Science & Technology Facilities Council

Sparse Matrices
Direct methods

Iain Duff

STFC Rutherford Appleton Laboratory and CERFACS

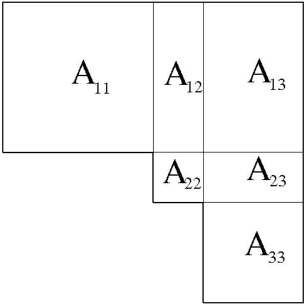
Summer School
The 6th de Brùn Workshop. Linear Algebra and Matrix Theory: connections, applications and computations. NUI Galway, Ireland. 3-7 December 2012.

Introduction

Lecture 2

- Block triangular form
- Symbolic factorization
- Elimination and assembly trees
- Multifrontal elimination

Block Triangular Form



The overall system is solved through the steps ....

$$
\begin{aligned}
A_{33}X_3 &= B_3 \\
A_{22}X_2 &= B_2 - A_{23}X_3 \\
A_{11}X_1 &= B_1 - A_{13}X_3 - A_{23}X_3
\end{aligned}
$$

So only A_{11}, A_{22}, and A_{33} are involved in the solution operations.
The off-diagonal blocks are ONLY used in matrix-matrix multiplies.

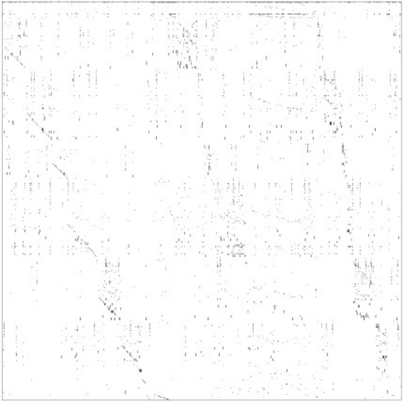
Block Triangular Form

Linear Programming

Econometrics

Chemical Engineering

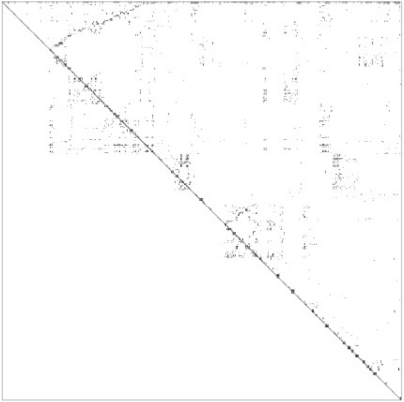
Block Triangular Form



Linear Programming; BP1600

Unordered BP1600 matrix

Block Triangular Form



BP1600 matrix reordered to BTF

Benefits of Block Triangular Form

Times in seconds on an HP/Compaq Alpha DS 20 workstation

Matrix	SHYY161	LHR71C
Order	76480	70304
Entries	329762	1528092
Entries in factors		
Using BTF	8845668	7880997
No BTF	10864045	8947643
Analyse/Factorize time		
Using BTF	144	18
No BTF	222	33
Factorize time		
Using BTF	23	4
No BTF	38	7

Reduction to Block Triangular Form

A matrix is reducible if there exists a permutation matrix **P** such that

$$\mathbf{PAP^T}$$

is in block triangular form.

A matrix is bireducible if there exists a permutation matrices **P** and **Q** such that

$$\mathbf{PAQ}$$

is in block triangular form.

Theorem 1
A matrix is bireducible if and only if any permutation of it with a zero-free diagonal is reducible.

Theorem 2
The block triangular form is essentially independent of this initial permutation.

An algorithm for obtaining the block triangular form

Because of theorems 1 and 2, we can split the calculation of the permutation into two steps.

Step 1.

Find a column permutation of the matrix so that the resulting permuted form has a zero-free diagonal.

$$\mathbf{A} \longrightarrow \mathbf{AQ} = \mathbf{A_1}$$

where $\mathbf{A_1}$ has a zero-free diagonal.

Worst case bound of best algorithm is $\mathcal{O}(n^{\frac{1}{2}}\tau)$ [n: order; τ: number of entries] but most algorithms are $\mathcal{O}(n\tau)$ [worst case bound]

Step 2.

