Data structures for the distributed iterative solution of non-conventional finite element models

Ildi Cismasăiu a,*, J.P. Moitinho de Almeida b

a Centro de Investigação em Estruturas e Construção – UNIC, Departamento de Engenharia Civil, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, Quinta da Torre, 2829-516 Monte de Caparica, Portugal
b Departamento de Engenharia Civil e Arquitectura, Instituto Superior Técnico, Universidade Técnica de Lisboa, Av. Rovisco Pais, 1049-001 Lisboa, Portugal

Received 16 November 2004; accepted 14 August 2006
Available online 11 January 2007

Abstract

A class of specialised data structures designed for the distributed solution of non-conventional finite element formulations, which are equally effective when used in conjunction with conventional formulations, is presented. We begin by briefly discussing how the non-conventional finite element formulations being developed within the structural analysis group at IST [Freitas JAT, Almeida JPM, Pereira EMBR. Non-conventional formulations for the finite element method. Comput Mech 1999;23(5–6):488–501] lead to systems of equations that appear to be naturally suited for parallel processing, but we also recognise that to take full advantage of the characteristics of these systems – large dimension, non-overlapping block structure and sparsity – it is necessary to use appropriate data structures. The approach presented, which references the logical subdivisions of the system matrices, was designed to fulfil these objectives. Examples of parallel performance and efficiency on an homogeneous distributed platform are presented.

2006 Published by Elsevier Ltd.

Keywords: Parallel processing; Matrix handling data structures; Hybrid finite elements; Domain decomposition

1. Introduction

In the context of the research carried out by the Structural Analysis Group at IST, to develop and test the behaviour of various non-conventional finite element models, an unified computational framework has been developed [2] to facilitate the implementation of the corresponding computer codes, offering an efficient and effective solution for the manipulation of the information associated with the problem in each phase of a FE algorithm. This framework allows to consider the specificity of each finite element model and includes parallel processing as an alternative to the sequential approach.

Two classes of data structures for distributed computation were proposed, whose goal is to take full advantage of

* Corresponding author. Fax: +351 21 2948398.
E-mail addresses: ildi@fct.unl.pt (I. Cismasăiu), moitinho@ist.utl.pt (J.P. Moitinho de Almeida).
2. Brief characterisation of the non-conventional finite element models

The non-conventional finite element models (NFEM), developed by the Structural Analysis Group at IST [1], are formulated using an hybrid approach where the boundary approximation functions may be independently defined on each side, reducing the restrictions that are normally imposed on the domain approximation functions as the concept of nodal approximation is no longer necessary. For hybrid-mixed models these restrictions are further reduced, allowing for instance to use discontinuous functions inside each element, e.g. Walsh functions [4]. These models have been implemented for the elastostatic, elastoplastic and elastodynamic analysis of planar and three dimensional problems. One important characteristic is that, depending of the model and under certain conditions, they can provide solutions that locally satisfy a set of field conditions, either compatibility or equilibrium. This allows to use a pair of solutions so obtained for error estimation, either global or local, and adaptivity [5].

The systems of equations associated to these formulations have a common structure, which is generically represented by Eq. (1), where \( X \) represents the set of variables used in the approximation of the solution in the domain and \( Y \) the variables used in the boundaries. Then \( X \) is a block matrix with at most as many blocks in each super-column, i.e. element, as the number of fields approximated in that element. \( Y \) is a block matrix, with at most two non-zero blocks per super-column, i.e. side, corresponding to the elements adjacent to that side.

\[
\begin{bmatrix}
X & Y \\
Y^T & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix} =
\begin{bmatrix}
x \\
y
\end{bmatrix},
\]

NFEM share a set of feature common characteristics, identified by Almeida et al. [6], which indicate that they are suitable for parallel processing:

- discretisations with a small number of elements of high degree;
- the computation of each element matrix has an high computational cost;
- the resulting systems of equations can be large, but are highly sparse;
- the associated variables in the FE system are strictly element dependent and are shared at most by two connecting finite elements.

