CARI'10

Parallel GMRES with a multiplicative Schwarz preconditioner

Désiré NUENTSA WAKAM[†], **Guy-Antoine** ATENEKENG-KAHOU[‡§]

[†] INRIA, Campus de Beaulieu,
35042 Rennes Cedex, France,
desire.nuentsa_wakam@inria.fr

[‡] INRIA and LRI, Parc Orsay Université,
91893 Orsay Cedex, France
Guy_Antoine.Atenekeng_Kahou@inria.fr
[§]Faculty of sciences
University of Dschang, Cameroon
atenekeng@yahoo.com

ABSTRACT. This paper presents a robust hybrid solver for linear systems that combines a Krylov subspace method as accelerator with a Schwarz-based preconditioner. This preconditioner uses an explicit formulation associated to one iteration of the multiplicative Schwarz method. The Newton-basis GMRES, which aim at expressing a good data parallelism between subdomains is used as accelerator. In the first part of this paper, we present the pipeline parallelism that is obtained when the multiplicative Schwarz preconditioner is used to build the Krylov basis for the GMRES method. This is referred as the *first level of parallelism*. In the second part, we introduce a *second level of parallelism* inside the subdomains. For Schwarz-based preconditioners, the number of subdommains are keeped small to provide a robust solver. Therefore, the linear systems associated to subdomains are solved efficiently with this approach. Numerical experiments are performed on several problems to demonstrate the benefits of using these two levels of parallelism in the solver, mainly in terms of numerical robustness and global efficiency.

RÉSUMÉ. Cet article présente un solveur hybride robuste pour des systèmes linéaires. Ce solveur parallèle construit un préconditionneur de type Schwarz pour accélerer une méthode basée sur les sous-espaces de Krylov. Le préconditionneur est défini à partir d'une formulation explicite correspondant à une itération de Schwarz multiplicatif. Dans le but de réduire les communications et les dépendences entre les sous-domaines, nous utilisons la version de GMRES qui dissocie la construction de la base de Krylov et son orthogonalisation. Nous présentons dans un premier temps le parallélisme qui est obtenu lorsque ce préconditionneur Schwarz multiplicatif est utilisé dans la construction de la base de Krylov. C'est le *premier niveau de parallélisme*. Dans la deuxième partie de ce travail, nous introduisons un *deuxième niveau de parallélisme* à l'intérieur de chaque sous-domaine. Pour des décompositions de domaines avec recouvrement, le nombre de sous-domaines doit rester faible pour fournir un solveur robuste. De ce fait, les systèmes linéaires associés aux sous-domaines sont résolus de manière efficace avec ce deuxième niveau de parallélisme. Plusieurs tests numériques sont présentés à la fin du document pour valider l'efficacité de cette approche.

KEYWORDS : domain decomposition, preconditioning, multiplicative Schwarz, Parallel GMRES, Newton basis, multilevel parallelism

MOTS-CLÉS : Decomposition de domaine, preconditionnement, Schwarz multiplicatif, GMRES parallèle, Base de Newton, parallélisme multiniveaux.

1. Introduction

In this paper, we are interested in the parallel computation of the solution of the linear system (1)

$$Ax = b \tag{1}$$

with $A \in \mathbb{R}^{n\times n}$, $x, b \in \mathbb{R}^n$. Over the two past decades, the GMRES iterative method proposed by Saad and Schultz [25] has been proved very successful for this type of systems, particularly when A is a large sparse nonsymmetric matrix. Usually, to be robust, the method solves a preconditioned system (2)

$$M^{-1}Ax = M^{-1}b \quad \text{or} \quad AM^{-1}(Mx) = b \tag{2}$$

where M^{-1} is a preconditioner operator that accelerates the convergence of the iterative method.

On computing environments with a distributed architecture, preconditioners based on domain decomposition are of natural use. Their formulation reduces the global problem to several subproblems, where each subproblem is associated to a subdomain; therefore, one or more subdomains are associated to a node of the parallel computer and the global system is solved by exchanging informations between neighboring subdomains. Generally, in domain decomposition methods, there are two ways of deriving the subdomains : (i) from the underlying physical domain and (ii) from the adjacency graph of the coefficient matrix A. In any of these partitioning techniques, subdomains may overlap. Overlapping domain decomposition approaches are known as Schwarz methods while non-overlapping approaches refer to Schur complement techniques. Here, we are interested in preconditioners based on the first class. Depending on how the global solution is obtained, the Schwarz method is *additive* or *multiplicative* [29, Ch. 1]. The former approach computes the solution of subproblems simultaneously in all subdomains. It is akin to the block Jacobi method; therefore, the additive Schwarz method has a straightforward implementation in a parallel environment [8]. Furthermore, it is often used in conjunction with Schur complement techniques to produce hybrid preconditioners [9, 16, 26].

The multiplicative Schwarz method builds a solution of the global system by alternating successively through all the subdomains; it is therefore similar to the block Gauss-Seidel method on an extended system; Thus, compared to the additive approach, it will theoretically require fewer iterations to converge. However, good efficiency is difficult to obtain in a parallel environment due to the high data dependencies between the subdomains. The traditional approach to overcome this is through graph coloring by associating different colors to neighboring subdomains. Hence, the solution in subdomains of the same color could be updated in parallel[29, Ch. 1]. Recently, a different approach has been proposed [2, 4]. In that work, the authors proposed an explicit formulation associated to one iteration of the multiplicative Schwarz method. This formulation requires that the matrix is partitioned in block diagonal form [3] such that each block has a maximum of two neighbors; from this explicit formula, the residual vector is determined out of the computation of the new approximate global solution, and therefore could be parallelized through sequences of matrix-vector products and local solutions in subdomains.

The first purpose of this paper is to present the parallelism that is obtained while using this explicit formulation to build the preconditioned Krylov basis for the GMRES method. This is achieved through the Newton basis implementation proposed in [5] and applied in

[13, 28]. The usual inner products and global synchronizations are avoided across all the subdomains and the resulted algorithm leads to a pipeline parallelism. We will refer to this as a *first-level of parallelism* in GMRES. The second and main purpose of our work here is to further use parallel operations when dealing with subdomains. Generally, for Schwarz-based preconditioners, the number of subdomains are kept small to guarantee the convergence and consequently, the linear systems associated to subdomains can be very large. It is therefore natural to introduce a *second-level of parallelism* when solving those subsystems. This approach is further motivated by the architecture of the current parallel computers made from several interconnected nodes and multi-core processors inside each node. Indeed, these two levels of parallelism use efficiently the compute resources by dividing tasks across and inside all the allocated nodes of the parallel computer. A similar approach has been recently used to enhance scalability of hybrid preconditioners based on the additive Schwarz preconditioner for the Schur complement techniques [15].

The remaining part of this paper is organized as follows. Section 2 recalls the explicit formulation of the Multiplicative Schwarz preconditioner. After that, a parallel implementation of the preconditioned Newton-basis GMRES is given. Section 3 provides the second level of parallelism introduced to solve linear systems in subdomains. As those systems should be solved several times with different right hand sides, the natural way is to use a parallel third-party solver based on LU factorization. In section 4, we provide intensive numerical results that reveal good performance of this parallel hybrid solver. The matrices of tests are taken either from public academic repositories or from industrial test cases. Concluding remarks and future directions of this work are given at the end of the paper.

