy, and reduce number of iterative refinements.
- Improve reliability of memory estimates.

Preprocessing - Maximum weighted matching

- *Influence of maximum weighted matching* (Duff and Koster (99,01) on the performance

Matrix		Symmetry	$\|LU\|$ (10^6)	Flops (10^9)	Backwd Error
twotone	**OFF**	28	235	1221	10^{-6}
	ON	43	22	29	10^{-12}
fidapm11	**OFF**	100	16	10	10^{-10}
	ON	46	28	29	10^{-11}

- On very unsymmetric matrices : reduce flops, factor size and memory used.
- In general improve accuracy, and reduce number of iterative refinements.
- Improve reliability of memory estimates.

(Pre)Processing sparse matrices for efficiency and accuracy

- Fill-in and reordering
- Numerical threshold pivoting
- Preprocessing unsymmetric matrices
- Preprocessing symmetric matrices

Preprocessing symmetric matrices (Duff and Pralet (2004, 2005)

- $\boxed{\text{Symmetric scaling}}$: Adapt `MC64` (Duff and Koster, 2001) unsymmetric scaling :
  let $D = \sqrt{D_r D_c}$, then $B = DAD$ is a symmetrically scaled matrix which satisfies

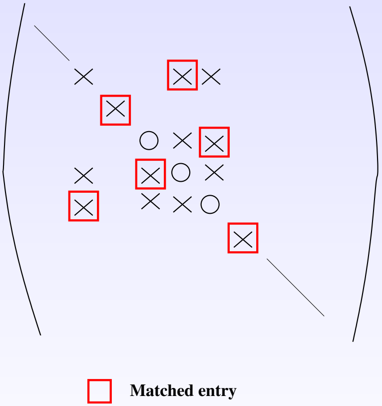
$$\forall i, |b_{i\sigma(i)}| = ||b_{.\sigma(i)}||_\infty = ||b_{i.}^T||_\infty = 1$$

  where σ is the permutation from the unsym. transv. algo.
- Influence of scaling on augmented matrices $K = \begin{pmatrix} H & A \\ A^T & 0 \end{pmatrix}$

	Total time (seconds)		Nb of entries in factors (millions) (estimated)		(effective)	
Scaling :	OFF	ON	OFF	ON	OFF	ON
cont-300	45	5	12.2	12.2	32.0	12.4
cvxqp3	1816	28	3.9	3.9	62.4	9.3
stokes128	3	2	3.0	3.0	5.5	3.3

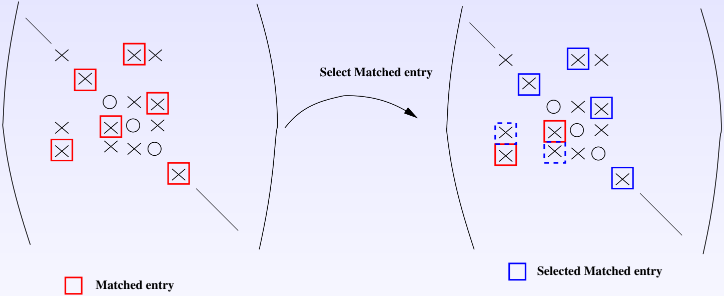
Preprocessing - Compressed ordering

- Perform an unsymmetric weighted matching



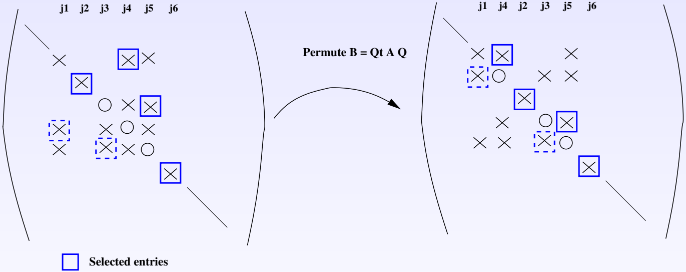
Preprocessing - Compressed ordering

- Perform an unsymmetric weighted matching
- Select matched entries



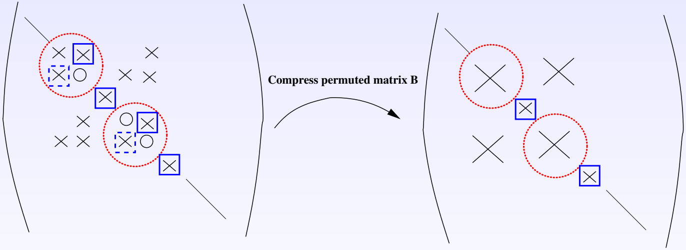
Preprocessing - Compressed ordering

- Perform an unsymmetric weighted matching
- Select matched entries
- Symmetrically permute matrix to set large entries near diagonal



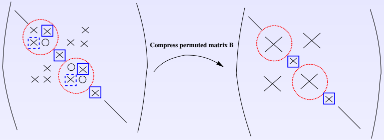
Preprocessing - Compressed ordering

- Perform an unsymmetric weighted matching
- Select matched entries
- Symmetrically permute matrix to set large entries near diagonal
- Compression : 2×2 diagonal blocks become supervariables.



Preprocessing - Compressed ordering

- Perform an unsymmetric weighted matching
- Select matched entries
- Symmetrically permute matrix to set large entries near diagonal
- Compression : 2×2 diagonal blocks become supervariables.

