Gassav S. Abdoulaev
Yves Achdou
Yuri A. Kuznetsov
Christophe Prud'homme

On a parallel implementation of the mortar element method

ESAIM: Modélisation mathématique et analyse numérique, tome 33, nº 2 (1999), p. 245-259
<http://www.numdam.org/item?id=M2AN_1999__33_2_245_0>

Mathematical Modelling and Numerical Analysis M2AN, Vol 33, N° 2, 1999, p 245–259
Modélisation Mathématique et Analyse Numérique

ON A PARALLEL IMPLEMENTATION OF THE MORTAR ELEMENT METHOD

GASSAV S. ABDOULAEV[1], YVES ACHDOU[2], YURI A. KUZNETSOV[1] AND CHRISTOPHE PRUD'HOMME[3]

Abstract. We discuss a parallel implementation of the domain decomposition method based on the macro-hybrid formulation of a second order elliptic equation and on an approximation by the mortar element method The discretization leads to an algebraic saddle-point problem An iterative method with a block-diagonal preconditioner is used for solving the saddle-point problem A parallel implementation of the method is emphasized Finally the results of numerical experiments are presented

Résumé. Nous présentons ici un algorithme parallèle pour un problème elliptique du second ordre en trois dimensions discrétisé par une méthode d'éléments finis avec joints Le problème est écrit sous forme de point selle Nous utilisons une méthode itérative avec un préconditionneur diagonal par blocs Après avoir décrit l'algorithme parallèle, nous présentons des tests numériques

AMS Subject Classification. 65N55, 65M55.

Received June 10, 1997 Revised September 29, 1997

1. INTRODUCTION

Domain decomposition methods for solving the linear systems arising from the approximations of elliptic partial differential equations have been studied intensively by many researchers, motivated by the need to develop new algorithms for parallel computers.

In this paper, we deal with an algorithm for solving the linear systems arising from the *mortar* method, introduced by Bernardi, Maday and Patera [11]. The main feature of this method is that the continuity condition at the interface of the subdomains is treated in a weak form, *i.e.* the jump of the finite element solution on the interfaces should be L_2-orthogonal to a well chosen finite element space on the interface. Thus, it allows us to combine different approximations in different subdomains. In [11], it is shown that the approximation properties of such a method are optimal in the sense that the error is bounded by the sum of the subdomain-by-subdomain approximation errors. In this paper, we focus our interest on the *mortar finite element methods*, which permits in particular to choose the mesh independently in each subdomain. Therefore, it is possible to choose structured grids in the subdomains. This will give an opportunity to take advantage of very efficient preconditioners

Keywords and phrases Domain decomposition, mortar finite element method, saddle-point problem, preconditioned iterative method, parallel computing

[1] Institute of Numerical Mathematics, Russian Academy of Sciences, Ul Gubkina 8, GSP-1, Moscow, Russia

[2] INSA Rennes, 20 avenue des Buttes de Coesmes, 35043 Rennes, France and CMAP, École Polytechnique, 91128 Palaiseau Cedex, France e-mail `yves.achdou@insa-rennes.fr`

[3] ASCI, CNRS, Université Paris Sud, bâtiment 506, 91405 Orsay, France and INRIA Rocquencourt

It is possible to write a hybrid formulation of the mortar method, by introducing a Lagrange multiplier for the weak continuity constraint. The present paper is devoted to an iterative algorithm for the algebraic saddle point problem coming from this formulation. Other algorithms for this kind of saddle point problems have been studied in [3,16,24]. It is also possible to design algorithms for solving the systems arising from the formulation in the constrained space *i.e.* without Lagrange multipliers. For the two dimensional case, such algorithms are proposed in [5,6,13].

The efficiency of the iterative method to solve the algebraic system with a large-scaled matrix depends heavily on the preconditioner used. For the present system, we shall take the block diagonal preconditioner designed by Kuznetsov [23], where all the blocks but one correspond to the subdomains, and one smaller block corresponds to the interfaces. For building the blocks of the preconditioner corresponding to the subdomains, the multilevel structure of the grid is used: indeed, the subdomain preconditioner is based on a multilevel BPX-like method [12,19,27].

The blockwise structure of both the matrix and the preconditioner allows to parallelize efficiently the matrix multiplication and preconditioning procedures, the communication cost being very low compared to the cost of computation.

The paper is divided into four sections. In Section 2 the mortar finite element approximation of a self-adjoint elliptic equation in a 3D domain is considered. For the sake of simplicity the computational domain is assumed to be a union of parallelepipeds, and Q_1 finite elements on a uniform grid are used for each subdomain. In Section 3 the iterative method for the saddle-point problem and the preconditioner are described. Next, some remarks on parallel implementation aspects are given in Section 4. Finally, we present the results of numerical experiments on the parallel computer Cray T3E. The algebraic optimality of the method, as well as the weak dependence on the parameter c (see below), had been demonstrated numerically in the paper [1]. Recent results on application of the mortar element method for solving the Navier–Stokes equations and the Helmholtz wave equation can be found in [2].

2. The mortar method with Lagrange multipliers

Let Ω be a bounded domain of $\mathbb{R}^3$. We denote its boundary by $\partial\Omega$. We shall suppose that Ω is a union of m parallelepipeds Ω_k:

$$\Omega = \bigcup_{k=1}^{m} \Omega_k.$$

We shall suppose that the domain decomposition is geometrically conforming. It means that if $\gamma_{kl} = \overline{\Omega}_k \cap \overline{\Omega}_l \quad (k \neq l)$ and $\gamma_{kl} \neq \emptyset$, then γ_{kl} can be either a common vertex of Ω_k and Ω_l, or a common edge, or a common face. In the last case we define $\Gamma_{kl} = \gamma_{kl}$ as the interface between Ω_k and Ω_l. Note that $\Gamma_{kl} = \Gamma_{lk}$.

We consider the Neumann boundary value problem, but all the considerations below can be extended to a more general case:

$$\begin{aligned} -\Delta u + cu &= f \quad \text{in} \quad \Omega, \\ \frac{\partial u}{\partial \mathbf{n}} &= 0 \quad \text{on} \quad \partial\Omega, \end{aligned} \tag{1}$$

where $c \leq 1$ is a nonnegative constant, $f \in L^2(\Omega)$ is a given function.

The usual weak formulation of (1) reads as follows:

Problem I. *Find $u \in H^1(\Omega)$ such that*

$$\int_\Omega (\nabla u \nabla v + c\,uv)\,\mathrm{d}x = \int_\Omega fv\,\mathrm{d}x, \qquad \forall v \in H^1(\Omega). \tag{2}$$

Let us denote by $H^{\frac{1}{2}}(\Gamma_{kl})$ the trace space of one of the spaces $H^1(\Omega_k)$ or $H^1(\Omega_l)$ on the interface Γ_{kl}. We can define two product spaces:

$$V = \prod_{k=1}^{m} H^1(\Omega_k), \quad \Lambda = \prod_{k=1}^{m} \prod_{\substack{0 \le l < k \\ |\Gamma_{kl}| \neq 0}} \left(H^{\frac{1}{2}}(\Gamma_{kl})\right)'.$$

The space Λ will be a trial space for the weak continuity conditions on the interfaces.

