A comparison of selected solvers for coupled FEM/BEM linear systems arising from discretization of aeroacoustic problems

Emmanuel Agullo, Marek Felšöci, Guillaume Sylvand

RESEARCH REPORT
N° 9412
June 2021
Project-Team HiePACS

ISSN 0249-6399 ISRN INRIA/RR--9412--FR+ENG

A comparison of selected solvers for coupled FEM/BEM linear systems arising from discretization of aeroacoustic problems

Emmanuel Agullo*, Marek Felšöci†, Guillaume Sylvand‡

Project-Team HiePACS

Research Report n° 9412 — June 2021 — 52 pages

Abstract: When discretization of an aeroacoustic physical model is based on the application of both the Finite Elements Method (FEM) and the Boundary Elements Method (BEM), this leads to coupled FEM/BEM linear systems combining sparse and dense parts. In this preliminary study, we compare a set of sparse and dense solvers applied on the solution of such type of linear systems with the aim to identify the best performing configurations of existing solvers.

Key-words: aeroacoustics, modelization, finite elements, boundary elements, solver comparison, coupled FEM/BEM linear systems, sparse and dense matrices

* Inria Bordeaux Sud-Ouest (emmanuel.agullo@inria.fr)
† Inria Bordeaux Sud-Ouest (marek.felsoci@inria.fr)
‡ Airbus Central R&T / Inria Bordeaux Sud-Ouest (guillaume.sylvand@airbus.com)

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Une comparaison de solveurs choisis pour la résolution de systèmes linéaires couplés FEM/BEM résultant de la discrétisation de problèmes aéroacoustiques

Résumé : Lorsque la discrétisation d'un modèle aéroacoustique repose sur l'application d'à la fois la méthodes des éléments fini (FEM) et de la méthode des éléments finis de frontière (BEM), celle-ci conduit à des systèmes linéaires couplés FEM/BEM ayant des parties creuses ainsi que des parties denses. Dans cette étude préliminaire, nous faisons la comparaison d'un ensemble de solveurs creux et denses appliqués à la résolution de ce type de systèmes linéaires dans le but d'identifier les configurations les plus performantes des solveurs existants.

Mots-clés : aéroacoustique, modélisation, éléments finis, éléments finis de frontière, comparaison de solveurs, systèmes linéaires couplés FEM/BEM, matrices creuses et denses

Contents

1 Introduction

Performing physical experiments while studying complex phenomena may be an expensive and sometimes also a dangerous matter. In many scientific fields, numerical simulations have been accompanying physical experiments for decades and they are widely used in the aircraft industry too. For example, while testing and validating prototypes, various electromagnetic and aeroacoustic problems come into play. In the industrial context of Airbus, we are particularly interested in models of aeroacoustic phenomena such as the propagation of acoustic waves generated by an aircraft on take-off, landing and taxiing. Such physical models are typically expressed using Partial Differential Equations (PDE) and they are likely to involve concepts that can not be modelled on computers, e. g. equations of integral functions. Therefore, prior to computing the model numerically, an approximation of its original physical expression must be made over a limited domain using an appropriate discretization technique. For the problems we are interested in, we rely on a combination of the Finite Elements Method (FEM) and the Boundary Elements Method (BEM).

Discretization allows one to compute a system of linear equations approximating the original continuous model. The higher the frequency, the more accurate the approximation and the bigger the resulting linear system. Such a transformation ultimately leads to coupled FEM/BEM systems with coefficient matrices having a dense part corresponding to surface mesh discretization with BEM as well as sparse parts corresponding to volume mesh discretization using FEM and the interactions between the surface and the volume meshes.

To solve these linear systems, we build on novel efficient solving methods, algorithms and data structures optimized in such a way as to take advantage of massively parallel hardware configurations and the characteristics of the input linear system as much as possible. Throughout the report, we discuss also the positionning of our work relatively to the previous contributions to the domain of the solution methods for coupled FEM/BEM or more generally coupled sparse and dense linear systems [33, 41, 79, 39, 70, 56, 63, 49, 54, 53, 57, 66, 48, 25, 70, 26, 73].

Within this preliminary study, we want to identify the best performing configurations of selected sparse and dense linear system solvers (see Section 7.3) on a test case (see Section 7.2) yielding coupled FEM/BEM systems with characteristics close enough to those arising from real-life aeroacoustic models.

In the first place, we focus separately on linear systems emerging from BEM discretization and leading to purely dense matrices (see Section 7.5), then on linear systems emerging from FEM discretization and leading to purely sparse matrices (see Section 7.6). Beginning to evaluate dense and sparse solvers apart should allow us to detect their performance specificities for a better understanding of their behavior when committed to solve coupled FEM/BEM linear systems. In other words, after isolating performance issues linked to the individual solution of FEM and BEM systems, we are more likely to identify the best performing coupled solver configurations (see Section 7.7) while focusing only on the parameters specific to the latter.

Before considering substantially larger systems in an MPI distributed parallel environment on multiple computational nodes, the goal here is to evaluate the performance of the existing solver implementations on a single node.

The present research report is organized as follows. In Section 2 we describe the construction of the continuous model of the aeroacoustic problem we are interested in. In Section 3, we detail the discretization of the original physical model. In Section 4, we present different numerical approaches of solving linear systems. In Section 5, we explain the solution process of a coupled FEM/BEM linear system. In Section 6, we discuss concerns on accuracy of computed solutions. Section 7 features our experimental study and result analysis. We conclude in Section 8.

2 Continuous model

We focus on a particular kind of aeroacoustic phenomenon and study the propagation of sound waves produced by aircraft's engine at take-off (see Figure 1). It can be seen as an acoustic wave propagation problem and expressed using Partial Differential Equations (PDE).

FIGURE 1: Airbus A319-112 at take-off producing a jet of exhaust gas traversed by sound waves emitted by the engine [72].

There are multiple media aeroacoustic waves may interfere with. The model takes into consideration aircraft's and engine's surface as well as exhaust gas and air flows (see Figure 2). On the other hand, some aspects such as engine interior or retro-action sound waves are omitted to prevent the model from being unnecessarily too complex.

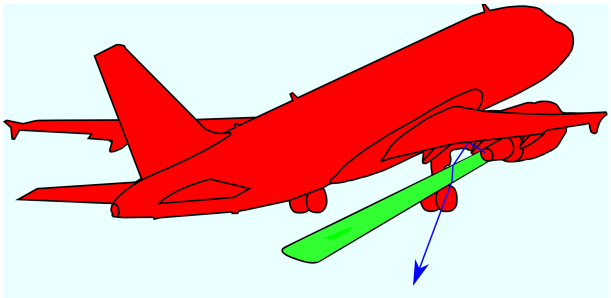

FIGURE 2: An aeroacoustic wave (dark blue arrow) produced by the aircraft's engine, reflected on the wing and traversing the jet of exhaust gas and the ambient air (light blue). The red part represents the media considered as homogeneous and the green part represents the media considered as heterogeneous.

Eventually, the model allows us to study how acoustic waves produced by a jet engine propagate throughout a heterogeneous environment. Although some waves may only go through, for example, ambient air considered to be a homogeneous medium, others may traverse environments with varying parameters. In our case, the homogeneous Helmholtz PDE is used to model the domains considered as homogeneous, such as the aircraft's surface (see Figure 3). On the contrary, the jet of exhaust gas produced by the aircraft's engine must not be considered as such. Both temperature and velocity vary inside of the jet flow depending on the engine operation condition (see Figure 3). This is represented thanks to an anisotropic second-order PDE.

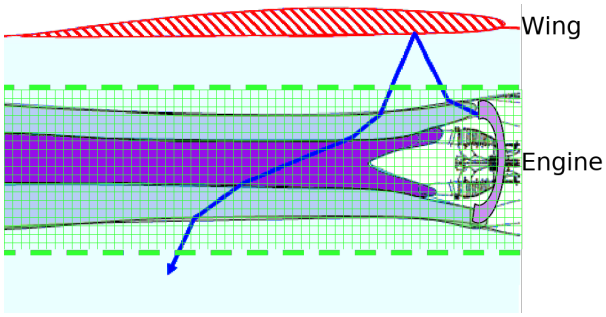

FIGURE 3: Zoom on Figure 2 with a detailed view of the different types of media a sound wave (blue arrow) may interfere with. Dashed lines delimit the green-striped domain considered heterogeneous including cold jet flow (dark blue) passing through the ducted fan, slower hot jet flow (violet) passing through the engine's core as well as a part of ambient air (light blue) in order for discretization to yield a simpler cylindrical 3D form. The aircraft's surface (boundaries of the red-striped domain) is considered homogeneous.

3 Discrete model

Heterogeneous media are discretized using the Finite Elements Method (FEM) [24, 36, 65, 80] and the Boundary Elements Method (BEM) [23, 69, 76] is applied on media considered as homogeneous (see figures 2 and 3).

The idea behind FEM is to partition the target domain into smaller parts referred to as *finite elements* such as tetrahedrons. Eventually, to each of these elements, a local linear equation is attributed which approximates the original PDE on the selected part of the domain. The set of these linear equations forms the final linear system that approximates the original continuous model to the extent of the domain of interest. In our case, we model the jet flow beginning at the rear part of the aircraft's engine up to a limited distance from the engine and for a concrete frequency of the problem.

On the other hand, the goal of BEM is to solve a given problem only on the *boundary values* of the considered domain. As the media it is applied on is considered to be homogeneous, the method does not mesh over the entire domain, only over its surface. By definition, discretization of a three-dimensional problem using BEM makes it two-dimensional.

Both methods may be used together thanks to a coupling technique [30, 29] allowing one to relate the unknowns of linear systems resulting from both methods with each other if, as in our case, the problems expose identical properties on the interface between FEM and BEM domains.

Figure 4 shows examples of a 3D cylinder volume mesh resulting from a FEM discretization and a 2D surface mesh resulting from a BEM discretization, used on the outer surface of the volume mesh as well.

The global linear system (see Equation 1) resulting from a FEM/BEM coupling features three main categories of unknowns: x_1 associated with the formulation of the FEM discretization on the three-dimensional domain corresponding to the jet exhaust flow, x_2 associated with the coupling where unknowns are shared between the BEM-discretized domain and the boundaries of the FEM-discretized domain on the exterior surface of the jet exhaust flow, x_3 associated with the formulation of the BEM discretization on the two-dimensional domain corresponding to the surface of the aircraft.

$$\begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & A_{23} \\ 0 & A_{32} & A_{33} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{1}$$

In practice, x_2 and x_3 can be grouped to one unique surface mesh s related to the BEM discretization while x_1 is associated to the volume mesh v resulting from the FEM discretization. Consequently, in the simplified formulation of the system, x_2 and x_3 from Equation 1 become x_s and x_1 becomes x_v:

$$\begin{bmatrix} A_{vv} & A_{vs} \\ A_{sv} & A_{ss} \end{bmatrix} \times \begin{bmatrix} x_v \\ x_s \end{bmatrix} = \begin{bmatrix} b_v \\ b_s \end{bmatrix}. \tag{2}$$

The coefficient matrix A becomes a 2×2 matrix where A_{vv} is symmetric and represents the action of the volume part on itself, A_{sv} represents the action of the volume part on the exterior surface, A_{vs} is the transpose of A_{sv} representing the action of the exterior surface on the volume part and A_{ss} is symmetric and represents the action of the exterior surface on itself.

FIGURE 4: Example of a FEM/BEM discretization. The red mesh corresponds to the BEM discretization of the surface of the aircraft as well as the outer surface of the green 3D cylinder volume mesh representing the FEM discretization of the jet of exhaust gas produced by the aircraft's engine.

4 Solution of linear systems

In Section 3, we have discussed the linear systems arising from FEM, BEM and the coupling of both methods allowing us to discretize the continuous model presented in Section 2 and yielding the target linear system that we need to solve (see Equation 2).

We consider two classes of linear systems, sparse and dense. In sparse linear systems, the coefficient matrix is mostly composed of zeros and has only a few non-zero entries. Conversely, in dense linear systems, the coefficient matrix has no or hardly any null entry. There is no exact condition for to determine whether a matrix should be considered sparse or not. Although, in practice, a matrix can be characterized as sparse when the computation involving the matrix can take advantage of the high count of zero entries and their locations [68].

In our case, the linear systems resulting from FEM are termed sparse as each element of the corresponding mesh interacts only with its direct neighbors. The linear systems resulting from BEM are termed dense as each element of the associated mesh interacts with all the others. Eventually, in the target system resulting from the coupling of both methods (see Equation 2), A_{vv} is a large and sparse submatrix, A_{ss} is a smaller dense submatrix and A_{vs} and A_{sv} are sparse submatrices (see Figure 5).

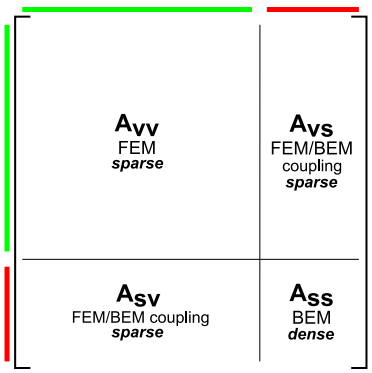

FIGURE 5: Internal dimensions and sparsity of the coefficient matrix A in the target linear system (see Equation 2).

Regardless its sparsity, a linear system may be solved using either a direct or an iterative approach. The goal of iterative methods is to find a good approximation of the solution. A sequence of terms x_k is calculated, in which the term x_i is based on previous ones and approximates the solution x. The idea is to converge as much as possible to the exact solution while preserving the performance advantage over direct methods. For solving coupled FEM/BEM systems, numerous iterative approaches has been proposed in the last decades [73, 26, 63, 49, 54, 53, 57, 66, 48, 25, 70]. 3However, in order to not broaden the scope of our work too much and keep a reasonable number of parameters we operate on within this study, we focus exclusively on direct methods. This section gives an overview of selected direct solving techniques used in case of dense and sparse systems prior to explaining how they are applied on the solution of the target coupled linear system in Section 5.

