AN EFFICIENT PARALLEL DISCRETE PDE SOLVER

Y. Notay [1]

Service de Métrologie Nucléaire
Université Libre de Bruxelles (C.P. 165)
50, Av. F.D. Roosevelt, B-1050 Brussels, Belgium.
email : ynotay@ulb.ac.be

Report NM-IBM 94-01

June 1994

[1]Supported by the "Fonds National de la Recherche Scientifique", Chercheur qualifié.

Abstract

We present a parallel iterative solver for discrete second order elliptic PDEs. It is based on the conjugate gradient algorithm with incomplete factorization preconditioning, using a domain decomposed ordering to allow parallelism in the triangular solves, and resorting to some special recently developed parallelization technique to avoid communication bottleneck for the computation associated to the internal boundary nodes.

Numerical results are given for a transputer network with up to 512 processors and a few workstation cluster.

keywords: iterative methods for linear systems, parallel computing, preconditioning.

AMS CLASSIFICATION : 65F10, 65B99, 65N20.

1 Introduction

We present here a parallel solver for linear systems arising from the discretization of second order two or three dimensional elliptic PDEs of the type

$$\begin{array}{ll} -\partial_x a_x \partial_x u \;-\; \partial_y a_y \partial_y u \;(-\; \partial_z a_z \partial_z u) \;=\; f & \text{in } \Omega \\ u \;=\; f_1 & \text{on } \Gamma_1 \subset \partial\Omega \\ \frac{\partial u}{\partial n} \;=\; f_2 & \text{on } \Gamma_2 \;=\; \partial\Omega\backslash\Gamma_1 \end{array} \tag{1.1}$$

As is well known, for such systems, the best methods on sequential computers are difficult to parallelize while the truly parallel algorithms compare unfavourably in term of global arithmetic work, making the problem of designing a parallel solver very challenging, especially if one targets efficiency for a large number of processors or on systems like workstation clusters for which the relative cost of the communications is very high.

Clearly, two philosophies may be followed: either one implements a method of choice from the sequential point of view, trying to minimize the effects of communication, synchronization and unperfect load balancing, or one tries to derive a truly parallel method for which the increase of the total number of arithmetic operations is kept minimal.

Up to now, the former approach has given satisfying results for parallel computers with moderated number of processors (see for instance [4, 14]), while the second one seems more adapted to massively parallel systems (see [6, 11], among many others).

The solver presented here is based on the conjugate gradient algorithm and belongs to the latter category.

As usual with this method, the tricky point is the choice of the preconditioner, on which relies both the numerical efficiency of the method and its intrinsic parallelism, the remaining of the algorithm being not difficult to parallelize on most architectures.

Incomplete factorization preconditioners allow an efficient reduction of the number of iterations (see e.g. [2, 5, 8, 21]), but require two triangular solves per iterations, which, when using classical ordering schemes, prevents large scale parallelization, even if interesting implementations where developed for a moderate number of processors thanks to a careful dependency analysis of the involved recursions [4, 14]. The latter implementations require in addition relatively fast communication for small messages and are therefore not necessarily adapted to all systems.

That is why we consider here the use of "domain decomposed" orderings. This consists in dividing the discrete domain in as many subdomains as avalaible processors and in numbering separately the interior nodes of the different sudomains on the one hand, and the internal boundary nodes on the other hand, in such a way that the part of the computation related to the interior nodes reduces to local triangular solves which may be carried out independently.

Some degradation is expected in the convergence rate but, as will be seen, limited compared with the gain in parallelism, see also [9, 10, 20].

Here, we consider the use of this principle within the framework of the results of [19].

In the latter paper, a parallelization technique is proposed which is based on the equivalence between any iterative scheme applied to the global system with the same iterative scheme applied to an augmented system, in which internal boundary nodes are represented a number of times equal to the number of subdomains to which they belongs.

The advantage is that the algorithm on this augmented system is straightforward to parallelize, since it involves only purely local computations and basic communication routines similar to that needed without preconditioning. In particular, one avoids the bottleneck potentially caused by the computation associated to internal boundary nodes [22].

Another advantage of this technique is that it leads naturally to consider ordering schemes that are more efficient from the conditioning point of view than those used in standard implementations.

In [19], we focus on the theoretical proof of the equivalence between the iterations on the augmented system and the corresponding process on the true system, the discussion of the needed assumptions in practical contexts being limited to general considerations.

Here, we address the particular case of eq. (1.1), assuming a rectangular (2D) or parallelipedal (3D) discretization grid with $n_x \times n_y$ $(\times n_z)$ nodes and a $p_x \times p_y$ $(\times p_z)$ process grid. In this context, we show in Section 2 and 3 how the framework developed in [19] leads to a very nice parallel solution algorithm. Its efficiency on a transputer network with up to 512 processors is discussed in Sections 4 while in Section 5 we give the results obtained on a 8 workstation cluster.

2 The parallel solution algorithm in 2D

We divide the physical domain Ω according to the processor grid in $p_x \times p_y$ subdomains by $(p_x - 1)$ vertical and $(p_y - 1)$ horizontal lines which are enforced to be node lines. Each subdomain is assigned to a processor which will deal with a local grid whose nodes are all nodes, including internal boundary nodes, belonging to the concerned subdomain. Hence, the union of the local node sets is larger then the true node set, but the results in [19] allow to fairly develop an iterative algorithm on such an extended variable set.

To present this algorithm and the underlying assumptions, we need, as in [19] , to introduce, for all node i in a local grid, the set $\mathcal{E}(i)$ of nodes distributed on the different local grids that correspond to the same geometrical gridpoint. For convenience, we assume $i \in \mathcal{E}(i)$, so that the operator

$$\Sigma \ : \ \forall x \ : \ (\Sigma x)_i \ = \sum_{j\in\mathcal{E}(i)} x_j \quad \text{for all } i \tag{2.1}$$

corresponds to a basic communication routine which exchanges the value of a given vector at each node for the sum of these values at the different nodes corresponding to the same gridpoint.

To proceed further we need also to choose, for each node i, two subsets $\mathcal{E}_f(i)$ and $\mathcal{E}_\ell(i)$ of $\mathcal{E}(i)$, which, besides more specific requirements, are such that $\mathcal{E}_f(i) \cap \mathcal{E}_\ell(i) \ = \ \{i\}$. Here, we make the following choices, in accordence with the recommandation in [19].

