Tema
(cc) BY
Tendências em Matemática Aplicada e Computacional, **19**, N. 1 (2018), 59-77

www.scielo.br/tema
doi: 10.5540/tema.2018.019.01.0059

Parallel Implementation of a Two-level Algebraic ILU(k)-based Domain Decomposition Preconditioner†

I.C.L. NIEVINSKI[1*], M. SOUZA[2], P. GOLDFELD[3], D.A. AUGUSTO[4], J.R.P. RODRIGUES[5] and L.M. CARVALHO[6]

Received on November 28, 2016 / Accepted on September 26, 2017

ABSTRACT. We discuss the parallel implementation of a two-level algebraic ILU(k)-based domain decomposition preconditioner using the PETSc library. We present strategies to improve performance and minimize communication among processes during setup and application phases. We compare our implementation with an off-the-shelf preconditioner in PETSc for solving linear systems arising in reservoir simulation problems, and show that for some cases our implementation performs better.

Keywords: Two-level preconditioner, domain decomposition, Krylov methods, linear systems, parallelism, PETSc.

1 INTRODUCTION

This paper discusses the formulation and the parallel implementation of an algebraic ILU(k)-based two-level domain decomposition preconditioner first introduced in [2].

In this work we present and discuss details of the implementation using the MPI-based PETSc suite [3], a set of data structures and routines for the parallel solution of scientific applications modeled by partial differential equations. We also present results of computational experiments involving matrices from oil reservoir simulation. We have tested with different number of processes and compared the results with the default PETSc preconditioner, block Jacobi, which is a usual option in the oil industry.

---

†This article is based on work presented at CNMAC2016.

*Corresponding author: Ítalo Nievinski – E-mail: italonievinski@gmail.com.

[1]Faculdade de Engenharia Mecânica, PPGEM, UERJ - Universidade do Estado do Rio de Janeiro, 20550-900 Rio de Janeiro, RJ, Brasil.

[2]Departamento de Estatística e Matemática Aplicada DEMA UFC - Universidade Federal do Ceará, Campus do PICI, 60455-760, Fortaleza, CE, Brasil. E-mail: michael@ufc.br

[3]Departamento de Matemática Aplicada, IM-UFRJ, Caixa Postal 68530, CEP 21941-909, Rio de Janeiro, RJ, Brasil. E-mail: goldfeld@ufrj.br

[4]Fundação Oswaldo Cruz, Fiocruz, Av. Brasil, 4365, 21040-360 Rio de Janeiro, RJ, Brasil. E-mail: daa@fiocruz.br

[5]PETROBRAS/CENPES Av. Horácio Macedo 950, Cidade Universitária, 21941-915 Rio de Janeiro, RJ, Brasil. E-mail: jrprodrigues@petrobras.com.br

[6]Instituto de Matemática e Estatística, IME, UERJ - Universidade do Estado do Rio de Janeiro, 20550-900 Rio de Janeiro, RJ, Brasil. E-mail: luizmc@ime.uerj.br

The multilevel preconditioner has been an active research area for the last 30 years. One of the main representatives of this class is the algebraic multigrid (AMG) method [19, 22, 30, 31], when used as a preconditioner rather than as a solver. It is widely used in oil reservoir simulation as part of the CPR (constrained pressure residual) preconditioner [6, 33]. Despite its general acceptance there is room for new alternatives, as there are problems where AMG can perform poorly, see, for instance [13]. Among these alternatives we find the two-level preconditioners. There are many variations within this family of preconditioners, but basically we can discern at least two subfamilies: *algebraic* [1, 4, 12, 14, 18, 20, 23, 28, 32] and *operator dependent* [15, 16, 17]. Within the operator dependent preconditioners we should highlight the spectral methods [21, 25].

Incomplete LU factorization ILU(k) [26] has long been used as a preconditioner in reservoir simulation (an ingenious parallel implementation is discussed in [11]). Due to the difficulty in parallelizing ILU(k), it is quite natural to combine ILU(k) and block-Jacobi, so much so that this combination constitutes PETSc's default parallel preconditioner [3]. The algorithm proposed in [2], whose parallel implementation we discuss in this article, seeks to combine the use of (sequential) ILU(K) with two ideas borrowed from domain decomposition methods: (i) the introduction of an interface that connects subdomains, allowing, as opposed to block-Jacobi, for the interaction between subdomains to be taken into account, and (ii) the introduction of a second level, associated to a coarse version of the problem, that speeds up the resolution of low frequency modes. These improvements come at the cost of greater communication, requiring a more involved parallel implementation.

The main contribution of the two-level preconditioner proposed in [2] is the fine preconditioner, as the coarse level component is a quite simple one. Accordingly, the main contribution in our article is the development of a careful parallel implementation of that fine part; nonetheless, we also take care of the coarse part by proposing low-cost PETSc-based parallel codes for its construction and application. Besides the parallel implementation, we present and discuss a set of performance tests of this preconditioner for solving synthetic and real-world oil reservoir simulation problems.

The paper is organized as follows. Section 2 introduces the notation used throughout the paper. Section 3 describes in detail the proposed two-level algebraic ILU(k)-based domain decomposition preconditioner, which we call iSchur. Its parallel implementation is detailed in Section 4, including the communication layout and strategies to minimize data transfer between processes, while results of performance experiments and comparisons with another preconditioner are presented and discussed in Section 5. The conclusion is drawn in Section 6 along with some future work directions.

2 NOTATION

In this section, we introduce some notation that will be necessary to define iSchur.

Consider the linear system of algebraic equations

$$Ax = b \tag{2.1}$$

arising from the discretization of a system of partial differential equations (PDEs) modeling the multiphase flow in porous media by a finite difference scheme with n gridcells. We denote by n_{dof} the number of degrees of freedom (DOFs) per gridcell, so that A is a matrix of dimension

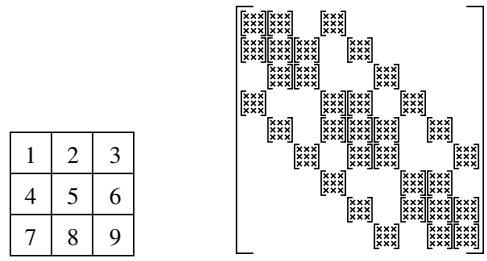
$n_{\text{dof}}n \times n_{\text{dof}}n$. For instance, for the standard *black-oil* formulation, see [24], the DOFs are oil pressure, oil saturation and water saturation, so that $n_{\text{dof}} = 3$.

It will be convenient to think of A as a *block-matrix*:

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix},$$

where a *block*, denoted a_{ij}, is a matrix of dimension $n_{\text{dof}} \times n_{\text{dof}}$.

The block-sparsity pattern of A is determined by the stencil of the finite-difference scheme used for the discretization of the PDEs. Figure 1 depicts a bidimensional domain discretized by a 3×3 mesh and the sparsity pattern of the associated matrix for the black-oil model, assuming a five-point finite-difference stencil. We say that two gridcells i and j are neighbors if either one of the blocks a_{ij} or a_{ji} is not null. For the five-point stencil, gridcells are neighbors when they share an edge. The gridcells and their respective indices are identified, in a way that Ω denotes either the



(a) 3×3 grid. (b) Associated matrix.

