Parallel algebraic domain decomposition solver for the solution of augmented systems

Emmanuel Agullo, Luc Giraud, Abdou Guermouche, Azzam Haidar, Jean Roman

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Parallel algebraic domain decomposition solver for the solution of augmented systems

E. Agullo — L. Giraud — A. Guermouche — A. Haidar — J. Roman

N° 7516

January 2011

Distributed and High Performance Computing

ISSN 0249-6399 ISRN INRIA/RR--7516--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
BORDEAUX - SUD OUEST

Parallel algebraic domain decomposition solver for the solution of augmented systems

E. Agullo* , L. Giraud† , A. Guermouche‡ , A. Haidar§ , J. Roman¶

Theme : Distributed and High Performance Computing
Networks, Systems and Services, Distributed Computing
Équipe-Projet HiePACS

Rapport de recherche n° 7516 — January 2011 — 18 pages

Abstract: We consider the parallel iterative solution of indefinite linear systems given as augmented systems where the $(1,1)$ block is symmetric positive definite and the $(2,2)$ block is zero. Our numerical technique is based on an algebraic non overlapping domain decomposition technique that only exploits the graph of the sparse matrix. This approach to high-performance, scalable solution of large sparse linear systems in parallel scientific computing is to combine direct and iterative methods. Such a hybrid approach exploits the advantages of both direct and iterative methods. The iterative component allows us to use a small amount of memory and provides a natural way for parallelization. The direct part provides favorable numerical properties. The graph of the sparse matrix is split into sub-graphs and a condensed linear system, namely the Schur complement system, is solved iteratively for the unknowns associated with the interface between the sub-graphs; a sparse direct solver is used for the variables associated with the internal parts of the sub-graphs. For augmented systems a special attention should be paid to ensure the non singularity of the local sub-problems so that the Schur complement is defined. For augmented systems, where the $(2,2)$ block is small compared to the $(1,1)$ block, we design a special technique that enforces the Lagrange multiplier variables (associated with the $(2,2)$ block) to be in the interface unknowns. This technique has two main advantages. First, it ensures that none of the local sub-systems is structurally singular and for symmetric positive definite $(1,1)$ block, it even ensures that those sub-matrices are also symmetric positive definite. This latter property enables us to use a Cholesky factorization for the internal sub-problems which reduces the computational complexity (in term of floating point operation counts and memory consumption) compared to a more general LU decomposition. In this paper, we describe how the graph partitioning problem is formulated to comply with the above mentioned constraints. We report numerical and parallel performance of the scheme on large matrices arsing from the finite element discretization of linear elasticity in structural mechanic problems. For those problems some boundary conditions are modeled through the use of Lagrange multipliers.

Key-words: Hybrid direct/iterative solver, augmented systems, parallel scientific computing, scalable preconditioner, large 3D problems, high performance computing.

* INRIA Bordeaux - Sud-Ouest - HiePACS joint INRIA-CERFACS Lab. Emmanuel.Agullo@inria.fr
† INRIA Bordeaux - Sud-Ouest - HiePACS joint INRIA-CERFACS Lab. luc.giraud@inria.fr
‡ LaBRI, INRIA Bordeaux - Sud-Ouest - HiePACS joint INRIA-CERFACS Lab. abdou.guermouche@labri.fr
§ Innovative Computing Laboratory, Computer Science Department, University of Tennessee, USA. haidar@utk.edu
¶ IPB, LaBRI, INRIA Bordeaux - Sud-Ouest - HiePACS joint INRIA-CERFACS Lab. Jean.Roman@inria.fr

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération 33405 Talence Cedex
Téléphone : +33 5 40 00 69 00

Solution parallèle par décomposition de domaines algébrique pour des systèmes augmentés

Résumé : On considère la résolution itérative parallèle de systèmes indéfinis donnés sous la forme de systèmes augmentés où le bloc $(1,1)$ est symétrique défini positif et le bloc $(2,2)$ est nul. Notre méthode numérique est basée sur une technique algébrique de décomposition de domaine sans recouvrement qui exploite uniquement le graphe d'adjacence de la matrice creuse. Cette approche parallèle, qui passe à l'échelle pour la résolution de systèmes linéaires, combine des techniques itératives et directes. Une telle approche hybride tire partie des avantages des approches itératives et directes. La composante itérative permet l'utilisation d'un espace mémoire modéré et permet une parallélisation naturelle. La partie directe induit de la robustesse. Le graphe de la matrice creuse est décomposée en sous-graphes et un système réduit, le complément de Schur, est résolu itérativement pour les inconnues associées aux interfaces entre les sous-graphes; un solveur direct creux est utilisé pour les variables internes aux sous-graphes. Pour la résolution des systèmes augmentés une attention supplémentaire doit être apportée pour éviter toute singularité structurelle des sous-problèmes locaux qui définissent le complément de Schur. Pour des systèmes augmentés dont le bloc $(2,2)$ est petit comparé au bloc $(1,1)$, nous avons développé une technique qui contraint les variables associées aux multiplicateurs de Lagrange (associées au bloc $(2,2)$) d'être des variables interfaces. Cette technique a deux avantages principaux. En premier lieu, elle assure qu'aucun des problèmes locaux n'est structurellement singulier, elle assure même que les problèmes locaux sont symétriques définis positifs. Cette caractéristique autorise l'utilisation de factorisation de Cholesky pour les problémes locaux et de réduire l'effort de calcul (mémoire et opérations flottantes) comparé à une décomposition LU. Dans ce rapport, nous décrivons comment le problème de partitionnement de graphe est formulé pour satisfaire les conditions mentionnées ci-dessus. Nous présentons des performances numériques et parallèles sur des systèmes de grande taille issus de la discrétisation par élements finis de problèmes d'élasticité linéaire en mécanique des structures tridimensionnelle. Pour ces problèmes, des conditions aux limites sont imposées en utilisant des multiplicateurs de Lagrange.

Mots-clés : Solveurs linéaires hybrides directs/itératifs, sysèmes augmentés, calcul paralléle scientifique, problémes 3D de grandes tailles, calcul haute performance.

