Journal of Computational and Applied Mathematics 34 (1991) 93–117
North-Holland

Domain decomposition methods for large linearly elliptic three-dimensional problems

P. Le Tallec
CEREMADE, Université de Paris-Dauphine, Place du Maréchal de Lattre de Tassigny, 75775 Paris Cedex 16, France

Y.H. De Roeck
CERFACS, 42 avenue Gustave Coriolis, 31057 Toulouse Cedex, France

M. Vidrascu
INRIA, Domaine de Voluceau, 78153 Le Chesnay Cedex, France

Received 5 April 1990
Revised 9 July 1990

Abstract

Le Tallec, P., Y.H. De Roeck and M. Vidrascu, Domain decomposition methods for large linearly elliptic three-dimensional problems, Journal of Computational and Applied Mathematics 34 (1991) 93–117.

The idea of solving large problems using domain decomposition techniques appears particularly attractive on present day large-scale parallel computers. But the performance of such techniques used on a parallel computer depends on both the numerical efficiency of the proposed algorithm and the efficiency of its parallel implementation.

The approach proposed herein splits the computational domain in unstructured subdomains of arbitrary shape, and solves for unknowns on the interface using the associated trace operator (the Steklov–Poincaré operator on the continuous level or the Schur complement matrix after a finite element discretization) and a preconditioned conjugate gradient method. This algorithm involves the solution of Dirichlet and of Neumann problems, defined on each subdomain and which can be solved in parallel.

This method has been implemented on a CRAY 2 computer using multitasking and on an INTEL hypercube. It was tested on a large scale, industrial, ill-conditioned, three-dimensional linear elasticity problem, which gives a fair indication of its performance in a real life environment. In such situations, the proposed method appears operational and competitive on both machines: compared to standard techniques, it yields faster results with far less memory requirements.

Résumé

La résolution numérique de problèmes de grande taille par des techniques de décomposition de domaines est très bien adaptée aux ordinateurs parallèles de la génération actuelle. Cependant, l'efficacité de ces techniques dépend fortement de l'algorithme choisi et de son implémentation.

L'approche proposée ici partage le domaine de calcul en sous-domaines non structurés de forme arbitraire et réduit le problème initial à un problème d'interface. L'opérateur associé (l'opérateur de Steklov–Poincaré au

niveau continu, la matrice complément de Schur au niveau discret) est ensuite inversé par un algorithme de gradient conjugué préconditionné. Cet algorithme exige à chaque étape la résolution en parallèle sur chaque sous-domaine d'un problème de Dirichlet et d'un problème de Neumann.
Cette méthode a été implémentée sur CRAY 2 et sur un hypercube INTEL. Elle a été étudiée sur un problème industriel d'élasticité linéaire tridimensionnel de grande taille. Sur cet exemple significatif, la méthode proposée est compétitive à la fois au niveau du temps calcul et de la place mémoire.

Keywords: Domain decomposition, Schur complement, conjugate gradient, linear elasticity, CRAY 2 and hypercube.

Mots clés: Décomposition de domaines, complément de Schur, gradient conjugué, élasticité linéaire, CRAY 2 et hypercube.

1. Introduction

The idea of solving large problems using domain decomposition techniques appears particularly attractive on present day large-scale parallel computers. But the performance of such techniques used on a parallel computer depends on both the numerical efficiency of the proposed algorithm and the efficiency of its parallel implementation.

The approach proposed herein splits the computational domain in unstructured subdomains of arbitrary shape, and solves for unknowns on the interface using the associated trace operator (the Steklov–Poincaré operator on the continuous level or the Schur complement matrix after a finite element discretization) and a preconditioned conjugate gradient method. This algorithm involves the solution of Dirichlet and of Neumann problems, defined on each subdomain and which can be solved in parallel.

This method has been implemented on a CRAY 2 computer using multitasking and on an INTEL hypercube. It was tested on a large scale, industrial, ill-conditioned, three-dimensional linear elasticity problem, which gives a fair indication of its performance in a real life environment. In such situations, the proposed method appears operational and competitive on both machines: compared to standard techniques, it yields faster results with far less memory requirements.

2. A simplified model problem

We first describe our approach on the following model problem:

$$-\Delta u = f \quad \text{on } \Omega, \qquad u = 0 \quad \text{on } \partial\Omega,$$

the domain Ω being decomposed as indicated in Fig. 1.

Our goal is to solve the above problem only on the subdomains Ω_i. If we knew the value λ of the solution u on the interface S, then the parallel solution of

$$\begin{aligned} -\Delta u_i &= f && \text{on } \Omega_i, \\ u_i &= 0 && \text{on } \partial\Omega \cap \partial\Omega_i, \\ u_i &= \lambda && \text{on } S, \end{aligned}$$

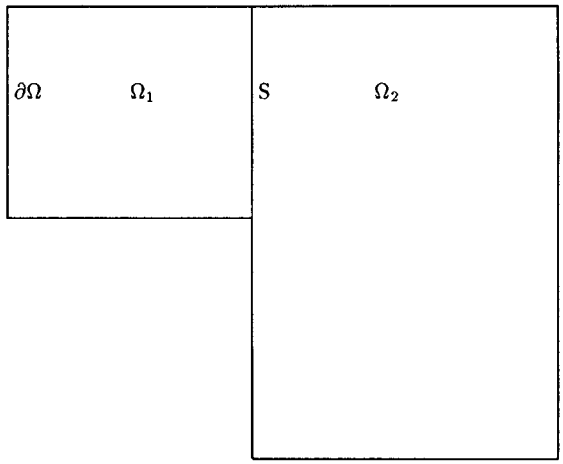

Fig. 1. The model problem.

would give the value u_i of u on each subdomain Ω_i and these values would satisfy the following compatibility condition (continuity of the normal derivative across S)

$$\frac{\partial u_1}{\partial n_1} + \frac{\partial u_2}{\partial n_2} = 0. \tag{1}$$

Our problem is thus reduced to the solution of (1) with unknown $\lambda = u|_S$.

To solve this problem, we introduce the function $u_i(\lambda, f)$ defined by

$$\begin{aligned} -\Delta u_i(\lambda, f) &= f \quad \text{on } \Omega_i, \\ u_i(\lambda, f) &= 0 \quad \text{on } \partial\Omega \cap \partial\Omega_i, \\ u_i(\lambda, f) &= \lambda \quad \text{on } S, \end{aligned}$$

the Steklov–Poincaré operator S_i given by

$$S_i\lambda = \left.\frac{\partial u_i(\lambda, 0)}{\partial n_i}\right|_s,$$

and the right-hand side b given by

$$b = -\frac{\partial u_1(0, f)}{\partial n_1} - \frac{\partial u_2(0, f)}{\partial n_2}.$$

With this notation, the compatibility condition (1) becomes

$$(S_1 + S_2)\lambda = b,$$

and can be solved by the following preconditioned gradient algorithm

$$\lambda^{n+1} = \lambda^n - \rho M\big((S_1 + S_2)\lambda^n - b\big), \tag{2}$$

where M is a preconditioning operator and ρ is a relaxation parameter.

The key point is then to choose an efficient and cheap preconditioner. Many choices have

been proposed in the literature [5,8,12,15]. The one we have picked has been discussed in [1,9,13], corresponds to the simple choice

$$M = \tfrac{1}{4}\left(S_1^{-1} + S_2^{-1}\right),$$

and will be exact in the symmetric case with $S_1 = S_2$.

With this choice, our domain decomposition algorithm becomes

$$\lambda^{n+1} = \lambda^n - \tfrac{1}{4}\rho\left(S_1^{-1} + S_2^{-1}\right)(S_1 + S_2)\lambda^n,$$

that is, by definition of S_i,

data: λ given in $H_{00}^{1/2}(S)$;
computation of the solution: for any i, solve in parallel the Dirichlet problems:

$$\begin{aligned} -\Delta u_i &= f && \text{on } \Omega_i, \\ u_i &= 0 && \text{on } \partial\Omega \cap \partial\Omega_i, \\ u_i &= \lambda && \text{on } S; \end{aligned}$$

computation of the gradient: for any i, solve in parallel the Neumann problems (preconditioner):

$$\begin{aligned} -\Delta\phi_i &= 0 && \text{on } \Omega_i, \\ \phi_i &= 0 && \text{on } \partial\Omega \cap \partial\Omega_i, \\ \frac{\partial\phi_i}{\partial n_i} &= \frac{1}{2}\left(\frac{\partial u_1}{\partial n_1} + \frac{\partial u_2}{\partial n_2}\right) && \text{on } S; \end{aligned}$$

updating: set $\lambda = \lambda - \tfrac{1}{2}\rho(\phi_1 + \phi_2)$ and iterate until convergence.