2. A parallel version of GMRES preconditioned by multiplicative Schwarz

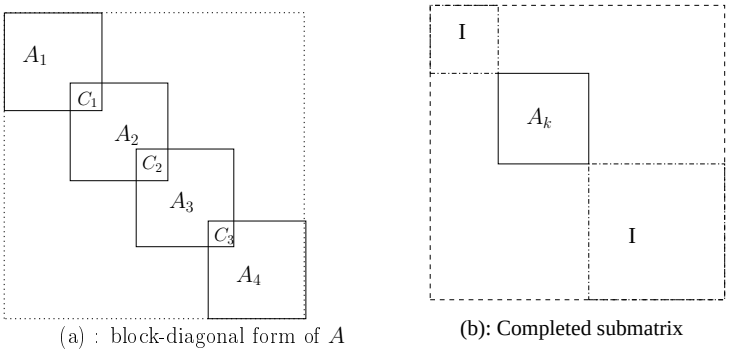
In this section, the explicit formulation of the multiplicative Schwarz method is introduced. Then we show how to use it efficiently as a parallel preconditioner for the GMRES method. A good parallelism is obtained due to the use of the Newton-Krylov basis.

2.1. Explicit formulation of the multiplicative Schwarz preconditioner

From the equation (1), we consider a permutation of the matrix A into p overlapping partitions A_i. We denote by C_i the overlapping matrix between A_i and A_{i+1} $(i = 1, \ldots, p-1)$. Here, each diagonal block has a maximum of two neighbors (see Figure 1.a). Assuming that there is no full line (or column) in the sparse matrix A, such partitioning can be obtained from the adjacency graph of A by means of profile reduction and level sets [3]. We define $\bar{A}_i$ (resp. $\bar{C}_i$) the matrix A_i (resp. C_i) completed by identity to the size of A (see Figure 1.b).

If $\bar{A}_i$ and $\bar{C}_i$ are nonsingular matrices, then the matrix associated to one iteration of the classical multiplicative Schwarz method is defined [4] as :

$$M^{-1} = \bar{A}_p^{-1}\bar{C}_{p-1}\bar{A}_{p-1}^{-1}\bar{C}_{p-2}\ldots\bar{A}_2^{-1}\bar{C}_1\bar{A}_1^{-1}. \quad (3)$$



(a) : block-diagonal form of A (b): Completed submatrix

Figure 1. *Partitioning of A into four subdomains*

This explicit formulation is very useful to provide stand-alone algorithm for the essential operation $y \leftarrow M^{-1}x$ used in iterative methods. However, since dependencies between subdomains are still present in this expression, no efficient parallel algorithm could be obtained for this single operation. Now, if a sequence of vectors v_i should be generated such that $v_i \leftarrow M^{-1}Av_{i-1}$, then a pipeline computation can be setup between all the v_i's and between all the subdomains for each single vector. This is described in Section 2.2.1. For the preconditioned Krylov method, the v_i's are simply the vectors basis of the Krylov subspace. If the classical Arnoldi process is used to generate those basis vectors, then the presence of global communication would destroy the pipelined computation. We therefore rely on the Newton basis implementation described in the next section.

2.2. Background on GMRES with the Newton basis

A left preconditioned restarted GMRES(m) method minimizes the residual vector $r_m = M^{-1}(b - Ax_m)$ in the Krylov subspace $x_0 + \mathcal{K}_m$ where x_0 is the initial approximation and x_m the current iterate. If r_0 is the initial residual vector, then $\mathcal{K}_m$ is defined as

$$span\{r_0, M^{-1}Ar_0, \ldots, (M^{-1}A)^{m-1}r_0\}. \tag{4}$$

The new approximation is of the form $x_m = x_0 + V_m y_m$ where y_m minimizes the euclidian norm $||r_m||_2$. The most time consuming part in this method is the construction of the orthonormal basis V_m of $\mathcal{K}_m$; the Arnoldi process is generally used for this purpose [24]. It constructs the basis and orthogonalizes it in the same time. The effect of this is the presence of global communication between all the processes. Hence, no parallelism could be obtained across the subdomains with this approach. However, synchronisation points can be avoided by decoupling the construction of V_m into two independent phases: first the basis is generated *a priori* then it is orthogonalized.

Many authors proposed different ways to generate this *a priori* basis [5, 31, 11]. With shifts λ_j and scaling factors μ_j $(j = 1, \ldots, n)$, Bai and Reichel [5] define the Newton basis as :

$$\widehat{V}_{m+1} = [\mu_0 r_0, \mu_1(M^{-1}A - \lambda_1 I)r_0, \ldots, \mu_m \prod_{j=1}^{m}(M^{-1}A - \lambda_j I)r_0]. \tag{5}$$

The values λ_j are chosen as approximate eigenvalues of $M^{-1}A$ and ordered with the modified Leja ordering[5] to get a well-conditioned basis. From a chosen initial vector $v_0 = r_0/||r_0||$, a sequence of vectors $v_1, v_2, \ldots, v_m$ is generated as follows: If $\lambda_j \in \mathbb{R}$:

$$v_j = \sigma_j(M^{-1}A - \lambda_j I)v_{j-1} \tag{6}$$

If $\lambda_j \in \mathbb{C}$:

$$v_j = \sigma_j(M^{-1}A - Re(\lambda_j)I)v_{j-1} \tag{7}$$

$$v_{j+1} = \sigma_{j+1}(M^{-1}A - \lambda_j I)(M^{-1}A - \bar{\lambda}_j I)v_{j-1} \tag{8}$$

where

$$\sigma_j = 1/||(M^{-1}A - \lambda_j I)v_{j-1}||. \tag{9}$$

To avoid global communication, the vectors v_j are normalized at the end of the process. Hence, the scalars μ_j from the equation (5) are easily computed as the product of scalars σ_j. These steps are explained in detail in sections 2.2.1 and 2.2.2. At this point, we get a normalized basis V_m such that

$$M^{-1}AV_m = V_{m+1}T_m \tag{10}$$

where T_m is a rectangular matrix formed with the scalars $1/\sigma_j$ and λ_j. V_{m+1} is be orthogonalized using a QR factorization :

$$V_{m+1} = Q_{m+1}R_{m+1}. \tag{11}$$

As we show in section 2.2.1 and following the distribution of vectors in Figure 2.(b), the v_is are distributed in blocks of consecutive rows between all the processors; However, their overlapped regions are not duplicated between neighboring processors. Thus, to perform the QR factorization, we use an algorithm introduced by Sameh [27] with a parallel implementation provided by Sidje [28]. Recently, a new approach called TSQR has been proposed by Demmel et al. [12] which aims to minimize the communications and better use the BLAS kernel operations. Their current algorithm used in [19] is implemented with POSIX threads and is used when a whole matrix V_m is available in the memory of one SMP node.

