

Pseudo-overlap and implicit fill-in for improving parallel block preconditionings

M. Magolu monga Made

Service de Métrologie Nucléaire (CP 165/84)

Université Libre de Bruxelles

50, avenue F.D. Roosevelt

B-1050 Brussels, Belgium

Phone: +32 2 6504042, Fax: +32 2 6504534, E-mail: magolu@ulb.ac.be

Abstract - Parallel block preconditionings based on incomplete block matrix factorizations are explored. The recently introduced concept of pseudo-overlap subdomains [M. Magolu monga Made, and H.A. van der Vorst, *Parallel incomplete factorizations with pseudo-overlapped subdomains, Parallel Computing, Vol. 27, 2001, pp. 989–1008*] is combined with implicit block fill-in technique [M. Magolu monga Made and B. Polman, *Efficient planewise like preconditioners to cope with 3D problems, Numerical Linear Algebra with Applications, Vol. 6, 1999, pp. 379–406*]. The parallel block preconditioners so obtained significantly improve the performance of existing parallel block incomplete factorizations.

Keywords— block tridiagonal matrices, parallel incomplete block factorizations, pseudo-overlap, implicit block fill-in, preconditioned conjugate gradient.

I. INTRODUCTION

Without any contest, incomplete factorizations of matrices are the most popular preconditioning techniques for solving large and sparse linear systems. Block (linewise) variants are more efficient than pointwise schemes of the same computational complexity [4], [13], [15]. This superiority of blockwise preconditionings is more pronounced when either of the following conditions holds(see, [9], [15]):

- there are strong jumps in the variation of the PDE coefficients;
- there are less Dirichlet boundary conditions;
- the total number of unknowns is very large;
- the number of unknowns in one direction is much larger than in the other direction(s), and the *lines* are taken along the *longest* direction.

Both the construction and the application of the preconditioners based on incomplete factorization use recursions, which are not easy to implement on parallel architectures. A common technique to increase the degree of parallelism in the preconditioners involved consists in changing the numbering of the unknowns. This usually leads to a deterioration of the convergence rate. This trade-off between high level parallelism and rate of convergence has been illustrated in [5]. As far as pointwise schemes are concerned, sophisticated techniques, that combine parallel reorderings with suitable selective fill-in strategies around the interfaces between adjacent subdomains, have been intro-

duced recently in [16], [17], [18], and theoretically analyzed in [19]. Properly handled, the methods proposed lead to a better balance between high level parallelism and fast rate of convergence. The technique has been successfully applied in [12] to unstructured finite element matrices from real life acoustic wave propagation problems. Other parallelization strategies may be found in [6], [23] and in [24, pp. 374-376].

When it comes to classical block incomplete factorizations, as the ones investigated in [4], several attempts have been reported in the literature, see, e.g., [1], [2], [10], [14], [20], [21], [22]. So far, none of the methods proposed leads to a almost perfect speed-up as achieved with the pointwise parallel preconditioners proposed in [16]-[19]. Our goal in this study is to improve the parallel block preconditioners investigated in [10]. To this end, as in [17], we will start by combining ingredients from domain decomposition with overlap, with local re-ordering strategy. Beside this, we will also make use of the implicit block fill-in technique used in [14].

The paper is organized as follows. The definitions and notation we need are summarized in Section 2. In Section 3, we describe our parallelizable incomplete block-matrix factorizations through a two-dimensional model problem. Numerical experiments performed on a 16-processor SGI Origin 2000, are discussed in Section 4. Concluding remarks are given in Section 5.

II. BACKGROUND

A. General terminology and notation

Below are some symbols that are used in our work. A denotes a given square matrix of order n .

A^t	:	transpose of A
$\text{diag}(A)$:	pointwise diagonal matrix whose diagonal entries coincide with those of A
$\text{offdiag}(A)$:	$A - \text{diag}(A)$
$\text{tridiag}(A)$:	pointwise tridiagonal matrix whose main three diagonals coincide with those of A
$\text{diag}_{\text{block}}(A)$:	block diagonal matrix whose block diagonal entries coincide with those of A
$\text{offdiag}_{\text{block}}(A)$:	$A - \text{diag}_{\text{block}}(A)$
m	:	number of main diagonal blocks in A
p	:	number of available processors

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.

By the LPL^t factorization of a Stieltjes matrix S , we understand the (exact) factorization $S = L_s P_s L_s^t$, where P_s is a diagonal matrix and L_s is a lower triangular matrix with $\text{diag}(L_s) = I$.