First, we define, for a processor in position (i_p, j_p),

$$\text{its} \left\{ \begin{array}{c} hf \\ h\ell \\ vf \\ v\ell \end{array} \right\} \begin{array}{l} \text{boundary as the} \\ \text{set of nodes be-} \\ \text{longing to the} \end{array} \left\{ \begin{array}{l} \text{bottom (resp. top)} \\ \text{top (resp. bottom)} \\ \text{left (resp. right)} \\ \text{right (resp. left)} \end{array} \right\} \begin{array}{l} \text{boundary} \\ \text{when} \end{array} \left\{ \begin{array}{l} \left\{ \begin{array}{l} j_p \text{ is odd} \\ \text{(resp. even)} \end{array} \right\} \\ \left\{ \begin{array}{l} i_p \text{ is odd} \\ \text{(resp. even)} \end{array} \right\} \end{array} \right\};$$

$$\text{the} \left\{ \begin{array}{c} hf \\ h\ell \\ vf \\ v\ell \end{array} \right\} \begin{array}{l} \text{boundary is said} \\ \text{internal whenever} \end{array} \left\{ \begin{array}{l} j_p > 1 \text{ (resp. } j_p < p_y) \\ j_p > p_y \text{ (resp. } j_p > 1) \\ i_p > 1 \text{ (resp. } i_p < p_x) \\ i_p < p_x \text{ (resp. } i_p > 1) \end{array} \right\} \begin{array}{l} \text{and external} \\ \text{otherwise.} \end{array}$$

Then,

$$\text{when the} \left\{ \begin{array}{c} hf \\ h\ell \\ vf \\ v\ell \end{array} \right\} \begin{array}{l} \text{boundary is internal,} \\ \text{all its nodes have} \\ \text{their correspondant} \\ \text{on the processor} \end{array} \left\{ \begin{array}{l} (i_p \ , \ j_p + 1 - 2 \bmod (j_p, 2) \ ) \\ (i_p \ , \ j_p - 1 + 2 \bmod (j_p, 2) \ ) \\ (i_p + 1 - 2 \bmod (i_p, 2) \ , \ j_p) \\ (i_p - 1 + 2 \bmod (i_p, 2) \ , \ j_p) \end{array} \right\} \text{in} \left\{ \begin{array}{c} \mathcal{E}_f \\ \mathcal{E}_\ell \\ \mathcal{E}_f \\ \mathcal{E}_\ell \end{array} \right\}.$$

Concerning the corner nodes at the intersection of two internal boundaries, their correspondant on the processor $(i_p \pm 1, j_p \pm 1)$ (depending on the case at hand) is in $\mathcal{E}_f$ when both others are in $\mathcal{E}_f$ (hf bound. $\cap$ vf bound.), in $\mathcal{E}_\ell$ if they are in $\mathcal{E}_\ell$ ($h\ell$ bound. $\cap$ $v\ell$ bound.) and in none of these sets otherwise (hf bound. $\cap$ vl bound. and hl bound. $\cap$ vf bound.)

Note that, in the present case, $i \in \mathcal{E}_f(j) \;\Leftrightarrow\; j \in \mathcal{E}_f(i)$ (which is theoretically required), but also $i \in \mathcal{E}_\ell(j) \;\Leftrightarrow\; j \in \mathcal{E}_\ell(i)$.

These choices are illustrated on Fig. 1. for a 3×3 processor grid. The interested reader will easily check that all requirements stated in [19] are satisfied.

In particular, defining the operators Σ_f and Σ_ℓ by

$$\Sigma_f \ : \ \forall_x \ : \ (\Sigma_f x)_i \ = \sum_{j \in \mathcal{E}_f(i)} x_j \text{ for all } i \tag{2.2}$$

$$\Sigma_\ell \ : \ \forall_x \ : \ (\Sigma_\ell x)_i \ = \sum_{j \in \mathcal{E}_\ell(i)} x_j \text{ for all } i \tag{2.3}$$

it is easily seen that

$$\Sigma \ = \ \Sigma_\ell \ \Sigma_f \ .$$

The parallelization technique developed in [19] consists then in performing a conjugate gradient solution on an extended variable set corresponding to the union of the local node sets. The system matrix A_d and the preconditioner $B_d \ = \ U_d^t \ P_d^{-1} \ U_d$, where $P_d \ = \ diag(U_d)$, are block diagonal which respect the partitioning of the nodes between the different processors, so that the corresponding computation may be carried out in parallel without any communication, except that the preconditioning step

$$\text{compute } g \ = \ B_d^{-1} \ r$$

is exchanged for:

$$\text{compute } g \ = \ \Sigma_f \ (P_d^{-1} U_d)^{-1} \ \Delta_f \Sigma_\ell \ (U_d^t)^{-1} \ \Sigma_f \quad r \ , \tag{2.4}$$

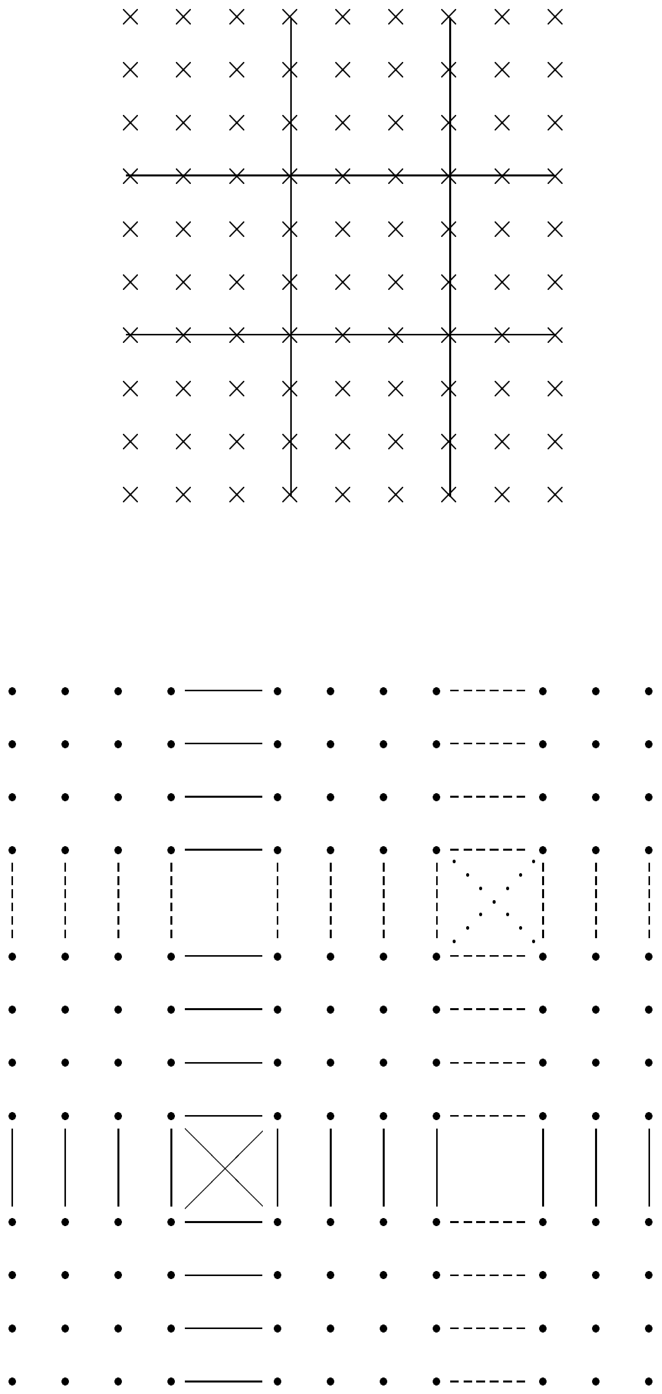

Figure 1: 9×10 discretization grid with the division of the corresponding domain in 9 subdomains (above), and associated local grids (below); nodes i, j corresponding to a same gridpoint are joined by straight lines if $i \in \mathcal{E}_\ell(j)$ $(j \in \mathcal{E}_\ell(i))$, and by dashed lines if $i \in \mathcal{E}_f(j)$ $(j \in \mathcal{E}_f(i))$.

where Δ_f is the diagonal matrix defined by

$$\Delta_f \ = \ \left(\frac{1}{\#(\mathcal{E}_f(i))} \ \delta_{ij} \right) . \tag{2.5}$$

Hence, apart from the update of the scalar products by summing the different local contributions, and from the computation and communications involved by the operators Σ_f and Σ_ℓ, each processor p just performs a local conjugate gradient solve with, as local system matrix A_p and local preconditioner $B_p \ = \ U_p^t \ P_p^{-1} \ U_p$, the diagonal blocks of respectively A_d and B_d corresponding to the nodes of its local grid.

