Parallel computation of inelastic problems with boundary element methods
S. S. Jiang* and Y. S. Jiang**
*LTAS Fracture Mechanics, University of Liège, Belgium
**J&W Services, Rue Lovinfosse 23, 4030 Grivegnée (Liège), Belgium

Abstract

The parallel performance and efficiency of boundary element methods in the solution of inelastic problems are investigated. A parallel algorithm based on the so-called initial stress scheme for inelastic analysis is developed and implemented in BEMGEO (Boundary Element Method program for GEOmechanics applications). Numerical examples solved on an SP2 of 34 thin nodes are presented which highlight the advantage of boundary element methods in high performance computing.

1. Introduction

Boundary element method (BEM) is generally considered to be more efficient than domain type methods such as finite element methods (FEM) in the solution of linear problems. This is because BEM reduces by one order the computation domain of problems. However, for nonlinear problems, BEM has to resort to domain type of discretization. The kernel integration in BEM is much more CPU time consuming than that in FEM and variables throughout the discretized domain are all coupled together (no banded matrix at all). BEM loses its efficiency and is much less used in the solution of nonlinear problems except in a few of special cases such as contact problems (geometrical nonlinear), infinite domains. Recent development of high performance computing has led to re-evaluate the most of usual algorithms in function of new criterions of viability and of performance. From the point of view of parallel computation, good methods are those consisting of the most independent calculation.

* Communication should be addressed to this author.
Roughly speaking, both FEM and BEM spend computing time essentially in two phases of calculation: the integration over elements and the solution of the system of equations. But most of time (up to 90%) that FEM spends is in the solution of the system of equations while BEM (up to 90%) in the integration over elements (also called cells in case of nonlinear analysis). It is well known that the solution of the system of equations is difficult to be parallelized and generally results in poor parallel performance for large number of parallel processors. On the contrary, BEM results in small system of equations and spends the most time in the integration over elements, which is perfectly parallelizable. One can therefore anticipate that parallel computation with BEM would lead to good performance and efficiency. This paper aims to demonstrate the intrinsic advantage of BEM in parallel computation of non-linear problems with BEM.

2. Formulation of Inelastic problems

In following what Tells and Brebbia described in [1], we give below a brief formulation of boundary element method for inelastic problems.

The basic assumption of inelastic analysis consists in splitting the strain tensor as well as the stress tensor into the elastic part and the plastic part:

\[ \varepsilon = \varepsilon^e + \varepsilon^p \]  
\[ \sigma = \sigma^e - \sigma^p \]  

Under this assumption, one can deduce boundary integral equations as demonstrated in Brebbia [1] for the displacement at a point inside the domain of problems to be solved (body forces are omitted):

\[ \dot{u}_i = \int \dot{u}_j \hat{p}_j d\Gamma - \int p_j \dot{u}_j d\Gamma + \int \varepsilon_{ij} \dot{\sigma}_{ij} \, d\Omega \]  

while the stresses are

\[ \dot{\sigma}_{ij} = \int \dot{u}_{ik} \hat{p}_k d\Gamma - \int p_{ik} \dot{u}_k d\Gamma + \int \varepsilon_{ijk} \dot{\sigma}_{kl} \, d\Omega \]  

Application of equation (3) along boundaries leads to the following equations in matrix form:

\[ H \ddot{u} = G \dot{p} + D \dot{\sigma}^p \]
Denoting $\mathbf{x}$ the vector of boundary unknowns, the above equation may be written as:

$$A \mathbf{x} = \mathbf{F} + D \mathbf{\dot{\sigma}}^p \quad (6)$$

Solving (6) for $\mathbf{x}$ by premultiplying $A^{-1}$, the inverse of $A$, one obtains:

$$\mathbf{x} = \mathbf{N} + Q \mathbf{\dot{\sigma}}^p \quad (7)$$

where $\mathbf{N} = A^{-1} \mathbf{F}$ and $Q = A^{-1}D$.