The domain decomposition method that we will now introduce is simply a generalization of this algorithm.

3. The original problem

Consider the domain Ω, partitioned into subdomains Ω_i as indicated in Fig. 2.

Let us introduce the boundaries (see Fig. 2)

$$\begin{aligned} &\partial\Omega = \partial\Omega_N \cup \partial\Omega_D, \\ &\Gamma_i = \partial\Omega_D \cap \partial\Omega_i, \\ &S_i = \partial\Omega_i - \Gamma_i - \text{interior}(\partial\Omega_N \cap \partial\Omega_i), \end{aligned}$$

together with the spaces

$$\begin{aligned} V &= \left\{ v \in H^1(\Omega;\ R^p):\ v = 0 \text{ on } \partial\Omega_D \right\}, \\ V_i &= \left\{ v \in H^1(\Omega_i;\ R^p):\ v = 0 \text{ on } \Gamma_i \right\}, \\ V_{0i} &= \left\{ v \in H^1(\Omega_i;\ R^p):\ v = 0 \text{ on } \Gamma_i \cup S_i \right\}. \end{aligned}$$

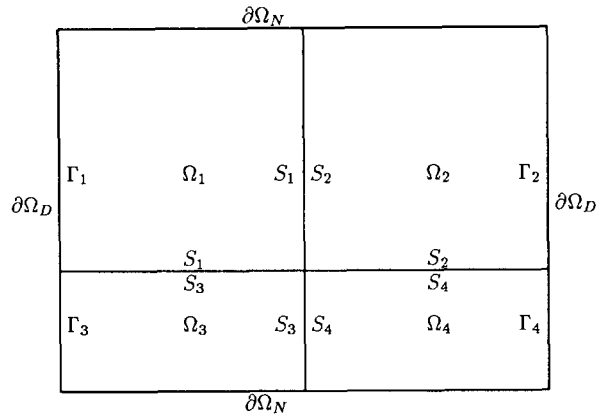

Fig. 2. Definition of the subdomains and of the boundaries.

Moreover, Y will represent the space of traces on S of functions of V, and $\mathrm{Tr}_i^{-1}(\lambda)$ will represent any element z of V_i whose trace on S_i is equal to λ. Finally, we introduce the elliptic form

$$a_i(u, v) = \int_{\Omega_i} A_{mnkl}(x) \frac{\partial u_m}{\partial x_n} \frac{\partial v_k}{\partial x_l},$$

with $A \in L^\infty(\Omega)$ symmetric and satisfying the strong ellipticity condition

$$A_{mnkl}(x) F_{mn} F_{kl} \geqslant c_0 |F|^2, \quad \forall F \in \mathbb{R}^{p \times N}.$$

Such an assumption is typically satisfied in linear elasticity where we have

$$a_i(u, v) = \int_{\Omega_i} A\varepsilon(u) : \varepsilon(v),$$

with A the elasticity tensor and $\varepsilon(u)$ the linearized strain tensor

$$\varepsilon(u) = \tfrac{1}{2}\big(\nabla u + (\nabla u)^t\big).$$

With this notation, the problem becomes

$$\text{Find } u \in V \text{ such that } \sum_i a_i(u, v) = \langle f, v \rangle, \quad \forall v \in V. \tag{3}$$

4. Decomposition of the original problem

4.1. Transformation

With $\lambda \in Y$ we now associate $z_i(\lambda, f)$ as the solution of

$$a_i(z_i, v) = \langle f, v \rangle, \quad \forall v \in V_{0i},$$
$$z_i \in V_i, \qquad z_i = \lambda \quad \text{on } S_i,$$

the Steklov operator S_i given by

$$\langle S_i\lambda, \mu\rangle = a_i\big(z_i(\lambda, 0), \mathrm{Tr}_i^{-1}\mu\big), \quad \forall\mu \in Y,$$

and the right-hand side L given by

$$\langle L, \mu\rangle = -\sum_i a_i\big(z_i(0, f), \mathrm{Tr}_i^{-1}\mu\big), \quad \forall\mu \in Y.$$

Observe that in computing $a_i(z_i(\lambda, 0), \mathrm{Tr}_i^{-1}\mu)$, the choice of the representative element of Tr_i^{-1} is of no importance since, by construction, $z_i(\lambda, 0)$ is orthogonal to any component of this element in $\mathrm{Ker}(\mathrm{Tr}_i)$.

With this new notation, our initial variational problem finally reduces to the interface problem

$$\Big(\sum_j S_j\Big)\lambda = L \quad \text{in } Y^*. \tag{4}$$

4.2. Preconditioner

We first define a trace operator $\alpha_i\mathrm{Tr}$ from V_i into Y satisfying

$$\sum_i \alpha_i\,\mathrm{Tr}(v) = \mathrm{Tr}(v), \quad \forall v \in V. \tag{5}$$

For example, at the continuous level, we can often set $\alpha_i = \frac{1}{2}$. A different definition should be used at the finite element level in order that condition (5) is still satisfied after discretization. A possible choice will be described in Section 5.

Generalizing our first choice, we now propose as preconditioner the operator

$$M = \sum_i (\alpha_i\,\mathrm{Tr})S_i^{-1}(\alpha_i\,\mathrm{Tr})^{\mathrm{t}}.$$

By definition of S_i, the action of the preconditioner M on L is then given by

$$ML = \sum_i \alpha_i\,\mathrm{Tr}(\psi_i)$$

with ψ_i the solution of

$$a_i(\psi_i, v) = L(\alpha_i\,\mathrm{Tr}\,v), \quad \forall v \in V_i,\ \psi_i \in V_i. \tag{6}$$

Remark. In the absence of Dirichlet boundary conditions in the definition of V_i, problem (6) is not well posed. In such situations, we replace α_i in (6) by an equivalent symmetric bilinear form $\tilde{a}_i$ which we take positive definite on V_i and such that

$$\sum_i \tilde{a}_i(z, z) \geqslant \sum_i a_i(z, z) \geqslant c_1 \sum_i \tilde{a}_i(z, z), \quad \forall z \in V.$$

For example, on the discrete level, the bilinear form $\tilde{a}_i$ is simply obtained by replacing in the factorization of the finite element matrix of a_i all the singular pivots by an averaged strictly positive pivot.

4.3. Conjugate gradient algorithm

The solution of the interface problem (4) by a standard preconditioned conjugate gradient method, with preconditioner M, now leads to the following algorithm:

Conjugate gradient iteration on interface operator

(i) *Computation of* $MS\lambda$

λ_n given on S (descent direction)

$\Rightarrow$ On each subdomain solve in parallel

$$a_i(z_i, v) = 0, \quad \forall v \in V_{0i}, \quad z_i = \lambda_n \text{ on } S, \quad z_i \in V_i.$$

$\Rightarrow$ Set $L(\mu) = \sum_j a_j(z_j, \mathrm{Tr}_j^{-1}\mu)$.

$\Rightarrow$ On each subdomain solve in parallel

$$a_i(\psi_i, v) = L(\alpha_i \,\mathrm{Tr}\, v), \quad \forall v \in V_i, \quad \psi_i \in V_i.$$

$\Rightarrow$ Set $\psi = \sum_i \alpha_i \,\mathrm{Tr}\, \psi_i = MS\lambda_n$.

(ii) *Update*

$$\begin{aligned}
&\rho_n = d_n / L(\lambda_n) && \text{(interface)},\\
&u_{n+1} = u_n - \rho_n z && \text{(parallel update)},\\
&\varphi_{n+1} = \varphi_n - \rho_n \psi && \text{(interface)},\\
&R_{n+1} = R_n - \rho_n L && \text{(interface)}.
\end{aligned}$$

(iii) *New descent direction*

$$\begin{aligned}
&d_{n+1} = R_{n+1}(\varphi_{n+1}) && \text{(interface)},\\
&\lambda_{n+1} = \varphi_{n+1} + (d_{n+1}/d_n)\lambda_n && \text{(interface)}.
\end{aligned}$$

A reorthogonalization of the descent directions is strongly advised here and leads to a slight change in the update formula of λ_n (see Section 6 for more details).

