Parallel incomplete factorizations with pseudo-overlapped subdomains

Mardochée Magolu monga Made [a,1] Henk A. van der Vorst [b]

[a] *Université Libre de Bruxelles, Service des Milieux Continus (CP 194/5), 50, avenue F.D. Roosevelt, B-1050 Brussels, Belgium. magolu@ulb.ac.be*

[b] *Utrecht University, Mathematical Institute, Mailbox 80.010, 3508 Utrecht, The Netherlands. vorst@math.uu.nl*

Abstract

We address the hard question of efficient use on parallel platforms, of incomplete factorization preconditioning techniques for solving large and sparse linear systems by Krylov subspace methods. A novel parallelization strategy based on pseudo-overlapped subdomains is explored. This results in efficient parallelizable preconditioners. Numerical results give evidence that high performance can be achieved.

Key words: Large sparse linear systems; incomplete factorizations; preconditioned conjugate gradient; multiprocessor computers; domain decomposition

1 Introduction

Combined with suitable preconditioners, Krylov subspace methods can be powerful (iterative) methods for solving the large sparse linear systems that arise in many scientific computations [6,23]. In particular, incomplete factorizations as preconditioning techniques are often efficient [31,32]. Their major drawback is that they are not easy to parallelize without seriously affecting the convergence. Several attempts have been reported in the literature, including reordering strategies, see, e.g., [1,4,8,11,16,20,21,30,38,45–47,49], domain decomposition type approaches [9,10,22,25,27,36,41,42], and truncated Neumann series approaches, [44,3,48]. This reflects the difficulty of the task. Recent surveys of techniques for achieving parallelism may be found in [13,17].

[1] Research supported by the Commission of the European Communities, within ESPRIT IV project, under contract nr. 25009.

We aim at designing a new and more efficient parallelization strategy. We particularize an improved version of the parallel block method proposed in [27] to the pointwise incomplete factorization preconditionings. Our approach may be seen as a generalized domain decomposition (DD) method. If necessary, it may be implemented as a (global) re-ordering technique. In contrast to classical DD methods, communication between adjacent subdomains is required during the construction and during the application of the preconditioner. A special treatment of the interface gridpoints allows to alleviate the significant decrease of the convergence rate that is characteristic for DD methods and for most of the orderings that have been suggested for general parallel computations (see, e.g., [16,14]).

Our exposition is organized as follows. In Section 2, we give a brief overview of our terminology and notation. Section 3 consists of background material, including a description of the preconditioned conjugate gradient (PCG) method, and a description of the generalized incomplete factorization preconditioner. In Section 4, we introduce and motivate our parallelization approach. Results of numerical experiments are reported in Section 5. Section 6 summarizes some concluding remarks and future directions for investigation.

2 Terminology and notation

2.1 Stieltjes matrices

A real square matrix A is called a *Stieltjes matrix* (or equivalently, a *symmetric M-matrix*) if it is symmetric positive definite and none of its offdiagonal entries is positive (see, e.g., [43]).

2.2 Miscellaneous symbols

Our matrices will be real, square and nonsingular, and of order n. We use A^t to denote the transpose of A, and $diag(A)$ denotes the diagonal matrix whose diagonal entries coincide with those of A.

Two gridpoints i and j are *connected*, with respect to the graph of A, if $a_{i,j} \neq 0$ or $a_{j,i} \neq 0$.

The symbol $\mathbf{e}$ represents the vector with all components equal to 1.

2.3 LPL^t-factorization

By the LPL^t *factorization* of a nonsingular Stieltjes matrix S we understand the (complete) factorization $S = L_s P_s L_s^t$ where P_s is a diagonal matrix while L_s is a lower triangular matrix such that $diag(L_s) = I$.

3 Background

For illustration purposes, we consider the following self-adjoint second order two-dimensional elliptic PDE

$$\begin{aligned} -p\,u_{xx} - q\,u_{yy} + t\,u &= f(x,y) \qquad && \text{in } \ \Omega = (0,1)\times(0,1) \\ u &= 0 && \text{on } \ , \\ u_n &= 0 && \text{on } \ \partial\Omega\backslash, \end{aligned} \tag{1}$$

where , denotes a portion of the boundary $\partial\Omega$ of Ω. We assume that if $t = 0$ then , $\neq \emptyset$. The coefficients p and q are positive, bounded and piecewise constant, and t is nonnegative, bounded and piecewise constant. We discretize (1) over a uniform rectangular grid of mesh size h in both directions with the five-point point box integration scheme [34]. The mesh points are ordered lexicographically in the (x, y)-plane, that is, starting from (or near) the origin $(x = 0, y = 0)$ and counting first in the x-direction. The matrix of the resulting linear system

$$Au = b \tag{2}$$

is a block-tridiagonal, irreducibly diagonally dominant, nonsingular Stieltjes matrix. In this case, PCG with an incomplete factorization as preconditioning is a popular solution method. For completeness, we represent the PCG algorithm in Fig. 1. The preconditioning matrix B is selected as the generalized relaxed incomplete LPL^t factorization described in Fig. 2. The set $\mathcal{D}$ specifies where fill-in entries have to be ignored, while the ρ_j are the relaxation parameters: $\rho_j = \rho$, $-\infty < \rho \leq 1$. This corresponds to the relaxed method [5], which includes the standard incomplete Cholesky factorization ($\rho = 0$) [31,32], as well as the classical modified variant ($\rho = 1$), for which $B\mathbf{e} = A\mathbf{e}$ [18,24]. The variables ρ_j encompass dynamically relaxed methods [7,35,28].

Two basic strategies for accepting or discarding fill-in have been developped.

1. $r^{(0)} := b - A\,u^{(0)}$
2. For $i = 1, 2, \ldots$ (until convergence)
3. Solve $w^{(i)}$ from
   $Bw^{(i)} := r^{(i)}$
4. $\gamma_i := (w^{(i)}, r^{(i)})$
5. $\beta_i := \begin{cases} 0 & \text{if} \quad i = 0 \\ \frac{\gamma_i}{\gamma_{i-1}} & \text{otherwise} \end{cases}$
6. $p^{(i)} := w^{(i)} + \beta_i\, p^{(i-1)}$
7. $w^{(i)} := A\, p^{(i)}$
8. $\alpha_i := \frac{\gamma_i}{(p^{(i)}, w^{(i)})}$
9. $u^{(i+1)} := u^{(i)} + \alpha_i\, p^{(i)}$
10. $r^{(i+1)} := r^{(i)} - \alpha_i\, w^{(i)}$
11. If satisfied Stop

Fig. 1. Preconditioned conjugate gradient method.

(1) **Level fill**. The level $lev(l_{k,i})$ of the coefficient $l_{k,i}$ of L is defined by (see Fig. 2 for notation),

Initialization

$$lev(l_{k,i}) := \begin{cases} 0 & \text{if} \;\; l_{k,i} \neq 0 \;\; \text{or} \;\; k = i \\ \infty & \text{otherwise} \end{cases}$$

Factorization

$$lev(l_{k,i}) := \min\{\, lev(l_{k,i})\,,\, lev(l_{i,j}) + lev(l_{k,j}) + 1 \,\} \;\; .$$

$$\mathcal{D} = \{\, (k,i) \mid lev(l_{k,i}) > \ell \,\} \;\; .$$

where integer ℓ stands for a user specified maximal fill-in level [39].

(2) **Drop-tolerance**. Fill-in is ignored if it is “too small” according to some prescribed tolerance (see, e.g., [33,13]).