Abstract

We consider the parallel iterative solution of indefinite linear systems given as augmented systems where the $(1,1)$ block is symmetric positive definite and the $(2,2)$ block is zero. Our numerical technique is based on an algebraic non overlapping domain decomposition technique that only exploits the graph of the sparse matrix. This approach to high-performance, scalable solution of large sparse linear systems in parallel scientific computing is to combine direct and iterative methods. Such a hybrid approach exploits the advantages of both direct and iterative methods. The iterative component allows us to use a small amount of memory and provides a natural way for parallelization. The direct part provides favorable numerical properties. The graph of the sparse matrix is split into sub-graphs and a condensed linear system, namely the Schur complement system, is solved iteratively for the unknowns associated with the interface between the sub-graphs; a sparse direct solver is used for the variables associated with the internal parts of the sub-graphs. For augmented systems a special attention should be paid to ensure the non singularity of the local sub-problems so that the Schur complement is defined. For augmented systems, where the $(2,2)$ block is small compared to the $(1,1)$ block, we design a special technique that enforces the Lagrange multiplier variables (associated with the $(2,2)$ block) to be in the interface unknowns. This technique has two main advantages. First, it ensures that none of the local sub-systems is structurally singular and for symmetric positive definite $(1,1)$ block, it even ensures that those sub-matrices are also symmetric positive definite. This latter property enables us to use a Cholesky factorization for the internal sub-problems which reduces the computational complexity (in term of floating point operation counts and memory consumption) compared to a more general LU decomposition. In this paper, we describe how the graph partitioning problem is formulated to comply with the above mentioned constraints. We report numerical and parallel performance of the scheme on large matrices arsing from the finite element discretization of linear elasticity in structural mechanic problems. For those problems some boundary conditions are modeled through the use of Lagrange multipliers.

Keywords: augmented/indefinite linear systems, sparse linear systems, direct-iterative hybrid methods, high-performance computing.

1 Overview

Solving large sparse linear systems $\mathcal{A}x = b$, where A is a given matrix, b is a given vector, and x is an unknown vector to be computed, appears often in the inner-most loops of intensive simulation codes, and is consequently the most time-consuming computation in many large-scale computer simulations in science and engineering.

There are two basic approaches for solving linear systems of equations: sparse direct methods and iterative methods. Sparse direct solvers have been for years the methods of choice for solving linear systems of equations because of their reliable numerical behavior. However, it is nowadays admitted that such approaches are not scalable in terms of computational complexity or memory for large problems such as those arising from the discretization of large 3-dimensional partial differential equations (PDEs). Iterative methods, on the other hand, generate sequences of approximations to the solution. These methods have the advantage that the memory requirements are small. Also, they tend to be easier to be parallelized than direct methods. However, the main problem with this class of methods is the rate of convergence, which depends on the properties of the matrix. One way to improve the convergence rate is through preconditioning, which is another difficult problem.

Our approach to high-performance, scalable solution of large sparse linear systems in parallel scientific computing is to combine direct and iterative methods. Such a hybrid approach exploits the advantages of both direct and iterative methods. The iterative component allows us to use a small amount of memory and provides a natural way for parallelization. The direct part provides favorable numerical properties.

The general underlying ideas are not new. They have been used to design domain decomposition techniques for the numerical solution of PDEs. Domain decomposition refers to the splitting of the computational domain into sub-domains with or without overlap. The splitting strategies are generally governed by various constraints/objectives but the main one is to enhance parallelism. The numerical properties of the PDEs to be solved are usually extensively exploited at the continuous or discrete levels to design the numerical algorithms. Consequently, the resulting specialized technique will only work for the class of linear systems associated with the targeted PDEs. In our work, we develop domain decomposition techniques for general unstructured linear systems. More precisely, we consider numerical techniques based on a non-overlapping decomposition of the graph associated with the sparse matrices. The vertex separator, constructed using graph partitioning, will define the interface variables that will be solved iteratively using a Schur complement approach, while the variables associated with the interior sub-graphs will be handled by a sparse direct solver. Although the Schur complement system is usually more tractable than the original problem by an iterative technique, preconditioning treatment is still required.

In this paper, we describe how our approach has to be adpated to cope with the solution of augmented systems as those arising for instance in constrained optimization or numerical scheme using penalization techniques.

2 Numerical Scheme

2.1 General presentation

In this section, methods based on non-overlapping regions are described. Let us now describe this technique and let

$$\mathcal{A}x = b \tag{1}$$

be the linear problem. For the sake of simplicity, we assume that $\mathcal{A}$ is symmetric in pattern and we denote $G = \{V, E\}$ the adjacency graph associated with $\mathcal{A}$. In this graph, each vertex is associated with a row or column of the matrix $\mathcal{A}$ and it exists an edge between the vertices i and j if the entry $a_{i,j}$ is non zero.

The governing idea behind substructuring or Schur complement methods is to split the unknowns in two subsets. We assume that the graph G is partitioned into N non-overlapping sub-graphs G_1, ..., G_N with interiors I_i and boundaries Γ_1, ..., Γ_N. We note I and Γ the entire interior ($I = \cup I_i$) and entire interface ($\Gamma = \cup \Gamma_i$), respectively. Equation (1) can then be written as the following block reordered linear system:

$$\begin{pmatrix} \mathcal{A}_{II} & \mathcal{A}_{I\Gamma} \\ \mathcal{A}_{\Gamma I} & \mathcal{A}_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} x_I \\ x_\Gamma \end{pmatrix} = \begin{pmatrix} b_I \\ b_\Gamma \end{pmatrix}, \tag{2}$$

where x_Γ contains all unknowns associated with sub-graph interfaces and x_I contains the remaining unknowns associated with sub-graph interiors. Because the interior points are only connected to either interior points in the same sub-graph or with points on the boundary of the sub-graphs, the matrix $\mathcal{A}_{II}$ has a block diagonal structure, where each diagonal block corresponds to one sub-graph. Eliminating x_I from the second block row of Equation (2) leads to the reduced system

$$\mathcal{S} x_\Gamma = f, \tag{3}$$

where

$$\mathcal{S} = \mathcal{A}_{\Gamma\Gamma} - \mathcal{A}_{\Gamma I}\mathcal{A}_{II}^{-1}\mathcal{A}_{I\Gamma} \text{ and } f = b_\Gamma - \mathcal{A}_{\Gamma I}\mathcal{A}_{II}^{-1} b_I. \tag{4}$$

The matrix $\mathcal{S}$ is referred to as the *Schur complement matrix*. This reformulation leads to a general strategy for solving (1). Specifically, an iterative method can be applied to (3). Once x_Γ is known, x_I can be computed with one additional solve on the sub-graph interiors.