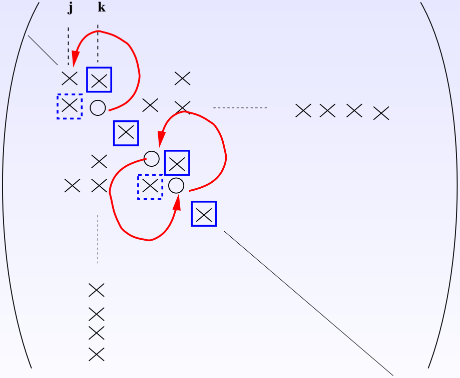


Influence of using a compressed graph (with scaling)

	Total time (seconds)		Nb of entries in factors in Millions (estimated)		(effective)	
Compression :	OFF	ON	OFF	ON	OFF	ON
cont-300	5	4	12.3	11.2	32.0	12.4
cvxqp3	28	11	3.9	7.1	9.3	8.5
stokes128	1	2	3.0	5.7	3.4	5.7

Preprocessing - Constrained ordering

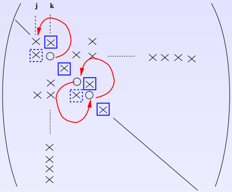
- Part of matrix sparsity is lost during graph compression
- $\boxed{\text{Constrained ordering}}$: *only pivot dependency within* 2×2 *blocks need be respected.*
  Ex : $k \rightarrow j$ indicates that if k is selected before j then j must be eliminated together with k.



if j is selected first then no more constraint on k.

Preprocessing - Constrained ordering

- $\boxed{\text{Constrained ordering}}$: *only pivot dependency within* 2×2 *blocks need be respected.*



Influence of using a constrained ordering (with scaling)

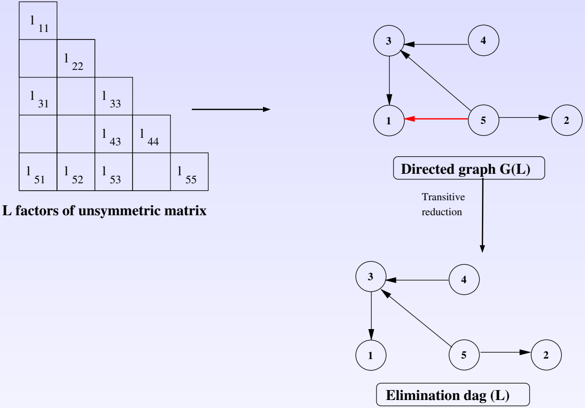
	Total time (seconds)		Nb of entries in factors in Millions (estimated)		(effective)	
Constrained :	OFF	ON	OFF	ON	OFF	ON
cvxqp3	11	8	7.2	6.3	8.6	7.2
stokes128	2	2	5.7	5.2	5.7	5.3

Outline

Approaches for parallel factorization

- Elimination trees
- Distributed memory sparse solvers
- Some parallel solvers
- Case study : comparison of MUMPS and SuperLU

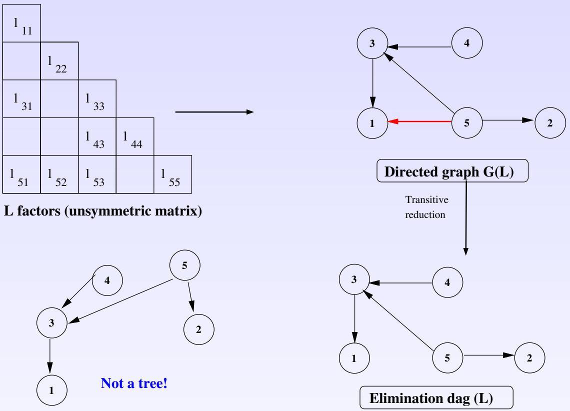
Elimination DAG and unsymmetric matrices



Elimination dags : transitive reduction of the G(L)

- Because of unsymmetry the transitive reduction is not a tree
- What makes L be the factors of an unsymmetric matrix ?

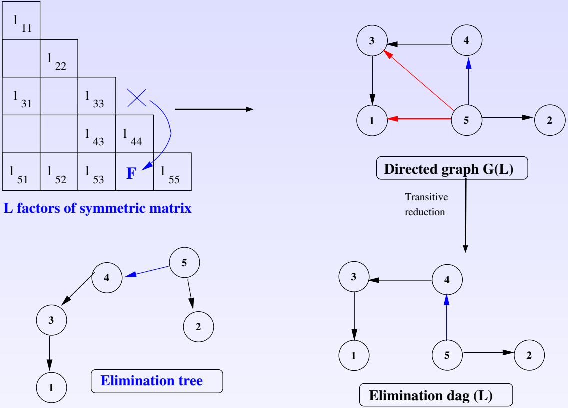
Elimination DAG and unsymmetric matrices



Elimination dags : transitive reduction of the G(L)

- Because of unsymmetry the transitive reduction is not a tree
- What makes L be the factors of an unsymmetric matrix ?

Elimination tree and symmetric matrices



Elimination dags : transitive reduction of the G(L)

- Because of unsymmetry the transitive reduction is not a tree
- What makes L be the factors of an unsymmetric matrix ?