So far, at the end of the factorization, we get an orthogonal basis Q_{m+1} implicitly represented as a set of orthogonal reflectors. The matrix R_{m+1} is available in the memory of the last processor. To perform the minimization step in GMRES, we derive an Arnoldi-like relation [24, Section 6.3] using equations (10) and (11)

$$M^{-1}AV_m = Q_{m+1}R_{m+1}T_m = Q_{m+1}\bar{G}_m. \tag{12}$$

Hence the matrix $\bar{G}_m$ is in Hessenberg form and the new approximate solution is given by $x_m = x_0 + V_m y_m$ where the vector y minimizes the function J defined by

$$J(y) = ||\beta e_1 - \bar{G}_m y||, \quad \beta = ||r_0|| \quad e_1 = [1, 0, \ldots, 0]^T \in \mathbb{R}^m. \tag{13}$$

The matrix $\bar{G}_m$ is in the memory of the last processor. Since $m << n$, this least-square problem is sequentially and easily solved using a QR factorization of $\bar{G}_m$. More details on the form of T_m and the algorithm to compute $\bar{G}_m$ can be found in [5, 28]. The outline of the GMRES algorithm with the Newton basis is in [13].

2.2.1. Parallel processing of the preconditioned Newton basis

In this section, we generate the Krylov vectors $v_j, (j = 0, \ldots, m)$ of V_{m+1} from the equation (6). Consider the partitioning of the Figure 1 with a total of p subdomains. At this point, we assume that the number of processes is equal to the number of subdomains. Thus, each process computes the sequence of vectors $v_j^{(k)} = (M^{-1}A - \lambda_j I)v_{j-1}^{(k)}$ where $v_j^{(k)}$ is the set of block rows from the vector v_j owned by process P_k. The kernel computation reduces to some extent to two major operations $z = Ax$ and $y = M^{-1}z$. For these operations, we consider the matrices and vector distribution on Figure 2. On each process P_k, the overlapping submatrix with the process P_{k+1} is zeroed to yield a matrix B_k for the matrix-vector multiplication. In Figure 2.(b), the distribution for the vector x is plotted. The overlapping parts are repeated on all processes. Hence, for each subvector $x^{(k)}$ on process P_k, $x^{(k)^u}$ and $x^{(k)^d}$ denote respectively the overlapping parts with $x^{(k-1)}$ and $x^{(k+1)}$. The pseudocode for the matrix-vector product $z = Ax$ follows then in Algorithm 1.

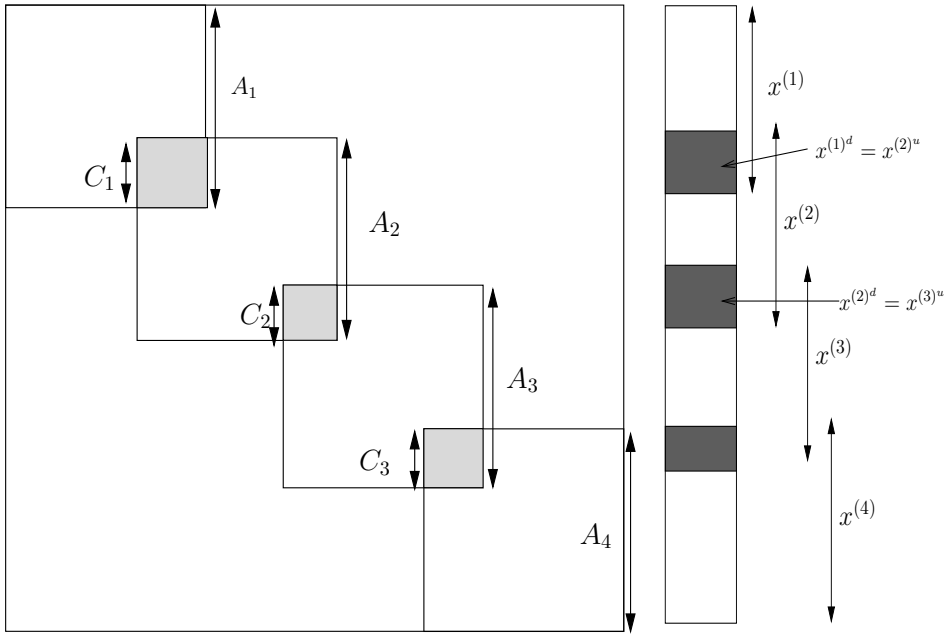


Figure 2. *Distribution of the matrices and vectors for the operation* $y \leftarrow M^{-1}x$

Now we consider the matrix distribution in Figure 2.(a) for the second operation $y = M^{-1}z$. According to relation (3), each process k solves locally the linear system $A_k t^{(k)} = z^{(k)}$ for $t^{(k)}$ followed by a product $y^{(k)^d} = C_k t^{(k)^d}$ with the overlapped matrix C_k. However, the process P_k should receive first the overlapping part of $y^{(k-1)^d}$ from the process P_{k-1}. Algorithm 2 describes the application of the preconditioner M^{-1} to a vector z. The form of the M^{-1} operator produces a data dependency between neighboring processes. Hence the computation of a single vector $v_j = (M^{-1}A - \lambda_j I)v_{j-1}$ is sequential overall the processses. However since the v_j are computed one after another,

Algorithm 1 $z = Ax$

1: /* Process P_k holds B_k, $x^{(k)}$ */
2: $k \leftarrow myrank()$
3: $z^{(k)} \leftarrow B_k x^{(k)}$; /* local matrix-vector product */
4: **if** $k < p$ **then**
5: Send $z^{(k)^d}$ to process P_{k+1}
6: **end if**
7: **if** $k > 1$ **then**
8: Receive $z^{(k-1)^d}$ from process P_{k-1}
9: $z^{(k)^u} \leftarrow z^{(k)^u} + z^{(k-1)^d}$
10: **end if**
11: /* Communication for the consistency of overlapped regions */
12: **if** $k > 1$ **then**
13: Send $z^{(k)^u}$ to process P_{k-1}
14: Receive $z^{(k+1)^u}$ from process P_{k+1}
15: $z^{(k)^d} \leftarrow z^{(k+1)^u}$
16: **end if**
17: **return** $z^{(k)}$

a process can start to compute its own part of the vector v_j even if the whole previous vector v_{j-1} is not available. We refer to this as a *pipeline parallelism* since the vectors v_j are computed across all the processes as in a pipeline. This would not be possible with the Arnoldi process as all the global communications introduce synchronizations points between the processes and prevent the use of the pipeline; see for instance the data dependencies implied by this process in Erhel [13].

Algorithm 2 $y = M^{-1}z$

1: /* Process P_k holds A_k, $z^{(k)}$ */
2: $k \leftarrow myrank()$
3: **if** $k > 1$ **then**
4: Receive $y^{(k-1)^d}$ from process P_{k-1}
5: $z^{(k)^u} \leftarrow y^{(k-1)^d}$
6: **end if**
7: Solve local system $A_k y^{(k)} = z^{(k)}$ for $y^{(k)}$
8: **if** $k < p$ **then**
9: $y^{(k)^d} = C_k y^{(k)^d}$
10: Send $y^{(k)^d}$ to process P_{k+1}
11: **end if**
12: /* Communication for the consistency of overlapped regions */
13: **if** $k > 1$ **then**
14: Send $y^{(k)^u}$ to process P_{k-1}
15: **end if**
16: **if** $k < p$ **then**
17: Receive $y^{(k+1)^u}$ from process P_{k+1}
18: $y^{(k)^d} = y^{(k+1)^u}$
19: **end if**
20: **return** $y^{(k)}$