Equation (4) may be rewritten in the following matrix form:

$$\mathbf{\dot{\sigma}} = \mathbf{G}\mathbf{\dot{p}} - \mathbf{H}\mathbf{\dot{u}} + \mathbf{D}\mathbf{\dot{\sigma}}^p \quad (8)$$

which may be in turn rearranged for known and unknown boundary variables:

$$\mathbf{\dot{\sigma}} = \mathbf{A}'\mathbf{\dot{\mathbf{x}}} + \mathbf{\dot{F}}' + \mathbf{D}'\mathbf{\dot{\sigma}}^p \quad (9)$$

Substitution of $\mathbf{x}$ with equation (7) leads to

$$\mathbf{\dot{\sigma}} = \mathbf{\dot{M}} + Q'\mathbf{\dot{\sigma}}^p \quad (10)$$

in which

$$\mathbf{\dot{M}} = \mathbf{A}'\mathbf{\dot{N}} + \mathbf{\dot{F}}' \quad (11)$$

and

$$Q' = \mathbf{A}'\mathbf{Q} + \mathbf{D}' \quad (12)$$

Taking into account equation (2), we have:

$$\mathbf{\dot{\sigma}}^e = \mathbf{\dot{M}} + Q^*\mathbf{\dot{\sigma}}^p \quad (13)$$

where $Q^* = Q' + I$.

Equation (13) gives an iterative relation for calculating increments of elastic stresses for given plastic stress increments. Once $\mathbf{\dot{\sigma}}^e$ are known, the true stress increments may be calculated by [1]:

$$\mathbf{\dot{\sigma}}_{ij} = \mathbf{\dot{\sigma}}^e_{ij} - \frac{C_{ijmn}a_{mn}a_{kl}\mathbf{\dot{\sigma}}^e_{kl}}{a_{ij}C_{ijkl}a_{kl} + H'} \quad (14)$$

in which:

$C_{ijmn}$ : the tensor of coefficients in Hook’s law;
Boundary Elements

\( \alpha_{ij} \): the vector of derivatives of the yield criterion;
\( H' \): slope of the uniaxial stress-plastic strain curve.

Based on equations (13) and (14), an iterative solution scheme may be summarized in the following steps:

(i) calculate elastic stress increments \( \dot{\sigma}^e \) by using equation (13). The vector of load increment \( M \) is applied only when the iterative calculation for the precedent load increment has converged.

(ii) calculate true stress increments \( \dot{\sigma} \) by using equation (14)

(iii) calculate plastic stress increments by

\[
\dot{\sigma}^p = \dot{\sigma}^e - \dot{\sigma}
\]  

(iv) accumulate stress increments into total stresses

(v) verify convergence in comparing \( \dot{\sigma}^p \) with a predefined norm and go to step (i) as long as the total load has not been applied or the maximum number of iterations has not been exceeded.

4. Analysis of computational cost

In the above initial stress solution scheme, most of CPU time are spent in the following three parts of calculation:

(i) calculation of matrices (\( H, G, D, D', M, Q^* \))

(ii) inverse the matrice \( A \)

(iii) iterative scheme

In order to have an idea about the distribution of CPU time, let us consider an example of 2D circular tunnel in an infinite plane subjected to a field of residual stresses (\( \sigma_y = -1.0 \text{ t/m}^2, \sigma_x = 0.4\sigma_y \)) as depicted in figure 1.a. The tunnel diameter is equal to 4 m and material parameters are given in table 1. Drucker-Prager model is used with assumption of perfect plasticity (\( H' = 0.0 \)).

| Table 1. Material parameters of deep tunnel problem |
|-----------------------|-------|-----------------|-------|--------|-------|
| \( E \ (\text{t/m}^3) \) | \( \mu \) | \( \sigma \ (\text{t/m}^2) \) | \( C \ (\text{t/m}^3) \) | \( \phi \) | \( H' \) |
| 5x10^5 | 0.2 | -1.0 | 0.28 | 30° | 0.0 |

\( E \): elasticity modular, \( \mu \): Poisson’s ratio, \( C \): cohesion, \( \phi \): friction angle.
This problem was solved by Venturini [2] and Jiang [3] using a coarse mesh of 9x5 cells as depicted in figure 1.a. The distribution of CPU time in percentage is shown in figure 2.