We introduce the bilinear forms $a(\mathbf{u}, \mathbf{v}) : V \times V \to \mathbb{R}$, $b(\lambda, \mathbf{v}) : \Lambda \times V \to \mathbb{R}$ and the linear functional $f(\mathbf{v}) : V \to \mathbb{R}$:

$$\begin{aligned}
a(\mathbf{u}, \mathbf{v}) &= \sum_{k=1}^{m} a_k(\mathbf{u}, \mathbf{v}), & a_k(\mathbf{u}, \mathbf{v}) &= \int_{\Omega_k} (\nabla u_k \nabla v_k + c\, u_k v_k)\, \mathrm{d}x, \\
b(\lambda, \mathbf{v}) &= \sum_{k=1}^{m} \sum_{\substack{l=0 \\ |\Gamma_{kl}| \neq 0}}^{m} b_{kl}(\lambda, \mathbf{v}), & b_{kl}(\lambda, v) &= \langle \lambda_{kl}, v_k \rangle |_{\Gamma_{kl}}, \\
f(\mathbf{v}) &= \sum_{k=1}^{m} \int_{\Omega_k} f v_k\, \mathrm{d}x,
\end{aligned}$$

where $\lambda_{kl} = -\lambda_{lk}$, and where $\langle \cdot, \cdot \rangle |_{\Gamma_{kl}}$ stands for the duality product between $\left(H^{\frac{1}{2}}(\Gamma_{kl})\right)'$ and $H^{\frac{1}{2}}(\Gamma_{kl})$. The bilinear form $a_k(\cdot, \cdot)$ corresponds to the Neumann problem in the subdomain Ω_k for the operator $-\Delta + c\mathbf{I}$.

Provided the solution is smooth enough, (2) can be rewritten in the equivalent macro-hybrid form [3, 9]:

Problem II. *Find* $(\mathbf{u}, \lambda) \in V \times \Lambda$ *such that*

$$\begin{aligned}
a(\mathbf{u}, \mathbf{v}) + b(\lambda, \mathbf{v}) &= f(\mathbf{v}), \\
b(\mu, \mathbf{u}) &= 0, \\
\forall (\mathbf{v}, \mu) &\in V \times \Lambda.
\end{aligned} \tag{3}$$

For each subdomain Ω_k we use a uniform tensor-product grid Ω_{kh}:

$$\Omega_{kh} = \bigcup_{e_h \in \mathcal{T}_k} e_h.$$

If $e_h = [x_i; x_i + h_x] \times [y_j; y_j + h_y] \times [z_k; z_k + h_z]$, then denote $h_e = \max\{h_x, h_y, h_z\}$, and define a grid size of Ω_{kh} as $h_k = \max_{e_h \in \mathcal{T}_k} h_e$. Assume that there exist two positive constants c_1 and c_2, such that $c_1 h \le h_k \le c_2 h$ for some positive h and for all k. We define the subspace V_{kh} of $H^1(\Omega_k)$ as the space of the Q_1 finite elements [9] on the grid Ω_{kh}. Denote Γ_{klh} the trace of Ω_{kh} on Γ_{kl} and by W_{klh} the trace of the finite element space V_{kh} on Γ_{kl}. Let Λ_{klh} be a well chosen subspace [9] of one of the spaces W_{klh} or W_{lkh}. The mortar finite element problem in its hybrid formulation reads:

Problem III. *Find* $(\mathbf{u}_h, \lambda_h) \in V_h \times \Lambda_h$ *such that*

$$\begin{aligned}
a(\mathbf{u}_h, \mathbf{v}) + b(\lambda_h, \mathbf{v}) &= f(\mathbf{v}), \\
b(\mu, \mathbf{u}_h) &= 0, \\
\forall (\mathbf{v}, \mu) &\in V_h \times \Lambda_h,
\end{aligned} \tag{4}$$

with

$$V_h = \prod_{k=1}^{m} V_{kh}, \qquad \Lambda_h = \prod_{k=1}^{m} \prod_{\substack{0 \le l < k \\ |\Gamma_{kl}| \neq 0}} \Lambda_{klh}.$$

The approximation and convergence issues of the mortar element method were considered for example, in [8,10]. If $\tilde{u}$ and $\tilde{\lambda}$ denote the vectors of the components of $\mathbf{u}_h$ and λ_h in the corresponding nodal bases, the discrete problem is equivalent to the following saddle point system:

$$\mathcal{A}\begin{pmatrix}\tilde{u}\\ \tilde{\lambda}\end{pmatrix} = \begin{pmatrix}f\\ 0\end{pmatrix}, \quad \mathcal{A} = \begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix},$$

where

$$A = \begin{pmatrix} A_1 & & 0 \\ & \ddots & \\ 0 & & A_m \end{pmatrix}, \qquad B^T = \begin{pmatrix} B_1^T \\ \vdots \\ B_m^T \end{pmatrix},$$

and $n_k \times n_k$-matrices A_k correspond to the Neumann problem for each subdomain. The matrices A_k and B_k^T have the following block representation:

$$A_k = \begin{pmatrix} A_{Ik} & A_{I\Gamma k} \\ A_{\Gamma Ik} & A_{\Gamma k} \end{pmatrix}, \qquad B_k^T = \begin{pmatrix} 0 \\ B_{\Gamma k}^T \end{pmatrix}, \tag{5}$$

where index I stands for the components corresponding to the grid nodes which are inside Ω_k, and Γ — to the nodes on $\partial\Omega_k$.

3. Iterative procedure and preconditioner

In this section we give a brief description of the iterative method and the preconditioning algorithm, particularly emphasizing the implementational aspects. Here we only state the spectral equivalence of the constructed preconditioner to the matrix $\mathcal{A}$, whereas the full proof can be found in [23].