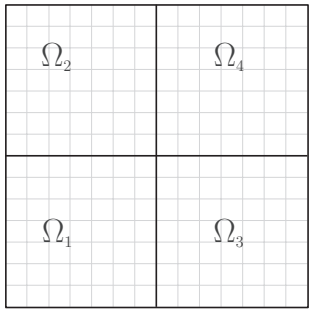
Figure 1: Discretization grid and associated matrix, assuming three unknowns per gridcell and a five-point stencil.

domain of the PDE or the set of indices $\{1,2,\ldots,n\}$. We introduce a disjoint partition of Ω, i.e., we break $\Omega = \{1,2,\ldots,n\}$ into P pieces $\Omega_1, \Omega_2, \ldots, \Omega_P$, in such a way that each gridcell belongs to exactly one subdomain Ω_k, see Figure 2. More precisely, we define

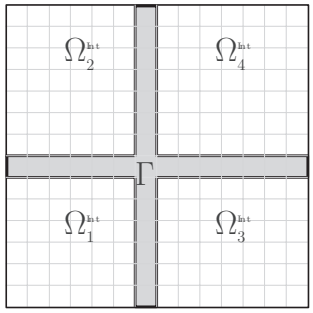
$$\{\Omega_J\}_{1 \le J \le P} \quad \text{s.t.} \quad \bigcup_{J=1}^{P} \Omega_J = \Omega \quad \text{and} \quad \Omega_I \cap \Omega_J = \emptyset \ \ \forall I \neq J. \tag{2.2}$$

We note that there are gridcells that, while belonging to one subdomain, have neighbors in other subdomains. Our domain decomposition approach requires the definition of disconnected subdomains, that can be dealt with in parallel. For that sake, we define a separator set, called interface and denoted by Γ, and (disconnected) subdomain interiors Ω_J^{Int}. A gridcell j is said to belong to Γ if it is in a subdomain Ω_J while being neighbor of at least one gridcell in another subdomain with greater index, i.e., Ω_K with $K > J$:

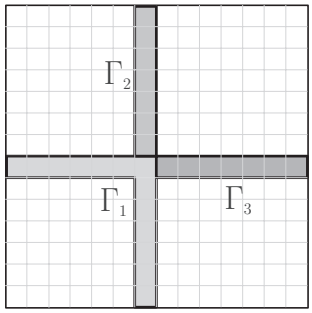
$$\Gamma = \left\{ j \in \Omega \middle| \exists k, J, K \text{ s.t. } j \in \Omega_J, k \in \Omega_K, K > J, (a_{jk} \neq 0 \text{ or } a_{kj} \neq 0) \right\}. \tag{2.3}$$



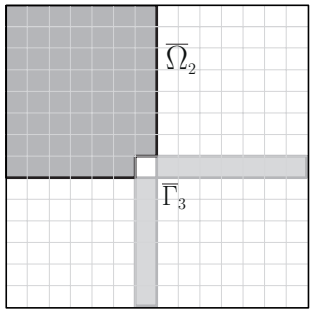
(a) Disjoint Subdomains Ω_J.



(b) Interface Γ and Subdomain Interiors Ω_J^{Int}.



(c) Local Interfaces Γ_J.



(d) Ext. Interface $\overline{\Gamma}_3$ and Ext. Subdomain $\overline{\Omega}_2$.

Figure 2: 2D Domain partitioned into 4 subdomains.

We now define the subdomain interior Ω_J^{Int} as the portion of Ω_J not in Γ:

$$\Omega_J^{\text{Int}} = \Omega_J - \Gamma, \tag{2.4}$$

see Figure 2. These definitions are just enough to ensure that if gridcell j is in Ω_J^{Int} and gridcell k is in Ω_K^{Int}, with $J \neq K$, then they are not neighbors.[1] Indeed, to fix the notation, assume $J < K$. Since $j \in \Omega_J^{\text{Int}} \subset \Omega_J$ and $k \in \Omega_K^{\text{Int}} \subset \Omega_K$, if j and k were neighbors, j would be in Γ by definition (2.3) and therefore not in Ω_J^{Int}.

We now define the local interface Γ_J associated with each subdomain Ω_J as the intersection of Γ and Ω_J, or equivalently,

$$\Gamma_J = \left\{ j \in \Omega_J \;\middle|\; \left(\exists K > J \text{ and } \exists k \in \Omega_K\right) \text{ s.t. } (a_{jk} \neq 0 \text{ or } a_{kj} \neq 0) \right\}. \tag{2.5}$$

Notice that $\{\Gamma_J\}_{1 \leq J \leq P}$ form a disjoint partition of Γ. See Figure 2.

Finally, we define extended subdomains $\overline{\Omega}_J$ and extended local interfaces $\overline{\Gamma}_J$, which incorporate the portions of the interface connected to the subdomain interior Ω_J^{Int}. We define $\overline{\Omega}_J$ as

$$\overline{\Omega}_J = \Omega_J \cup \left\{ k \in \Gamma \;\middle|\; \exists j \in \Omega_J^{\text{Int}} \text{ s.t. } (a_{jk} \neq 0 \text{ or } a_{kj} \neq 0) \right\}. \tag{2.6}$$

[1] Had the condition $K > J$ been dropped from definition (2.3), Γ would still be a separator set. But it would be unnecessarily large, which would yield a more expensive preconditioner.

and $\overline{\Gamma}_J$ as its restriction to Γ, i.e., $\overline{\Gamma}_J = \Gamma \cap \overline{\Omega}_J$, see Figure 2.

Notice that $\Gamma_J \subset \overline{\Gamma}_J \subset \Gamma$. We point out that $\overline{\Gamma}_J$ is the result of augmenting Γ_J with the gridcells of the interface Γ that are neighbors of gridcells in Ω_J. We refer to $\overline{\Gamma}_J$ as an extended interface, see Figure 2.

If the equations/variables are reordered, starting with the ones corresponding to Ω_1^{Int}, followed by the other Ω_J^{Int} and finally by Γ, then A has the following block-structure:

$$A = \left[\begin{array}{ccc|c} A_{11} & & & A_{1\Gamma} \\ & \ddots & & \vdots \\ & & A_{PP} & A_{P\Gamma} \\ \hline A_{\Gamma 1} & \cdots & A_{\Gamma P} & A_{\Gamma\Gamma} \end{array}\right], \tag{2.7}$$

where the submatrices A_{JJ} contain the rows and columns of A associated with Ω_J^{Int}, $A_{\Gamma\Gamma}$ the ones associated with Γ, $A_{J\Gamma}$ the rows associated with Ω_J^{Int} and the columns associated with Γ, and $A_{\Gamma J}$ the rows associated with Γ and the columns associated with Ω_J^{Int}. Therefore, denoting by $|S|$ the number of elements in a set S, the dimensions of A_{JJ}, $A_{\Gamma\Gamma}$, $A_{J\Gamma}$ and $A_{\Gamma J}$ are, respectively, $n_{\text{dof}}n_J \times n_{\text{dof}}n_J$, $n_{\text{dof}}m \times n_{\text{dof}}m$, $n_{\text{dof}}n_J \times n_{\text{dof}}m$, and $n_{\text{dof}}m \times n_{\text{dof}}n_J$, where $n_J = \left|\Omega_J^{\text{Int}}\right|$ and $m = |\Gamma|$. It is important to notice that, since $a_{jk} = 0$ for any $j \in \Omega_J^{\text{Int}}$ and $k \in \Omega_K^{\text{Int}}$ with $J \neq K$, the submatrices A_{JK} of rows associated with Ω_J^{Int} and columns associated with Ω_K^{Int} are all null (and therefore omitted in (2.7)). This block-diagonal structure of the leading portion of the matrix (which encompasses most of the variables/equations) allows for efficient parallelization of many tasks, and was the ultimate motivation of all the definitions above. We point out that although not necessary by the method, the implementation discussed in this work assumes that the matrix A is structurally symmetric[2], in this case the matrices A_{JJ} and $A_{\Gamma\Gamma}$ are also structurally symmetric. Furthermore, $A_{J\Gamma}$ and $A_{\Gamma J}^T$ have the same nonzero pattern.