B. Model problem

We consider the following self-adjoint second order two-dimensional elliptic boundary value problem:

$$\begin{aligned} -(\phi u_x)_x - (\psi u_y)_y &= f(x, y) & \text{in } \Omega = (0, 1) \times (0, 1) \\ u &= 0 & \text{on } \Gamma \\ \frac{\partial u}{\partial n} &= 0 & \text{on } \partial\Omega \setminus \Gamma, \end{aligned} \quad (1)$$

where n denotes the outer normal to the boundary $\partial\Omega$ of Ω , Γ stands for a nonempty portion of $\partial\Omega$. The coefficients ϕ and ψ are positive, bounded and piecewise constant. Eq. (1) is discretized over a uniform rectangular grid of mesh size h , with the five-point finite volume method. The lexicographical ordering in the (x, y) -plane is used to number the unknowns. This yields a linear system of the form

$$A\mathbf{u} = \mathbf{b} \quad (2)$$

where

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & 0 & \cdots & 0 \\ A_{2,1} & A_{2,2} & A_{2,3} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & A_{m-1,m} & A_{m-1,m-1} & A_{m-1,m} \\ 0 & \cdots & 0 & A_{m,m-1} & A_{m,m} \end{bmatrix} \quad (3)$$

is a nonsingular block-tridiagonal, irreducibly diagonally dominant, Stieltjes matrix. Each $A_{i,i}$ is a (small) pointwise tridiagonal matrix, while each $A_{i,j}$, $i \neq j$, is a (small) pointwise diagonal matrix. We will solve (2) by means of the preconditioned conjugate gradient (PCG) method, whose algorithm is given in Fig. 1.

C. Incomplete block-matrix factorizations

A may be split as

$$A = D - L - L^t \quad (4)$$

with

$$\begin{aligned} D &= \text{diag}_{\text{block}}(A) \\ -(L + L^t) &= \text{offdiag}_{\text{block}}(A) \end{aligned} \quad (5)$$

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

Fig. 1. PCG method. B stands for the preconditioning matrix.

where L is strictly block lower triangular. The matrix

$$B = (P - L)P^{-1}(P - L^t) \quad (6)$$

where P denotes the block diagonal matrix computed according to Algorithm 2.1 is a block incomplete LU-factorization (BILU) of A . $P_{i,i} = L_{p_{i,i}} P_{p_{i,i}} L_{p_{i,i}}^t$ stands

for the LPL^t (exact) factorization of $P_{i,i}$.

In our context, each $P_{i,i}$ is a tridiagonal Stieltjes matrix,

see, e.g., [4]. $\widetilde{P}_{i,i}^{-1} = \text{tridiag}(P_{i,i}^{-1})$ may be cheaply com-

puted from the LPL^t factorization of $P_{i,i}$. This stems from the fact, that for any tridiagonal matrix T such that $T = L_T P_T L_T^t$, T^{-1} may be rewritten as

$$T^{-1} = T^{-1}(I - L_T) + L_T^{-t} P_T^{-1}. \quad (7)$$

The lower part of $\widetilde{T} = \text{tridiag}(T^{-1})$ may be computed then by simple identification of the relevant entries in (7). This is performed in Algorithm 2.2 where, for simplicity, we set $P_T = Q = (q_{i,i} \delta_{i,j})$ and $L_T = G = (g_{i,j})$.

III. PARALLEL BLOCK INCOMPLETE FACTORIZATION PRECONDITIONERS

The domain (square) is partitioned into p subdomains. Data involved in the i -th subdomain ($i = 1, 2, \dots, p$) are allocated to processor \mathcal{P}_{i-1} . The unknowns are renumbered

Algorithm 2.1 (Block incomplete LU-factorization)

Compute

- $P_{1,1} = A_{1,1}$
- $P_{1,1} = L_{p_{1,1}} P_{p_{1,1}} L_{p_{1,1}}^t$
- $\widetilde{P}_{1,1}^{-1} = \text{tridiag}(P_{1,1}^{-1})$
- for $i = 2, 3, \dots, m$
 - $P_{i,i} = A_{i,i} - A_{i,i-1} \widetilde{P}_{i-1,i-1}^{-1} A_{i-1,i}$
 - $P_{i,i} = L_{p_{i,i}} P_{p_{i,i}} L_{p_{i,i}}^t$
 - if $i = m$ then stop
 - $\widetilde{P}_{i,i}^{-1} = \text{tridiag}(P_{i,i}^{-1})$

Algorithm 2.2 ($\widetilde{T} = \text{tridiag}(T^{-1})$)

- $\tilde{t}_{n,n} = \frac{1}{q_{n,n}}$
- for $i = n, n-1, \dots, 2$
 - $\tilde{t}_{i,i-1} = -\tilde{t}_{i,i} g_{i,i-1}$
 - $\tilde{t}_{i-1,i-1} = \frac{1}{q_{i-1,i-1}} - \tilde{t}_{i,i-1} g_{i,i-1}$

T : tridiagonal Stieltjes matrix of order n

$T = GQG^t$: LPL^t factorization of T

line by line as illustrated in Fig. 2 in the case of horizontal stripe partitioning and eight subdomains. As in [10], [22], we have used twisting: lines are numbered from bottom to top in the bottom half portion of the physical domain, and from top to bottom in the top half portion. For $p = 2$, without twisting, any incomplete factorization based preconditioning does not anymore give rise to approximately the same remainder matrix as for the lexicographical ordering [5], [10], [19], [22].