Hybrid approaches that combine (1) and (2) are discussed in, a.o., [39]. There is no generally accepted strategy that is a panacea for a wide class of problems of the type (1). An adequate choice of ℓ or the drop tolerance depends on the specific problem at hand and the workspace available. Selection of

Compute P **and** L $(B = LPL^t \quad \text{with} \quad diag(L) = I)$

Initialization phase

$p_{i,i} := a_{i,i}\ , \quad i = 1, 2, \cdots, n$

$l_{i,j} := a_{i,j}\ , \quad i = 2, 3, \cdots, n\ , \qquad j = 1, 2, \cdots, i-1$

Incomplete factorization process

do $j = 1, 2, \cdots, n-1$

  compute parameter ρ_j

  do $i = j+1, j+2, \cdots, n$

    $l_{i,i} := l_{i,i} - \frac{l_{i,j}^2}{l_{j,j}}$

    $l_{i,j} := \frac{l_{i,j}}{l_{j,j}}$

    do $k = i+1, i+2, \cdots, n$

      if $(k,i) \notin \mathcal{D} \quad l_{k,i} := l_{k,i} - l_{i,j}\, l_{k,j}$

      otherwise

$$\begin{cases} l_{i,i} := l_{i,i} - \rho_j\, l_{i,j}\, l_{k,j} \\ l_{k,k} := l_{k,k} - \rho_j\, l_{i,j}\, l_{k,j} \end{cases}$$

    end do

  end do

end do

Fig. 2. Generalized relaxed incomplete factorization (GRIC).

these parameters is an art rather than a science. As is well known, potential bottlenecks for PCG methods, as described above, are the construction of the preconditioner B and the preconditioning step at each PCG iteration (Step 3), see e.g. [46].

In our analysis, we shall make use of GRIC with level fill ℓ which, according to [31], is denoted by $GRIC(\ell)$. **Observe that any node j that is connected, with respect to the graph of L, with two nodes i and k such that $j < i < k$ gives rise to a fill-in element in position (k,i) of L, if $\ell \geq 1$.**

To solve a linear system of the form $LPL^t w = r$, that occurs at each PCG iteration (step 3 on Fig. 1), one may proceed with the two steps as described in Fig. 3. The construction of $GRIC(\ell)$ and the preconditioning step involve recurrence relations that inhibit efficient parallel computation, most notably for lexicographical ordering.

Solve $LPL^t w = r$ **for** w

• **Forward solve** (v from $Lv = r$)

$v_i := r_i\ ,\quad i = 1, 2, \cdots, n$
do $j = 1, 2, \cdots, n-1$
  do $i = j+1, j+2, \cdots, n$
    $v_i := v_i - l_{i,j} v_j$
  end do
end do

• **Backward solve** (w from $L^t w = P^{-1} v$)

$w_i := \frac{1}{p_{i,i}} v_i\ ,\quad i = 1, 2, \cdots, n$
do $j = n, n-1, \cdots, 2$
  do $i = j-1, j-2, \cdots, 1$
    $w_i := w_i - l_{j,i} w_j$
  end do
end do

Fig. 3. Solution of the preconditioning system.

4 ParGRIC : A family of parallel incomplete factorizations

4.1 Motivation

We will first consider GRIC(0), in which the sparsity structure of A is preserved. In Fig. 4, the graph of A is depicted with a stencil graph notation [28] : a diagonal entry $a_{i,i}$ is represented by circle number i; the edge $\{i, j\}$ (here, thin lines) corresponds to a nonzero offdiagonal entry $a_{i,j}$. Oblique thick lines represent the discarded level 1 fill-in entries that determine the remainder matrix $R = B - A$. The smaller $\|R\|$, the faster the convergence. The values n_x and n_y denote the number of unknowns in x and y direction, respectively. In Fig. 4, we have taken $n_x = n_y = 5$. Except for the boundary nodes where a Dirichlet boundary condition holds, the graph of A relates directly to the discretization grid.

For simplicity, the domain will be partitioned into stripes : p rectangular boxes that are assigned to p processors as depicted in Fig. 5 for $p = 6$.

Let us consider now, in Fig. 6, a portion of the graph of A, in which two adjacent subdomains are assigned to two processors ($\mathcal{P}_s$ and $\mathcal{P}_{s+1}$). We will

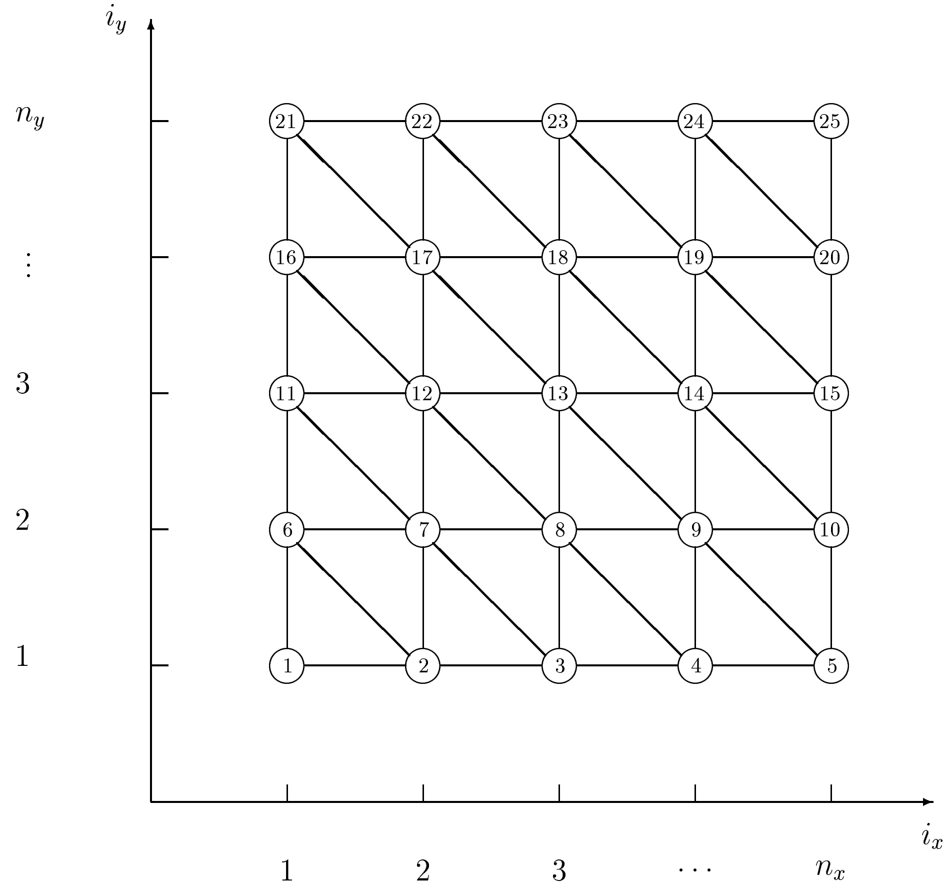


Fig. 4. Graph of matrix A (in thin lines). The jth node is $j = (i_y - 1)n_x + i_x$. Thick (oblique) lines correspond to level 1 fill-in entries.