4.4. Convergence analysis

To study the convergence of the above conjugate gradient algorithm, we have to prove the spectral boundedness of $M\sum_i S_i$, that is to exhibit two positive constants k and K such that

$$k((\lambda, \lambda)) \leqslant ((MS\lambda, \lambda)) \leqslant K((\lambda, \lambda)),$$

with $((\cdot, \cdot))$ a given scalar product on Y and $S = \sum_i S_i$. For this purpose, we endow Y with the scalar product

$$((\lambda, \lambda')) = \langle S\lambda, \lambda' \rangle.$$

Under the notation and assumptions of Section 3, this is indeed a scalar product, because by construction we have

$$\langle S\lambda, \lambda' \rangle = \sum_j a_j\big(z_j(\lambda, 0), \mathrm{Tr}_j^{-1}(\lambda')\big) = \sum_j a_j\big(z_j(\lambda, 0), z_j(\lambda', 0)\big).$$

With this scalar product, we have, under the notation of Section 4.3,

$$\begin{aligned}((MS\lambda, \lambda)) &= \langle SMS\lambda, \lambda\rangle = \langle MS\lambda, S\lambda\rangle \\ &= \sum_j a_j\big(z_j(\lambda, 0), \mathrm{Tr}_j^{-1}(MS\lambda)\big) \\ &= \sum_j a_j\Big(z_j(\lambda, 0), \mathrm{Tr}_j^{-1}\Big(\sum_i \alpha_i \mathrm{Tr}\ \psi_i\Big)\Big) \\ &= \sum_{ij} a_j\big(z_j(\lambda, 0), \mathrm{Tr}_j^{-1}(\alpha_i \mathrm{Tr}\ \psi_i)\big) \\ &= \sum_i L(\alpha_i \mathrm{Tr}\ \psi_i) = \sum_i \tilde{a}_i(\psi_i, \psi_i).\end{aligned}$$

Therefore, the whole convergence analysis reduces to the verification of the inequality

$$k \| z(\lambda, 0)\|^2 \leqslant \|\psi\|^2 \leqslant K \| z(\lambda, 0)\|^2,$$

under the notation

$$\|\varphi\| = \Big(\sum_i \tilde{a}_i(\varphi_i, \varphi_i)\Big)^{1/2}.$$

Theorem 1. *We have*

$$c_1^2 \| z(\lambda, 0)\|^2 \leqslant \|\psi\|^2 \leqslant C^2 \| z(\lambda, 0)\|^2,$$

with

$$C = \sup_{\varphi \in \Pi V_i} \frac{\| z\big(\sum \alpha_i \mathrm{Tr}\ \varphi_i, 0\big)\|}{\|\varphi\|}.$$

Remark. In the case of no internal cross-points and if we choose smooth weights α_i, the constant C is well defined. Furthermore, in this case it does not depend on the discretization step h when the spaces V_i are replaced by conforming finite element spaces V_{ih} provided that we have

$$\mathrm{Tr}\ V_{ih}|_{\partial\Omega_i \cap \partial\Omega_j} = \mathrm{Tr}\ V_{jh}|_{\partial\Omega_i \cap \partial\Omega_j}.$$

Indeed, in this case z is the harmonic extension of the function μ defined in $\Pi_{ij}\ \mathrm{Tr}\ V_i \cap \mathrm{Tr}\ V_j$ by

$$\mu|_{\partial\Omega_i \cap \partial\Omega_j} = \alpha_i \mathrm{Tr}\ \psi_i + \alpha_j \mathrm{Tr}\ \psi_j,$$

and this obviously depends continuously on ψ_i and ψ_j. In the case of internal cross points, one can prove, following [5], that we have

$$C \leqslant C_2 \log\Big(\frac{d}{h}\Big).$$

A detailed analysis of this point will be given in a forthcoming paper.

Proof of Theorem 1. From the Cauchy–Schwarz inequality, we first have by construction,

$$\begin{aligned}
\| \psi \| \, \| z(\lambda, 0) \| &= \Big(\sum_i \tilde{a}_i(\psi_i, \psi_i) \Big)^{1/2} \Big(\sum_j \tilde{a}_j\big(z_j(\lambda, 0), z_j(\lambda, 0)\big) \Big)^{1/2} \\
&\geqslant \sum_i \tilde{a}_i(\psi_i, z_i(\lambda, 0)) \geqslant \sum_i L(\alpha_i \operatorname{Tr} z_i(\lambda, 0)) \\
&\geqslant \sum_i \sum_j a_j\big(z_j(\lambda, 0), \operatorname{Tr}_j^{-1}(\alpha_i \operatorname{Tr} z_i(\lambda, 0))\big) \\
&\geqslant \sum_j a_j\Big(z_j(\lambda, 0), \operatorname{Tr}_j^{-1}\Big(\sum_i \alpha_i \operatorname{Tr} z_i(\lambda, 0)\Big)\Big) \\
&\geqslant \sum_j a_j\big(z_j(\lambda, 0), \operatorname{Tr}_j^{-1}(\lambda)\big) \\
&\geqslant \sum_j a_j\big(z_j(\lambda, 0), z_j(\lambda, 0)\big) \geqslant c_1 \| z(\lambda, 0) \|^2.
\end{aligned}$$

On the other hand, we have

$$\begin{aligned}
\sum_i \tilde{a}_i(\psi_i, \psi_i) &= \sum_i L(\alpha_i \operatorname{Tr} \psi_i) \\
&= \sum_i \sum_j a_j\big(z_j(\lambda, 0), \operatorname{Tr}_j^{-1}(\alpha_i \operatorname{Tr} \psi_i)\big) \\
&= \sum_j a_j\Big(z_j(\lambda, 0), \operatorname{Tr}_j^{-1}\Big(\sum_i \alpha_i \operatorname{Tr} \psi_i\Big)\Big) \\
&= \sum_j a_j\Big(z_j(\lambda, 0), z_j\Big(\sum_i \alpha_i \operatorname{Tr} \psi_i, 0\Big)\Big) \\
&\leqslant \| z(\lambda, 0) \| \, \Big\| z\Big(\sum_i \alpha_i \operatorname{Tr} \psi_i, 0\Big) \Big\| \\
&\leqslant C \| z(\lambda, 0) \| \, \| \psi \|.
\end{aligned}$$

By combining the two inequalities, we finally obtain

$$c_1^2 \| z(\lambda, 0) \|^2 \leqslant \| \psi \|^2 \leqslant C^2 \| z(\lambda, 0) \|^2,$$

which is the desired result. □

Theorem 2. *Our preconditioned conjugate gradient algorithm converges at least linearly with asymptotic constant* $(C/c_1 - 1)/(C/c_1 + 1)$.

Proof. From Theorem 1, we have seen that the spectrum of the operator MS was bounded above by C^2 and bounded below by c_1^2. Hence, the condition number of MS is bounded above by $(C/c_1)^2$ which ensures that the associated preconditioned conjugate gradient algorithm converges as announced (see [10] for more details). □

Actually, this convergence result is rather conservative. Indeed, our numerical tests have indicated that the larger eigenvalues of MS are well separated, which accelerates the convergence of the conjugate gradient algorithm.

5. Numerical results

5.1. Implementation

We first describe the implementation on a four-processors CRAY 2.

The following numerical results deal either with the Laplace operator for which we have

$$a_i(u, v) = \int_{\Omega_i} \beta_i \nabla u \cdot \nabla v,$$

or with three-dimensional anisotropic heterogeneous linear elasticity operators.

The spaces V_i are approximated by finite elements of $P1$-Lagrange type (2-D case) or by isoparametric reduced $Q2$-Lagrange elements (2-D case). The trace operators α_i are defined at each node of the interface by the formula

$$(\alpha_i \operatorname{Tr}(v))(M_k) = \frac{a_i(\varphi_k, \varphi_k)}{\sum_j a_j(\varphi_k, \varphi_k)} v(M_k),$$

where φ_k denotes the weighting function associated to the node M_k.

The numerical implementation on the CRAY 2 was done within the MODULEF library in a multielement, multiproblem framework. There are two different ways to achieve parallelism on a CRAY 2 computer: the *macrotasking* which parallelizes the whole program, and the *microtasking* which parallelizes DO loops. To achieve the macrotasking one has to rewrite the code and use a specific library. The microtasking is obtained by adding some directives to the program (as for vectorization). The interest of microtasking is that the code remains portable, which is not the case with macrotasking. This algorithm was implemented using multitasking (as microtasking was not yet available). Synchronization was achieved by using shared variables and a critical section. To each subdomain corresponds a task. The tasks are executed concurrently on all available processors while the computations on each subdomain are vectorized.

5.2. Laplace operator

The first computed geometry is described in Fig. 3. The internal mesh (on Ω_1) had 1222 triangles and 669 vertices, the external mesh (on Ω_2) had 1440 triangles and 780 vertices.

