Non-Nested Multigrid Methods for Linear Problems

MARCO L. BITTENCOURT (*)
CRAIG C. DOUGLAS (**)
RAÚL A. FEIJÓO (***)

* Center for Computational Sciences
University of Kentucky
325 McVey Hall, Lexington, KY, 40506-0045, USA
e-mail: mlb@ccs.uky.edu

** Department of Mathematics
University of Kentucky
715 Patterson Office Tower, Lexington, KY, 40506-0027, USA
e-mail: douglas@ccs.uky.edu

*** Laboratório Nacional de Computação Científica (LNCC/CNPq)
Av. Getulio Vargas 333, CEP 25651-070, Petrópolis, RJ, Brazil
e-mail: feij@alpha.lncc.br

SUMMARY

This paper presents an application of non-nested and unstructured multigrid methods to linear elastic problems. A variational formulation for transfer operators and some multigrid strategies, including adaptive algorithms, are presented. Expressions for the performance evaluation of multigrid strategies and its comparison with direct and preconditioned conjugate gradient algorithms are also presented. A C++ implementation of the multigrid algorithms and its quadtree and octree data structures are discussed. Some two and three dimensional elasticity examples are analyzed.

1 Introduction

The application of the finite element method [1] for solving linear elliptic problems requires the solution of

$$\mathbf{A}\mathbf{u} = \mathbf{b}, \tag{1}$$

where $\mathbf{A}$ is a symmetric matrix of order N and $\mathbf{u}$ and $\mathbf{b}$ are the vectors of unknown and independent terms. Direct, iterative, and multigrid algorithms are commonly used to solve (1) when it is sufficiently similar enough to Poisson's equation on an uniform mesh [1, 2, 11].

An example of a linear elliptic problem is linear elasticity. The weak form associated with a linear elasticity problem is given by

$$\int_\Omega \mathbf{T}\,(u)\cdot\mathbf{E}\,(v)\;\,dV = \int_{\Gamma^2} \mathbf{\Phi}\cdot v\;dA + \int_\Omega \mathbf{f}\cdot v\;dV, \qquad \forall v \in \mathcal{V}, \tag{2}$$

where $\mathcal{V}$ is the vector space of admissible displacements [17]. $\mathbf{T}$ is the Cauchy stress tensor and $\mathbf{f}$ is the body force vector field. For a linear elastic problem, the stress tensor is related to the infinitesimal strain tensor $\mathbf{E} = \frac{1}{2}\left(\nabla\mathbf{u} + \nabla\mathbf{u}^T\right)$ linearly as $\mathbf{T} = \mathsf{C}\,[\mathbf{E}] = 2\mu\mathbf{E} + \lambda(\mathrm{tr}\;\mathbf{E})\mathbf{I}$, where C is the elasticity tensor, $\mathbf{u}$ is the displacement vector field, and λ and μ are the Lamé's coefficients. The essential and natural boundary conditions on $\partial\Omega$ are defined by $\mathbf{u} = \mathbf{0}$ $(\mathbf{x} \in \Gamma^1)$ and $\mathbf{T}\mathbf{n} = \mathbf{\Phi}$ $(\mathbf{x} \in \Gamma^2)$. We partition the boundary into $\partial\Omega = \Gamma^0 \cup \Gamma^1 \cup \Gamma^2$ and $\Gamma^0 \cap \Gamma^1 \cap \Gamma^2 = \emptyset$. Γ^0 is a

part of the boundary $\partial\Omega$ without any prescribed boundary condition, $\mathbf{\Phi}$ is the surface force vector field, and $\mathbf{n}$ is the outward normal vector. The discrete application of a Ritz-Galerkin method [1] for problem (2) requires the solution of (1).

Multigrid methods use several meshes for solving (1). Computational elements like nested iterations, coarse grid correction, transfer operators, and relaxation schemes are applied [2]. Traditionally, multigrid methods have been used with nested meshes. This simplifies transfer operators, mesh generation, definition of adaptive refinement criteria [11, 19, 25], and convergence theory for some class of problems [12, 18, 22].

However, engineering problems have complex geometries, which sometimes makes it difficult to generate a sequence of nested meshes. Thus, using non-nested approximation spaces is an interesting option. This option was applied to fluid flow problems in [21, 23] assuming that the number of variables in the finest mesh is sufficiently large so that the cost of mesh generation is negligible when compared to the cost of solving the problem. Appropriate data structures are necessary to transfer information among the meshes.

The convergence of non-nested multigrid methods is analyzed in [10, 14, 26]. The operators and meshes used in this paper meet the requirements given in there. Hence, the convergence of our multigrid methods applied to our examples is guaranteed.

This paper treats non-nested and unstructured multigrid methods applied to linear elastic problems. A generic variational formulation for restriction and prolongation operators is presented in Section 2. Data structures for data retrieval between meshes are presented in Section 3. Multigrid strategies including adaptive procedures are presented in Section 4. CPU and memory costs are presented in Section 5. A C++ implementation of the algorithms described in this paper is presented in Section 6. The results for two and three dimensional linear elastic examples modeled with triangular and tetrahedral finite elements are presented in Section 7. Finally, we draw some concrete conclusions in Section 8.

2 Variational Formulation of Operators

This section defines the restriction operators I_k^{k-1} and the prolongation operators I_{k-1}^k that are used to transfer information between two meshes Ω^{k-1} and Ω^k. The approach used here is valid for both nested and non-nested meshes. The definition allows transfer operators to be determined for problems with multiple degrees of freedom (e.g., plate and shell bending and mixed models). Its extension to nonlinear problems is immediate.

The following standard variational problem is solved by a multigrid method: *find* $u \in H_0^1(\Omega)$ *such that*

$$a(u,v) = b(v), \qquad \forall v \in H_0^1(\Omega). \tag{3}$$

$H_0^1(\Omega)$ is the Hilbert space whose elements are functions with zero values on the boundary, are square integrable, and whose first derivatives are also square integrable.

The approximation by finite elements and multigrid techniques requires solving the following finite dimensional problems: *find* $u_k \in V_k$ *such that*

$$a(u_k,v) = b(v), \qquad \forall v \in V_k. \tag{4}$$

V_k is a finite dimensional subspace of $H_0^1(\Omega)$.

For linear finite elements, $V_k(\Omega) = \{v \in C_0(\Omega) \mid v|_K \text{ is linear } \forall K \in \mathcal{T}_k\} \subset H_0^1(\Omega)$. $C_0(\Omega)$ is the space of continuous functions in Ω that are zero along the boundary. $\{\mathcal{T}_k, k = 1, 2, \ldots\}$ is a family of non-nested triangular meshes in the Ω domain. Hence, $\mathcal{T}_k \not\subset \mathcal{T}_{k+1}$ and $V_k(\Omega) \not\subset V_{k+1}(\Omega)$.

Let the usual projection operator I_k in V_k be given by $I_k : C_0(\Omega) \to V_k(\Omega)$ such that

$$u(x) \to I_k u(x) = \sum_{n_i \in \mathcal{N}_k} u(n_i)\phi_{k,i}(x) = \Phi_k \cdot \mathbf{u}_k.$$

$\mathcal{N}_k$ is the set of nodal points corresponding to partition $\mathcal{T}_k$, $\phi_{k,i}(x)$ is the interpolation function associated with node n_i corresponding to the finite element type defining the triangulation $\mathcal{T}_k$, and Φ and $\mathbf{u}$ are the vectors of the interpolation function and nodal values.