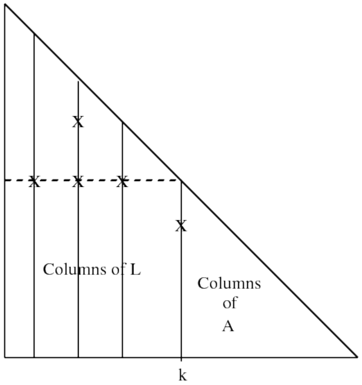
Find symmetric permutation of $\mathbf{A_1}$ to block triangular form, viz.

$\mathbf{PA_1P^T} = \mathbf{A_2}$ where $\mathbf{A}_2$ is in block triangular form.

Depth first search algorithm of Tarjan is $\mathcal{O}(n) + \mathcal{O}(\tau)$.

Symbolic Factorization

$$A \to LL^T$$



Generate columns of L one at a time in sequence.

At stage k the structure of column k of L is the union of the structure of column k of A with previous columns of L whose FIRST nonzero is in row k.

Elimination Tree

An elimination tree is fundamentally associated with the symbolic factorization of a sparse symmetric matrix.

It can be generated from any matrix and ordering but subsequently only imposes a partial ordering.

We will use this tree, or modifications thereof, in our factorization scheme that we shortly discuss.

Generation of elimination tree

Each node corresponds to a column of the matrix/factor. We generate the structure of L ($A = LDL^T$) at the same time as the elimination tree.

for k from 1 to n do .....

> Generate structure of column k of L from symbolic sum of columns from 1 to $k-1$ which have their **first** nonzero entry in row k.
>
> Draw edges in graph between node k and the nodes corresponding to columns just used above.

Note that a node corresponding to any column k with no nonzero entry in columns 1 to $k-1$ of row k will be a leaf node of this tree.

Note also that the complexity of this algorithm is clearly

$$\mathcal{O}\,(n) + \mathcal{O}\,(\tau)$$

Some properties of the elimination tree

- The elimination tree is a spanning tree of the elimination graph.
- All nodes corresponding to columns j with $l_{kj} \neq 0$, $j < k$ are descendants of the node corresponding to column k.
- The tree can be considered as an information flow or computational graph for the factorization.
- Eliminations at any leaf node can proceed immediately and simultaneously.
- Eliminations at “unrelated” nodes are independent.
- When all the children of a node have been processed, the parent can be processed.

Elimination Tree

The elimination tree can be used as a computational tree where there is a single elimination at each node.

However, it is computationally more efficient to consider eliminating a block at a time and this would correspond to merging some of the nodes of the elimination tree.

This results in an assembly tree.

MULTIFRONTAL METHOD

Direct method ... LU factorization
“Independent” of ordering

$$
\begin{array}{cccccccc}
x & x & x & x & & x & & \\
x & x & x & x & x & & x & x \\
x & x & x & & x & & & x \\
 & . & & & & & & \\
 & . & & & & & & \\
 & . & & & & & & \\
\multicolumn{3}{c}{\textit{Symm.}} & & & & &
\end{array}
$$

[only nonzeros marked]

Gather together

MULTIFRONTAL METHOD

$$
\begin{array}{ccc|ccccc}
x & x & x & x & 0 & x & 0 & 0 \\
x & x & x & x & x & 0 & x & x \\
x & x & x & 0 & x & 0 & 0 & x \\
\hline
x & x & 0 & & & & & \\
0 & x & x & & & & & \\
x & 0 & 0 & & & & & \\
0 & x & 0 & & & & & \\
0 & x & x & & & & &
\end{array}
$$

Perform eliminations on dense “frontal” matrix, choosing pivots from top left-hand block.

Multifrontal method

$$
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\times & & \times & \times \\
& \times & \times & \times \\
\times & \times & \times & \\
\times & \times & & \times
\end{array}
$$

First step

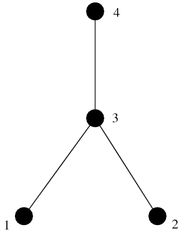
$$
\begin{array}{ccc}
1 & 3 & 4 \\
\times & \times & \times \\
\times & \square & \square \\
\times & \square & \square
\end{array}
$$

Second step

$$
\begin{array}{ccc}
2 & 3 & 4 \\
\times & \times & \times \\
\times & \square & \square \\
\times & \square & \square
\end{array}
$$