Elimination tree

<u>**To summarize (for symmetric structured matrices)**</u> **:**

- The elimination tree expresses dependencies between the various steps of the factorization.
- It also exhibits parallelism arising from the sparse structure of the matrix.

Building the elimination tree

- Permute matrix (to reduce fill-in) $\mathbf{PAP}^{\mathrm{T}}$.
- Build filled matrix $\mathbf{A}_F = \mathbf{L} + \mathbf{L}^{\mathrm{T}}$ where $\mathbf{PAP}^{\mathrm{T}} = \mathbf{LL}^{\mathrm{T}}$
- Transitive reduction of associated filled graph

$\rightarrow$ Each column corresponds to a node of the graph. Each node k of the tree corresponds to the factorization of a frontal matrix whose row structure is that of column k of $\mathbf{A}_F$.

Approaches for parallel factorization

Elimination trees

Distributed memory sparse solvers

Some parallel solvers

Case study : comparison of MUMPS and SuperLU

Distributed memory sparse solvers

Computational strategies for parallel direct solvers

- The parallel algorithm is characterized by :
  - Computational graph dependency
  - Communication graph
- Three classical approaches
  1. "Fan-in"
  2. "Fan-out"
  3. "Multifrontal"

Preamble : left and right looking approaches for Cholesky factorization

- $cmod(j,k)$: Modification of column j by column k, $k < j$,
- $cdiv(j)$ division of column j by the pivot

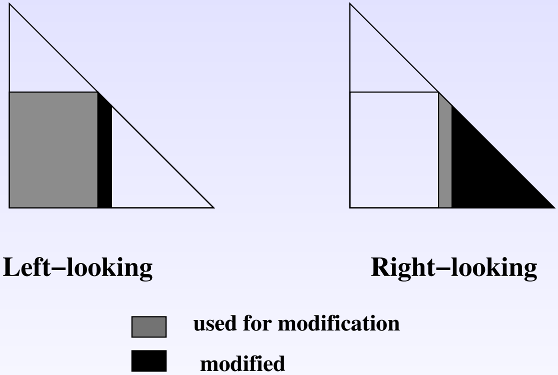
$\boxed{\text{Left-looking approach}}$

for $j = 1$ to n **do**
  for $k \in Struct(row\ \mathbf{L}_{j,1:j-1})$ **do**
    $cmod(j,k)$
  $cdiv(j)$

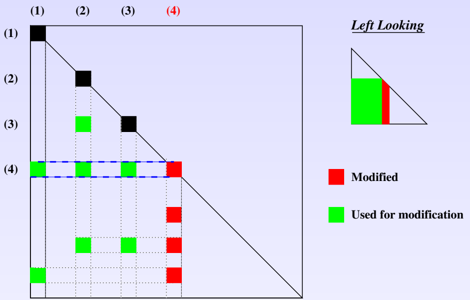
$\boxed{\text{Right-looking approach}}$

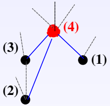
for $k = 1$ to n **do**
  $cdiv(k)$
  for $j \in Struct(col\ \mathbf{L}_{k+1:n,k})$ **do**
    $cmod(j,k)$

Illustration of Left and right looking



Fan-in variant (similar to left looking)



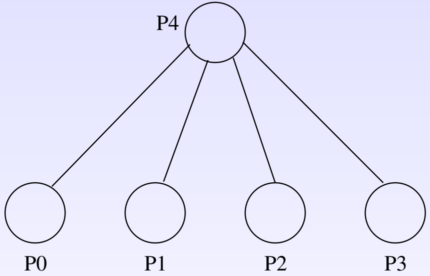


Algorithm: (Cholesky)

For j=1 to n do
  For k in Struct($L_{j,*}$) do
    cmod(j,k)
  Endfor
  cdiv(j)
Endfor

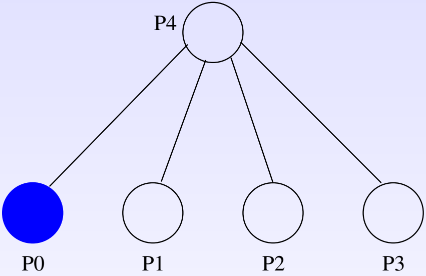
if $map(1) = map(2) = map(3) = p$ *and* $map(4) \neq p$ *(only) one message sent by* p *to update column* 4 $\rightarrow$ *exploits data locality in the tree.*

Fan-in variant



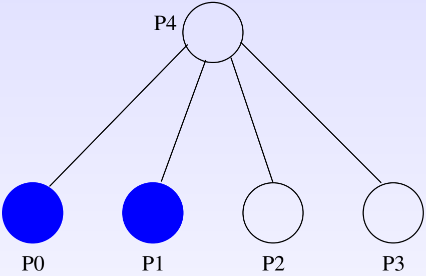
if $\forall i \in$ *children* $map(i) = P0$ *and* $map(father) \neq P0$ *(only) one message sent by* $P0 \rightarrow$ *exploits data locality in the tree.*

Fan-in variant



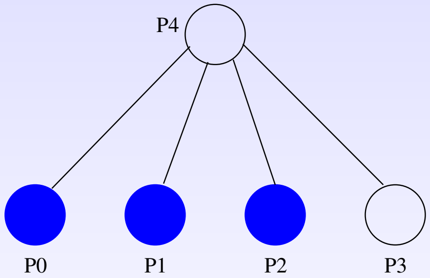
if $\forall i \in$ *children* $map(i) = P0$ *and* $map(father) \neq P0$ *(only) one message sent by* $P0 \rightarrow$ *exploits data locality in the tree.*