Let $\tilde{u}_k \in V_k$ be an approximation to the solution of (4). The associated residual is

$$\tilde{r}_k(v) = b(v) - a(\tilde{u}_k, v) = \sum_{n_i \in \mathcal{N}_k} [b(\phi_{k,i}) - a(\tilde{u}_k, \phi_{k,i})]v(n_i) = \mathbf{r}_k \cdot \mathbf{v}_k, \quad \forall v \in V_k. \tag{5}$$

The residual is defined for all $v \in V_k$. For $v = I_k v_{k-1} \in V_k$ we can define the functional $r_k^{k-1} \in \mathcal{L}(V_{k-1}, \Re)$ by

$$\begin{aligned}
r_k^{k-1}(v_{k-1}) &\equiv \tilde{r}_k(I_k v_{k-1}) = b(I_k v_{k-1}) - a(\tilde{u}_k, I_k v_{k-1}) \\
&= \sum_{n_i \in \mathcal{N}_k} [b(\phi_{k,i}) - a(\tilde{u}_k, \phi_{k,i})]v_{k-1}(n_i) \\
&= \sum_{n_i \in \mathcal{N}_k} \left([b(\phi_{k,i}) - a(\tilde{u}_k, \phi_{k,i})] \sum_{n_j \in \mathcal{N}_{k-1}} v_{k-1}(n_j)\phi_{k-1,j}(n_i) \right) \\
&= \sum_{n_j \in \mathcal{N}_{k-1}} \left(\sum_{n_i \in \mathcal{N}_k} [b(\phi_{k,i}) - a(\tilde{u}_k, \phi_{k,i})]\phi_{k-1,j}(n_i) \right) v_{k-1}(n_j) \\
&= \mathbf{r}_{k-1} \cdot \mathbf{v}_{k-1}.
\end{aligned} \tag{6}$$

The transfer of the residual $\mathbf{r}_k$ (which is related to $\tilde{u}_k$) from mesh $\mathcal{T}_k$ to mesh $\mathcal{T}_{k-1}$ consists in determining the residual

$$\mathbf{r}_{k-1} = I_k^{k-1}\mathbf{r}_k = (\mathbf{N}_{k-1}^k)^T \mathbf{r}_k, \tag{7}$$

where

$$\mathbf{N}_{k-1}^k = \begin{bmatrix} \phi_{k-1,1}(n_1) & \phi_{k-1,2}(n_1) & \dots & \phi_{k-1,N_{k-1}}(n_1) \\ \phi_{k-1,1}(n_2) & \phi_{k-1,2}(n_2) & \dots & \phi_{k-1,N_{k-1}}(n_2) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_{k-1,1}(n_{N_k}) & \phi_{k-1,2}(n_{N_k}) & \dots & \phi_{k-1,N_{k-1}}(n_{N_k}) \end{bmatrix}_{N_k \times N_{k-1}}. \tag{8}$$

Note that I_k^{k-1} is a $N_{k-1} \times N_k$ sparse matrix. Its generic element $(I_k^{k-1})_{pq} = \phi_{k-1,p}(n_q)$, $p \in \{1, \dots, N_{k-1}\}, q \in \{1, \dots, N_k\}$, corresponds to the value of the interpolation function associated with node n_p of triangulation $\mathcal{T}_{k-1}$ when calculated on the coordinates of node n_q from triangulation $\mathcal{T}_k$. From (7), I_k^{k-1} is consistent in the sense that the residual is consistent between two consecutive meshes.

From a computational viewpoint the restriction operator is calculated by taking the local contribution of the residual on node $n_i \in \mathcal{T}_k$ of the nodes of triangle $T_g \in \mathcal{T}_{k-1}$ where n_i is located. Figure 1 shows the operator for linear triangles. Residue $\mathbf{r}^I$ on fine node I is transferred as $\mathbf{r}^1$, $\mathbf{r}^2$, and $\mathbf{r}^3$ to the nodes of the coarser element using area coordinates A_1, A_2, and A_3. Using the notation indicated in Figure 1, the residual for each node of the triangle T_g due to the residual at node I contained by T_g is given by

$$\mathbf{r}^i = A_i \mathbf{r}^I, \qquad 1 \leq i \leq 3. \tag{9}$$

After ν_1 iterations of the relaxation scheme on level k, the following residual correction problem is solved on level $k-1$: *find* $e_{k-1} \in V_{k-1}$ *such that*

$$a(e_{k-1}, v) = r_k^{k-1}(v), \quad \forall v \in V_{k-1}, \tag{10}$$

where $r_k^{k-1}(v) = b(I_k v) - a(\tilde{u}_k, I_k v),\ \forall v \in V_{k-1}$.

Another multigrid component is prolonging (interpolating) the solution between levels $k-1$ and k in a consistent way with the restriction and diffential operators used. Note that $\mathbf{e}_k = \mathbf{N}_{k-1}^k \mathbf{e}_{k-1}$ and $I_{k-1}^k = \mathbf{N}_{k-1}^k$. Hence, $I_{k-1}^k = (I_k^{k-1})^T$. Once corrections $\mathbf{e}^1$, $\mathbf{e}^2$, and $\mathbf{e}^3$ are known, the value for finer node I is calculated as $\mathbf{e}^I = A_1\mathbf{e}^1 + A_2\mathbf{e}^2 + A_3\mathbf{e}^3$ (see Figure 1).

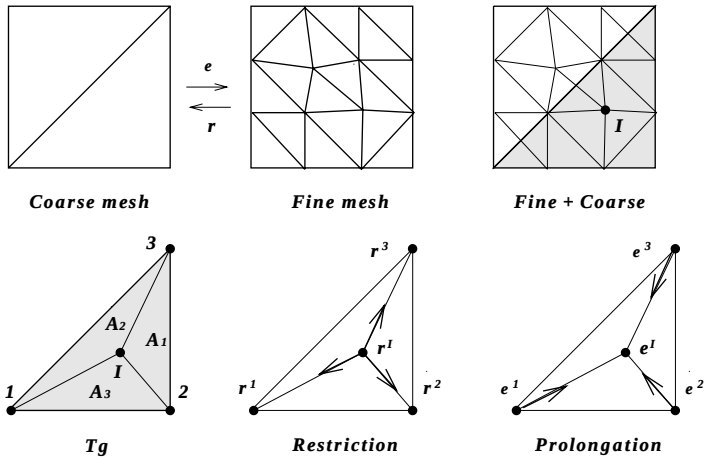


Figure 1: Restriction and prolongation operators.

3 Information Retrieval between Meshes

For the application of non-nested restriction and prolongation operators we need to know for each finer node the coarse grid element that contains the node and the respective values of the shape functions taken on the local coordinates of the finer node (see Figure 1). It is very important to use appropriate data structures and procedures for efficiently retrieving the necessary information for the calculation of operators.

We must do a *geometric search*. This procedure is frequently found in other areas, e.g., mesh generation and solid modeling [9, 13]. Given n entities, we need to know which ones contain a certain point. The entities may be a single or a set of mesh elements. The trivial or direct solution consists of sweeping the list of entities and checking if the specified point is inside one of the entities. The computational time for this procedure is $\mathcal{O}(n)$. Procedures to decrease this linear cost are discussed in [13] for two dimensional problems. The extension to three dimensional problems is immediate.

We decrease the order of the search problem by dividing the considered region into a set of cells with simple shapes containing a number of elements. Initially it is necessary to determine the cell containing the point. Then a search of the associated elements is conducted to obtain the elements where the point is located.

A preprocessing ordering step is used. We divide the region into cells and obtain the associated elements. The cost is at least $\mathcal{O}(n)$, which is similar to the direct search procedure. The application of the search procedure is essential only when the total number of search operations is large. The

memory requirements for auxiliary data structures to store the cells and their elements must be considered when balancing the CPU cost versus the memory use [13].

The quadtree data structure can be adapted easily to any set of elements by limiting the maximum number of elements per cell. A quadtree is a sequence of squares superimposed on a region. They are recursively divided according to cell and element sizes in each part of the domain (see Figure 2). The bigger square is known as the tree root, the squares with no subdivisions are the leaves, and the others are the branches. Procedures for data structure creation and information search are presented in [13] with a total cost of $\mathcal{O}(logn)$.

