A Parallel Direct Solver for Implicit Finite Element Problems Based on Automatic Domain Decomposition

O. Zone, D. Vanderstraeten, P. Henriksen, and R. Keunings

Centre for Systems Engineering and Applied Mechanics, Université Catholique de Louvain, B-1348 Louvain-la-Neuve, Belgium

Keywords: Direct finite element solver, domain decomposition, partitioning heuristics.

1. INTRODUCTION

In the present paper, we review our on-going efforts towards the development of a direct, parallel solver for general finite element equation sets, and the associated automatic mesh partitioning tools. Target applications for the proposed methodology are implicit finite element problems solved in complex geometries using unstructured finite element meshes. A typical example considered in this paper is the flow of viscoelastic fluids, whose governing equations are vastly more complex than the classical Navier-Stokes equations [1].

The solver is based on domain decomposition. It uses the frontal method in each subdomain, and a divide-and-conquer parallel treatment of the interface problem. The basic algorithm has been proposed in [2], where performance modeling and preliminary results are reported for relatively simple problems (heat diffusion and incompressible Navier-Stokes flow) and topologically-linear decompositions. The parallel solver has been implemented on a variety of MIMD distributed memory computers, including the Intel iPSC/860 hypercube, the Convex Meta Series, and a heterogeneous network of engineering workstations.

The proposed mesh partitioning strategy involve two steps: the use of a direct partitioning scheme to provide an initial partition, followed by an optimization step based on graph theory and using non-deterministic combinatorial heuristics. An adaptive cost function is introduced such as to equalize the frontal compute load in each subdomain and to minimize the size of the interface problem. The basic algorithms are reported in [3], together with preliminary results on both 2D and 3D unstructured meshes.

In this paper, the parallel frontal algorithm is extended to topologically-general decompositions. Moreover, the partitioning heuristics are extended such as to minimize the subdomain compute load by means of a new element renumbering scheme proposed in [4]. Finally, we report new results obtained in the context of large-scale parallel simulations of viscoelastic flows in complex geometries.
2. PARALLEL DIRECT SOLVER

In the context of implicit finite element problems, we consider the solution of a large linear set of algebraic equations of the form

\[ Ku = f, \]

where the matrix \( K \) is sparse, non-symmetric and indefinite. The main features of the proposed direct parallel solver [2,5] are described below. Let \( P \) be the number of available processors. First, the finite element mesh is decomposed in \( P \) subdomains. The set of equations can be written as

\[
\begin{pmatrix}
  K_{11} & \cdots & 0 & K_{1P} \\
  \vdots & \ddots & \vdots & \vdots \\
  0 & \cdots & K_{PP} & K_{PI} \\
  K_{I1} & \cdots & K_{IP} & K_{II}
\end{pmatrix}
\begin{pmatrix}
u_1 \\
u_P \\
u_I
\end{pmatrix}
= \begin{pmatrix}
f_1 \\
f_P \\
f_I
\end{pmatrix},
\]

where \( u_i \) (\( i = 1, \ldots, P \)) is the vector of variables internal to subdomain \( i \) and \( u_I \) represents the vector of interface variables. At the beginning of the calculations, each processor is allocated the elements of a subdomain. Processor \( i \) then performs the sequential frontal method on the system

\[
\begin{pmatrix}
  K_{ii} & K_{iI} \\
  K_{Ii} & K_{II}^{(i)}
\end{pmatrix}
\begin{pmatrix}
u_i \\
u_I^{(i)}
\end{pmatrix}
= \begin{pmatrix}
f_i \\
f_I^{(i)}
\end{pmatrix},
\]

where \( K_{II}^{(i)} \) and \( f_I^{(i)} \) are the local contribution of the subdomain \( i \) to \( K_{II} \) and \( f_I \), and \( u_I^{(i)} \) is the restriction of the interface variables \( u_I \) to subdomain \( i \). At the end of this step, the internal variables \( u_i \) have been eliminated as a function of the interface variables \( u_I^{(i)} \), and each processor holds an active interface system of the form

\[ S^{(i)} u_I^{(i)} = f_I^{(i)} - K_{II} K_{ii}^{-1} f_i. \]

The matrix \( S^{(i)} \) is the contribution of subdomain \( i \) to the Schur complement:

\[ S^{(i)} = K_{II}^{(i)} - K_{II} K_{ii}^{-1} K_{II}. \]

A communication phase is then needed to assemble the interface contributions of every subdomain \( i \) and to solve the interface problem. If we assume that \( P \) is an integral power of 2, we can view this phase as the traversal of a binary tree where the arcs represent a communication at each level, and the leaves correspond to the different processors (Fig. 1). The parallel processing of the interface problem involves two traversals of the tree. First, the tree is traversed from the leaves to the root. At each level, half of the active processors send in parallel their interface system to a neighbor processor, after which they become idle; the receiving processors assemble in parallel the neighbor contributions with their own, and eliminate the common interface variables by means of Gaussian elimination. This procedure is repeated until the “root” processor is reached. The second traversal goes from the root to the leaves. It consists of a backsubstitution followed by
the communication of the computed values of the interface variables. A processor having performed the elimination of an interface variable during the first traversal computes this variable by backsubstitution.