For the pure Dirichlet problem, the algorithm converged in 4 iterations for $\beta_1 = \beta_2 = 1$ and in 3 iterations for $\beta_1 = 0.1$, $\beta_2 = 1$.

The second domain that we have treated is the unit square divided as indicated in Fig. 4.

Here both the pure Dirichlet and the pure Neumann problem were considered, with $\beta_i = 10^{-3}$ if $x_1 - x_2 < 0$ and $\beta_i = 1$ if not. For the Dirichlet (respectively Neumann) problem, convergence was reached after 3 (respectively 8) iterations in the case of 4 subdomains and after 6 (resp. 20)

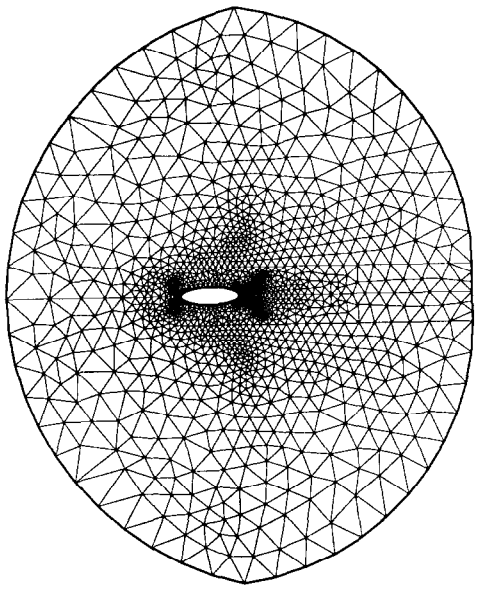

Fig. 3. Mesh of the two domains.

iterations in the case of 8 subdomains. Taking 200 triangles per subdomain instead of 100 did not change the required number of iterations.

5.3. *Numerical test in linear elasticity*

The domain considered is a TRIFLEX connection arm made of glass resin elastomere composite and submitted to various loads. The TRIFLEX arm is a cylinder (length = 343 mm) with a circular section (diameter = 55 mm) composed of 115 glass rods with circular sections (diameter = 3.2 mm).

The domain Ω was partitioned into hexahedra $Q2$. The finite element mesh (Fig. 5) contains 3944 hexahedra and 18156 nodes (54468 degrees of freedom).

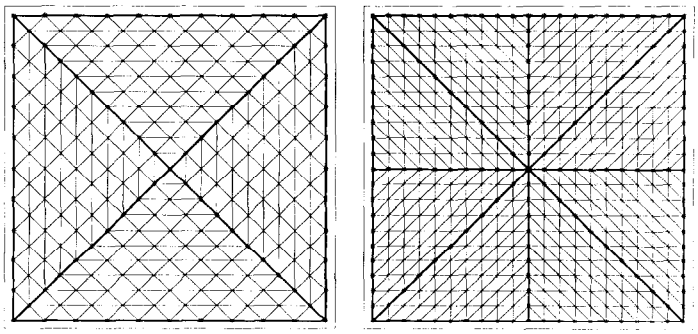

Fig. 4.

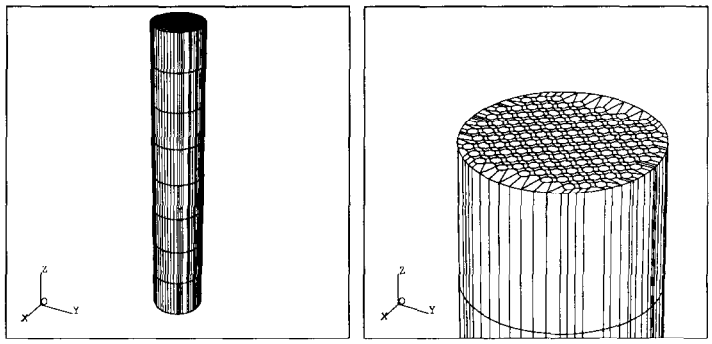

Fig. 5. The TRIFLEX conection arm.

This domain is split into four domains of equal size. Each one contains 986 elements and 5700 nodes (17100 d.o.f.). The interface between two domains is parallel to the base of the cylinder. The base contains 1548 nodes (4644 d.o.f.). So the total number of nodes on the interfaces is 4644 (13932 d.o.f.). The size of the matrix of each subdomain is 11,365,877 words.

Physical characteristic and loads

All the materials are supposed to be elastic.

-The glass rods have an orthotropic behaviour

$$E_1 = E_2 = 15800 \text{ MPa}, \qquad E_3 = 53000 \text{ MPa},$$

$$G_{31} = G_{32} = 4600 \text{ MPa}, \qquad G_{12} = 5600 \text{ MPa},$$

$$\nu_{31} = \nu_{32} = 0.24, \qquad \nu_{12} = 0.34.$$

-The resin elastomer is isotropic, nearly incompressible with

$$E = 7.8 \text{ MPa}, \qquad \nu = 0.499999.$$

The loading is achieved by an imposed displacement of the upper extremity of the cylinder, the lower one remaining fixed.

Numerical results (with reorthogonalization)

This test was run on nondedicated mode on the CRAY 2 of the CCVR at the Ecole Polytechnique.

Number of iterations (with orthogonalization): 67.

Residual: 0.653510^{-6}.

CPU-time for the matrix factorizations (one processor used): 1473 s.

CPU-time for the conjugate gradient algorithm (on four processors): 436 s.

Elapsed time for the conjugate gradient algorithm: 138 s.

Storage for the matrix of each subdomain: 11,365,877 words.

In order to assess the efficiency of the above method we also solved the same problem with more standard methods.

Direct method (Cholesky factorization). The storage for the matrix was 116,595,234 words (the pointers were not considered) and the CPU-time was 5043 s.

Iterative method. Due to the bad condition number of the problem and also because the stiffness matrix is not an M-matrix, the iterative solution of this problem by a preconditioned conjugate gradient method using either incomplete Cholesky factorization (L^tL) or incomplete Crout factorization (L^tDL) as a preconditioner was impossible. In this case the storage for the matrix was 3,443,580 words. Further experiments will consider other preconditioners which will not require an M-matrix.

6. Implementation on a distributed memory machine

6.1. The target machine

Thanks to a strong collaboration with the Goupe de Calcul Parallèle of ONERA, we have access to an INTEL iPSC/2 32 SX. This distributed memory machine employs a hypercube topology. We term by "node" any of its 32 scalar processors. In a multi-user environment, one has a private access to a subcube with a number of nodes equal to a power of two. On each node, there are 4 Megabytes of local memory, and d physical ports (if the dimension of the subcube is d) that can be accessed only sequentially.

The coarse granularity of the parallelism of the algorithm and the locality of the transfer of data at the interface give an indication of the interest of an implementation on such a machine.

6.2. The smoothed test problem

We again consider the TRIFLEX arm of Section 5. As our main interest lies in the numerical solver, and not too much on the accuracy of the discretized problem, we have chosen simpler $P1$-$Q1$ finite elements, however requiring a finer mesh. In order to simulate the incompressibility, we keep a Poisson coefficient $\nu = 0.49$, but we use under-integrated divergence. Previous experiments on the Alliant FX/80 at CERFACS have shown that for $\nu = 0.4999$, the preconditioner shows even more relative efficiency, but the convergence takes much longer. Here we wanted to save time on the computer.

In this implementation, the choice of the domain splitting strategy is wider: we call a "pencil" a carbon fiber wrapped in the elastomere with a losange cross-section (in the real case, 115 of these pencils are stacked together forming the TRIFLEX arm). In this framework, our subdomains will correspond to part of one or two pencils, sliced along the main axis of the arm.

Because of the local solver, each splitting strategy requires a different amount of local storage. Moreover, the convergence of the PCG seems to be very dependent on the splitting. A trade-off has thus to be found.

6.3. The local solvers

After discretization, a solver for the *Neumann* problem can be expressed by the factorization of the local stiffness matrix, and a solver for the *Dirichlet* problem by the factorization of the matrix obtained after elimination of the boundary degrees of freedom.

We actually perform the LDL^t factorization of these two matrices, once and for all at the beginning of the process. Thus, at each step of the PCG, one only needs to use backward and forward substitutions.

Special care must be taken of the *Neumann* subproblem: it might be singular, if there is no *Dirichlet* boundary condition at any of the boundaries of the subdomain. During the factorization, a regularization is performed whenever a null pivot is encountered. Notice that some splitting strategies avoid the occurrence of such singularities.

6.4. The reorthogonalization procedure

The first implementation on the Alliant FX/80 has shown the efficiency of using a reorthogonalization process, in order to overcome the jeopardizing of the descent direction due to rounding errors and amplified by the very ill-conditioning of the *Neumann* subproblems in the preconditioner.