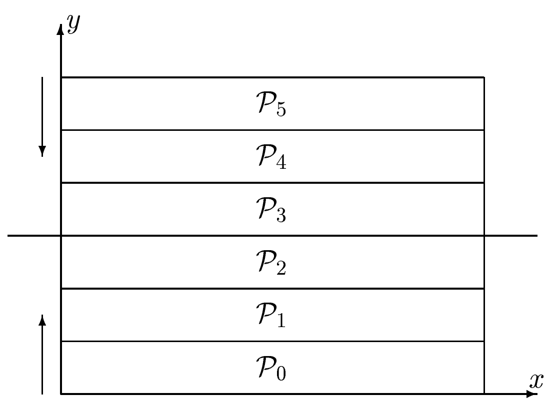


Fig. 5. Partitioning of the grid into stripes for 6 subdomains, $\mathcal{P}_i$, $i = 0, 1, \ldots, 5$. Vertical arrows indicate the flow of computation within each subdomain.

impose the following five conditions (see Figs. 6 and 7 for illustration):

(c1) processor $\mathcal{P}_{s+1}$ starts its computations at gridpoints "$\star$" (the "bottom layer" of $\mathcal{P}_{s+1}$) skipping the correction from gridpoints "$\bullet$" (the "top layer" of $\mathcal{P}_s$);

(c2) immediately after the computations at the bottom layer gridpoints of $\mathcal{P}_{s+1}$

have been completed, the relevant corrections from $\mathcal{P}_{s+1}$ for the top layer gridpoints of $\mathcal{P}_s$ can be sent to $\mathcal{P}_s$ (but these points have to wait for the final update when all other points of $\mathcal{P}_s$ have been completed);

(c3) the actual computations start from two sides: for the subdomains in the upper side of the physical domain the bottom layer and the top layer reverses (see Fig. 5);

(c4) for each subdomain the computation starts at the bottom layer gridpoints (and they have been handled before any other gridpoint) and finishes at the top layer grid points;

(c5) the numbering decreases or increases in the same way for neighbouring points, for the bottom layer gridpoints of $\mathcal{P}_{s+1}$ and the top layer gridpoints of $\mathcal{P}_s$ (compatible nunbering). This facilitates the implementation (communication). Each gridpoint at the top layer has "to know" where corrections come from.

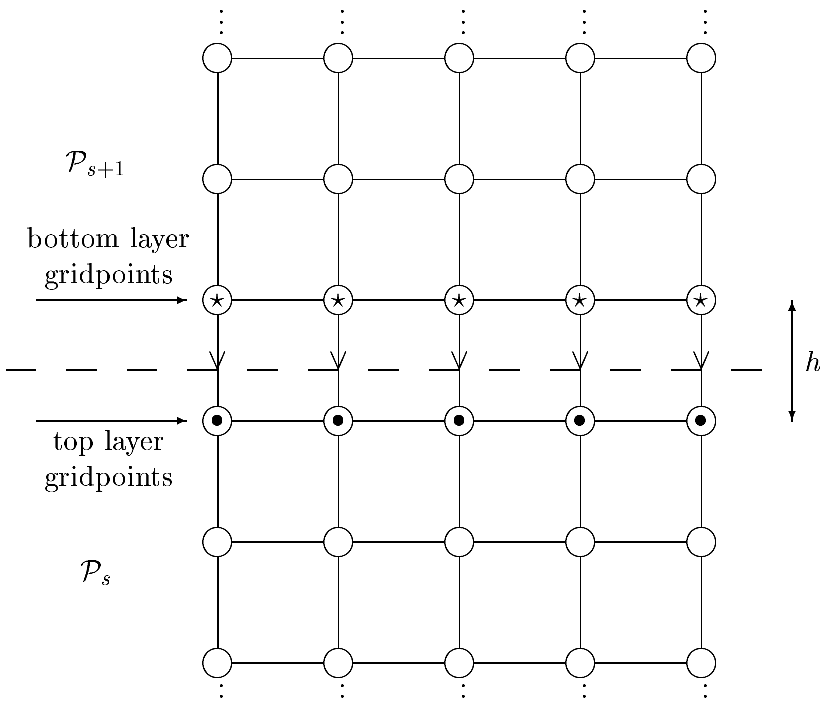


Fig. 6. Part of graph of matrix A assigned to two different processors ($\mathcal{P}_s$ and $\mathcal{P}_{s+1}$); the pseudo-overlap width is equal to h.

Condition (c1) means that, according to some implicit global ordering, all the bottom layer gridpoints "$\star$" have to be handled (numbered) prior to all the neighbouring gridpoints: so these must wait for the contribution from bottom layer gridpoints before being updated.

We introduce the following terminology.

Definition 1 *Since communication only involves the gridpoints in the bottom and top layer, we will call the union the pseudo-overlap. Equivalently, we will* *say that $\mathcal{P}_s$ is pseudo-overlapped by $\mathcal{P}_{s+1}$.*

The trouble with any parallelization technique, that (implicitly) resorts to a reordering strategy like ours, is that the convergence properties of PCG usually deteriorate as the number of subdomains increases, see, e.g., [25,27,36,37]. In order to get some feeling why this happens, let us examine the remainder matrix R. For this purpose, we add the (rejected) level-1 fill-in entries to the partial graph of Fig. 6. This gives Fig. 7, where the part relative to the original graph of A, as well as level-1 fill-in entries that are not significantly different from the case of one subdomain (Fig. 4) are drawn in thin lines. Thick lines (the arcs), that connect top layer gridpoints to gridpoints marked with "$\diamond$", correspond to the (neglected) fill-in entries that are mainly responsible for the degradation of the convergence. Observe that the number of such entries increases with the number of subdomains.

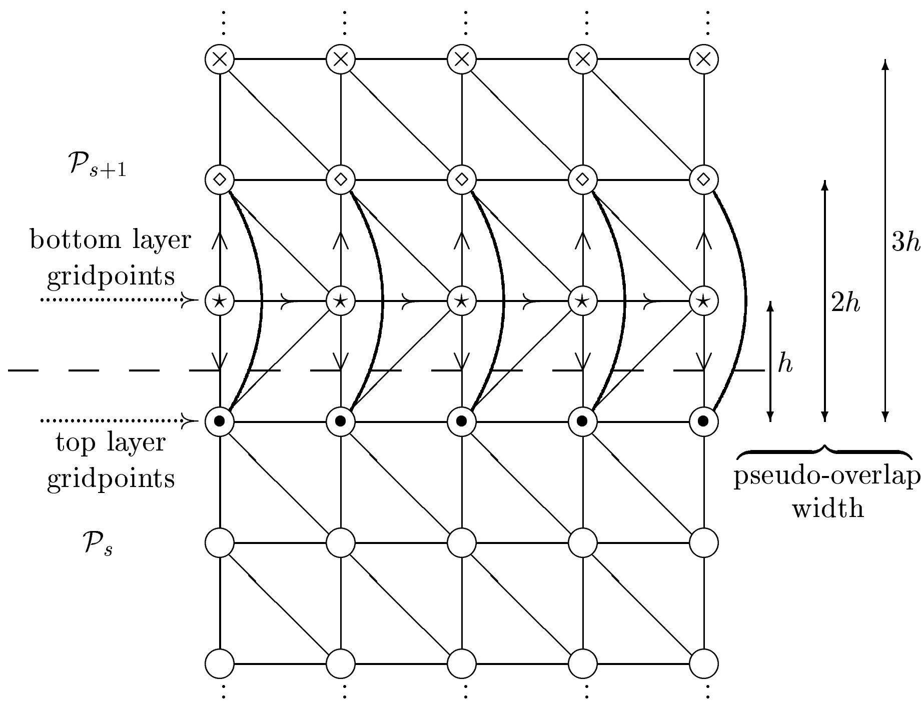


Fig. 7. Part of graph of matrix A assigned to two different processors ($\mathcal{P}_s$ and $\mathcal{P}_t$). Oblique lines and thick lines are (neglected) level 1 fill-in entries.