Let $\mathcal{B}$ be a symmetric and positive definite matrix. Suppose that eigenvalues of the spectral problem

$$\mathcal{A}x = \nu \mathcal{B}x$$

belong to the union of the segments $[d_1;\ d_2] \cup [d_3;\ d_4]$, where $d_1 \le d_2 < 0 < d_3 \le d_4$. Then it is possible to implement the generalized Lanczos method of minimal iterations [25] with the preconditioner $\mathcal{B}$ to solve the saddle point problem $\mathcal{A}x = y$. The algorithm is given below:

$$p_s = \begin{cases} \mathcal{B}^{-1}\xi^0, & s = 1, \\ \mathcal{B}^{-1}\mathcal{A}p_1 - \alpha_2 p_1, & s = 2, \\ \mathcal{B}^{-1}\mathcal{A}p_{s-1} - \alpha_s p_{s-1} - \beta_s p_{s-2}, & s \ge 3, \end{cases}$$

$$x^s = x^{s-1} - \gamma_s p_s, \quad s \ge 1,$$

$$\xi^s = \mathcal{A}x_s - y, \quad s \ge 0,$$

$$\alpha_s = \frac{(\mathcal{A}\mathcal{B}^{-1}\mathcal{A}p_{s-1}, \mathcal{B}^{-1}\mathcal{A}p_{s-1})}{(\mathcal{B}^{-1}\mathcal{A}p_{s-1}, \mathcal{A}p_{s-1})}, \quad s \ge 2,$$

$$\beta_s = \frac{(\mathcal{B}^{-1}\mathcal{A}p_{s-1}, \mathcal{A}p_{s-1})}{(\mathcal{B}^{-1}\mathcal{A}p_{s-2}, \mathcal{A}p_{s-2})}, \quad s \ge 3,$$

$$\gamma_s = \frac{(\mathcal{B}^{-1}\xi^{s-1}, \mathcal{A}p_{s-1})}{(\mathcal{B}^{-1}\mathcal{A}p_s, \mathcal{A}p_s)}, \quad s \ge 1.$$

The convergence rate for the method above can be estimated as follows:

$$||\xi^{2s}||_{\mathcal{B}^{-1}} \leq 2q^s||\xi^0||_{\mathcal{B}^{-1}},$$

$$q = \frac{\kappa - 1}{\kappa + 1}, \quad \kappa = \frac{\max\{d_4, |d_1|\}}{\min\{d_3, |d_2|\}}.$$

If we take as a preconditioner the block diagonal matrix

$$\mathcal{B} = \begin{pmatrix} R_u & 0 \\ 0 & R_\lambda \end{pmatrix},$$

where R_u is spectrally equivalent to A, R_λ is spectrally equivalent to $S_\lambda = BA^{-1}B^T$, and the corresponding spectral bounds do not depend on h and on the coefficient c, then we can prove [21, 23], that the iterative method has a rate of convergence also independent on h and c. The proof is based on a simple observation (Kuznetsov, 1990), that the eigenvalue problem

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} x = \nu \begin{pmatrix} A & 0 \\ 0 & S_\lambda \end{pmatrix} x$$

can have only three nontrivial solutions: $(1 - \sqrt{5})/2,\ 1,\ (1 + \sqrt{5})/2$.

Now the problem is to construct R_u and R_λ, satisfying the requirements on the spectra and such that solving systems with the matrices R_u and R_λ is inexpensive. It is natural to take R_u in the block-diagonal form

$$R_u = \begin{pmatrix} R_1 & & 0 \\ & \ddots & \\ 0 & & R_m \end{pmatrix}, \tag{6}$$

with the blocks R_k spectrally equivalent to the matrices A_k. The preconditioners R_k are local in the sense that each of them acts within a particular subdomain, and in general may be chosen independently, whereas R_λ acts in the whole mortar space Λ_h, and thus can be considered as global.

Let $\overset{\circ}{A}_k$ be a symmetric positive semi-definite matrix, generated by the bilinear form

$$a_k(u_h, v_h) = \int_{\Omega_k} \nabla u_h \cdot \nabla v_h \, dx, \qquad u_h, v_h \in V_{kh},$$

and denote M_k the subdomain mass matrix, defined by the relation

$$(M_k u_k, v_k) = \int_{\Omega_k} u_h v_h \, dx, \qquad u_h, v_h \in V_{kh}.$$

Hence, we have $A_k = \overset{\circ}{A}_k + cM_k$. Denote also

$$\widehat{A}_k = \overset{\circ}{A}_k + \frac{1}{d_k^2} M_k, \tag{7}$$

where d_k is a diameter of the subdomain Ω_k. We assume that there exist positive constants d, C_1, C_2, such that $C_1 d < d_k < C_2 d$ for all k.

Let P_k be the l_2-orthogonal projector onto the kernel of $\overset{\circ}{A}_k$, $\widehat{P}_k = I_k - P_k$ (I_k is the $n_k \times n_k$ identity matrix), and let H_k be any symmetric positive definite matrix, spectrally equivalent to $\widehat{A}_k^{-1}$. Then we define the preconditioner R_k as follows:

$$R_k^{-1} = \widehat{P}_k H_k \widehat{P}_k + \frac{1}{ch_k^3} P_k.$$

Estimates of the spectral bounds for this preconditioner can be found in [23].

The procedure of preconditioning S_λ consists of two stages: first, we construct some approximation $\widehat{S}_\lambda$ of the matrix S_λ, *i.e.* a matrix spectrally equivalent to S_λ such that the implementation of the product of a vector by $\widehat{S}_\lambda$ is relatively cheap; second, we "invert" $\widehat{S}_\lambda$ roughly by the generalized Chebyshev method with a preconditioner $\widehat{R}_\lambda$. Let us remind that matrices A_k and B_k can be decomposed blockwise as in (5).

Note that A_{I_k} is a symmetric positive definite matrix. Hence, the matrix S_λ can be written as

$$S_\lambda = \sum_{k=1}^{m} B_{\Gamma_k} S_{\Gamma_k}^{-1} B_{\Gamma_k}^T, \qquad S_{\Gamma_k} = A_{\Gamma_k} - A_{\Gamma I_k} A_{I_k}^{-1} A_{I\Gamma_k}. \tag{8}$$

Then, we take $\widehat{S}_\lambda$ in the form

$$\widehat{S}_\lambda = \sum_{k=1}^{m} B_{\Gamma_k} H_{\Gamma_k} B_{\Gamma_k}^T,$$

with the matrix H_{Γ_k}, spectrally equivalent to $S_{\Gamma_k}^{-1}$, constructed by

$$H_{\Gamma_k} = \widehat{P}_{\Gamma_k} \widehat{H}_{\Gamma_k} \widehat{P}_{\Gamma_k} + \frac{1}{cd_k h_k^2} P_{\Gamma_k},$$

where P_{Γ_k} is the l_2-orthogonal projector onto the one-dimensional space spanned by the constant vector of dimension n_{Γ_k}, n_{Γ_k} being the number of grid nodes on $\partial\Omega_k$, and $\widehat{P}_{\Gamma_k} = I_{\Gamma_k} - P_{\Gamma_k}$. The preconditioner $\widehat{H}_{\Gamma_k}$ must be spectrally equivalent to

$$\left(\overset{\circ}{S}_{\Gamma_k} + \frac{1}{d_k} M_{\Gamma_k} \right)^{-1},$$

where M_{Γ_k} denotes a $n_{\Gamma_k} \times n_{\Gamma_k}$ matrix, associated with the bilinear form

$$\int_{\partial\Omega_k} v_k u_k \, \mathrm{d}s,$$

and where $\overset{\circ}{S}_{\Gamma_k}$ is the Schur complement of the matrix $\overset{\circ}{A}_k$. One of possible definitions of the matrix $\widehat{H}_{\Gamma_k}$ will be given later in this section. Next, let us introduce an auxiliary matrix

$$\widetilde{R}_\lambda = \sum_{k=1}^{m} B_{\Gamma_k} \left(\frac{1}{h_k} \widehat{P}_{\Gamma_k} + \frac{1}{cd_k h_k^2} P_{\Gamma_k} \right) B_{\Gamma_k}^T,$$

which is derived from $\widehat{S}_\lambda$ by substituting $\widehat{H}_{\Gamma_k}$ by $(1/h_k) I_{\Gamma_k}$ (I_{Γ_k} – the $n_{\Gamma_k} \times n_{\Gamma_k}$ identity matrix) in the definition of H_{Γ_k}. Obviously, the matrix $\widetilde{R}_\lambda$ can be rewritten as

$$\widetilde{R}_\lambda = \sum_{k=1}^{m} \frac{1}{h_k} B_{\Gamma_k} B_{\Gamma_k}^T + \alpha_k B_{\Gamma_k} P_{\Gamma_k} B_{\Gamma_k}^T, \quad \alpha_k = \frac{1}{cd_k h_k^2} - \frac{1}{h_k}.$$

