Solution of dynamic solid deformation using hybrid parallelization with MPI and OpenMP

MSc. Miguel Vargas-Félix

ISUM 2012
We want to solve large scale dynamic problems with linear deformation modeled with the finite element method.

\[
\varepsilon = \begin{pmatrix}
\frac{\partial}{\partial x} & 0 & 0 \\
0 & \frac{\partial}{\partial y} & 0 \\
0 & 0 & \frac{\partial}{\partial z}
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_2 \\
u_3
\end{pmatrix}
\]

\[
\sigma = D (\varepsilon - \varepsilon_0) + \sigma_0
\]

Where \(u\) is the displacement vector, \(\varepsilon\) the stain, \(\sigma\) the stress. \(D\) is called the constitutive matrix. The solution is found using the finite element method with the Galerkin weighted residuals.
Schur substructuring method

This is a domain decomposition method without overlapping [Krui04].

We start with a system of equations resulting from a finite element problem
\[ K d = f, \] 
where \( K \) is a symmetric positive definite matrix of size \( n \times n \).

If we divide the geometry into \( p \) partitions, the idea is to split the workload to let each partition to be handled by a computer in the cluster.
We can arrange (reorder variables) of the system of equations to have the following form

\[
\begin{pmatrix}
K_{1}^{II} & 0 & 0 & K_1^{IB} \\
0 & K_2^{II} & 0 & K_2^{IB} \\
0 & 0 & K_3^{II} & K_3^{IB} \\
K_1^{BI} & K_2^{BI} & K_3^{BI} & \cdots & K_p^{BI} & K_p^{IB} & \cdots & K_p^{IB} & K_p^{BB} \\
\end{pmatrix}
\begin{pmatrix}
d_1^I \\
d_2^I \\
d_3^I \\
\vdots \\
d_p^I \\
\end{pmatrix}
= 
\begin{pmatrix}
f_1^I \\
f_2^I \\
f_3^I \\
\vdots \\
f_p^I \\
\end{pmatrix}.
\]

The superscript II denotes entries that capture the relationship between nodes inside a partition. BB is used to indicate entries in the matrix that relate nodes on the boundary. Finally IB and BI are used for entries with values dependent of nodes in the boundary and nodes inside the partition.

Figure 1. Partitioning example.
Thus, the system can be separated in \( p \) different systems,

\[
\begin{pmatrix}
    K_i^{II} & K_i^{IB} \\
    K_i^{IB} & K_i^{BB}
\end{pmatrix}
\begin{pmatrix}
    d_i^I \\
    d^B
\end{pmatrix}
= \begin{pmatrix}
    f_i^I \\
    f^B
\end{pmatrix}, \quad i = 1 \ldots p.
\]

For each partition \( i \) the vector of unknowns \( d_i^I \) as

\[
d_i^I = \left( K_i^{II} \right)^{-1} \left( f_i^I - K_i^{IB} d^B \right).
\]

(3)

After applying Gaussian elimination by blocks on (2), the reduced system of equations becomes

\[
\begin{pmatrix}
    K_i^{BB} - \sum_{i=1}^{p} K_i^{IB} \left( K_i^{II} \right)^{-1} K_i^{IB}
\end{pmatrix}
\begin{pmatrix}
    d^B
\end{pmatrix}
= f^B - \sum_{i=1}^{p} K_i^{IB} \left( K_i^{II} \right)^{-1} f_i^I.
\]

(4)

Once the vector \( d^B \) is computed using (4), we can calculate the internal unknowns \( d_i^I \) with (3).

It is not necessary to calculate the inverse in (4).

Let’s define \( \bar{K}_i^{BB} = K_i^{IB} \left( K_i^{II} \right)^{-1} K_i^{IB} \), to calculate it [Sori00], we proceed column by column using an extra vector \( t \), and solving for \( c = 1 \ldots n \)

\[
K_i^{II} t = \left[ K_i^{IB} \right]_c,
\]