In the case of Fig. 2, one possible allocation of *lines* to processor is as follows:

$$\begin{aligned} \mathcal{P}_0 &= \{1, 2, 3, 27\}, & \mathcal{P}_1 &= \{4, 5, 6, 28\}, \\ \mathcal{P}_2 &= \{7, 8, 9, 29\}, & \mathcal{P}_3 &= \{10, 11, 12, 13\}, \\ \mathcal{P}_4 &= \{23, 24, 25, 26, 33\}, & \mathcal{P}_5 &= \{20, 21, 22, 32\}, \\ \mathcal{P}_6 &= \{17, 18, 19, 31\}, & \mathcal{P}_7 &= \{14, 15, 16, 30\}. \end{aligned}$$

Obviously, there exists a permutation matrix \mathcal{P} such that

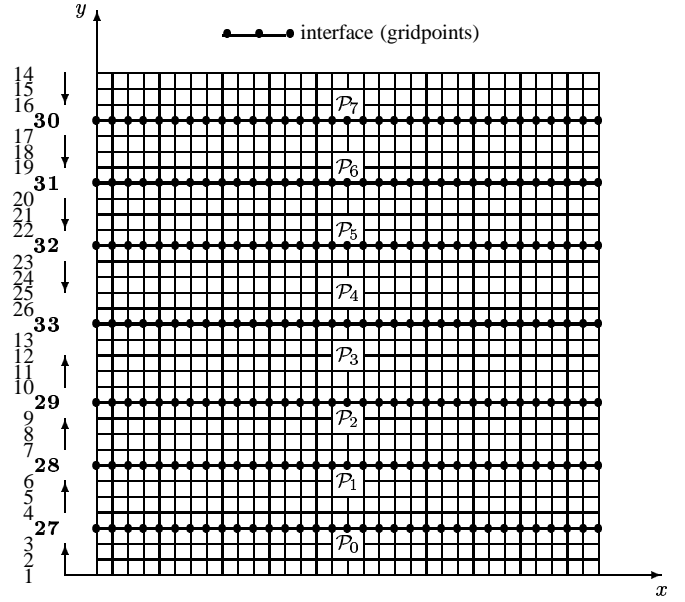


Fig. 2. Decomposition of the grid into 8 subdomains (stripes). Arrows indicate the progressing direction of the line numbering per subdomain. Numbers along the y -axis give the global line numbering of unknowns. Within each horizontal line, lexicographical ordering is used.

(2) changes to

$$\mathcal{A}\tilde{u} = \tilde{b} \quad \text{with} \quad \tilde{b} = \mathcal{P}^t b, \quad u = \mathcal{P}\tilde{u} \quad (8)$$

and

$$\mathcal{A} = \mathcal{P}^t \mathcal{A} \mathcal{P} = \begin{bmatrix} \mathcal{A}_{1,1} & 0 & \cdots & 0 & \mathcal{A}_{1,I} \\ 0 & \mathcal{A}_{2,2} & \ddots & \vdots & \mathcal{A}_{2,I} \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \ddots & 0 & \mathcal{A}_{p,p} & \vdots \\ \mathcal{A}_{I,1} & \mathcal{A}_{I,2} & \cdots & \cdots & \mathcal{A}_{I,I} \end{bmatrix} \quad (9)$$

where $\mathcal{A}_{i,i}$, $i = 1, 2, \dots, p$, are uncoupled block tridiagonal matrices associated to interior grid-points (lines 1 to 26 in the case of Fig. 2), while $\mathcal{A}_{I,I}$ is made up of uncoupled pointwise tridiagonal matrices associated to interface grid-points (lines 27 to 33 in Fig. 2). More detailed nonzero structure of the global matrix is depicted in Fig. 3, where level 1 fill-ins are additional block fill-in entries that would not appear if we would have numbered all the grid-points lexicographically. Level 2 fill-ins are block fill-ins generated by Level 1 fill-ins. How these additional fill-ins occur is illustrated in Fig. 4 by means of arcs that represent connections between blocks (or lines). It is obvious that, if we want the parallel block ordering to be as effective as the standard (lexicographic) block ordering, we have to accept

some levels of block fill-in such that the rejected fill-in entries are small enough. As illustrated in Fig. 4, this can be interpreted as increasing a kind of (small) ‘overlap’ between neighboring subdomains. This ‘overlap’ has been called pseudo-overlap in [16]-[19].

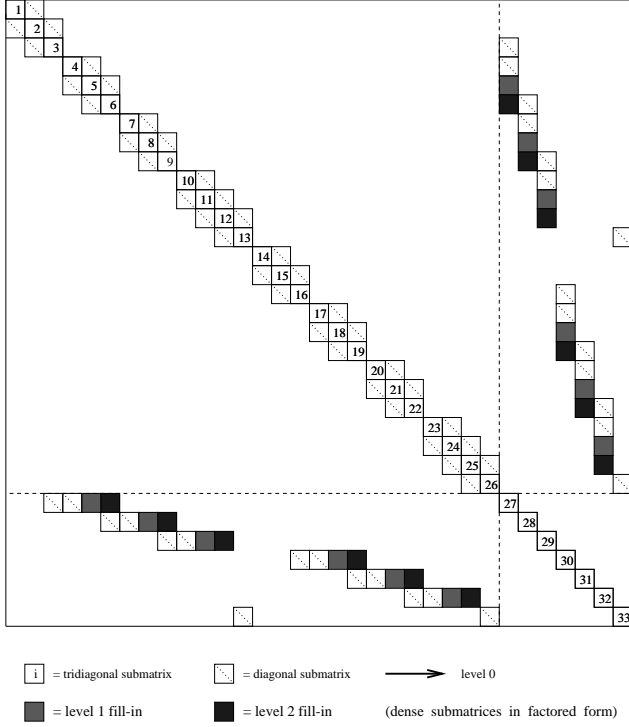


Fig. 3. Global matrix A associated to 8 subdomains (level 0). Additional level 1 and level 2 block fill-in entries. Here $m = 33$ and $p = 8$.