Namely, the algorithm to be executed on each processor is as follows, where b_p, $u_p^{(0)}$ are respectively the local right hand side and initial approximation, where $\Sigma_f(.)$, $\Sigma_\ell(.)$, called simultaneously by all processors, perform the data exchanges and computation corrresponding to the operators Σ_f and Σ_ℓ, and where $sum(.)$, called simultaneously by all processors with a scalar argument, gives as answer the sum of the arguments. A left arrow indicates a step requiring commmunication.

FOR ALL p :

$$r_p^{(0)} = b_p \ - \ A_p \ u_p^{(0)}$$

For $k = 0, 1, \ldots$ *until convergence :*

$$\begin{array}{rcl}
g_p^{(k)} & = & r_p^{(k)} \\
g_p^{k} & \leftarrow & \Sigma_f(g_p^{(k)}) \\
g_p^{(k)} & := & (U_p^t)^{-1} \ g_p^{(k)} \\
g_p^{(k)} & \leftarrow & \Sigma_\ell(g_p^{(k)}) \\
g_p^{(k)} & := & (P_p^{-1} \ U_p)^{-1} \ \Delta_f \ g_p^{(k)} \\
g_p^{(k)} & \leftarrow & \Sigma_f(g_p^{(k)}) \\
\alpha_p^{(k)} & = & (g_p^{(k)} \, , \, r_p^{(k)}) \\
\hat{\alpha}_k & \leftarrow & sum(\alpha_p^{(k)}) \ \ ; \ \ \delta_k = \hat{\alpha}_k / \hat{\alpha}_{k-1} \ \ (\delta_0 = 0) \\
d_p^{(k)} & = & g_p^{(k)} \ + \ \delta_k \ d_p^{(k-1)} \ \ (d_p^{(0)} = g_p^{(0)}) \\
t_p^{(k)} & = & A_p \ d_p^{(k)} \\
\gamma_p^{(k)} & = & (t_p^{(k)} \, , \, d_p^{(k)}) \\
\hat{\gamma}_k & \leftarrow & sum(\gamma_p^{(k)}) \ \ ; \ \ \beta_k = \hat{\alpha}_k / \hat{\gamma}_k \\
u_p^{(k+1)} & = & u_p^{(k)} \ + \ \beta_k \ d_p^{(k)} \\
r_p^{(k+1)} & = & r_p^{(k)} \ - \ \beta_k \ t_p^{(k)}
\end{array}$$

The conditions under which this process is equivalent to a standard PCG solution performed on the global system are given below:

(1) The global system matrix A_g and right hand side b_g correspond to the assembly of the different local system matrices and right hand sides. That is, an element $a_{ij}^{(g)}$

in A_g ($b_i^{(g)}$in b_g) has to be the sum of the corresponding contributions in A_p (b_p) from all the processors p where both gridpoints i and j are (the gridpoint i is) represented.

(2) When two processors have a set of gridpoints in common, the order relation between the corresponding nodes is the same in both local orderings.

(3) When one or two gridpoints are shared by more than one region, the corresponding diagonal or offdiagonal element takes the same value in all concerned local matrices U_p, so that all of them are the restriction to the local grid of a matrix U_g defined on the global grid. Zeroing in the latter any entry between two gridpoints not belonging to a same region, it turns out to be unique and $B_g = U_g^t\, P_g^{-1}\, U_g$, with $P_g = diag(U_g)$, is the corresponding global preconditioner.

(4) Calling successors of a node i in a local grid the nodes j, necessarily with larger indexes, such that $u_{ij}^{(p)} \neq 0$ in the corresponding local matrix U_p, all nodes of the $h\ell$ (respectively $v\ell$) boundary, whenever internal, have only as successors nodes belonging to the $h\ell$ (resp. $v\ell$) boundary.

(5) Calling precursors of a node i in a local grid the nodes j, necessarily with smaller indexes, such that $u_{ji}^{(p)} \neq 0$ in the corresponding local matrix U_p, all nodes belonging to the hf (respectively vf) boundary, whenever internal, have only as precursors nodes belonging to the hf (resp. vf) boundary.

(6) The value of the initial approximation at any node in a local grid is equal to the value of the global initial approximation $u_g^{(0)}$ at the corresponding gridpoint.

Under these conditions, Theorem 3 in [19] proves that the approximate solution obtained at some node in a local grid is equal, at any step, to the approximate solution that one would obtain for the same step at the corresponding gridpoint by performing a sequential PCG solution of the system $A_g u_g = b_g$ with $u_g^{(0)}$ as initial approximation and B_g as preconditioner.

More particularly, the interested reader may check that (1), (3) and (6) correspond the general theoretical requirements while (4) and (5) particularize to the context described above the more specific assumptions (1) and (2) of the theorem; (2) is not stricto sensu required but readily follows from (3) together with the fact that we want all local matrices U_p be upper triangular.

(1) is not difficult to satisfy in practice and we shall assume here that the discretization process directly provides local matrices A_p and right hand sides b_p, whose assembly gives the global system matrix and right hand side; when using a finite element discretization or the finite difference mesh box integration scheme [16], it suffices indeed to perform a local discretization on the subdomains, treating internal boundaries as if they were external with natural boundary conditions.

To manage (2) while minimizing the constraints raised by (4) and (5), we decided to order the nodes of each local grid lexicographically, starting at the intersection of the hf and vf boundaries, and progressing first in the x direction, i.e. along the hf boundary. This is illustrated on Fig. 2 for the example of Fig. 1. One easily checks that

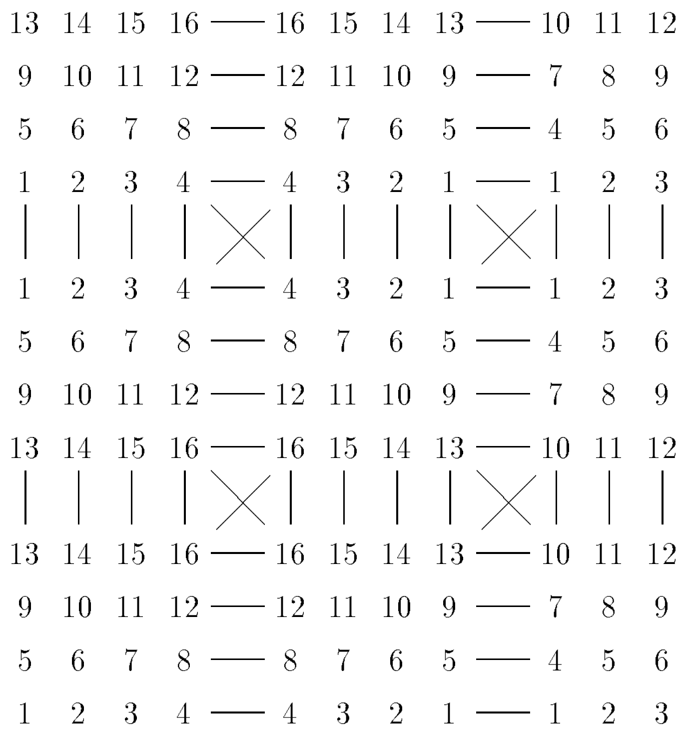

Figure 2: Local orderings for the example of Fig.1. Nodes corresponding to a same gridpoint are joined by straight lines.

all connections raised by a 5-point scheme are compatible with (4) and (5) when using a generalized SSOR factorization (that is when U_g and A_g have same upper triangular part).