Fan-in variant



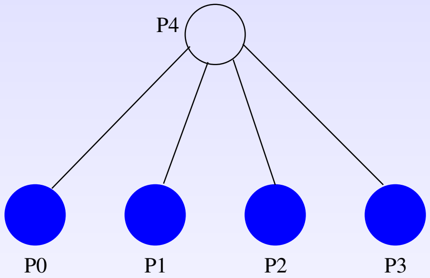
if $\forall i \in$ children $map(i) = P0$ and $map(father) \neq P0$ (only) one message sent by P0 $\rightarrow$ exploits data locality in the tree.

Fan-in variant



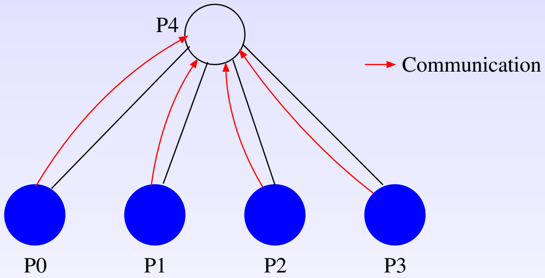
if $\forall i \in$ *children* $map(i) = P0$ *and* $map(father) \neq P0$ *(only) one message sent by* $P0 \rightarrow$ *exploits data locality in the tree.*

Fan-in variant



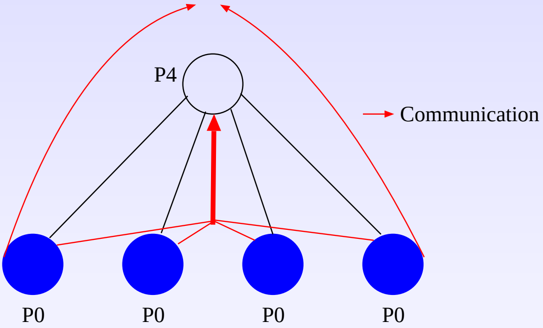
if $\forall i \in$ *children* $map(i) = P0$ *and* $map(father) \neq P0$ *(only) one message sent by* $P0 \rightarrow$ *exploits data locality in the tree.*

Fan-in variant



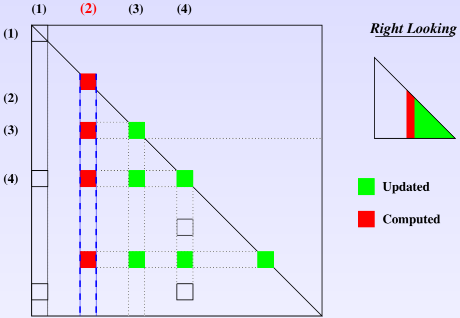
if $\forall i \in$ *children* $map(i) = P0$ *and* $map(father) \neq P0$ *(only) one message sent by* $P0 \rightarrow$ *exploits data locality in the tree.*

Fan-in variant



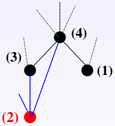
if $\forall i \in$ *children* $map(i) = P0$ *and* $map(father) \neq P0$ *(only) one message sent by* $P0$ $\rightarrow$ *exploits data locality in the tree.*

Fan-out variant (similar to right-looking)



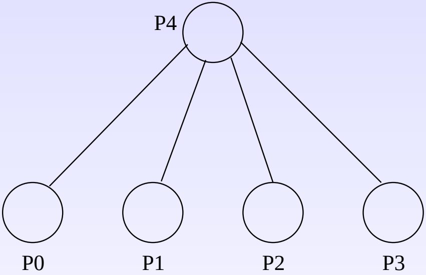
Algorithm: (Cholesky)

For k=1 to n do
  cdiv(k)
  For j in Struct($L_{*,k}$) do
    cmod(j,k)
  Endfor
Endfor



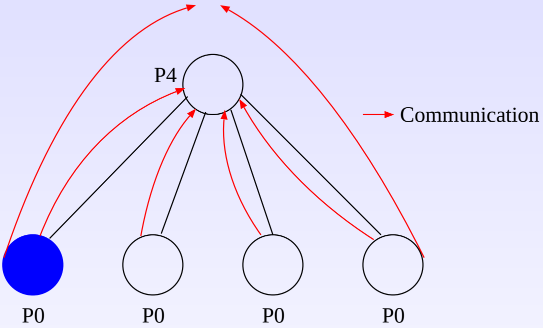
if $map(2) = map(3) = p$ *and* $map(4) \neq p$ *then 2 messages (for column 2 and 3) are sent by* p *to update column 4.*

Fan-out variant



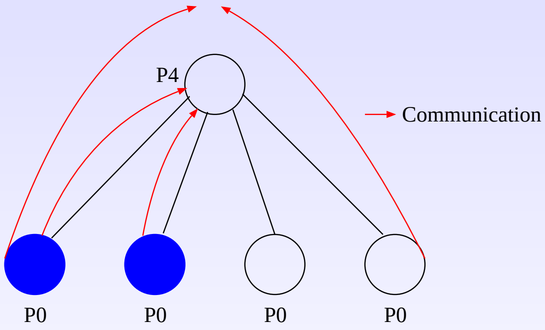
if $\forall i \in$ *children* $map(i) = P0$ *and* $map(father) \neq P0$ *then* n *messages (where* n *is the number of children) are sent by* $P0$ *to update the processor in charge of the father*

