PARALLEL
SEQUENTIAL NESTED DISSECTION
FOR SOLVING SYSTEM \( Ax = b \)

Outline of the talk:
1. Sequential solving
2. Parallelization
3. CM-implementation:
   - data mapping
   - algorithm
   - implementation of primitives (matrix operations in CM).

Lena Nekludova
TMC
June 1986
This is our problem:

Solve an equation $Ax = b$, $A$-n$x$n matrix, $b$-n-vector

where

1. $A$ is symmetric positive definite $\Rightarrow$
2. for any subset of variables, corresponding submatrix of $A$ has $\det > 0$. e.g.: $A = \begin{bmatrix} 1 & -5 & 3 \\ -5 & 2 & 0 \\ 3 & 0 & 10 \end{bmatrix}$ $\rightarrow$ $\begin{bmatrix} 1 & 3 \\ 3 & 10 \end{bmatrix}$; $\det = 1 > 0$.

2. $A$ is sparse $\Rightarrow O(n)$ elements are nonzeros.

3. $n > 1000$

4. Graph $(A)$ is known beforehand.

Def: Incidence graph Graph$(A)$ of matrix $A$:

Its vertices correspond to variables of $A$

edges to nonzero elements of $A$

E.g.: for $A$ above, Graph$(A)$ = $v_1 \rightarrow v_2$

Note: in most examples, I will be using matrices with Graph$(A)$ = grid.
DIRECT INVERSION: \( x = A^{-1}b \).

**CLAIM:** \( \exists! \) **DECOMPOSITION** \( A = L \cdot D \cdot U \),

where \( L = \begin{bmatrix} 1 & 0 & 0 \\ \ast & 1 & 0 \\ \ast & \ast & 1 \end{bmatrix} \), \( U = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \ast \end{bmatrix} \), \( D = \text{diag} \).

**COR:** \( A^{-1} = U^{-1} D^{-1} L^{-1} = (L^T)^{-1} D^{-1} L^{-1} \)

**HOW TO FIND** \( L^{-1} \) **AND** \( D^{-1} \)?

**BY GAUSSIAN ELIMINATION:**
GAUSSIAN ELIMINATION (NO PIVOTING)

\[
\begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & 0 & \cdots & 0
\end{bmatrix}
\]

1st COLUMN

\[
\begin{bmatrix}
1 & 0 & \cdots & 0 \\
-a_{21}/a_{11} & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
-a_{n1}/a_{11} & 0 & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
a_{11} & a_{12} & \cdots & a_{1n} \\
a_{21} & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & 0 & \cdots & 0
\end{bmatrix}
\]

2nd COLUMN

\[
L^{-1} = \begin{bmatrix}
1 & 0 & \cdots & 0 \\
0 & 1 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{bmatrix}
\begin{bmatrix}
a_{11} & d_{22} & \cdots & d_{nn} \\
0 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0
\end{bmatrix}
\]

\[
= D
\]

\[
A^{-1} = (L^T)^{-1}D^{-1}L^{-1}
\]
$L'$ can be dense even for sparse $L$.

Seg: $O(n^3)$

Par: $\frac{\text{SPACE/TIME}}{O(n^2) / O(n \log n)}$
Solve \( \text{c.e., each } L \), i.e. only \( (n) \) nonzero variables.

We want to choose their successors so that all \( L \) are here \( 5\), \( 6\), \( 7\),\ldots\( 5\)-af one some unordered of variables of

Thus decomposition is obtained by partial \( 4\). E. on \( 5\), \( 6\).

When \( x = A^{-1} G \)

Decomposition \( A' \) into \( 3d \) factors.
For many types of Graph(A) (e.g., for all planar graphs) can we shall choose $S_0, ..., S_d$ in some special way.

This will make all matrices in (a) sparse.

Namely, $S_0, ..., S_d$ will be constructed with the help of SEPARATOR TREE of A (no precise definition given here).
$O(\sqrt{n})$ - SEPARATOR FOR GRAPH/MATRIX

Separator tree has $d \cdot \log n$ levels. At level $i$, it gives decomposition of graph $G$ into $2^{d-i-1}$ subgraphs (not disjoint).
CLAIM: IF WE CHOOSE i-th SEPARATOR FOR S_i, then each factor in (\phi) has O(n) nonzero elements.

MOREOVER, WE CAN FIGURE OUT THEIR POSITIONS BY LOOKING AT THE SEP. TREE.
SEQUENTIAL NESTED DISSECTION:

- Construct separator tree for $G(A)$
- Factor $A^{-1}$ into product of $\sim 2\log n$ sparse $n \times n$ matrices.
- Solve: $x = A^{-1}b$

$\text{SEQ: } O(n^{3/2})$

$\text{PAR: } \text{SPACE/TIME} \leq O(n) / O(\sqrt{n})$

Need parallel nested dissection

That is, want to factor $A^{-1}$ in parallel

Recall: factorisation of $A^{-1}$ is obtained by partial Gaussian elimination
Recursive factorization of $A^{-1}$ by partial G.E.