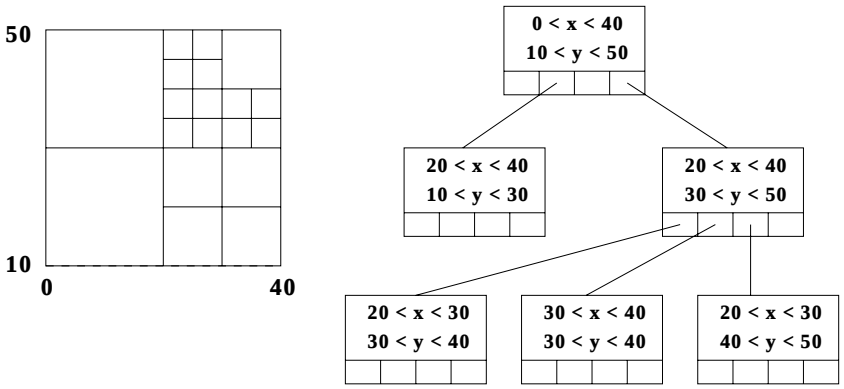


Figure 2: Auxiliary data structure based on quadtree [13].

We used a quadtree data structure in two dimensiona to find the coarser element containing a finer node and its corresponding linear shape function values. This is a preprocessing step before using a multigrid algorithm. We store a data structure that is accessed every time the interpolation and restriction operators are calculated. For three dimensional problems we used an octree data structure.

4 Multigrid Methods

Figure 3 shows some commonly used multigrid cycles. The V and W cycles are based on the recursive application of the coarse grid correction scheme. When going from a finer mesh Ω^k to a coarser mesh Ω^{k-1}, ν_1 pre-relaxations are performed on the original equation $\mathbf{Au} = \mathbf{b}$ or on the error equation $\mathbf{Ae} = \mathbf{r}$. Then the residual $\mathbf{r}$ is calculated and projected onto the mesh Ω^{k-1} using the restriction operator I_k^{k-1}. On the coarsest mesh the error equation is solved by a direct or iterative numerical method. Finally, corrections $\mathbf{e}$ are mapped onto finer levels using the prolongation operator I_{k-1}^k and ν_2 post-relaxations are then performed.

The FMV algorithm uses the concept of nested iterations. Each V cycle is preceded by $\nu_0 \geq 1$ V cycles on coarser grids. $\mathbf{Au} = \mathbf{b}$ is solved on the coarsest mesh and its solution is prolonged as an initial approximation to the next finer level. Then ν_0 V cycles are performed on that level. Ultimately a better approximation to the next mesh is obtained. This procedure is repeated until the finest mesh is reached. The FMW technique uses W cycles on coarser grids and is analogous to the FMV cycle. These procedures are illustrated in the Figure 3 for $\nu_0 = 1$. Some variations of these algorithms may be used, e.g., the FMVV scheme uses an FMV cycle followed by many V cycles.

Error estimators like the Zienkiewicz-Zhu (ZZ) procedure [27] may be used with multigrid strategies. We can define top down and bottom up adaptive multigrid procedures. Thus, we do not need to know all of the meshes in advance since they can be obtained by an adaptive process.

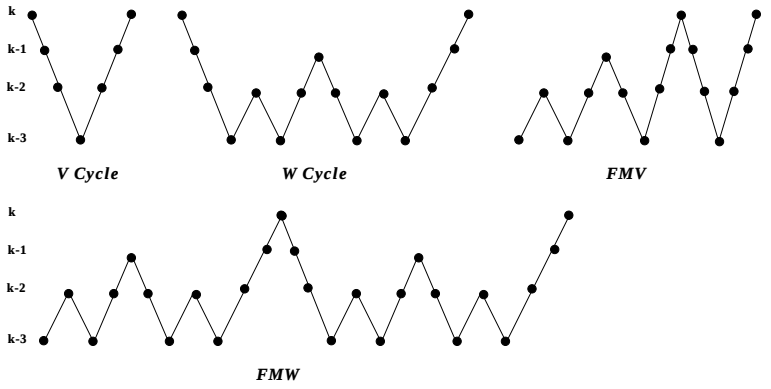


Figure 3: Multigrid cycles.

The top down approach produces a finest mesh to be used by a multigrid strategy. This mesh is obtained by an adaptive procedure using an error estimator, a refinement technique, and a direct or iterative solution method. A final error η smaller than a specified admissible error $\bar{\eta}$ must be attained on the finest grid. In the refinement procedure used in this work, the new average element size h' is calculated by dividing the current size of an element by 2 (i.e., $h' \approx h/2$). Sometimes the sequence of meshes obtained by this process does not guarantee the convergence of the multigrid procedure [2, 4]. Intermediate meshes must then be obtained by taking the finest mesh and using the mesh generator program with a new element size that grows by a factor of two [15]. After generating some meshes the multigrid algorithm provides the final approximate solution. A similar procedure without an adaptive refinement is defined in [20, 21, 23] in which the intermediate levels are generated from the finest mesh by applying frontal and Delaunay techniques.

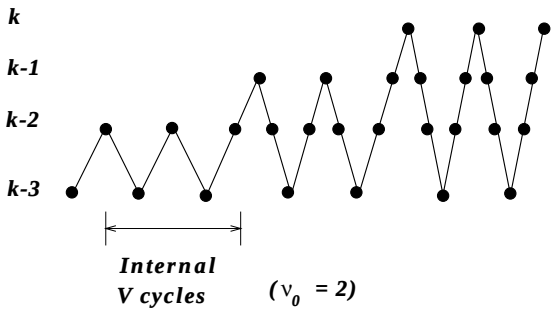


Figure 4: An adaptive FMV scheme.

Given a percentage error $\bar{\eta}$ and an accuracy ξ, the main idea of the bottom up approach is to start from the coarsest mesh and solve the problem by applying multigrid strategies. Meshes are generated according to the error distribution calculated by the ZZ error estimator. We chose the FMV cycle with $\nu_0 = 2$ (see Figure 4) as the multigrid strategy to generate the meshes. The coarsest mesh is solved by a direct method, the ZZ error estimator is applied, and a constrained refinement procedure is used to obtain the second mesh. Two intermediate V cycles are used and the third mesh is obtained similarly after the application of the ZZ procedure. This process is repeated until either reaching the specified maximum number of meshes or when the estimated error is close enough to the admissible value $\bar{\eta}$. For two dimensional problems the following refinement constraint was used to determine the new element size h':

$$\frac{1}{3}h \leq h' \leq \frac{3}{2}h. \tag{11}$$

This condition assured the convergence of the multigrid methods.

Note that in the intermediate levels, using two V cycles does not guarantee a solution with accuracy ξ. Since this strategy is used only for intermediate mesh definition it is not necessary to obtain an exact solution and therefore the adopted strategy is viable. However, when the finest mesh is obtained, a multigrid technique which is not necessarily a FMV with $\nu_0 = 2$ is used to solve the system of equations with accuracy ξ.

5 CPU and Memory Requirements

Table 2 presents the items used when calculating the number of operations and memory space required by the non-nested multigrid methods based on the computational implementation conducted [2]. Let N_m be the number of meshes. Table 1 gives the notation used for each mesh i.

NIT_i	Total number of relaxations conducted in all multigrid cycles
CIT_i	Cost of one iteration of the relaxation scheme
N_i	Number of equations
m_i	Average number of non-zero coefficients per row of the system matrix
n_i	Total number of times the multigrid technique visited level i
n_i^r	Total number of restriction operations
n_i^p	Total number of prolongation operations
N_{nodes}	Number of nodes in the considered element
n_i^{aux}	Memory allocation for auxiliary vectors required To implement the relaxation scheme
N_i^{nodes}	Total number of nodes
$n_t = 2.5$, $n_{eq} = 1$, $n_{inc} = 1.5$	For two-dimensional problems
$n_t = 3.5$, $n_{eq} = 1.5$, $n_{inc} = 2$	For three-dimensional problems
N_i^{els}	Total number of elements

Table 1: Notation for calculating the number of operations and memory space.