To better understand the actual *pipeline parallelism*, we recall here all the dependencies presented in [30]. For the parallel matrix-vector product in Algorithm 1, if we set $z = h(x) = Ax$, then $z^{(k)} = h_k(x^{(k-1)}, x^{(k)}, x^{(k+1)})$ and the Figure 3.(a) illustrates these dependencies. For the preconditioner application $y \leftarrow M^{-1}z$ as written in Algorithm 2, we set $y = g(z) = M^{-1}z$, then $y^{(k)} = g_k(y^{(k-1)}, z^{(k)}, z^{(k+1)})$ and dependencies are depicted on Figure 3.(b). Finally, for the operation $y \leftarrow M^{-1}Ax$, if $y = f(x) = AM^{-1}x = h \circ g(x)$, then $y^{(k)} = f_k(y^{(k-1)}, x^{(k)}, x^{(k+1)}, x^{(k+2)})$ and we combine the two graphs to have the dependencies on Figure 4.a. Hence the dependency $x^{(k-1)}$ is combined with that of $y^{(k-1)}$. In the pipeline flow computation $v_j = f(v_{j-1})$, the dependency $x^{(k+2)}$ in Figure 4.b delays the computation of the $v_j^{(k)}$ until the subvector $v_{j-1}^{(k+2)}$ is available. If there is a good load balancing between all the subdomains and if τ denotes a time to compute a subvector $x^{(k)}$ including the time of all the required MPI communications, then the time to compute one vector is

$$t(1) = p\tau \tag{14}$$

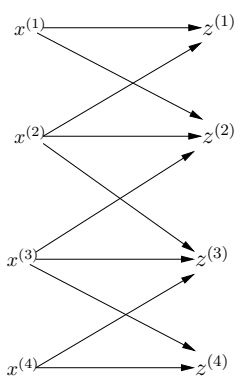
and the time to compute m vectors of the basis follows

$$t(m) = p\tau + 3(m-1)\tau. \tag{15}$$

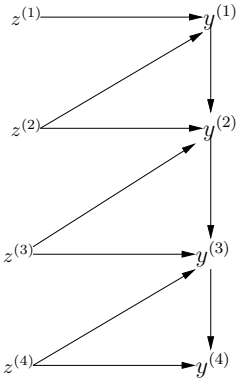
Finally, the first vector is available after $p\tau$ and then, a new vector is produced every 3τ (see Figure 5). The efficiency e_p of the overall algorithm is therefore computed as :

$$e_p = \frac{mt(1)}{pt(m)} = \frac{p\tau m}{p(p\tau + 3(m-1)\tau)} = \frac{m}{p + 3(m-1)}. \tag{16}$$

Hence, the efficiency grows with the value of m but is limited by the Amdhal law to $1/3$ when m tends to the infinity.



(a): Graph for the matrix-vector



(b): Graph for applying M^{-1}

Figure 3. *Dependency graphs*

2.2.2. Computation of shifts and scaling factors

So far, we have not yet explained how to compute the shifts λ_i and the scaling factors σ_i of the equation (6) and (9).

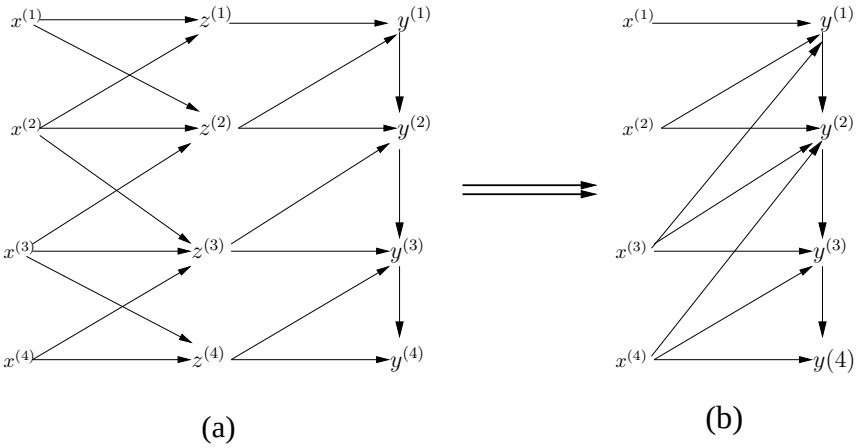


Figure 4. *Dependency graph for* $y = M^{-1}Ax$

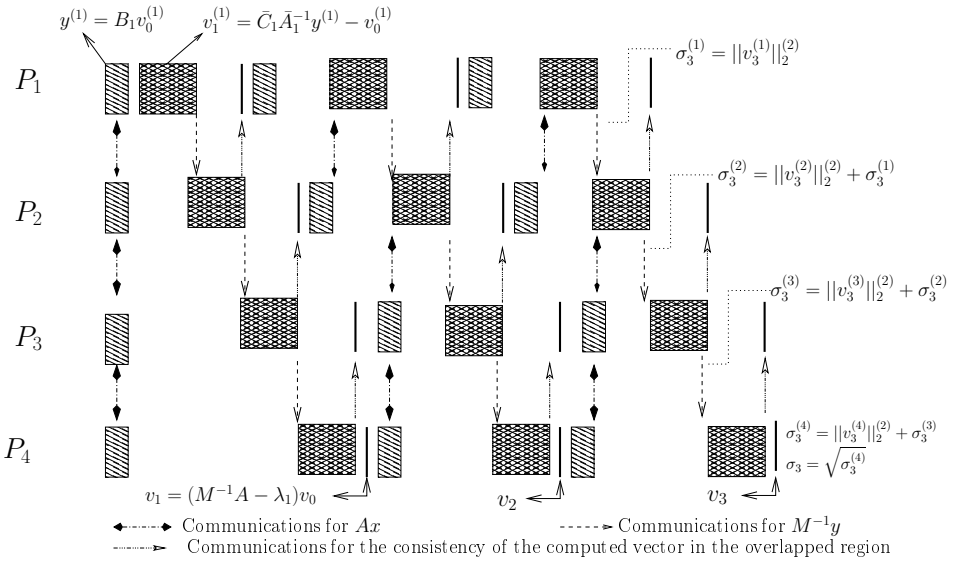


Figure 5. *Double recursion during the computation of the Krylov basis*

In order to get a well-conditioned Krylov basis, Reichel [23] and Bai and Reichel [5] suggest the use of approximate eigenvalues of $M^{-1}A$ as the shifts λ_j in the Newton basis polynomials. After one cycle of the classical GMRES(m) with the Arnoldi process, these shifts are obtained cheaply by computing the m eigenvalues of the leading $m \times m$ principal submatrix of the output Hessenberg matrix. These values, known as the Ritz values of $M^{-1}A$, are sorted using the modified Leja ordering [23] grouping together the complex conjugate pairs. Recently, Philippe and Reichel [22] proved that, in some cases, roots of the Chebychev polynomials can also be used efficiently as shifts for this Newton basis.