Want: $A^{-1} = (L_1^T)^{-1} D_1^{-1} \cdots (L_{d-1}^T)^{-1} D_{d-1}^{-1} L_{d-1}^{-1} \cdots L_2^{-1} L_1^{-1}$ \( \star \)

Reorder variables of $A$ so that variables in the group $S_0$ come first.

- $A \xrightarrow{\text{eliminate}} S_0$

- $A_1 \xrightarrow{\text{eliminate}} S_1$

$L_1$, $D_1$, $A_1$, $L_2$, $D_2$, $A_2$, etc.
TREE-NODE AT LEVEL $i$ CONTAINS:

- SOME SUBSET OF $S_i$
- SUBSET OF $U_{k>i}S_k$, adjacent to $S_i$
FACTORING $A^{-1}$ in parallel:
- Construct the separator tree with $d = \log n$ levels.
- Split our matrix $A$ into the strict "sum" of small matrices. Variables = vertices in corresponding tree-nodes on level 0.
- Inductive step: for tree level $i = 0, \ldots, d$:
  1. For all small matrices in parallel, do partial G.E.
  2. As a result, for each small matrix obtain $L^{-1}$ and $A_i$ (see identity (9) above).
  All $L^{-1}$ are left on the current level; all $A_i$ are moved to the next level.
  3. Each $A_i$ is added up to its sibling in the separator tree.

Claim: After we are done, sum of all small matrices $L$ from level $i$ equals to $L^{-1}$ (see f).

In other words: if variables are chosen according to the sep. tree then G.E. and $\oplus$ commute with each other.

Handwaving proof: (i) fill-ins in G.E. correspond to certain paths in Graph (A).
  (ii) Paths in Graph (A) "agree" with separator tree.

Rigorous proof is not very straightforward. I couldn't find any papers on this, so I wrote it down myself. Notes are available—
MAIN IDEA (PAN-REIF)

- Split the matrix into (+) small matrices.
- **PRIMITIVE OPERATIONS REQUIRED for factoring** will commute with (+)

We can apply these operations to small matrices on different levels of the separator tree.

In the end all factors of $A'$ will be distributed among the tree-nodes.

Our Implementation:

We use identity (2), with primitive operation

- Gaussian elimination

They used different identity, with much more complicated primitive operation.
SPACE REQUIREMENTS:
1 matrix element per processor
log(n) levels of the tree
At each level, \[ \text{SPACE} = \text{const} \cdot n \]

For "nice" grid graphs, \( \text{const} = 25 \)
For "general" grids, \( \text{const} \approx 50 \)
For planar graphs, might be worse.

TIME REQUIREMENTS:
Worst case: level \( d-1 \) \( \text{TIME} = O(n^2) \)
Backsolve

All factors of $A^-$ are now computed and stored in the CM.

So, given vector $b$, we should be able to compute $A^-b = x$.

This computation is called back solving.
BACKSOLVE:  \( L^{-1} \) PART (GOING UP)

Each matrix occupies a square in CM and corresponding piece of vector \( \beta \) occupies a column to the left of it.
OCCUPIES A COLUMN TO THE LEFT OF IT AND CORRESPONDING PIECE OF VECTOR B EACH MATRIX OCCUPIES A SQUARE IN

BACKSOLVE: L PART (GOING UP)
BACKSOLVE: U"D" PART (GOING DOWN)

RUNNING TIME: BACKSOLVE:

\[ \log(\sqrt{n}) + \log(\sqrt{\frac{n}{2}}) + \ldots \sim \text{const} \cdot \log n \]
CM-Implementation - 2:

DATA MAPPING

Construct the separator tree for Graph (A).

DATA:

Each tree-nod at level i is a struct corresponding to some small submatrix of matrix A (see \( \mathbf{A}_e \)).

slots of struct contain:

- list of variables of this submatrix
- its sublist which we eliminate
- address of the corner of this submatrix in the CM
- a dozen of other things, later converted into pointers for the CM part of the algorithm

In CM:

allocate space for all submatrices
create a structure for operating on a set of matrices (by next we have a standard way of doing it)
create pointers for sending matrix and vector elements from level to level.

To do this:

First we sequentially load data from Lispm struct into corner like corners of matrices in the CM. Then we spread data to all processors of the same matrix.
**Complete algorithm**

(1) Create the necessary structures in LispM and CM (= data mapping)

(2) Initialize submatrices and pieces of vector b at 0 level of CM, so that their direct sum equals A and b correspondingly.

(3) Factor $A^{-1}$

(4) Backsolve: going up and down.

(5) Read the solution $X$ from CM

(The value of $x$ is contained in the columns to the left of submatrices, where we initially put $b$. Each processor knows to which variable it corresponds. Processors, corresponding to the same variables, contain the same value of $x$.

**Note**: 2 and 5 are done sequentially.