We notice that if two sub-graphs G_i and G_j share an interface then $\Gamma_i \bigcap \Gamma_j \neq \emptyset$. As interior unknowns are no longer considered, new restriction operators must be defined as follows. Let $\mathcal{R}_{\Gamma_i} : \Gamma \rightarrow \Gamma_i$ be the canonical point-wise restriction which maps full vectors defined on Γ into vectors defined on Γ_i. Thus, in the case of many sub-graphs, the fully assembled global Schur $\mathcal{S}$ is obtained by summing the contributions over the sub-graphs. The global Schur complement matrix (4) can be written as the sum of elementary matrices

$$\mathcal{S} = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T \mathcal{S}_i \mathcal{R}_{\Gamma_i}, \tag{5}$$

where

$$\mathcal{S}_i = \mathcal{A}_{\Gamma_i\Gamma_i} - \mathcal{A}_{\Gamma_i\mathcal{I}_i}\mathcal{A}_{\mathcal{I}_i\mathcal{I}_i}^{-1}\mathcal{A}_{\mathcal{I}_i\Gamma_i} \tag{6}$$

is a local Schur complement associated with G_i. It can be defined in terms of sub-matrices from the local matrix $\mathcal{A}_i$ defined by

$$\mathcal{A}_i = \begin{pmatrix} \mathcal{A}_{\mathcal{I}_i\mathcal{I}_i} & \mathcal{A}_{\mathcal{I}_i\Gamma_i} \\ \mathcal{A}_{\Gamma_i\mathcal{I}_i} & \mathcal{A}_{\Gamma_i\Gamma_i} \end{pmatrix}. \tag{7}$$

While the Schur complement system is significantly better conditioned than the original matrix $\mathcal{A}$, it is important to consider further preconditioning when employing a Krylov method.

We introduce the general form of the preconditioner considered in this work. The preconditioner presented below was originally proposed in [3] in two dimensions and successfully applied to large three dimensional problems and real life applications in [7, 8]. To describe this preconditioner we define the local assembled Schur complement, $\bar{\mathcal{S}}_i = \mathcal{R}_{\Gamma_i}\mathcal{S}\mathcal{R}_{\Gamma_i}^T$, that corresponds to the restriction of the Schur complement to the interface Γ_i. This local assembled preconditioner can be built from the local Schur complements $\mathcal{S}_i$ by assembling their diagonal blocks.

With these notations the preconditioner reads

$$M_d = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T \bar{\mathcal{S}}_i^{-1} \mathcal{R}_{\Gamma_i}. \tag{8}$$

where d stands for *dense* to denote that each matrix $\bar{\mathcal{S}}_i$ is dense.

Such diagonal blocks, that overlap, are similar to the classical block overlap of the Schwarz method when writing in a matrix form for 1D decomposition. Because of this link, the preconditioner defined by (8) is referred to as algebraic additive Schwarz for the Schur complement. One advantage of using the assembled local Schur complements instead of the local Schur complements (like in the Neumann-Neumann [2, 4] where the preconditioner would be equal to $M_{NN} = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T \mathcal{S}_i^\dagger \mathcal{R}_{\Gamma_i}$) is that in the SPD case the assembled Schur complements cannot be singular (as $\mathcal{S}$ is SPD [3]).

The original idea of non-overlapping domain decomposition method consists into subdividing the graph into sub-graphs that are individually mapped to one processor. With this data distribution, each processor P_i can concurrently partially factorize it to compute its local Schur complement $\mathcal{S}_i$. This is the first computational phase that is performed concurrently and independently by all the processors. The second step corresponds to the construction of the preconditioner. Each processor communicates with its neighbors (in the graph partitioning sense) to assemble its local Schur complement $\bar{\mathcal{S}}_i$ and performs its factorization. This step only requires a few point-to-point communications. Finally, the last step is the iterative solution of the interface problem (3). For that purpose, parallel matrix-vector product involving $\mathcal{S}$, the preconditioner M_{d} and dot-product calculation must be performed. For the matrix-vector product each processor P_i performs its local matrix-vector product involving its local Schur complement and communicates with its neighbors to assemble the computed vector to comply with Equation (5). Because the preconditioner (8) has a similar form as the Schur complement (5), its parallel application to a vector is implemented similarly. Finally, the dot products are performed as follows: each processor P_i performs its local dot-product and a global reduction is used to assemble the result. In this way, the hybrid implementation can be summarized by the above main three phases.

The construction of the proposed local preconditioners can be computationally expensive because the dense matrices $\mathcal{S}_i$ should be factorized. We intend to reduce the storage and the computational cost to form and apply the preconditioner by using sparse approximation of $\bar{\mathcal{S}}_i$ in M_d following the strategy described by (9). The approximation $\hat{\mathcal{S}}_i$ can be constructed by dropping the elements of $\bar{S}_i$ that are smaller than a given threshold. More precisely, the following symmetric dropping formula can be applied:

$$\hat{s}_{\ell j} = \begin{cases} 0, & \text{if } |\bar{s}_{\ell j}| \leq \xi(|\bar{s}_{\ell\ell}| + |\bar{s}_{jj}|), \\ \bar{s}_{\ell j}, & \text{otherwise,} \end{cases} \tag{9}$$

where $\bar{s}_{\ell j}$ denotes the entries of $\bar{\mathcal{S}}_i$. The resulting preconditioner based on these sparse approximations reads

$$M_{\mathrm{sp}} = \sum_{i=1}^{N} \mathcal{R}_{\Gamma_i}^T \hat{\mathcal{S}}_i^{-1} \mathcal{R}_{\Gamma_i}.$$

We notice that such a dropping strategy preserves the symmetry in the symmetric case but it requires to first assemble $\bar{S}_i$ before sparsifying it.

As summary, the main phases of the method are:

- *Phase 0:* the partitioning of the adjacency graph of the sparse matrix and corresponding data distribution on the processors that hold the submatrices (7), part of the initial guess and right-hand side.
- *Phase 1:* the initialization phase that is the same for all the variants of the preconditioners. It consists into the factorization of the local internal problem and the computation of the Schur complement. It depends only on the size of the local subdomains and on the size of the local Schur complements;
- *Phase 2:* the preconditioner setup phase that differs between the dense and the sparse variants. It depends also on the size of the local Schur complements, and on the dropping parameter ξ for the sparse variants;

- *Phase 3:* the iterative loop which is related to the convergence rate and to the time per iteration. This latter depends on the efficiency of the matrix-vector product kernel (explicit v.s. implicit), and on the preconditioner application, that is the forward/backward substitutions (dense v.s. sparse).
- *Phase 4:* the concurent solution of the local sub-problems to compute the internal variables using the computed interface unknowns at Phase 3.

2.2 Dealing with augmented systems

In this paper we address the solution of augmented systems where the sparse matrix exhibits a 2×2 block structure where the $(2, 2)$ block is zero. It usually writes

$$\begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} \begin{pmatrix} x \\ \lambda \end{pmatrix} = b \tag{10}$$

where λ usually denotes the Lagrange multipliers introduce to cope with some constraints.

For such linear systems involving Lagrange multipliers an additional constraint should be taken into account. If the matrix graph is decomposed without considering the Lagrange multipliers we might end-up with a splitting for which Lagrange multipliers coupled unknowns that are on the interface while the Lagrange multiplier is considered as an "internal" unknown. In such a situation, the matrix associated with the internal unknowns has a zero row/column and is consequently structurally singular. The Schur complement does not exist and the hybrid technique breaks down.