4.1 Direct methods

Direct methods generally rely on an initial decomposition or factorization of the coefficient matrix into a product of matrices making the linear system easier to solve. For example, the LU factorization algorithm decomposes the coefficient matrix into a lower triangular matrix L and an upper triangular matrix U transforming an initial system of form $Ax = b$ to $LUx = b$ which

can be split into a couple of equations such as $Ly = b$ and $Ux = y$. These triangular systems may be eventually solved using the so-called forward substitution for the first equation and the so-called backward substitution for the second one.

The LU factorization first appeared in the work of Polish astronomer Tadeusz Banachiewicz from 1938 [22, 71]. It is also applicable on non-symmetric matrices unlike the earlier Cholesky decomposition, named after André-Louis Cholesky and published in 1924 after his death [32, 71], working only with symmetric matrices. These procedures may be considered as matrix forms of Gaussian elimination, described by Leonhard Euler as the most natural way of solving simultaneous linear equations [37] and the traces of which have been discovered in multiple ancient sources [45]. Other factorization methods appeared in the last decades such as the LL^T or LDL^T factorizations applicable to any square and symmetric matrix decomposing the latter into the product of two or three matrices respectively where L is a lower triangular matrix, D a diagonal matrix and L^T the transpose of L [27].

4.1.1 Hierarchical matrices

An important drawback of direct methods is a high consumption of computing resources. Hierarchical or $\mathcal{H}$-matrices [46] represent an optimized way to store matrices. It is an algebraic hierarchical structure consisting of either full-rank or compressed low-rank sub-matrices (see Figure 6). Being a major advantage of the format, the complexity of a matrix-matrix product of two dense matrices coming, for example, out of a BEM discretization may be significantly lowered from $\mathcal{O}(n^3)$ to $\mathcal{O}(nlog(n))$ [47] thanks to data compression. Despite the compression, the implied loss of computing precision remains acceptable in our context.

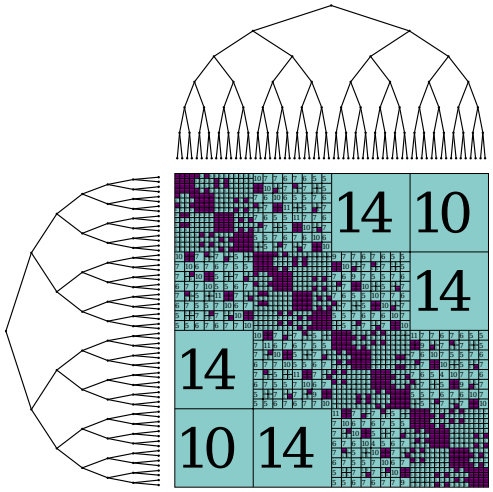

FIGURE 6: An $\mathcal{H}$-matrix where violet sub-matrices are stored as full-rank and light blue ones are in the compressed low-rank format with the inner values indicating the rank.

4.1.2 Sparse systems

Moreover, in the case of sparse systems, we can take advantage of the high number of zero entries in the coefficient matrix and reduce the time and the memory footprint of the computation by preventing the storage of null coefficients. However, factorization of a sparse matrix is likely to cause some initially zero entries to become non-zero. This effect is called fill-in [43] (see Figure 7) and it is directly related to the process of Gaussian elimination. Depending on the order of the unknowns in the linear system, the fill-in may be more or less important. There are multiple

reordering techniques to minimize the fill-in [18, 59, 62, 67, 42]. The problem is generally NP-complete [77] so these techniques have to rely on heuristics. The aim is to search for an optimal rearrangement of the equations in the linear system so as to minimize the appearance of new non-zero entries during the factorization.

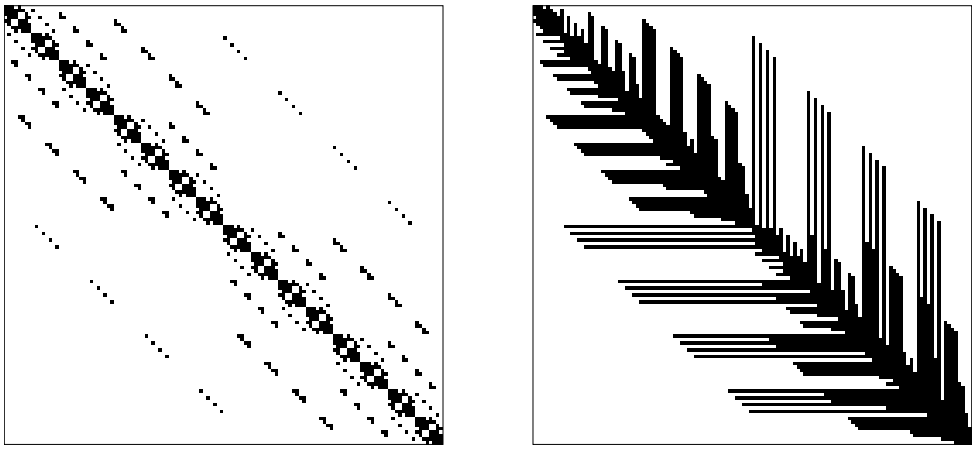

FIGURE 7: Example of a sparse matrix before factorization (left) and after an LU factorization (right) demonstrating the fill-in effect. Black squares represent non-zero entries.

The goal being to store only the non-zero elements of a sparse matrix, it is useful to estimate the memory that has to be allocated for the factorization phase. This is the role of symbolic factorization [58] which is a simulation of the actual factorization based on the zero/non-zero pattern of the matrix and allows one to locate where the fill-in appears. Eventually, the direct solution of a sparse system usually features more computation steps in addition to factorization and solve: reordering of unknowns and symbolic factorization.

4.2 Selected direct solvers

In this section, we give an overview of the dense and sparse direct solvers assessed within this study (see also Section 7.3).

4.2.1 SPIDO

SPIDO [61, 15] is a dense direct solver developed and maintained at Airbus. To solve linear systems, it relies either on LU or LDL^T factorization. When the linear system is too large to fit in memory, SPIDO can perform computations out-of-core. It splits the coefficient matrix evenly into multiple submatrices called blocks. The currently unused blocks are temporarily stored on disk (out of the core memory) to reduce memory consumption.

When an out-of-core block is loaded into memory for computation, it is further divided into smaller parts called processor blocks that may be distributed and processed in parallel using MPI (see Figure 8). Also, an additional thread-level parallelism relying on OpenMP can be enabled to potentially further speed-up the computation.

Block sizes may be either set manually or determined automatically by the solver itself based on the size of the problem, the count of available computation nodes and threads as well as the amount of free memory.

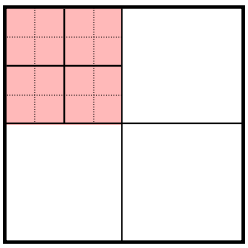

FIGURE 8: A coefficient matrix split into 4 out-of-core blocks. Currently processed block (in red) is divided into 4 processor blocks counting 4 thread blocks each.

4.2.2 HMAT

HMAT [5] is another solver coming from Airbus. It implements the hierarchical $\mathcal{H}$-matrix structure (see Section 4.1.1) and associated hierarchical variants of LU and LL^T factorization and solve algorithms. Being parallelized using the StarPU run-time [21], HMAT is implicitly capable of storing currently unused data out-of-core [75]. Distributed MPI parallelism is also available.

Unlike SPIDO, it works with sparse matrices too. To reduce fill-in (see Section 4.1.2) resulting from factorization, HMAT provides an implementation prototype of the Nested Dissection algorithm [42]. Improvements has been made in this direction [38], although further work is required to include these into the mainstream version of HMAT.

4.2.3 MUMPS

MUMPS [19] is a multifrontal [34, 35, 60] sparse direct solver relying on LU or LDL^T factorization providing also Schur complement (see Section 5.1) computation routines. MUMPS features both distributed and thread-level parallelism relying on MPI and OpenMP respectively. In addition, the solver implements a data compression capability through a single-level $\mathcal{H}$-matrix scheme [17]. The matrix is split into smaller sub-matrices of equal size and an admissibility criterion decides whether a sub-matrix should be compressed using the low-rank format or stored in full-rank.

5 Solution of FEM/BEM systems

In Section 3, we have defined the coupled FEM/BEM linear system we want to solve. In Section 4, we have then explained that the coefficient matrix of the target system is composed of a dense part and multiple sparse parts. Followed a presentation of different techniques to solve linear systems depending on their sparsity as well as a selection of dense and sparse solvers. In this section, we focus on the application of these on the solution of the target coupled system (see Equation 2) within the solver framework of Airbus.

The latter implements multiple direct as well as iterative approaches. The target system can be either solved as is or it can be simplified first by taking advantage of the Schur complement to reduce the whole problem on boundaries [78]. When the Schur complement is not involved, currently the only solution is to use an iterative method. Because iterative methods are out of the scope of this study, in this section, we focus exclusively on the approach relying on the Schur complement computation.

5.1 Schur complement

Let us remind the initial formulation of the target system with R_1 and R_2 denoting its first and its second row respectively:

$$\begin{matrix} R_1 \\ R_2 \end{matrix} \begin{bmatrix} A_{vv} & A_{vs} \\ A_{sv} & A_{ss} \end{bmatrix} \times \begin{bmatrix} x_v \\ x_s \end{bmatrix} = \begin{bmatrix} b_v \\ b_s \end{bmatrix}. \tag{3}$$

Based on the first row R_1 in Equation 3, we can express x_v as:

$$x_v = A_{vv}^{-1}(b_v - A_{vs}x_s). \tag{4}$$

Then, substituting x_v in the second row by Equation 4 yields in R_2 a reduced system where x_v does not appear any more. This operation corresponds to one step of Gaussian elimination, namely subtracting $A_{sv}A_{vv}^{-1}$ times R_1 from R_2:

$$\begin{matrix} R_1 \\ R_2 \leftarrow R_2 - A_{sv}A_{vv}^{-1} \times R_1 \end{matrix} \begin{bmatrix} A_{vv} & A_{vs} \\ 0 & A_{ss} - A_{sv}A_{vv}^{-1}A_{vs} \end{bmatrix} \times \begin{bmatrix} x_v \\ x_s \end{bmatrix} = \begin{bmatrix} b_v \\ b_s - A_{sv}A_{vv}^{-1}b_v \end{bmatrix}. \tag{5}$$

The expression $A_{ss} - A_{sv}A_{vv}^{-1}A_{vs}$ that appeared in R_2 in Equation 5 is often referred to as the Schur complement [78] noted S and associated with the partitioning, v and s, of the variables in the system:

$$\begin{bmatrix} A_{vv} & A_{vs} \\ 0 & S \end{bmatrix} \times \begin{bmatrix} x_v \\ x_s \end{bmatrix} = \begin{bmatrix} b_v \\ b_s - A_{sv}A_{vv}^{-1}b_v \end{bmatrix}. \tag{6}$$

In summary, we have to compute S such as

$$S = A_{ss} - A_{sv}A_{vv}^{-1}A_{vs} \tag{7}$$

and solve the reduced Schur complement system:

$$Sx_s = b_s - A_{sv}A_{vv}^{-1}b_v \tag{8}$$

corresponding to the second row of Equation 6 after the elimination of x_v. Once we have computed x_s, we use its value to determine x_v according to Equation 4.

Notice that, the Schur complement computation involves the inverse A_{vv}^{-1}. When solving the problem numerically, we factorize A_{vv} into a product of matrices making the system easier to solve (see Section 4.1) rather than actually performing the inverse of the matrix. This decomposition of A and the choice to eliminate x_v from the second row in Equation 3 allows us to take advantage of the sparsity of the sub-matrix A_{vv} during the solution process. On the contrary, eliminating x_s from the first row instead of the current choice would result in an important fill-in of A_{vv} where the Schur complement would have been computed. This way, we would not have preserved and exploited the sparsity of A_{vv} to accelerate the whole solution process.

5.2 Notes on symmetry and numerical computation of S

The discretization of the problem we want to solve leads to a linear system where coefficient matrix is symmetric (see Section 3). Working with symmetric matrices allows to reduce the amount of necessary storage space. There is no need to keep the coefficients above the diagonal (see Figure 9).

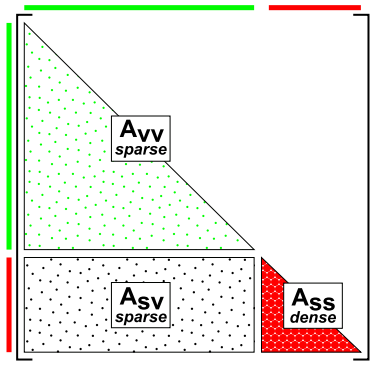

FIGURE 9: Because the coefficient matrix A of the linear system to solve (see Equation 3) is symmetric, we do not need to store the elements above the diagonal.

As A_{vv} is symmetric, during the Schur complement computation we rely on the LL^T factorization (see Section 4.1) to decompose the sub-matrix (see Section 5.1) into the product $L_{vv}L_{vv}^T$ where L_{vv} is a lower triangular matrix and L_{vv}^T its transpose. Therefore, in the formulation of the Schur complement S we can substitute A_{vv} by $L_{vv}L_{vv}^T$. Moreover, given that $A_{vs} = A_{sv}^T$, we can substitute A_{vs} by A_{sv}^T as well:

$$S = A_{ss} - A_{sv}(L_{vv}L_{vv}^T)^{-1}A_{sv}^T. \tag{9}$$

Developing Equation 9 yields

$$S = A_{ss} - (A_{sv}(L_{vv}^T)^{-1})(A_{sv}(L_{vv}^T)^{-1})^T \tag{10}$$

where $A_{sv}(L_{vv}^T)^{-1}$ corresponds to a triangular solve operation and $(A_{sv}(L_{vv}^T)^{-1})^T$ is implicitly known from $A_{sv}(L_{vv}^T)^{-1}$.