The number of operations is obtained by adding up the expressions given in Table 2 for relaxations, direct solution in the coarsest mesh, transfer operators, and calculation of residuals. The equivalent number of finest mesh iterations is obtained by dividing the number of operations by the cost of one finest mesh iteration of the relaxation scheme.

The higher memory requirements for multigrid methods compared with traditional iterative methods is due to the extra meshes combined with the extra space required for data used by the restriction and prolongation operators (e.g., incidence and equation numbering). Therefore it is necessary to take into account the systems of equations (matrices+vectors), tables of data for operators, equation numbering and incidence [2, 4]. The total memory space is given by summing all the expressions indicated in Table 2 multiplied by factor $\frac{8}{1024}$ in order to obtain results in kilobytes.

6 Computational Implementation

All procedures were implemented using C++ in a set of libraries with classes for a database, a matrix, and the finite elements [16]. Some classes for sparse matrices using a row-compressed format with direct and iterative solution methods were incorporated in these libraries [2]. For sparse Gaussian elimination, a symbolic procedure was used to identify the fill-in (i.e., to determine the elements that become non-zero along the factorization process) [24]. A minimum degree algorithm for renumbering of equations is included.

Table 2: Expressions for calculating the number of operations and memory space.

Operations	
Relaxations	$\sum_{i=2}^{N_m} (\mathrm{NIT}_i)\,(\mathrm{CIT}_i)$
Coarsest mesh	$\left[\sum_{i=1}^{N_1} (m_i - 1)(m_i + 2)\right] + 2N_1(m_1 - 1)n_1$
Operators	$N_{nodes} \sum_{i=2}^{N_m} N_i(n_i^r + n_i^p)$
Residual	$\sum_{i=2}^{N_m} N_i(2m_i - 1)n_i^r$

Memory space	
Systems	$\sum_{i=1}^{N_m} [N_i(m_i + 2) + n_i^{aux}]$
Operators	$n_t \sum_{i=2}^{N_m} N_i^{nodes}$
Equations	$n_{eq} \sum_{i=1}^{N_m} N_i^{nodes}$
Incidence	$n_{inc} \sum_{i=1}^{N_m} N_i^{els}$

The multigrid strategies were implemented in only one class with parameters for the database information, number of meshes, numbers of pre- and post-relaxations, and pointer vectors to the unknown and the independent terms of each mesh. The implementation of the multigrid techniques used the following methods:

```
int FineToCoarsePart(int Level);
int CoarseToFinePart(int Level);
```

The method FineToCoarsePart is responsible for the descendent part of the multigrid strategies: first ν_1 relaxations are executed, then the residual is calculated and transferred to the next coarse mesh. This is repeated until the coarsest mesh is reached.

The method CoarseToFinePart first solves the coarsest mesh by a direct method. Then it repeatedly prolongs corrections and executes ν_2 post-relaxations until the finest level is reached.

The multigrid strategies presented previously are easily implemented. For example, the main kernel of a V cycle consists of the following C++ commands:

```
//fine to coarse part
for(k = NumMeshes - 1; k >= 0; k--) FineToCoarsePart(k);

//coarse to fine part
for(k = 1; k < NumMeshes; k++) CoarseToFinePart(k);
```

The two methods FineToCoarsePart and CoarseToFinePart extensively use other methods for restriction and interpolation operators. Finally, it should be noted that the mapping between the meshes uses quadtree and octree data structures for two- and three-dimensional problems respectively. It is implemented using the classes developed in [13]. A more detailed description of the multigrid classes can be found in [8].

7 Numerical Experiments

To evaluate the performance of some multigrid schemes, several two and three dimensional linear elastic examples are examined. In Section 7.1, a plate in traction modeled with nested and non-nested meshes are considered [5, 7]. A fracture problem is studied in Section 7.2 [5, 6]. The same problem is analyzed in Section 7.3 using the adaptive procedures discussed in Section 4 and in [3]. In Section 7.4, two three dimensional examples (a beam and a piston) are studied [4]. All of the problems were discretized by linear triangle and tetrahedron meshes generated by frontal [15] and Delaunay's techniques [13].

The multigrid results were compared with ones for sparse Gaussian elimination (SGE) and iterative methods based on conjugated gradient (CG) with diagonal (CGD), SSOR (CGSS), and symmetric Gauss-Seidel (CGGS) preconditioners [1, 2]. For all of the examples the system matrix was stored in a row-compressed sparse format. In the case of iterative and multigrid methods the convergence criterion $\frac{||\mathbf{Au}-\mathbf{b}||_2}{||\mathbf{b}||_2} < \xi$ in the Euclidean norm with $\xi = 10^{-4}$ was used. Gauss-Seidel was used as a relaxation scheme in the multigrid cycles. The following relative errors were considered for comparisons of direct ($\mathbf{u}_{dir}$), iterative ($\mathbf{u}_{ite}$), and multigrid ($\mathbf{u}_{mg}$) methods:

$$||e_r^{dir/ite}||_2 = \frac{||\mathbf{u}_{dir} - \mathbf{u}_{ite}||_2}{||\mathbf{u}_{dir}||_2}, \qquad ||e_r^{dir/mg}||_2 = \frac{||\mathbf{u}_{dir} - \mathbf{u}_{mg}||_2}{||\mathbf{u}_{dir}||_2}, \text{ and} \qquad ||e_r^{ite/mg}||_2 = \frac{||\mathbf{u}_{ite} - \mathbf{u}_{mg}||_2}{||\mathbf{u}_{ite}||_2}. \quad (12)$$

For each method the equivalent number of finest mesh iterations (NIT) was determined by taking the corresponding overall computational cost divided by the cost of one Gauss-Seidel iteration on the finest mesh. A floating point operation (flop) is a multiplication.

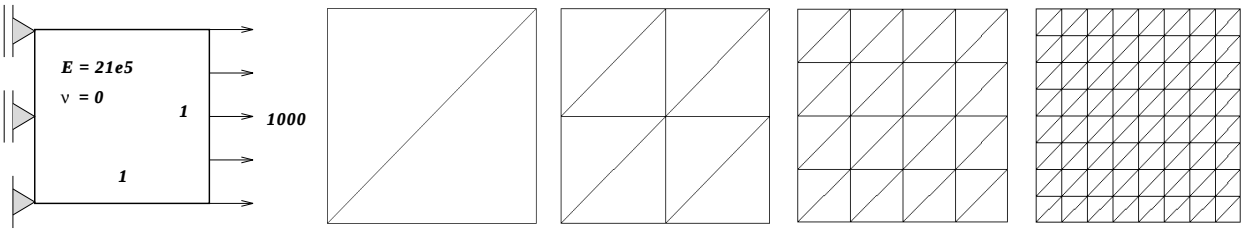


Figure 5: Plate under traction (nested meshes).

7.1 Plate under traction

Figure 5 shows a plate under traction with a distributed load. It is rigidly supported on a point and with a displacement constraint in the x direction in the other nodes. This problem is modeled as a plane stress case with a unit thickness. For multigrid analysis, four nested and non-nested meshes were generated (see Figures 5 and 6). The main features of these meshes are in Table 3: the numbers of nodes, elements, and equations. Also included are the total number of global matrix coefficients for direct (NCoef^{dir}) and iterative/multigrid (NCoef^{ite}) methods and the average number of coefficients per row of the sparse matrix for direct (m^{dir}) and iterative/multigrid (m^{ite}) methods. This example is equivalent to a unidimensional problem with an exact solution based on the approximation space obtained by linear finite elements since the Poisson coefficient is zero.