To ensure that α_k is positive, and hence, $\widetilde{R}_\lambda$ is positive definite, we have to impose the condition $h_k < 1/(cd_k)$ on the grid size h_k. The last inequality is not very restrictive and holds true for many practical cases, for instance, when $c \ll 1$ and $d_k = O(1)$.

Let D_λ be a diagonal matrix, spectrally equivalent to $\sum_{k=1}^m (1/h_k) B_{\Gamma_k} B_{\Gamma_k}^T$. The simplest choice, which had been implemented for numerical experiments, is $D_\lambda = h^3 \sum_{k=1}^m I_{\Gamma_k}$ [23]; another possibility consists in utilizing the mass lumping procedure for $B_{\Gamma_k} B_{\Gamma_k}^T$. The matrix $\widehat{R}_\lambda$ is next defined as

$$\widehat{R}_\lambda = D_\lambda + \sum_{k=1}^{m} \alpha_k B_{\Gamma_k} P_{\Gamma_k} B_{\Gamma_k}^T. \tag{9}$$

As shown in [23], the eigenvalues of the problem $\widehat{S}_\lambda w = \mu \widehat{R}_\lambda w$ belong to the segment $[\gamma_0,\, \gamma_1 d/h]$, where γ_0, γ_1 are positive constants, independent on h, d, c. Practically, these spectrum bounds can be computed in advance using, for example, the Lanczos algorithm [18].

Now we can finally define the preconditioner R_λ:

$$R_\lambda^{-1} = \left[I_\lambda - \prod_{l=1}^{L} \left(I_\lambda - \beta_l \widehat{R}_\lambda^{-1} \widehat{S}_\lambda \right) \right] \widehat{S}_\lambda^{-1},$$

where I_λ is the $n_\lambda \times n_\lambda$ identity matrix, β_l are the Chebyshev parameters [26], corresponding to the segment $[\gamma_0,\, \gamma_1 d/h]$. The algorithm for solving the problem $R_\lambda w_\lambda = v$ is given by:

$$\begin{aligned} & w^0 = 0, \\ & w^l = w^{l-1} - \beta_l \widehat{R}_\lambda^{-1} \left(\widehat{S}_\lambda w_{l-1} - v \right), \qquad l = 1, \dots, L, \\ & w_\lambda = w^L. \end{aligned}$$

Solving the system with the matrix $\widehat{R}_\lambda$ can be reduced to the factorization of a small matrix, its size being equal to the number of subdomains. Indeed, let us consider the system

$$\widehat{R}_\lambda w = v, \tag{10}$$

with $\widehat{R}_\lambda$, as defined in (9). Multiplying (10) with $P_{\Gamma_l} B_{\Gamma_l}^T D_\lambda^{-1}$, we can easily transform it to the system $(I_m + T)\widehat{w} = \widehat{v}$, where I_m is the $m \times m$ identity matrix (m - the number of subdomains), and entries of T, $\widehat{w}$, $\widehat{v}$ are defined as follows:

$$\begin{aligned} T_{kl} &= \alpha_l n_{\Gamma_l} \left(D_\lambda^{-1} B_{\Gamma_l} \mathbf{e}_l, B_{\Gamma_k} \mathbf{e}_k \right), \\ \widehat{w}_k &= \left(B_{\Gamma_k} \mathbf{e}_k, w \right), \\ \widehat{v}_k &= \left(D_\lambda^{-1} B_{\Gamma_k} \mathbf{e}_k, v \right), \\ \mathbf{e}_k &= \frac{1}{n_{\Gamma_k}} (1, \dots, 1)^T, \quad \|\mathbf{e}_k\|_{l_1} = 1, \\ k,\, l &= 1, \dots, m. \end{aligned}$$

The solution of (10) can be computed by $w = D_\lambda^{-1} \left(v - \sum_{k=1}^m \alpha_k n_{\Gamma_k} B_{\Gamma_k} \mathbf{e}_k \widehat{w}_k \right)$. The system $(I_m + T)\widehat{w} = \widehat{v}$ can be considered as a "coarse grid" problem, but here the nodes of the "coarse grid" correspond to the whole subdomains.

It is known [15,22], that the eigenvalues of the problem $\widehat{S}_\lambda w = \mu R_\lambda w$ belong to the segment $[1 - \delta_L,\, 1 + \delta_L]$, where

$$\delta_L = \frac{2q^L}{1 + q^{2L}}, \quad q = \frac{\sqrt{\nu} - 1}{\sqrt{\nu} + 1}, \quad \nu = \frac{\gamma_1 d}{\gamma_0 h}. \tag{11}$$

Simple calculations show that provided d is fixed, $L = O(h^{-1/2})$ preconditioned Chebyshev iterations are sufficient to guarantee that

$$R_\lambda \sim \widehat{S}_\lambda$$

and hence,

$$R_\lambda \sim S_\lambda.$$

To ensure the optimality of the preconditioner, we have to demand that both multiplication with $\widehat{S}_\lambda$ and inversion of $\widehat{R}_\lambda$ have the arithmetical complexity of the order of $O(h^{-5/2})$ at most. It is obviously the case for $\widehat{R}_\lambda$, which involves only computations on the interfaces. As for S_λ, we should have an inexpensive preconditioner $\widehat{H}_{\Gamma_k}$ for the boundary Schur complement matrix $\widehat{S}_{\Gamma_k}$. Construction of $\widehat{H}_{\Gamma_k}$ is based on the well known fact that corresponding blocks of spectrally equivalent matrices are also spectrally equivalent with at least the same spectrum bounds.