3 DESCRIPTION OF THE TWO-LEVEL PRECONDITIONER

The goal of a preconditioner is to replace the linear system (2.1) by one of the equivalent ones:

$$(MA)x = Mb \quad \text{(left preconditioned) or}$$

$$(AM)y = b \quad \text{(right preconditioned, where } x = My\text{)}.$$

A good preconditioner shall render AM or MA much better conditioned than A (i.e., M should approximate, in a sense, A^{-1}) and, in order to be of practical interest, its construction and application must not be overly expensive.

We now present our preconditioner, which combines ideas from domain decomposition methods, DDM, and level-based Incomplete LU factorization, ILU(k). In DDM, one tries to build an approximation to the action of A^{-1} based on the (approximation of) inverses of smaller, local versions of A (subdomain-interior submatrices A_{JJ}, in our case). In this work, the action of the local inverses is approximated by ILU(k_{Int}), while other components required by the preconditioner (see equation (3.4)) are approximated with different levels (k_{Bord}, k_{Prod} or k_Γ). The motivation

[2]If A is structurally symmetric, then A^T and A have the same nonzero structure but are not necessarily equal.

for this approach is that ILU(k) has been widely used with success as a preconditioner in reservoir simulation and DDM has been shown to be an efficient and scalable technique for parallel architectures.

It is well established that in linear systems associated with parabolic problems, the high-frequency components of the error are damped quickly, while the low-frequency ones take many iterations to fade, see [31]. In reservoir simulation, the pressure unknown is of parabolic nature. A two-level preconditioner tackles this problem by combining a "fine" preconditioner component M_F, as the one mentioned in the previous paragraph, with a coarse component M_C, the purpose of which is to annihilate the projection of the error onto a coarse space (associated with the low frequencies). If the two components are combined multiplicatively (see [29]), the resulting two-level preconditioner is

$$M = M_F + M_C - M_F A M_C. \tag{3.1}$$

In Subsection 3.1 we present a ILU(k)-based fine component M_F and in Subsection 3.2 we describe a simple coarse component.

3.1 ILU(k) based Domain Decomposition

The fine component of the domain decomposition preconditioner is based on the following block LU factorization of A:

$$A = LU = \left[\begin{array}{ccc|c} L_1 & & & \\ & \ddots & & \\ & & L_P & \\ \hline B_1 & \cdots & B_P & I \end{array}\right] \left[\begin{array}{ccc|c} U_1 & & & C_1 \\ & \ddots & & \vdots \\ & & U_P & C_P \\ \hline & & & S \end{array}\right], \tag{3.2}$$

where $A_{JJ} = L_J U_J$ is the LU factorization of A_{JJ}, $B_J = A_{\Gamma J} U_J^{-1}$, $C_J = L_J^{-1} A_{J\Gamma}$ and

$$S = A_{\Gamma\Gamma} - \sum_{J=1}^{P} A_{\Gamma J} A_{JJ}^{-1} A_{J\Gamma} = A_{\Gamma\Gamma} - \sum_{J=1}^{P} B_J C_J, \tag{3.3}$$

is the Schur complement of A with respect to the interior points.

From the decomposition in (3.2), we can show that the inverse of A is

$$A^{-1} = \left[\begin{array}{ccc|c} U_1^{-1} & & & \\ & \ddots & & \\ & & U_P^{-1} & \\ \hline & & & I \end{array}\right] \left[\begin{array}{c|ccc} I & C_1 & & \\ & \vdots & & \\ & C_P & & \\ \hline & -I & & \end{array}\right] \left[\begin{array}{c|c} I & \\ \hline & S^{-1} \end{array}\right]$$
$$\left[\begin{array}{ccc|c} & I & & \\ \hline B_1 & \cdots & B_P & -I \end{array}\right] \left[\begin{array}{ccc|c} L_1^{-1} & & & \\ & \ddots & & \\ & & L_P^{-1} & \\ \hline & & & I \end{array}\right]. \tag{3.4}$$

We want to define a preconditioner M_F approximating the action of A^{-1} on a vector. Therefore, we need to define suitable approximations for L_J^{-1}, U_J^{-1}, B_J and C_J, $J = 1, \ldots, P$, and for

S^{-1}. These approximations are denoted by $\widetilde{L}_J^{-1}$, $\widetilde{U}_J^{-1}$, $\widetilde{B}_J$, $\widetilde{C}_J$, and S_F^{-1}, respectively. The fine preconditioner is then defined as

$$M_F = \left[\begin{array}{ccc|c} \widetilde{U}_1^{-1} & & & \\ & \ddots & & \\ & & \widetilde{U}_P^{-1} & \\ \hline & & & I \end{array}\right] \left[\begin{array}{ccc|c} & & & \widetilde{C}_1 \\ & I & & \vdots \\ & & & \widetilde{C}_P \\ \hline & & & -I \end{array}\right] \left[\begin{array}{ccc|c} & & & \\ & I & & \\ & & & \\ \hline & & & S_F^{-1} \end{array}\right]$$
$$\left[\begin{array}{ccc|c} & & & \\ & I & & \\ & & & \\ \hline \widetilde{B}_1 & \cdots & \widetilde{B}_P & -I \end{array}\right] \left[\begin{array}{ccc|c} \widetilde{L}_1^{-1} & & & \\ & \ddots & & \\ & & \widetilde{L}_P^{-1} & \\ \hline & & & I \end{array}\right]. \quad (3.5)$$

In the remaining of this subsection, we describe precisely how these approximations are chosen.

First we define $\widetilde{L}_J$ and $\widetilde{U}_J$ as the result of the incomplete LU factorization of A_{JJ} with level of fill k_{Int}, $[\widetilde{L}_J, \widetilde{U}_J] = \text{ILU}(A_{JJ}, k_{\text{Int}})$. In our numerical experiments, we used $k_{\text{Int}} = 1$, which is a usual choice in reservoir simulation. Even though $\widetilde{L}_J$ and $\widetilde{U}_J$ are sparse, $\widetilde{L}_J^{-1} A_{J\Gamma}$ and $\widetilde{U}_J^{-T} A_{\Gamma J}^T$ (which would approximate C_J and B_J^T) are not. To ensure sparsity, $\widetilde{C}_J \approx \widetilde{L}_J^{-1} A_{J\Gamma}$ is defined as the result of the incomplete triangular solve with multiple right-hand sides of the system $\widetilde{L}_J \widetilde{C}_J = A_{J\Gamma}$, by extending the definition of *level of fill* as follows. Let v_l and w_l be the sparse vectors corresponding to the l-th columns of $A_{J\Gamma}$ and $\widetilde{C}$ respectively, so that $\widetilde{L}_J w_l = v_l$. Based on the solution of triangular systems by forward substitution, the components of v_l and w_l are related by

$$w_{l_k} = v_{l_k} - \sum_{i=1}^{k-1} \widetilde{L}_{J_{ki}} w_{l_i}. \quad (3.6)$$

The level of fill-in of component k of w_l is defined recursively as

$$\text{Lev}(w_{l_k}) = \min\left\{ \text{Lev}(v_{l_k}), \min_{1 \le i \le (k-1)} \left\{ \text{Lev}(\widetilde{L}_{J_{ki}}) + \text{Lev}(w_{l_i}) + 1 \right\} \right\}, \quad (3.7)$$

where $\text{Lev}(v_{l_k}) = 0$ when $v_{l_k} \neq 0$ and $\text{Lev}(v_{l_k}) = \infty$ otherwise, and $\text{Lev}(\widetilde{L}_{J_{ki}})$ is the level of fill of entry ki in the ILU(k_{Int}) decomposition of A_{JJ} when $\widetilde{L}_{J_{ki}} \neq 0$ and $\text{Lev}(\widetilde{L}_{J_{ki}}) = \infty$ otherwise. The approximation $\widetilde{C}_J$ to $C_J = L_J^{-1} A_{J\Gamma}$ is then obtained by an incomplete forward substitution (IFS) with level of fill k_{Bord}, in which we do not compute any terms with level of fill greater than k_{Bord} during the forward substitution process. We denote $\widetilde{C}_J = \text{IFS}(\widetilde{L}_J, A_{J_\Gamma}, k_{\text{Bord}})$. Notice that when $k_{\text{Bord}} = k_{\text{Int}}$, $\widetilde{C}_J$ is what would result from a standard partial incomplete factorization of A. The approximation $\widetilde{B}_J$ to $B_J = A_{\Gamma J} \widetilde{U}_J^{-1}$ is defined analogously.