During the Schur complement computation, the sparse character of A_{vv} and A_{sv} can be preserved. Although, the factorization of A_{vv} and the subsequent solve operation are likely to introduce some fill-in (see Section 4.1.2) into both A_{vv} and A_{sv} (see Figure 10).

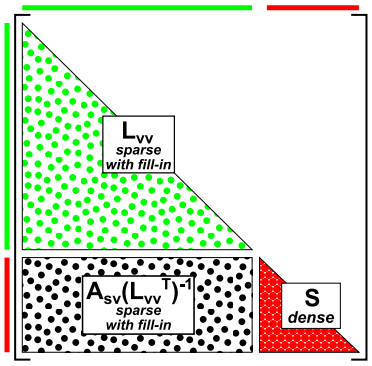

FIGURE 10: Contents and sparsity of the coefficient matrix A of the linear system to solve (see Equation 3) after the Schur complement computation phase (see Equation 7).

5.3 Single-stage implementations

The most straightforward way of solving the target linear system (see Equation 3) is to use a single solver taking the coefficient matrix A and the right-hand side on input and giving directly the final solution of the coupled linear system on output. Relying on a single solver could allow for a better control over the memory management and the usage of out-of-core (see Section 4.2) compared to the two-stage implementation (see Section 5.4). We distinguish three different implementations following this scheme.

5.3.1 Sparse-oblivious

The first possibility is to consider A as a single matrix without the partitioning into sub-matrices based on different levels of sparsity (see sections 3 and 4.1). This way, the matrix A is simply termed dense (see Figure 11).

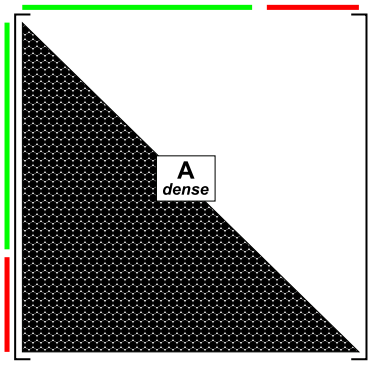

FIGURE 11: Structure of the coefficient matrix A of the linear system to solve (see Equation 3) in case of the sparse-oblivious implementation scheme.

This scheme does not allow us to benefit from the sparsity of the A_{vv} and A_{sv} sub-matrices which makes operations involving the coefficient matrix too costly both in terms of storage space and computing resources consumption. On the other hand, it is possible to achieve a non-negligible performance advantage thanks to the hierarchical low-rank format (see Section 4.1.1).

5.3.2 Sparse-aware

The ideal approach would be to rely on a single solver aware of the different levels of sparsity of A and capable of applying appropriate sparse or dense operations (see Figure 9) while preserving

the sparse structure of the coefficient matrix throughout the computation (see Figure 10). Such an implementation scheme does not exist in the Airbus solver framework yet. However, a progress has been made in this direction recently [38] (see also Section 4.2.2).

5.3.3 Partially sparse-aware

In the absence of the ideal implementation (see Section 5.3.2), it is possible to split the matrix A into three separate matrices following the partitioning presented in Section 3 and partially preserve the original sparse structure of A.

In this case, the A_{vv} sub-matrix can benefit from a sparse factorization. Even if during the operations related to the Schur complement computation (see Section 5.2) we do not consider the sparse structure of A_{sv}, dense data can still be compressed using the hierarchical low-rank format (see Section 4.1.1). The potential performance advantage associated with the latter further supports the interest in implementing this scheme.

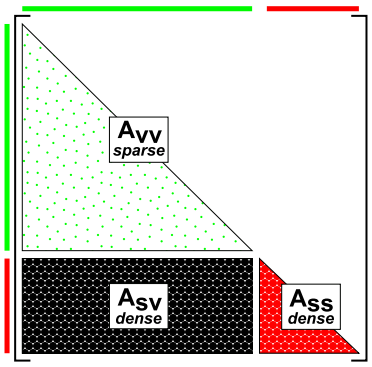

FIGURE 12: Structure of the coefficient matrix A of the linear system to solve (see Equation 3) in case of the partially sparse-aware implementation scheme.

5.4 Two-stage implementations

An alternative way of proceeding is to use a sparse direct solver to compute the Schur complement at first and then, call a dense solver to factorize the latter and solve the reduced system (see Section 5.1). The advantage of this approach is the possibility to use some well optimized community solvers.

Some sparse direct solvers, such as MUMPS and PaStiX [50], expose in their Application Program Interface (API) a Schur complement computation functionality (for MUMPS see option `ICNTL(19)` in [31] and for PaStiX see Section 4.4.4 in [52]). Provided with A_{vv}, A_{vs} and A_{sv} they can perform the operations associated with the computation (see Equation 5) and directly yield S in a dense matrix as a result. Afterwards, the solution of the reduced system (see Equation 8) is done using a dense solver which performs the factorization of S and solves the system using the so-called backward and forward substitutions on the resulting triangular systems (see Section 4.1).

On the other hand, even if the sparse solver provides the out-of-core feature (for MUMPS see option `ICNTL(22)` in [31] and for PaStiX see Section 1.8 in [52]), the Schur complement matrix has still to be stored in the computer's memory (for MUMPS see option `ICNTL(19)` in [31] and for PaStiX see Section 4.4.4 in [52]). When we consider memory constraints, this quickly becomes a limitation for solving large systems. We have to rely on an algorithm allowing us to store the Schur complement matrix out-of-core and keep using the solvers from the state of the art (see

Section 4.2). The idea is to compute the Schur complement block-wise and load into memory only the block necessary for the current computation step. There are two variants of such an algorithm implemented in the Airbus solver framework, multi-solve and multi-factorization.

5.4.1 Multi-solve

In the multi-solve approach, we compute the Schur complement S by blocks of columns (see Figure 13). Let us note $A_{sv_i}^T$ and A_{ss_i} blocks of n_c columns of A_{sv}^T and A_{ss} respectively. Then, based on Equation 9, S_i is a block of n_c columns of S defined as

$$S_i = A_{ss_i} - \underbrace{A_{sv} \overbrace{(L_{vv}L_{vv}^T)^{-1}A_{sv_i}^T}^{Y}}_{Z}. \tag{11}$$

The corresponding Algorithm 1 begins by the LL^T factorization of A_{vv} (line 3). Then, a loop (line 4) iterates over the blocks of A_{sv}^T and A_{ss} to compute all the blocks S_i of the Schur complement S while overwriting A_{ss}. Note that, the first step of this computation (line 5) corresponds to a triangular solve performed by the sparse solver in order to determine $Y = (L_{vv}L_{vv}^T)^{-1}A_{sv_i}^T$. It is possible to take advantage of the sparsity of the right-hand side matrix $A_{sv_i}^T$ during the solve operation [16] (for MUMPS see option `ICNTL(20)` in [31]). Nevertheless, independently from the sparsity of the input right-hand side matrix, the resulting Y is a **dense matrix** (for MUMPS see Section 5.13 in [31] and for PaStiX see Section 4.1.1 in [52]) with only a few zero entries. Also, by storing the block $A_{sv_i}^T$ explicitly, we **break the symmetry** of the global system (see Figure 13).

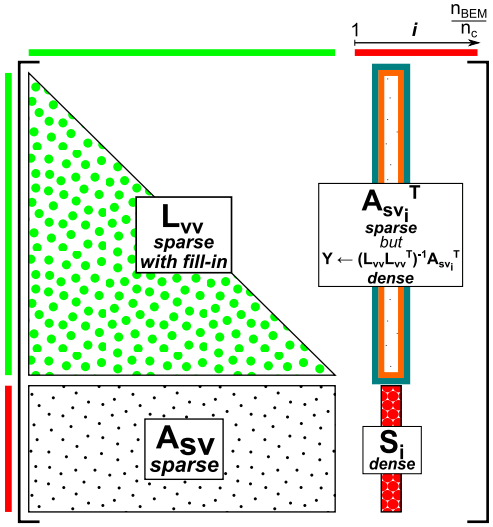

FIGURE 13: Graphical representation of the Schur computation loop in the multi-solve variant (see Algorithm 1). By storing $A_{sv_i}^T$ explicitly we **break the symmetry** of the global system. Moreover, the resulting Y is a **dense matrix** with only few zero entries.

Eventually, we use a dense solver to factorize A_{ss} containing the Schur complement and solve the simplified system (see Equation 8).

The number of columns n_c in S_i depends on n_{BEM}, the count of unknowns associated with the formulation of BEM in the global linear system, and the amount of available computer memory. If n_c is too small, a great number of solve operations are performed inside of the main loop, which may degrade the performance. On the other hand, if it is too high, data (such as Y and Z) may not fit in memory.

Algorithm 1: Multi-solve algorithm for computing the Schur complement S based on Equation 11 and solving the target system (see Equation 3).

1 **Function** `Multi-solve`(A, b)**:**
  ▷ Using a sparse solver:
2 $A_{vv} \leftarrow LL^T$-`Factorization`(A_{vv})
3 **for** $i = 1$ **to** n_{BEM}/n_c **do**
4 $Y \leftarrow$ `TriangularSolve`$(A_{vv}, A_{sv_i}^T)$ ▷ Using the i^{th} block of columns of A_{sv}^T as right-hand side.
5 $Z \leftarrow A_{sv} \times Y$ ▷ GEMM
6 $A_{ss_i} \leftarrow A_{ss_i} - Z$ ▷ AXPY
7 $b_v \leftarrow$ `TriangularSolve`(A_{vv}, b_v)
  ▷ Using a dense solver:
8 $A_{ss} \leftarrow LL^T$-`Factorization`(A_{ss})
9 $x_s \leftarrow$ `TriangularSolve`$(A_{ss}, b_s - A_{sv}b_v)$

5.4.2 Multi-factorization

In this variant, the A_{ss} matrix receiving the Schur complement S is split into square blocks of equal size (see Figure 14). Let us note A_{sv_i} a block of n rows of A_{sv} and $A_{sv_j}^T$ a block of n columns of A_{sv}^T. Then, based on Equation 9, S_{ij} is a block of n rows and columns of S such as:

$$S_{ij} = A_{ss_{ij}} - A_{sv_i}(L_{vv}U_{vv})^{-1}A_{sv_j}^T. \tag{12}$$

In the nested loop (line 2) of Algorithm 2, we construct a temporary matrix W from A_{vv}, A_{sv_i} and $A_{sv_j}^T$. Then, we call the `CreateSchurComplement` function on W (line 3) to compute the corresponding block S_{ij} of S relying on the Schur complement computation functionality provided by the sparse solver. Note that W is not symmetric except when $i = j$. Therefore, we store the copy of A_{vv} in W including its upper triangular part and use the LU factorization to decompose this submatrix within the `CreateSchurComplement` function. Here, it is the explicit storage of the upper triangular part of A_{vv} and the $A_{sv_j}^T$ block in W which **breaks the symmetry** of the A_{vv} submatrix and of the global system itself. Moreover, constructing W yields a **duplicated storage** of A_{vv}, A_{sv_i} and $A_{sv_j}^T$ (see Figure 14).

Finally, as in the case of the multi-solve variant (see Section 5.4.1), we use a dense solver to factorize the A_{ss} containing the Schur complement and solve the simplified system (see Equation 8).

Here, n_b represents the count of blocks per row or column of A_{ss}. The `SchurComplement` function operating on the temporary matrix W implies a re-factorization of the A_{vv} sub-matrix contained in W at each iteration, yet it does not change during the computation. Consequently, the more blocks the A_{ss} sub-matrix is split into, the higher the number of superfluous factorizations of A_{vv}. We shall see in Section 7.7, that a high number of factorization dramatically increases the computation time of this implementation variant compared to multi-solve. On the other hand, too small values of n_c in the case of the multi-solve algorithm imply a high number of solve operations. Despite being considerably less expensive than factorizations, a sufficiently high number of solves may make the multi-solve method perform worse than multi-factorization.

We highlight the fact that the methods we have described are both designed to compute the same thing, but differ only in the way they make use of the available API provided by the solvers of the state of the art (see Section 4.2).

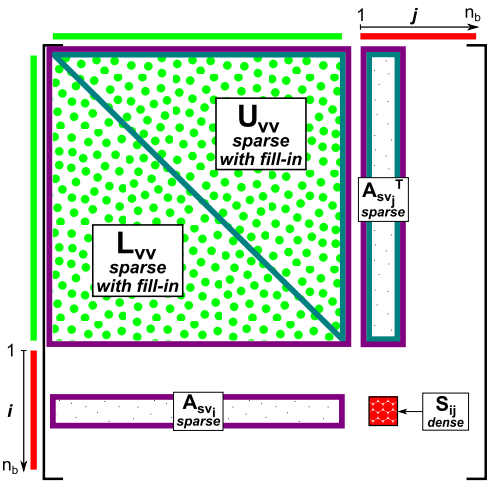

FIGURE 14: Graphical representation of the Schur computation loop in the multi-factorization variant (see Algorithm 1). The construction of W requires a **duplicated storage** of A_{vv}, A_{sv_i} and $A_{sv_j}^T$ and the explicit storage of the upper triangular part of A_{vv} and the $A_{sv_j}^T$ block in W **breaks the symmetry** of A_{vv} and of the global system itself.