Let $\widehat{H}_k$ be a symmetric positive definite matrix, spectrally equivalent to the matrix $\widehat{A}_k^{-1}$ defined in (7). The matrices $\widehat{H}_k$ and $\widehat{A}_k^{-1}$ can be decomposed blockwise as in (5):

$$\widehat{H}_k = \begin{pmatrix} \widehat{H}_{I_k} & \widehat{H}_{I\Gamma_k} \\ \widehat{H}_{\Gamma I_k} & \widehat{H}_{\Gamma_k} \end{pmatrix}, \quad \widehat{A}_k^{-1} = \begin{pmatrix} * & * \\ * & \widehat{S}_{\Gamma_k}^{-1} \end{pmatrix},$$

where $\widehat{S}_{\Gamma_k} = \widehat{A}_{\Gamma_k} - \widehat{A}_{\Gamma I_k} \widehat{A}_{I_k}^{-1} \widehat{A}_{I\Gamma_k}$. Hence, $\widehat{H}_{\Gamma_k} \sim \widehat{S}_{\Gamma_k}^{-1}$. As soon as we have the preconditioner $\widehat{H}_k$, the matrix-vector multiplication procedure $v_\Gamma \to \widehat{H}_{\Gamma_k} v_\Gamma$ can be implemented easily, namely

$$\widehat{H}_{\Gamma_k} v_\Gamma = \widehat{Q} \widehat{H}_k \widehat{Q}^T v_\Gamma.$$

Here $\widehat{Q} = (\mathbf{0} \;\; I_{\Gamma_k})$ is a $n_k \times n_{\Gamma_k}$ matrix. Particularly, the choice of a multigrid preconditioner (BPX, multilevel diagonal scaling [12,19,27]) for $\widehat{H}_k$ proves to be very efficient. As it is shown in [19], the BPX preconditioning procedure includes three main components: restriction of a grid function from the finer level grid to the coarser, scaling (diagonal matrix multiplication) of the grid function on each level or direct solver on the coarsest level, and interpolation (prolongation) from coarser grid to finer. The result of the restriction procedure applied to the grid function with nonzero values only on the nodes on the boundary of a subdomain, is a (coarser) grid function with nonzero values also only in the boundary nodes. Similarly, if we need the grid function to be interpolated to the boundary grid nodes, then only the boundary node values of the coarser grid function will be used. Thus we see that only those components of the grid function are involved in preconditioning computations, which correspond to the grid nodes on the boundary of the subdomain or to the coarse grid nodes. Hence, the arithmetical complexity of the action of matrix $\widehat{H}_{\Gamma_k} v_\Gamma$ on a vector can be bounded from above by $const \cdot (d/h)^2 \ln(d/h)$ operations.

4. Remarks on the parallel implementation

The parallelization of the algorithm described above is based on two very simple principles. First, we merge the subdomains into clusters, each of them corresponding to a processor. Clearly, load balancing considerations motivate such a clusterization procedure. Indeed, in some regions of the computational domain the grid may be finer, but the decomposition of the original domain into subdomains may not follow the size of the grid cells.

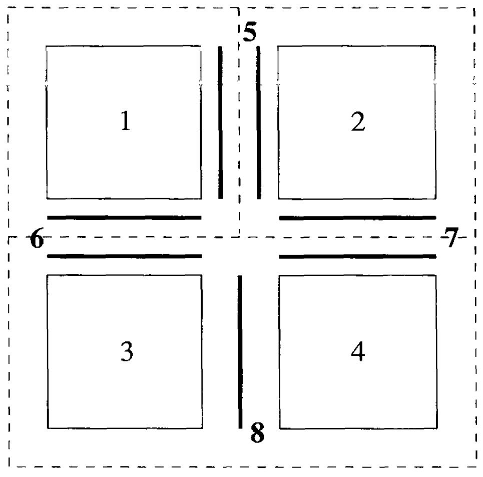

FIGURE 1. Domain decomposition.

It means that the number of the grid nodes can vary strongly in different subdomains. To ensure load balancing we merge the subdomains with the coarser subgrids, making thus the number of grid nodes in each cluster approximately equal. To illustrate the idea let us consider a decomposition of the unit cube into eight equal cubes. If subgrids are the same in all subdomains, then obviously the best choice is taking eight processors with one subdomain per processor. Suppose that the grid in one subdomain is uniformly refined to produce a twice finer mesh. Now there is no reason to use eight processors as the overall performance will be crucially limited by the computations in the subdomain with the finer grid. For the particular case considered above, it is sufficient to use only two processors, merging the seven subdomains with coarse grids into one cluster. Besides, clusterization turns to be very useful when the number of processors available is much less than the number of subdomains, which is very often the case in practice, especially when debugging on machines with only a few processors. For a given mesh, the load balancing/clustering process is achieved by a discrete optimization algorithm, namely a genetic algorithm [17, 20]. As shown in Section 5.1 the computing time increases linearly with the number of unknowns. Thus one possible discrete functional to minimize is the following one:

$$\frac{1}{N_{procs}} \sum_{i=0}^{N_{procs}} (S_i^2) - \left(\frac{\sum_{i=0}^{N_{procs}} S_i}{N_{procs}} \right)^2 \tag{12}$$

describing the deviation to grid size equirepartition, where N_{procs} is the number of processors and S_i is the grid size of the processor i.

Another feature of the parallel implementation considered here consists of duplicating the data at the interfaces between subdomains belonging to different clusters. If Γ_{kl} is such an interface, then the Lagrange multiplier vector λ_{kl} is stored in both the processors treating Ω_k and Ω_l. Although the data storage is increased a little, we gain a significant reduction of the communications.

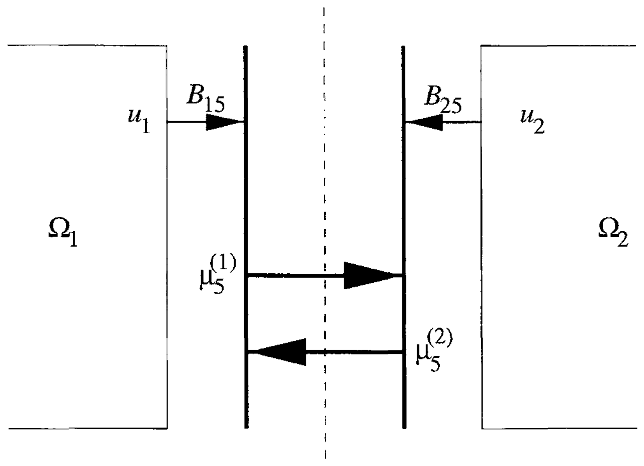

FIGURE 2. Communications for B matrix multiplication.