The scalars σ_i used to normalize the vectors of $\widehat{V}_{m+1}$ are determined without using global reduction operations. Indeed, such operations introduce synchronisation points between all the processes and consequently destroy the pipeline parallelism. The Algo-

rithms 1 and 2 compute in some extent the sequence $v_j = (M^{-1}A - \lambda_i I)v_{j-1}$. We are interested in the scalars $\sigma_j = ||v_j||_2$. For this purpose, on process P_k, we define by $\hat{v}_j^{(k)}$ the subvector $v_j^{(k)}$ without the overlapping part. In the Algorithm 2, after the line 8, an instruction is therefore added to compute the local sum $\sigma_j^{(k)} = ||\hat{v}_j^{(k)}||_2^2$. The result is sent to the process P_{k+1} at the same time as the overlapping subvector at line 10. The next process P_{k+1} receives the result and adds it to its own contribution. This is repeated until the last process P_p. At the end, the scalar $\sqrt{\sigma_j^{(p)}}$ gives the 2-norm of v_j. An illustration is given in the Figure 5 during the computation of the vector v_3.

3. Enhancing the parallelism in subdomains.

In this section, we propose two levels of parallelism to enhance the robustness and the efficiency of the method.

3.1. Motivations for two levels of parallelism

Preconditioners based on domain decomposition do no scale very well, particularly when the coefficient matrix of the linear system is nonsymmetric or symmetric indefinite or when the underlying PDE is far from elliptic. In those cases generally, the number of iterations of the preconditioned GMRES increases very fast with the number of subdomains and consequently the total time to converge is also increased. It becomes essential to keep constant and small the global number of subdomains in order to provide a robust iterative method.

In the particular case of the multiplicative Schwarz preconditioner, the startup time $p\tau$ of the pipeline parallelism introduced in section 2.2.1 and the identity 16 shows that the efficiency is limited by p, the number of subdomains. Thus the method should be more efficient if the number of vectors m to compute is large enough to annihilate this startup time. In other words, p should not be very large compared to m. Figure 6 gives a theoretical efficiency of the method with different values of p and m. It can be seen that the more the subdomains are, the more m should be large in order to get a significant efficiency; usually, no assumption should be made on the value of m as it depends on the problem difficulty. It is therefore essential to act more on p in order to enhance the parallel efficiency of the method.

If one subdomain is assigned to only one processor on modern supercomputers, then using a small number of subdomains as suggested by the above discussion could be a limitation of the method. If there are more processors than subdomains then a logical approach is to assign one subdomain to several processors and then to define several groups of processors, one for each subdomain. This distribution of data follow naturally the architecture of present parallel computers made with several interconnected nodes and with several processors inside one node. Hence all processors in one node deal with data inside the memory of this node. The next section shows how this second level of parallelism is used within the subdomains.

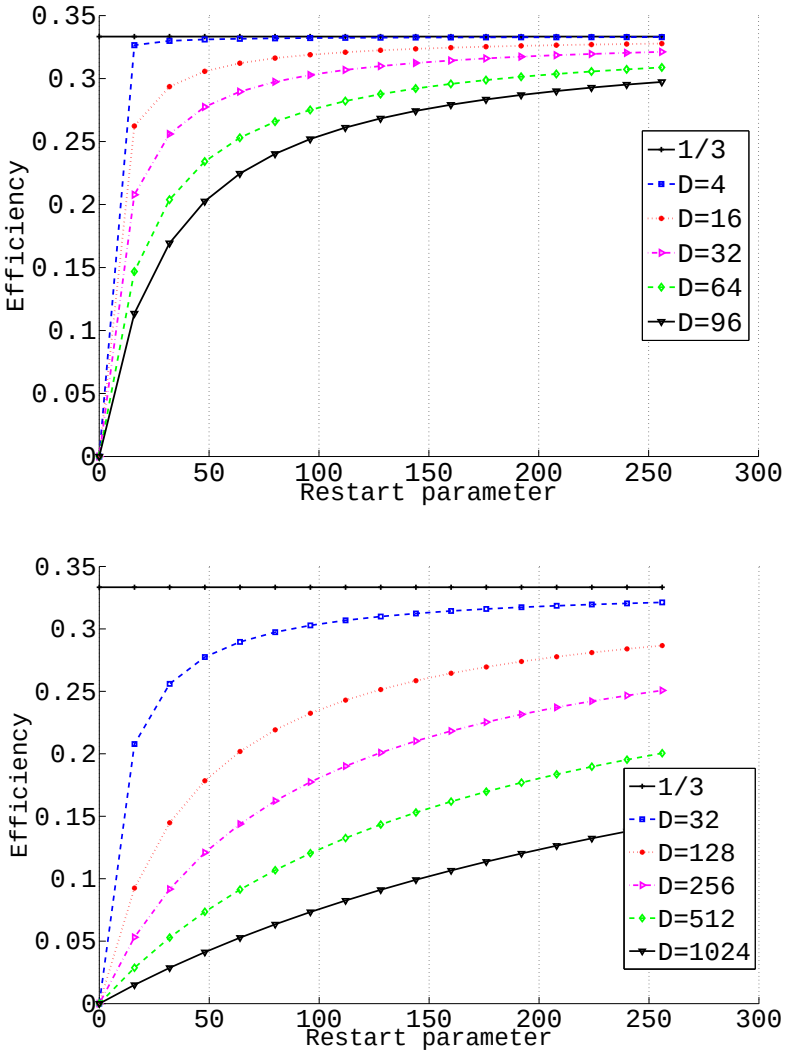


Figure 6. *Theoretical efficiency of GPREMS 1*

3.2. Practical implementation

Tasks to parallelize inside the subdomain are mainly the solution of linear systems induced by the application of the preconditioner M^{-1}. As those systems should be solved several times with the same coefficient matrix, it is natural to use a direct method. This approach is usually known as hybrid direct/iterative technique. The parallelism inside the subdomains relies on the message passing paradigm instead of a shared-memory model. The motivation for this choice is that many high-performant and public domain solvers are based on MPI [1, 17, 18]. Note that with the message passing model, a subdomain can be distributed on more than one node if the physical memory is small on that node.

So far with the two levels of data distribution on compute units, the solver goes through

all the following steps

1) ***Initialization :*** During the first step, the global matrix A is permuted in block diagonal form by the host process (see Figure 1). After that, the local matrices A_k and C_k should be distributed to other processes. If a distributed solver is used in subdomains, then the processors are dispatched in multiple communicators. Figure 7 shows a distribution of four nodes with four processes each around two levels of MPI communicators. The first level is intended for the communication across the subdomains. Hence in this communicator, the submatrices are distributed to the $level\ 0$ processes (i.e $P_{k,0}$ where $k = 0 \dots p-1$ and p the number of subdomains). For each subdomain k, a second communicator is created between the $level\ 1$ processes to manage communications inside the subdomains (i.e $P_{k,j}$ where $j = 0 \dots p_k - 1$ and p_k the number of processes in the subdomain k).