When the interface problem has been solved, the processors perform in parallel the backsubstitution of their internal variables.

![Diagram of T-shaped mesh and corresponding interface processing tree](image)

Figure 1. (a) Decomposition of a T-shaped mesh in 4 subdomains, and its interface graph (dashed lines); (b) Corresponding interface processing tree; an arrow represents a communication during the first traversal step.

The way the interface system is communicated, assembled, and solved is a key issue as far as parallel efficiency is concerned. In order to limit the loss of efficiency, we compute an interface processing tree in such a way that the maximum number of interface variables is eliminated at the lowest possible level of the tree.

Let $G(V, E)$ be the weighed interface graph of the decomposition, where the vertices of $V$ correspond to the subdomains, and the edges of $E$ represent the interfaces between subdomains. We set the weight of an edge equal to the number of variables of the corresponding interface. Fig. 1a shows a decomposition and its interface graph. In order to define the first level of the interface processing tree, we determine a perfect matching of $G$ (i.e. a subset of $E$ where no two edges are incident to the same vertex and where each vertex has an incident edge). Every edge of the matching will define a communication phase between two processors, the receiver processor being the one with the largest interface size. The cardinality of the matching equals $P/2$. Moreover, we select the matching that maximizes the sum of its edge weights, thus maximizing the number of variables to be eliminated at this level. In the example of Fig. 1a, the maximum perfect matching consists of edges (1, 2) and (3, 4). To proceed to the next level of the interface processing tree, we repeat the above procedure with a new graph $G'(V', E')$ obtained by collapsing the pair of communicating vertices at the previous level, and by summing their edge weights. Perfect matchings are found by exhaustive enumeration.

3. AUTOMATIC DOMAIN DECOMPOSITION

The efficiency of the above parallel direct solver depends on the quality of the decomposition of the finite element meshes. A good decomposition should ensure an equal and minimum computational load within the subdomains, and it should have an interface size as small as possible.
Our mesh partitioning technique involves two steps: the use of a *direct partitioning* scheme to provide an initial partition [6–8], followed by an *optimization step* based on graph theory and using combinatorial non-deterministic heuristics [3,9,10]. The optimization techniques are based on the following principle: elements located along the interface are randomly selected and transferred to a neighboring subdomain with some probability. Transfers that increase the quality of the decomposition are always accepted, while transfers that decrease it are accepted with a probability inversely proportional to the loss of quality. The quality of the decomposition is estimated by a cost function that takes into account the total interface size and the load imbalance:

\[ \text{Cost} = \alpha I + (1 - \alpha) \sum_{k=1}^{P} (N_k - N_{best,k})^2, \]

where \( I \) is the interface size, \( N_k \) (resp. \( N_{best,k} \)) is the number (resp. the optimal number) of elements in subdomain \( k \). Since the parallel solver uses a frontal method in each subdomain, we model the load in subdomain \( k \) by

\[ \text{Load}_k = N_{best,k} \times \text{Front}_k^{\text{max}} \times \text{Front}_k^{\text{mean}}, \]

where \( \text{Front}_k^{\text{max}} \) and \( \text{Front}_k^{\text{mean}} \) represent the maximum and the mean frontal widths, respectively. The value of \( N_{best,k} \) is computed such that \( \text{Load}_k \) is a constant independent of \( k \). Since the frontal widths vary everytime an element is moved from one subdomain to another, the \( N_{best,k} \)’s cannot be computed a priori. To avoid the need of recomputing the frontal widths too often, we use a decoupled approach. First, we perform the optimization with frozen values of the \( N_{best,k} \)’s. Given this optimized decomposition, new values of the \( N_{best,k} \)’s are evaluated. If the variation of \( N_{best,k} \) before and after the optimization is small (typically less than 5%), we end the optimization procedure. If not, we perform a new optimization with updated values of \( N_{best,k} \)’s.

In addition to reducing the interface and to equalizing the load, we try to minimize the maximum load. This is done by renumbering the elements within the subdomains in order to decrease the frontal widths [4]. The renumbering step takes place everytime the frontal widths are evaluated.

4. RESULTS

Decomposition results have been obtained for a variety of 2D and 3D meshes with a number of subdomains ranging between 2 and 64 [3,10]. We present below results for the typical finite element mesh of 1217 elements and 1307 vertices shown in Fig. 2a. We also discuss the efficiency of the parallel solver for a viscoelastic flow problem run on a Convex Meta Series with 8 processors.

4.1. Decomposition

In Fig. 2b, we show a decomposition in 8 subdomains generated in the first step of the automatic partitioning method. We have used the Recursive Graph Bisection technique (RGB). Fig. 2c shows the decomposition obtained by the second step, wherein the initial partition is optimized by means of a Simulated Annealing scheme (SA). We can observe that the number of interface nodes has been reduced by 25%. Moreover, the maximum
load $Load_k$ over the subdomains has been reduced by a factor 2. We also observe that the interface is much smoother. This can be of crucial importance to finite element techniques that use a Lagrange multiplier treatment of the interface conditions.