However, this does not means that our solver is limited to 5 point finite difference or linear finite element discretizations. Indeed, as the existence and conditioning properties of approximate factorization preconditioners are guaranteed only if the factorized matrix is a Stieltjes matrix (i.e. symmetric positive definite with nonpositive offdiagonal entries), it is a common technique to use as preconditioner for a further scheme the incomplete factorization of the corresponding 5 point finite difference or linear finite element matrix, see e.g. [2]. Other techniques used to derive Stieltjes approximations may also be used to discard some selected negative offdiagonal entries, see [17].

Hence, we may assume here, without loss of generality, that, together with the local system matrices A_p, it is furnished local approximations $\tilde{A}_p$ whose connections are all compatible with (4) and (5) and such that the matrix $\tilde{A}_g$ corresponding to their assembly is a Stieltjes matrix.

It remains to satisfy (3). Here, we choose to consider generalized SSOR factorizations, so that the first step of the factorization procedure consists in settting the offdiagonal entries in the upper triangular part of the local matrices U_p equal to the corresponding value in $\tilde{A}_g$, by means of the information stored in $\tilde{A}_p$ and basic communication between neighbour processors.

The data dependency in the corresponding U_g is then as illustrated on Fig. 3 for the

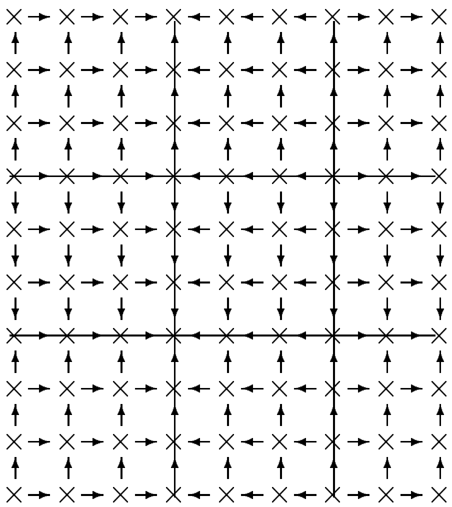

Figure 3: Data depending in u_g for the example of Fig. 1 & 2 with a 5 point grid and a generalized SSOR factorization. All connections in U_g are represented by an arrow which originates from the node with smallest index.

example of Fig. 1 & 2 with a 5 point grid.

It corresponds to that obtained with a domain decomposed ordering, except that here part of the boundaries are ordered first (the hf and vf ones), and the remaining last ($h\ell$ & $v\ell$), which results in some benefit compared with classical implementations where all boundaries are ordered last, a p processors ordering in the latter framework being more or less equivalent to a $4p$ processors ordering in the present implementation, see [19] for more details.

The diagonal of U_p is then computed as a vector π on the local grid according to the algorithm given below, where we take advantage of some features of the present implementation to simplify the communications required in the general algorithm given in [19].

We give the algorithm for the DRIC method [18, 21] which we chosed for the numerical tests in the next section; it is easily converted in an algorithm for the IC method [15], the MIC method (without perturbations) (see e.g. [8]), or the RIC method [3] by using respectively $\omega_i \equiv 0$, $\omega_i \equiv 1$ or $\omega_i \equiv \omega$ for some $0 < \omega < 1$. α is a parameter such that $0 < \alpha \leq 1$ (see Section 5), and σ an auxiliary vector.

FOR ALL p :

- *initialize, for all i*

$$\begin{aligned} \pi_i &= \tilde{a}_{ii}^{(p)} \\ \sigma_i &= \sum_{j>i} \tilde{a}_{ij}^{(p)} \end{aligned}$$

- *set*

$$\begin{aligned} \pi &\longleftarrow \Sigma_f(\pi) \\ \sigma &\longleftarrow \Sigma_f(\sigma) \\ \sigma &\longleftarrow \Sigma_\ell(\sigma) \end{aligned}$$

- *for* $i = 1, \ldots, n_p - 1$, *where* n_p *is the order of the local grid, do, whenever* i *does not belong to the* $h\ell$ *nor the* $v\ell$ *boundary*

$$\omega_i = min\left(\frac{2(1-\alpha)\pi_i}{-\sigma_i} - 1,\ 1\right)$$

$$\textit{for all } j > i \ :\ u_{ij}^{(p)} \neq 0$$

$$\pi_j = \pi_j - \frac{u_{ij}^{(p)}}{\pi_i}\,(u_{ij}^{(p)}(1-\omega_i) + \omega_i\sigma_i)$$

- *set*

$$\begin{aligned} \pi &\longleftarrow \Sigma_\ell(\pi) \\ \pi_{n_p} &:= \frac{\pi_{n_p}}{\#(\mathcal{E}_\ell(n_p))} \end{aligned}$$

- *for* $i = 1, \ldots, n_p - 1$, i *belonging to either the* $h\ell$ *or the* $v\ell$ *boundary, do*

$$\omega_i = min\left(\frac{2(1-\alpha)\ \pi_i}{\sigma_i} - 1,\ 1\right)$$

$$\textit{for all } j > i,\ j \neq n_p \ :\ u_{ij}^{(p)} \neq 0$$

$$\pi_j := \pi_j - \frac{u_{ij}^{(p)}}{\pi_i}\left(u_{ij}^{(p)}\ (1-\omega_i) + \omega_i\ \sigma_i\right)$$

$$\textit{if } u_{in_p} \neq 0 \ :$$

$$\pi_{n_p} := \pi_{n_p} - \frac{1}{\#(\mathcal{E}_\ell(i))}\,\frac{u_{in_p}^{(p)}}{\pi_i}(u_{in_p}^{(p)}\ (1-\omega_i) + \omega_i\ \sigma_i)$$

- *set*

$$\pi_{n_p} \longleftarrow \sum_{\hat{n}_p\ \in\ \mathcal{E}_\ell(n_p)} \pi_{\hat{n}_p}$$

Remark

It is relevant to compare here briefly our approach with the interesting method proposed by Radicati di Brozolo et al. [23, 12].

In the latter papers, it is suggested to split the unknowns into overlapping blocks, preferably in such a way that all nonzero entries in the system matrix appear in at least one block. Then, to each block is associated a “local” incomplete factorization, either by taking the restriction to the block of a beforehand computed global factorization, or by just performing an incomplete factorization of the "local" part of the system matrix.

The preconditioning step $g = B^{-1}r$ consists then in computing (in parallel) local triangular solves, the global vector g being set equal to the local ones in the nonoverlapped parts and to the average of them in the overlapped parts.

Clearly, there are similarities between this "Overlapped Partitioned ILU (OPI)", and our suggestion to iterate on an augmanted system, and one may even think that our method, although originating from a different point of view, is just a sophisticated version of OPI.

Now, the interesting point is that the main potential drawbacks of the OPI method are avoided by using our approach. Indeed :

- in [23], the authors obtain actually the best iteration counts when using as local triangular factors the restriction of a global ILU which has to be computed sequentially; our parallel factorization algorithm allows to compute a global factorization by means of local computation;

- even when using the restriction of a globally defined factorization, the averaging process in the OPI method implies the loss of the equivalence with the sequential algorithm, so that one may have some doubts on the convergence properties, espacially for difficult problems with discontinuities or in case of many processors; thanks to the operators Σ_f, Σ_ℓ and Δ_f, which are not more complicated to program than a mere average procedure, we maintain the equivalence with the sequential process; in particular, this allows to use incomplete factoriztion schemes based on a conditioning analysis proving that the number of iterations is bounded by $\mathcal{O}(h^{-\frac{1}{2}})$, while it is $\mathcal{O}(h^{-1})$ for standard ILU.