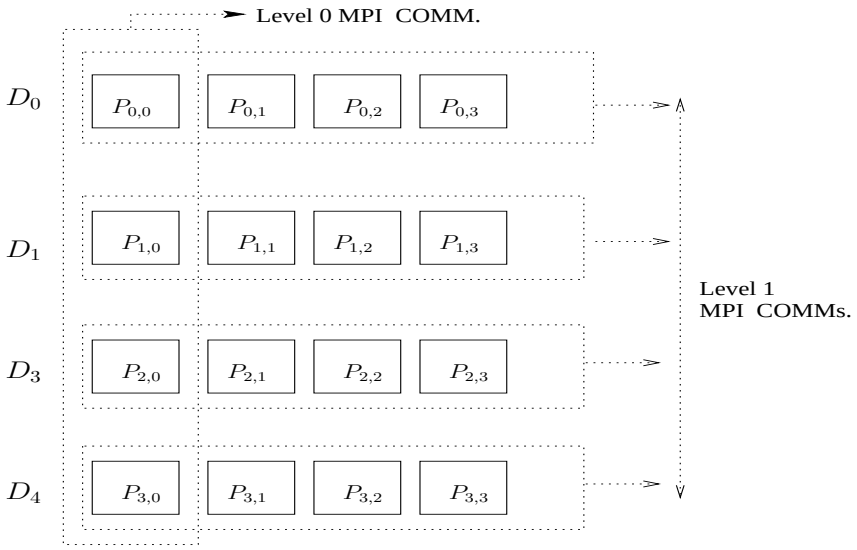


Figure 7. *MPI communicators for the two levels of parallelism*

2) ***Setup :*** In this phase, the symbolic and numerical factorization are performed on submatrices A_k by the underlying local solver. This step is purely parallel across all the subdomains. At the end of this phase, the factors L_k and U_k reside in the processors responsible for the subdomain k. This is totally managed by the local solver. Prior to this phase, a preprocessing step can be performed to scale the elements of the matrix.

3) ***Solve :*** This is the iterative phase of the solver. With an initial guess x_0, the solver computes all the equations (6-13) as outlined here:

a) Perform one cycle of GMRES with the Arnoldi process to compute the shifts λ_j.

b) Pipeline computation of V_{m+1}

c) Parallel QR factorization $V_{m+1} = Q_{m+1}R_{m+1}$

d) Sequential computation of $\bar{G}_m$ such that $M^{-1}AV_m = Q_{m+1}\bar{G}_m$ on the process $P_{p-1,0}$.

e) Sequential solution of least-square problem (13) for y_m on the process $P_{p-1,0}$.

f) Broadcast the vector y_m to processes $P_{k,0}$

g) Paralle computation of $x_m = x_0 + V_m y_m$

The convergence is reached when $||b - Ax|| < \epsilon||b||$ otherwise the process restarts with $x_0 = x_m$. When two levels of parallelism are used during the computation of V_m, $level$ 1 processors perform multiple backward and forward sweeps in parallel to solve local systems. After the parallel QR factorization in step 3c, the explicit Q factor is never formed explicitly. Indeed, only the unfactored basis V is used to apply the new correction as shown in step 3g. Nevertheless, a routine to form this factor is provided in the solver with the courtesy of Sidje [28].

4. Numerical experiments

In this section, we perform several experiments to give the numerical robustness of GPREMS and the benefits of using two levels of data distribution. We start by giving the software architecture of the solver in subsection 4.1 and the test cases in subsection 4.2.

4.1. Software and hardware framework

The solver is named GPREMS[1] (GMRES PRECONDITIONED BY MULTIPICATIVE SCHWARZ). It is intended to be deployed on distributed memory computers that communicate through message passing (MPI). The parallelism in subdomains is based either on message passing or threads model depending on the underlying solver. The whole library is built on top of PETSc (Portable, Extensible Toolkit for Scientific Computation) [6, 7]. The motivation of this choice is the uniform access to a wide range of external packages for the linear systems in subdomain. Moreover, the PETSc package provides several optimized functions and data structures to manipulate the distributed matrices and vectors. Figure 8 gives the position of GPREMS in the abstraction layer of PETSc. We

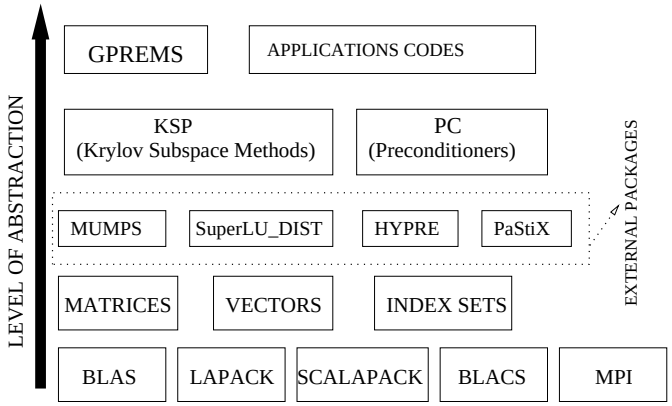


Figure 8. *GPREMS library in PETSc*

give only the components that are used by GPREMS. For a complete reference on PETSc

1. A public license will be released soon

environment, please refer to [6]. Although GPREMS uses the PETSc environment, it is not distributed as a part of PETSc libraries. Nevertheless, the library is configured easily once a usable version of PETSc is available on the targeted architecture.

All the tests in this paper are performed on the IBM p575 SMP nodes connected through the Infiniband DDR network. Each node is composed of 32 Power6 processors sharing the same global memory. A Power6 processor is a dual-core 2-way SMT with a peak frequency at 4.7 GHz. A total of 128 nodes is available in this supercomputer named *Vargas*[2] which is part of the French CNRS-IDRIS supercomputing facility.

4.2. Test matrices

Table 1. *General properties of the four test matrices*

Matrix	Size	Entries	Source
PARA_04	153,226	2,930,882	UFL
CASE_04	7,980	2,930,882	FLUOREM
CASE_07	233,786	11,762,405	FLUOREM
CASE_09	277,095	30,000,952	FLUOREM
CASE_17	381,689	37,464,962	FLUOREM

In Table 1, the main characteristics of the test cases are listed. The first matrix PARA-4 arises from 2D semiconductor device simulations and is taken from the University of Florida (UFL) Sparse Matrix Collection [10]. The remaining test cases are provided by FLUOREM [14], a software editor in fluid dynamics simulations. They correspond to linearized Navier-Stokes equations.

4.3. Numerical robustness of GPREMS

The main motivation of using the multiplicative Schwarz method is its robustness compared to the additive Schwarz implementation. There are some cases where the latter fails. In this case, the former can be a good alternative. We illustrate such situation on FLUOREM problem CASE_04. It is worth to note that previous work has been done to test similar problems on other hybrid solvers based on Schur complement techniques. Here we compare the multiplicative Schwarz in GPREMS with the restricted additive Schwarz (ASM) method used as a preconditioner for GMRES. The implementation provided in PETSc release 3.0.0-p2. In the latter, the Krylov basis is built with the modified Gram-Schmidt (MGS) method and the input matrix is partitioned in 4 subdomains using ParMETIS. With ASM, we note in Figure 9 that GMRES(16) stagnates from the first restart (the plain line). It is necessary to form 32 Krylov vectors at each restart in order to achieve a fair accuracy (dash curve). For GMRES(m), the size m of the Krylov subspace is critical. Generally, it is a trial and error process to get a good value of m. When GPREMS is used, a stagnation is less likely to occur. In this test case, a good convergence is obtained in a few number of iterations with the two values of m. For instance, the relative residual norm drops to 10^{-8} in less than 100 iterations (dot and dash-dot lines).