Similarly, in order to define an approximation $\widetilde{S}$ to S, we start by defining $F_J = \widetilde{B}_J \widetilde{C}_J$ and a level of fill for the entries of F_J,

$$\text{Lev}(F_{J_{kl}}) = \min\left\{ \text{Lev}(A_{\Gamma\Gamma_{kl}}), \min_{1 \le i \le m} \left\{ \text{Lev}(\widetilde{B}_{J_{ki}}) + \text{Lev}(\widetilde{C}_{J_{il}}) + 1 \right\} \right\}, \quad (3.8)$$

where $m = \#\Omega_J^{\text{Int}}$ is the number of columns in $\widetilde{B}_J$ and rows in $\widetilde{C}_J$, $\text{Lev}(A_{\Gamma\Gamma_{kl}}) = 0$ when $A_{\Gamma\Gamma_{kl}} \neq 0$ and $\text{Lev}(A_{\Gamma\Gamma_{kl}}) = \infty$, otherwise, and $\text{Lev}(\widetilde{C}_{J_{ki}})$ is the level of fill according to definition (3.7) when $\widetilde{C}_{J_{ki}} \neq 0$ and $\text{Lev}(C_{J_{ki}}) = \infty$, otherwise $\big(\text{Lev}(\widetilde{B}_{J_{il}})$ is defined in the same way$\big)$. Next, $\widetilde{F}_J$ is defined as the matrix obtained keeping only the entries in F_J with level less than or equal to k_{Prod} according to (3.8). We refer to this *incomplete product*(IP) as $\widetilde{F}_J = \text{IP}(\widetilde{B}_J, \widetilde{C}_J, k_{\text{Prod}})$.

$\widetilde{S}$ is then defined as

$$\widetilde{S} = A_{\Gamma\Gamma} - \sum_{J=1}^{P} \widetilde{F}_J. \tag{3.9}$$

While $\widetilde{S}$ approximates S, we need to define an approximation for S^{-1}. Since $\widetilde{S}$ is defined on the global interface Γ, it is not practical to perform ILU on it. Instead, we follow the approach employed in [7] and define for each subdomain a local version of $\widetilde{S}$,

$$\widetilde{S}_J = R_J \widetilde{S} R_J^T, \tag{3.10}$$

where $R_J : \Gamma \to \overline{\Gamma}_J$ is a restriction operator such that $\widetilde{S}_J$ is the result of pruning $\widetilde{S}$ so that only the rows and columns associated with $\overline{\Gamma}_J$ remain. More precisely, if $\{i_1, i_2, \ldots, i_{n_{\overline{\Gamma}_J}}\}$ is a list of the nodes in Γ that belong to $\overline{\Gamma}_J$, then the k-th row of R_J is $e_{i_k}^T$, the i_k-th row of the $n_\Gamma \times n_\Gamma$ identity matrix,

$$R_J = \begin{bmatrix} e_{i_1}^T \\ \vdots \\ e_{i_{n_\Gamma}}^T \end{bmatrix}.$$

Finally, our approximation S_F^{-1} to S^{-1} is defined as

$$S_F^{-1} = \sum_{J=1}^{P} T_J (L_{\widetilde{S}_J} U_{\widetilde{S}_J})^{-1} R_J \approx \sum_{J=1}^{P} T_J \widetilde{S}_J^{-1} R_J, \tag{3.11}$$

where $L_{\widetilde{S}_J}$ and $U_{\widetilde{S}_J}$ are given by $ILU(k_\Gamma)$ of $\widetilde{S}_J$. Here $T_J : \overline{\Gamma}_J \to \Gamma$ is an extension operator that takes values from a vector that lies in $\overline{\Gamma}_J$, scales them by $w_1^J, \ldots, w_{n_{\overline{\Gamma}_J}}^J$ (which we call weights), and places them in the corresponding position of a vector that lies in Γ. Therefore, using the same notation as before, the k-th column of T_J is $w_k^J e_{i_k}$,

$$T_J = \begin{bmatrix} w_1^J e_{i_1} & \cdots & w_{n_\Gamma}^J e_{i_{n_\Gamma}} \end{bmatrix}.$$

Different choices for the weights gives rise to different options for T_J, our choice is one that avoids communication among processes and is defined as follows.

$$w_k^J = \begin{cases} 1, \text{ if the } i_k\text{-th node of } \Gamma \text{ belongs to } \Gamma_J \\ 0, \text{ if the } i_k\text{-th node of } \Gamma \text{ belongs to } \overline{\Gamma}_J - \Gamma_J. \end{cases}$$

We note that $T_J, J = 1, \ldots, P$, form a partition of unity. Our approach is based on the Restricted Additive Schwarz method, proposed on [5].

3.2 Coarse Space Correction

We define a coarse space spanned by the columns of an $(n \times P)$ matrix that we call R_0^T. The J-th column of R_0^T is associated with the extended subdomain $\overline{\Omega}_J$ and its i-th entry is

$$(R_0^T)_{iJ} = \begin{cases} 0, \text{if node } i \text{ is not in } \overline{\Omega}_J \text{ and} \\ \mu_{\Omega_i}^{-1}, \text{if node } i \text{ is in } \overline{\Omega}_J, \end{cases}$$

where the i-th entry of μ_Ω, denoted μ_{Ω_i}, counts how many extended subdomains the i-th node belongs to. Notice that $(R_0^T)_{iJ} = 1 \ \ \forall i \in \Omega_J^{\text{Int}}$ and that the columns of R_0^T form a partition of unity, in the sense that their sum is a vector with all entries equal to 1.

We define M_C by the formula

$$M_C = R_0^T (R_0 A R_0^T)^{-1} R_0. \tag{3.12}$$

Notice that this definition ensures that $M_C A$ is a projection onto range(R_0^T) and for A symmetric positive definite this projection is A-orthogonal. Since $R_0 A R_0^T$ is small $(P \times P)$, we use exact LU (rather than ILU) when applying its inverse.

Summing up, the complete preconditioner has two components: one related to the complete grid, called M_F in equation (3.5), and another related to the coarse space (3.12). It is possible to subtract a third term that improves the performance of the whole preconditioner, although increasing its computational cost. The combined preconditioners are written down as

$$M = M_F + M_C \quad \text{or} \tag{3.13}$$

$$M = M_F + M_C - M_F A M_C. \tag{3.14}$$

This formulation implies that the preconditioners will be applied additively (3.13) or multiplicatively (3.1) and can be interpreted as having two levels, see [7]. In the following sections we address this two-level algebraic ILU(k)-based domain decomposition preconditioner by iSchur (for "incomplete Schur").