From an implementation viewpoint, we assume that \mathcal{A} is partitioned into (small) blocks as illustrated in Fig. 3 in the case of $p = 8$. The size of each (small) block is the same as in (3). For simplicity, we denote the corresponding block entries of \mathcal{A} by $A_{i,j}$. We will use $\tilde{A}_{i,j}$ to denote the block fill-in entries. Level 1 and level 2 block fill-ins have the form (see Fig. 3)

$$\tilde{A}_{i,j} = -A_{i,j-1}P_{j-1,j-1}^{-1}A_{j-1,j}$$

$$\tilde{A}_{i,j+1} = -\tilde{A}_{i,j}P_{j,j}^{-1}A_{j,j+1}$$

respectively. They are not explicitly computed but implicitly stored in factored form, as in [14]. Multiplication by $P_{k,k}^{-1}$ should be implemented as the solution of a (small) linear system. There is no additional storage with respect to Algorithm 2.1. Set

$$\mathcal{F} = \{(i, j) ; \text{fill-in is accepted in (block) position } (i, j)\} . \quad (10)$$

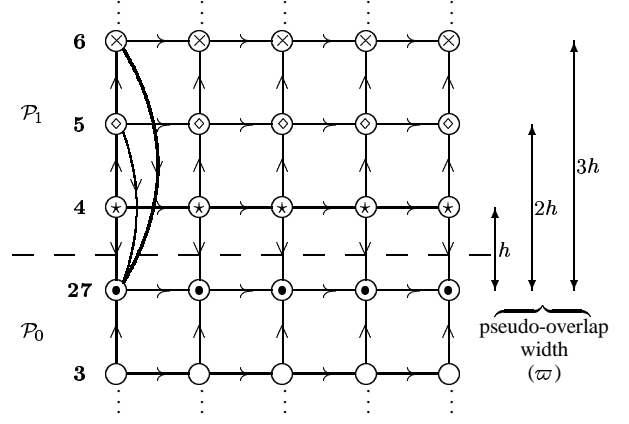


Fig. 4. Part of graph of matrix A assigned to two adjacent subdomains: \mathcal{P}_0 and \mathcal{P}_1 . Thin (short) arc and thick (long) arc correspond to level 1 and level 2 block fill-in entries, respectively.

Let \tilde{L} denote the block strictly lower triangular matrix whose block entries are defined by

$$\tilde{L}_{i,j} = \begin{cases} -A_{i,j} & \text{if } (i, j) \notin \mathcal{F} \\ -\tilde{A}_{i,j} & \text{otherwise} , \end{cases} \quad (11)$$

and let P stand for the block diagonal matrix whose block entries are computed by means of Algorithm 3.1

Definition 1: Similarly to [16], [17], the matrix

$$B = (P - \tilde{L})P^{-1}(P - \tilde{L}^t) , \quad (12)$$

will be referred to as $\text{ParBILU}(\ell; \varpi, \ell_\varpi)$, say the parallel block incomplete LU factorization with pseudo-overlap width ϖ , interior block fill-in level ℓ , and ℓ_ϖ as the block fill-in level in the pseudo-overlapping regions.

Following [16], [17], we will denote the pseudo-overlap width by $\varpi = 1, 2, 3, \dots$ rather than $\varpi = h, 2h, 3h, \dots$ in which case, the corresponding pseudo-overlapping regions between, for instance, subdomains \mathcal{P}_0 and \mathcal{P}_1 are made of lines $\{4, 27\}$, $\{4, 5, 27\}$, $\{4, 5, 6, 27\}$, \dots , respectively (see Fig. 4).

The existence of $\text{ParBILU}(\ell; \varpi, \ell_\varpi)$ for symmetric M-matrices may be established by means of the same arguments as in [4], [7]. $\text{ParBILU}(\ell; \varpi, \ell_\varpi)$ improves and generalizes the block methods developed in [22], [20], [10]. $\text{ParBILU}(0; 1, 0)$ is none else than a particular case (no additive perturbations) of the parallel block preconditioner introduced in [10]. The method investigated in [20] may be seen as an incomplete block-Jacobi preconditioner. The domain is partitioned into nonoverlapped subdomains: within each subdomain, local direct solver is replaced by one application of a standard block incomplete factorization. In the case of Fig. 2, the processor which handles, say, subdomain \mathcal{P}_1 applies Algorithm 2.1 to blocks (or lines) 4, 5, 6 and 28, without sending any information to \mathcal{P}_0 (line

27) and without receiving any contribution from \mathcal{P}_2 (line 7). In [22], the same procedure was used for constructing a parallel incomplete block preconditioner called INVpP, where p stands for the number of subdomains. But there, contrary to [20], adjacent subdomains exchange information during the solving of $Bw = r$ at each PCG iteration. In contrast to ParBILU($\ell; \varpi, \ell_\varpi$) where tridiagonal linear systems $P_{i,i}v = s$ are solved exactly, in INVpP, $P_{i,i}^{-1}$ is replaced by its seven main diagonals. This could be very fast on computers with vector processors. The other side of the coin is that the number of PCG iterations could significantly increase, especially for difficult PDEs. More severe is the fact that this may give rise to indefinite preconditioners, [2]. To get rid of such an inconvenience, some *modifications* have been proposed in [1], [2], whose rate of convergence is unpredictable.