From the viewpoint of parallel computation these non-conventional models can be considered as a particular case of the conventional formulation, offering new and more attractive perspectives in the development of parallel processing procedures.

The use by non-conventional formulations of variables that are strictly element dependent, shared by at most two connecting finite elements, contrasts with conventional formulations, where a node may be connected to a large number of elements. This implies that when a domain decomposition is used to partition the data for distributed computation, for conventional formulations the submatrices associated with shared nodes have at least two nonzero contributions to the diagonal of the system, while for NFEM the two submatrices associated with each shared edge do not overlap and are always off-diagonal.

3. Data structures for distributed matrix computations

The major computational cost in the analysis of a finite element model is the solution of the resulting system of linear equations. For parallel processing iterative solvers are computationally more attractive than the direct methods, particularly when the systems are large and sparse [7,8].

Domain decomposition techniques divide the model into subdomains and each domain is assigned to a processor. Then the system of equations can solved by a Krylov subspace method, e.g. the Conjugate Gradient method or one of its variants, involving the coefficient matrix only in the form of matrix-by-vector products. The parallel implementation of the CG algorithms requires two types of communications: global communications for the inner products and local communications for the update of the interface variables at each iteration step. On a distributed (LAN based) parallel platform communications are costly and the bottleneck is usually due to the inner products, which by becoming relatively more important when the number of processors is increased, may affect the parallel performance of the algorithm [9-11].

The performance of the solution algorithms also depends on the efficiency of the matrix storage schemes and of the communication patterns. The development of data structures appropriate for distributed matrix computations is an evolving field of research, as can be recognised in the technical literature, e.g. [12-16]. For most of these approaches the data structures for communications must be explicitly programmed and different schemes are adopted to keep track of the shared variables [15,17,18].

The present paper reports on a set of data structures for the representation of distributed linear systems, which were designed to attend a larger set of objectives:

- to efficiently organise the communications required by the parallel computation using iterative solution methods;
- to preserve generality and robustness, allowing to use both non-conventional and conventional formulations.

Details of these data structures and specific design options are presented next.

4. Data distribution and matrix data structures

The non-conventional finite element models provide a disjoint partition of the mesh, without overlapping,
inter-element continuity is assured by variables strictly associated with the boundaries. When solving linear elastic problems, with an uniform degree of approximation, the structural variables associated to each element have comparable dimensions, consequently, to obtain an equilibrated and well balanced partition, it is enough to assign to each processor an equal number of elements from a continuous domain, whenever possible reducing the interface boundary.

For the presentation of the distribution scheme and of the associated data structures a non-optimal partition of a simple finite element mesh into three subdomains is exemplified in Fig. 1a.

The subdomain data set, Fig. 1b, includes all the topological entities of the subdomain: Nodes, Loops and Edges [2,3]; as well as the edges connected to the boundary nodes, “ghost-Edges”, and all the elements adjacent to the boundary nodes, “ghost-Loops”. Elements (Loops), are owned by one subdomain. The “ghost” entities are used when the data locality information is organised, in order to minimise communications between neighbour processors.

4.1. Definition of the Part structure

The matrices arising from the alternative finite element models have a logical block structure, which is tightly connected to the topological structure of the mesh. This property is exploited to organise the structural variables and the associated matrix.

This is achieved by considering the vectors involved as “vectors of vectors” and therefore dividing the global vector into component arrays, called Parts. Each Part uses a real array to store the vector entries and an integer to describe the dimension of the corresponding vectors.

To exemplify the identification of the Part structures, the governing system of the Hybrid-Mixed Stress (HMS) model is used. However, all the assumptions made here are valid for other non-conventional finite element models.