4 PARALLEL IMPLEMENTATION

In this section we discuss the parallel implementation of the preconditioner, considering data locality, communication among processes, and strategies to avoid communication.

We use distributed memory model, so the matrix A is distributed among the processes and, as we use PETSc, the rows corresponding to the elements of each subdomain, including interior and local interfaces, reside in the same process, see Figure 3 for a four domain representation.

4.1 Preconditioner Setup

Algorithm 1 describes the construction of the fine and coarse components of the preconditioner.

Step 1 of Algorithm 1 is the incomplete LU factorization. Because each subdomain is completely loaded in only one process and there is no connections between subdomain interiors, this step can be done in parallel in each process without communication. We use a native and optimized PETSc routine that computes the incomplete LU factorization.

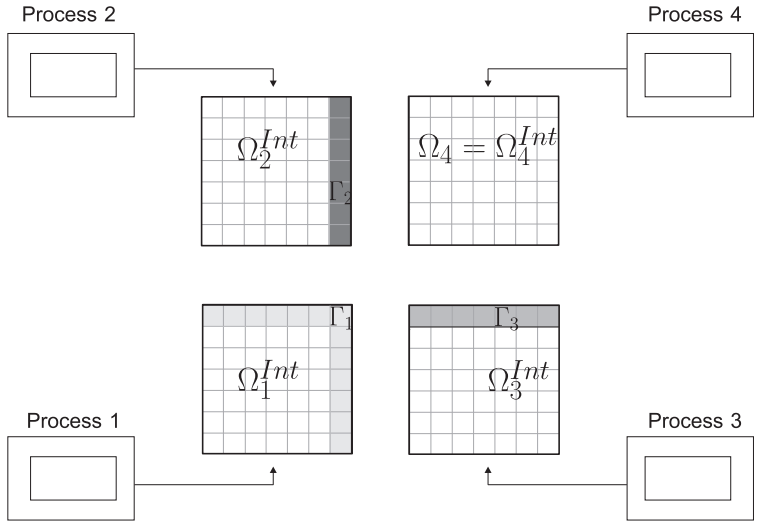


Figure 3: Subdomain and matrix distribution among processes.

Algorithm 1: Preconditioner Setup

Input: A, k_{Int}, k_{Bord}, k_{Prod}
Output: $\widetilde{L}_J$, $\widetilde{U}_J$, $\widetilde{C}_J$, $\widetilde{B}_J$, $L_{\widetilde{S}_J}$, $U_{\widetilde{S}_J}$, L_C, U_C
Fine Component

1: $[\widetilde{L}_J, \widetilde{U}_J] = \text{ILU}(A_{JJ}, k_{\text{Int}}), \quad J = 1, \ldots, P;$

2: $\widetilde{C}_J = \text{IFS}(\widetilde{L}_J, A_{J\Gamma}, k_{\text{Border}}), \quad J = 1, \ldots, P;$

3: $\widetilde{B}_J = \left(\text{IFS}(\widetilde{U}_J^T, A_{\Gamma J}^T, k_{\text{Border}}) \right)^T, \quad J = 1, \ldots, P;$

4: $\widetilde{S} = A_{\Gamma\Gamma} - \sum_{J=1}^{P} \text{IP}(\widetilde{B}_J, \widetilde{C}_J, k_{\text{Prod}})$

5: $\widetilde{S}_J = R_J \widetilde{S} R_J^T, \quad J = 1, \ldots, P;$

6: $[L_{\widetilde{S}_J}, U_{\widetilde{S}_J}] = \text{ILU}(\widetilde{S}_J, k_\Gamma), \quad J = 1, \ldots, P;$

Coarse Component

7. $[L_C, U_C] = \text{LU}(R_0 A R_0^T).$

In Step 2, $\widetilde{C}_J = \text{IFS}(\widetilde{L}_J, A_{J\Gamma}, 0)$ computes an approximation $\widetilde{C}_J = \widetilde{L}^{-1} A_{J\Gamma}$ through a level zero incomplete triangular solver applied to the columns of $A_{J\Gamma}$. For each column a_i of $A_{J\Gamma}$, we take $\mathscr{I}_i$, the set of indices of the nonzero elements of a_i. Then we solve $c_i(\mathscr{I}_i) = \widetilde{L}(\mathscr{I}_i, \mathscr{I}_i) \backslash a_i(\mathscr{I}_i)$, which is a small dense triangular system, where c_i is the i-th column of $\widetilde{C}_J$. See Figure 4. The

light gray rows and columns represent the $\mathscr{I}_i$ indices applied to the rows and columns of the matrix $\widetilde{L}_J$ and the dark gray elements are the ones in $\widetilde{L}(\mathscr{I}_i, \mathscr{I}_i)$. We use a PETSc routine to get the submatrices $\widetilde{L}(\mathscr{I}_i, \mathscr{I}_i)$ as a small dense matrix and use our own dense triangular solver to compute $c_i(\mathscr{I}_i)$, which is a small dense vector, and then use another PETSc routine to store it in the correct positions of c_i, which is a column of the sparse matrix $\widetilde{C}_J$. We observe that the nonzero columns of $A_{J\Gamma}$ that take part in these operations, in each subdomain, are related to the extended interface, but are stored locally.

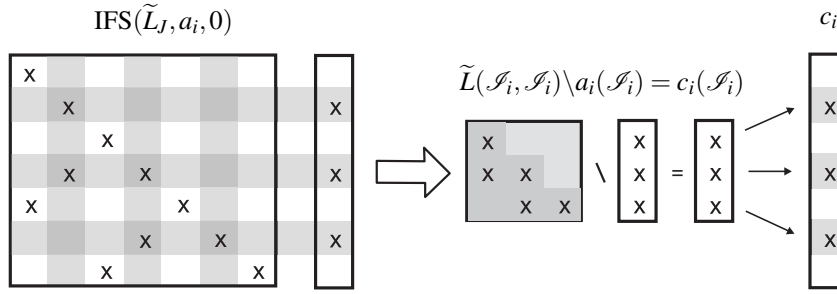


Figure 4: Incomplete Forward Substitution.

Note that $\widetilde{L}_J$ and $A_{J\Gamma}$ (see Equation (2.5)) are in process J, as can be seen in Figure 3, which means that Step 2 is done locally without any communication.

Step 3 computes $\widetilde{B}_J$ also through the incomplete forward substitution using the same idea described for Step 2 , but $A_{\Gamma J}$ is not stored entirely in process J. Actually, $A_{\Gamma J}$ has nonzero elements in the rows associated with the extended interface $\overline{\Gamma}_J$, therefore the elements of $A_{\Gamma J}$ are distributed among process J and its neighbors. So each process has to communicate only with its own neighbors to compute $\widetilde{B}_J$. The matrix $\widetilde{B}_J$ is stored locally in process J.

The communication layout of Step 3 is illustrated in an example for four subdomains in Figure 5, where the arrows show the data transfer flow. In this example, using, for instance, five-point centered finite differences, each subdomain has at most two neighbors, but in different 2-D or 3-D grids and with different discretization schemes, each subdomain can have more neighbors.

The Step 4 computes the Schur complement through an incomplete product as described by equation (3.9). The submatrix $A_{\Gamma\Gamma}$ is formed by the subdomains interfaces and therefore is distributed among the processes. Each process computes locally its contribution IP$(\widetilde{B}_J, \widetilde{C}_J, k_{Prod})$ and subtracts globally from $A_{\Gamma\Gamma}$ to build up $\widetilde{S}$, which is also distributed among the processes in the same way $A_{\Gamma\Gamma}$ is. Again, each subdomain communicates with their neighbors following the pattern illustrated in Figure 5, but with arrows pointers reversed, and there is a synchronization barrier to form the global matrix $\widetilde{S}$.