Accepting all the fill-in entries (of any level) that are induced by the parallel ordering will avoid to deteriorate the PCG convergence, but unfortunately, this will also prevent the processors from performing efficiently in parallel. Going back to the incomplete factorization philosophy [31], we will content ourselves with weakening the influence of the neglected fill-in by increasing the pseudo-overlap width (ϖ), as well as the fill-in level inside the pseudo-overlapping region. In Fig. 7, this means that the gridpoints marked with "$\diamond$" are included in the bottom layer for $\mathcal{P}_{s+1}$ (in which case $\varpi = 2h$). The bottom layer should comply with our requirements (c1)-(c4). In the terminology of Doi and Lichnewsky [12] (see also Doi and Washio [14]), we make an attempt to reduce the number of *incompatible nodes* (marked with "$\star$" in Fig. 7).

Definition 2 *Any GRIC preconditioner combined with our parallelization strategy is denoted by ParGRIC($\ell; \varpi, \ell_\varpi$), which reads as parallel generalized relaxed incomplete Cholesky factorization with pseudo-overlap width ϖ; ℓ_ϖ stands for the fill-in level in the pseudo-overlapping regions, and ℓ stands for the fill-in level in the remaining part of subdomains.*

In the specification of ϖ, the actual mesh size h will be dropped, say, k will stand for kh, in order to include variable mesh size problems and (graphs of) matrices that do not arise from discretized PDEs.

Remark 1 Under Condition (c5), and in contrast to the level zero parallel preconditionings discussed in [25,36], we are able to easily consider any fill-in level in the incomplete factorization schemes:

1. during the symbolic incomplete factorization phase, neighbouring subdomains may readily determine the (same) quantity and structure of information that they need to send or receive;
2. during the numeric incomplete factorization phase, each pseudo-overlapping subdomain should pack the information needed, fill-in contributions included, in a vector whose length has been computed during the symbolic incomplete factorization step.

We stress that in most realistic problems, level zero incomplete factorization methods are seldomly efficient. In particular, on parallel architectures, classical overlapping (or non-overlapping) domain decomposition methods, that combine ingredients of both direct methods (as local solver) and iterative methods (as global solver), are in general more competitive. See, e.g., [39,40,15].

4.2 Illustration

We assume, for ease of presentation, that the number p of subdomains $\mathcal{P}_j$, $j = 0, 1, \ldots, p-1$, is even. The two-sided handling of the subdomains is indicated in Fi.g 8 by arrows at the left. Within each subdomain, row-wise numbering is used. The pseudo-overlapping regions are marked with "$\times \cdots \times$". To sum up :

- $\mathcal{P}_i$ pseudo-overlaps $\mathcal{P}_{i-1}$ for $i = 1, 2, \ldots, \frac{p}{2} - 1$;
- $\mathcal{P}_j$ pseudo-overlaps $\mathcal{P}_{j+1}$ for $j = \frac{p}{2}, \frac{p}{2} + 1, \ldots, p-2$;
- $\mathcal{P}_{\frac{p}{2}}$ is pseudo-overlapped by $\mathcal{P}_{\frac{p}{2}-1}$ with $\varpi = 1$.

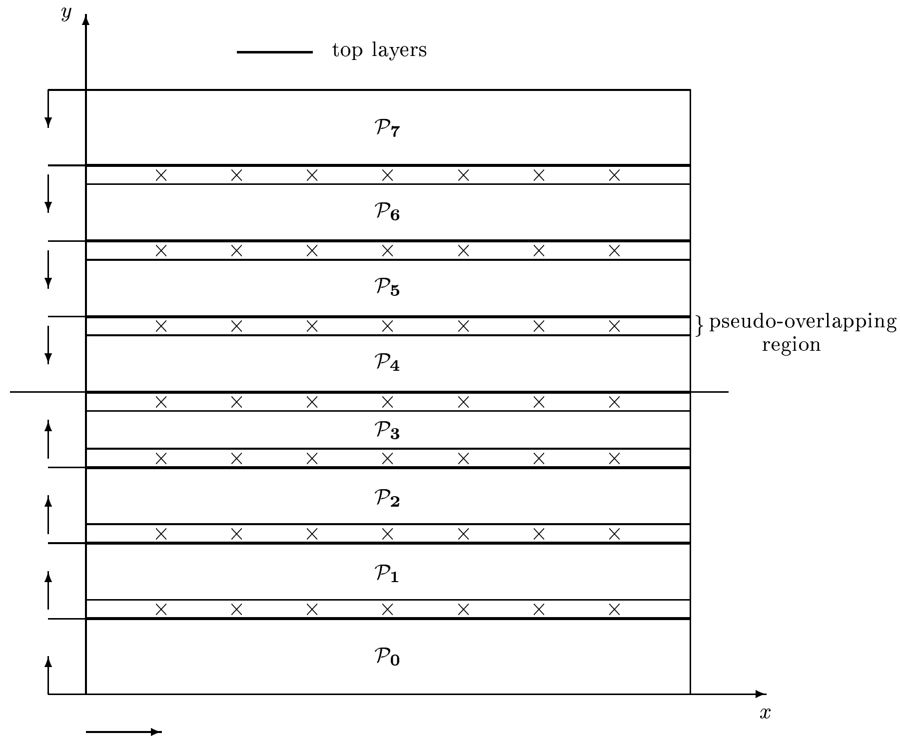


Fig. 8. Specification of pseudo-overlapping regions for $p = 8$. Vertical arrows indicate the progressing direction of subdomain local numbering along the y-axis. Within each horizontal line, grid points are ordered rightwards.

All the processors contain (approximately) the same number of horizontal (grid) lines. It is obvious that for all the tasks involving preconditioning, data dependency occurs only at the interfaces between the subdomains.

Remark 2 The partitionings depicted in Figs. 5 and 8 are not the optimal ones whenever the number of subdomains is larger than three, unless the original physical domain is elongated in the y-direction, or equivalently, when the number of unknowns along the y-direction is fairly larger than the number of unknowns along the x-direction. As already mentioned, our stripe partitionings are only used for simplicity, in order to illustrate how pseudo-overlapping could improve the convergence rate. For more or less symmetric regions, it would be better to split domains also in the x-direction.

5 Numerical results

As illustrative examples, we consider the following three problems that are particular cases of PDE (1):

Problem 1 $p = q = 1$, $t = 0$, , $= \Omega$ *and* $u(x,y) = x(x-1)y(y-1)e^{xy}$; $h = 1/(n_y + 1)$.

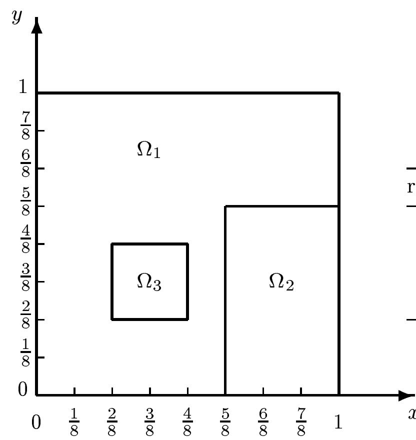
Problem 2 , $= \{(x,y);\ 0 \le x \le 1,\ y = 0\}$, $t = 0$,

$$p = q = \begin{cases} 100 \text{ in } (1/4, 3/4) \times (1/4, 3/4) \\ \\ 1 \quad \text{elsewhere} \end{cases}$$

$$f(x,y) = \begin{cases} 100 \text{ in } (1/4, 3/4) \times (1/4, 3/4) \\ \\ 0 \quad \text{elsewhere} \end{cases}$$

Here $h = 1/n_y$, *where* n_y *is a multiple of* 4 *(in order to avoid problems at discontinuities of the PDE coefficients).*

Problem 3 , $= \emptyset$, *the coefficients* p, q, *and* t *are specified in Fig. 9. One has* $h = 1/(n_y - 1)$. *For simplicity* $n_y - 1$ *is taken as a multiple of* 8. *The right-hand side of the linear system is chosen such that the function* $u_0(x,y) = x(1-x)y(1-y)e^{xy}$ *generates the solution on the grid.*



region	p	q	t
Ω_1	1	1	0
Ω_2	1	10^3	0
Ω_3	10^3	1	1

Fig. 9. Problem 3. Configuration and specification of the PDE coefficients.