Table 4 presents the equivalent number of finest mesh iterations (NIT) for SGE, CGGS, and several multigrid strategies. In the latter case the total number of cycles (NC) and the number of pre- (ν_1) and post-relaxations (ν_2) are given. The relative errors (12) are also presented. The results in terms of megaflops are illustrated in Figure 7, where the coefficients of the fitted straight lines are given between brackets. For multigrid methods, 2-4 meshes were tried. The results obtained

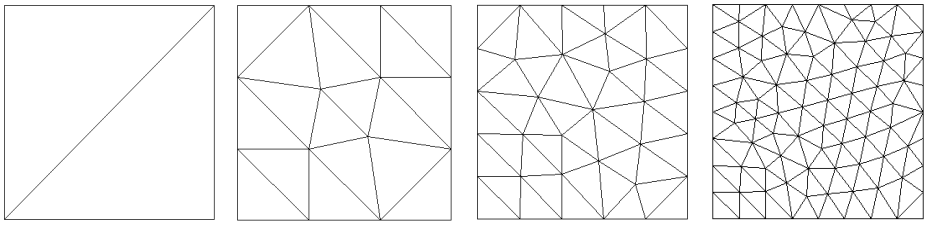


Figure 6: Plate under traction (non-nested meshes).

are shown in Figure 7. The solution on the coarsest mesh was computed by a direct method while in the other cases the same multigrid strategy indicated in Figure 7 was used.

Table 3: Mesh features for the plate.

Nested meshes							
Mesh	Nodes	Elements	Equations	NCoef^{dir}	m^{dir}	NCoef^{ite}	m^{ite}
1	4	2	5	13	2.6	13	2.6
2	9	8	14	73	5.2	63	4.5
3	25	32	44	357	8.1	253	5.8
4	81	128	152	2183	14.4	993	6.5
Non-nested meshes							
Mesh	Nodes	Elements	Equations	NCoef^{dir}	m^{dir}	NCoef^{ite}	m^{ite}
1	4	2	5	13	2.6	13	2.6
2	16	18	27	173	6.4	145	5.4
3	35	48	63	548	8.7	380	6.0
4	89	144	168	2223	13.2	1113	6.6

7.2 Fracture problem

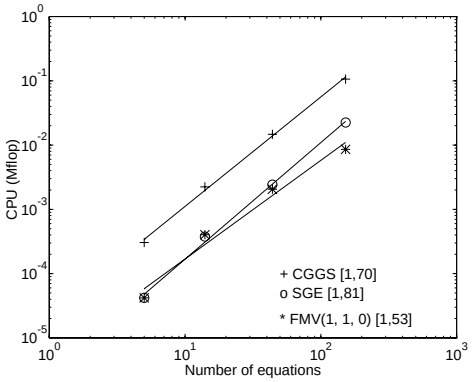
In this section, a fracture problem is analyzed by taking four linear finite element meshes (see Figure 8). Its corresponding features are given in Table 5. Table 6 presents the results for SGE, CGGS, and some multigrid methods. The results in terms of megaflops and memory space in kilobytes are illustrated in Figure 9.

7.3 Application for the adaptive procedure

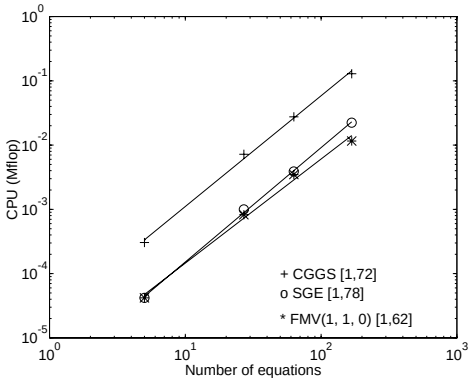
The fracture problem was analyzed with the adaptive multigrid strategy in Section 4. Taking a percentage error equal to $\bar{\eta} = 1\%$ with accuracy $\xi = 10^{-4}$. Three and six meshes were necessary for the direct and multigrid methods to obtain final estimated errors of 1.1% and 1.3%. The meshes obtained are illustrated in Figures 10 and 11. The main features are given in Table 7 where η is the error obtained at the end of the mesh generation process. These results are shown in Table 8 and Figure 12.

Table 4: Results for the plate under traction.

Nested meshes					
Method	NIT	NC, ν_1, ν_2	$\|\|e_r^{dir/ite}\|\|_2$	$\|\|e_r^{dir/mg}\|\|_2$	$\|\|e_r^{ite/mg}\|\|_2$
SGE	12	–	–	–	–
CGGS	58	–	1.21×10^{-5}	–	–
V	70	15,1,1	–	4.24×10^{-4}	4.29×10^{-4}
W	44	7,1,1	–	1.01×10^{-4}	1.06×10^{-4}
FMV	5	1,1,0	–	4.33×10^{-15}	1.21×10^{-5}
FMW	6	1,1,0	–	3.57×10^{-15}	1.21×10^{-5}
Non-nested meshes					
Mesh	NIT	NC, ν_1, ν_2	$\|\|e_r^{dir/ite}\|\|_2$	$\|\|e_r^{dir/mg}\|\|_2$	$\|\|e_r^{ite/mg}\|\|_2$
SGE	11	–	–	–	–
CGGS	63	–	9.85×10^{-6}	–	–
V	57	11,1,1	–	6.46×10^{-4}	6.49×10^{-4}
W	39	5,1,1	–	6.80×10^{-5}	7.18×10^{-5}
FMV	6	1,1,0	–	6.59×10^{-8}	9.87×10^{-6}
FMW	8	1,1,0	–	3.10×10^{-8}	9.86×10^{-6}



(a) Nested meshes.



(b) Non-nested meshes.

Figure 7: Number of operations for the plate under traction.

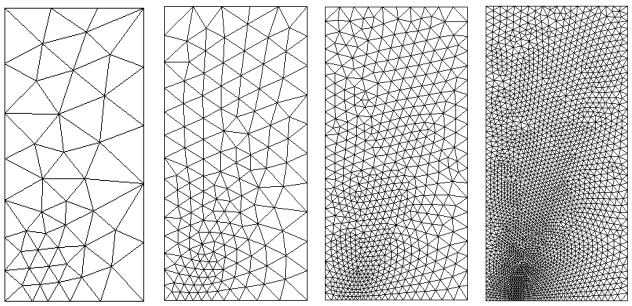


Figure 8: Meshes for the fracture problem.

Table 5: Mesh features for the fracture problem.

Mesh	Nodes	Elements	Equations	NCoef^{dir}	m^{dir}	NCoef^{ite}	m^{ite}
1	51	80	89	911	10.2	555	6.2
2	170	299	317	5607	17.7	2171	6.8
3	626	1171	1207	32422	26.9	8647	7.2
4	2383	4608	4679	204249	43.7	34312	7.3

Table 6: Results for the fracture problem.