4.1.1. Identification of the Parts on the domain

The behaviour of a 2D problem with mixed boundary conditions, presented in Fig. 2, modelled with HMS elements is described by system equation.

\[
\begin{bmatrix}
F & A_f' & -A_{f1} & -A_{f2} \\
A_f' & 0 & 0 & 0 \\
-A_{f1} & 0 & 0 & 0 \\
-A_{f2} & 0 & 0 & 0 \\
-A_{f3} & 0 & 0 & 0 \\
-A_{f4} & 0 & 0 & 0
\end{bmatrix}
\begin{bmatrix}
s_f \\
qu_f \\
qu_{f1} \\
qu_{f2} \\
qu_{f3} \\
qu_{f4}
\end{bmatrix}
= \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
-\Omega_{f4}
\end{bmatrix}
\]

Eq. (2) is a particular case of Eq. (1), where vector \( x \) has one super-column corresponding to the variables used for the approximation of the stresses, \( s_f \), and displacements, \( q_v \), in the element and vector \( y \) has three super-columns, corresponding to the variables used for the approximation of the displacements on each side, \( q_{\gamma_i} \).

Different groupings of the components of the solution vector may be carried out. The most obvious is to assign a different Part to each component [19], i.e. to vectors \( s_f, q_V \) and \( q_{\gamma_{\gamma}} \), \( i = 1, 2, 4 \) will correspond the \( P_{s_f}, P_{q_V} \) and \( P_{q_{\gamma_{\gamma}}} \) Parts, creating a direct relation between the mesh entities and the corresponding Parts.

This option implies the generation of a high number of Parts associated to the model and implicitly increases the time necessary to process them. One alternative, which reduces the number of Parts, is to group shared or non-shared component vectors. For this example, which has only one subdomain, it would imply grouping variables \( s_f, q_V \) and \( q_{\gamma_{\gamma}} \), into a single Part.

4.1.2. Identification of the Parts on distributed data

In a distributed environment we reorder the solution vector components using a criterion based on the sharing of data. Then one Part holds the vectors associated to the internal variables of the subdomain, while the variables shared with each adjacent subdomain are grouped in different Parts. The number of shared Parts is equal to the number of subdomains adjacent to the current subdomain, as illustrated in Fig. 3, when the 2D problem is discretised, using the mesh shown in Fig. 1a, for subdomain S1.

Fig. 1. Domain decomposition and data distribution: (a) global mesh, (b) partitioned mesh.

Fig. 2. Sample problem modelled with Hybrid-Mixed Stress elements.
The governing system, for subdomain \( S_1 \), associated to this organisation of the variables is described by,

\[
\begin{bmatrix}
S_{11} & -A_{f_{12}} & -A_{f_{13}} \\
-A_{f_{12}} & 0 & 0 \\
-A_{f_{13}} & 0 & 0
\end{bmatrix}
\begin{bmatrix}
x_{11} \\
x_{f_{12}} \\
x_{f_{13}}
\end{bmatrix} =
\begin{bmatrix}
b_{11} \\
0 \\
0
\end{bmatrix},
\]

(3)

where vector \( x_{11} \) corresponds to the variables of the internal problem of subdomain \( S_1 \) as identified by \( \text{Part} P_{S_{11}} \) and matrix \( S_{11} \) has a block structure.

For computations involving interface variables data exchange takes place between the corresponding shared Parts. For this reason information defining the locality of data is needed at Part level. This is controlled using the corresponding fields, \( \text{NSubDomains} \) indicates whether the part is shared or not, while the subdomains that share that part are indicated by the \( \text{*SubDomains} \) integer array of length \( \text{NSubDomains} \). Variable number holds, in a condensed form, the same information.

These fields are sufficient to manage the interprocessor communications and are simple to set up.

The simplified implementation of the Part structure is given next.

```c
struct Part {
    int id;
    int number;
    int NSubDomains, *SubDomains;
    int Dimension;
    double *x[MAX_VECTORS];
    ...
    Part *previous, *next;
};
```

The vector of pointers \( x \), of dimension \( \text{MAX_VECTORS} \), is used to access the different vectors intervening in the solution process. These vectors are dynamically allocated.