This means that we replace the descent update:

$$\lambda_{k+1} = MR_{k+1} + \rho\lambda_k \qquad \text{with } \rho = \frac{\langle MR_{k+1},\, R_{k+1}\rangle}{\langle MR_k,\, R_k\rangle}$$

$$\text{by:} \quad \lambda_{k+1} = MR_{k+1} + \sum_{j=0}^{k} \rho_j \lambda_j \qquad \text{with } \rho_j = -\frac{\langle S\lambda_j,\, MR_{k+1}\rangle}{\langle S\lambda_j,\, \lambda_j\rangle}$$

where k stands for the iteration count, λ for the descent direction, R for the residual, S for the Schur complement (discretization of the *Stecklov–Poincaré* operator, associated with the *Dirichlet* problems), and M for the *Neumann* preconditioner. In fact, the new update amounts to enforcing the S orthogonality of the descent directions. This operation is cheap, even if it is performed on the whole range of descent directions, because it only involves the degrees of freedom at the interface of the subdomains.

6.5. The data structure for the interface

The distributed architecture of the computer must be taken into account in the choice of a suitable data structure for the interface.

At each iteration of the PCG, there are three interface vectors to be stored: $\overline{U}_k$, R_k and λ_k the displacement, the residual and the descent direction, respectively; and two interface vectors to be used: $\overline{T} = \sum_i S_i \lambda_k$ and $\overline{Z} = MR_k$ the conjugate direction and the preconditioned residual, respectively.

We describe here two strategies of implementation: one called *local interface*, where each node of the machine only knows the interface degrees of freedom which are in its immediate neighbourhood, the other one called *global interface*, where each node of the machine redundantly stores the whole interface.

Local interface

Allocating one subdomain to one node, only the degrees of freedom related to the interface surrounding the subdomain are stored in the associated node, see Fig. 6.

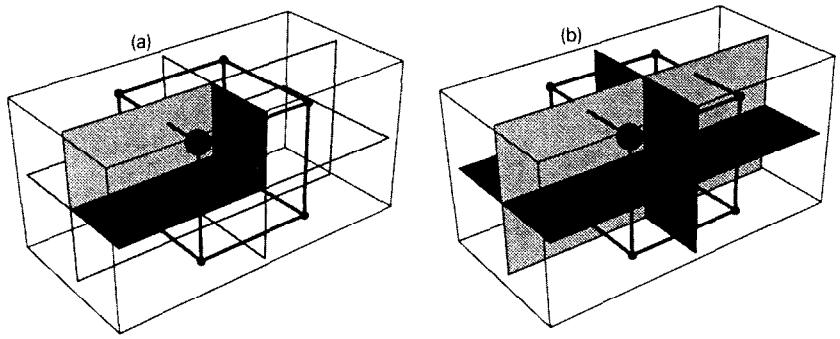

Fig. 6. 2 implementations = 2 different data structures for the interface; (a) local interface; (b) global interface.

This is the approach which leads to the minimal memory requirement, and the minimal vector length for communications.

With respect to the *gathering* operation, the data are exchanged with the neighbouring subdomains in a scheduled process, for the *send* procedure on the iPSC/2 is sequential (the physical links can only be accessed sequentially). The subdomains transfer the data along the sides they are sharing, (inducing some indirect addressing), as is shown in Fig. 7.

In this framework, the *Gray Code* is the natural way of mapping the subdomains onto the nodes: in 1-D, it produces a sequence of integers that differ from one bit to another. This allows us to map a ring onto the hypercube architecture (cf. Fig. 8). In 2-D and 3-D, the Gray code

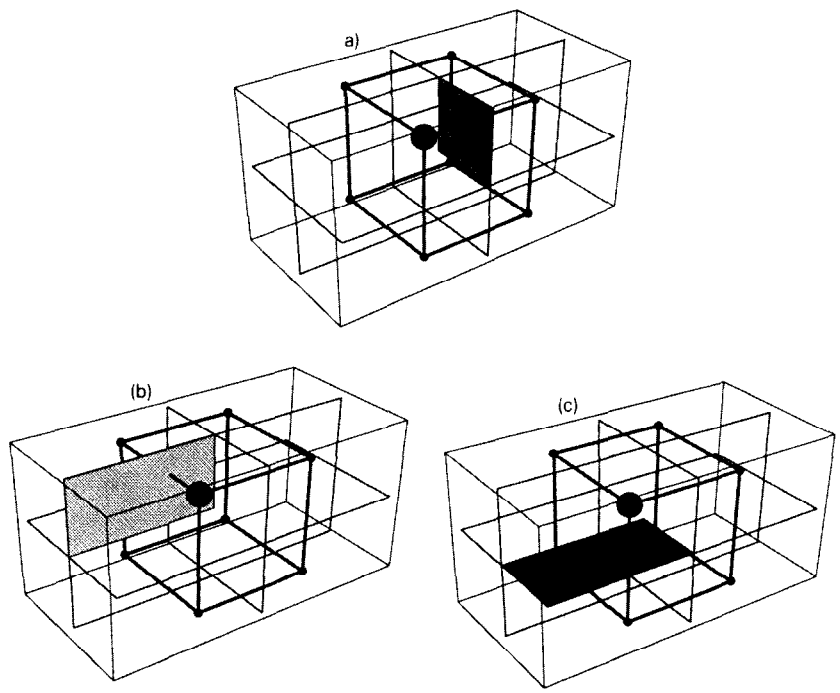

Fig. 7. Gathering at the interface in scheduled steps; (a) step 1; (b) step 2; (c) step 3.

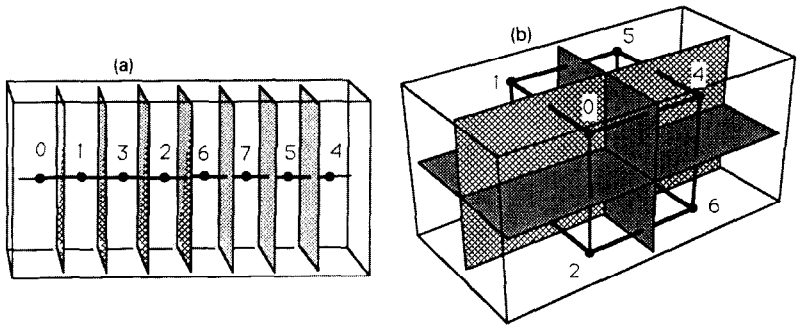

Fig. 8. Gray code numbering of a structured partition; (a) 1-D Gray Code; (b) 3-D Gray Code.

numbers integers so that neighbours on a structured grid will be actually linked on the hypercube. However, another feature of the INTEL hypercube consists in not penalizing the multiple hops communications (intermediate nodes do not use their CPU to forward the messages, but only add a slight overhead, see [4]). Thus, the repartition strategy of the subdomains among the nodes does not necessarily have to follow the Gray Code.

In summary, because of the technology of the target machine, only a natural ordering of the subdomains is needed (we assume that, as in our test problem, the splitting of the overall domain into subdomains is structured). This ordering leads to communications involving several hops, but there is no contention on any of the physical links.

Global interface

The complete interface vectors are stored and updated in each node (see Fig. 6). This induces some redundant computation, but reduces the number of communications. On the one hand, the vectors to be transferred are longer: on the other hand, no communications are needed to perform the dot-products (whereas in the other approach, local weighted subproducts are computed and then gathered).

For instance, savings in these operations leads to very low costs of reorthogonalization. The descent directions are split between all the nodes, so that a partial reorthogonalization is performed on each, and then a global summation is achieved. This avoids any redundancy and any waste of storage for this part of the computation.

With this data structure, the only exchanges are the global summations: those are scheduled to be performed by physical directions or links (see [14]), thus no special mapping of the subdomains onto the nodes is required. Thanks to the unicity of the structure of the interface, one level of indirect addressing is suppressed at the gathering operation.

A complex splitting becomes more difficult to handle with the global data structure. However, with no complementary communications, the code has been implemented with the following feature: two nodes are dedicated to each subdomain, one storing and using the *Dirichlet* solver, the other one taking care of the *Neumann* solver. Notice that on a distributed memory machine, this approach leads to a good parallelization of the memory management and postpones the risk of lack of memory for bigger problems. Of course, these two solvers are still accessed sequentially, but it implies that for a given problem, fitted to n processors in memory requirements, one splits the body into $\frac{1}{2}n$ subdomains. Hence, the convergence becomes easier, but each iteration takes longer. We will show in the examples that it is again a matter of trade-off.