Straight partitioning. A simple and systematic way to fix this weakness it to enforce the Lagrange multiplier unknowns to be moved into the interface. If the partitioner has produced balanced subgraphs with minimal interfaces, moving the Lagrange multipliers into the interfaces significantly deteriorates the quality of the partition. A good partitioning strategy should then balance the size of the subgraphs while minimizing and balancing the interface sizes but also balance the distribution of the Lagrange multipliers among the subgraphs. In that respect, when the Lagrange multipliers are moved into the interfaces, the interfaces remain balanced.

Weighted partitioning. In order to achieve this, we do not apply a graph partitioner on the adjacency graph of the matrix but add some weights to the vertices. The mesh partitioner we use is Metis [9] that enables us to consider two weights per vertex, one associated with the workload ($weight\_vertex$) and the other with the amount of communication ($weight\_comm$). In order to balance the Lagrange multipliers among the sub-graphs, for the Lagrange multiplier variables we relax the weight associated with the communication (i.e. communication weight set to zero) and penalize their workload by setting their work weight to a large value. For the other unknowns, the work weights are set to one, while the communication weight is set to the number of adjacent vertices. Among the numerous weighted variants that we have experimented with, the following one was the best we found:

$$\begin{cases} weight\_comm(var) = 0 & \text{if } var \text{ belongs to the (2,2) block} \\ weight\_comm(var) = nadj & \text{otherwise,} \end{cases}$$

$$\begin{cases} weight\_vertex(var) = large\ value & \text{if } var \text{ belongs to the (2,2) block} \\ weight\_vertex(var) = 1 & \text{otherwise.} \end{cases}$$

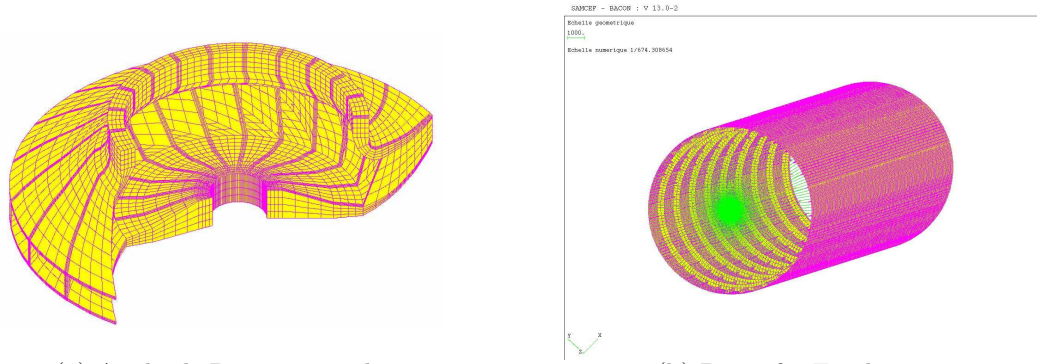
Thus, the objective is to try to fit the maximum of unknowns associated with Lagrange equations into the interface and to minimize the number of these equations belonging to one subdomain. With this strategy, we first attempt to have as much Lagrange unknowns as possible in the interface. Secondly, we try to have the remaining ones equitably distributed among the subdomains. This latter feature enables us to preserve a good balance of the interface even after these Lagrange unknowns are moved into the subdomain interfaces as treatment of the possible singularity.

This technique has two main advantages. First, it ensures that none of the local sub-systems is structurally singular and for symmetric positive definite $(1,1)$ block, it even ensures that those sub-matrices are also symmetric positive definite. This latter property enables us to use a Cholesky factorization for the internal sub-problems which reduces the computational complexity (in term of floating point operation counts and memory consumption) compared to a more general LU decomposition.

3 Numerical and parallel performances

3.1 Experimental framework

For the sake of exposure, we display in Figure 1 the mesh associated with the matrices. We remind that our numerical scheme does not use any geometrical mesh information but only the pattern of the matrices. In Table 1 we display for the different mesh geometries the various sizes of the



(a) A wheel: Rouet example. (b) Part of a Fuselage.

Figure 1: structural mechanics meshes.

problems we have experimented.

Rouet example	
# degrees of freedoms	# of Lagrange equations
$1.3{\cdot}10^6$	13,383

Fuselage example	
# degrees of freedoms	# of Lagrange equations
$6.5{\cdot}10^6$	3,024

Table 1: Characteristics of the various structural mechanics problems.

Our target parallel machine is an IBM JS21 supercomputer installed at CERFACS to address diverse applications in science and engineering. It works currently with a peak computing performance of 2.2 TeraFlops. This is a 4-core blade server for applications requiring 64-bit computation. It is ideal for computer-intensive applications and transactional Internet servers. We use IBM ESSL vendor BLAS and LAPACK libraries (LAPACK is used only for the construction of a preconditioner of type M_{d} in Phase 2). The MUMPS package [1] is used to perform the partial sparse factorization of a local matrix (Phase 1) and eventually the related solution step on the interior unknowns (Phase 4). It is also used for the construction of a preconditioner of type M_{sp} (in Phase 2). For the solution of the indefinite systems in Phase 3, we choose full-GMRES as Krylov subspace method [11, 10]. We consider the implementation [5] with the ICGS (Iterative Classical

Gram-Schmidt) orthogonalization variant that provides us with the best trade-off between numerical orthogonality and parallel performance (as the dot-product are merged). As mentioned earlier, Metis is used to perform the initial domain decomposition (in Phase 0).

3.2 Partitioning strategy effect

		Maximal interface size	Preconditioner setup	# iter	Iterative loop	Total time
Rouet	Straight	13492	255	76	77	473
16 subdomains	Weighted	10953	137	79	61	264
Rouet	Straight	11176	141	108	80	242
32 subdomains	Weighted	7404	44	106	45	111
Fuselage	Straight	11739	168	162	144	347
32 subdomains	Weighted	6420	30	176	58	124
Fuselage	Straight	10446	120	217	170	305
64 subdomains	Weighted	4950	15	226	54	82

Table 2: Effects of the partitioning strategy (using a preconditioner based on dense local assembled Schur complements).

The effect of the partitioning strategies presented in Section 2.2 on the efficiency of the parallel hybrid solver is illustrated in Table 2. We observe that the weighted strategy often generates partitions with smaller interface sizes than the straight strategy. Furthermore, a poor load balance causes inadequate performance of the algorithm. The main drawback is that the number of equations assigned to each processor as well as the local interface sizes vary uncommonly. Thus the local Schur complement sizes highly vary causing unbalanced preconditioner setup where the fastest processors should wait the slowest one before starting the iterative step. It also induces large idle time at each global synchronization points implemented for the calculation of the dot-product (global reduction) in the iterative process. Moreover, the computing and memory cost of the preconditioner is related to the interface size, thus large subdomain interfaces imply inefficiency in both computing time and memory required to store the preconditioner. The importance of this latter is clearly exposed in Table 2. On the first hand, we can see that for large subdomain sizes, the preconditioner setup time, which cost increases as function of $O(n^3)$ (where n denotes the size of the largest interface of a sub-graph), becomes very expensive. On the other hand, the iterative loop time depends closely on the matrix-vector product and from the preconditioner application costs. For large interfaces (large local Schur complements), both the matrix-vector calculation and the preconditioner application become expensive (because unbalanced), thus increasing the overall computing time and requiring a large amount of data storage.