In Step 5 and 6, each process takes a copy of the local part of $\widetilde{S}$ relative to the extended interface $\overline{\Gamma}_J$ as in (3.10), communicating only with neighbors, and makes an incomplete LU factorization. The use of local copies avoids communication during the parallel application of $\widetilde{S}$ throughout the linear solver iterations when using the RAS [5] version of the preconditioner.

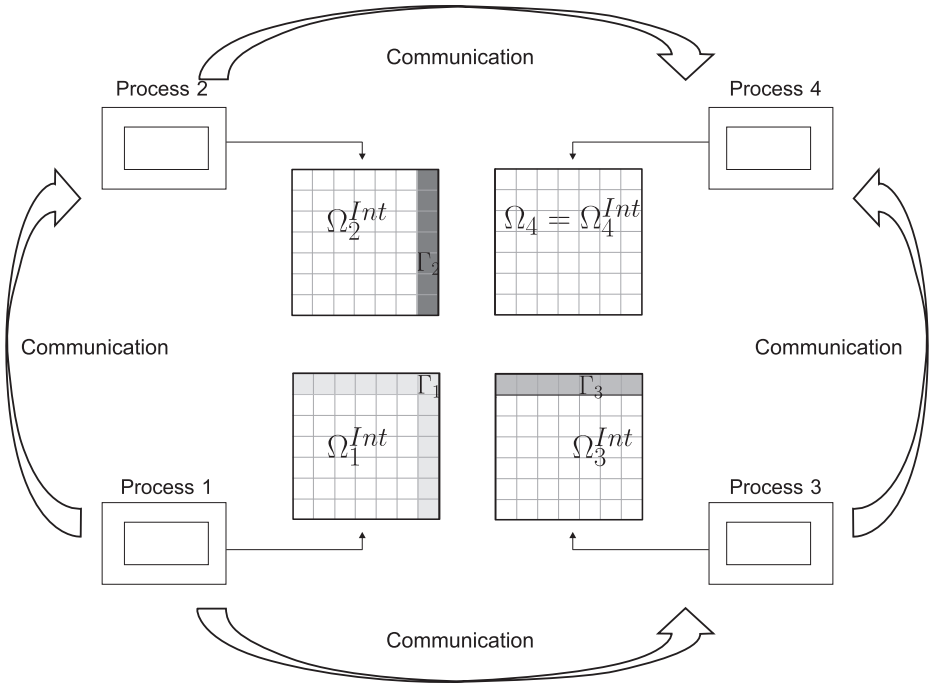


Figure 5: Domain distribution among processes.

In Step 7 the matrix $R_0 A R_0^T$ is initially formed as a parallel matrix, distributed among the processes. Local copies of this matrix is made in all processes and then a LU factorization is done redundantly. The order of this matrix is P, which is in general quite small.

4.2 Preconditioner Application

Algorithm 2 describes the application of the preconditioner M to the vector r, i.e., the computation of $z = Mr$. We define two additional restriction operators, $R_{\Omega_J} : \Omega \to \Omega_J^{\text{Int}}$ and $R_\Gamma : \Omega \to \Gamma$. These restriction operators are needed to operate on the correct entries of the vector, it can be seen as a subset of indices in the implementation.

Step 1 of Algorithm 2 solves a triangular linear system in each process involving the $\widetilde{L}_J$ factor and the local interior portion of the residual, obtaining z_J. This process is done locally and no communication is necessary because the residual r is distributed among the processes following the same parallel layout of the matrix.

Step 2 applies $\widetilde{B}_J$ multiplying z_J and subtracting from r_Γ obtaining z_Γ and the communication between the neighbor subdomains is necessary only in r_Γ and z_Γ because $\widetilde{B}_J$ is stored locally in process J.

The Schur complement application is done in Step 3. Because each process has a local copy of its parts of $\widetilde{S}$ factored into $L_{\widetilde{S}}$ e $U_{\widetilde{S}}$, only the communication in z_Γ between the neighbor subdomains is necessary. As described above, T_J is a diagonal operator, so we can compute it as a vector,

Algorithm 2: Preconditioner Application

Here we adopt the Matlab-style notation $A \setminus b$ to denote the exact solution of $Ax = b$.
Input: $r, \widetilde{L}_J, \widetilde{U}_J, \widetilde{C}_J, \widetilde{B}_J, L_{\widetilde{S}_J}, U_{\widetilde{S}_J}, L_C, U_C$
Output: z

1: $z_J = \widetilde{L}_J \setminus (R_{\Omega_J} r), \quad J = 1, \ldots, P;$

2: $z_\Gamma = R_\Gamma r - \sum_{J=1}^{P} \widetilde{B}_J z_J$

3: $z_\Gamma = \sum_{J=1}^{P} T_J \left(U_{\widetilde{S}_J} \setminus \left(L_{\widetilde{S}_J} \setminus (R_J z_\Gamma) \right) \right);$

4: $z_J = \widetilde{U}_J \setminus \left(z_J - \widetilde{C}_J z_\Gamma \right), \quad J = 1, \ldots, P;$

5: $z_F = R_\Gamma^T z_\Gamma + \sum_{J=1}^{P} R_{\Omega_J}^T z_J$

Coarse correction:

6. $z = z_F + R_0^T \left(U_C \setminus (L_C \setminus R_0 r) \right).$

which is applied through an element wise product. The result is then summed globally in z_Γ. At this point there is a synchronization barrier among all processes.

In Step 4 we apply the $\widetilde{U}_J$ factor to $z_J - \widetilde{C}_J z_\Gamma$. The matrix-vector product $\widetilde{C}_J z_\Gamma$ requires communication among neighbors in z_Γ and then the triangular solver is done locally with no communication.

The vectors z_J and the vector z_Γ are gathered in z_F in Step 5, where z_F is the residual preconditioned by the fine component of the preconditioner.

Step 6 applies the coarse component, completing the computation of the preconditioned residual. Triangular solvers involved are local because there are local copies of L_C and U_C in each process, so no communication is necessary.

5 COMPARISON TESTS

We use two metrics to evaluate the preconditioners: the number of iterations and the solution time. In this section we describe the platform where the tests were run, present the results of the proposed preconditioner, and compare with a native PETSc preconditioner.

We consider five matrices from reservoir simulation problems. They are linear systems arising from applying Newton's method to solve the nonlinear equations discretizing the system of PDEs that model multiphase flow in porous media. Matrices F8, U26, U31 and Psalt were generated from a Petrobras reservoir simulation tool. The Psalt problem is a synthetic presalt reservoir

model. The fifth matrix is the standard SPE10 model [10] with over a million active cells. All the problems are black oil models using fully implicit method and there are four degrees of freedom per grid cell.

Table 1 shows the problems' size considering the total number of cells and the size of their respective matrices.

Table 1: Test case problems.

Matrix	Problem Size	Matrix Size
F8	182558	730232
U26	408865	1635460
U31	617459	2469836
Psalt	765620	3062480
SPE10	1094421	4377684

For both preconditioners the problem was solved using native PETSc right preconditioned GMRES method [27], with relative residual tolerance of 1e-4, maximum number of iterations of 1000, and restarting at every 30^{th} iteration. The mesh was partitioned using PT-Scotch [9]. ISchur has fill-in level 1 in the interior of the subdomains and zero in the interfaces, namely $k_{\text{Int}} = 1$, $k_{\text{Bord}} = k_{\text{Prod}} = k_{\Gamma} = 0$, as this specific configuration presented a better behavior in preliminary Matlab tests.