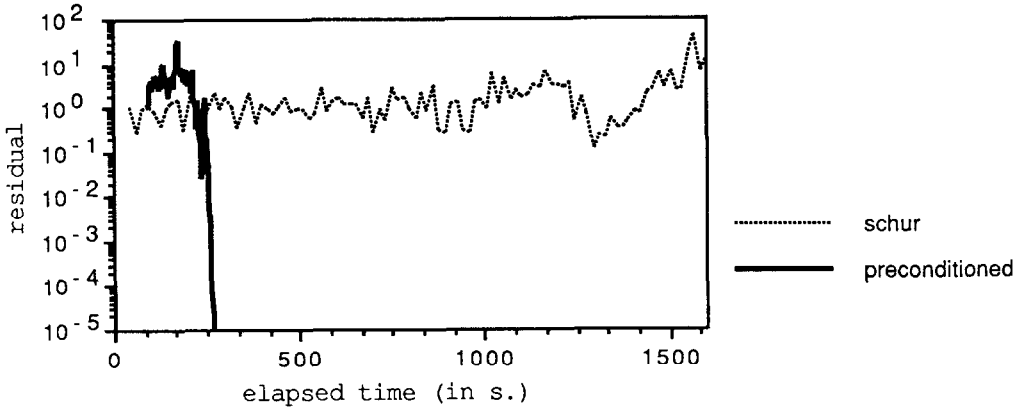

Fig. 9.

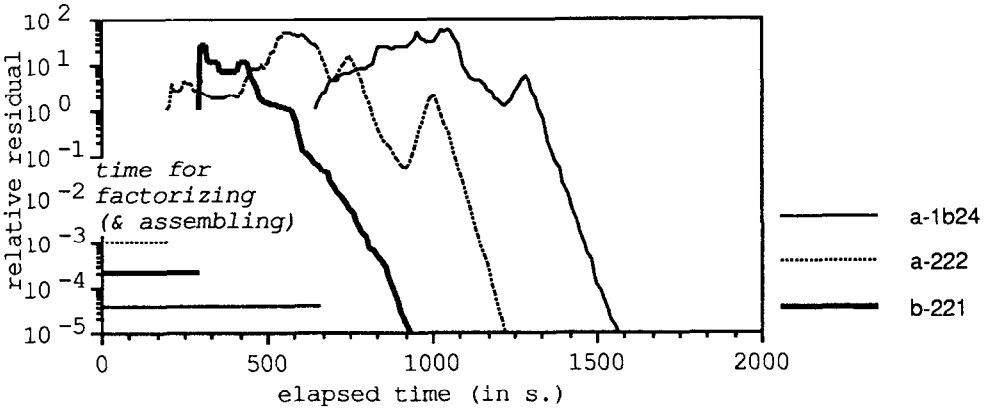

Fig. 10.

6.6. Results

Figures 9 and 10 and Table 1 show the behaviour of the interface residual versus the elapsed time. The timing begins when launching the whole code, in dedicated mode (because we deal with a private subcube).

The first step consists in building the stiffness matrix in the subdomains, but it is relatively cheap.

Then the local factorization for the *Dirichlet* solver and for the *Neumann* solver (if needed) begins, and this step corresponds to 95% of the elapsed time shown before the broken line of the residual starts. We call it *LU time*.

Table 1 — see Figs. 9 and 10

Program	Iteration	Residual	Total elapsed
Schur	1000	0.343	1670 s
w/preconditioner	76	$<10^{-5}$	279 s

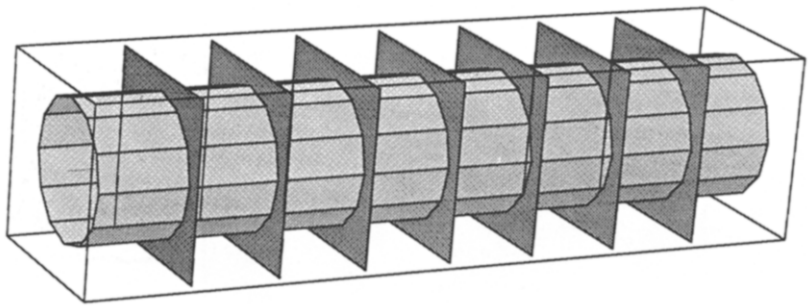

Fig. 11. Test case: 1 pencil, 8 domains.

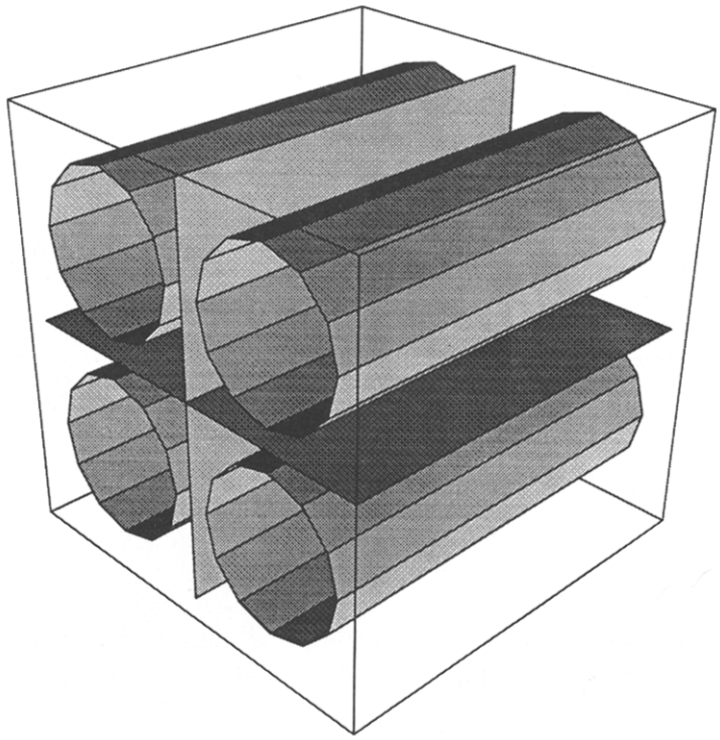

Fig. 12.

Table 2 — see Fig. 12

ndom X	2
ndom Y	2
ndom Z	1
Total d.o.f.	13005
Interface d.o.f.	1989

Table 3

"Matrix"	Iterations $r<10^{-5}$	Eigenvalues				$K=\lambda_n/\lambda_1$
		λ_1	λ_2	λ_{n-1}	λ_n	
A	direct	1.6×10^{-3}	na	2397.7204	2397.7205	1.5×10^{6}
S	286	6.38×10^{-4}	6.39×10^{-4}	2.50	2.51	3.9×10^{3}
M	na	0.411	0.415	1875.73	1875.79	4.6×10^{3}
MS	21	1.0001	1.0002	19.4	23.2	2.3×10^{1}

Table 4 — see Fig. 12

ndom X	2
ndom Y	2
ndom Z	1
Total d.o.f.	8415
Interface d.o.f.	1287
Local int. d.o.f.	693

Table 5

Program	Cube size	Maximum storage [1] per node	Iterations [2]	Elapsed time in seconds		
				LU	PCG	total
a (local)	4 nodes	194314 dpW	54	380	694	1091
b (global)	4 nodes	195322 dpW	54	375	686	1078
b (global)	8 nodes	136705 dpW	54	108	814	941

[1] Storage required before reorthogonalization, in double-precision words.
[2] Number of iterations needed to reduce the scaled residual to 10^{-5}.

Later, the value of the relative residual is plotted at each iteration, during all the *PCG time*.

Thus it is easy to notice that some strategies lead to more cost on the initialization step, losing the benefit of a fast convergence of the PCG.

Efficiency of the preconditioner

In this particular case of a beam sliced into eight subdomains, no convergence occurs for the Schur complement problem unless we use the preconditioner (see Fig. 11).

However, in the following case (see Fig. 12 and Tables 2 and 3) which is more regular, we have calculated the two largest and the two smallest eigenvalues of the different systems that we are interested in, which are:

(1) the complete stiffness matrix A, assembled over the four subdomains: rank of the matrix, 13005;

(2) the implicit Schur complement matrix S: the interface has 1989 degrees of freedom;

(3) the implicit matrix M of the preconditioner described herein: it is still an SPD matrix;

(4) the implicit matrix MS of the preconditioned iteration matrix: note that this matrix is no longer symmetric.

The eigenvalues have been found using software developed by Miloud Sadkane at CERFACS, based on a modified Arnoldi's method. This iterative method described in [6] only uses matrix–vector products. This is very well adapted to our case, since our matrices are not explicitly known.