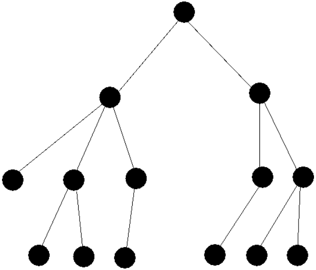
are “totally” independent

Dependence given by elimination tree



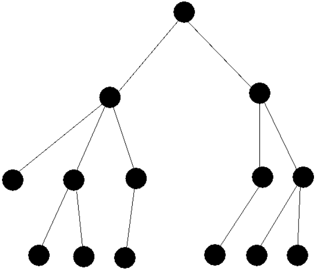
Multifrontal method

ASSEMBLY TREE



Multifrontal method

ASSEMBLY TREE

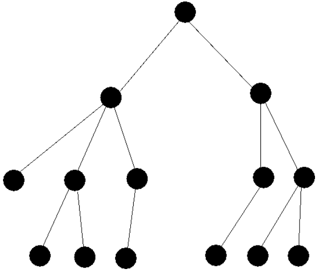


AT EACH NODE

$$\begin{array}{|c|c|} \hline F_{11} & F_{12} \\ \hline F_{12}^T & F_{22} \\ \hline \end{array}$$

Multifrontal method

ASSEMBLY TREE

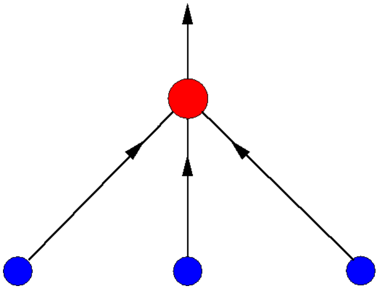


AT EACH NODE

$$\begin{array}{|c|c|} \hline F_{11} & F_{12} \\ \hline F_{12}^T & F_{22} \\ \hline \end{array}$$

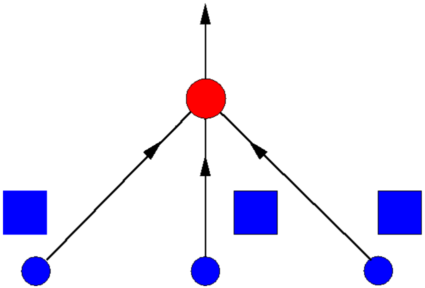
$$F_{22} \leftarrow F_{22} \; - \; F_{12}^T F_{11}^{-1} F_{12}$$

Multifrontal method



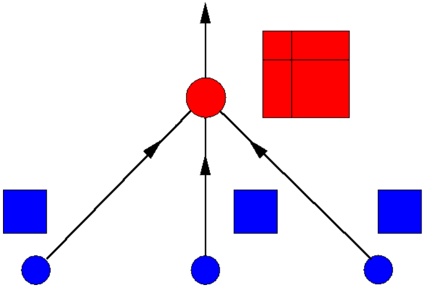
- From children to parent

Multifrontal method



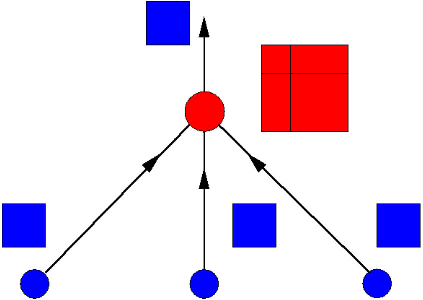
- From children to parent
- **ASSEMBLY**
  Gather/Scatter operations
  (indirect addressing)

Multifrontal method



- From children to parent
- **ASSEMBLY**
  Gather/Scatter operations (indirect addressing)
- **ELIMINATION** Full Gaussian elimination, Level 3 BLAS (TRSM, GEMM)

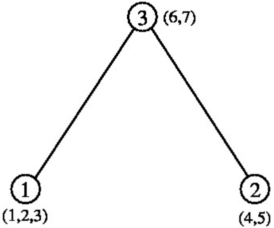
Multifrontal method



- From children to parent
- **ASSEMBLY**
  Gather/Scatter operations (indirect addressing)
- **ELIMINATION** Full Gaussian elimination, Level 3 BLAS (TRSM, GEMM)