(5)

note that many \( \left[ K_i^{IB} \right]_c \) are null. Next we can complete \( K_i^{BB} \) with,

\[
\left[ \bar{K}_i^{BB} \right]_c = K_i^{IB} t.
\]
Now let’s define \( \bar{\mathbf{f}}_i^B = \mathbf{K}_i^{BI} \left( \mathbf{K}_i^{II} \right)^{-1} \mathbf{f}_i^I \), in this case only one system has to be solved
\[
\mathbf{K}_i^{II} \mathbf{t} = \mathbf{f}_i^I,
\]
and then
\[
\bar{\mathbf{f}}_i^B = \mathbf{K}_i^{BI} \mathbf{t}.
\]

Each \( \mathbf{K}_i^{BB} \) and \( \bar{\mathbf{f}}_i^B \) holds the contribution of each partition to (4), this can be written as
\[
\left( \mathbf{K}_i^{BB} - \sum_{i=1}^{p} \mathbf{K}_i^{BB} \right) \mathbf{d}^B = \mathbf{f}^B - \sum_{i=1}^{p} \bar{\mathbf{f}}_i^B, \tag{7}
\]
once (7) is solved, we can calculate the inner results of each partition using (3).

Since \( \mathbf{K}_i^{II} \) is sparse and has to be solved many times in (5), a efficient way to proceed is to use a Cholesky factorization of \( \mathbf{K}_i^{II} \). To reduce memory usage and increase speed a sparse Cholesky factorization has to be implemented, this method is explained below.

In case of (7), \( \mathbf{K}_i^{BB} \) is sparse, but \( \bar{\mathbf{K}}_i^{BB} \) are not. To solve this system of equations an sparse version of conjugate gradient was implemented, the matrix \( \left( \mathbf{K}_i^{BB} - \sum_{i=1}^{p} \bar{\mathbf{K}}_i^{BB} \right) \) is not assembled, but maintained distributed.
Matrix storage

An efficient method to store and operate matrices of this kind of problems is the Compressed Row Storage (CRS) [Saad03 p362]. This method is suitable when we want to access entries of each row of a matrix $A$ sequentially.

For each row $i$ of $A$ we will have two vectors, a vector $v_i^A$ that will contain the non-zero values of the row, and a vector $j_i^A$ with their respective column indexes. For example a matrix $A$ and its CRS representation

$$A = \begin{pmatrix}
8 & 4 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 3 & 0 & 0 \\
2 & 0 & 1 & 0 & 7 & 0 \\
0 & 9 & 3 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 5
\end{pmatrix},$$

The size of the row will be denoted by $|v_i^A|$ or by $|j_i^A|$. Therefore the $q$th non zero value of the row $i$ of $A$ will be denoted by $(v_i^A)_q$ and the index of this value as $(j_i^A)_q$, with $q=1,\ldots,|v_i^A|$. 

$$v_4^A = (9, 3, 1)$$
$$j_4^A = (2, 3, 6)$$
Cholesky factorization for sparse matrices

For full matrices the computational complexity of Cholesky factorization $A = LL^T$ is $O(n^3)$.

To calculate entries of $L$

$$L_{ij} = \frac{1}{L_{jj}} \left( A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right), \text{ for } i > j$$

$$L_{jj} = \sqrt{A_{jj} - \sum_{k=1}^{j-1} L^2_{jk}}.$$

We use four strategies to reduce time and memory usage when performing this factorization on sparse matrices:

1. Reordering of rows and columns of the matrix to reduce fill-in in $L$. This is equivalent to use a permutation matrix to reorder the system $(P A P^T)(P x) = (P b)$.
2. Use symbolic Cholesky factorization to obtain an exact $L$ factor (non zero entries in $L$).
3. Organize operations to improve cache usage.
4. Parallelize the factorization.
Matrix reordering

We want to reorder rows and columns of $A$, in a way that the number of non-zero entries of $L$ are reduced. $\eta(L)$ indicates the number of non-zero entries of $L$.

The stiffness matrix to the left $A \in \mathbb{R}^{556 \times 556}$, with $\eta(A) = 1810$. To the right the lower triangular matrix $L$, with $\eta(L) = 8729$.

There are several heuristics like the minimum degree algorithm [Geor81] or a nested dissection method [Kary99].
By reordering we have a matrix $A'$ with $\eta(A') = 1810$ and its factorization $L'$ with $\eta(L') = 3215$. Both factorizations solve the same system of equations.