2. http://www.idris.fr/su/Scalaire/vargas/hw-vargas.html

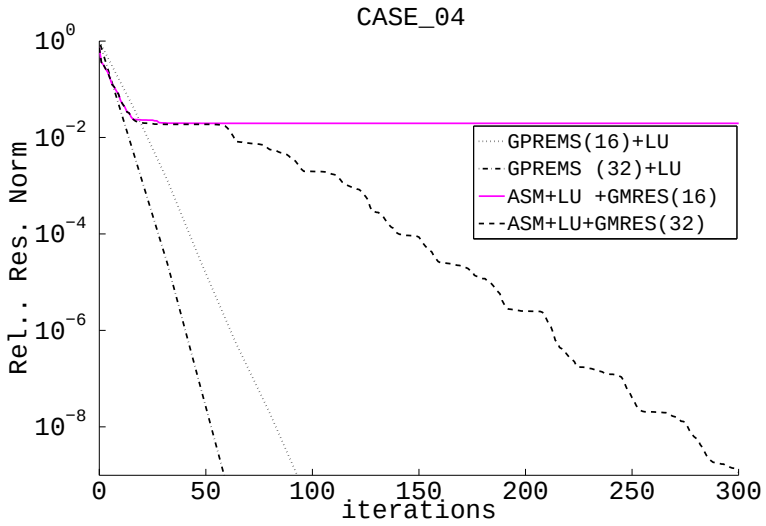


Figure 9. *The multiplicative Schwarz approach (GPREMS) compared to the restricted additive Schwarz (ASM) on CASE_004; 16 and 32 vectors in the Krylov subspace at each restart; 4 subdomains (block diagonal partitioning in GPREMS and Parmetis for ASM); LU factorization on local matrices with MUMPS package*

4.4. Benefits of two levels of parallelism

In this part, we give the gain of using two levels of data distribution. We first consider the problem CASE_017 listed in Table 1. The geometry of this 3D case is a jet engine compressor. This test case is difficult to solve as shown in a short comparative study involving some distributed linear solvers [21]. From this study, solvers based on overlapping Schwarz decomposition provide an efficient way to deal with such problems. However, the number of subdomains should be kept small to provide a good convergence. In Table 2, We keep 4 and 8 subdomains and we increase the total number of processors. As a result, almost all the steps in GPREMS get a noticeable speedup. Typically, with 4 subdomains, when only one processor is active, the time to setup the block matrices is almost 203 s. and the time spent in the iterative loop (under Time/Iter) is 622 s. Moving to 8 active processors decreases these times to 59 s. and 181 s. respectively. The same observation can be done when using 8 subdomains, in which case the overall time drops from 540 s. to 238 s. Note that the overall time includes the first sequential step that permutes the matrix in block-diagonal form and distributes the block matrices to all active processors. *Intranode speedup and efficiency* are reported in the last two columns of Table 2 . This is different from the ones computed with only one level of parallelism. The main objective here is to evaluate if it is worthy to add more processors in a subdomain. If d is the number of subdomains, $s_p = T_d/T_p$ and $e_p = s_p/p$ where T_p is the CPU time on p processors. For 4 subdomains, using 8 processors in each subdomain gives a speedup of 2.88. This speedup is 2.28 when 8 subdomains and 64 processors are used. The major speedup is observed for the setup phase thanks to the direct solver MUMPS [1]. The solve phase shows a noticeable speedup as well. The overall efficiency is decreasing very fast due to the fact that all the processors in one subdomain are scheduled in one SMP node. During the computations, all the processors share the same memory bandwith and access irregularly data in the global memory. With the low granularity and the irregular access of

data in sparse matrix computations, the efficiency is determined mostly by the size of the memory bandwith rather than the clock rate of processors. Nevertheless, the two levels of parallelism help to keep busy the processors in the SMP node as they would be idle otherwise.

Table 2. *Benefits of the two-levels of parallelism for various phases of GPREMS on CASE_17 with a restart of 64 and MUMPS direct solver in subdomains*

#D	#Proc.	Iter.	CPU Time (s.)						
			Init	Setup	Solve	Time/Iter	Total	s_p	e_p
4	4	128	70.78	203.67	622.94	4.87	897.39	-	-
	8	128	70.77	125.77	411.42	3.21	607.96	1.48	0.74
	16	128	69.79	73.15	280.41	2.19	423.35	2.12	0.53
	32	128	70.77	59.44	181.45	1.42	311.66	2.88	0.36
8	8	256	69.91	71.73	399.34	1.56	540.98	-	-
	16	256	70.36	42.99	343.25	1.34	456.59	1.18	0.59
	32	256	71.61	28.72	206.7	0.81	307.02	1.76	0.44
	64	256	71.85	20.85	144.79	0.57	237.49	2.28	0.28

We further point out the benefits of adding several processors in each subdomain for the problems PARA_4, CASE_07 and CASE_09. The CPU times for the main phases in the solver are reported in Table 3 and 4: the setup time, the Iterative loop time (under Solve), the mean time spent in each iteration (under Time/Iter). The total time includes the time for the preprocessing step which is not reported. The size of the Krylov basis is 32 for PARA_4, and 40 for CASE_07 and CASE_09. MUMPS[1] is used as direct solver in each subdomain. The iterative process stops when the relative residual norm $||b-Ax||/||b||$ is less than 10^{-8}. The statistics show a good improvement with two levels

Table 3. *CPU Time of GPREMS on PARA_4*

#Proc.	#D	PARA_4			
		Setup	Solve	Time/Iter	Total
4	4	11.75	58.31	0.40	72.84
8	4	8.28	58.68	0.41	69.74
	8	3.79	46.40	0.24	52.97
16	4	5.79	40.55	0.28	49.11
	8	2.86	31.64	0.16	37.26
	16	1.12	41.85	0.17	46.09
32	4	4.57	33.80	0.23	41.15
	8	1.78	20.94	0.11	25.47
	16	0.86	31.10	0.13	35.09
64	8	1.51	17.97	0.09	22.25

of parallelism. As noted before, the major improvement is in the setup phase but the iterative phase benefits from this approach as well. It is worth to note the time spent for each singe iteration. Indeed, this time decrease, although not fast, as more processors are added in subdomains.

Table 4. *Setup and solve phase of GPREMS on CASE_07 and CASE_09*

#Proc.	#D	CASE_07			CASE_09		
		Setup	Solve	Time/Iter	Setup	Solve	Time/Iter
4	4	19.05	77.38	0.64	57.00	121.85	1.52
16	4	10.27	47.86	0.40	24.94	60.18	0.75
48	12	1.95	28.49	0.09	4.72	51.99	0.22
96	12	1.67	21.75	0.07	3.45	38.08	0.16

5. Concluding remarks

In this paper, we give an implementation of a parallel solver for the solution of linear systems on distributed-memory computers. This implementation is based on an hybrid technique that combines a multiplicative Schwarz preconditioner with a GMRES accelerator. It is known that the multiplicative formulation of the Schwarz method does not have a natural parallelism. Thanks to the Newton basis GMRES implementation, a good pipeline parallelism can be obtained through the subdomains. It is also admitted that Schwarz-based preconditioners do not scale very well with the number of subdomains. In this work, we implement two levels of data distribution to limit the number of subdomains. To this end, we define two levels of parallelism during the computation of the orthonormal basis needed by GMRES : The first level is expressed through pipeline operations across all the subdomains. The second level uses a parallelism inside third-party solvers to build the solution of subsystems induced by the domain decomposition. It is obvious that even with these two levels of parallelism, the proposed approach can not compete with multilevel methods based on Schur complement techniques. Nevertheless, there are some problems where Schwarz preconditioners offer more robustness.