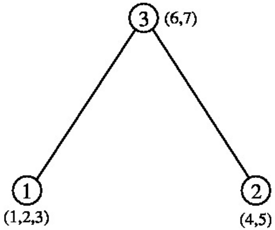
Small example of using an assembly tree

$$
\begin{matrix}
\times & \times & \times & \bullet & \bullet & \times & \bullet \\
\times & \times & \times & \bullet & \bullet & \bullet & \times \\
\times & \times & \times & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \times & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \bullet & \bullet \\
\times & \bullet & \bullet & \times & \bullet & \times & \times \\
\bullet & \times & \bullet & \bullet & \bullet & \times & \times
\end{matrix}
$$



Small example of using an assembly tree

×	×	×	•	•	×	•
×	×	×	•	•	•	×
×	×	×	•	•	•	•
•	•	•	×	×	×	•
•	•	•	×	×	•	•
×	•	•	×	•	×	×
•	×	•	•	•	×	×

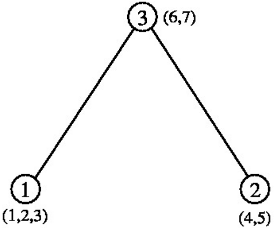


At step 1

	1	2	3	6	7
1	×	×	×	×	
2	×	×	×	×	×
3	×	×	×	×	×
6	×	×	×	×	×
7		×	×	×	×

Small example of using an assembly tree

$$\begin{array}{ccccccc}
\times & \times & \times & \bullet & \bullet & \times & \bullet \\
\times & \times & \times & \bullet & \bullet & \bullet & \times \\
\times & \times & \times & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \times & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \bullet & \bullet \\
\times & \bullet & \bullet & \times & \bullet & \times & \times \\
\bullet & \times & \bullet & \bullet & \bullet & \times & \times
\end{array}$$

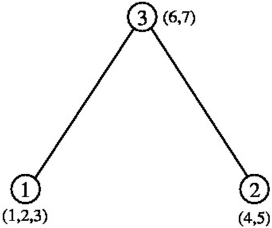


At step 2

$$\begin{array}{c|ccc}
 & 4 & 5 & 6 \\
\hline
4 & \times & \times & \times \\
5 & \times & \times & \times \\
6 & \times & \times & \times
\end{array}$$

Small example of using an assembly tree

$$
\begin{array}{ccccccc}
\times & \times & \times & \bullet & \bullet & \times & \bullet \\
\times & \times & \times & \bullet & \bullet & \bullet & \times \\
\times & \times & \times & \bullet & \bullet & \bullet & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \times & \bullet \\
\bullet & \bullet & \bullet & \times & \times & \bullet & \bullet \\
\times & \bullet & \bullet & \times & \bullet & \times & \times \\
\bullet & \times & \bullet & \bullet & \bullet & \times & \times
\end{array}
$$



At step 3

$$
\begin{array}{ccc}
 & 6 & 7 \\
6 & \times & \times \\
7 & \times & \times
\end{array}
$$

$$
+ \begin{array}{ccc}
 & 6 & 7 \\
6 & \times & \times \\
7 & \times & \times
\end{array}
+ \begin{array}{cc}
 & 6 \\
6 & \times
\end{array}
\longrightarrow
\begin{array}{ccc}
 & 6 & 7 \\
6 & \times & \times \\
7 & \times & \times
\end{array}
$$

Analysis

Perform the analysis on the pattern of

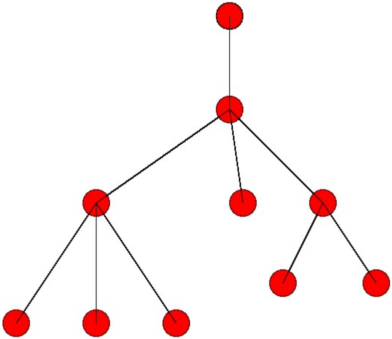
$$A + A^T$$

Can generate assembly tree symbolically using any sparsity preserving ordering

Then perform numerical pivoting *a posteriori* on the tree.
Note the flexibility afforded by the assembly tree

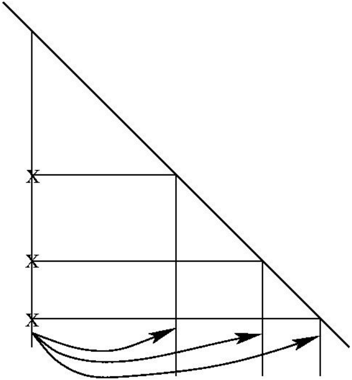
If we amalgamate two nodes, we can reduce the amount of indirect addressing although we will normally increase the number of elimination operations.
This flexibility can be useful in tailoring code to different architectures.