Fan-out variant



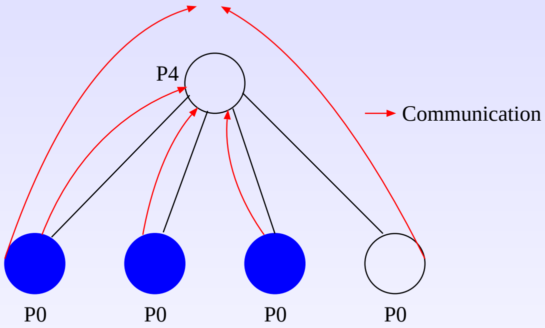
if $\forall i \in children\ map(i) = P0$ *and* $map(father) \neq P0$ *then* n *messages (where* n *is the number of children) are sent by* $P0$ *to update the processor in charge of the father*

Fan-out variant



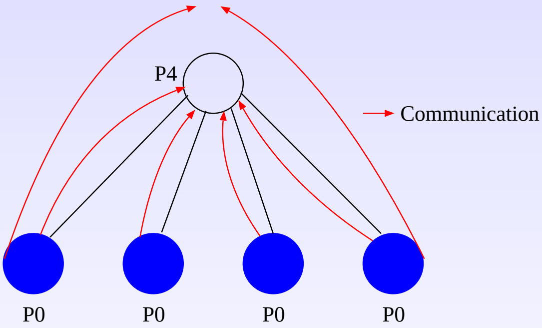
if $\forall i \in$ *children* $map(i) = P0$ *and* $map(father) \neq P0$ *then* n *messages (where* n *is the number of children) are sent by* $P0$ *to update the processor in charge of the father*

Fan-out variant



if $\forall i \in children\ map(i) = P0$ *and* $map(father) \neq P0$ *then* n *messages (where* n *is the number of children) are sent by* $P0$ *to update the processor in charge of the father*

Fan-out variant



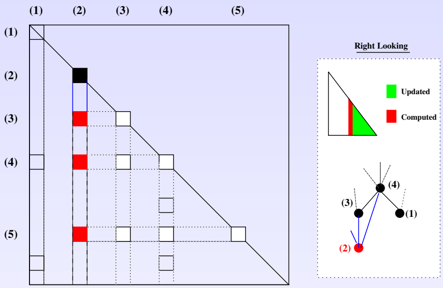
if $\forall i \in children \ map(i) = P0$ *and* $map(father) \neq P0$ *then* n *messages (where* n *is the number of children) are sent by* $P0$ *to update the processor in charge of the father*

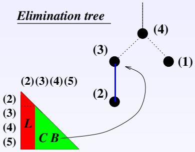
Fan-out variant

Properties of fan-out :

- Historically the first implemented.
- Incurs greater interprocessor communications than fan-in (or multifrontal) approach both in terms of
  - total number of messages
  - total volume
- Does not exploit data locality in the mapping of nodes in the tree
- Improved algorithm (local aggregation) :
  - send aggregated update columns instead of individual factor columns for columns mapped on a single processor.
  - Improve exploitation of data locality
  - But memory increase to store aggregates can be critical (as in fan-in).

Multifrontal variant





Algorithm:

For k=1 to n do

Build full frontal matrix
with all indices in Struct($L_{*,k}$)

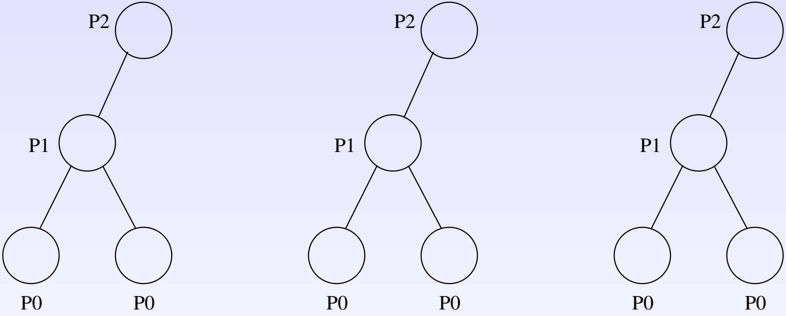
Partial factorisation

Send Contribution Block to Father

Endfor

"Multifrontal Method"

Multifrontal variant



Fan-in. Fan-out. Multifrontal.

Figure: Communication schemes for the three approaches.

Multifrontal variant

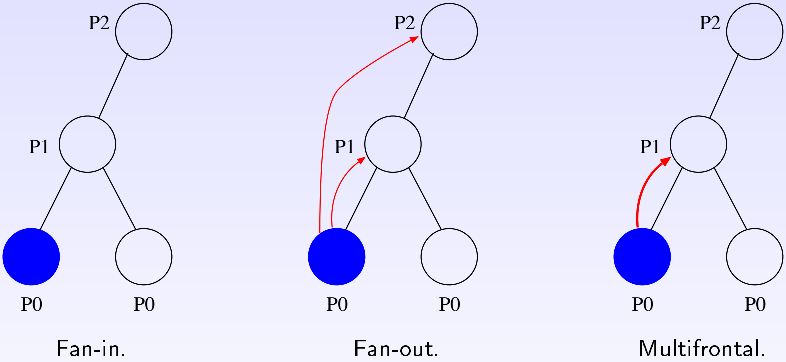


Figure: Communication schemes for the three approaches.