The PCG algorithm is executed with the zero vector as initial approximation, and the relative residual error $\|r^{(i)}\|_2 \,/\, \|r^{(0)}\|_2 \le 10^{-6}$ as convergence criterion. To save computer time, we first work with the preconditioned residual, till $\sqrt{\gamma_i/\gamma_0} \le 10^{-6}$ is satisfied (see Fig. 1 for the definition of γ_i); then we start checking whether the true residual is also sufficiently reduced. This check requires computing an additional inner product. We have opted for non blocking communications, which enables us to overlap computations with communications, whenever possible [13,17]. The computations are carried out in double precision Fortran on a 16-processor SGI Origin 2000 (4 Gbytes memory, 32 KBytes Data Cache, 195 MHz MIPS Processor), using the MPI library for interprocessor communications. The preconditionings include :

(1) **ParIC($\ell; \varpi, \ell_\varpi$)** : the standard incomplete Cholesky ($\ell_\varpi \geq \varpi - 1$);
(2) **AS($\ell; \varpi$)** : The additive Schwarz with overlap ([40]). Each local problem is handled with one IC(ℓ) solve, ℓ denotes the fill-in level. ϖ stands here for the actual overlap width. We use $\varpi = h_0,\ h,\ 2h$, where h_0 means that only one line of nodes is shared by the neighbouring subdomains.

For simplicity, no global coarse grid correction has been added to improve the performance of the preconditionings involved (such global corrections have been advocated in [37,40]). It is worthwhile to note that ParMIC($\ell; \varpi, \ell_\varpi$), that is the parallel version of the classical modified incomplete Cholesky factorization, should not be used without perturbations to the diagonal. This is necessary to avoid singular preconditioners [16,19]. These perturbations (of low order in the gridsize) are discussed in [18,24,7].

Experiment 1 : In order to see how pseudo-overlapping reduces the negative influence of parallel orderings on the convergence rate, we run ParIC($0; \varpi, \varpi - 1$), and we let ϖ vary from 1 to 8. It appears that, the more difficult the problem is (or the larger its size), the bigger is the advantage of increased pseudo-overlap. By way of illustration, we report in Fig. 10 the case of 8 subdomains, for Problem 1 with $h^{-1} = 129$, Problem 2 with $h^{-1} = 128$ and Problem 3 with $h^{-1} = 128$.

Experiment 2 : For both preconditioners, we have observed that fill-in level $\ell = 4$ is in general efficient, in the sense that it minimizes the overall elapsed time on a quiet system (only one user). We collect in Tables 1–3, and Fig. 11, the performances for ParIC($0; 1, 0$), ParIC($4; 5, 4$), and AS($4; \varpi$). We use the parallel speed-up, which is defined as the ratio between the execution time of the parallel algorithm on one processor and the time taken by the same algorithm on p processors. For $p = 1$ the parallel code is, except for some negligible overhead for checking of parameters, equivalent to the serial process with incomplete Cholesky preconditioning.

Note that in the context of parallel incomplete factorization based methods, the preconditioning changes with the number of subdomains. This together with our definition of speed-up, may explain why in some cases, the actual speed-up observed is larger than the number of processors. The following trends are evident.

(1) ParIC($4; 5, 4$) is in general twice as fast as ParIC($0; 1, 0$), but the latter exhibits a slightly better speed-up. In our experiments, it has proved to be advantageous to take into account (some) fill-in entries induced by the parallelization (reordering) strategy. In this respect, we emphasize that

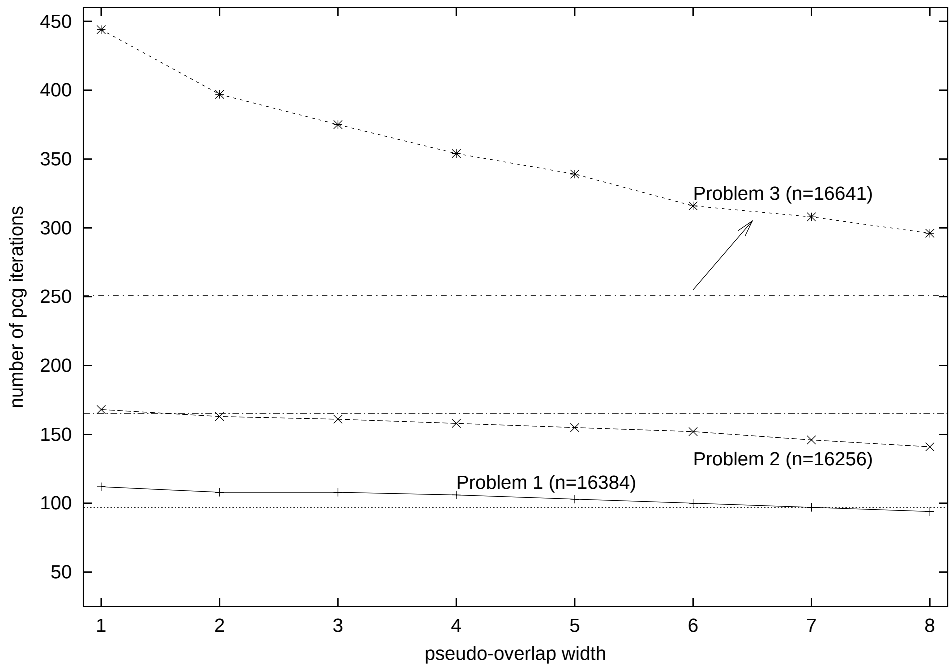


Fig. 10. Effects of pseudo-overlap width ϖ on the number of pcg iterations, for 8 processors and ParIC$(0;\varpi,\varpi-1)$. Horizontal (non grid) lines display the number of pcg iterations for IC(0) on 1 processor.

Table 1
Problem 1. $h^{-1} = 513$; $n = 262144$. Number of PCG iterations (iter.); elapsed time in seconds for: the computation of the preconditioning matrix (fact.), the solver, and overall time; speed-up, for np processors.

Precond.	np	iter.	Time fact.	Time pcg	Time overall	overall speed-up
ParIC(0;1,0)	1	398	0.16	191.31	192.40	1.00
	2	398	0.14	93.50	94.13	2.04
	4	435	0.07	41.64	41.95	4.57
	8	437	0.03	17.87	18.05	10.66
	16	440	0.02	9.46	9.65	19.94
ParIC(4;5,4)	1	122	4.76	80.59	86.20	1.00
	2	122	2.48	42.84	45.82	1.88
	4	128	1.24	19.30	20.78	4.15
	8	131	0.65	8.50	9.32	9.24
	16	137	0.37	4.26	4.80	17.96
AS(4,h_0)	2	171	1.98	62.56	65.09	1.32
	4	179	0.94	26.85	28.01	3.08
	8	180	0.48	11.38	11.98	7.19
	16	197	0.24	6.24	6.57	13.12
AS(4,h)	2	163	1.90	52.85	55.15	1.56
	4	167	0.94	23.06	24.20	3.56
	8	172	0.48	10.54	11.14	7.38
	16	181	0.24	5.40	5.75	14.99
AS(4,$2h$)	2	161	1.89	52.41	54.71	1.58
	4	164	0.96	23.04	24.21	3.56
	8	165	0.52	11.18	11.82	7.29
	16	175	0.25	5.17	5.57	15.46

Table 2

Problem 2. $h^{-1} = 512$; $n = 262656$. Number of PCG iterations (iter.); elapsed time in seconds for: the computation of the preconditioning matrix (fact.), the solver, and overall time; speed-up, for np processors..