Assembly Tree



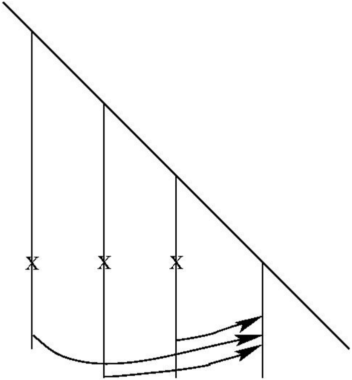
ASSEMBLY TREE

Cholesky factorization



Fan-out

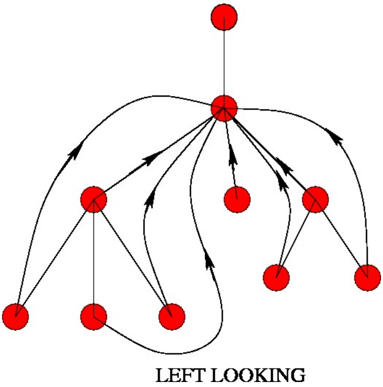
Right-looking



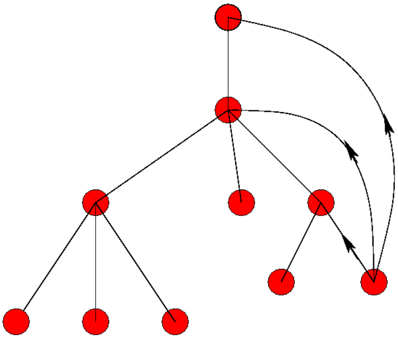
Fan-in

Left-looking

Assembly Tree

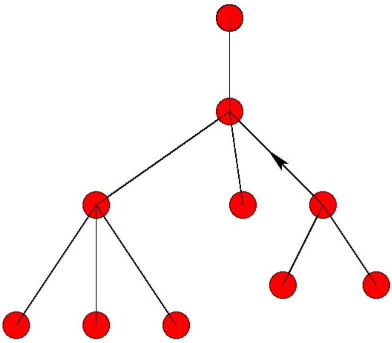


Assembly Tree



RIGHT–LOOKING

Assembly Tree



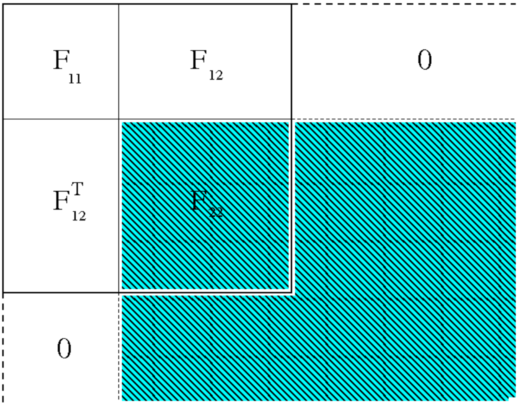
MULTIFRONTAL

Multifrontal method

$$\begin{array}{|c|c|} \hline F_{11} & F_{12} \\ \hline F_{12}^{T} & F_{22} \\ \hline \end{array}$$

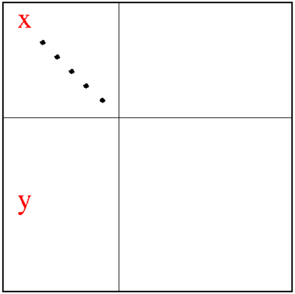
Pivot can only be chosen from F_{11} block since F_{22} is **NOT** fully summed.

Multifrontal method



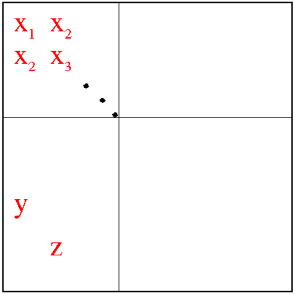
Situation wrt rest of matrix

Pivoting (1×1)



Choose x as 1×1 **pivot** if $|x| > u|y|$
where $|y|$ is the largest in column.