Method	NIT	NC, ν_0, ν_1, ν_2	$\|\|e_r^{dir/ite}\|\|_2$	$\|\|e_r^{dir/mg}\|\|_2$	$\|\|e_r^{ite/mg}\|\|_2$
SGE	130	–	–	–	–
CGGS	273	–	1.37×10^{-5}	–	–
FMV	39	4,1,2,1	–	8.76×10^{-6}	1.86×10^{-5}
FMVV	41	8,2,2,1	–	2.42×10^{-5}	3.17×10^{-5}
FMW	38	3,1,2,1	–	3.43×10^{-6}	1.52×10^{-5}

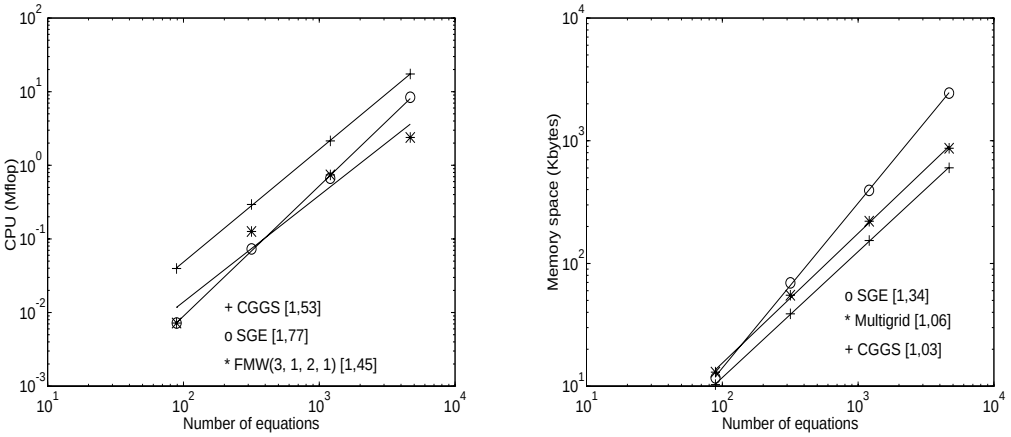


Figure 9: Number of operations and memory space for the fracture problem.

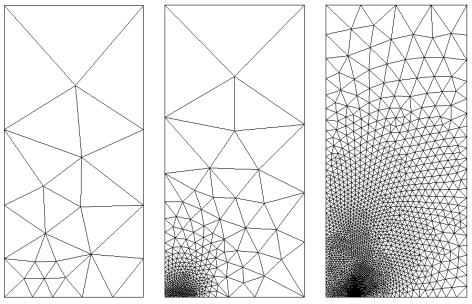


Figure 10: Meshes for the fracture problem using adaptive Gauss factorization method.

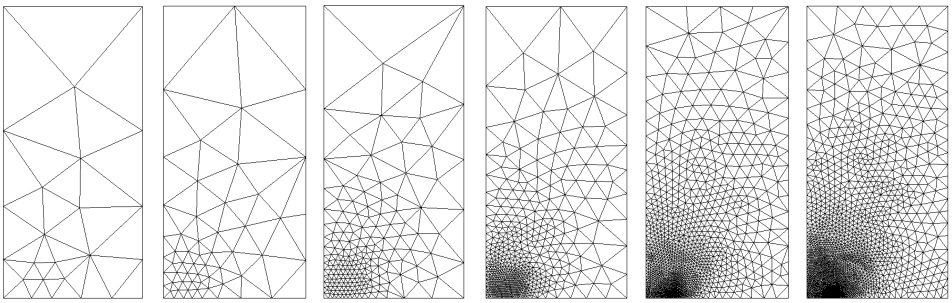


Figure 11: Meshes for the fracture problem using adaptive multigrid.

Table 7: Mesh features for the adaptive fracture problem.

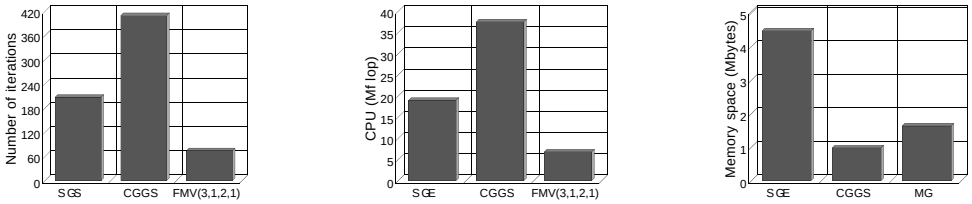
Adaptive direct procedure								
Mesh	η (%)	Nodes	Elements	Equations	m^{dir}	NCoef^{dir}	m^{ite}	NCoef^{ite}
1	9.3	25	34	40	7.1	283	5.5	220
2	4.2	404	747	775	25.9	20062	7.1	5503
3	1.3	3772	7350	7440	49.7	369737	7.4	54836
Adaptive multigrid procedure								
Mesh	η (%)	Nodes	Elements	Equations	m^{dir}	NCoef^{dir}	m^{ite}	NCoef^{ite}
1	9.3	25	34	40	7.1	283	5.5	220
2	7.7	61	97	107	11.8	1261	6.3	676
3	5.6	198	358	371	19.9	7393	6.9	2575
4	3.8	592	1119	1144	29.4	33666	7.2	8228
5	2.2	1681	3240	3293	40.3	132808	7.3	24076
6	1.4	3369	6560	6641	46.7	310313	7.4	48904

Table 8: Results for the fracture problem using adaptive procedures.

Method	NIT	NC, ν_0, ν_1, ν_2	$\|\|e_r^{dir/ite}\|\|_2$	$\|\|e_r^{dir/mg}\|\|_2$	$\|\|e_r^{ite/mg}\|\|_2$
SGE	207	–	–	–	–
CGGS	409	–	3.24×10^{-5}	–	–
FMV	89	4,2,1,1	–	3.34×10^{-5}	4.45×10^{-5}
	80	2,2,2,1	–	6.03×10^{-5}	6.57×10^{-5}
	88	5,1,1,1	–	2.93×10^{-5}	4.19×10^{-5}
	74	3,1,2,1	–	6.95×10^{-5}	7.39×10^{-5}
V	73	9,1,1,1	–	2.13×10^{-4}	2.11×10^{-4}
	73	7,1,2,1	–	1.96×10^{-4}	1.94×10^{-4}

7.4 Three dimensional problems

Figure 13 shows a cantilever beam with its dimensions and the applied load and four meshes generated with linear tetrahedral elements. As a second three dimensional example, we have a piston. We used four meshes (see Figure 14). On the coarsest mesh the boundary conditions and the concentrated load considered can be observed. The main features of these meshes are indicated in Table 9.



(a) NIT. (b) CPU. (c) Memory.

Figure 12: Results for the adaptive fracture problem.

Table 10 shows the number of iterations on the fine mesh (NIT) for every algorithm. For the multigrid strategies, the number of cycles (NC), the number of pre- and post-relaxations (ν_1,ν_2), the number of intermediary cycles (ν_0), and the relative errors (12) are computed. The finest mesh of the piston problem could not be solved by SGE due to memory limitations. Therefore only the relative error $||e_r^{ite/mg}||_2$ is shown. Figure 15 shows the behavior of the solution methods as a function of the number of operations.

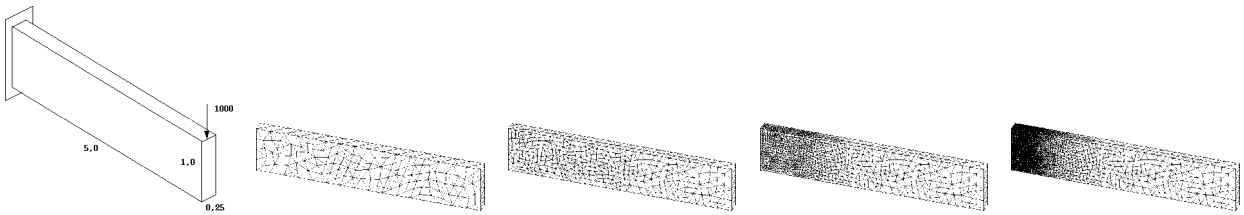


Figure 13: Meshes generated for the cantilevered beam example ($E = 1.0 \times 10^5$; $\nu = 0.3$).