Algorithm 2: Multi-factorization algorithm for computing the Schur complement S based on Equation 12 and solving the target system (see Equation 3).

Function `Multi-factorization`(A, b)**:**
- ▷ USING A SPARSE SOLVER:
- **for** $i = 1$ **to** n_b **do**
  - **for** $j = 1$ **to** n_b **do**
    - $W \leftarrow \begin{bmatrix} A_{vv} & A_{sv_j}^T \\ A_{sv_i} & 0 \end{bmatrix}$
    - $A_{ss_{ij}} \leftarrow A_{ss_{ij}} +$ `SchurComplement`(W)
- $A_{vv} \leftarrow LL^T$-`Factorization`(A_{vv})
- $b_v \leftarrow$ `TriangularSolve`(A_{vv}, b_v)
- ▷ USING A DENSE SOLVER:
- $A_{ss} \leftarrow LL^T$-`Factorization`(A_{ss})
- $x_s \leftarrow$ `TriangularSolve`$(A_{ss}, b_s - A_{sv}b_v)$

5.5 Positioning regarding the related work

To the best of our knowledge, the approaches proposed in the literature for solving coupled FEM/BEM or more generally coupled sparse and dense linear systems based on direct methods, as considered here, make the assumption that the factors fit in memory [33, 41, 79, 39, 70]. For example, in [33], the authors target systems where the dense part takes up to 80 GB although they evaluate the proposed implementation on a smaller data set using up to four computational nodes with 64 GB of RAM. In [41], the author presents benchmark results exclusively for the dense BEM-discretized part counting at most 1,536 boundary elements. Moreover, in [70], the parameters of the largest test cases considred are explicitly adapted in such a way as to fit entirely in memory.

On the other hand, the implementation schemes for solving the target coupled FEM/BEM linear system (see Equation 3) we propose in sections 5.3 and 5.4 has been specifically designed with support for out-of-core computation. Our ultimate goal is to be able to process very large coupled systems, e.g. counting up to 10^9 FEM-related and 10^6 BEM-related unknowns (see Section 3).

5.6 Proposed implementation schemes

In sections 5.3 and 5.4, we have described various implementation schemes for solving the target coupled FEM/BEM linear system (see Equation 3) numerically through the computation of the Schur complement (see sections 5.1 and 5.2). Here, we list the schemes implemented in the Airbus solver framework.

5.6.1 Single-stage implementations

The single-stage implementations (see Section 5.3) rely on a single solver for the entire solution process. Within the Airbus solver framework, the **sparse-oblivious** scheme (see Section 5.3.1) is implemented with HMAT (see Section 4.2.2) using a unique $\mathcal{H}$-Matrix constructed over the combined surface and volume meshes. Then, the **partially sparse-aware** scheme (see Section 5.3.3) is also implemented with HMAT using three separate $\mathcal{H}$-Matrices for A_{vv}, A_{sv} and A_{ss}. In this case, we rely on HMAT as a sparse solver for the Schur complement computation operations (see Section 5.2) but also as a dense solver for the final solution of the reduced system (see Section 5.1).

5.6.2 Two-stage implementations

On the other hand, the two-stage implementations (see Section 5.4) use a coupling of a sparse and a dense solver. Within the Airbus solver framework, the **multi-solve** scheme (see Section 5.4.1) is implemented for use with MUMPS (see Section 4.2.3) or PaStiX as a sparse solver and SPIDO (see Section 4.2.1) as a dense solver. Then, the **multi-factorization** scheme (see Section 5.4.2) is implemented for use with MUMPS as a sparse solver and SPIDO or HMAT as a dense solver.

6 Solution accuracy

The quality of the numerical solution of a linear system depends on various factors. On one hand, we have the errors due to the approximation of the concepts of continuous mathematics such as

real and complex numbers on computers working in finite precision arithmetics. On the other hand, the design of the algorithms involved in the computation comes into play. For example, in many cases, we can opt for trading-off an acceptable level of accuracy of the solution against a significant decrease of computation time and memory consumption. Eventually, it is important to be able to evaluate the accuracy of the numerical solution which is only an approximation of the exact solution to the system.

6.1 Forward error

The forward error is an approach to measure the accuracy of a numerical approximation of the exact solution to a given linear system [51]. In this section, let $Ax = b$ be a linear system, where A is the coefficient matrix, x the unknown solution vector and b the right-hand side vector. Following [51], we note $\hat{x}$ a numerically computed approximation of the exact solution x.

6.1.1 Definitions

The absolute forward error E_{abs} of $\hat{x}$ is defined as the difference between the exact solution and its approximation:

$$E_{abs}(\hat{x}) = ||\hat{x} - x||.$$

Then, the relative forward error of $\hat{x}$, referred to as E_{rel}, corresponds to:

$$E_{rel}(\hat{x}) = ||\hat{x} - x||/||x||. \tag{13}$$

6.1.2 Computation

In practice, the exact solution x is not known and therefore, we have to rely on an estimation of $E_{rel}(\hat{x})$. However, in order to assess a numerical solver, we can make use of an artificial test case allowing us to actually compute the relative forward error. We give ourselves A and x as well as a right-hand side b such that:

$$b = Ax.$$

Then, assuming that the computation of $b = Ax$ was exact, we solve $A\hat{x} = b$ using a numerical method:

$$\hat{x} = A^{-1}b$$

which involves the factorization of A and the solve operation performing the so-called backward and forward substitutions on the resulting triangular systems (see Section 4). As both x and $\hat{x}$ are eventually known, we are able to determine $E_{rel}(\hat{x})$ (see Equation 13).

6.2 Trade-off on accuracy

A machine working in finite precision arithmetic can be used to simulate an arbitrary high precision, which means that theoretically, the accuracy of the solution is not limited by the machine precision [51]. However, within this study, we do not consider such a simulation.

Therefore, we assume that the accuracy of a numerically computed solution using a direct method (see Section 4) is proportional to the precision of the target machine provided that the problem is not ill-conditioned. According to [51, 74], the problem is considered ill-conditioned when the condition number of the matrix A is high. This means that inaccuracies in A or b may have a great impact and significantly worsen the accuracy of the solution approximation. The machine precision, noted u, is defined as the difference between 1.0 and the next greater floating-point value [51]. When using a 64 bits IEEE floating-point representation, u is approximately $2^{-53} \approx 1.11 \times 10^{-16}$ [44, 20], assuming u rounded to the nearest representable number.

Computing in high precision may be too consuming in terms of time and memory consumption. With the aim to reduce the cost of the computation, solvers providing some kind of data compression can be instrumented to compute an approximate solution $\hat{x}$ of a system $Ax = b$ in a lower precision. Eventually, the relative forward error of $\hat{x}$ (see Section 6.1.1) is higher but the accuracy of the result typically remains acceptable in our context, although there is no absolute gurantee of achieving such a result, especially in the case of very ill-conditionned problems [51].

The MUMPS and HMAT solvers rely on the block low-rank compression (see sections 4.2.3 and 4.2.2) and both expose a threshold parameter referred to as ϵ. During the compression, all the values that are smaller than this low-rank compression threshold are discarded, i.e. they are rounded to zero.

7 Experimental study

In Section 3, we have presented the coupled FEM/BEM linear system we want to solve. Then, in sections 4 and 5, we have explained that the solution process of such a linear system involves operations on both sparse and dense matrices. Moreover, there are multiple implementation schemes for solving the system numerically. We can rely either on a single solver capable of applying both sparse or dense operations depending on the underlying matrix structure (see Section 5.3) or on the coupling of a sparse and a dense solver (see Section 5.4).

The goal of this study is to identify the best performing configurations. To this end, the first step we take is to benchmark the solvers of the state of the art (see Section 7.3) separately on dense linear systems arising from a BEM discretization and then on sparse linear systems arising from a FEM discretization. Eventually, being aware of the potential impact of each solver on the performance of an implementation scheme for solving the target coupled FEM/BEM system, we benchmark the existing implementation schemes in the Airbus solver framework (see Section 5.6).

This section is organized as follows: in Section 7.1, we introduce the concept of building a literate and reproducible experimental environment; in Section 7.2, we present the test case we rely on; in Section 7.3, we detail the solvers and the metrics we are evaluating; in Section 7.4, we describe the experimental environment of the study; in sections 7.5 and 7.6, we present the benchmark results of the selected dense and sparse direct solvers on purely BEM and purely FEM systems respectively; in Section 7.7, we provide the comparison of the existing implementation schemes for solving coupled FEM/BEM systems.

7.1 Literate and reproducible environment

With the aim of keeping the experimental environment of the study reproducible, we manage the associated software framework with the GNU Guix transactional package manager [4]. However, we are confronted to two major limitations in our effort. On one hand, we rely on some non-free software packages without the access to their source code. On the other hand, the source of the

Airbus software we work on is proprietary and can not be disclosed to the public. We address these issues further in [14].

Moreover, relying on the principles of literate programming [55], we provide a full documentation on the construction process of the experimental environment as well as on the execution of benchmarks, the collection and the visualization of results in a dedicated technical report associated with this study [14].

7.2 Test case

We perform performance benchmarks on a short pipe (2 meters long and 4 meters wide) test case (see Figure 15) yielding linear systems close enough to those arising from real life models (see Figure 4) while sharing with the scientific community a reproducible example [12].

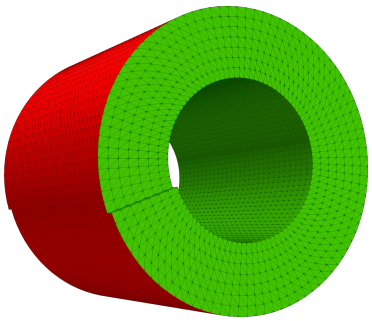

FIGURE 15: Short pipe test case (length: 2 m, radius: 4 m) with BEM surface mesh in red and FEM volume mesh in green.

The vertices of the pipe mesh represent the unknowns in the linear system coming out of the problem discretization (see Section 3). The volume part of the pipe (green portion in Figure 15) is discretized using FEM where each vertex interact solely with its immediate neighbors. The outer surface of the pipe (red portion in Figure 15) is discretized using BEM where each vertex interact with all the others. All the unknowns rely on the same mesh. The unknown count depends on the wavelength λ of the physical problem being simulated. The more there are vertices in the mesh, the more there are unknowns and the more the model is accurate (see Figure 16). For the experiments, we consider the wavelength parameter λ to be set such that there are 10 vertices per wavelength.

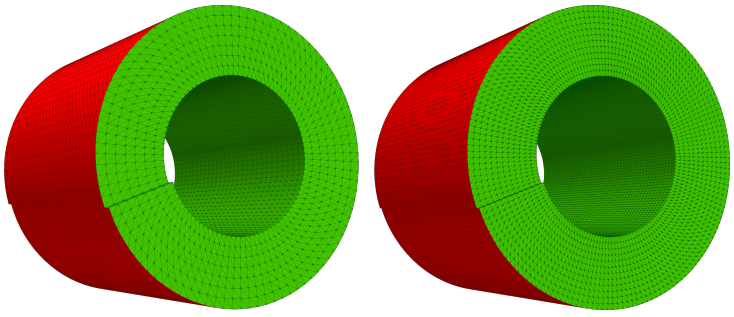

FIGURE 16: Comparison between a short pipe mesh counting 20,000 vertices (on the left) and a short pipe mesh counting 100,000 vertices (on the right) with BEM surface meshes in red and FEM volume meshes in green. Both are displayed using the same scale.

The elements of the coefficient matrix A of the linear system to solve (see Equation 3) are defined depending on the category of unknowns they are related to (see Section 3). For two unknowns i

and j associated with the BEM discretization, with $k = 2\pi/\lambda$ and r the distance between these unknowns, the interaction a_{ij} of the latter is defined as follows:

$$a_{ij} = \frac{e^{ikr}}{4\pi r}. \tag{14}$$

If r is 0, we set r to the size of an average mesh step divided by 2. Then, for two unknowns i and j associated with the FEM discretization (connected by one edge), the interaction a_{ij} of these two unknowns is defined as follows:

$$a_{ij} = \begin{cases} 0.05 \times (1 - \frac{i}{3}) & \text{if } i \neq j \\ 0.1 \times (1 - \frac{i}{3}) & \text{if } i = j. \end{cases} \tag{15}$$

Note that this is not the way the interactions are determined in real-life aeroacoustic models. Nevertheless, the definitions in equations 14 and 15 allow for a simple construction of coefficient matrices for testing purposes while providing values that are similar enough to real-life cases.

7.3 Scope of the study

For now, we limit our study to use of a single computation node and do not investigate here the behavior of the solvers in case of distributed memory parallelism. Table 1 lists the solvers being subject of the study (see also Section 4.2). On one hand, we make vary a set of solver parameters considering a fixed parallel configuration and observe the evolution of:

- **computation time**, measured in seconds [s],
- **relative forward error** $E_{rel}(\hat{x})$ of the solution approximation (see Section 6),
- **real or random access memory (RAM) peak usage** in gibibytes [GiB], representing the highest amount of data residing in the random access memory (RAM) of the machine during the computation,
- **disk space peak usage** in gibibytes [GiB], giving the highest amount of disk space used by the solver during the computation

according to:

- **unknown count** referred to as N and going up to 1,000,000 for BEM systems, up to 10,000,000 for FEM systems and up to 4,000,000 for coupled FEM/BEM systems,
- **low-rank compression threshold** referred to as ϵ (see Sections 6 and 6.2) and when using data compression set to either 1×10^{-3} or 1×10^{-6}.

