Improving multifrontal methods by means of low-rank approximation techniques

Joint work with Patrick Amestoy, Cleve Ashcraft, Olivier Boiteau, Alfredo Buttari and Jean-Yves L’Excellent, PhD started on October 1st, 2010 and financed by EDF.

Clément Weisbecker, ENSEEIHT-IRIT, University of Toulouse, France

SIAM LA12, Valencia, Spain, June 22, 2012
Improving **MULTIFRONTAL METHODS** by means of **LOW-RANK APPROXIMATION TECHNIQUES**

**Multifrontal solver**
- direct solver for large linear systems
- well known and studied

**Low-rank approximations**
- compression and flop reduction
- accuracy controlled by a numerical parameter

⇒ Combine these two notions to improve multifrontal solvers (in the context of **MUMPS**)

SIAM LA12, Valencia, Spain, June 22, 2012
The multifrontal method
1 variable of the matrix $A = 1$ node in the graph $G$

$A(i,j) \neq 0 \equiv$ an edge between nodes $i$ and $j$ exists
Nested dissection

\[ \begin{array}{ccc}
D_1 & S & D_2 \\
\end{array} \]

\[
\begin{array}{ccc}
D_1 & 0 & D_2 \\
D_1^S & D_2^S & S \\
\end{array}
\]

\[ D_1 \text{ and } D_2 \text{ are INDEPENDENT !} \]
Elimination of $D_i$:

- $S \leftarrow S + CB_1 + CB_2$
- variables of $S$ ready to be eliminated

$D_i \quad D_i^S(21)$

$D_i^S(12) \quad 0$

$U_i \quad U_i^S$

$L_i \quad L_i^S$

$CB_i$
At each node, a partial factorization of the frontal matrix is performed:

\[
\begin{align*}
&\begin{array}{c}
\text{assembly} \\
\oplus \\
\text{elim.}
\end{array} \\
\begin{array}{c}
\text{CB}_{D_1} \\
\text{CB}_{D_2}
\end{array} \\
\begin{array}{c}
\text{FS} \\
\text{CB}
\end{array}
\end{align*}
\]
Goal of this study

Reveal and exploit low-rank structures within the fronts
Reveal and exploit low-rank structures within the fronts

<table>
<thead>
<tr>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- memory efficient: $A_{m,n} = U_{m,k} V_{n,k}^T + E_\varepsilon$ (&quot;demote&quot;)</td>
</tr>
<tr>
<td>- computationally efficient: $AB = U_A (V_A^T U_B) V_B^T$</td>
</tr>
</tbody>
</table>
Goal of this study

Reveal and exploit low-rank structures within the fronts

Why?

- memory efficient: \( A_{m,n} = U_{m,k} V_{n,k}^T + E_\varepsilon \) ("demote")
- computationally efficient: \( AB = U_A (V_A^T U_B) V_B^T \)

Problem:

The fronts are FULL rank
Goal of this study

Reveal and exploit low-rank structures within the fronts

Why ?

- memory efficient: \( A_{m,n} = U_{m,k}V_{n,k}^T + E_\varepsilon \) (“demote”)
- computationally efficient: \( AB = U_A(V_A^T U_B)V_B^T \)

Problem:
The fronts are FULL rank

Solution:
Group variables to obtain low-rank subblocks
Grouping variables
Is it *theoretically* worth demoting the block \((t, s)\)?

**Admissibility condition for elliptic PDEs (Börm, Grasedyck, Hackbusch)**

Let \(t \subset \mathcal{I}\) and \(s \subset \mathcal{J}\) be subsets of indices. The block \((t, s)\) is said admissible if \(t\) and \(s\) satisfy

\[
\text{diam}(t) + \text{diam}(s) \leq 2 \eta \cdot \text{dist}(t, s)
\]

where \(\eta\) is a problem dependant fixed parameter (usually 1 or 0.5)
How to group the variables of a separator?

*Constraint*: the admissibility condition should be satisfied

- **large diameters**
  - fraction of memory used 83%

- **small diameters**
  - fraction of memory used 57%
Rank vs distance (1)
Rank vs distance (1)
Rank vs distance (1)
81x81 separator, 10x10 domains

singular values of submatrix as a function of distance

log_{10}(sv)

81 x 81 separator, 10 x 10 domains, abstol = 1.e-16

rank of submatrix as a function of distance
Halo algorithm to group a separator

- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool
Halo algorithm to group a separator

- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool

1. The separator
Halo algorithm to group a separator

- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool

1. The separator
2. The halo
Halo algorithm to group a separator

- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool

1. The separator
2. The halo
3. Extraction of the halo
Halo algorithm to group a separator

- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool

1. The separator
2. The halo
3. Extraction of the halo
4. Partition of the halo
Halo algorithm to group a separator

- Designed to catch the geometry of the problem
- Computed with the graph instead of the mesh
- Coupled with a third party partitioning tool

1. The separator
2. The halo
3. Extraction of the halo
4. Partition of the halo
5. Partition of the separator
   (block size is fixed)
How to group the variables of a front?

⇒ front = separator + border
How to group the variables of a front?

⇒ front = separator + border

1. separator : halo
How to group the variables of a front?

\[ \text{front} = \text{separator} + \text{border} \]

1- separator : halo

2- border ? 2 choices:
How to group the variables of a front?

\[ \text{front} = \text{separator} + \text{border} \]

1. separator: halo

2. border? 2 choices:

EXPLICIT

```
  S
```

15/25
How to group the variables of a front?