Precond.	np	iter.	Time fact.	Time pcg	Time overall	overall speed-up
ParIC(0;1,0)	1	628	0.25	254.88	255.73	1.00
	2	628	0.13	128.52	129.01	1.98
	4	638	0.07	53.80	54.06	4.73
	8	641	0.03	23.13	23.27	10.99
	16	644	0.02	12.83	13.01	19.66
ParIC(4;5,4)	1	185	4.84	106.17	111.61	1.00
	2	187	2.48	60.60	63.45	1.76
	4	200	1.22	27.48	28.88	3.86
	8	205	0.62	12.60	13.33	8.37
	16	219	0.33	6.49	7.01	15.92
AS(4,h_0)	2	257	1.88	83.48	85.66	1.30
	4	257	0.95	36.59	37.68	2.96
	8	274	0.49	16.70	17.27	6.46
	16	300	0.24	8.81	9.14	12.21
AS(4,h)	2	252	1.88	82.37	84.55	1.32
	4	248	0.96	34.79	35.89	3.11
	8	258	0.50	16.52	17.10	6.53
	16	277	0.25	8.99	9.33	11.96
AS(4,2h)	2	245	1.89	80.53	82.71	1.35
	4	249	0.97	35.13	36.24	3.08
	8	256	0.51	18.69	19.30	5.78
	16	268	0.25	8.71	9.10	12.26

Table 3

Problem 3. $h^{-1} = 512$; $n = 263169$. Number of PCG iterations (iter.); elapsed time in seconds for: the computation of the preconditioning matrix (fact.), the solver, and overall time; speed-up, for np processors.

Precond.	np	iter.	Time fact.	Time pcg	Time overall	overall speed-up
ParIC(0;1,0)	1	1075	0.25	438.21	439.30	1.00
	2	1076	0.13	224.85	225.47	1.95
	4	1381	0.04	116.97	117.22	3.74
	8	1386	0.03	51.01	51.20	8.58
	16	1638	0.03	33.93	34.04	12.91
ParIC(4;5,4)	1	325	4.77	187.52	193.12	1.00
	2	328	2.47	106.26	109.22	1.77
	4	456	1.21	62.73	64.19	3.01
	8	541	0.61	32.68	33.46	5.77
	16	692	0.34	18.50	19.02	10.15
AS(4,h_0)	2	634	1.88	202.17	204.45	0.94
	4	731	0.94	100.72	101.87	1.90
	8	900	0.47	53.91	54.51	3.54
	16	1201	0.23	33.41	33.75	5.72
AS(4,h)	2	595	1.89	193.92	196.19	0.98
	4	688	0.94	95.42	96.57	2.00
	8	838	0.48	50.34	50.94	3.79
	16	1091	0.24	30.51	30.88	6.25
AS(4,2h)	2	567	1.88	183.68	185.95	1.04
	4	651	0.95	90.65	91.79	2.10
	8	787	0.49	48.00	48.61	3.97
	16	1008	0.24	28.60	29.07	6.64

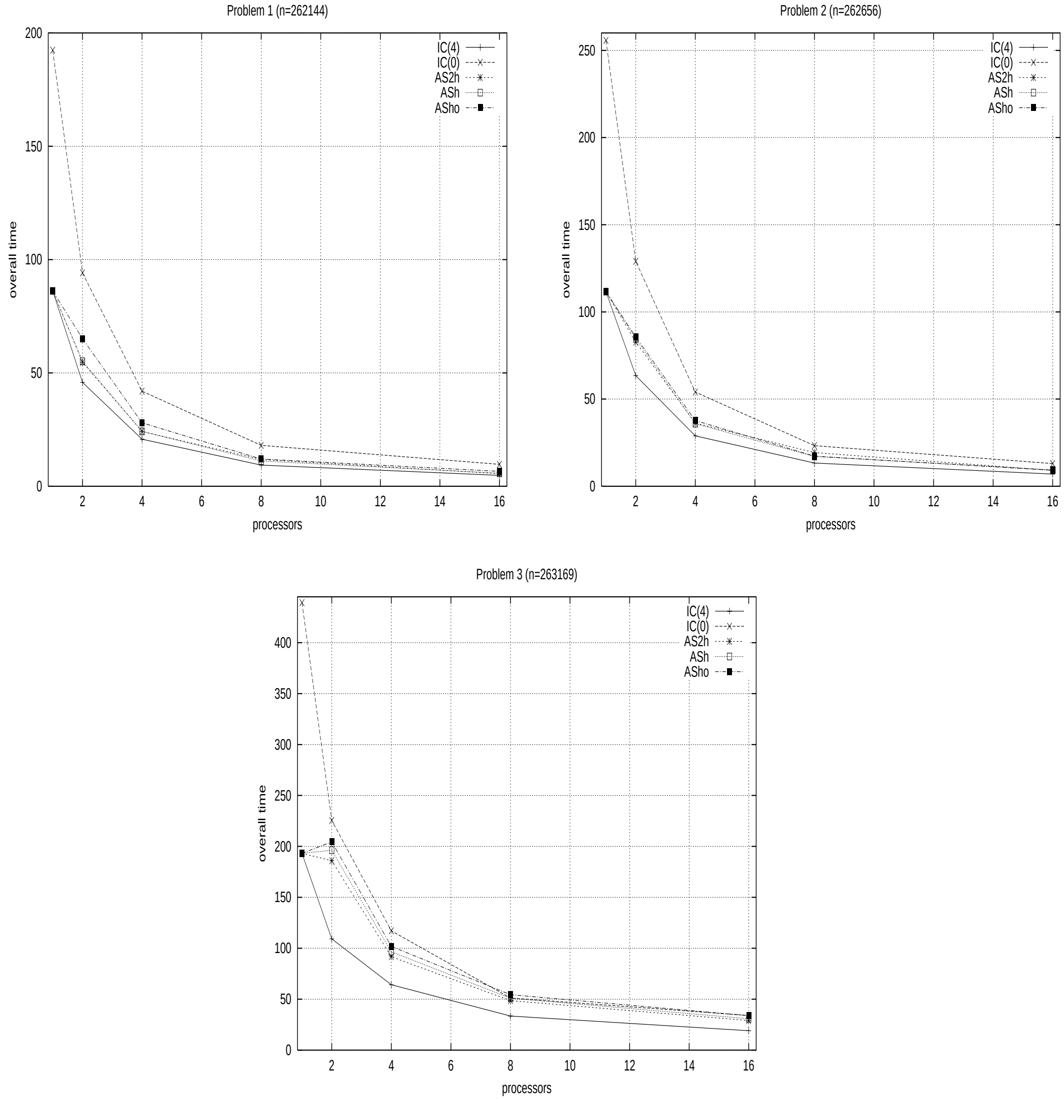


Fig. 11. Overall computational time for ParIC(0;1,0), ParIC(4;5,4), AS(4,h_0), AS(4,h) and AS(4,2h).

ParIC$(4;1,4)$, which applies locally the same level of fill as ParIC$(4;5,4)$ but discards any induced fill-in entry, gives rise to a poor performance (not reported here).