IV. NUMERICAL RESULTS

We use the zero vector as initial guess, and the residual error reduction $\|r^{(i)}\|_2/\|b\|_2 \leq 10^{-6}$ as convergence criterion. We first check whether $\gamma_i/\gamma_0 \leq 10^{-6}$ is satisfied (See the PCG algorithm, Fig. 1, for the notation). The experiments are conducted on a 16-processor SGI Origin 2000 (195 MHz). The MPI library is used for communication between processors. Optimized BLAS routines are invoked. The preconditionings include:

- (1) ParBILU($\ell; \varpi, \ell_\varpi$);
- (2) ParIC($\ell; \varpi, \ell_\varpi$): the parallel pointwise incomplete preconditioner introduced in [17]. ϖ denotes the pseudo-overlap width, which is defined as in Fig. 4, ℓ_ϖ stands for the fill-in level in the pseudo-overlapping regions ($\ell_\varpi \geq \varpi - 1$), while ℓ denotes the fill-in level in the remaining part of the subdomains;
- (3) INVpP: the parallel block preconditioner introduced in [22].

We consider two particular cases of PDE (1) :

Problem 1:

- $\Gamma = \partial\Omega$, $\phi = \psi = 1$ (see (1));
- The rhs of the linear system to solve is chosen such that the function $u_0(x, y) = x(1-x)y(1-y)e^{xy}$ generates the solution on the grid.

Problem 2:

- $\Gamma = \{(x, y); 0 \leq x \leq 1, y = 0\}$;
- $\phi = \psi = \begin{cases} 100 & \text{in } (1/4, 3/4) \times (1/4, 3/4) \\ 1 & \text{elsewhere} \end{cases}$
- $f(x, y) = \begin{cases} 100 & \text{in } (1/4, 3/4) \times (1/4, 3/4) \\ 0 & \text{elsewhere} \end{cases}$

The results of our numerical experiments can be seen in Tables I and II. We give the elapsed time in seconds for constructing the preconditioning matrix, and for computing

Algorithm 3.1 : ParBILU($\ell; \varpi, \ell_\varpi$)

(1) For $k = 1, 2, \dots, p$, apply Algorithm 2.1 to $\mathcal{A}_{k,k}$ (in parallel)

(2) For $i = m - p + 2, \dots, m - 1$ (in parallel, with communication)

- let $j, j < i$ be such that $A_{i,j} \neq 0$ and $A_{i,j+1} \neq 0$ (see Fig. 3)

- $P_{i,i} = A_{i,i} - A_{i,j} \widetilde{P}_{j,j}^{-1} A_{j,i} - A_{i,j+1} \widetilde{P}_{j+1,j+1}^{-1} A_{j+1,i}$
- if $\ell_\varpi \geq 1$

- store $\widetilde{A}_{i,j+2} = -A_{i,j+1} P_{j+1,j+1}^{-1} A_{j+1,j+2}$

- $P_{i,i} := P_{i,i} - \text{tridiag} \left(\widehat{A}_{i,j+2} \widetilde{P}_{j+2,j+2}^{-1} \widehat{A}_{j+2,i} \right)$

where

$$\widehat{A}_{i,j+2} = -A_{i,j+1} \widetilde{P}_{j+1,j+1}^{-1} A_{j+1,j+2}$$

$$\widehat{A}_{j+2,i} = -A_{j+2,j+1} \widetilde{P}_{j+1,j+1}^{-1} A_{j+1,i}$$

- if $\ell_\varpi \geq 2$

- store $\widetilde{A}_{i,j+3} = -\widetilde{A}_{i,j+2} P_{j+2,j+2}^{-1} A_{j+2,j+3}$

- $P_{i,i} := P_{i,i} - \text{tridiag} \left(\widehat{A}_{i,j+3} \widetilde{P}_{j+3,j+3}^{-1} \widehat{A}_{j+3,i} \right)$

where

$$\widehat{A}_{i,j+3} = -A_{i,j+2} \widetilde{P}_{j+2,j+2}^{-1} A_{j+2,j+3}$$

$$\widehat{A}_{j+3,i} = -A_{j+3,j+2} \widetilde{P}_{j+2,j+2}^{-1} A_{j+2,i}$$

(3) For $i = m$ (with communication)

- let $r, s, r < s < m$ be such that $A_{m,r} \neq 0$ and $A_{m,s} \neq 0$ (see Fig. 3)

- $P_{m,m} = A_{m,m} - A_{m,r} \widetilde{P}_{r,r}^{-1} A_{r,m} - A_{m,s} \widetilde{P}_{s,s}^{-1} A_{s,m}$

the approximate solution by the PCG algorithm. We also report the parallel speed-up, say the ratio between the time taken by the sequential code (one processor) and the computational time for p processors. Observe that, for $p = 1$ and for fixed interior fill-in level $\ell = \hat{\ell}$, ParIC($\hat{\ell}; \varpi, \ell_\varpi$)

and ParIC($\hat{\ell}; \varpi, \ell_\varpi$) do not depend on ϖ and ℓ_ϖ . In our implementation, this also holds for $p = 2$ because of twisting. That is the reason why the performance of ParBILU(0;2;1), ParBILU(0;3,2), and ParIC(1;3,2) are not reported for $p \leq 2$. For all the preconditioners involved, the results show, in general, a linear speed-up. In some cases, the speed-up observed is larger than the number of processors.