We do not study the impact of:

- **right-hand side count** set by default to 50;
- **arithmetic type** always set to double complex;
- **out-of-core computation** [1] always disabled

Solver	Sparsity	Parallelization
SPIDO	dense	MPI + OpenMP
HMAT	dense and sparse	MPI + StarPU
MUMPS	sparse	MPI + OpenMP

TABLE 1: List of dense and sparse direct solvers being objects of the study precising the parallelization frameworks they rely on.

		Parallel configurations																	
MPI	Binding	node					socket					NUMA				core			
	# Processes	1					2					4				1	6	12	18	24
# OpenMP and MKL threads or StarPU workers		1	6	12	18	24	1	2	4	8	12	1	2	4	6	1				

TABLE 2: Complete list of parallel configurations evaluated for each solver.

On the other hand, we consider linear systems having a fixed count of unknowns and evaluate the scalability of different solvers for various parallel configurations.

For the first category of benchmarks, we always use 1 MPI process and 24 OpenMP and MKL threads and StarPU workers. In case of scalability benchmarks, we make vary the number of processor cores, noted c, made available for a given computation from 1 to 24. The parallel efficiency e, given in percentage, of a benchmark run is determined based on Equation 16 where T_s refers to the best sequential computation time and T_p to the parallel computation time.

$$e = \frac{T_s}{T_p \times c} \times 100 \tag{16}$$

We propose to evaluate the parallel configurations listed in Table 2 in order to study the impact of all the levels of parallelism the solvers are capable of. As we mentionned earlier, we currently limit ourselves to a single computational node.

7.4 Experimental environment

7.4.1 Computing platform

To run our experiments, we rely on the PlaFRIM platform [8]. It is an experimental testbed supported by Inria, CNRS (LaBRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine. From among various computational node families available on the platform, we use miriel nodes (see Figure 17) in particular. Each of these nodes is equipped with 2 Intel(R) Xeon(R) E5-2680 v3 processors making a total of 24 cores running at 2.5 GHz in 4 NUMA sub-nodes and 2 sockets, 126 GiB of random access memory (RAM) and 292 GiB of space for the `/tmp` file system on a local SATA Seagate ST9500620NS hard drive spinning at 7200 rpm. Resource allocation and job scheduling are managed with slurm [11]. We refer the reader to [10] to get further information on how the tool works.

[1] An out-of-core feature allows solvers to store on disk the parts of matrices not being actively used for computation in order to reduce system memory consumption and be able to process bigger problems with the same amount of memory.

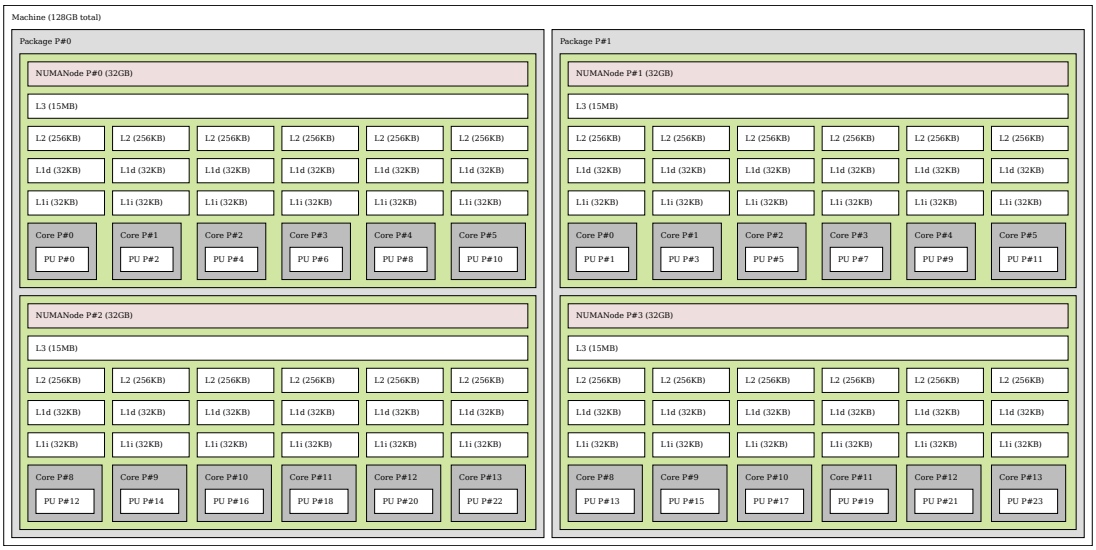

FIGURE 17: Hardware configuration scheme of miriel nodes determined using the Linux Shell command `hwloc-ls`.

7.4.2 Software

We use the open-source test_FEMBEM benchmark software suite [12] provided by Airbus relying on their proprietary HMAT, MPF and SCAB packages. The BLAS library is the Intel(R) Math Kernel Library (MKL) [6] and we compile using the OpenMPI [7] implementation of the Message Passing Interface (MPI). To process and visualize benchmark results, we use R [13] together with the ggplot2 [2] and StarVZ [64, 40] graphics libraries.

Table 3 lists commit numbers of the chosen Guix system derivation and the Airbus packages. Table 4 lists version numbers of other important packages in the environment for which we do not pick a specific version and we use the latest release provided in the selected Guix derivation. Note that it is possible to retrieve version and commit numbers of all the other software packages based on the Guix derivation commit number. We refer the reader to [14] for an exhaustive environment description and instructions on how to reproduce the experiments.

Software	Commit	Licensing
GNU/Guix [4]	1ac4959c	open-source
HMAT	81db5564	proprietary
HMAT-OSS [5]	4ef1b0ad	open-source
MPF	fec66d43	proprietary
SCAB	297fe52c	proprietary
gcvb [1]	40d88ba2	open-source

TABLE 3: Explicitly requested commit numbers of core packages in the software environment of the study.

Software	Version	Licensing
GNU C Compiler [3]	9.3.0	open-source
OpenMPI	4.0.3	open-source
Intel(R) Math Kernel Library	2019.1.144	proprietary
MUMPS [19]	5.2.1	open-source
StarPU [21]	1.3.5	open-source

TABLE 4: Version numbers of selected packages used in the software environment of the study based on the Guix derivation number in Table 3.

7.5 BEM systems

In the first place, we evaluate the performance of the direct solvers SPIDO and HMAT (see sections 4.2.1 and 4.2.2) on dense linear systems resulting from BEM discretization and counting from 25,000 up to 1,000,000 unknowns. We consider symmetric coefficient matrices and rely on LDL^T factorization in case of SPIDO and LL^T factorization in case of HMAT (see Section 4).

According to Figure 18, the computation times of SPIDO are significantly higher than those measured in the case of HMAT. For example, the factorization time of SPIDO on a system with 100,000 unknowns is comparable to the factorization time of HMAT on a system having 1,000,000 unknowns. Also, without out-of-core (see Section 7.3) and lacking any kind of data compression, SPIDO quickly approaches the 126 GiB memory limit of the miriel nodes (see Section 7.4.1) leaving it unable to process linear systems with 200,000 or more unknowns (see Figure 19).

These results show the advantages the hierarchical matrix structure and the low-rank compression capabilities (see Section 4.1.1) implemented in HMAT (see Section 4.2.2) represent for solving dense linear systems. We have observed both better computation times as well as lower memory footprint allowing the HMAT solver to process systems with up to 1,000,000 of unknowns when the low-rank compression threshold ϵ is set to 10^{-3}. Nevertheless, after tightening up the threshold to 10^{-6}, the factorization time increases more rapidly and the memory limit is reached sooner too. Ultimately, this makes the runs on systems counting more than 400,000 (see Figures 18 and 19) fail due to insufficient memory. The difference is naturally less noticeable for the solve phase having a considerably lower complexity compared to factorization.

Out-of-core being disabled, HMAT shows no disk space consumption (see Figure 19). One would expect the same for SPIDO, although the version of the solver used for the experiment stores an auxiliary matrix on disk regardless the out-of-core setting. This happens only when the LDL^T-factorization is used and it has been corrected by commit `e057b1d6` in the MPF package by the time of finishing this report.

Figure 20 validates the stability of the solvers for given problem sizes. The relative error of the solution approximations computed by HMAT exceeds the given low-rank compression thresholds ϵ (see Section 6). However, it is merely a small deviation and we consider it as non-significant in this case. Regarding SPIDO using no data compression, the relative error is naturally approaching the machine precision u (see Section 6.2).

When it comes to evaluate the scalability and the parallel efficiency of SPIDO and HMAT, we consider dense systems with 100,000 unknowns processed using multiple parallel configurations (see Section 7.3). In the case of HMAT, we consider also systems with 1,000,000 unknowns. Note that the low-rank compression threshold ϵ for HMAT has been set to 10^{-3}.

In Figure 21 we show the computation times and in Figure 22 we show the parallel efficiency of factorization and solve phases of SPIDO and HMAT for different counts of processor cores. SPIDO scales well for all of the assessed parallel configurations. Unlike in case of the solve

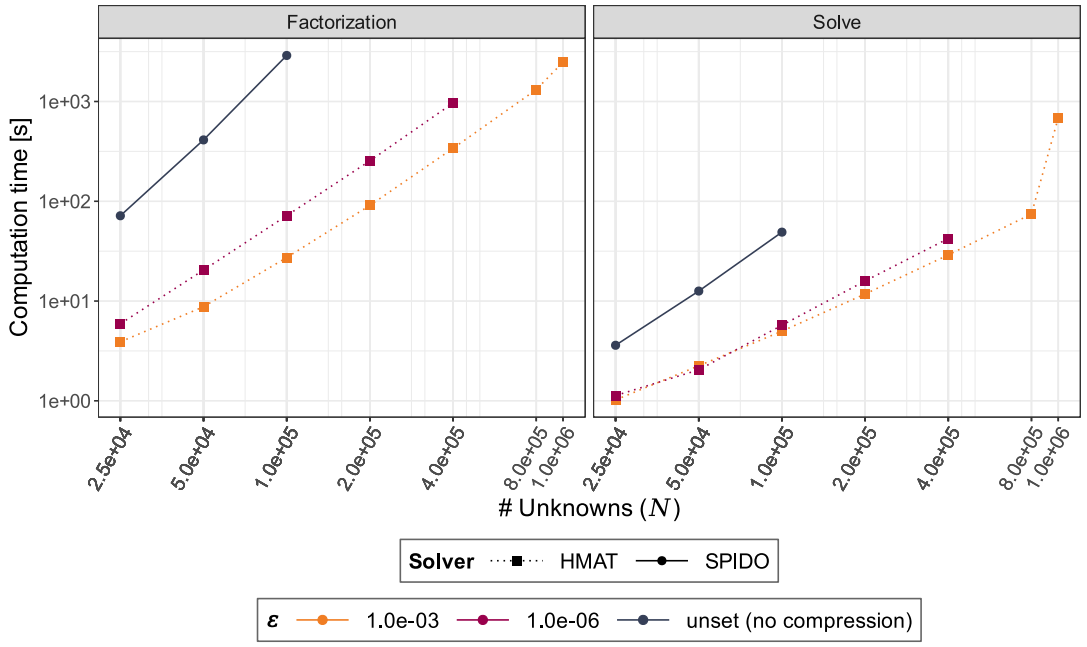

FIGURE 18: Computation times of SPIDO and HMAT on BEM systems run in parallel using 1 MPI process, 24 OpenMP and MKL threads or 24 StarPU workers on single miriel node for different low-rank compression thresholds ϵ (see Section 6.2) in case of HMAT and no ϵ set for SPIDO. Out-of-core has been disabled.

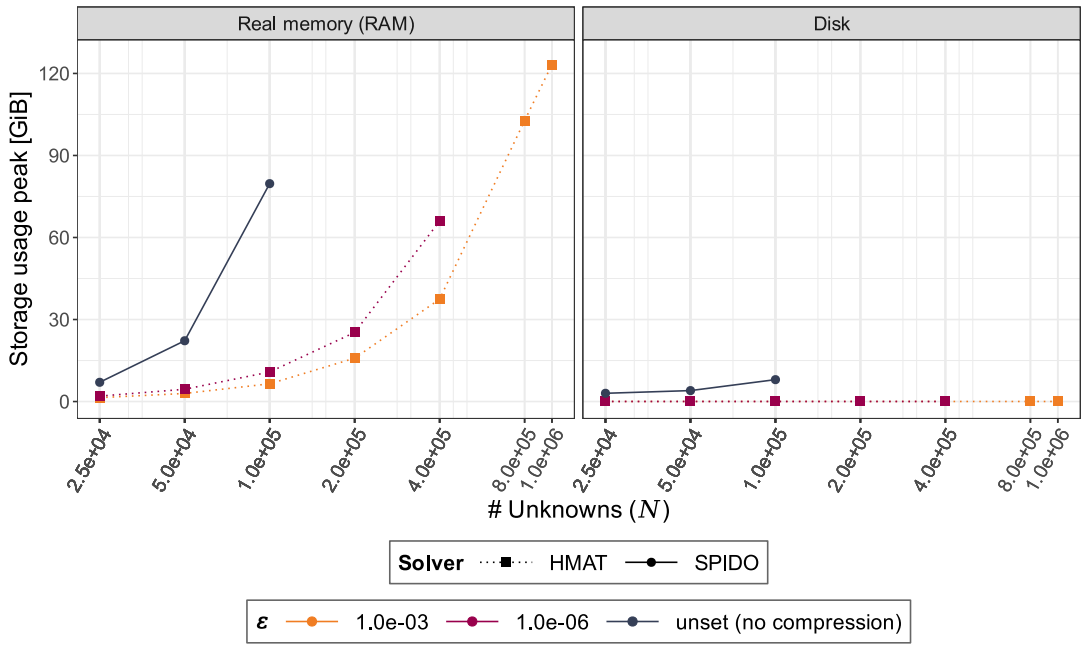

FIGURE 19: Real memory (RAM) and disk space usage peaks of SPIDO and HMAT on BEM systems run in parallel using 1 MPI process, 24 OpenMP and MKL threads or 24 StarPU workers on single miriel node for different low-rank compression thresholds ϵ (see Section 6.2) in case of HMAT and no ϵ set for SPIDO. Out-of-core has been disabled.