It can be seen that the calculation of matrices and the iterative scheme take more than 99% of CPU time while the inverse of the matrices A consumes less than 1% of CPU time. It is thus not important to parallelize the calculation of the inverse of A which would involve quite complicate parallel programming. It is justifiable to concentrate one’s attention on parallelizing the calculation of matrices and of the iterative scheme. This forms the basic assumption of the parallel algorithm presented in the following section.

### 5. Parallel algorithm and implementation

The parallel algorithm developed in this work is based on the partition of matrices in function of number of processors. The interference between partitions of matrices is managed with a message passing library, PARMACS [4]. Parallel computers of distributed memory are aimed platforms in which the SP2 of IBM and cluster of workstations are typical examples.
588 Boundary Elements

paradigm supported by PARMACS may be classified as SPMD (Single Program Multiple Data) type. There are one host process and a number of slave processes created by the former. Slave processes generally perform computational tasks while the host process serves mainly as a controller of the whole procedure of calculation.

Without loss of generality, let us consider the case of one host and two slave processes. The matrices $H$, $G$ and $D$ in equation (5) are partitioned as shown in the following equation:

$$
\begin{bmatrix}
H_1 \\
H_2
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2
\end{bmatrix} =
\begin{bmatrix}
G_1 \\
G_2
\end{bmatrix}
\begin{bmatrix}
u_2 \\
u_3
\end{bmatrix} +
\begin{bmatrix}
D_1 \\
D_2
\end{bmatrix}
\begin{bmatrix}
\sigma^p_1 \\
\sigma^p_2
\end{bmatrix}
$$

in which indices 1 and 2 denote slave process numbers. Matrices are quoted inside square brackets while vectors in braces. The line in square brackets (or braces) represents a limit separating two partitions.

Introduction of boundary conditions into the above equation leads to a partitioned system of equation:

$$
\begin{bmatrix}
A_1 \\
A_2
\end{bmatrix}
\begin{bmatrix}
\hat{\chi}_1 \\
\hat{\chi}_2
\end{bmatrix} =
\begin{bmatrix}
\hat{F}_1 \\
\hat{F}_2
\end{bmatrix} +
\begin{bmatrix}
D_1 \\
D_2
\end{bmatrix}
\begin{bmatrix}
\sigma^p_1 \\
\sigma^p_2
\end{bmatrix}
$$

At this moment, the slave process 1 holds in its memory the partition $A_1$ and the slave process 2 holds $A_2$. To go ahead in our calculation, we need to calculate the inverse of the global matrix $A (= A_1 + A_2)$. This can not be calculated separately. As has been stated in the precedent section, the calculation of the inverse of $A$ takes less than 1% of CPU time. It is performed by the host process in sequential calculation. To do this, the host process receives the partitions $A_1$ and $A_2$ from slave processes and assemble them into a global matrix $A$. After calculating the inverse $A^{-1}$, the host process sends the horizontal partitions

$$
A^{-1} = 
\begin{bmatrix}
A_1^{-1} & A_2^{-1}
\end{bmatrix}
$$

(18)
to slave processes. The latters calculate then the unknown boundary values by

$$
\begin{bmatrix}
\hat{\chi}_1 \\
\hat{\chi}_2
\end{bmatrix} =
\begin{bmatrix}
A_1^{-1} & A_2^{-1}
\end{bmatrix}
\begin{bmatrix}
\hat{F}_1 \\
\hat{F}_2
\end{bmatrix} +
\begin{bmatrix}
A_1^{-1} & A_2^{-1}
\end{bmatrix}
\begin{bmatrix}
D_1 \\
D_2
\end{bmatrix}
\begin{bmatrix}
\sigma^p_1 \\
\sigma^p_2
\end{bmatrix}
$$
One can see that the unknown boundary values $\mathbf{x}$ are obtained by summing up contributions from each slave process. To calculate stresses, equation (8) has been partitioned in the same way:

$$\begin{bmatrix} \dot{\mathbf{\sigma}}_1 \\ \dot{\mathbf{\sigma}}_2 \end{bmatrix} = \begin{bmatrix} G'_1 \\ G'_2 \end{bmatrix} \{\mathbf{p}\} + \begin{bmatrix} H'_1 \\ H'_2 \end{bmatrix} \{\mathbf{u}\} + \begin{bmatrix} D'_1 \\ D'_2 \end{bmatrix} \{\mathbf{\dot{\sigma}}^p\}$$

After rearranging the known and unknown boundary terms, one has

$$\begin{bmatrix} \dot{\mathbf{\sigma}}_1 \\ \dot{\mathbf{\sigma}}_2 \end{bmatrix} = \begin{bmatrix} A'_1 \\ A'_2 \end{bmatrix} \{\mathbf{\dot{x}}\} + \begin{bmatrix} \dot{\mathbf{F}}'_1 \\ \dot{\mathbf{F}}'_2 \end{bmatrix} + \begin{bmatrix} D'_1 \\ D'_2 \end{bmatrix} \{\mathbf{\dot{\sigma}}^p\} \quad (21)$$

Substitution of $\mathbf{x}$ with equation (19) into (21) leads to

$$\begin{bmatrix} \dot{\mathbf{\sigma}}_1 \\ \dot{\mathbf{\sigma}}_2 \end{bmatrix} = \begin{bmatrix} A'_1 \end{bmatrix} \{\mathbf{\dot{N}}_1 + \mathbf{\dot{N}}_2\} + [\mathbf{Q}_1 + \mathbf{Q}_2] \{\mathbf{\dot{\sigma}}^p\} + \begin{bmatrix} \dot{\mathbf{F}}'_1 \\ \dot{\mathbf{F}}'_2 \end{bmatrix} + \begin{bmatrix} D'_1 \\ D'_2 \end{bmatrix} \{\mathbf{\dot{\sigma}}^p\}$$

$$\begin{bmatrix} \dot{\mathbf{M}}_1 \\ \dot{\mathbf{M}}_2 \end{bmatrix} = \begin{bmatrix} A'_1 \end{bmatrix} \begin{bmatrix} \mathbf{Q}_1 + \mathbf{Q}_2 \\ + D'_1 \end{bmatrix} \{\mathbf{\dot{\sigma}}^p\}$$

$$\begin{bmatrix} \dot{\mathbf{Q}}_1 \\ \dot{\mathbf{Q}}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{M}_1 \\ \mathbf{M}_2 \end{bmatrix} \{\mathbf{\dot{\sigma}}^p\} \quad (22)$$

One can see that the calculation of $\mathbf{Q}'_1$ on the slave process 1 needs the submatrice $\mathbf{Q}_2$ that exists only on the slave process 2. For a configuration of $n$ slave processes, this would involve at least $(n-1)\log_2(n-1)$ transfers of submatrices $\mathbf{Q}_i, i = 1, n-1$. To avoid such awkward transfer operations, it has been managed in our implementation that all slave processes send their $\mathbf{Q}_i$ to the host process which assembles them into a global matrix $\mathbf{Q}$. Then the host process sends $\mathbf{Q}$ back to all slave processes. It worth it to note that these matrix transmissions happen only once time in the whole calculation.
Substituting the above equation into equation (13), we obtain the final partitioned equation for the iterative scheme:

\[
\begin{bmatrix}
\hat{\sigma}_1^* \\
\hat{\sigma}_2^*
\end{bmatrix} = 
\begin{bmatrix}
\hat{M}_1 \\
\hat{M}_2
\end{bmatrix} + 
\begin{bmatrix}
\hat{Q}_1^* \\
\hat{Q}_2^*
\end{bmatrix} \hat{\sigma}^p
\]