We reduce the factorization fill-in by

\[
\frac{\eta(L')}{\eta(L)} = \frac{3215}{8729} = 0.368.
\]

To determine a “good” reordering for a matrix $A$ that minimize the fill-in of $L$ is an NP complete problem [Yann81].
Symbolic Cholesky factorization

The algorithm to determine the $L_{ij}$ entries that are non-zero is called symbolic Cholesky factorization [Gall90].

Let be, for all columns $j = 1 \ldots n$,

$$a_j \overset{\text{def}}{=} \{ k > j \mid A_{kj} \neq 0 \},$$

$$l_j \overset{\text{def}}{=} \{ k > j \mid L_{kj} \neq 0 \}.$$  

The sets $r_j$ will register the columns of $L$ which structure will affect the column $j$ of $L$.

$r_j \leftarrow \emptyset$, $j \leftarrow 1 \ldots n$

for $j \leftarrow 1 \ldots n$

$l_j \leftarrow a_j$

for $i \in r_j$

$$l_j \leftarrow l_j \cup l_i \setminus \{ j \}$$

end_for

$p \leftarrow \min\{ i \in l_j \}$ if $l_j \neq \emptyset$

$j$ otherwise

$r_p \leftarrow r_p \cup \{ j \}$

end_for

This algorithm is very efficient, its complexity in time and space has an order of $O(\eta(L))$. 

\[
A = \begin{bmatrix}
  a_{11} & a_{12} & & & a_{16} \\
  a_{21} & a_{22} & a_{23} & a_{24} & \\
  & a_{32} & a_{33} & a_{34} & a_{35} \\
  a_{42} & & a_{44} & & a_{46} \\
  a_{52} & a_{53} & & a_{55} & a_{56} \\
  a_{61} & a_{63} & a_{65} & a_{66} & \\
  a_2 & [3, 4] & & & \\
\end{bmatrix}
\]

\[
L = \begin{bmatrix}
  l_{11} & l_{12} & & & l_{16} \\
  l_{21} & l_{22} & l_{32} & l_{33} & l_{35} \\
  l_{42} & l_{43} & l_{44} & l_{53} & l_{54} \\
  l_{52} & & l_{55} & l_{63} & l_{65} \\
  l_{61} & l_{62} & l_{63} & l_{64} & l_{66} \\
  l_2 & [3, 4, 6] & & & \\
\end{bmatrix}
\]
Parallelization of the factorization

The calculation of the non-zero $L_{ij}$ entries can be done in parallel if we fill $L$ column by column [Heat91]. Let $J(i)$ be the indexes of the non-zero values of the row $i$ of $L$. Formulae to calculate $L_{ij}$ are:

\[
L_{ij} = \frac{1}{L_{jj}} \left( A_{ij} - \sum_{k \in J(i) \cap J(j), k < j} L_{ik} L_{jk} \right), \quad \text{para } i > j
\]

\[
L_{jj} = \sqrt{A_{jj} - \sum_{k \in J(j), k < j} L_{jk}^2}.
\]

The parallelization was made using the OpenMP schema.
How efficient is it?

The next table shows results solving a 2D Poisson equation problem, comparing Cholesky and conjugate gradient with Jacobi preconditioning. Several discretizations where used.

<table>
<thead>
<tr>
<th>Equations</th>
<th>nnz(A)</th>
<th>nnz(L)</th>
<th>Cholesky [s]</th>
<th>CGJ [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,006</td>
<td>6,140</td>
<td>14,722</td>
<td>0.086</td>
<td>0.081</td>
</tr>
<tr>
<td>3,110</td>
<td>20,112</td>
<td>62,363</td>
<td>0.137</td>
<td>0.103</td>
</tr>
<tr>
<td>10,014</td>
<td>67,052</td>
<td>265,566</td>
<td>0.309</td>
<td>0.184</td>
</tr>
<tr>
<td>31,615</td>
<td>215,807</td>
<td>1’059,714</td>
<td>1.008</td>
<td>0.454</td>
</tr>
<tr>
<td>102,233</td>
<td>705,689</td>
<td>4’162,084</td>
<td>3.810</td>
<td>2.891</td>
</tr>
<tr>
<td>312,248</td>
<td>2’168,286</td>
<td>14’697,188</td>
<td>15.819</td>
<td>19.165</td>
</tr>
<tr>
<td>909,540</td>
<td>6’336,942</td>
<td>48’748,327</td>
<td>69.353</td>
<td>89.660</td>
</tr>
<tr>
<td>3’105,275</td>
<td>21’681,667</td>
<td>188’982,798</td>
<td>409.365</td>
<td>543.110</td>
</tr>
<tr>
<td>10’757,887</td>
<td>75’202,303</td>
<td>743’643,820</td>
<td>2,780.734</td>
<td>3,386.609</td>
</tr>
</tbody>
</table>