### 4.2. Relations between the topological entities and the Part structures

The approach used for the design of the Part assumes that a direct relation between each topological entity and the corresponding \( \text{Part} \) can always be established. This mapping information is held by each topological entity (Loop, Edge or Node) using pointers to the \( \text{Part} \), via an auxiliary Local variable. This pointer is used when the mesh entity generates structural variables or is zero if the associated solution vector is empty. This connection between the Loop, Edge and the Part structures is exemplified in Fig. 4. Details of the procedure used to setup the \( \text{Part} \) pointers are given in Section 4.5.

The links between the topological entities that induce structural variables and the \( \text{Part} \) structures are formulation dependent, as shown in Table 1 for conventional and non-conventional formulations.

Any combinations of the entities shown above can be carried out, to handle other formulations or combinations of formulations.

### 4.3. Definition of the Block structure

The submatrices in governing system (2) can be indexed by the corresponding sets of variables or \( \text{Parts} \), \( \text{i.e} \), the \( \text{Parts} \) can be used as a coordinate system to access the submatrices, termed Blocks, corresponding to each set of variables, or \( \text{Part} \).

The Block structure is designed to store the matrix entries and to describe the behaviour of the corresponding \( \text{Parts} \). Its main components, where the two pointers to the \( \text{Part} \) structures, row and column, define the position of the Block within the matrix, are presented next.

```c
struct Block {
    /* corresponding vectors information */
    Part *row, *column;
    int allocdim;
    /* matrix storage - Block independent */
    int nonzeros;
    int *irow, *icolum;
    double *vals;
    ...
    Block *previous, *next;
};
```

Within each Block, in the present approach, the matrix coefficients are stored using a general sparse (row, column, value) format [20,21], but the Block structure has no limitations regarding the storage scheme. A change would only imply the modification of the routines operating on matrices.

This structure may be extended to attend specific problems. For example, in the case of elastoplastic problems, the system matrix may change some of its coefficients during the incremental process. Then only these coefficients are reevaluated. When formulation specific preconditioners are used the preconditioning matrix can also be included in the Block structure.

In the Block mapping of system (2), presented in Fig. 5a, the structure associated to Block \( B_{d_{i}} \) will have
the row and column pointer set to Parts $P_{sv}$ and $P_{sv}$, respectively. This corresponds to a fine grained data organisation.

In a distributed environment a coarse grained grouping of the variables is adopted on the subdomain data (Section 4.1.2). The corresponding Block mapping of system (3) is presented in Fig. 5b.

4.4. Part–Block relations

As each Part is linked to a given set of variables and the numbering of the variables in each Part starts from zero, each Part identified by the same number holds the same set of structural variables. Using this scheme there is no need to establish a global numbering of the variables, because the Part structures manage the translation of the information for each submatrix or Block. The Parts for a non-conventional formulation and for a conventional (variables associated with nodes) formulation are identified, for the domain decomposition of Fig. 1, in Fig. 6a and b, respectively.

The corresponding global matrices and the subdomain matrices are presented in Fig. 7. It can be seen in Fig. 7a, that for non-conventional formulations the contributions of each edge/element from different subdomains do not overlap, whereas for conventional formulations the coefficients are obtained by summing the contributions of adjacent elements, represented by Part2, Part4, Part5 and Part6 in Fig. 7b.

4.5. Parallel implementation aspects for the Part and Block structures

The data structures, Part and Block, are created only at subdomain level. They are stored in chained lists and an hash table is used to reference the numbers.

The Part structures are set up using a simple approach. All mesh entities which induce structural variables, as defined in Table 1, are checked and the following variables which identify the corresponding Part are computed: (i) the number and list of subdomains which share the entity; (ii) a unique number associated to each subdomain list; and (iii) the dimension/number of the associated variables.
If a Part with the same unique number already exists, the Part pointer in the Local structure of the mesh entity is set and the offset value of the variable or variables set is also determined. If the search result is zero, then a new Part will be created and inserted in the Part list.