Multifrontal variant

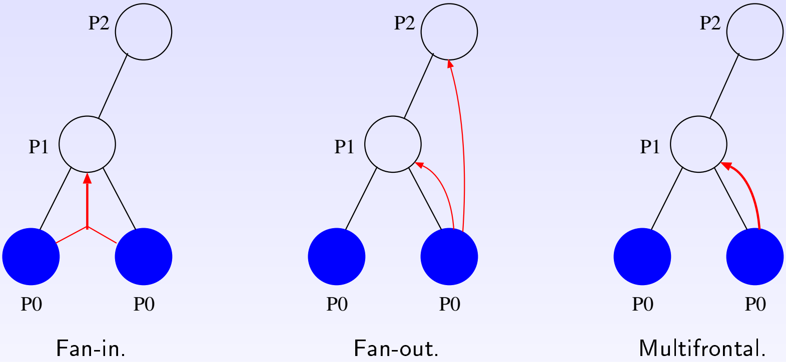


Figure: Communication schemes for the three approaches.

Multifrontal variant

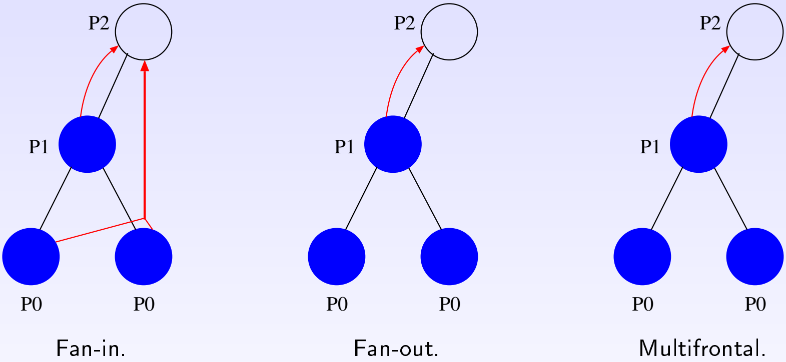
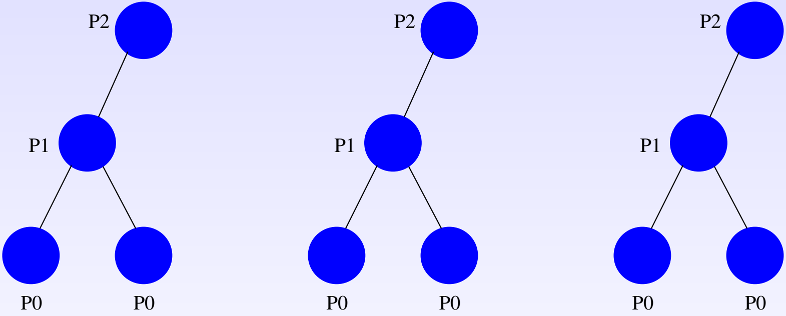


Figure: Communication schemes for the three approaches.

Multifrontal variant



Fan-in. Fan-out. Multifrontal.

Figure: Communication schemes for the three approaches.

Approaches for parallel factorization

- Elimination trees
- Distributed memory sparse solvers
- Some parallel solvers
- Case study : comparison of MUMPS and SuperLU

Some parallel solvers

Distributed-memory sparse direct codes

Code	Technique	Scope	Availability (www.)
DSCPACK	Multifr./Fan-in	SPD	cse.psu.edu/∼raghavan/Dscpack
MUMPS	Multifrontal	SYM/UNS	MUMPS Bordeaux-Lyon-Toulouse
PaStiX	Fan-in	SPD	labri.fr/perso/ramet/pastix
PSPASES	Multifrontal	SPD	cs.umn.edu/∼mjoshi/pspases
SPOOLES	Fan-in	SYM/UNS	netlib.org/linalg/spooles
SuperLU	Fan-out	UNS	nersc.gov/∼xiaoye/SuperLU
S+	Fan-out†	UNS	cs.ucsb.edu/research/S+
WSMP †	Multifrontal	SYM	IBM product

‡ Only object code is available.

Distributed-memory sparse direct codes

Code	Technique	Scope	Availability (www.)
DSCPACK	Multifr./Fan-in	SPD	cse.psu.edu/~raghavan/Dscpack
MUMPS	Multifrontal	SYM/UNS	MUMPS Bordeaux-Lyon-Toulouse
PaStiX	Fan-in	SPD	labri.fr/perso/ramet/pastix
PSPASES	Multifrontal	SPD	cs.umn.edu/~mjoshi/pspases
SPOOLES	Fan-in	SYM/UNS	netlib.org/linalg/spooles
SuperLU	Fan-out	UNS	nersc.gov/~xiaoye/SuperLU
S+	Fan-out†	UNS	cs.ucsb.edu/research/S+
WSMP †	Multifrontal	SYM	IBM product

Case study : Comparison of MUMPS and SuperLU

Approaches for parallel factorization

- Elimination trees
- Distributed memory sparse solvers
- Some parallel solvers
- Case study : comparison of MUMPS and SuperLU