As consequence, it is imperative to strive for a very balanced load. We have resorted these problems by the use of multiple constraints graphs partitioning. In our numerical experiments, the partitioning is applied with weighted vertices based on a combination of the characteristics described above.

3.3 Impact of the sparsification

While the primary purpose of this section is to focus on the way the numerical performance of the sparse preconditioner can be stated, it also gives us tips for describing both computational benefits and data storage of the sparse algorithm. In this section we first investigate the advantages in term of setup cost and memory storage associated with the use of the sparsification strategy for the preconditioner. Then we focus on the numerical behaviour of the resulting preconditioners.

We report results for the Fuselage test case with 6.5 million unknowns and 1.3 million unknowns for the Rouet example. We display in Table 3, the memory space and the computing time required by the preconditioner on each processor for different values of the sparsification dropping parameter ξ. The results presented in this table are for both test cases with 16 subdomains. The maximal local subdomain interface size for the Fuselage problem on this decomposition has 9444 unknowns whereas for the Rouet problem it has of 10953 unknowns. It can be seen that a lot of storage can be saved and that the preconditioner setup phase is strongly reduced.

ξ	0	5.10^{-7}	10^{-6}	5.10^{-6}
Rouet problem with 1.3 Mdof				
Memory	960_{MB}	384_{MB}	297_{MB}	153_{MB}
Kept percentage	100%	40%	31%	16%
Preconditioner setup	137	145	96	37
Fuselage problem with 6.5 Mdof				
Memory	710_{MB}	122_{MB}	92.7_{MB}	46.3_{MB}
Kept percentage	100%	17%	13%	7%
Preconditioner setup	89	26	19.5	10.8

Table 3: Preconditioner computing time (*sec*) and amount of memory (MB) in M_{sp} v.s. M_{d} for various choices of the dropping parameter, when the problems are mapped onto 16 processors.

The cost of the preconditioner is not the only component to consider for assessing its interest. We should analyze its numerical performance. For that purpose, we report in Figure 2, the convergence history for various choices of ξ depicted in Table 3, for both the Fuselage and the Rouet problem. For both test cases, we depict on the left-hand side of the Figure the convergence history as a function of the iterations, and on the right-hand side, the convergence history as a function of the computing time. For the sake of completeness, we also report on the performance of a parallel sparse direct solution, that can be considered for these sizes of problems. It is clear that the sparsified variant outperforms its dense counterpart. However, for these real engineering problems, the sparse preconditioner has to retain more information about the Schur complement than for the academic cases. For these problems, in order to preserve the numerical quality we need to keep more than 10% of the Schur entries whereas 2% in easier academic cases were sufficient [7, 8].

In term of computing time, we can see that the sparse preconditioner setup is more than 3 times faster than the dense one. For example if we look at Table 3 for the Fuselage example, we can see that for $\xi = 10^{-6}$ it is 4 times faster than the dense one (19.5 v.s. 89 seconds). In term of global computing time it can be seen that the sparse algorithm is about twice faster. The very few extra iterations introduced by the sparse variant are compensated by a faster iteration due to the reduced floating-point operations associated with it. For the Rouet test case the sparse variants behave comparably as for the Fuselage problem.

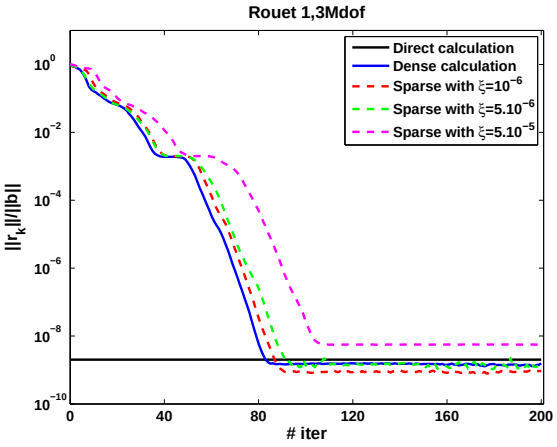
3.4 Parallel performance

In this section we report on parallel experiments and study strong scalability. The initial guess is always the zero vector and convergence is detected when the normwise backward error becomes less than 10^{-8} or when 300 steps have been unsuccessfully performed.

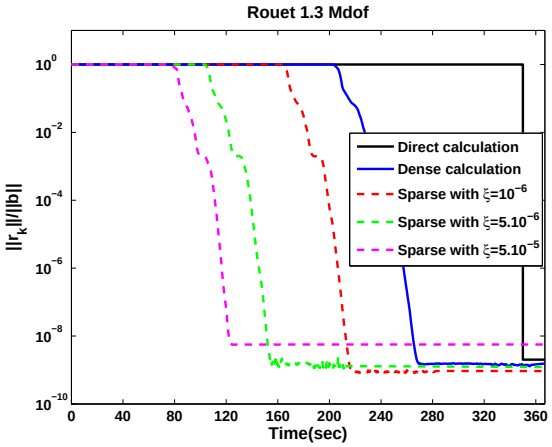
3.4.1 Numerical scalability on parallel platforms

We now describe how both preconditioners (M_{d} and M_{sp}) affect the convergence rate of the iterative hybrid solver and what numerical performance is achieved. In Table 4 we display the number of iterations obtained for different problem sizes. Each original problem is split into 8, 16, 32, and

Rouet 1.3 Mdof

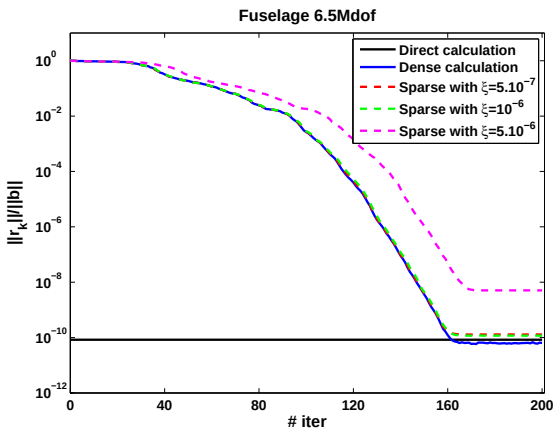


(a) history v.s. iterations.