The system in each domain/subdomain is described by a matrix of pointers to Blocks with dimension \( \text{num-Parts} \times \text{numParts} \). Thus, the construction of the Block structure is based only on the Parts, abstracting from the mesh entities. As can be seen in Fig. 5 some of the Blocks will be empty according with the specific pattern of the formulation. After the matrix computation has been completed the empty blocks are eliminated.

The interdependency between these structures is guaranteed by the two pointers to Part structures present in the Block. During the matrix computation phase the Blocks are addressed using a matrix of pointers to avoid the repeated search over the Part list. In the solution step the Parts are addressed directly from the Blocks and the matrix of Block pointers may be freed.

The insertion of the component matrices in the Block structure is done using interface routines, where the Block "location" described by its Parts, the offset values within the Block and the matrix values in different formats are the arguments. The exclusive use of these interface routines is required for two reasons. The first and most important is to preserve the modularity of the computer code by providing an abstraction layer, the second is that the insertion of the submatrices may require memory reallocations which must be globally controlled. The location and offset are directly obtained from the corresponding topological data structures.

5. Procedures which operate on Blocks and Parts

5.1. Matrix vector product

Matrix vector products are implemented using general algorithms for sparse (row, column, value) and symmetric or general matrix vector operations.

The operation \( y = Ax \) is performed at subdomain level, using a general sparse matrix–vector procedure, i.e. each Block is multiplied by the corresponding column Part and result is added to the row Part. After these
calculations have been performed the result obtained in each shared Part (the ones that belong to more than one subdomain) is sent to the neighbour(s), while expecting to receive as many sets of values as the number of neighbours. The result of the matrix vector product is obtained by adding all these values to the local vector.

5.2. Interprocessor communications

After performing the local matrix vector product each processor needs to receive the corresponding result vector from NumParts-1 neighbouring processors. Conversely, each processor needs to send a contribution to its counterparts. The Part structure holds the data necessary to control interprocessor communications, by checking the Part list and identifying the shared Parts. From the corresponding field of the Part structure the neighbour subdomains and their processors are identified.

To update the global vector, the local product is sent to each “other shared part” of the neighbour processors, as indicated by NSubDomains and *SubDomains. When data is received from any processor its origin is determined using the number identifier and the local values are updated with the terms received, using a saxpy [22] like routine.

The algorithm described in Fig. 8 summarises the local or inter-processor communications in order to update a global vector using single message buffers.

Integers, ix and w, indicate the vector to be updated and the working vector used to collect the partial contributions, integer icount is the number of messages which are expected to be received by this processor.

The routine presented in Fig. 8 may be serialised in order to avoid network congestion, without affecting the organisation of the data structures. Another variant that uses multiple message buffers for communications, reducing the buffer initialisation overhead was also implemented, but the gain in execution time was found to be minimal.

5.3. Vector operations

The vector operations, inner product of two vectors, vector sums and vector updates (saxpy’s), were implemented using the highly optimised Blas [22] library routines at the level of the Parts.

```c
for (part = FirstPart; part; part = part->next)
{
    if(part->NSubDomains > 1) {
        pvm_initsend(ENCODING);
        pvm_pkint(&part->number, 1, 1);
        pvm_pkdouble(part->x[ix], part->Dimension, 1);
        
        for (i = 0; i < part->NSubDomains; i++)
            if((sd_id = part->SubDomains[i]) != me) {
                pvm_send(TaskDef.tids[sd_id-1], msgtag);
                icount++;
            }
    }
} /* end for part */

/* Receive data from tasks */
while(icount > 0)
{
    if((bufid = pvm_trecv(-1, msgtag, &twait))
    {
        pvm_upkint(&nr, 1, 1);
        part = GetFromNr(nr);
        pvm_upkdouble(part->x[w], part->Dimension, 1);
        saxpy(&part->Dimension, 1, part->x[ix], 1, part->x[w], 1);
    }
    icount--;
} /* end while */
```

Fig. 8. Local communication algorithm: UpdatePart().
For saxpy operations, each processor updates the components of the vectors it is mapping. These operations remain local without need for communications.