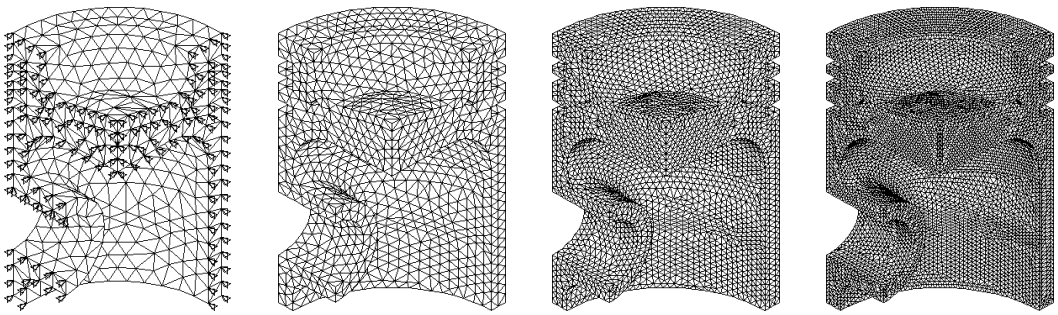


Figure 14: Meshes generated for the piston ($E = 1.0 \times 10^5$; $\nu = 0.3$).

Table 9: Mesh features for the beam and piston examples.

Beam							
Mesh	Nodes	Elements	Equations	m^{dir}	NCoef^{dir}	m^{ite}	NCoef^{ite}
1	335	921	963	40.0	38511	16.0	15444
2	1130	3922	3279	81.2	266226	17.6	57804
3	3024	11813	8724	164.2	1432572	18.6	162276
4	8633	37976	24606	361.9	8904168	19.6	482364
Piston							
Mesh	Nodes	Elements	Equations	m^{dir}	NCoef^{dir}	m^{ite}	NCoef^{ite}
1	861	2622	2416	75.2	181675	16.4	39654
2	2943	10888	8444	154.5	1304874	18.0	151630
3	12727	55480	37188	373.2	13900617	19.4	722428
4	32348	155792	95277	722.1	68798689	20.5	1948811

Table 10: Results for the beam and piston examples.

	Beam				Piston		
Method	NIT	NC, ν_0, ν_1, ν_2	$\|\|e_r^{dir/mg}\|\|_2$	$\|\|e_r^{ite/mg}\|\|_2$	NIT	NC, ν_0, ν_1, ν_2	$\|\|e_r^{ite/mg}\|\|_2$
SGE	4345	–	–	–	16934	–	–
CG	2361	–	–	–	5492	–	–
CGD	1279	–	–	–	4764	–	–
CGSS	844	–	–	–	3192	–	–
CGGS	672	–	–	–	2707	–	–
FMV	89	9,1,1,1	2.80×10^{-6}	2.80×10^{-6}	66	7,1,1,1	9.29×10^{-5}
	98	6,2,1,1	9.83×10^{-7}	9.84×10^{-7}	63	4,2,1,1	1.07×10^{-4}
FMW	101	8,1,1,1	5.74×10^{-8}	6.60×10^{-8}	61	4,1,2,1	8.96×10^{-5}
	111	5,2,1,1	3.59×10^{-8}	4.82×10^{-8}	62	3,2,1,1	8.76×10^{-5}
FMVV	117	25,1,1,1	1.28×10^{-5}	1.28×10^{-5}	78	16,1,1,1	1.27×10^{-4}
	115	23,2,1,1	1.17×10^{-5}	1.17×10^{-5}	75	15,2,1,1	1.30×10^{-4}

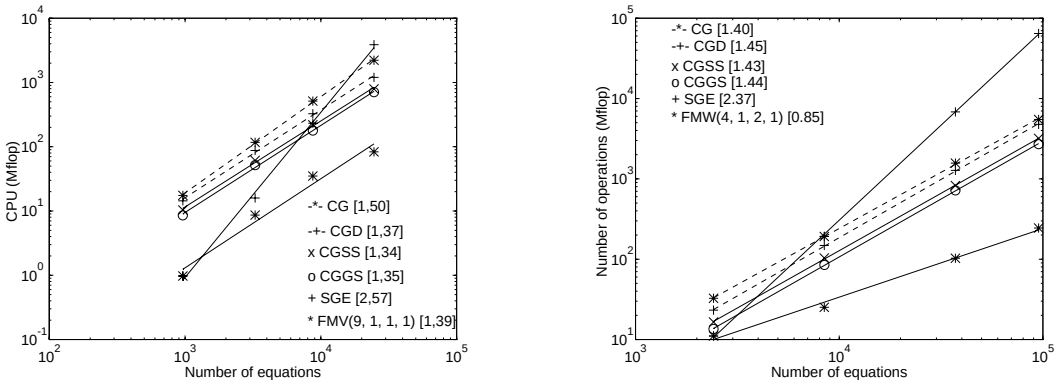
8 Conclusions

Based on the results obtained from the two dimensional examples and on those given in [3, 5], a number of conclusions can be drawn.

For the simple example of a plate under traction, the FMV and FMW algorithms are superior to the V and W cycles for problems in which it is possible to calculate a good initial approximation from the coarsest mesh. Also, the numerical conditioning of the solutions obtained from the FMV and FMW algorithms (measured by the relative errors (12)) is better than those obtained from the V and W cycles. Only one cycle of either the FMV or FMW algorithms with $\nu_1 = 1$ and $\nu_2 = 0$ is sufficient to solve this problem.

The number of operations required by the FMV and FMW algorithms is smaller than for either the SGE and CGGS methods. This is shown in Table 4 by comparing the number of iterations on the finest mesh. Also, see Figure 7. As expected, the CGGS method requires less memory space while the multigrid strategies and SGE have similar behavior [5]. Even for a small order example the use of multigrid becomes advantageous when compared to SGE and CGGS.

It was observed that the numerical conditioning of the solution obtained from the FMW and FMW strategies on non-nested meshes is inferior to the one obtained with nested meshes. However, the order of the relative error is acceptable. This indicates that we have actually produced a good approximation to the actual solution.



(a) Beam. (b) Piston.

Figure 15: Number of floating point operations.

For the fracture example (see Table 6 and on Figure 9) we note that the FMV, FMVV, and FMW procedures behave similarly: all of them are superior to SGE and CGGS. However, the FMVV method requires a larger number of cycles to obtain the same numerical conditioning. When considering the number of pre- and post-relaxations, both $\nu_1 = 1$ and $\nu_1 = 2$ usually reduce the total number of cycles [2].

Note that the angular coefficients of the lines representing the number of operations for the multigrid strategies are inferior to those for SGE and CGGS. The average behavior of the number of operations for the problems studied here and in [2] is shown in Figure 16(a). These coefficients are similar to those obtained previously in the examples presented in this paper and in [5].

In the adaptive procedure presented here, sequences of meshes were obtained by applying the ZZ error estimator. The purpose was to solve problems using either SGE, conjugated gradient, or a multigrid method. The strong singularity of the fracture problem resulted in a larger number of

equations needed to attain the 1% level of admissible error. The multigrid techniques are superior to SGE with respect to both the number of operations and the amount of memory [3]. For this quite singular problem and due to the limitation in modifying the element size (11), the number of meshes for multigrid is larger than the number of meshes for SGE.

We now draw a number of conclusions specific to the 3D problems in Section 7.4.

The iterative methods based on the conjugated gradient technique show a superior behavior when compared to SGE. It was observed that the demand for memory increases significantly when using SGE, making it impossible for us to solve the finest mesh of the piston example. The memory requirements were about 800 megabytes for storing the matrix and auxiliary vector. It is necessary to use a model reduction or a domain decomposition technique to solve this problem using SGE.

The meshes for the beam example result in a smaller number of equations (compared to the piston example), but still required a large number of iterations to converge (for both multigrid and the iterative techniques). This is due to the presence of large gradients in the solution. This fact explains the behavior of the coefficients of multigrid strategies in Figure 15.

Both FMV and FMW are better than FMVV. The former two methods required a smaller number of cycles to reach the specified accuracy.