As an example, consider the splitting of a unit square into four little squares, as in Figure 1, where the dash rectangles denote clusters and the bold segments correspond to mortar interfaces. Note that when neighbour subdomains belong to different clusters, we have two copies of the mortar interface variables, stored in different clusters. We enumerate the interfaces as shown in the picture. The matrix $\mathcal{A}$ has the following form:

$$\mathcal{A} = \left(\begin{array}{cccc|cccc} A_1 & & & \mathbf{0} & B_{15}^T & B_{16}^T & \mathbf{0} & \mathbf{0} \\ & A_2 & & & B_{25}^T & \mathbf{0} & B_{27}^T & \mathbf{0} \\ & & A_3 & & \mathbf{0} & B_{36}^T & \mathbf{0} & B_{38}^T \\ \mathbf{0} & & & A_4 & \mathbf{0} & \mathbf{0} & B_{47}^T & B_{48}^T \\ \hline B_{15} & B_{25} & \mathbf{0} & \mathbf{0} & & & & \\ B_{16} & \mathbf{0} & B_{36} & \mathbf{0} & & & & \\ \mathbf{0} & B_{27} & \mathbf{0} & B_{47} & & & & \\ \mathbf{0} & \mathbf{0} & B_{38} & B_{48} & & & & \end{array}\right).$$

Let us consider the matrix-vector multiplication procedure with the matrix $\mathcal{A}$ and the vector $(\tilde{u}, \tilde{\lambda})$, where $\tilde{u}$ and $\tilde{\lambda}$ have the following componentwise representation, according to the enumeration in Figure 1: $\tilde{u} = (u_1^T, u_2^T, u_3^T, u_4^T)^T$, $\tilde{\lambda} = (\lambda_5^T, \lambda_6^T, \lambda_7^T, \lambda_8^T)^T$. The resulting vector $(\tilde{v}, \tilde{\mu}) = \mathcal{A} \cdot (\tilde{u}, \tilde{\lambda})$ can easily be computed as

$$\left(\begin{array}{c} v_1^{(1)} \\ v_2^{(2)} \\ v_3^{(3)} \\ v_4^{(3)} \\ \hline \mu_5^{(1,2)} \\ \mu_6^{(1,3)} \\ \mu_7^{(2,3)} \\ \mu_8^{(3)} \end{array}\right) = \left(\begin{array}{l} A_1 u_1^{(1)} + B_{15}^T \lambda_5^{(1)} + B_{16}^T \lambda_6^{(1)} \\ A_2 u_2^{(2)} + B_{25}^T \lambda_5^{(2)} + B_{27}^T \lambda_7^{(2)} \\ A_3 u_3^{(3)} + B_{36}^T \lambda_6^{(3)} + B_{38}^T \lambda_8^{(3)} \\ A_4 u_4^{(3)} + B_{47}^T \lambda_7^{(3)} + B_{48}^T \lambda_8^{(3)} \\ \hline B_{15} u_1^{(1)} + B_{25} u_2^{(2)} \\ B_{16} u_1^{(1)} + B_{36} u_3^{(3)} \\ B_{27} u_2^{(2)} + B_{47} u_4^{(3)} \\ B_{38} u_3^{(3)} + B_{48} u_4^{(3)} \end{array}\right), \tag{13}$$

where the upper indices denote the cluster (the processor), in which this variable is stored. Two upper indices mean that this variable is stored in both processors. We should keep in mind that $\lambda_i^{(k)} \equiv \lambda_i^{(l)}$. Note that so far

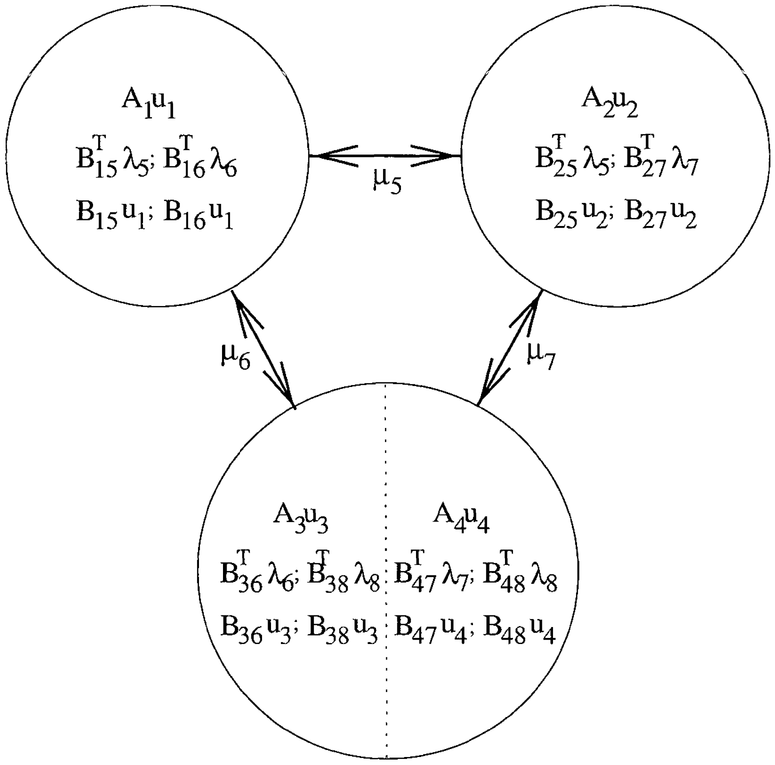
A_1u_1
$B^T_{15}\lambda_5; B^T_{16}\lambda_6$
$B_{15}u_1; B_{16}u_1$
μ_5
A_2u_2
$B^T_{25}\lambda_5; B^T_{27}\lambda_7$
$B_{25}u_2; B_{27}u_2$
μ_6
μ_7
A_3u_3
$B^T_{36}\lambda_6; B^T_{38}\lambda_8$
$B_{36}u_3; B_{38}u_3$
A_4u_4
$B^T_{47}\lambda_7; B^T_{48}\lambda_8$
$B_{47}u_4; B_{48}u_4$

FIGURE 3. Parallel matrix-vector multiplication.

we need communications only when computing μ_i. For example, we have

$$\mu_5 = \mu_5^{(1)} + \mu_5^{(2)}, \qquad \mu_5^{(1)} = B_{15}u_1, \quad \mu_5^{(2)} = B_{25}u_2.$$

We see that $\mu_5^{(1)}$ and $\mu_5^{(2)}$ are computed in parallel, and then should be interchanged and summed (Fig. 2). In Figure 3, the general scheme of the parallel matrix-vector multiplication (13) is shown.

For preconditioning, only the multiplication by the block R_λ requires communication. More precisely, communications occur when multiplying with the matrices B_{Γ_k}. As shown in Section 3, inverting of $\widehat{R}_\lambda$ reduces to solving a "coarse grid" linear system. Since the number of subdomains, which equals the size of the "coarse grid" system, is not too large in our experiments, we have chosen to store this *global* "coarse grid" matrix in each processor and solve the same *global* problem for each cluster, rather than storing this matrix in only one processor and broadcasting the solution afterwards.

It follows from the definitions of $\widehat{S}_\lambda$ and $\widehat{R}_\lambda$ in the previous section, that the amount of data to be communicated at each Chebyshev iteration is more than twice bigger than for the matrix-vector multiplication (13). Thus, the less Chebyshev iterations are executed in R_λ, the higher is the speed-up. Therefore, for minimizing the CPU time, the optimal number of Chebyshev iterations for the parallel implementation may be less than for the sequential one. This statement should be checked up numerically.

As for the parallel implementation, a message passing strategy has been chosen, using the message passing interface (MPI) library. The code is single program multiple data (SPMD) and we use asynchronous non blocking communications in order to compete with shared memory strategies such as SHMEM on the Cray T3E. The code is written in C++, and has been ported on several machines (Cray T3E, IBM SP2, HP 800 with 4 processors).