3 The algorithm in 3D

In 2D, both directions x and y play exactly the same role, and the 3D algorithm is just an extension of the 2D one to a third demension.

In fact, the only tricky point is the adaptation of the setting of the subsets $\mathcal{E}_f(i)$ and $\mathcal{E}_l(i)$ of $\mathcal{E}(i)$ to the case of a 3D node grid distributed on a 3D processor grid.

To this aim, we extend as follows the definitions given in 2D : for a processor in position (i_p, j_p, k_p),

$$\text{its} \left\{ \begin{array}{c} xf \\ x\ell \\ yf \\ y\ell \\ zf \\ z\ell \end{array} \right\} \begin{array}{l} \text{boundary is the} \\ \text{set of nodes be-} \\ \text{longing to the} \end{array} \left\{ \begin{array}{l} \text{left (resp. right)} \\ \text{right (resp. left)} \\ \text{back (resp. front)} \\ \text{front (resp. back)} \\ \text{bottom (resp. top)} \\ \text{top (resp. bottom)} \end{array} \right\} \begin{array}{l} \text{boundary} \\ \text{when} \end{array} \left\{ \begin{array}{c} \left\{ \begin{array}{l} i_p \text{ is odd} \\ \text{(resp. even)} \end{array} \right\} \\ \left\{ \begin{array}{l} j_p \text{ is odd} \\ \text{(resp. even)} \end{array} \right\} \\ \left\{ \begin{array}{l} k_p \text{ is odd} \\ \text{(resp. even)} \end{array} \right\} \end{array} \right\};$$

the $\left\{\begin{array}{c} xf \\ x\ell \\ yf \\ y\ell \\ zf \\ z\ell \end{array}\right\}$ boundary is said internal whenever $\left\{\begin{array}{l} i_p > 1 \text{ (resp. } i_p < p_x) \\ i_p < p_x \text{ (resp. } i_p > 1) \\ j_p > 1 \text{ (resp. } j_p < p_y) \\ j_p > p_y \text{ (resp. } j_p > 1) \\ k_p > 1 \text{ (resp. } k_p < p_z) \\ k_p < p_z \text{ (resp. } k_p > 1) \end{array}\right\}$ and external otherwise.

Then,

when the $\left\{\begin{array}{c} xf \\ x\ell \\ yf \\ y\ell \\ zf \\ z\ell \end{array}\right\}$ boundary is internal, all its nodes have their correspondant on the processor $\left\{\begin{array}{l} (i_p + 1 - 2 \bmod (i_p, 2)\ ,\ j_p\ ,\ k_p) \\ (i_p - 1 + 2 \bmod (i_p, 2)\ ,\ j_p\ ,\ k_p) \\ (i_p\ ,\ j_p + 1 - 2 \bmod (j_p, 2)\ ,\ k_p) \\ (i_p\ ,\ j_p - 1 + 2 \bmod (j_p, 2)\ ,\ k_p) \\ (i_p\ ,\ j_p\ ,\ k_p + 1 - 2 \bmod (k_p, 2)) \\ (i_p\ ,\ j_p\ ,\ k_p - 1 + 2 \bmod (k_p, 2)) \end{array}\right\}$ in $\left\{\begin{array}{c} \mathcal{E}_f \\ \mathcal{E}_\ell \\ \mathcal{E}_f \\ \mathcal{E}_\ell \\ \mathcal{E}_f \\ \mathcal{E}_\ell \end{array}\right\}$.

In addition, the ridge nodes at the intersection of two internal boundaries have their correspondant on the processor $(i_p \pm 1, j_p \pm 1, k_p)$, $(i_p \pm 1, j_p, k_p \pm 1)$ or $(i_p, j_p \pm 1, k_p \pm 1)$ (depending on the case at hand) in $\mathcal{E}_f$ when both others are in $\mathcal{E}_f$, in $\mathcal{E}_\ell$ if they are in $\mathcal{E}_\ell$, and in none of these sets otherwise. Similarly, the corner nodes at the intersection of three internal boundaries have their correspondant on the processor $(i_p \pm 1, j_p \pm 1, k_p \pm 1)$ in $\mathcal{E}_f$ if all 7 others are in $\mathcal{E}_f$, in $\mathcal{E}_\ell$ if they are in $\mathcal{E}_\ell$, and in none of these sets otherwise.

With (2.2), (2.3) and (2.5), this is sufficient to fix the operators Σ_f, Σ_ℓ and Δ_f. The parallel solution algorithm is then the same as that given above in the 2D case (see [19]). Moreover, the conditions under which it is equivalent to a standard PCG solution performed on the global system are the same conditions (1)-(6) given above, except that here (4) is to be applied to the $x\ell$, $y\ell$ and $z\ell$ boundaries and (5) to the xf, yf and zf boundaries.

The local orderings are still lexicographic ones, here starting at the corner intersecting the xf, yf and zf bondaries. As in the preceding case, this implies that all connections raised by a 7 point scheme are compatible with the given conditions. Hence, considering again generalized SSOR factorizations, the offdiagonal entries in the local upper triangular factors are just equal to the corresponding entries in the global system matrix. The parallel factorization algorithm to determine the diagonal part is here slightly more comlicated, and we refer to [19] for its detailed description.

4 Numerical results on Parsytec GCel

We tested the method described in Sections 2 and 3 on a transputer (T 805) network (Parsytec GCel-3\512 multiprocessor).

We exclusively used the communication routines provided by the PARIX virtual topology library, which is invoked by the program to build a processor grid with user specified dimensions, see [7] for details. We therefore did not consider the use of low level programation language to reduce the communication cost.

The program was written in double precision FORTRAN. We did not consider scaling to save some multiplications during the iterations, and, more generaly, no particular optimization effort was made, so that the timing results given below cannot pretend to be optimal and should be used only as a relative criteria.

The stopping criterion was $\| r_g^{(k)} \|_{B^{-1}} < 10^{-6} \| b_g \|_{B^{-1}}$, where $r_g^{(k)}$ is the residual (of the global equivalent process), b_g the right hand side and $\| \, . \, \|_{B_g^{-1}} = \sqrt{(., B_g^{-1} .)}$. This makes the test dependent of the preconditioner, but avoids the computation of any additional scalar product, reason for which this test serves as basis for the "natural" stopping criterion proposed in [1]. In practice, we observed that this dependence has only a limited influence on the number of iterations. Another advantage of this criterion in our context is that $\| r_g^{(k)} \|_{B_g^{-1}}$ may be directly computed on the augmented variable set since, see [19], $\| r_g^{(k)} \|_{B_g^{-1}} = \hat{\alpha}_k$ for the quantity $\hat{\alpha}_k$ computed as in the algorithm given in Section 2.