MUMPS (Multifrontal sparse solver)

http://mumps.enseeiht.fr or
http://graal.ens-lyon.fr/MUMPS

1. Analysis and Preprocessing
   - Preprocessing (max. transversal, scaling)
   - Fill-in reduction on $\mathbf{A} + \mathbf{A}^T$
   - Partial static mapping (elimination tree) with dynamic scheduling during factorization.
2. Factorization
   - Multifrontal (elimination tree of $\mathbf{A} + \mathbf{A}^T$)
     $Struct(\mathbf{L}) = Struct(\mathbf{U})$
   - Partial threshold pivoting
   - Node and tree level asynchronous parallelism
     - Partitioning (1D Front - 2D Root)
     - Dynamic distributed scheduling
3. Solution step and iterative refinement

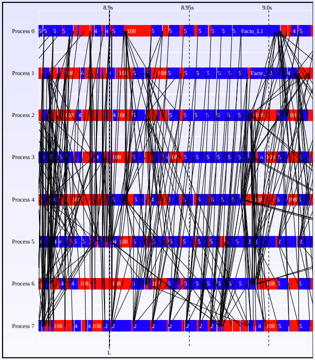
SuperLU (Gaussian elimination with static pivoting)

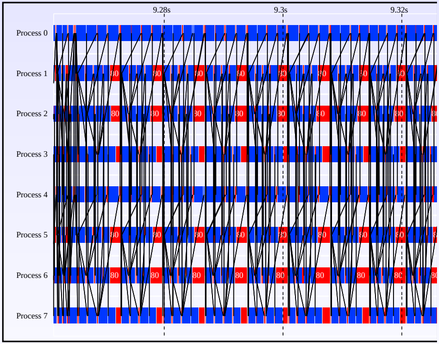
X.S. Li and J.W. Demmel

1. **Analysis and Preprocessing**
   - Preprocessing (Max. transversal, scaling)
   - Fill-in reduction on $\mathbf{A} + \mathbf{A}^T$
   - Static mapping on a 2D grid of processes
2. **Factorization**
   - Fan-out based on elimination DAGs (preserves unsymmetry)
   - Static pivoting

     if $(|a_{ii}| < \sqrt{\varepsilon}\|\mathbf{A}\|)$ **set** a_{ii} to $\sqrt{\varepsilon}\|\mathbf{A}\|$
   - 2D irregular block cyclic partitioning (based on supernode structure)
   - Pipelining / BLAS3 based factorization
3. **Solution step and iterative refinement**

Traces of execution(bbmat, 8 proc. CRAY T3E)





Influence of maximum wheighted matching MC64 on flops (10^9) for factorization (AMD ordering)

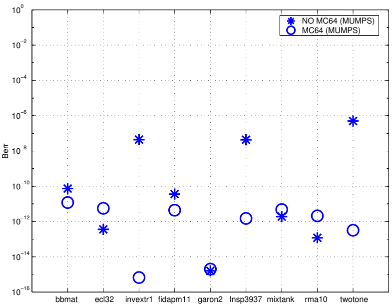
Matrix	MC64	StrSym	MUMPS	SuperLU
lhr71c	**No**	0	1431.0(*)	–
	Yes	21	1.4	0.5
twotone	**No**	28	1221.1	159.0
	Yes	43	29.3	8.0
fidapm11	**No**	100	9.7	8.9
	Yes	29	28.5	22.0

(*) Estimated during analysis,

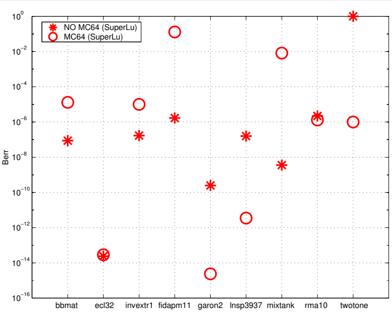
– Not enough memory to run the factorization.

Backward error analysis : $Berr = \max_i \frac{|r|_i}{(|A| \cdot |x| + |b|)_i}$

MUMPS



SuperLU

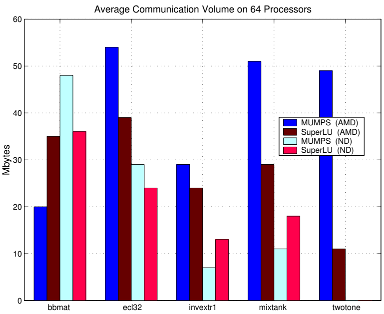


One step of iterative refinement generally leads to $Berr \approx \varepsilon$

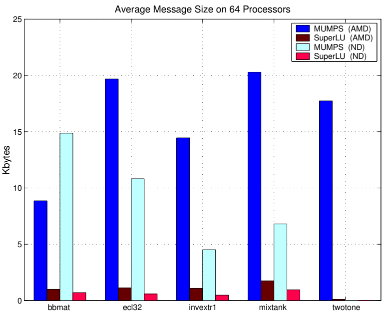
Cost (1 step of iterative refinement) $\approx$ Cost ($\mathbf{LU}x = b - \mathbf{A}x$)

Communication issues

Average Vol.(64 procs)