Pivoting (2×2)

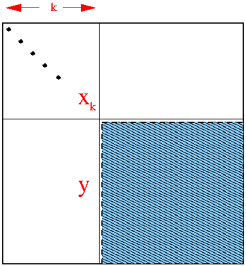


For the indefinite case, we can choose 2×2 **pivot** where we require

$$\left| \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \end{bmatrix}^{-1} \right| \begin{bmatrix} |y| \\ |z| \end{bmatrix} \leq \begin{bmatrix} \frac{1}{u} \\ \frac{1}{u} \end{bmatrix}$$

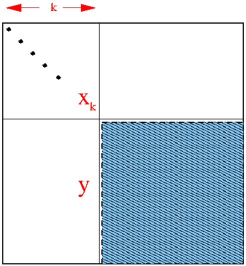
where again $|y|$ and $|z|$ are the largest in their columns.

Pivoting



If we assume that $k-1$ pivots are chosen but $|x_k| < u|y|$:

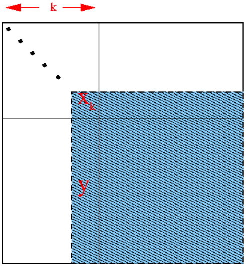
Pivoting



If we assume that $k-1$ pivots are chosen but $|x_k| < u|y|$:

- we can either take the **RISK** and use it or

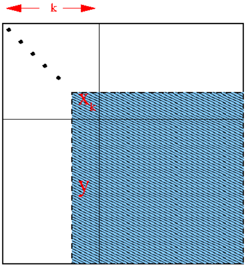
Pivoting



If we assume that $k-1$ pivots are chosen but $|x_k| < u|y|$:

- we can either take the **RISK** and use it or
- **DELAY** the pivot and then send to the parent a larger Schur complement.

Pivoting



If we assume that $k-1$ pivots are chosen but $|x_k| < u|y|$:

- we can either take the **RISK** and use it or
- **DELAY** the pivot and then send to the parent a larger Schur complement.

This can cause more work and storage

Threshold pivoting

Test for 1×1 pivot:

$$|f_{lk}| \geq u.\max_i |f_{ik}|, \text{ where } u \in (0,1]$$

Test for 2×2 pivot:

$$\left| \begin{pmatrix} f_{kk} & f_{kk+1} \\ f_{k+1k} & f_{k+1k+1} \end{pmatrix} \right| \begin{pmatrix} \max_{j \neq k,k+1} |f_{kj}| \\ \max_{j \neq k,k+1} |f_{kj}| \end{pmatrix} \leq \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}$$

where $0 < u \leq 0.5$

Symmetric indefinite matrices

These are matrices with both negative and positive eigenvalues and they will require numerical pivoting.
For example, the matrix

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

cannot be factorized by a Cholesky factorization.
However, we can always factorize an indefinite matrix if we allow both 1×1 and 2×2 pivots. The above matrix would be factorized using just one 2×2 pivot.

Pivoting for indefinite systems

If it is not possible to select pivots from F_{11} the factorization can continue but the pivots will be delayed.
Can be a problem if many "small" entries in F_{11}
This will often be the case for matrices of the form

$$\begin{pmatrix} \mathbf{H} & \mathbf{A}^{\mathbf{T}} \\ \mathbf{A} & \mathbf{0} \end{pmatrix}$$

Typically there are twice the number of entries in the factors than forecast but in a bad case (DTOC)

Number forecast:	187,639
Actual number:	4,714,248

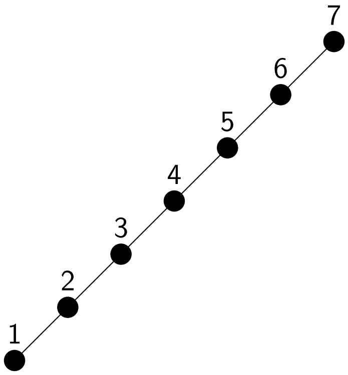
Parallelism from computational trees

TRIDIAGONAL MATRIX

$$\begin{bmatrix} X & X & & & & & \\ X & X & X & & & & \\ & X & X & X & & & \\ & & X & X & X & & \\ & & & X & X & X & \\ & & & & X & X & X \\ & & & & & X & X \end{bmatrix}$$