We compare the proposed two-level preconditioner with PETSc's native block Jacobi preconditioner (see [3]) combined (as in equation (3.1)) with the same coarse component presented in this paper. We refer to this combination simply as block Jacobi. We note that block-Jacobi/ILU is the default preconditioner in PETSc and is a common choice of preconditioner in reservoir simulation, which makes it an appropriate benchmark.

The first set of tests was run on an Intel Xeon(R) 64-bit CPU E5-2650 v2 @ 2.60GHz with 16 cores (hyper-threading disabled), 20MB LLC, 64GB RAM (ECC enabled). The operating system is a Debian GNU/Linux distribution running kernel version 3.2.65. The compiler is Intel Parallel Studio XE 2015.3.187 Cluster Edition.

The iSchur preconditioner, both in setup and application phases, requires more computation and communication than block Jacobi, so it has to present a pronounced reduction in number of iterations compared with block Jacobi in order to be competitive. Table 2 shows the number of iterations and the total time, in seconds, for both preconditioners, for all the tested problems, ranging from one to sixteen processes. For each test, the best results are highlighted in gray. We can observe that the iSchur preconditioner does reduce the number of iterations compared to block Jacobi for all problems in any number of processes. For one process (sequential execution), both preconditioners are equal, as there are no interface points, so we show the results just for the sake of measuring speed up and scalability. In average, iSchur does 80% as many iterations as block Jacobi. In this set of experiments, iSchur was more robust than block Jacobi, as the latter failed to converge in two experiments.

We also observe that, for some of the matrices, the number of iterations increases with the number of processes. This is to be expected, since block Jacobi neglects larger portions of the matrix as

the size of the subdomains decreases. A correlated fact occurs with iSchur. In this case as we are using different levels of fill in the interface related parts and in the interior parts, more information is neglected with more subdomains, as the number of interface points increases.

Table 2: Number of iterations and the total time for both preconditioners from 1 to 16 processes. (**NC**) stands for "not converged".

Mat	Prec	Number of Iterations Processes					Total Time (s) Processes				
		1	2	4	8	16	1	2	4	8	16
F8	iS	104	113	106	92	102	10.8	7.9	4.1	2.1	1.8
	bJ		119	109	125	135		6.1	3.0	1.9	1.7
U26	iS	92	131	199	218	303	22.0	19.8	14.5	9.1	9.9
	bJ		252	**NC**	404	349		28.2	**NC**	13.3	9.7
U31	iS	69	110	449	517	864	28.0	28.8	48.7	32.1	43.1
	bJ		124	593	639	**NC**		24.1	56.7	35.4	**NC**
Psalt	iS	34	68	57	72	54	17.8	24.3	11.9	8.2	5.6
	bJ		87	76	117	76		19.7	9.2	8.0	5.0
SPE10	iS	116	135	148	148	176	71.5	53.9	30.9	18.0	17.5
	bJ		142	149	164	198		43.6	24.1	15.4	16.1

Table 2 also shows the total time of the solution using both preconditioners. We can see that despite the reduction in the number of iterations the solver time of iSchur is smaller than block Jacobi only in a few cases. One of the main reasons is that the iSchur setup time is greater than block Jacobi's, especially when running on few processes. Although iSchur's setup is intrinsically more expensive than block Jacobi's, we believe that this part of our code can still be further optimized to reduce this gap.

In reservoir simulation, the sparsity pattern of the Jacobian matrix only changes when the operating conditions of the wells change, and therefore it remains the same over the course of many nonlinear iterations. This is an opportunity to spare some computational effort in the setup phase, since in this case the symbolic factorization does not need to be redone. We can see in Table 3 the time comparison when the symbolic factorization is not considered; in this scenario iSchur is faster in most cases. Furthermore, block Jacobi is a highly optimized, mature code (the default PETSc preconditioner), while iSchur could possibly be further optimized.

In order to examine the behavior of the preconditioners as the number of subdomains increases, we ran tests up to 512 processes on a cluster consisting of nodes with 16GB RAM and two Intel Xeon E5440 Quad-core processors each, totaling 8 CPU cores per node. The results are depicted in Figure 6. We see that iSchur consistently outperforms block Jacobi in terms of number of iterations. Nonetheless, the behaviour of both preconditioners, with respect to scalability, cannot be predicted from these experiments. Tests with more MPI processes, in a larger cluster, are still required.

Table 3: Time of iterative solver without the symbolic factorization.

Iterative Solver Time (s)						
Mat	Prec	Processes				
		1	2	4	8	16
F8	iS	10.5	6.3	3.3	1.7	1.6
	bJ		6.1	3.0	1.9	1.7
U26	iS	21.7	16.2	12.7	8.2	9.4
	bJ		28.1	NC	13.3	9.7
U31	iS	27.5	22.8	45.6	30.5	42.2
	bJ		23.9	56.7	35.3	NC
Psalt	iS	17.3	17.5	8.4	6.4	4.6
	bJ		19.6	9.2	8.0	5.0
SPE10	iS	70.8	44.2	26.0	15.5	16.1
	bJ		43.4	24.0	15.3	16.1

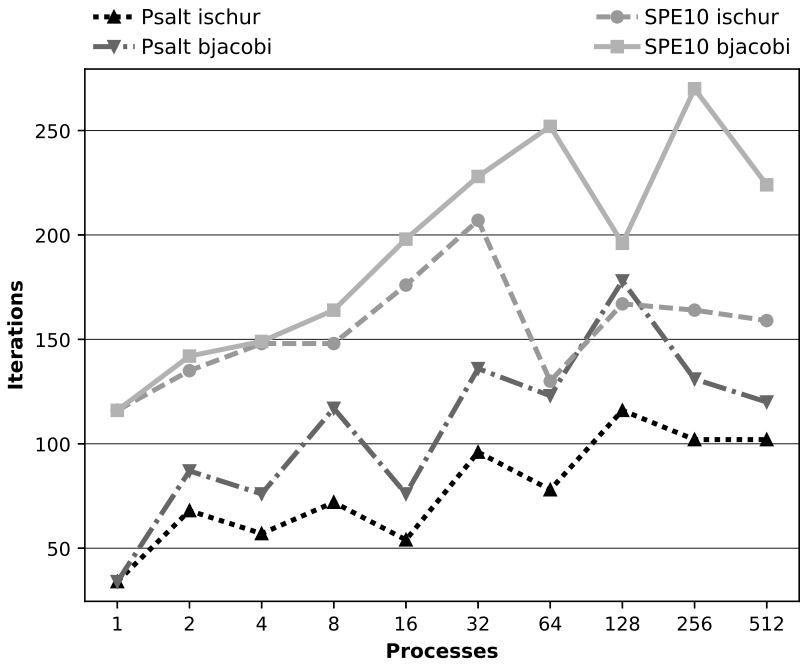


Figure 6: Comparison between block Jacobi and iSchur up to 512 processes.

6 CONCLUSIONS AND REMARKS

The presented preconditioner successfully reduced the number of iterations for the target reservoir problems compared to block Jacobi for the full range of partitions tested.

The scalability of both preconditioners is an on-going research as tests in a larger cluster are still necessary, but these initial results indicate a little advantage to iSchur.

For the current implementation the total solution time, setup and iterative solver, of the iSchur is smaller in only a few cases, where one important factor is the inefficiency in the setup step. The block Jacobi in PETSc is highly optimized while our implementation has room for improvement.

Removing the symbolic factorization time, considering the case when it is done once and used multiple times, which is usual in reservoir simulation, the solution using the iSchur preconditioner is faster for most tested cases. It shows that the presented preconditioner can be a better alternative than block Jacobi for the finer component when using two-level preconditioners to solve large-scale reservoir simulation problems. Furthermore, we deem that the application part of the code can still be optimized.