From Fig. 12, Tables 2 and 3, we first verify that the condition number of the Schur complement matrix S is much lower than the one of the complete stiffness matrix A. Here, we gain three orders of magnitude, while keeping only one seventh of the degrees of freedom.

Then, as expected (because of the longer Neumann boundary), the condition number of the preconditioner M is larger than the one of the Schur complement itself, and the largest eigenvalues are more difficult to separate.

Finally, the preconditioned system MS has a reasonable condition number of 23, and this reduces the number of iterations from 286 to 21 in order to decrease the residual to 10^{-5} with the conjugate gradient algorithm. Note that the two largest eigenvalues are now well separated, whereas the two smallest are tightly clustered.

Thus, this "Neumann–Neumann" preconditioner is proved to work satisfactorily on this ill-conditioned problem. The following results will show how to implement it efficiently on a distributed memory machine.

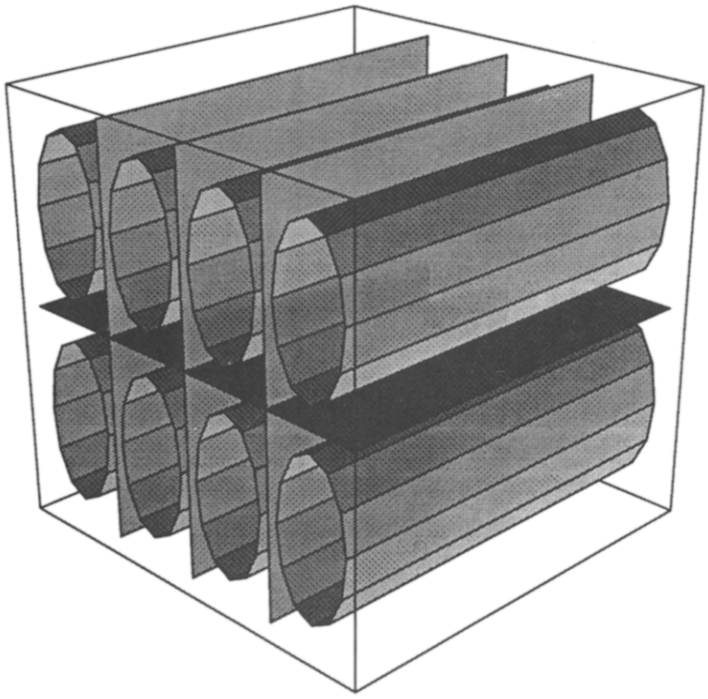

Fig. 13.

Comparison of the two data structures

In the following test case, 55 iterations are needed to reduce the relative residual to 10^{-5}. We have run the preconditioned version according to two implementations and we obtain Fig. 12 and Tables 4 and 5.

In the next test case, 108 iterations are needed. However, the implementation *a* needs more iterations, because of a lack of storage, which makes the complete reorthogonalization no longer possible (see Fig. 13 and Tables 6 and 7).

About these two data structures, the conclusion is that *b* is easier to implement, and as it can be used with two nodes per domain (without adding any more communications), it allows to

Table 6 — see Fig. 13

ndom X	2
ndom Y	4
ndom Z	1
Total d.o.f.	16137
Interface d.o.f.	3069
Local int. d.o.f.	693

Table 7

Program	Cube size	Maximum storage [1] per node	Iterations [2]	Elapsed time in seconds		
				LU	PCG	total
a (local)	8 nodes	194314 dpW	144 [3]	379	1967	2362
b (global)	8 nodes	206014 dpW	108	375	1430	1821
b (global)	16 nodes	147397 dpW	108	109	1527	1652

[1] Storage required before reorthogonalization, in double-precision words.
[2] Number of iterations needed to reduce the scaled residual to 10^{-5}.
[3] Only 97 descent directions can be stored ⇒ worse convergence.

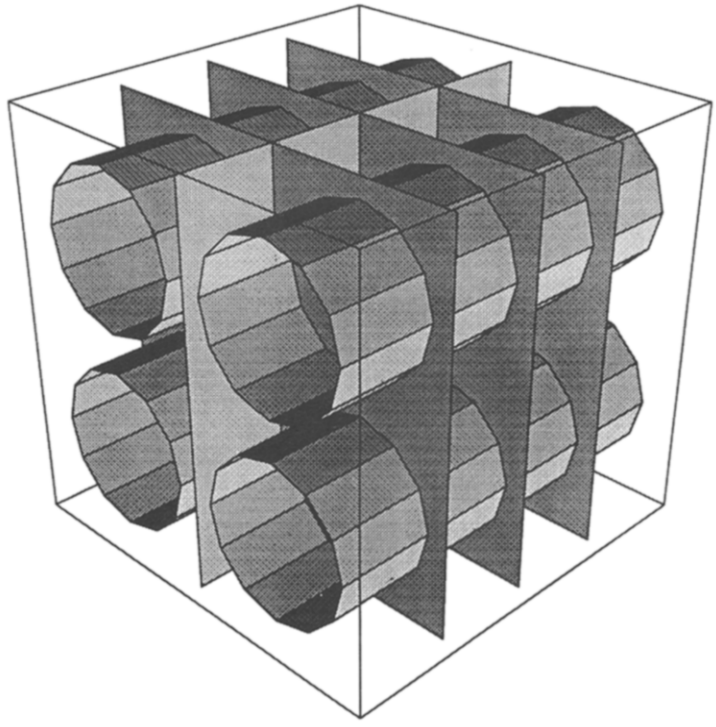

Fig. 14.

introduce a parallelism in memory storage. Hence bigger cases can be split into fewer subdomains, ensuring a faster convergence.

As the descent directions in the global interface can be split between the nodes, it avoids the redundancy occurring for storing these vectors with the local interface strategy (each degree of freedom belongs at least to two subdomains, thus twice as many descent directions can be saved with *b*).

Comparison of different splittings

We want to show that for a given problem, with a given number of processors, there are several ways of splitting the body and allocating the tasks. Here is a case of a problem with four

Table 8 — see Fig. 14

Case	a-1b24
ndom X	1
ndom Y	2
ndom Z	4
Total d.o.f.	8415
Interface d.o.f.	1395
Local int. d.o.f.	423

Table 9 — see Fig. 15

Case	a-222
ndom X	2
ndom Y	2
ndom Z	2
Total d.o.f.	8415
Interface d.o.f.	1503
Local int. d.o.f.	411

Table 10 — see Fig. 12

Case	b-221
ndom X	2
ndom Y	2
ndom Z	1
Total d.o.f.	8415
Interface d.o.f.	1287

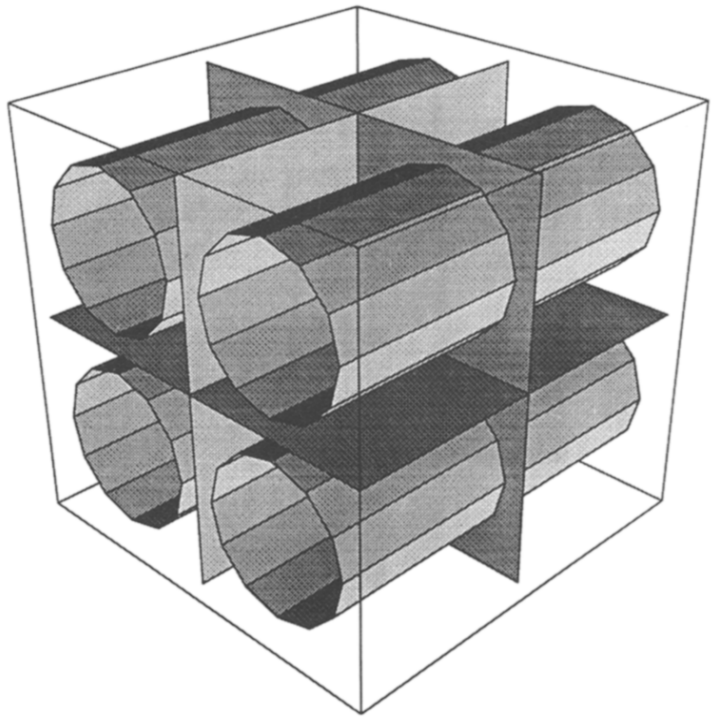

Fig. 15.

pencils, fitting on an eight node subcube (see Fig. 14 and Table 8; Fig. 15 and Table 9; Fig. 12 and Table 10; and Fig. 10 and Table 11).

The solution leading to the smallest number of domains gives the fastest result. Note also the nonnegligible time spent in factorizing the local solvers.