5. Numerical results

The goal of this section is to demonstrate the very good parallel properties of the mortar element method and the block-diagonal preconditioner (6) described in Section 3. Therefore we consider only uniform grids and decompositions into equal subdomains. The equation

$$-\Delta u + cu = f$$

with $c = 10^{-4}$ is solved in a parallelepiped. All the computations have been performed on the Cray T3E computer with up to 64 processors used. As the stopping criterion for the iterative method the reduction of the preconditioned residual is taken: $||\xi^{k_\varepsilon}||_{\mathcal{B}^{-1}} \leq \varepsilon||\xi^0||_{\mathcal{B}^{-1}}$, while the number of Chebyshev iterations is constant and equal to 8, except when mentioned explicitly. For all computations, the number of multigrid levels is equal to 4, except in Section 5.1.

Note that even with matched grids at subdomains' interfaces the solution is different from that of the single domain case, because of the mortar element discretization.

5.1. Computing time versus problem size

The unit cube is decomposed into 64 cubic subdomains. In each subdomain the grid is uniform and contains N nodes, N taking the value 25^3, 33^3, 41^3. The total number of nodes varies from 1 000 000 to 4 410 944. The table 1 displays the dependence of the elapsed CPU time and of the number of iterations on N. The desired accuracy is 10^{-7} and the number of Chebyshev iterations is 8 or 16. For these tests, the number of multigrid levels equals 3, and the number of processors is fixed at 64.

TABLE 1. Number of iterations and elapsed CPU time *vs.* the number of unknowns in the subdomains and the number of Chebyschev iterations, with 64 processors.

N		25^3	33^3	41^3
8 Cheb. it.	#iter	82	82	82
	T_{cpu}	39	81	141
16 Cheb. it.	#iter	66	68	68
	T_{cpu}	48	97	165

The CPU time varies slightly sublinearly with the number of unknowns. This can be explained by cache effects when the size of the problem is increased.

5.2. Computing time versus stopping criterion and number of processors

In Table 2 the elapsed CPU time (in seconds) *versus* the stopping criterion ε and the number of processors is shown.

We wish to estimate the speed-up of the method, *i.e.* the dependence of the elapsed CPU time with the number of processors, the global mesh size of the problem and the number of subdomains being fixed. The total number of grid nodes is equal to $129 \times 129 \times 129 = 2\,146\,689$, and the number of subdomains is $4 \times 4 \times 4 = 64$. The subdomains are grouped into 16, 32 or 64 clusters ($N_c = N_c^x \times N_c^y \times N_c^z$), so that $16 = 4 \times 2 \times 2$ (4 subdomains per cluster), $32 = 4 \times 4 \times 2$ (2 subdomains per cluster), $64 = 4 \times 4 \times 4$ (1 subdomains per cluster). In Table 2 the elapsed CPU time (in seconds) *versus* the stopping criterion ε and the number of processors is shown. In the last column we give the Euclidean norm of Bu^{k_ε}, which is nothing but the jump of the computed solution on the interfaces.

The actual speed-up, given in parentheses, is very close to the ideal, demonstrating thus very good parallel properties of the method. The speed-up is estimated with respect to the 16-processors case.

TABLE 2. CPU time (speed-up) *vs.* the number of processors and stopping criterion.

Number of processors	16 4 sd./pr.	32 2 sd./pr.	64 1 sd./pr.	Number of iterations	$\|\|Bu^{k_\varepsilon}\|\|$
$\varepsilon = 10^{-5}$	89(1)	45(1.97)	23(3.86)	24	1.5e-5
$\varepsilon = 10^{-6}$	247(1)	124(1.99)	64(3.85)	65	1.1e-6
$\varepsilon = 10^{-7}$	351(1)	177(1.98)	91(3.85)	92	6.5e-8

5.3. Scalability

The next series of results (Tab. 3) prove the excellent scalability of the algorithm. Now each subdomain is a cube with the edge length 0.5 and has a grid composed of $33 \times 33 \times 33$ nodes. The number of processors used for computation increases linearly with the number of subdomains, so that there is always one subdomain per processor. As we can see from the Table 3, the convergence rate is almost independent on the number of subdomains, although the boundary value problem is not the same, provided the grid in each subdomain does not change, and the CPU time increases less than 19%, when the number of processors goes from 16 to 64. This increase of CPU time can be explained by the fact that the average number of mortar sides per subdomain increases with the number of subdomains.

TABLE 3. Number of iterations and CPU time *vs.* the number of processors and stopping criterion for cubic subdomains.

N_{sd}		16	32	64
$\varepsilon = 10^{-5}$	#iter	27	29	29
	T_{cpu}	23	27	28
$\varepsilon = 10^{-6}$	#iter	43	45	46
	T_{cpu}	37	42	45
$\varepsilon = 10^{-7}$	#iter	67	69	70
	T_{cpu}	58	63	69

5.4. Influence of the shape of the subdomains

The series of results, shown in Table 4, concerns the behavior of the algorithm when the unit cube is divided into 16, 32, 64 rectangular parallepipeds respectively similar to $[0,0.5]\times[0,1]\times[0,1]$, resp. $[0,0.5]\times[0,0.5]\times[0,1]$, resp. $[0,0.5]\times[0,0.5]\times[0,0.5]$. As in the previous tests, each processor corresponds to a single subdomain, and the number of unknowns per subdomain is $33 \times 33 \times 33$. The difference with the previous series of tests lies in the fact that the subdomains are no longer cubic. It is clear that the performances deteriorate when the aspect ratio of the subdomains increases. This problem should be solved in a near future by replacing the multilevel preconditioner by a fast direct method.

5.5. Nonmatching uniform grids

In this series of tests, we focus on the effect of nonmatching grids on the performances of the solver. The unit cube is divided into $4 \times 4 \times 4$ subdomains. We compare two cases: in the first case, all the subdomains have a grid with $33 \times 33 \times 33$ nodes, so the grids are matched at the interfaces. In the second case, only the subdomains

TABLE 4. Number of iterations and CPU time *vs.* the number of processors and stopping criterion for stretched subdomains.

N_{sd}		16	32	64
shape of a subdomain		$0.5 \times 1 \times 1$	$0.5 \times 0.5 \times 1$	$1 \times 1 \times 1$
$\varepsilon = 10^{-5}$	#iter	40	40	24
	T_{cpu}	34	37	23
$\varepsilon = 10^{-6}$	#iter	92	108	65
	T_{cpu}	80	102	64
$\varepsilon = 10^{-7}$	#iter	142	159	92
	T_{cpu}	123	149	91

TABLE 5. Effect of nonmatching grids.

	case 1	case 2
#iter	82	90
T_{cpu}	81	87

located in the half space $x_3 > 0.5$ have $33 \times 33 \times 33$ nodes while the other subdomains have $25 \times 25 \times 25$ nodes. For these tests, the number of multigrid levels equals 3, and the number of Chebyshev iterations is 8.