RESUMO. Discutimos a implementação paralela de um precondicionador algébrico de decomposição de domínios em dois níveis baseado em ILU(k), utilizando a biblioteca PETSc. Apresentamos estratégias para melhorar a performance, minimizando a comunicação entre processos durante as fases de construção e de aplicação. Comparamos, na solução de sistemas lineares provenientes de problemas de simulação de reservatórios, a nossa implementação com um precondicionador padrão do PETSc. Mostramos que, para alguns casos, nossa implementação apresenta um desempenho superior.

Palavras-chave: Precondicionador de dois níveis, decomposição de domínio, métodos de Krylov, sistemas lineares, paralelismo, PETSc.

REFERENCES

[1] T. M. Al-Shaalan, H. Klie, A.H. Dogru, & M. F. Wheeler. Studies of robust two stage preconditioners for the solution of fullyimplicit multiphase flow problems. In *SPE Reservoir Simulation Symposium, 2-4 February, The Woodlands, Texas*, number SPE-118722 in MS, (2009).

[2] D. A. Augusto, L. M. Carvalho, P. Goldfeld, I. C. L. Nievinski, J.R.P. Rodrigues, & M.Souza. An algebraic ILU(k) based two-level domain decomposition preconditioner. In *Proceeding Series of the Brazilian Society of Computational and Applied Mathematics*, **3** (2015), 010093–1– 010093–7.

[3] S. Balay, S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, K. Rupp, B.F. Smith, S. Zampini, H. Zhang, & H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.7, Argonne National Laboratory, (2016).

[4] M. Bollhöfer & V. Mehrmann. Algebraic multilevel methods and sparse approximate inverses. *SIAM Journal on Matrix Analysis and Applications*, **24**(1) (2002), 191–218.

[5] X. Cai & M. Sarkis. A Restricted Additive Schwarz Preconditioner for General Sparse Linear Systems *SIAM Journal on Scientific Computing*, **21**(2) (1999), 792–797.

[6] H. Cao, H. Tchelepi, J. Wallis & H. Yardumian. Parallel scalable unstructured CPR-type linear solver for reservoir simulation. In *SPE Annual Technical Conference and Exhibition, 9-12 October 2005, Dallas, Texas*. Society of Petroleum Engineers, (2005).

[7] L. M. Carvalho, L. Giraud & P. L. Tallec. Algebraic two-level preconditioners for the Schur complement method, *SIAM Journal on Scientific Computing*, **22**(6) (2001), 1987–2005.

[8] T. F. Chan, J. Xu & L. Zikatanov. An agglomeration multigrid method for unstructured grids. *Contemporary Mathematics*, **218** (1998), 67–81.

[9] C. Chevalier & F. Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering *Parallel computing*, **34**(6) (2008), 318–331.

[10] M. A. Christie & M. J. Blunt. Tenth SPE comparative solution project: A comparison of upscaling techniques in SPE Reservoir Simulation Symposium, (2001).

[11] D. A. Collins, J.E. Grabenstetter & P. H. Sammon. A Shared-Memory Parallel Black-Oil Simulator with a Parallel ILU Linear Solver. SPE-79713-MS. In *SPE Reservoir Simulation Symposium*, Houston, 2003. Society of Petroleum Engineers.

[12] L. Formaggia & M. Sala. *Parallel Computational Fluids Dynamics. Practice and Theory.*, chapter Algebraic coarse grid operators for domain decomposition based preconditioners. Elsevier, (2002), 119–126.

[13] S. Gries, K. Stüben, G. L. Brown, D. Chen & D. A. Collinns. Preconditioning for efficiently applying algebraic multigrid in fully implicit reservoir simulations. *SPE Journal*, **19**(04): 726–736, August 2014. SPE-163608-PA.

[14] H. Jackson, M. Taroni & D. Ponting. A two-level variant of additive Schwarz preconditioning for use in reservoir simulation. *arXiv preprint arXiv:1401.7227*, (2014).

[15] P. Jenny, S. Lee & H. Tchelepi. Multi-scale finite-volume method for elliptic problems in subsurface flow simulation. *Journal of Computational Physics*, **187**(1) (2003), 47– 67.

[16] P. Jenny, S. Lee & H. Tchelepi. Adaptive fully implicit multi-scale finite-volume method for multi-phase flow and transport in heterogeneous porous media. *Journal of Computational Physics*, **217**(2) (2006), 627–641.

[17] P. Jenny, S. H. Lee & H. A. Tchelepi. Adaptive multiscale finite-volume method for multiphase flow and transport in porous media. *Multiscale Modeling & Simulation*, **3**(1) (2005), 50–64.

[18] A. Manea, J. Sewall & H. Tchelepi. Parallel multiscale linear solver for reservoir simulation. Lecture Notes - slides - 4th SESAAI Annual Meeting Nov 5, 2013, (2013).

[19] A. Muresan & Y. Notay. Analysis of aggregation-based multigrid. *SIAM Journal on Scientific Computing*, **30**(2) (2008), 1082–1103.

[20] R. Nabben & C. Vuik. A comparison of deflation and coarse grid correction applied to porous media flow. *SIAM J. Numer. Anal.*, **42** (2004), 1631–1647.

[21] F. Nataf, H. Xiang & V. Dolean. A two level domain decomposition preconditioner based on local Dirichlet-to-Neumann maps. *Comptes Rendus Mathematique*, **348**(21-22) (2010), 1163–1167.

[22] Y. Notay. Algebraic multigrid and algebraic multilevel methods: a theoretical comparison. *Numer. Linear Algebra Appl*, **12**(5-6) (2005), 419–451.

[23] Y. Notay. Algebraic analysis of two-grid methods: The nonsymmetric case. *Numerical Linear Algebra with Applications*, **17**(1) (2010), 73–96.

[24] D. W. Peaceman Fundamentals of Numerical Reservoir Simulation. *Elsevier*, (1977).

[25] A. Quarteroni & A. Valli. *Applied and Industrial Mathematics: Venice - 1, 1989*, chapter Theory and Application of Steklov-Poincaré Operators For Boundary-Value Problems. Kluwer, (1991), 179–203.

[26] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2nd edition, (2003).

[27] Y. Saad & M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems *SIAM Journal on scientific and statistical computing*, **7**(3) (1986), 856–869.

[28] Y. Saad & J. Zhang. BILUM: Block versions of multielimination and multilevel ILU preconditioner for general sparse linear systems. *SIAM Journal on Scientific Computing*, **20**(6) (1999), 2103–2121.

[29] B. Smith, P. Bjørstad & W. Gropp. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, New York, 1st edition, (1996).

[30] J. Tang, R. Nabben, C. Vuik & Y. Erlangga. Theoretical and numerical comparison of various projection methods derived from deflation, domain decomposition and multigrid methods. Reports of the Department of Applied Mathematical Analysis 07-04, Delft University of Technology, January 2007.

[31] U. Trottenberg, C. Oosterlee & A. Schuller. *Multigrid*. Academic Press, Inc. Orlando, USA, (2001).

[32] C. Vuik & J. Frank. Coarse grid acceleration of a parallel block preconditioner. *Future Generation Computer Systems*, JUN, **17**(8) (2001), 933–940.

[33] J. Wallis, R. Kendall & T. Little. Constrained residual acceleration of conjugate residual methods. In *SPE Reservoir Simulation Symposium*, number SPE 13563 in 8 th Symposium on Reservoir Simulation, Dallas, 1985. Society of Petroleum Engineers.