TABLE I

Problem 1. $h^{-1} = 513$; $n = 262144$. Number of PCG iterations (iter.); elapsed time in seconds for constructing the preconditioner (fact.), executing the PCG algorithm, and overall time; speed-up, for p processors.

Precond.	p	iter.	Time			overall speed-up
			fact.	pcg	overall	
ParBILU(0;1,0)	1	189	0.11	59.91	60.57	1.00
	2	192	0.07	28.41	28.87	2.10
	4	224	0.04	14.51	14.76	4.10
	8	229	0.02	6.91	7.09	8.54
	16	238	0.02	4.21	4.42	13.70
ParBILU(0;2;1)	4	203	0.04	13.05	13.31	4.55
	8	203	0.04	5.91	6.10	9.93
	16	210	0.05	3.57	3.80	15.94
ParBILU(0;3;2)	4	194	0.13	12.50	12.87	4.71
	8	195	0.13	5.78	6.12	9.90
	16	200	0.20	3.33	3.80	15.94
INVpP	1	206	1.18	80.04	81.77	1.00
	2	213	0.17	39.01	39.56	2.07
	4	226	0.08	17.46	17.82	4.59
	8	230	0.04	8.27	8.60	9.51
	16	243	0.02	5.41	5.66	14.45
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(1;2,1)	1	266	0.99	137.47	139.38	1.00
	2	266	0.49	67.60	68.58	2.03
	4	270	0.23	28.43	28.91	4.82
	8	272	0.13	12.09	12.37	11.27
	16	276	0.07	6.33	6.48	21.51
ParIC(1;3,2)	4	268	0.23	28.19	28.64	4.87
	8	269	0.12	11.93	12.20	11.42
	16	271	0.08	6.14	6.30	22.12

TABLE II

Problem 2. $h^{-1} = 512$; $n = 262656$. Number of PCG iterations (iter.); elapsed time in seconds for constructing the preconditioner (fact.), executing the PCG algorithm, and overall time; speed-up, for p processors.

Precond.	p	iter.	Time			overall speed-up
			fact.	pcg	overall	
ParBILU(0;1,0)	1	238	0.11	69.48	69.91	1.00
	2	238	0.06	31.34	31.65	2.21
	4	291	0.03	16.33	16.55	4.22
	8	301	0.02	7.76	7.95	8.79
	16	314	0.02	4.60	4.80	14.56
ParBILU(0;2;1)	4	258	0.04	14.50	14.70	4.76
	8	266	0.03	6.73	6.93	10.09
	16	273	0.04	4.00	4.22	16.57
ParBILU(0;3;2)	4	242	0.11	13.63	13.95	5.01
	8	245	0.12	6.34	6.63	10.54
	16	250	0.18	3.74	4.15	16.85
INVpP	1	258	0.22	86.38	87.04	1.00
	2	261	0.12	41.71	42.12	2.07
	4	278	0.06	18.15	18.42	4.73
	8	291	0.03	9.44	9.65	9.02
	16	306	0.02	5.45	5.70	15.27
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(1;2,1)	1	405	1.01	176.30	177.93	1.00
	2	405	0.49	89.00	89.89	1.98
	4	418	0.26	39.10	39.53	4.50
	8	421	0.13	17.08	17.33	10.27
	16	428	0.07	8.95	9.09	19.57
ParIC(1;3,2)	4	407	0.25	38.44	38.87	4.58
	8	408	0.13	16.29	16.53	10.76
	16	412	0.08	8.77	8.97	19.84

It should be emphasized that the global (parallel) ordering changes with the number of subdomains. Therefore, in reality we are not comparing the same preconditioning matrix, see, e.g., [5].

As regards the pseudo-overlap, the results are consistent with what has been observed in [16]-[19] for parallel pointwise incomplete factorization preconditioners. It is mandatory to increase the pseudo-overlap width, accepting thereby some fill-in induced by the parallel (re)numbering, in order to avoid a degradation of the convergence rate. This holds in particular for (the difficult) Problem 2. We know from [4] that the computational complexity of level-0 block methods is essentially equal to that of level-1 pointwise preconditioners. Therefore, it is fair to compare ParBILU(0;k,k-1) with ParIC(1;k,k-1), $k = 2, 3$. From the tables it can be concluded that parallel block methods

are at least twice as fast as the corresponding pointwise methods. The results observed compare ParBILU(0;k,k-1) favorably with INVpP, both in terms of number of PCG iterations as well as the execution time. This is better illustrated in Fig. 5 and in Table III, where the performances of ParBILU(0;k,k-1) and INVpP are given for larger linear systems. Problem 1 is discretized with the mesh size $h^{-1} = 1025$, which yields a linear system of 1 048 576 unknowns, and 3 143 680 nonzero matrix entries to store. Problem 2 is solved for $h^{-1} = 1024$, which gives rise to a matrix of order $n = 1 049 600$ and 3 146 751 nonzero entries to store. Obviously, INVpP is more expensive than ParBILU(0;k,k-1), $k = 1, 2, 3$. INVpP keeps seven diagonals for the computation of the approximate inverse of pivot block entries. Though only the tridiagonal part is used for the construction of the preconditioner, all the seven diagonals are taken into account during each solving

of the preconditioning system $Bw = r$. This requires seven flops per unknown for handling each pivot block linear system, as opposed to three flops per unknown in the case of ParBILU(0;k,k-1), where pivot blocks are pointwise tridiagonal matrices.