4.1 2D problems

As test problems, we considered the PDE (1.1) with $\Omega = (0,1) \times (0,1)$ and

PROBLEM 1 : $a_x = a_y = f \equiv 1$ in Ω , $u = 0$ on $\partial\Omega$

PROBLEM 2 :

$$a_x = a_y = \begin{cases} 100 & \text{in } (\frac{1}{4}, \frac{3}{4}) \times (\frac{1}{4}, \frac{3}{4}) \\ 1 & \text{elsewhere ,} \end{cases}$$

$$f = \begin{cases} 100 & \text{in} (\frac{1}{4}, \frac{3}{4}) \times (\frac{1}{4}, \frac{3}{4}) \\ 0 & \text{elsewhere ,} \end{cases}$$

$$\begin{cases} u = 0 & \text{for } 0 \le x \le 1 \text{ , } y = 0 \\ \frac{\partial u}{\partial n} = 0 & \text{on the remaining part of the boundary} \end{cases}$$

PROBLEM 3 :

$$a_x = 1 \quad \text{and} \quad a_y = \begin{cases} .001 & \text{in } (\frac{1}{4}, \frac{3}{4}) \times (\frac{1}{4}, \frac{3}{4}) \\ 1 & \text{elsewhere ,} \end{cases}$$

$$f = \begin{cases} 1 & \text{in} (\frac{1}{4}, \frac{3}{4}) \times (\frac{1}{4}, \frac{3}{4}) \\ 0 & \text{elsewhere ,} \end{cases}$$

$$\begin{cases} u = 0 & \text{for } 0 \le x \le 1 \text{ , } y = 1 \text{ and } 0 \le y \le 1 \text{ , } x = 1 \\ \frac{\partial u}{\partial n} = 0 & \text{on the remaining part of the boundary} \end{cases}$$

In all cases, we considered the 5 point finite difference approximation (mesh box integration scheme [16]) with uniform mesh size h in both directions. We used as preconditioner for the conjugate gradient solution, the domain decomposed generalized SSOR, DRIC factorization described in Section 2 with parameter $\alpha = h$ following the recommendations in [21].

For comparison purpose, we give also the results obtained for the Jacobi preconditioner with 256 processors. For convenience, we implemented it within the framework described in Section 2, using

$$g \;=\; D_d^{-1}\, \Sigma_\ell\, \Sigma_f\, r \tag{4.1}$$

instead of the preconditioning step (2.4), where D_d is the diagonal matrix whose each diagonal entry is equal to the diagonal entry of the global system matrix at the corresponding gridpoint [1]. The communications are then the same as that needed with an optimal implementation [2], but, since one iterates on an augmented variable set, one may have an extra arithmetic cost of at most 25%, 12.5%, 6% and 3% for respectively $h^{-1} = 128,\ 256,\ 512$ and 1024 [3]

The observed computing time for the solution part, as well as the number of iterations, are reported in Table 1. In any case, we used square processor grids ($p_x = p_y$) and zero initial approximations.

From the table, it is seen that the computing time decreases in any case when the number of processors increases. We do not give efficiencies because the comparative run on a single processor was in most cases not possible, due to excessive memory requirements. However, one may check that, when the number of processors is multipled by 4, the computing time is reduced by a factor more or less equal to 3, with large variations due to the irregularities in the increase of the number of iterations. For the smallest problems and the largest number of processors, this factor is further reduced because the relative communication cost increases prohibitevely.

This may appear not really impressive, but it is precisely one of the major feature of our solver that the amount of parallelism offered is automatically adapted to the number of available processors, so that the global arithmetic work increases continuously with the latter, but without any minimal number of processors required to run faster than the sequential execution of a comparison method. In fact, our solver, in the 1 processor case, reduces to the preconditioned conjugate gradient method with a generalized SSOR-DRIC factorization preconditioner computed with respect the standard lexicographic ordering, which may fairly be considered as a comparision method for discrete problems of the type (1.1) solved with the conjugate gradient algorithm, see [18, 21].

Moreover, from our tests, it appears that this number of iterations do generally not increases when *#proc.* increases from 1 to 4 (in accordance with the theoretical and experimental results in [9, 20]). Then, it increases slightly less then $O((\#proc.)^{1/4})$, and, for 256 processors, remains not greater than about twice that required by the single or four processor version, which represents a quite reasonable cost for such a massively parallel method.

[1] indeed, for the choice $U_d = P_d = D_d$, which is compatible with our assumptions, (2.4) reduces to $g = \Sigma_f \Delta_f D_d^{-1} \Sigma_\ell \Sigma_f\ r$, which is readily found equivalent to (4.1), while $D_g = diag(A_g)$ is obviously the equivalent sequential preconditioner

[2] that is 1 real data exchange per internal boundary node and per iteration, with respect a box partitioning which minimizes the number of internal boundary nodes.

[3] these estimations assume a perfect load balancing for the optimal implementation, which is seldomly attainable in practice.

PROBLEM 1

h^{-1}	128		256		512		1024	
# proc.	time	(#it)	time	(#it)	time	(#it)	time	(#(it)
1	47.1	(36)	-	(52)	-	(77)	-	(114)
4	9.79	(29)	60.1	(45)	-		-	
16	3.10	(32)	16.3	(46)	97.1	(71)	-	
64	1.29	(42)	6.44	(66)	36.6	(103)	220.	(161)
256	1.02	(58)	3.14	(90)	14.2	(140)	75.0	(209)
Jacobi, 256 proc.	3.48	(203)	12.0	(409)	66.0	(827)	464.	(1671)

PROBLEM 2

h^{-1}	128		256		512		1024	
# proc.	time	(#it)	time	(#it)	time	(#it)	time	(#(it)
1	74.7	(56)	-	(82)	-	(123)	-	(182)
4	17.6	(51)	99.9	(74)	-		-	
16	5.77	(60)	32.2	(91)	190.	(139)	-	
64	2.41	(89)	11.1	(114)	62.4	(176)	356.	(261)
256	1.73	(100)	5.24	(150)	22.4	(222)	118.	(330)
Jacobi, 256 proc.	6.78	(452)	25.9	(910)	146.	(1840)	1027.	(3705)

PROBLEM 3

h^{-1}	128		256		512		1024	
# proc.	time	(#it)	time	(#it)	time	(#it)	time	(#(it)
1	80.7	(61)	-	(88)	-	(127)	-	(183)
4	24.5	(71)	142.	(105)	-		-	
16	7.02	(73)	38.2	(108)	221.	(162)	-	
64	2.98	(98)	13.8	(142)	74.8	(211)	423.	(310)
256	2.45	(132)	6.51	(187)	27.8	(275)	142.	(397)
Jacobi, 256 proc.	8.59	(618)	34.1	(1162)	200.	(2556.)	1441.	(5203)

Table 1: Solution time (in seconds) and number of iterations for 2D problems on Parsytec GCel

For large systems, this cost is unavoidable, due to the limited amount of memory available on each processor. Classical computers do not suffer from this penalty, but increasing problem sizes means increasing use of virtual memory and slowing down of the whole process.

This penalty is also avoided with implementations like that described in [4, 14], where the incomplete factorization preconditioners with lexicographic ordering are parallelized by means of a wavefront like method. However, it turns out that such techniques require relatively fast communication for small messages, and can in addition not be efficient for large numbers of processors, since they require a division of the discrete domain into strips; for instance a strip partitioning in 256 processors is not possible for $h^{-1} = 128$, and causes a severe load balancing problem for $h^{-1} = 512$, the number of columns to handle being actually 513 when both left and right sides of the domain have been specified Neumann boundary conditions. The same remark holds for the implementation of block preconditionings like that proposed in [13].