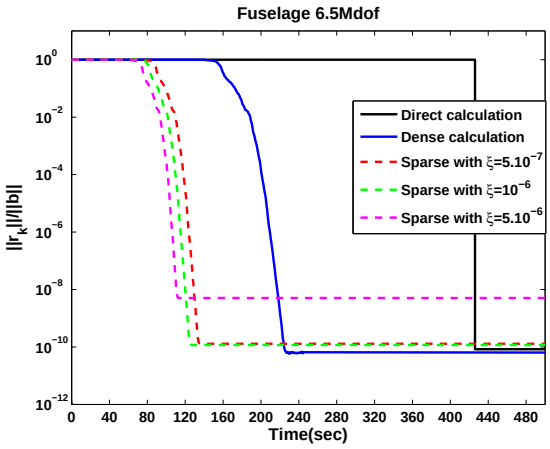


(b) history v.s. time.

Fuselage 6.5 Mdof



(a) history v.s. iterations.



(b) history v.s. time.

Figure 2: Convergence history of full-GMRES for Fuselage and Rouet problems mapped onto 16 processors, of the direct, the Hybrid-dense (M_{d}) and the Hybrid-sparse (M_{sp}) solvers for various sparsification dropping thresholds (Left: scaled residual versus iterations, Right: scaled residual versus time).

64 subdomains. First we test the quality of the sparse preconditioner M_{sp} generated by varying the sparsification threshold ξ and compare the results to the dense preconditioner M_{d}. The results show that the sparse preconditioner convergence is similar to the one observed using the dense preconditioner. The Fuselage is a relatively difficult problem with badly scaled matrices; when increasing the number of subdomains, the reduced system (global interface system) inherits the bad scaling. The values of the entries vary by more than 15 orders of magnitude. It is much more difficult to compute the solution, thus the small increase in the number of iterations when increasing the number of subdomains is not surprising. The attractive feature is that both preconditioners still achieve the same backward error level even for large number of subdomains. A similar analysis was performed for the Rouet test case, investigating the effect of domain decomposition. As expected, the sparse preconditioner performs as well as the dense preconditioner M_{d}, for the different decomposition considered here. The gap in the number of iterations is between 1 and 5 for the most difficult cases, whereas the sparse variants save a lot of computing resources as described in the next subsection. Regarding the number of iterations when increasing the number of processors, we can still observe, as in the Fuselage test case, a slight growth in the iteration numbers. For example when we increase the number of subdomains 8 times, the iteration number is multiplied by 2.8.

To conclude on this aspect, we would like to underline the fact that either the dense preconditioner M_{d}, or the sparse variant M_{sp} are able to ensure fast convergence of the Krylov solvers on our test cases of structural mechanical applications, and even when increasing the number of subdomains. More precisely some tests on the 6.5 million Fuselage were performed on more than 64 processors. The results show that the preconditioner still guarantees a reasonable number of iterations (for example 275 iterations on 96 processors).

3.4.2 Parallel scalability performance

This subsection is devoted to the presentation and analysis of the parallel performance of both preconditioners. A brief comparison with a direct method solution is also given. We consider experiments where we increase the number of processors while the size of the initial linear system is kept constant. Such experiments mainly emphasize the interest of parallel computation in reducing the elapsed time to solve a problem of a prescribed size.

For the sake of completeness, we report in tables 5 and 6 a detailed description of the computing time for all problems described above, for both preconditioners, and for different choices of the dropping parameter ξ. We also report the solution time using the parallel sparse direct solver, where neither the associated distribution or redistribution of the matrix entries, nor the time for

Fuselage example				
# processors	8	16	32	64
M_{d}	98	147	176	226
M_{sp} ($\xi = 5.10^{-7}$)	99 (13%)	147 (17%)	176 (22%)	226 (30%)
M_{sp} ($\xi = 10^{-6}$)	101 (10%)	148 (13%)	177 (18%)	226 (24%)
M_{sp} ($\xi = 5.10^{-6}$)	121 (5%)	166 (7%)	194 (9%)	252 (13%)
Rouet example				
# processors	8	16	32	64
M_{d}	59	79	106	156
M_{sp} ($\xi = 10^{-6}$)	59 (24%)	83 (31%)	108 (39%)	157 (47%)
M_{sp} ($\xi = 5.10^{-6}$)	60 (11%)	87 (16%)	114 (21%)	162 (27%)
M_{sp} ($\xi = 10^{-5}$)	63 (7%)	89 (11%)	116 (15%)	166 (20%)

Table 4: Number of preconditioned GMRES iterations (and percentage of kept entries) to achieve a normwise backward error of 10^{-8}.

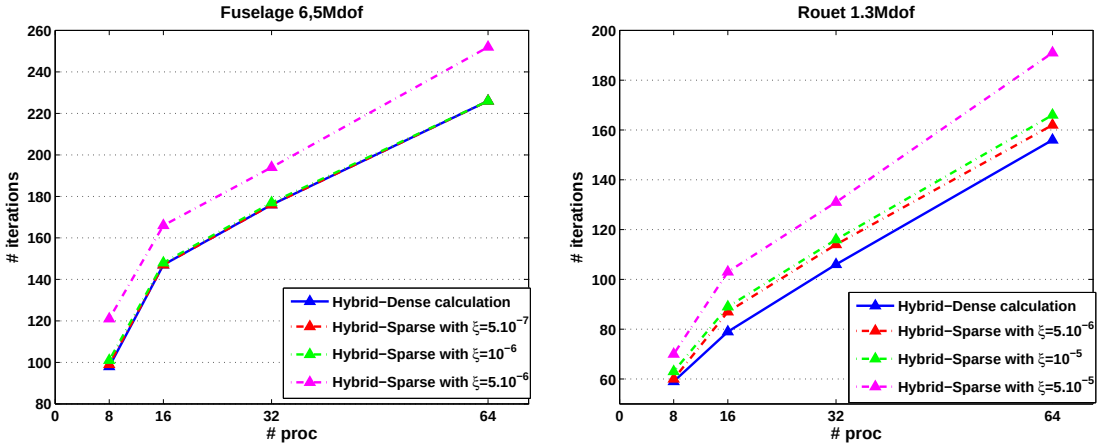


Figure 3: Numerical behaviour (number of iterations) for the test cases, when increasing the number of processors and for different values of ξ.

the symbolic analysis are taken into account in our time measurements. The main purpose of this detailed presentation is to evaluate the performance of the three main computationally intensive phases (Phase 1 to Phase 3) of the hybrid solver in a very comprehensive way.

Regarding the preconditioning step, it is still clear that the sparsified variant is of great interest as it reduces considerably the time to apply the preconditioner, which leads to a significant reduction of the time per iteration compared to the dense counterpart.