Table 11

Progam	Iterations $r<10^{-5}$	Elapsed time		
		LU	PCG	Total
a-1b24	91	626	939	1580
a-222	143	175	1060	1250
b-221	54	110	814	947

Table 12 — see Fig. 16

Case	a-1b44
ndom X	1
ndom Y	4
ndom Z	4
Total d.o.f.	16137
Interface d.o.f.	3357
Local int. d.o.f.	570

Table 13 — see Fig. 17

Case	a-242
ndom X	2
ndom Y	4
ndom Z	2
Total d.o.f.	16137
Interface d.o.f.	3465
Local int. d.o.f.	555

Table 14 — see Fig. 13

Case	b-241
ndom X	2
ndom Y	4
ndom Z	1
Total d.o.f.	16137
Interface d.o.f.	3069

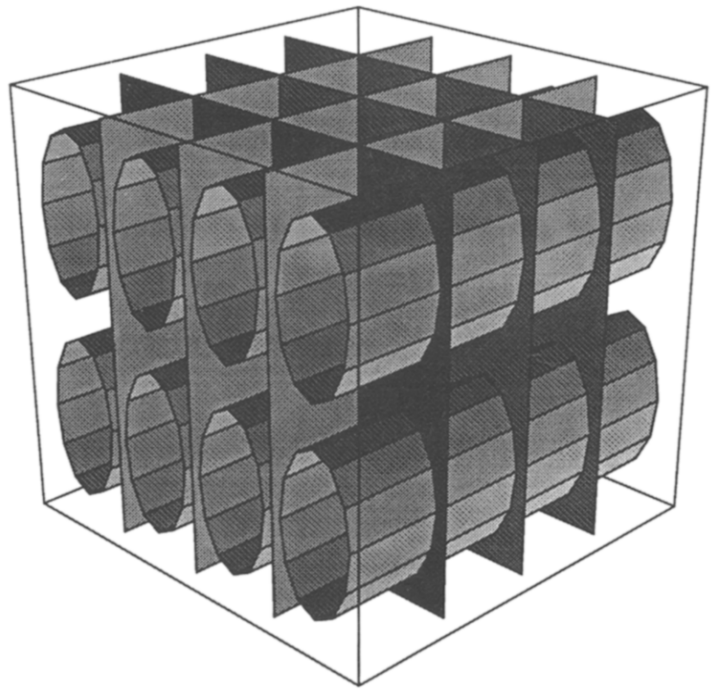

Fig. 16.

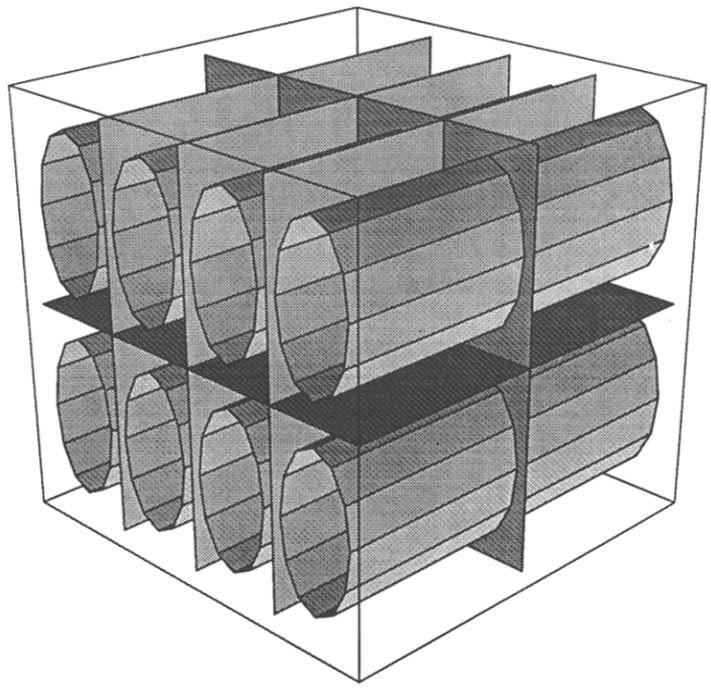

Fig. 17.

Table 15

Program	Iterations $r<10^{-5}$	Elapsed time		
		LU	PCG	Total
a-1b44	288	628	3130	3780
a-242	307	182	2660	2860
b-241	108	109	1540	1670

Table 16

Case	a-442
ndom X	4
ndom Y	4
ndom Z	2
Total d.o.f.	30987
Interface d.o.f.	7563
Local int. d.o.f.	651

Table 17

Case	b-441
ndom X	4
ndom Y	4
ndom Z	1
Total d.o.f.	30987
Interface d.o.f.	6831

Table 18

Program	Iterations	Elapsed time		
		LU	PCG	Total
a-442	1000 [1]	188	9390	9600
b-241	189 [2]	109	2800	2930

[1] To reach: $r = 8.7210^{-5}$.
[2] To reach: $r<10^{-5}$.

Next comes the case of a problem with eight pencils, fitting on a sixteen node subcube (see Fig. 16 and Table 12; Fig. 17 and Table 13; Fig. 13 and Table 14; and Table 15).

Again, the solution leading to the smallest number of subdomains converges faster.

Finally, a case with sixteen pencils, allocated on 32 nodes (see Tables 16–18).

Note that the solution *a*-442 almost fails to converge.

References

[1] V.I. Aghoskov, Poincaré–Steklov's operators and domain decomposition methods in finite dimensional spaces, in: R. Glowinski, G.H. Golub and J. Periaux, Eds., *Proc. First Internat. Symp. on Domain Decomposition Methods for Partial Differential Equations* (SIAM, Philadelphia, PA, 1988).

[2] V.I. Aghoskov and V.I. Lebedev, The variational algorithms of domain decomposition methods, Preprint 54, Dept. Numerical Math. Acad. Sci. USSR, 1983 (in Russian).

[3] V.I. Aghoskov and V.I. Lebedev, The Poincaré Steklov's operators and the domain decomposition methods in variational problems, in: *Computational Processes and Systems* (Nauka, Moscow, 1985, in Russian) 173–227.

[4] L. Bomans and D. Roose, Communication benchmarks for the iPSC/2, in: F. Andre and J.P. Vergins, Eds., *Proc. 1st European Workshop on Hypercubes and Distributed Computers* (North-Holland, Amsterdam, 1989).

[5] J.H. Bramble, J.E. Pasciak and A.H. Schatz, The construction of preconditioners for elliptic problems by substructuring, *Math. Comp.* **47** (1986) 103–134.
[6] Diem Ho, Tchebychev iteration and its optimal ellipse for nonsymmetric matrices, Rapport Technique du Centre Scientifique IBM, Paris, 1987.
[7] R. Glowinski, *Numerical Methods for Nonlinear Variational Problems* (Springer, Berlin, 1984).
[8] R. Glowinski, G.H. Golub and J. Periaux, Eds., *Proc. First Internat. Symp. on Domain Decomposition Methods for Partial Differential Equations* (SIAM, Philadelphia, PA, 1988).
[9] R. Glowinski and M. Wheeler, Domain decomposition methods for mixed finite element approximations, in: R. Glowinski, G.H. Golub and J. Periaux, Eds., *Proc. First Internat. Symp. on Domain Decomposition Methods for Partial Differential Equations* (SIAM, Philadelphia, PA, 1988).
[10] G.H. Golub and C.F. Van Loan, *Matrix Computations* (North Oxford Academic, Oxford, 1983).
[11] V.I. Lebedev, The decomposition method, Dept. Numerical Math. Acad. Sci. USSR, 1986 (in Russian).
[12] L.D. Marini and A. Quarteroni, An iterative procedure for domain decomposition methods: a finite element approach, in: R. Glowinski, G.H. Golub and J. Periaux, Eds., *Proc. First Internat. Symp. on Domain Decomposition Methods for Partial Differential Equations* (SIAM, Philadelphia, PA, 1988).
[13] P. Morice, Transonic computations by a multidomain technique with potential and Euler solvers, in: J. Zienep and H. Oertel, Eds., *Symposium Transsonicum III* (Springer, Berlin, 1989).
[14] Y. Saad and M.H. Schultz, Data communication in hypercubes, *J. Parallel Distributed Comput.* **5** (1988).
[15] O. Widlund, Iterative methods for elliptic problems on regions partitioned into substructures and the biharmonic Dirichlet problem, in: R. Glowinsky and J.-L. Lions, Eds., *Proc. 6th Internat. Symp. on Computing Methods in Applied Sciences and Engineering*, Versailles (North-Holland, Amsterdam, 1983).