The graphs show the relationship between the number of equations and the time and memory usage for both Cholesky and CGJ methods.
We used a cluster with 15 nodes, each one with two dual core Intel Xeon E5502 (1.87GHz) processors, a total of 60 cores.

The problem tested is a 3D solid model of a building that is deformed due to self weight. The geometry is divided in 1’336,832 elements, with 1’708,273 nodes, with three degrees of freedom per node the resulting system of equations has 5’124,819 unknowns. Tolerance used is 1x10^{-10}.
## Numerical experiment, building deformation

### Table of Results

<table>
<thead>
<tr>
<th>Number of processes</th>
<th>Partitioning time [s]</th>
<th>Inversion time (Cholesky) [s]</th>
<th>Schur complement time (CG) [s]</th>
<th>CG steps</th>
<th>Total time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>47.6</td>
<td>18520.8</td>
<td>4444.5</td>
<td>6927</td>
<td>23025.0</td>
</tr>
<tr>
<td>28</td>
<td>45.7</td>
<td>6269.5</td>
<td>2444.5</td>
<td>8119</td>
<td>8771.6</td>
</tr>
<tr>
<td>56</td>
<td>44.1</td>
<td>2257.1</td>
<td>2296.3</td>
<td>9627</td>
<td>4608.9</td>
</tr>
</tbody>
</table>

### Graphs

- **Time [s] vs. Number of processes**
  - Schur complement time (CG) [s]
  - Inversion time (Cholesky) [s]
  - Partitioning time [s]

- **Memory [GB] vs. Number of processes**
  - Slave processes [GB]

### Memory Consumption

<table>
<thead>
<tr>
<th>Number of processes</th>
<th>Master process [GB]</th>
<th>Slave processes [GB]</th>
<th>Total memory [GB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>1.89</td>
<td>73.00</td>
<td>74.89</td>
</tr>
<tr>
<td>28</td>
<td>1.43</td>
<td>67.88</td>
<td>69.32</td>
</tr>
<tr>
<td>56</td>
<td>1.43</td>
<td>62.97</td>
<td>64.41</td>
</tr>
</tbody>
</table>
Larger systems of equations

To test solution times in larger systems of equations we set a simple geometry. We calculated the temperature distribution of a metallic square with Dirichlet conditions on all boundaries.

The domain was discretized using quadrilaterals with nine nodes, the discretization made was from 25 million nodes up to 150 million nodes.

In all cases we divided the domain into 116 partitions, each partition is solved in one core.
### Larger systems of equations

<table>
<thead>
<tr>
<th>Equations</th>
<th>Partitioning time [min]</th>
<th>Inversion time (Cholesky) [min]</th>
<th>Schur complement time (CG) [min]</th>
<th>CG steps</th>
<th>Total time [min]</th>
</tr>
</thead>
<tbody>
<tr>
<td>25,010,001</td>
<td>6.2</td>
<td>17.3</td>
<td>4.7</td>
<td>872.0</td>
<td>29.4</td>
</tr>
<tr>
<td>50,027,329</td>
<td>13.3</td>
<td>43.7</td>
<td>6.3</td>
<td>1012.0</td>
<td>65.4</td>
</tr>
<tr>
<td>75,012,921</td>
<td>20.6</td>
<td>80.2</td>
<td>4.3</td>
<td>1136.0</td>
<td>108.3</td>
</tr>
<tr>
<td>100,020,001</td>
<td>28.5</td>
<td>115.1</td>
<td>5.4</td>
<td>1225.0</td>
<td>152.9</td>
</tr>
<tr>
<td>125,014,761</td>
<td>38.3</td>
<td>173.5</td>
<td>7.5</td>
<td>1329.0</td>
<td>224.2</td>
</tr>
<tr>
<td>150,038,001</td>
<td>49.3</td>
<td>224.1</td>
<td>8.9</td>
<td>1362.0</td>
<td>288.5</td>
</tr>
</tbody>
</table>
Larger systems of equations