At each iteration, the third step also performs global reduction. It was observed that, the relative cost of this reduction is negligible compared to the other steps of the algorithm, it increases by less than 2.10^{-3} seconds when increasing the number of processors from 8 to 64.

Thus, by looking at the time per iteration, we might conclude that the extra number of iterations cost introduced when increasing the number of subdomains is in most cases compensated by the cheaper cost of the resulting time per iteration. Regarding the sparse variants, we notice a significant gain in computing time for suited dropping thresholds. We should mention that dropping too many entries often lead to a significant increase of the iterative loop time as the number of iterations grows notably. On the other side, only dropping a very few entries (very small ξ) also leads to higher time per iteration than a dense approach. A good trade-off between numerical robustness and fast calculation should be found to ensure the best performance of the sparsified approach.

In Figure 4, we finally compare our hybrid approach (actually the two variants of the preconditioners discussed in this article) against the use of a pure direct solver, MUMPS to solve the full linear system. The direct solver is used in the context of general symmetric matrices, In terms of permutation, we have performed some experiments to compare between the nested dissection Metis routine of ordering and the Approximate Minimum Degree (AMD) ordering. The best performance was observed when we used the Metis routine. We performed runs using assembled and distributed matrix entries. We mention that neither the associated distribution or redistribution of the matrix entries, nor the time for the symbolic analysis are taken into account in our time measurements. For the direct method, we only report the minimum elapsed time for the factorization and for the backward/forward substitutions. We illustrate the corresponding computing time when increasing the number of processors, for all the tests cases. Compared with the direct method, our hybrid approach gives always the fastest scheme. Over the course of a long simulation, where each step requires the solution of a linear system, our approach represents a significant saving in computing resources; this observation is especially valid for the attractive sparse variant.

	Total solution time			
# processors	8	16	32	64
Direct	655.5	330.0	201.4	146.3
M_{d}	525.1	217.2	124.1	82.2
M_{sp} ($\xi = 5.10^{-7}$)	338.0	129.0	94.2	70.2
M_{sp} ($\xi = \ 10^{-6}$)	322.8	120.1	87.9	65.1
M_{sp} ($\xi = 5.10^{-6}$)	309.8	110.9	82.8	63.2
	Time in the iterative loop			
# processors	8	16	32	64
M_{d}	94.1	77.9	58.1	54.2
M_{sp} ($\xi = 5.10^{-7}$)	59.4	52.6	42.2	42.9
M_{sp} ($\xi = \ 10^{-6}$)	57.6	50.3	38.9	40.7
M_{sp} ($\xi = 5.10^{-6}$)	60.5	49.8	40.7	45.4
	# iteration			
# processors	8	16	32	64
M_{d}	98	147	176	226
M_{sp} ($\xi = 5.10^{-7}$)	99	147	176	226
M_{sp} ($\xi = \ 10^{-6}$)	101	148	177	226
M_{sp} ($\xi = 5.10^{-6}$)	121	166	194	252
	Time per iteration			
# processors	8	16	32	64
M_{d}	0.96	0.53	0.33	0.24
M_{sp} ($\xi = 5.10^{-7}$)	0.60	0.36	0.24	0.19
M_{sp} ($\xi = \ 10^{-6}$)	0.57	0.34	0.22	0.18
M_{sp} ($\xi = 5.10^{-6}$)	0.50	0.30	0.21	0.18
	Preconditioner setup time			
# processors	8	16	32	64
M_{d}	208.0	89.0	30.0	15.0
M_{sp} ($\xi = 5.10^{-7}$)	55.6	26.1	16.0	14.3
M_{sp} ($\xi = \ 10^{-6}$)	42.2	19.5	13.0	11.4
M_{sp} ($\xi = 5.10^{-6}$)	26.3	10.8	6.1	4.8
	Max of the local Schur size			
# processors	8	16	32	64
All preconditioners	12420	9444	6420	4950
	Initialization time			
# processors	8	16	32	64
All preconditioners	223.0	50.3	36.0	13.0

Table 5: Detailed performance for the Fuselage problem with 6.5 million unknowns when the number of processors is varied for the various variants of the preconditioner and for various choices of ξ. We also report the "factorization+solve" time using the parallel sparse direct solver.

	Total solution time			
# processors	8	16	32	64
Direct	435.1	350.0	210.7	182.5
M_{d}	453.7	264.6	110.9	70.1
M_{sp} ($\xi = 5.10^{-6}$)	277.5	151.7	86.5	51.4
M_{sp} ($\xi = 10^{-5}$)	246.7	134.6	70.5	47.2
M_{sp} ($\xi = 5.10^{-5}$)	214.0	122.1	63.4	46.2
	Time in the iterative loop			
# processors	8	16	32	64
M_{d}	57.2	60.8	44.5	42.1
M_{sp} ($\xi = 5.10^{-6}$)	42.0	47.9	37.6	35.6
M_{sp} ($\xi = 10^{-5}$)	42.2	41.8	30.2	33.2
M_{sp} ($\xi = 5.10^{-5}$)	38.5	44.3	34.1	34.4
	# iteration			
# processors	8	16	32	64
M_{d}	59	79	106	156
M_{sp} ($\xi = 5.10^{-6}$)	60	87	114	162
M_{sp} ($\xi = 10^{-5}$)	63	89	116	166
M_{sp} ($\xi = 5.10^{-5}$)	70	103	131	191
	Time per iteration			
# processors	8	16	32	64
M_{d}	0.97	0.77	0.42	0.27
M_{sp} ($\xi = 5.10^{-6}$)	0.70	0.55	0.33	0.22
M_{sp} ($\xi = 10^{-5}$)	0.67	0.47	0.26	0.20
M_{sp} ($\xi = 5.10^{-5}$)	0.55	0.43	0.26	0.18
	Preconditioner setup time			
# processors	8	16	32	64
M_{d}	235.0	137.0	43.5	19.0
M_{sp} ($\xi = 5.10^{-6}$)	74.0	37.0	26.0	6.8
M_{sp} ($\xi = 10^{-5}$)	43.0	26.0	17.5	5.0
M_{sp} ($\xi = 5.10^{-5}$)	14.0	11.0	6.5	2.8
	Max of the local Schur size			
# processors	8	16	32	64
All preconditioners	13296	10953	7404	5544
	Initialization time			
# processors	8	16	32	64
All preconditioners	161.5	66.8	22.9	9.0

Table 6: Detailed performance for the Rouet problem with 1.3 Mdof when the number of processors is varied for the various variants of the preconditioner and for various choices of ξ. We also report the "factorization+solve" time using the parallel sparse direct solver.