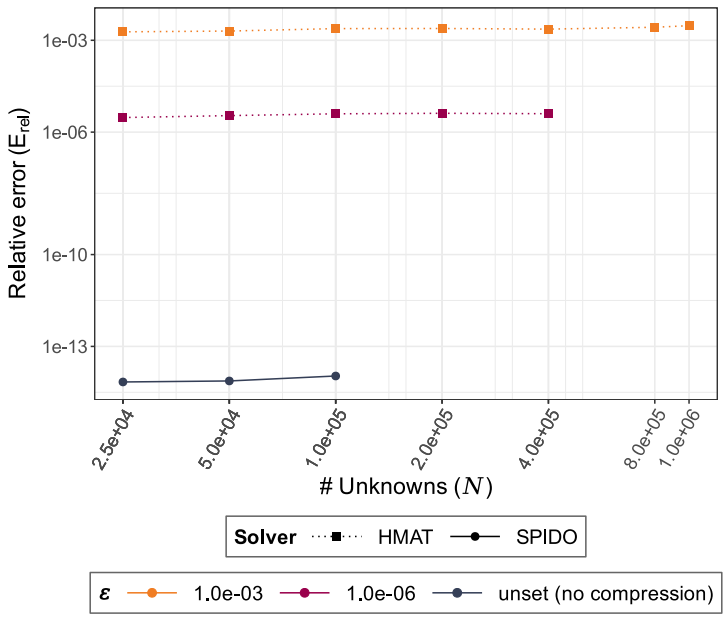

FIGURE 20: Relative solution error E_{rel} of SPIDO and HMAT on BEM systems run in parallel using 1 MPI process, 24 OpenMP and MKL threads or 24 StarPU workers on single miriel node for different low-rank compression thresholds ϵ (see Section 6.2) in case of HMAT and no ϵ set for SPIDO. Out-of-core has been disabled.

phase, there is no significant difference in computation times of the factorization phase. In terms of parallel efficiency, the two best performing configurations regarding the factorization phase are the one using only MPI parallelism and the one combining 4 MPI processes times 1 to 6 OpenMP and MKL threads yielding almost 90% parallel efficiency on 24 cores. The solve phase appears to scale best when relying exclusively on MPI parallelism yielding nearly 80% parallel efficiency on 24 cores.

Unlike SPIDO, HMAT scales well when relying exclusively on StarPU worker parallelism (and a single MPI process) for both factorization and solve phases. The parallel efficiency of factorization reaches approximately 60% on 24 cores considering a system with 100,000 unknowns and approximately 50% with 1,000,000 unknowns. The solve phase represents only a small part in the overall computation time which explains its low efficiency in a multi-threaded environment. The MPI parallelization of HMAT is not optimized for computations on a single node. Therefore, we dropped the scalability tests of HMAT for the parallel configurations involving more than one MPI process as the results would not be representative. A preliminary investigation revealed that the significant decrease in performance, when MPI processes are involved, comes from a poorly balanced workload. Regarding the compression algorithm in HMAT, it is difficult to determine an efficient static mapping for MPI processes. Yet in the explored model we rely on static mapping which explains the execution trace provided by StarPU and presented in Figure 23. The more MPI processes are involved in the computation, the more time StarPU workers spend in inactivity (sleeping).

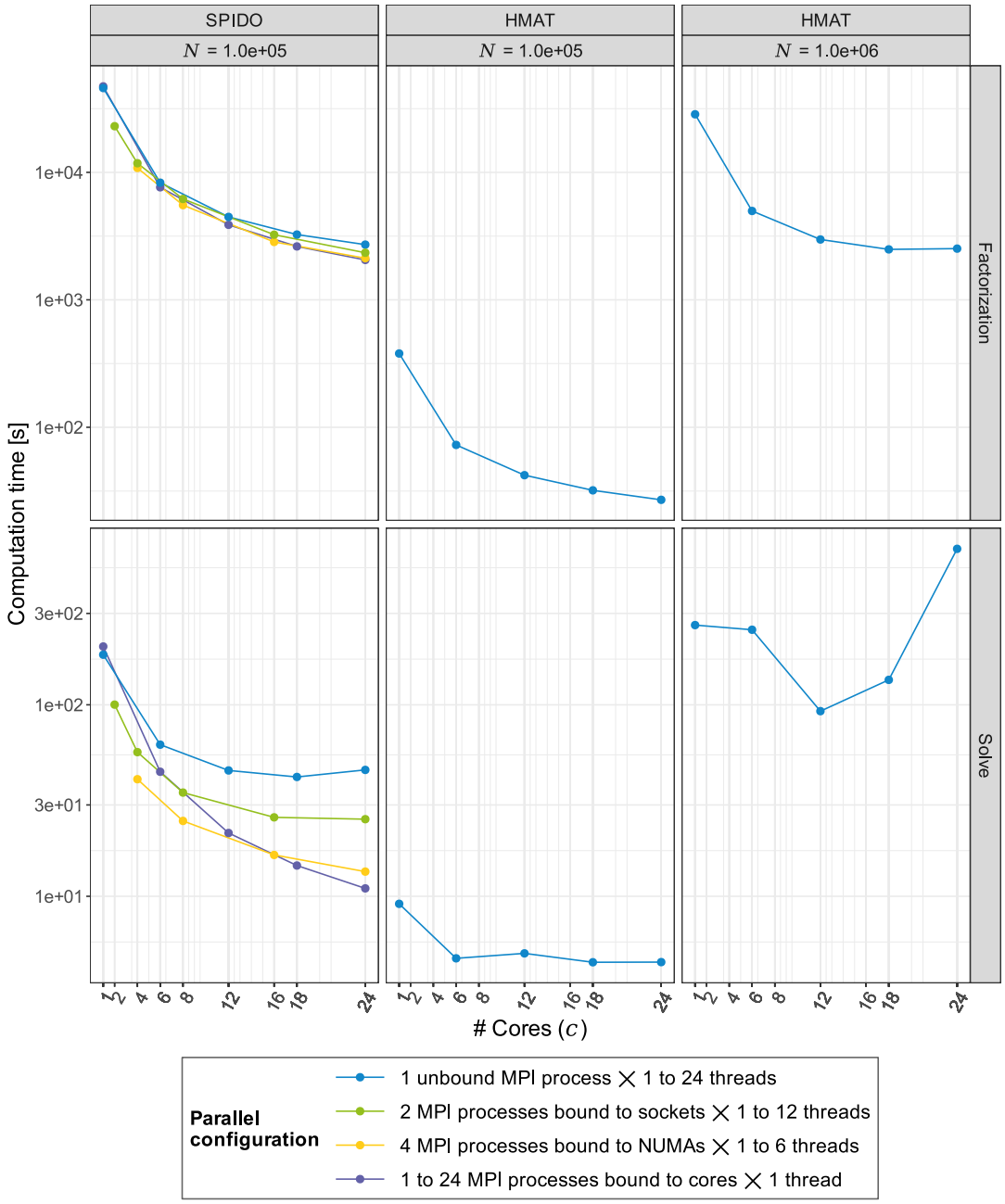

FIGURE 21: Scalability of SPIDO and HMAT on BEM systems run in 4 different kinds of parallel configurations using only 1 MPI process without binding times 1 to 24 OpenMP, MKL threads and StarPU workers or 2 MPI processes bound to sockets times 1 to 12 OpenMP, MKL threads and StarPU workers or 4 MPI processes bound to NUMA sub-nodes times 1 to 6 OpenMP, MKL threads and StarPU workers or 1 to 24 MPI processes bound to cores times 1 OpenMP, MKL thread and StarPU worker on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} in case of HMAT and no ϵ set for SPIDO. Out-of-core has been disabled.

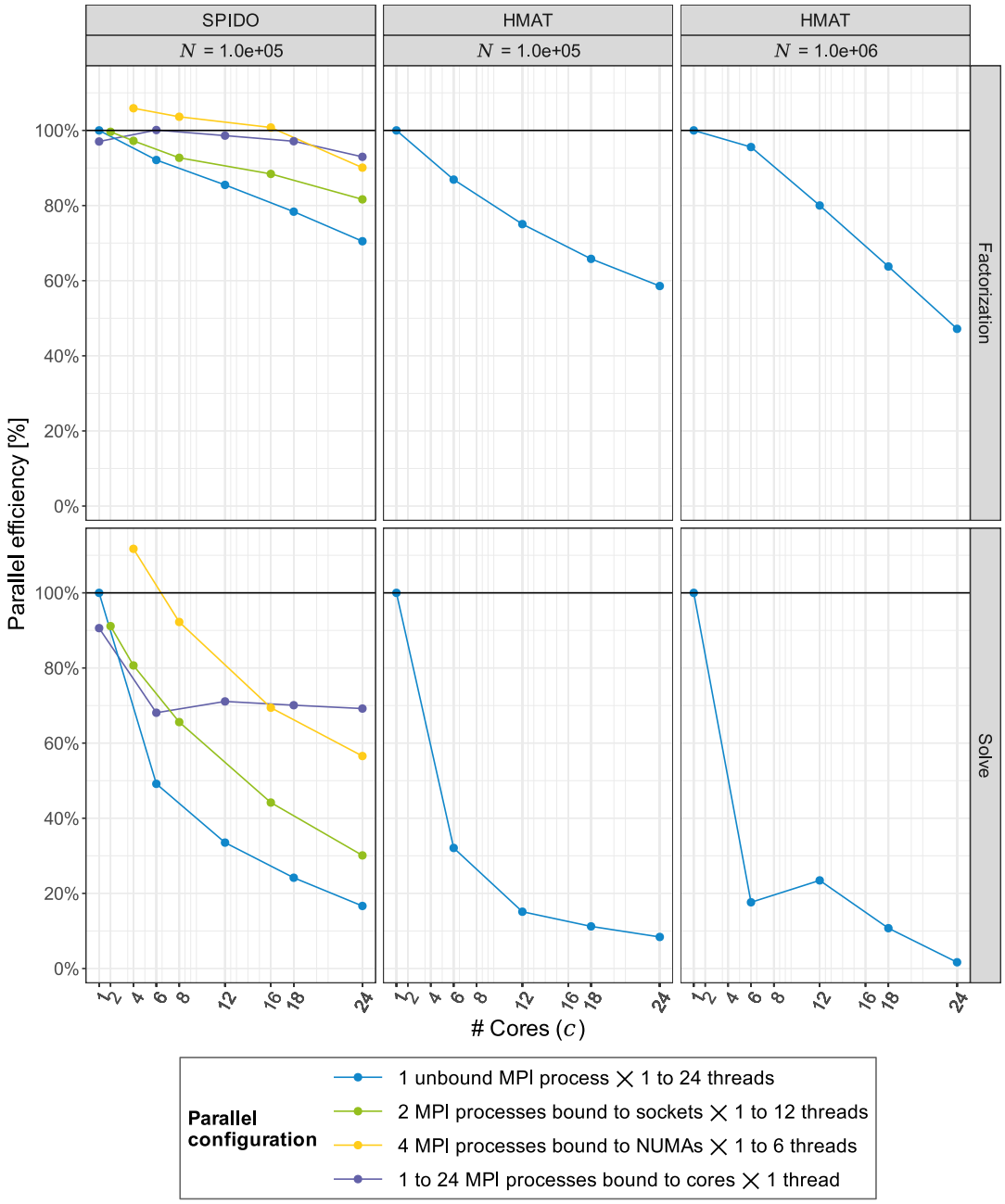

FIGURE 22: Parallel efficiency of SPIDO and HMAT on BEM systems run in 4 different kinds of parallel configurations using only 1 MPI process without binding times 1 to 24 OpenMP, MKL threads and StarPU workers or 2 MPI processes bound to sockets times 1 to 12 OpenMP, MKL threads and StarPU workers or 4 MPI processes bound to NUMA sub-nodes times 1 to 6 OpenMP, MKL threads and StarPU workers or 1 to 24 MPI processes bound to cores times 1 OpenMP, MKL thread and StarPU worker on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} in case of HMAT and no ϵ set for SPIDO. Out-of-core has been disabled.

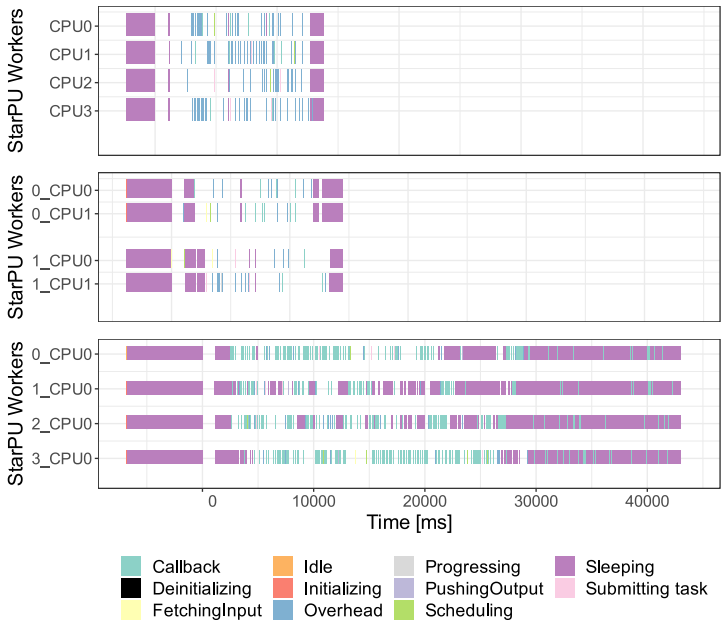

FIGURE 23: Graphical visualization of the execution traces provided by the StarPU runtime corresponding to the execution of HMAT on a BEM system having 25,000 unknowns using 1 MPI process times 4 StarPU workers, 2 MPI processes times 2 StarPU workers, 4 MPI processes with 1 StarPU worker per process on a single miriel node respectively. Blank spaces represent the time spent in computation. The colors represent the actions of the StarPU runtime. Out-of-core has been disabled.