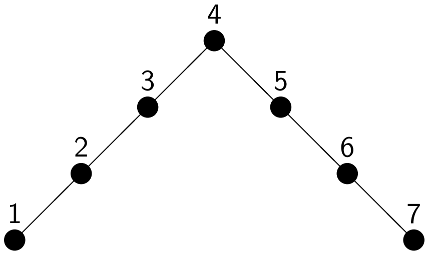
Parallelism from computational trees

Classical Gaussian elimination .. Thomas algorithm



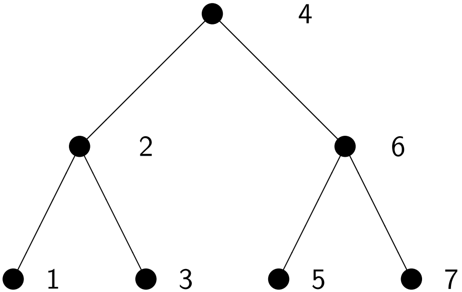
Parallelism from computational trees

LINPACK 7.4. .. BABE

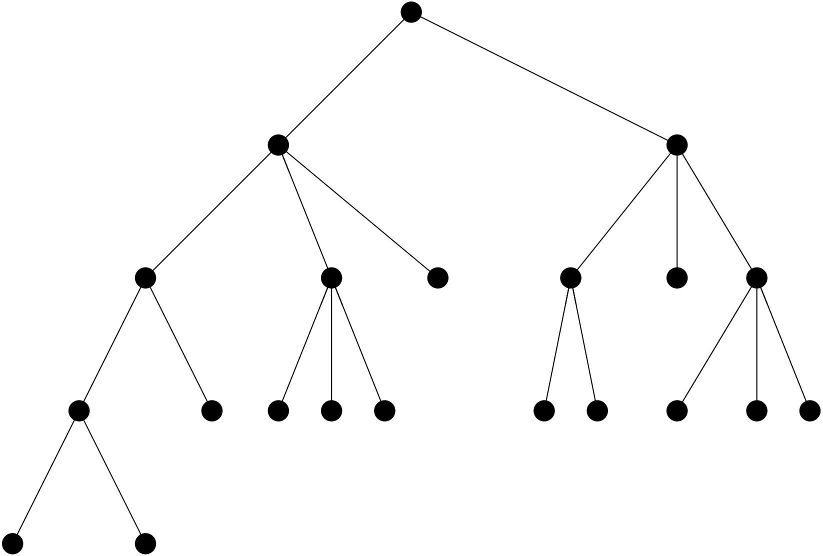


Parallelism from computational trees

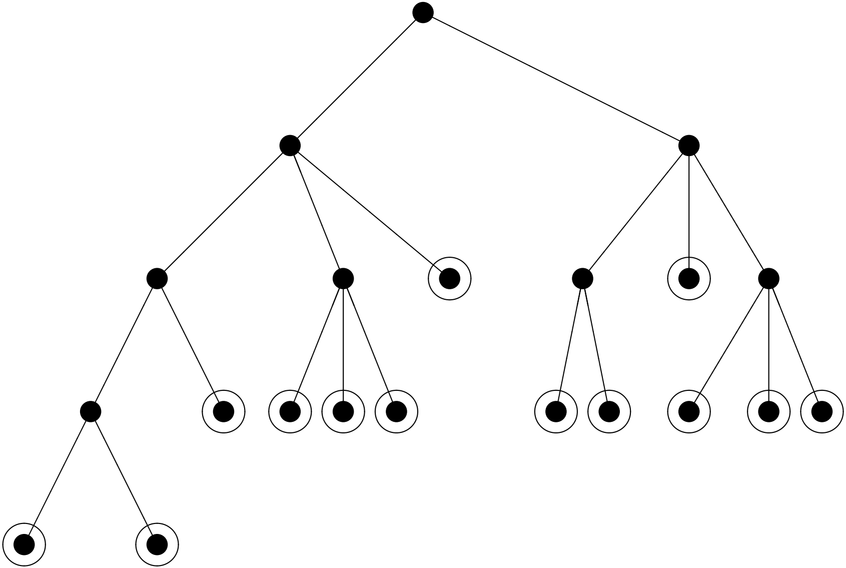
Nested dissection (odd-even reduction)