(2) In order to remain competitive with ParIC, the AS method must be applied with a sufficiently large overlap width, which dramatically increases the computational complexity. For Problem 3, $\varpi = 2h$ is no longer appropriate.

(3) For our test problems, ParIC$(4;5,4)$ emerges as the most efficient choice.

Experiment 3 : The rather low "optimal" fill-in level observed (in our case: 4) accounts for the fact that the linear system is solved only once. In the case of time-dependent PDEs, nonlinear problems, or strongly indefinite linear systems, higher fill-in levels may be better [40,15,29]. In such cases, the increase of the (incomplete) factorization cost is amortized by the decrease of the number of iterations. Even in this case, ParIC should be preferred over AS, as can be seen from Fig. 12. There we show the performance of AS($\infty; \varpi$) and ParIC($\infty; \varpi_{\max}, \infty$) for 8 and 16 processors. By $\varpi_{\max}$ we mean that all fill-in entries induced by the parallelization (renumbering) strategy are accepted, except those that connect any couple of mesh nodes that belong to two non-adjacent layers. In the case of Problem 3, the convergence suffers from the presence of *many* well separated eigenvalues near the origin, [28]. We note that for 2 processors, as well as for the VDV 4-processor orderings (see, [16], [45]), ParIC($\infty; \varpi_{\max}, \infty$) becomes a direct solver, whereas AS remains an iterative one.

6 Conclusions

We have first identified reasons why the performance of parallel incomplete factorizations deteriorates with increasing number of subdomains. To remedy this, we have designed a new family of robust variants, that compare favorably with the popular additive Schwarz (AS) method. A salient feature of our approach is that no overlap seems necessary. The performance may be improved by a proper choice of the (possibly variable) relaxation parameters ρ_i. Preliminary numerical experiments indicate that optimal values depend on the number of subdomains, in agreement with [36].

Our approach may be adapted to unstructured grids as well. This is relatively easy when the domain is (approximately) partitioned into stripes, or in such a way that each subdomain has a limited number of neighbours. In other cases, care should be taken to define some *logical* hierarchy between neighbouring subdomains. For instance, if there holds $i < j$ then processor i pseudo-overlaps processor j, or vice-versa. By "logical" we mean that deadlocks have to be avoided (that is when two or more processors wait for information from each other). A variant of our approach, with an ordering induced pseudo-overlapping strategy, that will help to tackle intricate geometries and partitionings, will be published elsewhere (after completion of all experiments).

Acknowledgement

Part of this work was done while the first author was holding a postdoctoral position at the Mathematical Institute of Utrecht University, under the Commission of the European Communities HCM Contract No. ERB-CHBG-CT93-0420.

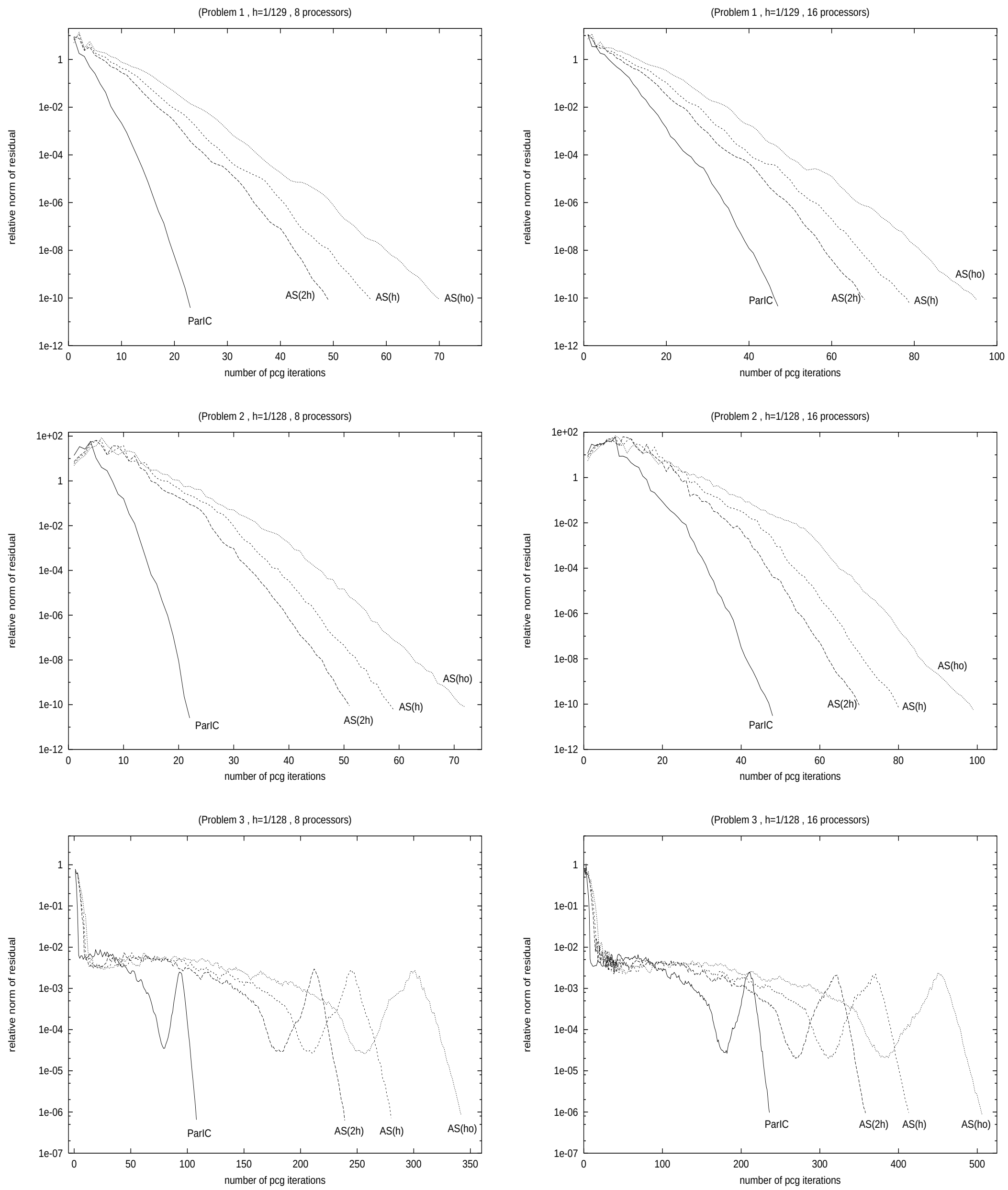


Fig. 12. Evolution of the relative residual error for 8 and 16 processors. The fill-in level $\ell = \infty$ (locally) for each preconditioner involved.

References

[1] C.C. Ashcraft and R.G. Grimes, On vectorizing incomplete factorization and SSOR preconditioners, *SIAM J. Sci. Stat. Comput.* 9 (1988), 122–151.

[2] O. Axelsson, *Iterative Solution Methods* (Cambridge University Press, Cambridge, 1994).

[3] O. Axelsson and V. Eijkhout, Vectorizable preconditioners for elliptic difference equations in three space dimensions, *J. Comput. Appl. Math.* 27 (1989) 299–321.

[4] O. Axelsson, G. Carey and G. Lindskog, On a class of preconditioned iterative methods on parallel computers, *Int. J. Numer. Meth. Engng.* 27 (1989) 637–654.

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

[6] R.F. Barret, M. Berry, T.F. Chan, J. Demmel, J. Donato, J.J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems : Building Blocks for Iterative Methods* (SIAM, Philadelphia, 1994).

[7] R. Beauwens, Modified incomplete factorization strategies, in : O. Axelsson and L. Kolotilina, eds., *Preconditioned Conjugate Gradient Methods* (Lectures Notes in Mathematics No. 1457, Springer-Verlag, Berlin, 1990) 1–16.