Iterative methds should be used for three dimensional problems due to the smaller memory requirements. The acceleration obtained with the use of multigrid methods is significantly superior to conjugated gradient-based methods in terms of number of operations. The increase in memory space caused by the use of multigrid methods is insignificant, allowing all data used in the solution process to be loaded in the main memory. For the linear elastic problem studied in this paper, multigrid is significantly superior to the iterative algorithms based on either conjugated gradientsor SGE.

In the beam example, the meshes were generated for simulating the behavior of the error estimator. Due to the lower memory requirements and smaller number of iterations required for convergence, the adaptive procedure is recommended for three dimensional problems.

The adaptive multigrid procedure is a viable alternative. It performs no worse than SGE even for small order problems requiring a larger number of meshes. This is relevant even to quite singular problems. The procedure presented here allows the automation of the mesh generation procedure, in order to obtain a better sequence of meshes to achieve the specified admissible error $\bar{\eta}$.

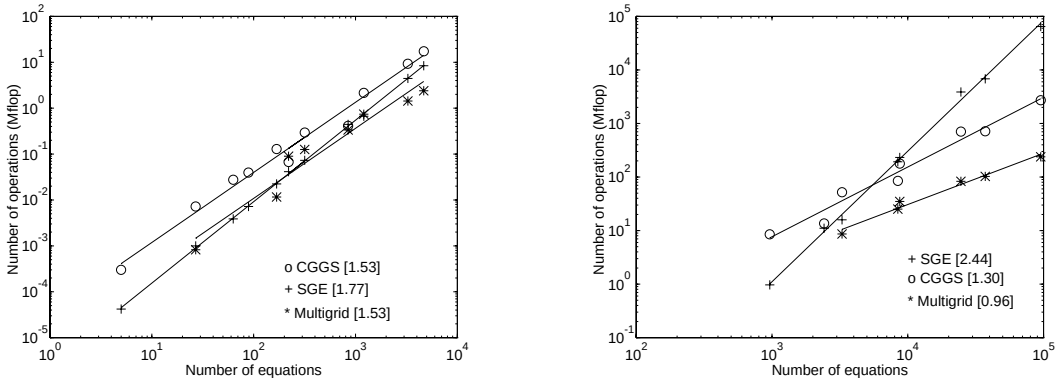
Figure 16(b) shows the behavior of SGE, CGGS, and multigrid. Note that for multigrid, the number of operations varied linearly with respect to the number of equations. Taking the average behavior of these two examples, the cost of the solution was approximately $\mathcal{O}(N)$ for a number of equations less than 100,000 on non-nested meshes.

Acknowledgments

This work was partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Fundação de Amparo a Pesquisa do Estado de São Paulo (FAPESP), NSF grants 9707040 and 9721388, NATO grant CRG 971574, and the University of Kentucky Center for Computational Sciences. The authors are grateful for the software facilities provided by the TACSOM Group (http://www.lncc.br/~tacsom).

References

[1] O. Axelsson and V. A. Barker. *Finite Element Solution of Boundary Value Problems – Theory and Computation.* Academic Press, Orlando, 1984.



(a) 2-D problems.

(b) 3-D problems.

Figure 16: Number of floating point operations for the examples analyzed.

[2] M. L. Bittencourt. *Adaptive Iterative and Multigrid Methods Applied to Non-Structured Meshes (in Portuguese)*. PhD thesis, State University of Campinas, Brazil, 1996.

[3] M. L. Bittencourt, C. C. Douglas, and R. A. Feijóo. Adaptive non-nested multigrid methods. Submitted for publication, 1998.

[4] M. L. Bittencourt, C. C. Douglas, and R. A. Feijóo. Non-nested and non-structured multigrid methods applied to elastic problems. Part II: The three-dimensional case. Submitted for publication, 1998.

[5] M. L. Bittencourt, C. C. Douglas, and R. A. Feijóo. Non-nested and non-structured multigrid methods applied to elastic problems. Part I: The two-dimensional case. Submitted for publication, 1998.

[6] M. L. Bittencourt and R. A. Feijóo. A comparative analysis between direct and iterative methods for the solution of systems of equations (in Portuguese). *Revista Internacional de Métodos Numéricos y Diseño em Ingenieria*, 13(2):123–148, 1997.

[7] M. L. Bittencourt and R. A. Feijóo. Multigrid methods in non-nested meshes applied to elastic problems (in Portuguese). *Revista Internacional de Métodos Numéricos y Diseño em Ingenieria*, 14(1):3–23, 1998.

[8] M. L. Bittencourt and R. A. Feijóo. Object-oriented non-nested multigrid methods. In *IV World Conference on Computational Mechanics*, Buenos Aires, June 1998.

[9] J. Bonet and J. Peraire. An alternating digital tree (ADT) algorithim for 3D geometric searching. *International Journal for Numerical Methods in Engineering*, 38:3529–3544, 1995.

[10] J. H. Bramble, J. E. Pasciak, and J. Xu. The analysis of multigrid algorithms with nonnested quadratic forms. *Mathematics of Computation*, 56(193):1–34, 1991.

[11] A. Brandt. Multi-level adaptive solutions to boundary-value problems. *Mathematics of Computation*, 31(138):333–390, 1977.

[12] W. L. Briggs. *A Multigrid Tutorial.* SIAM Books, Philadelphia, 1987.

[13] E. Dari. *Contribuciones a la Triangulación Automática de Dominios Tridimensionales.* PhD thesis, Instituto Balseiro, Bariloche, Argentina, 1994.

[14] C. C. Douglas. Multi-grid algorithms with applications to elliptic boundary-value problems. *SIAM Journal on Numerical Analysis*, 21:236–254, 1984.

[15] E. A. Fancello, A. C. S. Guimarães, R. A. Feijóo, and M. Venere. Automatic two-dimensional mesh generation using object-oriented programming. In *Proceedings of 11th Brazilian Congress of Mechanical Engineering*, pages 635–638, São Paulo, December 1991. Brasilian Association of Mechanical Sciences.

[16] A. C. S. Guimarães and R. A. Feijóo. The ACDP System. Research report 27/89, National Laboratory for Scientific Computation, Rio de Janeiro, Brazil, 1989.

[17] M. E. Gurtin. *An Introduction to Continuum Mechanics*, volume 158 of *Mathematics in Science and Engineering.* Academic Press, 1981.

[18] W. Hackbush and U. Trottenberg. *Multigrid Methods.* Springer-Verlag, Berlin, 1982.

[19] P. Leinen. Data structures and concepts for adaptive finite element methods. *Computing*, 55:325–354, 1995.

[20] R. Löhner and K. Morgan. An unstructured multigrid method for elliptic problems. *International Journal for Numerical Methods in Engineering*, 24:101–115, 1987.

[21] D. J. Mavriplis. Multigrid solution of the two-dimensional Euler equations on unstructured triangular meshes. *AIAA Journal*, 26(7):325–354, 1987.

[22] S. F. McCormick. *Multigrid Methods*, volume 3 of *Frontiers Series.* SIAM Books, Philadelphia, 1987.

[23] J. Peraire, J. Peiro, and K. Morgan. Multigrid solution of the 3D compressible Euler equations on unstructured tetrahedral grids. *International Journal for Numerical Methods in Engineering*, 36:1029–1044, 1993.

[24] S. Pissanetzky. *Sparse Matrix Technology.* Academic Press, London, 1984.

[25] U. Rüde. Fully adaptive multigrid methods. *SIAM Journal on Numerical Analysis*, 30(1):230–248, 1993.

[26] S. Zhang. *Multi-Level Iterative Techniques.* PhD thesis, Department of Mathematics, Pennsylvania State University, 1988.

[27] O. C. Zienkiewicz and J. Z. Zhu. A simple error estimator and adaptative procedure for practical engineering analysis. *International Journal for Numerical Methods in Engineering*, 24:337–357, 1987.