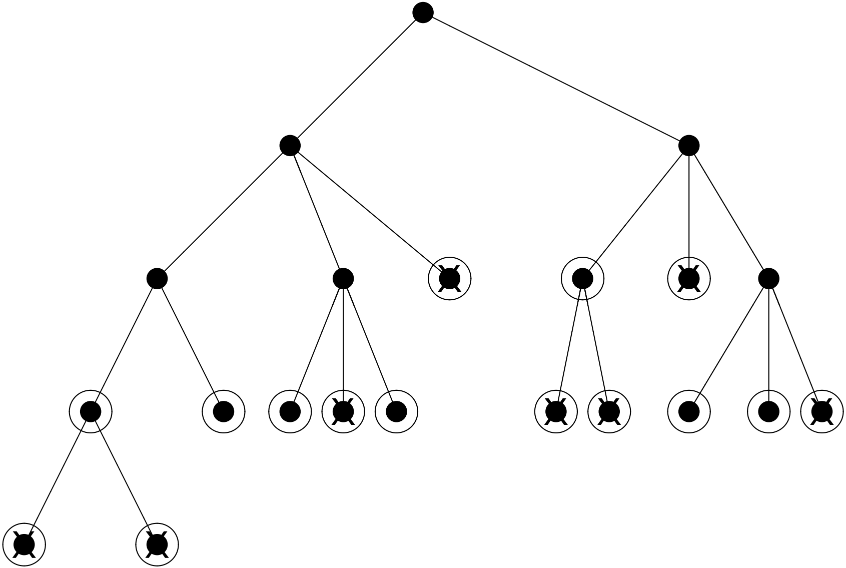
Parallel Implementation



Parallel Implementation



Parallel Implementation



Statistics on front sizes in tree

			Leaf nodes		Top 3 levels	
Matrix	Order	Tree nodes	Number	Av. size	Number	Av. size
BCSSTK15	3948	576	317	13	10	376
BCSSTK33	8738	545	198	5	10	711
BBMAT	38744	5716	3621	23	10	1463
GRE1107	1107	344	250	7	12	129
SAYLR4	3564	1341	1010	5	12	123
GEMAT11	4929	1300	973	10	112	148

Typically 75% work in top three levels.

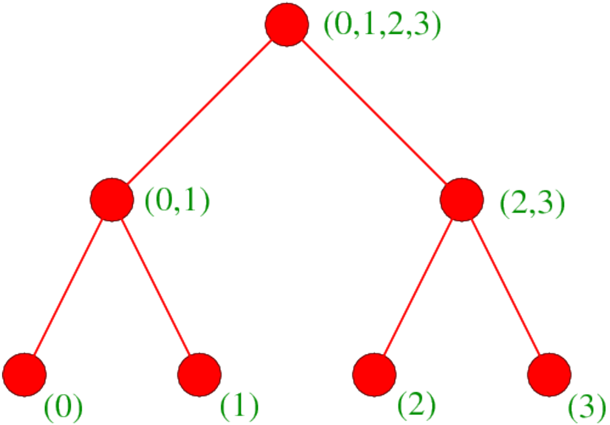
Two forms of parallelism

1. Tree parallelism [from assembly tree]
2. Node parallelism

$$\begin{bmatrix} \mathbf{F}_{11} & \mathbf{F}_{12} \\ \mathbf{F}_{21} & \mathbf{F}_{22} \end{bmatrix}$$

as in full linear algebra (eg ScaLAPACK)

Distributed memory



MUMPS

MUltifrontal
Massively
Parallel
Solver

Amestoy, Duff, Koster, and L'Excellent (2001)

MUMPS theses

Abdou Guermouche and Stéphane Pralet (2004)

Emmanuel Agullo and Mila Slavova (2008 and 2009)

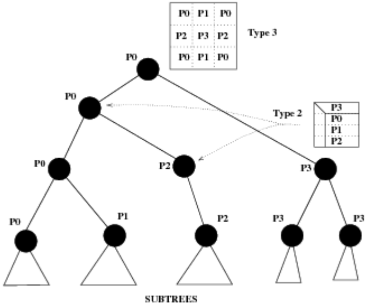
François-Henry Rouet (2012)

mumps@cerfacs.fr

and `http://mumps.enseeiht.fr/`

MUMPS .. levels of parallelism

- Type 1: Parallelism of the tree
- Type 2: 1D partitioning of frontal matrices with distributed assembly process
- Type 3: 2D partitioning of root nodes (ScaLAPACK)



MUMPS

Features of MUMPS code

- Element or equation entry
- Symmetric or unsymmetric
- Original matrix can be distributed
- Range of orderings and scalings
- Iterative refinement and error analysis
- Rank detection and null space basis
- Partial factorization and return of Schur complement

Available from:
`http://mumps.enseeiht.fr/`