The experimental tests have pointed out this robustness on tests cases arising from linearized Navier Stokes equations. This robustness is enhanced with a direct solver for linear systems in subdomains. We have shown that the gain of using a parallel solver inside the subdomains is two-fold: the convergence is guaranteed when the number of processors grows as the number of subdomains remains the same. The global efficiency increases as we add more processors in subdomains. For large block matrices, the use of a direct solver in subdomains implies to access the whole memory of a SMP node. Therefore, this approach keeps busy all the processors of the SMP node and enables a good usage of allocated computing resources.

However, more work needs to be done to achieve very good scalability on massively parallel computers. Presently, an attempt to increase the number of subdomains up to 32 increases the number of iterations as well. So we are investigating ways to maintain the latter as small as possible. A first attempt is to use more than two levels of splitting but the efficiency of such approach is not always guaranteed, specifically if the linear system arises from non-elliptic partial differential equations. An ongoing work is to keep the two-levels of splitting for the preconditioner operator and then to further accelerate the GMRES method with spectral informations gathered during the iterative process.

Acknowledgments

The authors wish to thank Jocelyne ERHEL and Bernard PHILIPPE from INRIA-Rennes for helpful discussions on this work. This work was performed using HPC re-

sources from GENCI-IDRIS (Grand Equipement National de Calcul Intensif - Institut du Développement et des Ressources en Informatique Scientifique). Early experiments were carried out using the Grid'5000 experimental testbed (https://www.grid5000.fr).

A short version of this paper appeared in the proceedings of the CARI conference, see [20].

6. References

[1] Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[2] Guy-Antoine Atenekeng-Kahou. *Parallélisation de GMRES préconditionné par une itération de Schwarz multiplicatif*. PhD thesis, University of Rennes 1 and University of Yaounde 1, 2008. ftp://ftp.irisa.fr/techreports/theses/2008/atenekeng.pdf.

[3] Guy-Antoine Atenekeng-Kahou, Laura Grigori, and Masha Sosonkina. A partitioning algorithm for block-diagonal matrices with overlap. *Parallel Computing*, 34(6-8):332–344, 2008.

[4] Guy-Antoine Atenekeng-Kahou, Emmanuel Kamgnia, and Bernard Philippe. An explicit formulation of the multiplicative Schwarz preconditioner. *Applied Numerical Mathematics*, 57(11-12):1197 – 1213, 2007.

[5] Z. Bai, D. Hu, and L. Reichel. A Newton basis GMRES implementation. *IMA J Numer Anal*, 14(4):563–581, 1994.

[6] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.0.0, Argonne National Laboratory, 2008.

[7] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2009. http://www.mcs.anl.gov/petsc.

[8] Xiao-Chuan Cai and Yousef Saad. Overlapping domain decomposition algorithms for general sparse matrices. *Numerical Linear Algebra with Applications*, 3(3):221–237, 1996.

[9] L.M. Carvalho, L. Giraud, and G. Meurant. Local preconditioners for two-level non-overlapping domain decomposition methods. *Numerical Linear Algebra with Applications*, 8(4):207–227, 2001.

[10] Tim Davis. The university of Florida sparse matrix collection. *ACM Trans. Math. Software(to appear)*, 2010.

[11] E. de Sturler. A parallel variant of GMRES(m). In J. J. H. Miller and R. Vichnevetsky, editors, *Proceedings of the 13th IMACS World Congress on Computation and Applied Mathematics, Dublin, Ireland*. Criterion Press, 1991.

[12] James Demmel, Laura Grigori, M. Hoemmen, and Julien Langou. Communication-optimal parallel and sequential QR and LU factorizations. Technical Report UCB/EECS-2008-89, University of California EECS, May 2008.

[13] Jocelyne Erhel. A parallel GMRES version for general sparse matrices. *Electronic Transaction on Numerical Analysis*, 3:160–176, 1995.

[14] FLUOREM. The FLUOREM matrix collection, 2009. LIB0721 2.0 / FP-SA http://www.fluorem.com.

[15] L. Giraud, A. Haidar, and S. Pralet. Using multiple levels of parallelism to enhance the performance of domain decomposition solvers. *Parallel Computing*, In Press, 2010.

[16] L. Giraud, A. Haidar, and L. T. Watson. Parallel scalability study of hybrid preconditioners in three dimensions. *Parallel Comput.*, 34(6-8):363–379, 2008.

[17] Pascal Hénon, Francois Pellegrini, Pierre Ramet, Jean Roman, and Yousef Saad. High Performance Complete and Incomplete Factorizations for Very Large Sparse Systems by using Scotch and PaStiX softwares. In *Eleventh SIAM Conference on Parallel Processing for Scientific Computing*, 2004.

[18] Xiaoye Sherry Li and James W. Demmel. SuperLU DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29:110–140, 2003.

[19] Marghoob Mohiyuddin, Mark Hoemmen, James Demmel, and Katherine Yelick. Minimizing communication in sparse matrix solvers. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM.

[20] Désiré Nuentsa Wakam, Jocelyne Erhel, and Édouard Canot. Parallélisme à deux niveaux dans GMRES avec un préconditionneur Schwarz multiplicatif. In É. Badouel, A. Sbihi, and I. Lopko, editors, *CARI 2010 Actes du 10ème Colloque Africain sur la Recherche en Informatique et Mathématiques Appliquées*, pages 189–196, Yamoussoukro, Côte D'Ivoire, 10 2010. INRIA.

[21] Désiré Nuentsa Wakam, Jocelyne Erhel, Édouard Canot, and Guy-Antoine Atenekeng Kahou. A comparative study of some distributed linear solvers on systems arising from fluid dynamics simulations. In *Parallel Computing: From Multicores and GPU's to Petascale*, volume 19 of *Advances in Parallel Computing*, pages 51–58. IOS Press, 2010.

[22] Bernard Philippe and Lothar Reichel. On the generation of Krylov subspace bases. *Applied Numerical Mathematics*, In Press, Corrected Proof:–, 2011.

[23] Lothar Reichel. Newton interpolation at Leja points. *BIT Numerical Mathematics*, 30:332–346, 1990. 10.1007/BF02017352.

[24] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2000.

[25] Yousef Saad and Martin H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869, 1986.

[26] Yousef Saad and Masha Sosonkina. Distributed Schur complement techniques for general sparse linear systems. *SIAM Journal on Scientific Computing*, 21(4):1337–1356, 1999.

[27] A.H. Sameh. Solving the linear leastsquares problem on a linear array of processors. In D. Lawrie D. Kuck and A. Sameh, editors, *High Speed Computer and Algorithm Organization*, pages 207–228. Academic Press, 1977.

[28] Roger B. Sidje. Alternatives for parallel Krylov subspace basis computation. *Numerical Linear Algebra with Applications*, 4(4):305–331, 1997.

[29] Barry Smith, Petter Bjϕrstad, and William Gropp. *Domain Decomposition, Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

[30] Innocent Souopgui. Parallélisation d'une double recurrence dans GMRES. Master's thesis, University of Yaounde 1, 2005.

[31] Homer F. Walker. Implementation of the GMRES method using Householder transformations. *SIAM Journal on Scientific and Statistical Computing*, 9(1):152–163, 1988.