### Memory (Giga bytes)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>25,010,001</td>
<td>4.05</td>
<td>0.41</td>
<td>47.74</td>
<td>51.79</td>
</tr>
<tr>
<td>50,027,329</td>
<td>8.10</td>
<td>0.87</td>
<td>101.21</td>
<td>109.31</td>
</tr>
<tr>
<td>75,012,921</td>
<td>12.15</td>
<td>1.37</td>
<td>158.54</td>
<td>170.68</td>
</tr>
<tr>
<td>100,020,001</td>
<td>16.20</td>
<td>1.88</td>
<td>217.51</td>
<td>233.71</td>
</tr>
<tr>
<td>125,014,761</td>
<td>20.25</td>
<td>2.38</td>
<td>276.04</td>
<td>296.29</td>
</tr>
<tr>
<td>150,038,001</td>
<td>24.30</td>
<td>2.92</td>
<td>338.29</td>
<td>362.60</td>
</tr>
</tbody>
</table>
Dynamic problem formulation

The Hilber-Hughes-Taylor (HHT) method [Hilb77] is used to solve a dynamic problem, equations of motion have the form

\[ M \ddot{u} + C \dot{u} + K u = F(t), \]

where \( M, C \) and \( K \) are the mass, damping and stiffness matrices of size \( n \times n \), these are constant, \( F \) is the time depending force.

The integration formulas depend on two parameters \( \beta \) and \( \gamma \),

\[ u_{i+1} = u_i + h \dot{u}_i + \frac{h^2}{2} \left[ (1 - 2 \beta) \ddot{u}_i + 2 \beta \ddot{u}_{i+1} \right], \]
\[ \dot{u}_{i+1} = \dot{u}_i + h \left[ (1 - \gamma) \ddot{u}_i + \gamma \ddot{u}_{i+1} \right], \]

they are used to discretize the equations of motion at time \( t_{i+1} \), \( h \) is the step size, \( M \ddot{u}_{i+1} + (1 + \alpha) C \dot{u}_{i+1} - \alpha C \dot{u}_i + (1 + \alpha) K u_{i+1} - \alpha K u_i = F(\tilde{t}_{i+1}), \)

where \( \tilde{t}_{i+1} = t_n + (1 + \alpha) h. \)

The HTT method is second order accurate and unconditionally stable with

\[ -1/3 \leq \alpha \leq 0, \quad \beta = (1 - \alpha)^2/4 \text{ and } \gamma = (1 - 2 \alpha)/2. \]
Infante Henrique bridge over the Douro river, Portugal
**Simulation**

Simulation of a 18 wheels 36 metric tons truck crossing the **Infante D. Henrique Bridge**.
Pre and post-process using GiD ([http://gid.cimne.upc.es](http://gid.cimne.upc.es)).

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>332,462</td>
</tr>
<tr>
<td>Elements</td>
<td>1,381,944</td>
</tr>
<tr>
<td>Element type</td>
<td>Tetrahedron</td>
</tr>
<tr>
<td>Time steps</td>
<td>372</td>
</tr>
<tr>
<td>HHT alpha factor</td>
<td>0</td>
</tr>
<tr>
<td>Rayleigh damping a</td>
<td>0.5</td>
</tr>
<tr>
<td>Rayleigh damping b</td>
<td>0.5</td>
</tr>
<tr>
<td>Degrees of freedom</td>
<td>997,386</td>
</tr>
<tr>
<td>nnz(K)</td>
<td>38,302,119</td>
</tr>
<tr>
<td>Partitioning time</td>
<td>32.9 s</td>
</tr>
<tr>
<td>Factorization time</td>
<td>87.3 s</td>
</tr>
<tr>
<td>Time per step (CGJ)</td>
<td>132.9 s</td>
</tr>
<tr>
<td>Total time</td>
<td>13.7 h</td>
</tr>
</tbody>
</table>
Thank you! Questions?

miguelvargas@cimat.mx
http://www.cimat.mx/~miguelvargas
References