TABLE III

Problem 1: $h^{-1} = 1025$; $n = 1\,048\,576$. Problem 2: $h^{-1} = 1024$; $n = 1\,049\,600$. Number of PCG iterations (iter.); elapsed time in seconds for constructing the preconditioner (fact.), executing the PCG algorithm, and overall time; speed-up, for p processors, ParBILU(0; k, k - 1), $k = 1, 2, 3$, and INVpP.

Precond.	p	iter.	Time			overall speed-up
			fact.	pcg	overall	
Problem 1						
ParBILU(0;1,0)	1	362	0.47	632.22	635.07	1.00
	2	381	0.28	375.74	377.72	1.68
	4	441	0.13	187.62	188.87	3.36
	8	446	0.06	79.84	80.63	7.88
	16	459	0.11	39.94	41.13	15.44
ParBILU(0;2;1)	4	400	0.16	171.95	173.18	3.67
	8	403	0.12	73.27	74.15	8.56
	16	408	0.16	35.89	37.08	17.13
ParBILU(0;3;2)	4	384	0.48	163.36	164.87	3.85
	8	386	0.46	69.46	70.63	8.99
	16	390	0.64	34.64	36.38	17.46
INVpP	1	409	4.41	843.51	850.15	1.00
	2	420	2.75	476.00	480.42	1.77
	4	442	1.36	219.80	222.18	3.83
	8	447	0.18	97.60	98.49	8.63
	16	459	0.09	50.35	51.51	16.50
Problem 2						
ParBILU(0;1,0)	1	478	0.47	683.41	685.45	1.00
	2	479	0.28	396.44	397.86	1.72
	4	577	0.13	197.85	198.75	3.45
	8	587	0.07	80.91	81.65	8.39
	16	603	0.12	39.71	40.80	16.80
ParBILU(0;2;1)	4	523	0.16	179.42	180.03	3.81
	8	527	0.11	72.59	73.34	9.35
	16	537	0.20	35.21	36.39	18.84
ParBILU(0;3;2)	4	485	0.44	166.88	168.13	4.08
	8	488	0.41	67.72	68.76	9.97
	16	493	0.62	32.94	34.54	19.85
INVpP	1	520	0.93	860.57	862.95	1.00
	2	521	0.50	487.47	489.11	1.76
	4	545	0.25	216.76	217.78	3.96
	8	556	0.13	92.21	92.98	9.28
	16	573	0.07	47.40	48.44	17.81

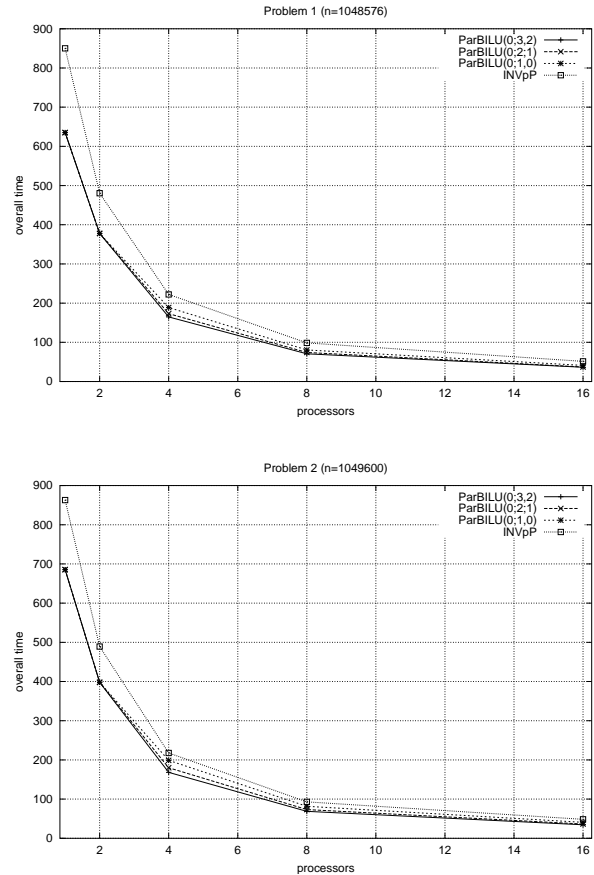


Fig. 5. Problem 1: $h^{-1} = 1025$, $n = 1\,048\,576$. Problem 2: $h^{-1} = 1024$, $n = 1\,049\,600$. Overall computational time for ParBILU(0;1,0), ParBILU(0;2;1), ParBILU(0;3;2), and INVpP.