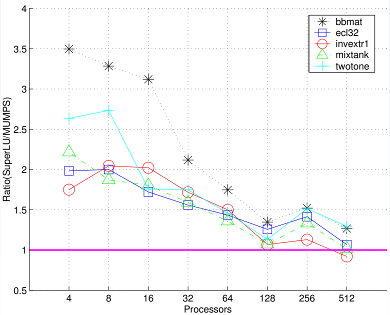


Average Message Size (64 procs)

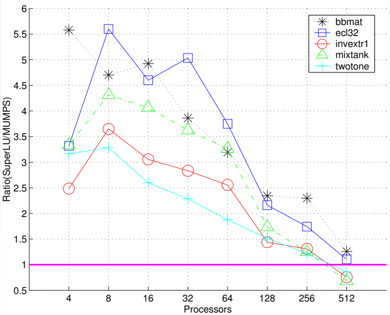


Time Ratios of the numerical phases
Time(SuperLU) / Time(MUMPS)

Factorization



Solve



Summary

- **Sparsity and Total memory**
  -`SuperLU` preserves better sparsity
  -`SuperLU` ($\approx 20\%$) less memory on 64 Procs (Asymmetry - Fan-out/Multifrontal)
- **Communication**
  -Global volume is comparable
  -`MUMPS` : much smaller ($/10$) nb of messages
- **Factorization / Solve time**
  -`MUMPS` is faster on nprocs ≤ 64
  -`SuperLU` is more scalable
- **Accuracy**
  -`MUMPS` provides a better initial solution
  -`SuperLU` : one step of iter. refin. often enough

Outline

Conclusion (Part I)

Sparse solver : only a black box ?

Default (often automatic/adaptive) setting of the options is often available ; However, a better knowledge of the options can help the user to further improve its solution.

- Preprocessing options are critical to both performance and accuracy.
- Preprocessing may influence :
  - Operation cost and/or computational time
  - Size of factors and/or memory needed
  - Reliability of our estimations
  - Numerical accuracy.
- Therefore, not a real black box . . .
- Even if in general more a black box than most iterative solvers . . .

Sparse solver : only a black box ?

Default (often automatic/adaptive) setting of the options is often available ; However, a better knowledge of the options can help the user to further improve its solution.

- Preprocessing options are critical to both performance and accuracy.
- Preprocessing may influence :
  - Operation cost and/or computational time
  - Size of factors and/or memory needed
  - Reliability of our estimations
  - Numerical accuracy.
- Therefore, not a real black box . . .
- Even if in general more a black box than most iterative solvers . . .

Sparse solver : only a black box ?

Default (often automatic/adaptive) setting of the options is often available; However, a better knowledge of the options can help the user to further improve its solution.

- Preprocessing options are critical to both performance and accuracy.
- Preprocessing may influence :
  - Operation cost and/or computational time
  - Size of factors and/or memory needed
  - Reliability of our estimations
  - Numerical accuracy.
- Therefore, not a real black box . . .
- Even if in general more a black box than most iterative solvers . . .

Direct solver : also kernels for iterative solvers ?

Direct

- Very general/robust
  - Numerical accuracy
  - Irregular/unstructured problems
- Factorization of A
  - May be costly (memory/flops)
  - Factors can be reused for multiple/successive right-hand sides

Iterative

- Efficiency depends on :
  - Convergence preconditioning
  - Numerical prop./struct. of A
- Rely on efficient Mat-Vect product
  - Memory effective
  - Successive right-hand sides is problematic

Hybrid approaches
(Domain Decompostion, Schur, Block Cimmino)

often strongly rely on both iterative and direct technologies

Outline

Appendix

Unsymmetric test problems

	Order	nnz	$nnz(L\|U)$ $\times 10^6$	Ops $\times 10^9$	Origin
conv3d64	836550	12548250	2693.9	23880	CEA/CESTA
fidapm11	22294	623554	11.3	4.2	Matrix market
lhr01	1477	18427	0.1	0.007	UF collection
qimonda07	8613291	66900289	556.4	45.7	QIMONDA AG
twotone	120750	1206265	25.0	29.1	UF collection
ultrasound80	531441	33076161	981.4	3915	Sosonkina
wang3	26064	177168	7.9	4.3	Harwell-Boeing
xenon2	157464	3866688	97.5	103.1	UF collection

Ops and $nnz(L|U)$ when provided obtained with `METIS` and default `MUMPS` input parameters.
UF Collection : University of Florida sparse matrix collection.
Harwell-Boeing : Harwell-Boeing collection.

PARASOL : Parasol collection

Symmetric test problems

	Order	nnz	$nnz(L)$ $\times 10^6$	Ops $\times 10^9$	Origin
audikw_1	943695	39297771	1368.6	5682	PARASOL
brgm	3699643	155640019	4483.4	26520	BRGM
coneshl2	837967	22328697	239.1	211.2	Samtech S.A.
coneshl	1262212	43007782	790.8	1640	Samtech S.A.
cont-300	180895	562496	12.6	2.6	Maros & Meszanos
cvxqp3	17500	69981	6.3	4.3	CUTEr
gupta2	62064	4248386	8.6	2.8	A. Gupta, IBM
ship_003	121728	4103881	61.8	80.8	PARASOL
stokes128	49666	295938	3.9	0.4	Arioli
thread	29736	2249892	24.5	35.1	PARASOL