The performances of the solver are slightly affected by the presence of nonmatching grids. In this example, the load balancing is very bad, so the CPU time is governed by the processors taking care of the finest grids.

6. CONCLUSIONS

For finite elements, the mortar method permits to use independent meshes in different subdomains. For solving the linear systems arising from this technique, parallel algorithm have to be developed.

It has turned out that the macro-hybrid formulation of the mortar method, leading to a saddle point algebraic problem, and the block-diagonal preconditioner are particularly well suited for parallel computations. The convergence rate of the proposed algorithm does neither depend on the grid size nor on the size of the subdomains, and the experiments have demonstrated very nice scalability properties. Moreover, the constructed iterative method is algebraically optimal, *i.e.* the total amount of work to reduce the residual by a given factor is proportional to the number of unknowns, which has been confirmed by the numerical experiments.

However, there is still a number of unsolved problems. Though the methods seems to work very well for low aspect ratio domains and uniform grids, in more general cases, and in particular for distorted domains and highly nonuniform grids, the convergence slows down. This drawback can probably be overcome by introducing a subdomain preconditioner taking advantage of a fast direct method instead of the multigrid method proposed above. Another way of improvement consists of implementing a non iterative preconditioner for the mortar variables, also based on a fast direct solver in subdomains. With this approach, the method would be less dependent upon the shape of the subdomains and the quality of the grid, but not necessarily algebraically optimal.

This work has been done at the ASCI laboratory, in Orsay. We would like to thank P. Lallemand and O. Pironneau for their support. We have used the CRAY T3E of the IDRIS institute for all computations. The work of the first author has been partially supported by the A.M. Liapunov French-Russian institute on informatics and applied mathematics (Moscow).

References

[1] G.S. Abdoulaev, Yu.A. Kuznetsov and O. Pironneau, The Numerical Implementation of the Domain Decomposition Method with Mortar Finite Elements for a 3D Problem, in *Proc. of the 1st European Conf. on Numer. Math. and Advanced Appl. (Paris, September 1995). SMAI/GAMNI* (to appear).

[2] G.S. Abdoulaev, Y. Achdou, Yu.A. Kuznetsov, K.N. Lipnikov, J. Periaux and O. Pironneau, Finite element methods with nonmatching grids and applications, in *Proc. of the Conf. on Applied Math. and Computer Science. Moscow University/INRIA* (1997).

[3] Y. Achdou and Yu.A. Kuznetsov, Substructuring preconditioners for finite element methods on nonmatching grids. *East-West J. Numer. Math.* **3** (1995) 1–28.

[4] Y. Achdou, Yu.A. Kuznetsov and O. Pironneau, Substructuring preconditioners for Q_1 mortar element method. *Numer. Math.* **79** (1995) 419–449.

[5] Y. Achdou, Y. Maday and O.B. Widlund, Méthodes itératives de sous-structuration pour les éléments avec joints. *Note CRAS, Paris, série I* **t 322** (1996) 185–190.

[6] Y. Achdou, Y. Maday and O.B. Widlund, Iterative substructuring preconditioners for mortar element methods in two dimensions. *SIAM J. Numer. Anal.* **36** (1999) 551–580.

[7] Y. Achdou and O. Pironneau, A fast solver for Navier–Stokes equations in the laminar regime using mortar finite element and boundary element methods. *SIAM J. Numer. Anal.* **32** (1995) 985–1016.

[8] F. Ben Belgacem, Discrétisations 3D nonconformes par la méthode de décomposition de domaines des éléments avec joints: analyse mathématique et mise en œuvre pour le problème de Poisson. Ph.D. thesis, University of Pierre et Marie Curie, France. *Note technique EDF, ref. HI72/93017* (1993).

[9] F. Ben Belgacem, The mortar finite element method with Lagrange multipliers. *Rapport interne MIP n° $\tilde{9}$4-1, University of Paul Sabatier, France* (1994).

[10] F. Ben Belgacem and Y. Maday, The mortar element method for three dimensional finite elements. *Rapport interne MIP n° $\tilde{9}$4-19, University of Paul Sabatier* (1994).

[11] C. Bernardi, Y. Maday and A. Patera, A new nonconforming approach to domain decomposition: the mortar element method, in *Nonlinear Partial Differential Equations and Their Applications.* H. Brezis, J.L. Lions Eds., Pitman, New York (1993).

[12] J. Bramble, J. Pasciak and J. Xu, Parallel multilevel preconditioners. *Math. Comp.* **31** (1990) 333–390.

[13] M.A. Casarin and O.B. Widlund, A hierarchical preconditioner for the mortar finite element method. *ETNA* **4** (1996) 75–88.

[14] P. Ciarlet, *The Finite Element Method for Elliptic Problems.* North-Holland, Amsterdam (1978).

[15] E.G. D'yakonov, *Minimization of Computations: Asymptotically Optimal Algorithms* (in Russian). Nauka, Moscow (1989).

[16] C. Farhat and F.X. Roux, Implicit parallel processing in structural mechanics. *Comput. Mech. Adv.* **2** (1994) 1–124.

[17] D.E. Goldberg, *Genetic algorithms in search, optimization and machine learning.* Addison Wesley (1989).

[18] G.H. Golub and C.F. Van Loan, *Matrix Computations.* The Johns Hopkins University Press, Baltimore and London (1983).

[19] M. Griebel, Multilevel algorithms considered as iterative methods on semidefinite systems. *SIAM J. Sci. Comput.* **15** (1994) 547–565.

[20] J.H. Holland, *Adaptation in natural and artificial systems.* University of Michigan Press, Ann Arbor (1975).

[21] Y. Iliash, T. Rossi and J. Toivanen, Two iterative methods for solving the Stokes problem. *Tech. Report 2, University of Jyväskylä, Department of Mathematics, Laboratory of Scientific Computing* (1993).

[22] Yu.A. Kuznetsov, Multigrid domain decomposition methods for elliptic problems. *Comput. Methods Appl. Mech. Eng.* **75** (1989) 185–193.

[23] Yu.A. Kuznetsov, Efficient iterative solvers for elliptic finite element problems on nonmatching grids. *Russ. J. Numer. Anal. Math. Modelling* **10** (1995) 187–211.

[24] P. Le Tallec, T. Sassi and M. Vidrascu, Domain decomposition method with nonmatching grids. *Proceedings of DDM 9* AMS (1994) 61–74.

[25] G.I. Marchuk and Yu.A. Kuznetsov, Méthodes itératives et fonctionnelles quadratiques, in *Méthodes Mathématiques de L'Informatique – 4: Sur les Méthodes Numériques en Sciences, Physiques et Economiques.* J.L. Lions, G.I. Marchuk Eds., Dunod, Paris (1974).

[26] R.S. Varga, *Matrix Iterative Analysis.* Prentice Hall, Englewood Cliffs (1961).

[27] X. Zhang, Multilevel Schwarz methods. *Numer. Math.* **63** (1992) 521–539.