Based on this equation, the sequence of parallel implementation of the iterative scheme may be described as the followings:

Slave process \(i\) (\(i = 1,2\)):
- Receive the global vector of plastic stress increments \(\hat{\sigma}^p\),
- Calculate its subvector of elastic stress increments by
  \[
  \{\hat{\sigma}^e_i\} = \{\hat{M}_i\} + \{\hat{Q}_i^*\} \hat{\sigma}^p
  \]
- Calculate the true stress increments by using equation (14),
- Calculate the plastic stress increments with equation (15),
- Verify the local convergence and inform the host process on the local convergence,
- Receive the information on the global convergence from the host process. If the global convergence has been reached (i.e., the local convergence has been verified on all slave processes), send its subvector of true stress increments to the host process and begin a new iteration.
- Send its subvector of plastic stress increments to the host process and begin a new iteration.

The host process:
- Send the total vector of plastic stress increments to slave processes,
- Receive the local convergence information from each process. The global convergence is considered to be reached if all processes reach the convergence on their subvector of plastic stresses. In such case, receive true stress increments from slave processes and print results of the current load increment.
- Receive subvector of plastic stresses from each slave process and assemble it into a total vector of plastic stress increments which is then sent back to all slave processes.

6. Benchmarking

The parallel algorithm described in the above section has been implemented in an existent BEM program, called BEMGEO developed initially in [3]. The parallel version, called PARBEM, uses PARMACS as the message passing interface which is supported on most parallel computers. PARBEM has been
ported on an SP2 that is composed of 34 thin nodes. Each node has the following configuration:

<table>
<thead>
<tr>
<th>Processor type</th>
<th>Core memory</th>
<th>Swap space</th>
<th>Local disk</th>
<th>Communication Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power2,66 MHz</td>
<td>128 MB</td>
<td>512 MB</td>
<td>1 GB</td>
<td>80 MB/s</td>
</tr>
</tbody>
</table>

Table 2. Configuration of one node of SP2

In order to make the test problem large enough to measure the meaningful performance in parallel computing, the mesh in figure 1.a has been refined to be of 25x16 cells as shown in figure 1.b. With a single node of SP2, the problem defined on the refined mesh led to 2602 seconds of solution time. Parallel computing time (elapsed time) in function of number of processors is given in the following figure.

![Elapsed time against the number of processors](image)

Figure 3. Elapsed time against the number of processors

The elapsed time decreases monotonously from 2602 seconds with a single processor to 130 seconds with 33 processors. Based on this result, the speedup has been calculated and given in figure 4.

![Speedup of parallel computing against the number of processors](image)

Figure 4. Speedup of parallel computing against the number of processors

It can be observed that up to 15 processors, the speedup increases almost linearly as in the ideal case. However the rate of speedup increase is slow down when the number of processors is larger than 15. This is because when increasing the number of processors, matrix partitions on each processor become smaller and smaller and the communication overheat in the iterative
scheme degrades more and more the global performance. Nevertheless, the speedup is kept in increasing up to 33 processors.

7. Conclusions

We have shown that more than 99% of computation time in the solution of inelastic problems with BEM are spent in the formation of matrices and in the iterative scheme (the initial stress solution techniques). These two parts of calculation may be easily parallized. Only less than 1% of CPU time is used in calculation of the inverse of the system of equations of boundary unknowns. This part of calculation may be kept sequential and has not significant impact to the performance of parallel computing. Parallel programming is greatly simplified and the resulted parallel implementation is robust and leads to very good performance of parallel computing. By using 33 processors, we have solved a deep tunnel in an infinite plane of inelastic material and reduced the elapsed time to only 130 seconds with a speedup as high as 20. The advantage of BEM in the solution of inelastic problems has been highlighted through this work. The resort to domain discretization, which has been widely considered as a drawback of BEM for non-linear problems, may become a definitive advantage in massively parallel computing

References


Acknowledgement

The authors would like to express their gratitude to POLYFLOW, s.a., for its offer of parallel computing facilities.