⇒ front = separator + border

1- separator : halo

2- border ? 2 choices :

EXPLICIT
How to group the variables of a front?

⇒ \( \text{front} = \text{separator} + \text{border} \)

1- separator : halo

2- border ? 2 choices :

EXPLICIT

INHERITED (\textit{top down})
How to group the variables of a front?

\[ \Rightarrow \text{front} = \text{separator} + \text{border} \]

1- separator : halo
2- border ? 2 choices :

- EXPLICIT
- INHERITED (top down)
How to group the variables of a front?

⇒ front = separator + border

1- separator : halo

2- border ? 2 choices :

EXPLICIT

INHERITED (top down)
How to group the variables of a front?

$$\Rightarrow \text{front} = \text{separator} + \text{border}$$

1- separator : halo

2- border ? 2 choices :

EXPLICIT

INHERITED (top down)
Comparison

• “inherited” version is more than 2 times faster
• “inherited” version is more than 2 times faster
• same results on $L_{11}$

• optimal $\times$ optimal = optimal block
Comparison

- “inherited” version is more than 2 times faster
- same results on $L_{11}$
- comparable results on $L_{21}$ ($\sim 2 - 3\%$ lost)

- optimal $\times$ optimal = optimal block
- small $\times$ optimal = large enough block
• “inherited” version is more than 2 times faster
• same results on $L_{11}$
• comparable results on $L_{21}$ ($\sim 2 - 3\%$ lost)
• a little less good on CBs ($\sim 10\%$ lost)

- optimal $\times$ optimal = optimal block
- small $\times$ optimal = large enough block
- small $\times$ small = too small block
Exploiting low-rank blocks
Many matrix structures ...

\(H\) and \(H^2\) matrices (Hackbusch et al.), HSS matrices (Li, Napov, Xia et al.), HBS matrices (Gillman & Martinsson), BLR (us !) ...

Block Low-Rank (BLR) ...

- has less constraints for the grouping;
- is more flexible (pivoting, ordering);
- seems more adapted to a multifrontal solver (small domains);
Experiments
Set of problems

<table>
<thead>
<tr>
<th>Name</th>
<th>Arith.</th>
<th>N</th>
<th>NZ</th>
<th>memory</th>
<th>flops</th>
<th>CSR (^{(1)})</th>
<th>appli.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curl 5000(^2)</td>
<td>D</td>
<td>50.10(^6)</td>
<td>2.10(^8)</td>
<td>29 GB</td>
<td>5.10(^{12})</td>
<td>2.10(^{-15})</td>
<td>▽</td>
</tr>
<tr>
<td>Geoazur 128(^3)</td>
<td>C</td>
<td>2.10(^6)</td>
<td>55.10(^9)</td>
<td>54 GB</td>
<td>6.10(^{13})</td>
<td>3.10(^{-4})</td>
<td>wave prop.</td>
</tr>
<tr>
<td>EDF_A_MECA_R12</td>
<td>D</td>
<td>134.10(^6)</td>
<td>1.10(^9)</td>
<td>151 GB</td>
<td>2.10(^{14})</td>
<td>4.10(^{-15})</td>
<td>mechanics</td>
</tr>
<tr>
<td>EDF_D_THER_R7</td>
<td>D</td>
<td>8.10(^6)</td>
<td>118.10(^9)</td>
<td>229 GB</td>
<td>1.10(^{14})</td>
<td>8.10(^{-15})</td>
<td>thermics</td>
</tr>
</tbody>
</table>

\(^{(1)}\) CSR = Componentwise Scaled Residual = \(\max_i \frac{|b - A\bar{x}|_i}{(|b| + |A| \cdot |\bar{x}|)_i}\)

- 1 toy and 3 real industrial problems
- different applications
- target: harder and harder problems ⇒ industrial partners
2D problems

<table>
<thead>
<tr>
<th>Name</th>
<th>N</th>
<th>memory</th>
<th>flops</th>
<th>CSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDF_A_MECA_R12</td>
<td>134.10^6</td>
<td>151 GB</td>
<td>2.10^{14}</td>
<td>4.10^{-15}</td>
</tr>
<tr>
<td>Curl-Curl 5000^2</td>
<td>50.10^6</td>
<td>29 GB</td>
<td>5.10^{12}</td>
<td>2.10^{-15}</td>
</tr>
</tbody>
</table>
3D problems

<table>
<thead>
<tr>
<th>Name</th>
<th>N</th>
<th>memory</th>
<th>flops</th>
<th>CSR</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDF_D_THER_R7</td>
<td>8.10^6</td>
<td>229 GB</td>
<td>1.10^{14}</td>
<td>8.10^{-15}</td>
</tr>
<tr>
<td>Geoazur 128^3</td>
<td>2.10^6</td>
<td>54 GB</td>
<td>6.10^{13}</td>
<td>3.10^{-4}</td>
</tr>
</tbody>
</table>
Conclusion & perspectives

⇒ efficient method on applicative problems

• good memory reduction & large decrease in computations:
  150 GB → 60 GB  7 H → 1 H
• tree dependent method (any global ordering works)
⇒ can be used as a preconditioner or as an accurate solver

⇒ in parallel, significant part of memory consumption is temporary data
⇒ rooms for improvements (CB compression)

• pivoting
• diversify tested problems
• error propagation study
• **Olivier Boiteau** (EDF R&D) for the matrices
• **Stéphane Operto** (SEISCOPE Project) for the 3D Geoazur generator
• the **Toulouse Computing Center** (CICT) and **Nicolas Renon**
• the **LBNL** (Berkeley)
¡Muchas gracias!

Any questions?