Figure 2. Typical example. The original mesh (a) is decomposed in 8 subdomains with a direct RGB scheme (b) and then optimized by means of a SA scheme (c).

Table 1 gives the number of interface nodes and the reduction afforded by an optimization step of SA, compared to a direct Recursive Graph Bisection Scheme (RGB) for the mesh of Fig. 2a. In “SA + F”, only one renumbering phase is performed, at the end of the optimization, while in “SA + G”, we use the mixed optimization-renumbering scheme. The reduction in the number of interface nodes ranges between 20% and 50%. Similar reductions are observed for the entire set of problems tested [3].

<table>
<thead>
<tr>
<th>P</th>
<th>Number of interface nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>RGB</td>
<td>40</td>
</tr>
<tr>
<td>SA + F</td>
<td>21</td>
</tr>
<tr>
<td>SA + G</td>
<td>26</td>
</tr>
</tbody>
</table>

Table 1. Number of interface nodes and percentage reduction between a direct algorithm (RGB) and an optimizer (SA) with final (F) or global (G) renumbering scheme for the mesh shown in Fig. 2a.
Table 2. Maximum frontal width and percentage difference compared to the mono-domain frontal width for different decomposition and renumbering techniques (mesh of Fig. 2a).

<table>
<thead>
<tr>
<th>P</th>
<th>Max. frontal width (380 for P=1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
</tr>
<tr>
<td>RGB + F</td>
<td>432</td>
</tr>
<tr>
<td>SA + F</td>
<td>365</td>
</tr>
<tr>
<td>SA + G</td>
<td>352</td>
</tr>
</tbody>
</table>

Table 2 shows the maximum frontal widths over the subdomains obtained with the renumbering scheme. They are compared to the frontal width of 380 for the mono-domain mesh (26082 variables). The interface variables cannot be eliminated and increase therefore the frontal width. Though, we observe in this case that the maximum frontal width can be smaller than the mono-domain value for some optimized decompositions.

4.2. Parallel Solver

We consider the planar, steady-state flow of an Oldroyd-B differential fluid using the mixed finite element method MIX1 [1]. Tests have been carried on a Convex Meta Series with 8 processors. Table 3 gives the observed speedup relative to the mono-domain sequential run, for the decompositions of the mesh of Fig. 2 obtained with a direct method (RGB) and with an optimizer (SA + G). The increase in speedup for the optimizer compared to the direct method is caused by the decrease of the frontal width and of the interface size. Since we measure parallel speedup with respect to the sequential run on the mono-domain mesh, superlinear speedup can be obtained when the maximum frontal width within the subdomains is lower than that of the mono-domain mesh, and when a good load balance is achieved. When superlinear speedup happens, an improved sequential algorithm is readily available. Indeed, it suffices to run the parallel code on a single processor, which is possible in the PVM environment of the Convex Meta Series.

<table>
<thead>
<tr>
<th>P</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB + F</td>
<td>1.66</td>
<td>2.48</td>
<td>3.77</td>
</tr>
<tr>
<td>SA + G</td>
<td>2.63</td>
<td>3.76</td>
<td>11.81</td>
</tr>
</tbody>
</table>

Table 3. Parallel speedup relative to the mono-domain sequential time obtained for the mesh of Fig. 2a. The decompositions are computed with a direct algorithm (RGB + F) and an optimizer (SA + G).

In Fig. 3, the evolution of the sequential solution time is plotted for the best decomposition technique as a function of the number of subdomains. For a coarse mesh of 460 elements (10282 variables), the sequential time first decreases due to the decrease of the frontal width. Then, the interface problem becomes dominant and overcomes the reduction in frontwidth. For the mesh of Fig. 2, we observe that the minimum is not yet reached with 8 subdomains.
Figure 3. Evolution of sequential time for two different meshes as a function of the number of subdomains. The decompositions are computed with the optimizer (SA + G).

<table>
<thead>
<tr>
<th>P</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>SA + G</td>
<td>1.35</td>
<td>1.94</td>
<td>6.08</td>
</tr>
</tbody>
</table>

Table 4. “True” speedup relative to the best available sequential time for the mesh of Fig. 2a (obtained with 8 subdomains).

We can derive a more realistic measure of speedup by evaluating it relatively to the best sequential time available (with 8 subdomains in our example). Table 4 gives the “true” speedup thus obtained.

5. ACKNOWLEDGEMENTS

This work is supported by the European Commission (BRITE/EURAM programme), and the Belgian Services de Programmation de la Politique Scientifique (Interuniversity Attraction Poles programme). The Ph.D. work of Denis Vanderstraeten is supported by the Fonds National de la Recherche Scientifique of the Belgian State. Access to the Convex Meta Series has been granted by the UCL in the framework of the GPU Calcul Intensif project.

REFERENCES