[8] R. Beauwens, L. Dujacquier, S. Hitimana and M. Magolu monga Made, MILU factorizations for 2-processor orderings, in : I.T. Dimov, Bl. Sendov and P.S. Vassilevski, eds., *Advances in Numerical Methods and Applications* $\mathcal{O}(h^3)$ (World Scientific, Singapore, 1994) 26–34.

[9] E. Brakkee, C. Vuik and P. Wesseling, An investigation of Schwarz domain decomposition using accurate and inaccurate solution of subdomains, Report 95-18, Faculty of Technical Mathematics and Informatics, Delft University of Technology, 1995.

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

[11] E.M. Daoudi and P. Manneback, Implementation of ICCG algorithm on distributed memory architecture, in : R. Beauwens and P. de Groen, eds., *Iterative Methods in Linear Algebra* (North-Holland, Amsterdam, 1992) 339–347.

[12] S. Doi and A. Lichnewsky: A graph-theory approach for analyzing the effects of ordering on ILU preconditioning. INRIA report 1452, INRIA-Rocquencourt, France, 1991.

[13] J.J. Dongarra, I.S. Duff, D.C. Sorensen and H.A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers* (SIAM, Philadelphia, 1998).

[14] S. Doi and T. Washio, Ordering strategies and related techniques to overcome the trade-off between parallelism and convergence in incomplete factorizations, *Parallel Comput.* 25 (1999) 1995–2014.

[15] I.S. Duff, Direct methods, Technical Report RAL-TR-1998-054, Rutherfore Appleton Laboratory, Chilton, Didcot, OX11 0QX, UK, 1998.

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

[17] I.S. Duff and H.A. van der Vorst, Developments and Trends in the Parallel Solution of Linear Systems, *Parallel Comput.* 25 (1999) 1931–1970.

[18] T. Dupont, R.P. Kendall, and H.H. Rachford, An approximate factorization procedure for solving self-adjoint elliptic difference equations, *SIAM J. Numer. Anal.* 5 (1968) 559–573.

[19] V. Eijkhout, Beware of unperturbed modified incomplete factorizations, in Iterative Methods, in : R. Beauwens and P. de Groen, eds., *Iterative Methods in Linear Algebra* (North-Holland, Amsterdam, 1992) 583–591.

[20] H. Elman and E. Agrón, Ordering techniques for the preconditioned conjugate gradient method on parallel computers, *Comput. Phys. Comm.* 53 (1989) 253-269.

[21] S. Fujino and S. Doi, Optimizing multicolor ICCG methods on some vectorcomputers, in : R. Beauwens and P. de Groen, eds., *Iterative Methods in Linear Algebra* (North-Holland, Amsterdam, 1992) 349–358.

[22] M. B. van Gijzen, An analysis of element-by-element preconditioners for nonsymmetric problems, *Comput. Methods Appl. Mech. Engrg.* 105 (1993) 23–40.

[23] G.H. Golub and C.F. van Loan, *Matrix Computations (third ed.)* (The John Hopkins University Press, Baltimore, Maryland, 1996).

[24] I. Gustafsson, A class of first order factorization methods, *BIT* 18 (1978) 142–156.

[25] G. Haase, Parallel incomplete Cholesky preconditioners based on the nonoverlapping data distribution, *Parallel Comput.* 24 (1998) 1685–1703.

[26] M. Magolu monga Made, Ordering strategies for modified block incomplete factorizations, *SIAM J. Sci. Comput.* 16 (1995) 378–399.

[27] M. Magolu monga Made, Implementation of parallel block preconditionings on a transputer-based multiprocessor, *Future Generation Computer Systems* 11 (1995) 167–173.

[28] M. Magolu monga Made, Taking advantage of the potentialities of dynamically modified block incomplete factorizations, *SIAM J. Sci. Comput.* 19 (1998) 1083–1108.

[29] M. Magolu monga Made, R. Beauwens and G. Warzee. Preconditioning by moving the spectrum of discrete Helmholtz operators along the imaginary axis. Technical Report, Service des Milieux Continus, Université Libre de Bruxelles, June 1999.

[30] M. Magolu monga Made and B. Polman, Efficient planewise like preconditioners to cope with 3D problems, *Numer. Linear Algebra Appl.* 6 (1999) 379–406.

[31] 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) 148–162.

[32] J.A. Meijerink and H.A. van der Vorst, Guidelines for the usage of incomplete decompositions in solving sets of linear equations as they occur in practical problems, *J. Comp. Physics* 44 (1981) 134–155.

[33] N. Munksgaard, Solving sparse symmetric sets of linear equations by preconditioned conjugate gradient method, *ACM Trans. Math. Softw.* 6 (1980) 206-219.

[34] S. Nakamura, *Computational Methods in Engineering and Science* (J. Wiley & Sons, New York, 1977).

[35] Y. Notay, DRIC : A Dynamic Version of the RIC Method, *Numer. Linear Algebra Appl.* 1, 511–532, 1994.

[36] Y. Notay, An efficient Parallel Discrete PDE Solver, *Parallel Comput.* 21 (1995) 1725–1748.

[37] Y. Notay and A. Van de Velde, Coarse grid acceleration of parallel incomplete preconditioners, in : S. Margenov and P. Vassilevski, eds., *Iterative Methods in Linear Algebra II* (IMACS series in Computational and Applied Mathematics 3, 1996) 106–130.

[38] T. Oppe and W. Joubert, Improved SSOR and incomplete Cholesky solution of linear equations on shared and distributed memory parallel computers, *Numer. Linear Algebra Appl.* 1 (1994) 287–311.

[39] Y. Saad, *Iterative Methods for Sparse Linear Systems* (PWS Publishing, Co., Boston, MA, 1996).

[40] B.F. Smith, P.E. Björstad and D. Gropp, *Domain Decomposition : Parallel Multilevel Methods for Elliptic Partial Differential Equations* (Cambridge University Press, Cambridge, 1996).

[41] E. de Sturler, Incomplete block LU preconditioners on slightly overlapping subdomains for a massively parallel computer, Technical Report CSCS-TR-94-03, Swiss Scientific Computing Center, ETH, Zurich, 1994.

[42] K.H. Tan, J. Groeneweg and M.J.A. Borsboom, Locally optimized block preconditioners based on domain decomposition, Preprint 880, Department of Mathematics, Utrecht University, 1994.

[43] R.S. Varga, *Matrix Iterative Analysis* (Prentice Hall, Englewood Cliffs, 1962).

[44] H.A. van der Vorst, A vectorizable variant of some ICCG methods, *SIAM J. Sci. Statist. Comput.* 3 (1982) 350–356.

[45] H.A. van der Vorst, Large tridiagonal and block tridiagonal linear systems on vector and parallel computers, *Parallel Comput.* 5 (1987) 54–54.

[46] H.A. van der Vorst, High performance preconditioning, *SIAM J. Sci. Statist. Comput.* 10 (1989) 1174–1185.

[47] H.A. van der Vorst, ICCG and related methods for 3D problems on vector computers, *Computer Physics Communications* 53 (1989) 223–235.

[48] T. Washio and K. Hayami, Parallel block preconditioning based on SSOR and MILU, *Numer. Linear Algebra Appl.* 1 (1994) 533–553.

[49] Y.G. Zhong, X.Y. Kong, G.M. Xu, and G.H. Kuang, Multiple sequential staging of tasks : A new approach to parallel computations, *Comm. Numer. Meth. Engng.* 15 (1999) 367–373.