Comparing now our solver with a standard Jacobi preconditioned conjugate gradient solution, it is clear that the latter is oumatched by far, even taking into account the factorization cost on the one hand and the non optimality of our implementation of the Jacobi preconditioning on the other hand. Actually, our incomplete factorization preconditioner allows as usual to save arithmetic work, but also broadcast communication for the update of the scalar products (since the number of iterations is cut down by a factor up to ten), and communication between neighbour processors (since, per iteration, our solver requires only 50% more such communication than the amount needed for a mere multiplication by the system matrix).

The number of iterations required with our preconditioner is $0(h^{-1/2})$ for fixed number of processors, and thus globally slightly better than $0(h^{-1/2}(\#proc.)^{1/4})$, while, for the Jacobi preconditioner, it grows like $O(h^{-1})$, which is less interesting even if the number of processors is increased together with the problem size so as to maintain the load per processor constant.

We dit not made direct comparison with other popular purely parallel preconditioners like polynomial preconditioners or those based on red-black or multicolor orderings. However, it is clear that these methods can improve the Jacobi method only by a constant factor, the associated number of iterations remaining $0(h^{-1})$, see e.g. [6]. On the other hand, such preconditioners invlove, per iterations, more communications than the one used in our solver within the framework of the implementation described here, see for instance [11]. Hence, we believe that our solver performs also better than the latter methods for examples like those considered here.

4.2 3D problems

We considered also the PDE (1.1) with $\Omega = (0,1) \times (0,1) \times (0,1)$ and

PROBLEM 4 : $a_x = a_y = a_z = f \equiv 1$ in Ω , $u = 0$ on $\partial\Omega$

PROBLEM 5 :

$$a_x = a_y = a_z = \begin{cases} 100 & \text{in } (\frac{1}{4},\frac{3}{4})\times(\frac{1}{4},\frac{3}{4})\times(\frac{1}{4},\frac{3}{4}) \\ 1 & \text{elsewhere ,} \end{cases}$$

$$f = \begin{cases} 100 & \text{in}(\frac{1}{4},\frac{3}{4})\times(\frac{1}{4},\frac{3}{4})\times(\frac{1}{4},\frac{3}{4}) \\ 0 & \text{elsewhere ,} \end{cases}$$

$$\begin{cases} u &= 0 \quad \text{for } 0 \le x, z \le 1 \text{ , } y = 0 \\ \frac{\partial u}{\partial n} &= 0 \quad \text{on the remaining part of the boundary} \end{cases}$$

PROBLEM 6 :

$$a_x = a_y = a_z = \begin{cases} .001 & \text{in } (\frac{1}{4},\frac{3}{4})\times(\frac{1}{4},\frac{3}{4})\times(1,1) \\ 1 & \text{elsewhere ,} \end{cases}$$

$$f = \begin{cases} 1 & \text{in } (\frac{1}{4},\frac{3}{4})\times(\frac{1}{4},\frac{3}{4})\times(1,1) \\ 0 & \text{elsewhere ,} \end{cases}$$

$$\begin{cases} u &= 0 \quad \text{for } 0 \le x, y \le 1 \text{ , } z = 0 \text{ , } 0 \le y, z \le 1 \text{ , } x = 0 \text{ and } 0 \le x, z \le 1 \text{ , } y = 0 \\ \frac{\partial u}{\partial n} &= 0 \quad \text{on the remaining part of the boundary} \end{cases}$$

In all cases, we considered the 7 point finite difference approximation (mesh box integration scheme [16]) with uniform mesh size h in all directions. We used as preconditioner for the conjugate gradient solution, the domain decomposed generalized SSOR, DRIC factorization described in Section 3 with parameter $\alpha = h$.

The observed computing time for the solution part, as well as the number of iterations, are reported in Table 2. In any case, we used cubic processor grids ($p_x = p_y = p_z$) and zero initial approximations.

For comparison purpose, we give also the results obtained for the Jacobi preconditioner with 512 processors.

Compared with the 2D results, it turns out that the increase of the number of iterations with the number of processors is here much more moderate. This explains because, in both cases, the leading parameters appears to be the number of processors per line in the grid. Therefore, as one may check by comparing Tables 1 and 2, the deterioration in convergence observed for 512 ($= 8 \times 8 \times 8$) processors in 3D is similar to that observed for only 64 ($= 8 \times 8$) processors in 2D.

Nevertheless, if the parallelization is more easy from this point of view, the timing results indicate that the communication overhead is here more significative. We explain this by two factors. On the one hand, the internal boundaries are here the faces of a cube instead of the edges of a square, so that each processor has to commuinicate with 6 neighbours rather than 4, while the messages are in addition longer. On the other hand, on the considered machine, the physical processor grid is a two dimensional one in which

PROBLEM 4

h^{-1}	32		64		128	
# proc.	time	(#it)	time	(#it)	time	(#(it)
1	62.2	(21)	-	(31)	-	(45)
8	7.83	(18)	93.8	(28)	-	
64	2.20	(19)	18.1	(29)	198.	(49)
512	1.01	(24)	5.99	(43)	47.9	(69)
Jacobi, 512 proc.	1.95	(63)	12.8	(127)	131.	(259)

PROBLEM 5

h^{-1}	32		64		128	
# proc.	time	(#it)	time	(#it)	time	(#(it)
1	127.	(37)	-	(55)	-	(81)
8	17.0	(33)	182.	(50)	-	
64	4.15	(36)	36.0	(58)	367.	(91)
512	1.88	(45)	9.89	(70)	75.5	(108)
Jacobi, 512 proc.	4.91	(156)	32.6	(316)	326.	(639)

PROBLEM 6

h^{-1}	32		64		128	
# proc.	time	(#it)	time	(#it)	time	(#(it)
1	87.6	(27)	-	(41)	-	(62)
8	15.4	(30)	164.	(45)	-	
64	4.17	(36)	34.3	(55)	339.	(84)
512	1.71	(41)	9.08	(64)	67.2	(96)
Jacobi, 512 proc.	4.49	(143)	30.5	(293)	305.	(600)

Table 2: Solution time (in seconds) and number of iterations for 3D problems on Parsytec GCel

each processor is directly connected to only four others, so that part of the interprocesor communications concern non neighbour processors and are therefore slower.

However, these problem are inherent to 3D calculation and validate our choice to first and foremost minimize the communications.

The comparison with the Jacobi preconditioning raises the same comments as in 2D, except that here the saving of iterations is somewhat less spectacular, essentially because it is above all function of the mesh size which is more modest than in 2D examples.

5 Numerical results on IBM cluster

We performed also some tests on a 8 workstation IBM RS6000 cluster, using the public domain PVM software.

Here, each processor appears sufficiently powerful to run efficiently sequentially, while the communications remain relatively slow compared with the speed of floating point calculation. This represents generally a major limitation on such systems, and makes the comparison of a parallel program with a sequential execution somewhat challenging in spite of the modest number of processors.

For this comparison, we considered some of the problems referred in the preceding section, using the same stopping criterion, discretization scheme, approximate factorization method and solution process.

The results are given in Table 3. We always used zero initial approximation and square (2D Problems 1 and 2) or cubic (3D Problems 4 and 5) processor grids, except in 2D cases when $\#proc = 8$, where we used $p_x = 4$ and $p_y = 2$.

Looking at the CPU times, it turns out that the parallelization is especially interesting when the memory requirements of the sequential version reach the available in core memory.

This explains why, for the lagest dimensions, efficiencies are always greater than 1 !