7.6 FEM systems

The next step consists of comparing the performance of the direct solvers MUMPS and HMAT (see sections 4.2.3 and 4.2.2) on sparse linear systems resulting from FEM discretization and counting from 250,000 up to 10,000,000 unknowns. As in the case of BEM systems (see Section 7.5), coefficient matrices are symmetric and we rely on the LDL^T factorization in case of MUMPS and LL^T factorization in case of HMAT (see Section 4).

The hierarchical matrix structure as well as its implementation in the HMAT solver were primarily intended for dense matrices. Indeed, according to the results featured in figures 24 and 25, MUMPS seems to perform better both in terms of computation time and memory consumption. For illustration, HMAT is unable to process systems with 4,000,000 unknowns and more due to memory limitations while MUMPS can go up to 10,000,000 unknowns provided that ϵ is set to 10^{-3}.

Unlike in the case of HMAT, different low-rank compression thresholds does not significantly impact the performance of MUMPS in terms of computation time. On the other hand and as expected, the more precision we ask for, the more memory the solver consumes. This is true both for MUMPS and HMAT. Although, the difference in memory consumption of MUMPS is more significant only when the compression is completely disabled (see Figure 25). Finally, the nearly non-existent disk space consumption of both MUMPS and HMAT confirms that the out-of-core computation has been disabled.

HMAT features an implementation prototype of the Nested Dissection (ND) algorithm (see Section 4.2.2) which is a heuristic reordering technique allowing to reduce matrix fill-in resulting from factorization (see Section 4.1.2) with the aim to reduce solver's memory requirements and improve its performance. Due to current implementation limitations, HMAT is able to

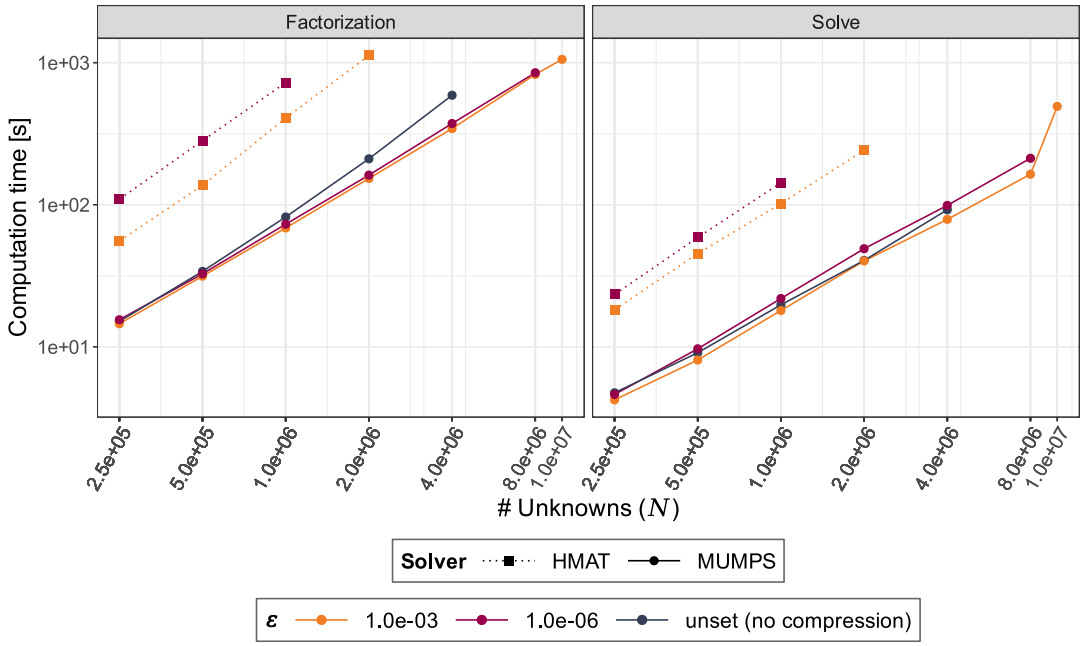

FIGURE 24: Computation time of MUMPS and HMAT on FEM systems run in parallel using 1 MPI process, 24 OpenMP, MKL threads and StarPU workers on single miriel node for different low-rank compression thresholds ϵ (see Section 6.2) and no ϵ for MUMPS when compression is disabled. Out-of-core has been disabled.

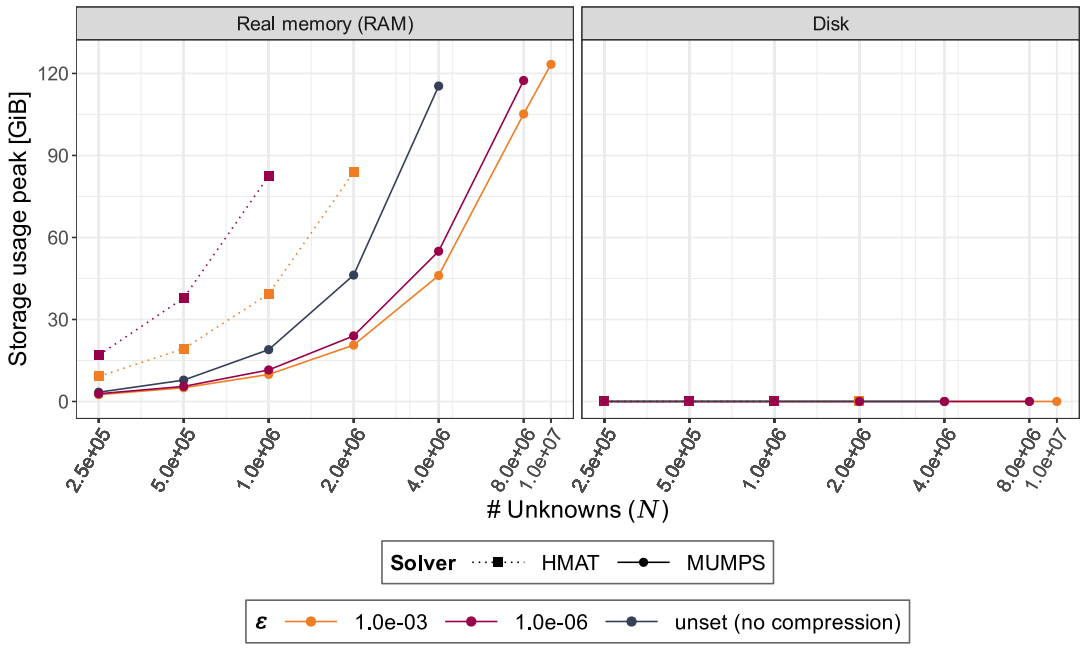

FIGURE 25: Real memory (RAM) and disk space usage peaks of MUMPS and HMAT on FEM systems run in parallel using 1 MPI process, 24 OpenMP, MKL threads and StarPU workers on single miriel node for different low-rank compression thresholds ϵ (see Section 6.2) and no ϵ set for MUMPS when compression is disabled. Out-of-core has been disabled.

apply the algorithm only on non-symmetric matrices. Moreover, a considerably higher memory consumption of the solver when using ND does not allow us to test cases with more than 250,000 unknowns on miriel nodes having 126 GiB of memory. Note that more advanced schemes for processing sparse matrices have been derived [38] but have not been integrated in the industrial framework and are thus not discussed in the present study.

In Figure 26, we compare HMAT with and without the use of ND on smaller sparse FEM systems using non-symmetric matrices and having from 25,000 up to 250,000 unknowns. We can not observe a significant difference in computation time among runs with different low-rank compression thresholds when using ND. Nevertheless, it appears to provide a clearly better performance in terms of computation time for cases with up to 100,000 unknowns. Although, starting from 200,000 unknowns, this trend seems to cease.

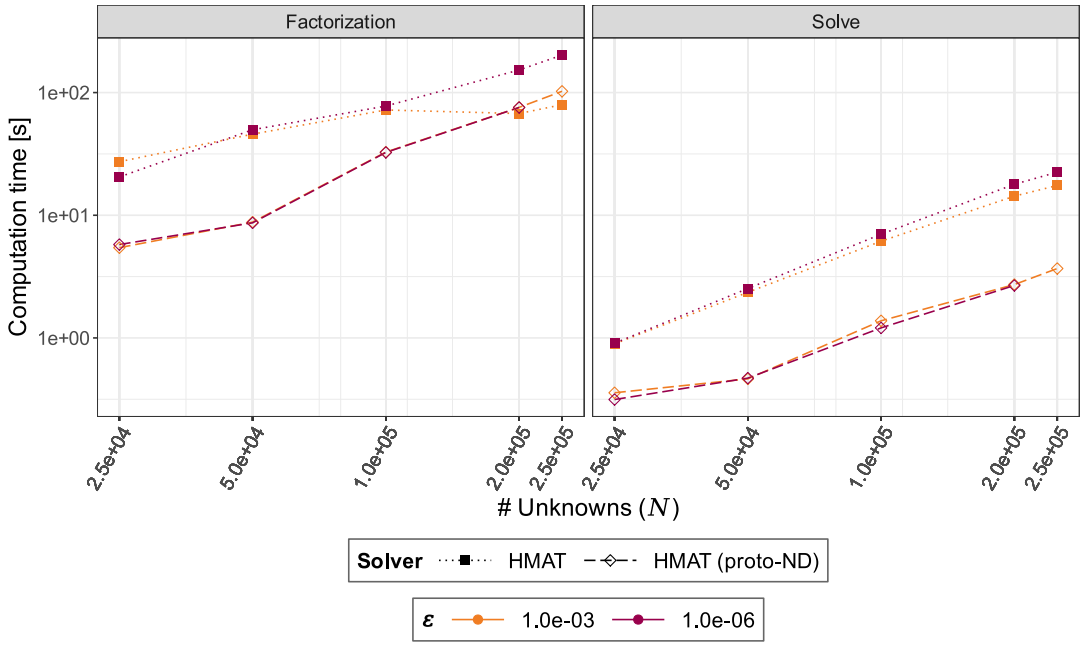

FIGURE 26: Computation time of HMAT on non-symmetric FEM systems with and without the Nested Dissection (ND) enabled, run in parallel using 1 MPI process, 24 StarPU workers on single miriel node for different low-rank compression thresholds ϵ (see Section 6.2). Out-of-core has been disabled.

Notice that, HMAT was run with ND only to produce the results featured in Figure 26. In all the other cases, ND was disabled.

According to Figure 27, HMAT yields a better solution accuracy on sparse FEM than on dense BEM systems (see Section 7.5). Although the relative error of the solution approximations computed by MUMPS when using compression is smaller than the corresponding low-rank compression thresholds, it is not as low as in the case of HMAT. When the compression is disabled, we naturally observe the relative error to approach the machine precision u (see Section 6.2). Eventually, the results also validates the stability of both solvers for given problem sizes.

To compare the scalability of MUMPS and HMAT on FEM systems while putting in action various parallel configurations (see Section 7.3), we consider systems having 2,000,000 unknowns for both MUMPS and HMAT. In the case of MUMPS, we consider systems with 4,000,000 unknowns as well. Although, the more MPI processes are involved in the computation, the more memory consumption increases (see Figure 28) which also explains that, the case with 4,000,000 unknowns relying exclusively on MPI parallelism causes a memory overflow using 16 cores and more. Finally, the low-rank compression threshold ϵ (see Section 6.2) has been set to

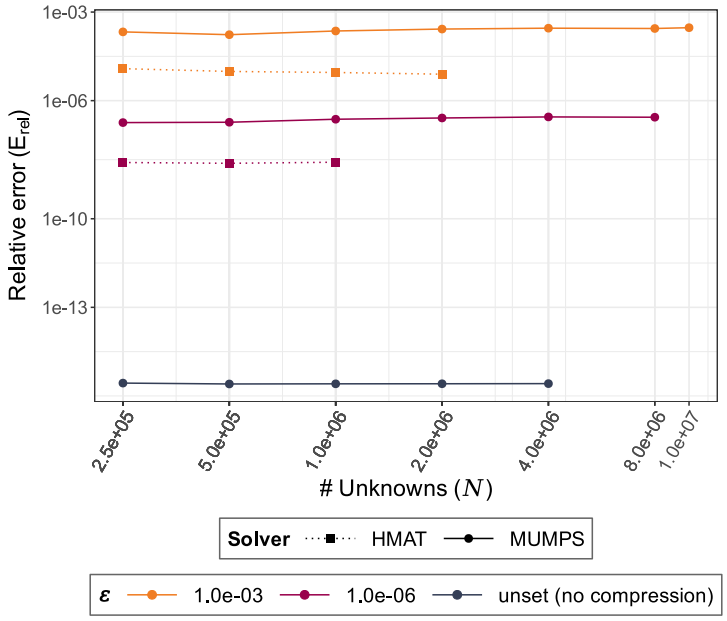

FIGURE 27: Relative solution error E_{rel} of MUMPS and HMAT on FEM systems run in parallel run using 1 MPI process, 24 OpenMP, MKL threads and StarPU workers on single miriel node for different low-rank compression thresholds ϵ (see Section 6.2) and no ϵ set for MUMPS when compression is disabled. Out-of-core has been disabled.