The inner product of two arbitrary vectors, \( x \) and \( y \), is only partially a local operation and requires interprocessor communications. The contribution of each subdomain to the inner product of two global vectors \( (x,y) \) is obtained by summing the partial results for all parts in the subdomain.

The repeated contribution of each shared part is corrected by applying a correction, equal to the inverse of the number of shared subdomains, to the partial result. The repeated calculation of some of the contributions to the inner product represents a parallel overhead, but ensures the correctness of the final result. This overhead is low in a non-conventional finite element model where the boundary edge can be shared only by two processors in the 2D case. In the case of conventional formulations the overhead is more important, but in “normal” meshes it is moderate because it depends on the number of processors that share a common vertex.

6. Numerical examples

Results obtained from two applications, consisting in the elastostatic analysis of a bending plate and of a 3D arch dam, are discussed to illustrate the parallel performance of the framework.

The parallel virtual machine is an homogeneous platform, a cluster of three bi-processor computers (AMD Athlon MP 1500+), connected using a Gigabit Ethernet switch. The PVM package (Version 3.4.0) [23] was the message-passing library used.

The iterative solution procedures implemented for the solution of the \( Ax = b \) linear system obtained from a domain decomposition, use the values of \( b \) as initial value for vector \( x_0 \) (\( x_0 = b \)) and the stopping criterion is \( ||r||/||b|| \leq \text{tol} \), with the value of \( \text{tol} \) equal to \( 10^{-6} \).

6.1. Plate bending problem

The first example is the elastostatic analysis of a built-in slab (Fig. 9) with uniformly distributed load, modelled with Hybrid Mixed Stress (HMS) and Displacement (HMD) elements using Legendre polynomials as approximation functions [24].

When the slab is modelled with HMS elements, the degrees of the polynomials used for the approximation of the moments, shear stresses, transversal displacements and rotations on the domain were 16, 15, 14 and 15, respectively. On the static boundaries the approximation of transversal displacements was of degree 15 and degree 16 was used for rotations. In the case of the HMD the degrees of the Legendre polynomials used for the approximations of the moments, displacements, shear stresses and rotations were 16, 15, 14 and 13 in the domain and 14 and 13 for moments and shear stresses on the boundaries. Table 2 presents, for a discretisation with 16 elements, the characteristics of associated system: dimension \( (n_{\text{dof}}) \), number of nonzero entries \( (n_{\text{nz}}) \) and sparsity index, \( (\eta) \), defined as the ratio of zero entries in the matrix relative to the total number of coefficients.

The parallel performance using two global algorithms was studied. The details of these algorithms, a rearranged version of the Conjugate Gradient iterations [9] applied to the pre-scaled matrix (CGE-NP-S2) and the scaled hybrid Bi-Conjugate Gradient–Quasi Minimal Residual [25] method for indefinite systems (BQMR-NP-S2), are discussed in [2]. The wall-clock times, number of iterations, speedup and efficiency when the number of processors, \( P \), is increased are presented in Figs. 10 and 11.

The convergence rate of the global solution methods in the presence of scaling S2 is fairly stable as the number of iterations remains practically unchanged when the number of subdomains increases. The only exception happens for the CGE-NP-S2 algorithm on one processor, then the higher number of iterations needed to converge, implies the super-linear behaviour observed in Fig. 11 for 2 and 3 processors.