On the other hand, efficiencies may appear relatively poor for the smallest load per processor, but one has to take into account that this concerns only cases where the program executes sequentially in less than a very few seconds, i.e. cases for which the interest of the parallelization is anyway doubtful.

Finally, we obtain efficiencies slightly less than 1 in intermediate cases, showing that our method allows a successful parallelization even below the limit where the sequential version suffers from the recourse to virtual memory. This conclusion could be strengthened if the memory per processor were higher; Indeed, we had here 32 MB per node, which is relatively small for such powerful processors and such memory intensive computations.

We also report on the real time needed to execute the program. Indeed, although it is subject to large variations from run to run, this is one of the parameter the user is interested in when parallelizing an application. Moreover, it appears that part of the costs inherent to the execution of the program are hidden when considering only the user CPU time, maybe because a process which is waiting for data is automatically unloaded even if no other program is running on the machine. So, the tendency to have a low use of CPU time when parallelizing with small load per processor on the one hand, or when

PROBLEM 1

h^{-1}	144			288			576		
# proc.	CPU	(#it)	%CPU	CPU	(#it)	%CPU	CPU	(#it)	%CPU
1	1.36	(38)	39%	7.92	(56)	38%	47.7	(82)	30%
4	.333	(32)	12%	1.92	(48)	21%	10.5	(72)	29%
8	.246	(35)	6%	1.09	(51)	16%	5.94	(76)	22%

PROBLEM 2

h^{-1}	144			288			576		
# proc.	CPU	(#it)	%CPU	CPU	(#it)	%CPU	CPU	(#it)	%CPU
1	2.20	(60)	25%	12.5	(88)	46%	76.8	(131)	44%
4	.635	(54)	17%	3.19	(79)	47%	16.8	(115)	70%
8	.514	(55)	8%	1.88	(82)	18%	9.41	(120)	39%

PROBLEM 4

h^{-1}	36			72		
# proc.	CPU	(#it)	%CPU	CPU	(#it)	%CPU
1	2.24	(23)	14%	96.9	(33)	4%
8	.321	(19)	12%	3.61	(31)	32%

PROBLEM 5

h^{-1}	36			72		
# proc.	CPU	(#it)	%CPU	CPU	(#it)	%CPU
1	4.35	(39)	44%	195.	(59)	2.7%
8	.644	(36)	13%	6.61	(54)	32%

Table 3: Solution time (in seconds) and number of iterations for Problems 1, 2, 4 and 5 on IBM cluster. %CPU is the ratio between the CPU time spent for the solution part (reported in the first column) and the total elapsed time during the execution of the whole program

resorting to virtual memory on the other hand, has been confimed by various runs.

This left unchanged our preceding conclusions, but makes them much more dramatic. For instance, in real time, the 8 processor version is always slower than the 4 processor one for 2D problems with $h^{-1} = 144$. On the other hand, for 3D problems whith $h^{-1} = 72$, this means that the sequential version requires actually, for Problem 4, 40min against 11sec for the 8 processor version, and, for Problem 5, more than 2 hours against 22 sec when parallelizing.

Acknowledgements

This work presents research results of the Belgian Incentive Program "Information Technology" - Computer Science of the future, initiated by the Belgian State - Prime Minister's Service - Federal Office for Scientific, Technical and Cultural Affairs (Contract No. IT/IF/14). The Scientific responsability is assumed by its author.

This work was also supported by IBM through a research contract between ULB and IBM.

We thank the Interdisciplinary Center for Computer Based Complex System Research, Amsterdam (IC^3A), for the facilities provided on their computer systems.

References

[1] S. Ashby, M. Holst, T. Manteuffel, and P. Saylor, *The role of the inner product in stopping criteria for the conjugate gradient method*, Tech. Rep. UCRL-JC-112586, Lawrence Livermore National Laboratory, Livermore, CA 94551, 1992.

[2] O. Axelsson and V. A. Barker, *Finite Element Solution of Boundary Value Problems. Theory and Computation*, Academic Press, New York, 1984.

[3] O. Axelsson and G. Lindskog, *On the eigenvalue distribution of a class of preconditioning methods*, Numer. Math., 48 (1986), pp. 479–498.

[4] P. Bastian and G. Horton, *Parallelization of robust multigrid methods: ILU factorization and frequency decomposition method*, SIAM J. Sci. Stat. Comput., 12 (1991), pp. 1457–1470.

[5] R. Beauwens and R. Wilmet, *Conditioning analysis of positive definite matrices by approximate factorizations*, J. Comput. Appl. Math., 26 (1989), pp. 257–269.

[6] T. Chan, C.-C. J. kuo, and C. Tong, *Parallel elliptic preconditioners: Fourier analysis and performance on the Connection Machine*, Comput. Phys. Comm., 53 (1989), pp. 237–252.

[7] M. Clauss and al., *PARIX 1.2, Software Documentation*, PARSYTEC Computer GmbH, Aachen, 1993.

[8] P. Concus, G. H. Golub, and G. Meurant, *Block preconditioning for the conjugate gradient method*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 220–252.

[9] I. S. Duff and G. A. Meurant, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[10] V. Eijkhout, *Analysis of parallel incomplete point factorizations*, Lin. Alg. Appl., 154-156 (1991), pp. 723–740.

[11] H. Elman and E. Agroń, *Ordering techniques for the preconditioned conjugate gradient method on parallel coomputers*, Comput. Phys. Comm., 53 (1989), pp. 253–269.

[12] S. Filippone, M. Marrone, and G. Radicati di Brozolo, *Parallel preconditioned conjugate-gradient type algorithms for general sparsity structures*, Journal of Computer Mathematics, (1992). to appear.

[13] M. M. Magolu, *Implementation of parallel block preconditionings on a transputer-based multiprocessor*, Tech. Rep. Report IT/IF/14-10, Université Libre de Bruxelles, 1993.

[14] P. Manneback and J. Qin, *Algorithmic on a distributed memory mimd computer: a case study*, in Proceedings of Transputers'92, M. B. et al., ed., Amsterdam, 1992, IOS Press, pp. 172–178.

[15] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[16] S. Nakamura, *Computational Methods in Engineering and Science*, John Wiley & Sons, New York, 1977.

[17] Y. Notay, *Résolution itérative de systèmes linéaires par factorisations approchées*, PhD thesis, Service de Métrologie Nucléaire, Université Libre de Bruxelles, Brussels, Belgium, 1991.

[18] Y. Notay, *A new incomplete factorization method*, in Incomplete Decomposition (ILU) - Algorithms, Theory and Applications, W. Hackbusch and G. Wittum, eds., vol. 41 of Notes on Numerical Fluid Mechanics, Vieweg, Braunschweig, 1993, pp. 103–112.

[19] Y. Notay, *On the parallelization of approximate factorization preconditioning*, Tech. Rep. Report IT/IF/14-12, Université Libre de Bruxelles, 1993.

[20] Y. Notay, *Ordering methods for approximate factorization preconditioning*, Tech. Rep. Report IT/IF/14-11, Université Libre de Bruxelles, 1993.

[21] Y. Notay, *DRIC : a dynamic version of the RIC method.* to appear, 1994.

[22] J. Ortega, *Orderings for conjugate gradient preconditionings*, SIAM J. Optimization, 1 (1990), pp. 565–582.

[23] G. Radicati di Brozolo and Y. Robert, *Parallel conjugate gradient-like algorithms for solving sparse nonsymmetric systems on a vector multiprocessor*, Parallel Computing, 11 (1989), pp. 223–239.