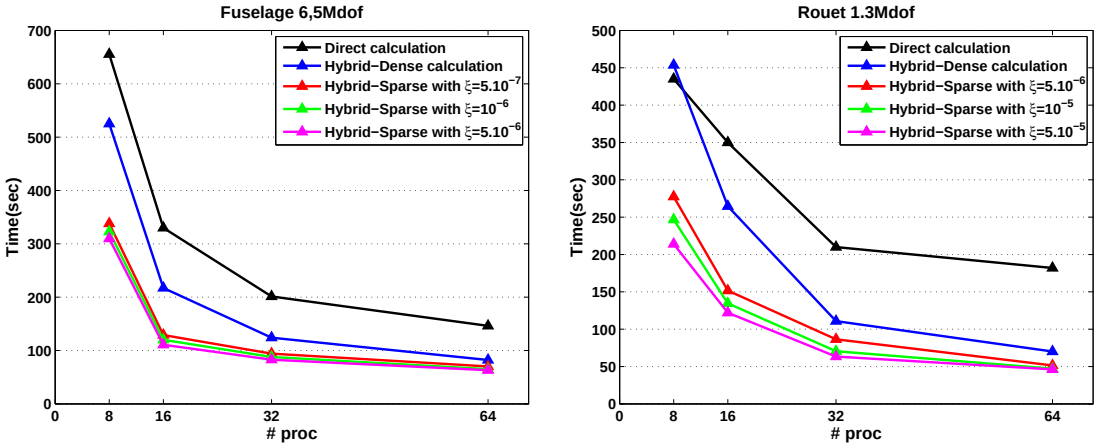


Figure 4: Parallel performance when increasing the number of processors for different values of ξ are reported.

We now investigate the analysis in term of memory requirement. We depict in Table 7 the maximal peak of memory required on the subdomains to compute the factorization and either the dense preconditioner for the $hybrid - M_{\rm d}$ method or the sparse preconditioner with $\xi = 5.10^{-6}$ for the $hybrid - M_{\rm sp}$ method. We report also the average memory required by the direct method.

For each test case, we report in each row of Table 7 the amount of memory storage required (in MB) for the different decomposition described in this subsection. This amount is huge for small number of subdomains. This is due to the fact that the size of the local Schur complements is extremely large. Furthermore the large number of unknowns associated with the interior of each subdomain leads to local factorizations that are memory consuming. A feature of the sparse variants is that they reduce the preconditioner memory usage.

# processors		4	8	16	32	64
Rouet $1.3{\cdot}10^6 dof$	*Direct*	-	5368	2978	1841	980
	$Hybrid - M_{\rm d}$	-	5255	3206	1414	739
	$Hybrid - M_{\rm sp}$	-	3996	2400	1068	560
Fuselage $3.3{\cdot}10^6 dof$	*Direct*	5024	3567	2167	990	669
	$Hybrid - M_{\rm d}$	6210	3142	1714	846	399
	$Hybrid - M_{\rm sp}$	5167	2556	1355	678	320
Fuselage $4.8{\cdot}10^6 dof$	*Direct*	9450	6757	3222	1707	1030
	$Hybrid - M_{\rm d}$	8886	4914	2994	1672	623
	$Hybrid - M_{\rm sp}$	7470	4002	2224	1212	495
Fuselage $6.5{\cdot}10^6 dof$	*Direct*	-	8379	5327	2148	1503
	$Hybrid - M_{\rm d}$	-	6605	3289	1652	831
	$Hybrid - M_{\rm sp}$	-	5432	2625	1352	660

Table 7: Comparison of the maximal local peak of the data storage (MB) needed by the hybrid and the direct method. "-" means that the result is not available because of the memory requirement.

4 Concluding remarks

In this paper, we have investigated the numerical behaviour of an algebraic domain decomposition technique for the solution of augmented systems arising in three dimensional structural mechanics

problems representative of difficulties encountered in this application area. In order to avoid the possible singularities related to the splitting of the Lagrange multipliers equations we propose a first solution that can surely be improved. In particular, other partitioning strategies would deserve to be studied and investigated. Some work in that direction would deserve to be undertaken prior the possible integration of this solution technique within the complete simulation chain. Other investigations should also be developed to study the effect of the stopping criterion threshold on the overall solution time when the linear solver would be embedded in the nonlinear Newton solver. Although a loose accuracy would certainly delay the nonlinear convergence some saving in computing time could be expected thanks to cheaper linear solves. In that context, we might also reuse the preconditioner from one nonlinear iteration to the next, specially close to the nonlinear convergence. Another possible source of gain for the sparse variant is a more sophisticated dropping strategy. Automatic tuning of the threshold parameter is also to be investigated as well as other preconditioning techniques that are less memory demanding [6].

References

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

[2] J.-F. Bourgat, R. Glowinski, P. Le Tallec, and M. Vidrascu. Variational formulation and algorithm for trace operator in domain decomposition calculations. In Tony Chan, Roland Glowinski, Jacques Périaux, and Olof Widlund, editors, *Domain Decomposition Methods*, pages 3–16, Philadelphia, PA, 1989. SIAM.

[3] 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.

[4] Y.-H. De Roeck and P. Le Tallec. Analysis and test of a local domain decomposition preconditioner. In Roland Glowinski, Yuri Kuznetsov, Gérard Meurant, Jacques Périaux, and Olof Widlund, editors, *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 112–128. SIAM, Philadelphia, PA, 1991.

[5] V. Frayssé, L. Giraud, S. Gratton, and J. Langou. Algorithm 842: A set of GMRES routines for real and complex arithmetics on high performance computers. *ACM Transactions on Mathematical Software*, 31(2):228–238, 2005. Preliminary version available as CERFACS TR/PA/03/3, Public domain software availabe on www.cerfacs.fr/algor/Softs.

[6] L. Giraud, A. Haidar, and Y. Saad. Sparse approximations of the Schur complement for parallel algebraic hybrid linear solvers in 3D. *Numerical Mathematics: Theory, Methods and Applications*, 3(3):276–294, 2010.

[7] L. Giraud, A. Haidar, and L. T. Watson. Parallel scalability study of hybrid preconditioners in three dimensions. *Parallel Computing*, 34:363–379, 2008.

[8] A. Haidar. *On the parallel scalability of hybrid solvers for large 3D problems.* Ph.D. dissertation, INPT, June 2008. TH/PA/08/57.

[9] G. Karypis and V. Kumar. METIS, unstructured graph partitioning and sparse matrix ordering system. version 2.0. Technical report, University of Minnesota, Department of Computer Science, Minneapolis, MN 55455, August 1995.

[10] Y. Saad. *Iterative Methods for Sparse Linear Systems.* SIAM, Philadelphia, 2003. Second edition.

[11] Y. Saad and M. H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comp.*, 7:856–869, 1986.

Centre de recherche INRIA Bordeaux – Sud Ouest
Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex (France)

Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399