Table 2

<table>
<thead>
<tr>
<th></th>
<th>( n_{\text{dof}} )</th>
<th>( n_{\text{nz}} )</th>
<th>( \eta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>HMS</td>
<td>33,456</td>
<td>308,944</td>
<td>0.99972</td>
</tr>
<tr>
<td>HMD</td>
<td>34,160</td>
<td>279,824</td>
<td>0.99976</td>
</tr>
</tbody>
</table>

Fig. 9. Built-in slab, geometry, problem definition and discretisation.
The curves, in Fig. 11, show that parallel performance diminishes with the increase of the number of processors. This behaviour appears because of the rapid growth of the communication/computation ratio, for fixed size problems [26], where most of the time is spent on solving the linear system.

The percentages of time spent on interprocessor communications, vector operations and matrix-by-vector products, for algorithm CGE-NP-S2, are shown in Fig. 12.

It can be observed that the load unbalance for five and six processors causes idling on some processors, increasing the communication time. This effect is not observed for three processors, when the parallel process is computationally dominant.

The CGE-NP-S2 algorithm achieves better performance compared to the BQMR-NP-S2 algorithm. This happens because of the lower communication/computation ratios obtained for the CGE-NP-S2 algorithm, where communications overlap computations, reducing the global communication cost.

The results obtained for the HMD model indicate a low convergence rate, when compared with those obtained for the HMS model, requiring an higher number of iterations to achieve convergence. This behaviour is a consequence of the worse conditioning of the system.

However, in the authors opinion, the parallel performance with an efficiency above 50% for up to six processors is a promising behaviour. The parallel performance of the matrix–vector and vector operations is good, as observed in Fig. 12, confirming the efficiency of the supporting data-structures.

6.2. Cabril dam

Results obtained from the elastostatic analysis of the Cabril arch dam, discretised with Hybrid Mixed Stress...
3D elements [27], are presented next. As the corresponding
3D mesh can be topologically projected on a plane, the 3D
model was implemented using an “hybrid” procedure, ori-
ented for plane data structures. Fig. 13 shows a projection
of the 55 element mesh, which was originally used at LNEC
[28].

The finite element model uses Legendre approximation
functions of degree 3 for the stresses in the volume and
for the displacements on the boundary faces and of degree
2 for the displacement field in the volume. The associated
system is indefinite and has 33,681 degrees of freedom with
4,383,930 nonzero entries. The sparsity decreases (0.99614),
compared with “equivalent” 2D models.

The mesh partitionings of the model are presented in
Fig. 14, while the corresponding distributions of variables
are presented in Table 3, in terms of the maximum ($M$)

\begin{align*}
E &= 20.0 \text{ Gpa} \\
\nu &= 0.15 \\
\bar{h}_{\text{water}} &= 110 \text{ m} \\
\gamma &= 24 \text{ kN/m}^3
\end{align*}

Fig. 13. Geometry and mesh discretisation of the Cabril dam.

Fig. 14. Mesh partitioning of the Cabril dam model for three, four, five and six processors.
Table 3
Cabril dam model

<table>
<thead>
<tr>
<th>P</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_{M_g}$</td>
<td>17,172</td>
<td>11,589</td>
<td>8586</td>
<td>6813</td>
<td>6150</td>
</tr>
<tr>
<td>$n_{M_s}$</td>
<td>16,509</td>
<td>11,046</td>
<td>7923</td>
<td>6717</td>
<td>5463</td>
</tr>
<tr>
<td>$n_{M_g}$</td>
<td>192</td>
<td>336</td>
<td>312</td>
<td>384</td>
<td>384</td>
</tr>
<tr>
<td>$n_{M_s}$</td>
<td>192</td>
<td>168</td>
<td>120</td>
<td>96</td>
<td>120</td>
</tr>
</tbody>
</table>

Maximum ($M$) and minimum ($m$) number of total ($g$) or shared ($s$) degrees of freedom, for different number of subdomains/processors.

The wall-clock times and speedup results are given in Fig. 15 for two solution algorithms, the preconditioned Conjugate Gradient method [29] applied to the reduced system (SCHUR-PS-S3) and the combined Bi-Conjugate Gradient–Quasi Minimal Residual iteration on the global pre-scaled system (BQMR-NP-S3). Details of these algorithms are also given in [2].