V. CONCLUDING REMARKS

In this paper, we have introduced a new parallel block incomplete matrix based preconditioner. Ingredients from parallel scalar preconditioners investigated previously, [17], are combined with an appropriate implicit block fill-in strategy, [14]. Numerical experiments indicate that a nice trade-off between parallelism and rate of convergence is achieved. Our results also display how the new scheme improve existing parallel block preconditioners, [9], [22]. The convergence rate is remarkably weakly sensitive to the number of subdomains. A formal proof of this, as obtained in [19] for pointwise preconditioners, awaits further investigation. Future works may also be conducted to explore three-dimensional problems, and to incorporate coarse grid acceleration.

REFERENCES

- [1] O. Axelsson and V. Eijkhout, Vectorizable preconditioners for elliptic difference equations in three space dimensions, *J. Comput. Appl. Math.* 27 (1989) 299–321.
- [2] O. Axelsson and B. Polman, An approximate factorization methods for block matrices suitable for vector and parallel processors, *Linear Algebra Appl.* 77 (1986) 3–26.
- [3] R. Beauwens and M. Ben Bouzid, Existence and conditioning properties of sparse approximate block factorizations, *SIAM J. Numer. Anal.* 25 (1988) 941–956.
- [4] P. Concus, G.H. Golub, and G.A. Meurant, Block preconditioning for the conjugate gradient method, *SIAM J. Sci. Statist. Comput.* 6 (1985) 220–252.
- [5] I.S. Duff and G.A. Meurant, The effect of ordering on preconditioned conjugate gradients, *BIT* 29 (1989) 635–657.
- [6] D. Hysom and A. Pothen, A scalable parallel algorithm for incomplete factor preconditioning, *SIAM J. Sci. Comput.* 22 (2001) 2194–2215.
- [7] M. Magolu monga Made, Modified block-approximate factorization strategies, *Numer. Math.* 61 (1992) 91–110.
- [8] M. Magolu monga Made, Analytical bounds for block approximate factorization methods, *Linear Algebra Appl.* 179 (1993) 33–57.
- [9] M. Magolu monga Made, Ordering strategies for modified block incomplete factorizations, *SIAM J. Sci. Comput.* 16 (1995) 378–399.
- [10] M. Magolu monga Made, Implementation of parallel block preconditionings on a transputer-based multiprocessor, *Future Generation Computer Systems* 11 (1995) 167–173.
- [11] M. Magolu monga Made, Taking advantage of the potentialities of dynamically modified block incomplete factorizations, *SIAM J. Sci. Comput.* 19 (1998) 1083–1108.
- [12] M. Magolu monga Made, Performance of parallel incomplete LDLt factorizations for solving acoustic wave propagation problems from industry, *Numer. Linear Algebra Appl.* 11(8–9) (2004) 813–830.
- [13] M. Magolu monga Made and Y. Notay, Dynamically relaxed block incomplete factorization methods for solving two- and three-dimensional problems, *SIAM J. Sci. Comput.* 21 (2000) 2008–2028.
- [14] M. Magolu monga Made and B. Polman, Efficient planewise like preconditioners to cope with 3D problems, *Numer. Linear Algebra Appl.* 6 (1999) 379–406.
- [15] M. Magolu monga Made and B. Polman, Experimental comparison of three-dimensional point and line modified incomplete factorizations, *Numer. Algorithms* 23 (2000) 51–70.
- [16] M. Magolu monga Made and H.A. van der Vorst, ParIC : A Family of Parallel Incomplete Cholesky Preconditioners, in: M. Bubak, H. Afsarmanesh, R. Williams, and B. Hertzberger, eds., *High Performance Computing and Networking*, Proc. HPCN Europe 2000, Amsterdam, Lecture Notes in Computer Science, 1823 (Springer-Verlag, Berlin, 2000) 89–98.
- [17] M. Magolu monga Made, and H.A. van der Vorst, Parallel incomplete factorizations with pseudo-overlapped subdomains, *Parallel Comput* 27 (2001) 989–1008.
- [18] M. Magolu monga Made, and H.A. van der Vorst, A generalized domain decomposition paradigm for parallel incomplete LU factorization preconditionings, *Future Generation Computer Systems* 17 (2001) 925–932.
- [19] M. Magolu monga Made, and H.A. van der Vorst, Spectral analysis of parallel incomplete factorizations with implicit pseudo-overlap, *Numer. Linear Algebra Appl.* 9 (2002) 45–64 .
- [20] V. Mehrmann, Preconditioning of block structured linear systems with block ILU on parallel computers, in: W. Hackbusch and G. Wittum, eds., *Incomplete Decomposition (ILU) - Algorithms, Theory and Applications*, Notes on Numerical Fluid Mechanics 41 (Vieweg, Braunschweig, 1993) 88–95.
- [21] G.A. Meurant, The block preconditioned conjugate gradient method on vector computers, *BIT* 24 (1984) 623–633.
- [22] G.A. Meurant, Multitasking the conjugate gradient method on the CRAY X-MP/48, *Parallel Comput.* 5 (1987) 267–280.
- [23] Y. Notay, An efficient Parallel Discrete PDE Solver, *Parallel Comput.* 21 (1995) 1725–1748.
- [24] Y. Saad, *Iterative methods for sparse linear systems* (PWS Publishing Company, Boston, 1996).