The speedup values for the global BQMR-NP-S3 algorithm show a “normal” behaviour of problems with low computation/communication ratios. The distribution of the percentages of the total time is shown in Fig. 16.

The global algorithm scales well with the number of processors, but the costly global communications influence the increase of the parallel speedup, as the number of processors exceeds a threshold value. Nearly linear speedup is obtained for up to three processors but the load unbalance.

Fig. 15. Wall-clock times and speedup for the Cabril dam model, using algorithms BQMR-NP-S3 and SCHUR-PS-S3.

Fig. 16. Percentage of the time for interprocessor communications, vector operations, matrix-by-vector product and internal systems factorization. Cabril dam model, using algorithms BQMR-NP-S3 and SCHUR-PS-S3.
that appears in the decomposition implies some irregularities, which can be observed for four and six processors.

The matrix associated to this problem is ill-conditioned and indefinite. To reduce the number of iterations required by iterative solvers to achieve convergence it is necessary to use robust preconditioners, but the design of such parallel preconditioners is a topic for future research.

The efficiency of the SCHUR-PS-S3 algorithm, in combination with a direct solver and the simple diagonal preconditioner, was tested, but its use is conditioned by the memory required to factorise the internal system. To overcome this problem, in the present computational context, performing all calculations in-core, a reference number of three processors was considered.

The super-linear behaviour that is observed for this algorithm is due to the combination of several factors: (i) the fast decrease of the computational cost of factorising the internal systems, which may be a consequence of changing the matrix structure, improving its conditioning and of the cache-effects; (ii) the local communications becoming dominant, unlike what happens in the case of global iterative solvers.

As the matrix–vector operations include back-substitutions of the factorised Schur complement, Fig. 16 must be interpreted with care. Effectively the growth of the percentage of time spent on matrix–vector operations balances with the decrease of the factorization time, i.e. the total matrix–vector operation time to be considered is equal to the sum of these two parcels, which is nearly constant in percentage of time, as observed for the BQMR-NP-S3 algorithm.

7. Conclusions

Unified data structures, for matrix computations in FEM analysis, which can be used for both sequential and parallel (distributed memory) approaches, were proposed.

These data structures, developed for matrix and vector storage, Block and Part, provide a natural mapping between a domain decomposition and the corresponding system. The Part structure can fully represent all the information about the variables associated with it, describing the interconnectivity relations between adjacent subdomains with a minimum effort. This feature avoids the explicit implementation of data structures for interprocessor communications, since the location of the variables sets is defined when the structure of the matrix is determined. The Part structure also embeds the information of the component matrices in the global matrix. The relations with the topological data structures, Nodes, Loops and Edges, are efficient and simple to organise.

The Block data structure and its relation with the corresponding Parts allows to organise the structural matrices irrespectively of the characteristics of the FE model and simplifies its handling in all the intervening operations. The Block structure does not have any limitation on the use of known matrix storage formats.

The parallel performance, due to the relative nature of the speedup and efficiency, must be interpreted in the context of the parallel system consisting of the parallel algorithm, the application and the parallel platform. The results obtained show that, for applications with a reasonable computation/communication ratio, the values of speedup and efficiency on an homogeneous, network based, parallel virtual machine, are promising. Further improvements and optimisations are needed in order to fully assess the proposed approach.

Acknowledgements

This work is part of the research activity carried out at ICIST, Instituto de Engenharia de Estruturas, Território e Construção, and has been partially supported by FEDER and Fundação para a Ciência e Tecnologia through grant PRAXIS XXI/BD/9754/96, project POCTI/ECM/33066/99 and the funding of the research unit.

References

[15] Zhang K, Wu YS, Ding C, Pruess K, Elmroth E. Parallel computing techniques for large-scale reservoir simulation of multicomponent