10^{-3} for all the runs.

In Figure 29 we show the computation times and in Figure 30 we show the parallel efficiency of factorization and solve phases of MUMPS and HMAT for different count of processor cores made available for the computation. Considered parallel configurations are described in Section 7.3. MUMPS scales well for all of the assessed parallel configurations with no significant difference in computation time of the most demanding factorization phase.

Regarding the parallel efficiency, the best performing MUMPS configuration for both problem sizes and the factorization phase seems to be the one using 4 MPI processes times 1 to 6 OpenMP and MKL threads yielding nearly 20% parallel efficiency on 24 cores. In the case of the solve phase, exclusively MPI parallelism yields the best results with 20% parallel efficiency on 24 cores considering a problem with 2,000,000 unknowns and nearly 30% on 12 cores considering a problem with 4,000,000 unknowns.

As it was the case for BEM systems (see Section 7.5), HMAT scales well only if we rely exclusively on local thread parallelism. The parallel efficiency of factorization reaches approximately 40% on 24 cores. We confirm the poor performance of HMAT when MPI parallelism is involved on a single node for sparse systems too. According to the execution trace provided by StarPU and presented in Figure 31, the more MPI processes are involved in the computation, the more time StarPU workers spend in inactivity (sleeping).

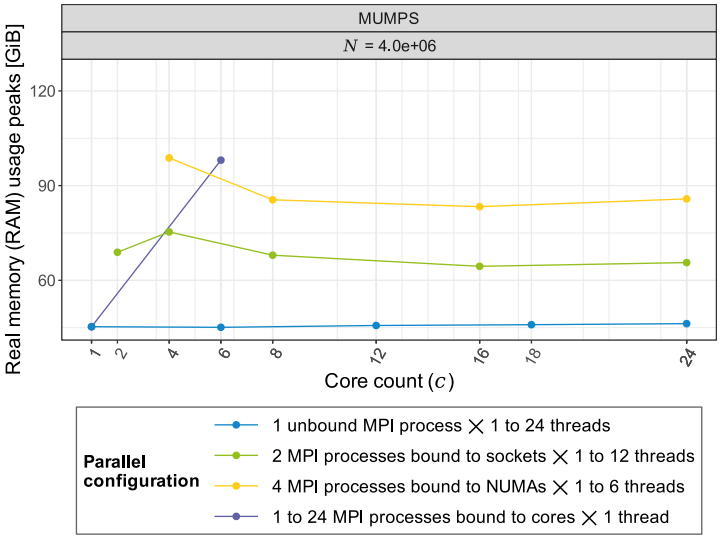

FIGURE 28: Real memory (RAM) usage peaks of MUMPS on FEM systems run in 4 different kinds of parallel configurations using only 1 MPI process without binding times 1 to 24 OpenMP, MKL threads and StarPU workers or 2 MPI processes bound to sockets times 1 to 12 OpenMP, MKL threads and StarPU workers or 4 MPI processes bound to NUMA sub-nodes times 1 to 6 OpenMP, MKL threads and StarPU workers or 1 to 24 MPI processes bound to cores times 1 OpenMP, MKL thread and StarPU worker on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} in case of HMAT and MUMPS when compression is enabled. Out-of-core has been disabled.

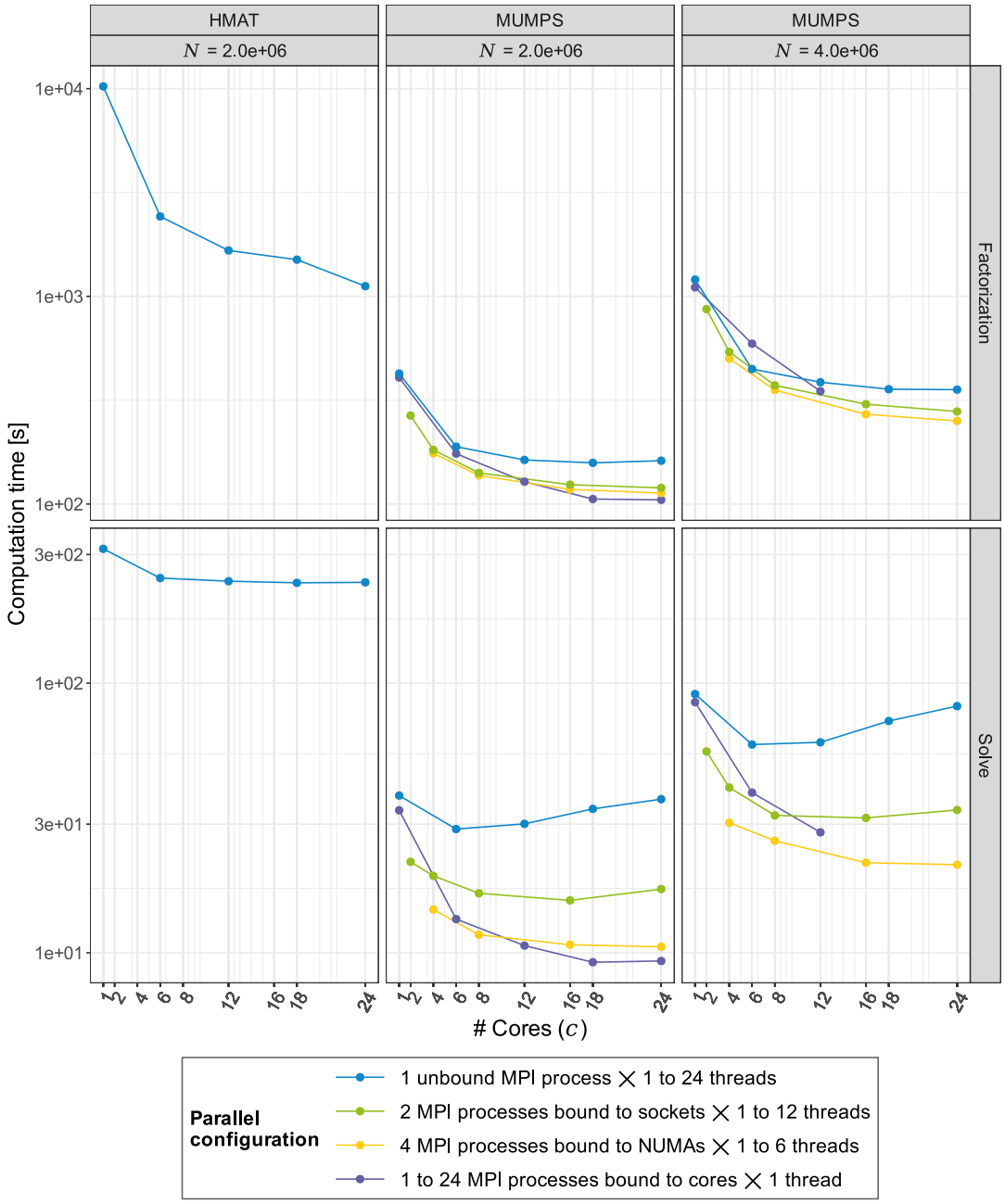

FIGURE 29: Scalability of MUMPS and HMAT on FEM systems run in 4 different kinds of parallel configurations using only 1 MPI process without binding times 1 to 24 OpenMP, MKL threads and StarPU workers or 2 MPI processes bound to sockets times 1 to 12 OpenMP, MKL threads and StarPU workers or 4 MPI processes bound to NUMA sub-nodes times 1 to 6 OpenMP, MKL threads and StarPU workers or 1 to 24 MPI processes bound to cores times 1 OpenMP, MKL thread and StarPU worker on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} in case of HMAT and MUMPS when compression is enabled. Out-of-core has been disabled.

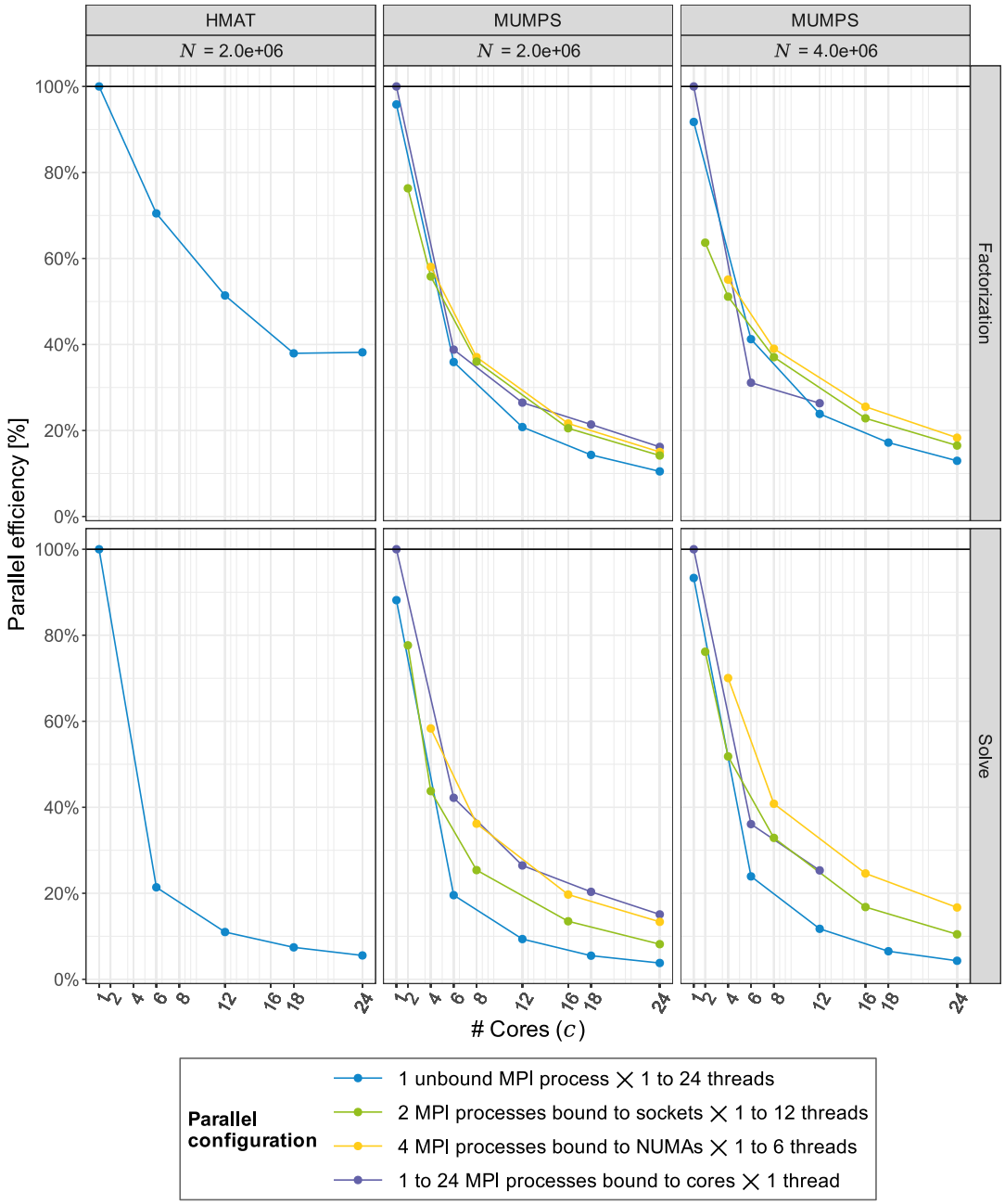

FIGURE 30: Parallel efficiency of MUMPS and HMAT on FEM systems run in 4 different kinds of parallel configurations using only 1 MPI process without binding times 1 to 24 OpenMP, MKL threads and StarPU workers or 2 MPI processes bound to sockets times 1 to 12 OpenMP, MKL threads and StarPU workers or 4 MPI processes bound to NUMA sub-nodes times 1 to 6 OpenMP, MKL threads and StarPU workers or 1 to 24 MPI processes bound to cores times 1 OpenMP, MKL thread and StarPU worker on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} in case of HMAT and MUMPS when compression is enabled. Out-of-core has been disabled.

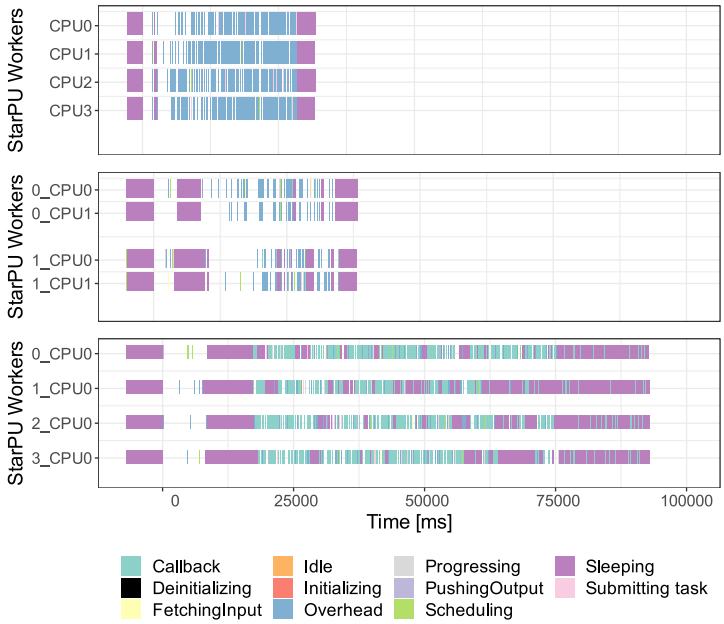

FIGURE 31: Graphical visualization of the execution traces provided by the StarPU runtime corresponding to the execution of HMAT on a FEM system having 25,000 unknowns using 1 MPI process times 4 StarPU workers, 2 MPI processes times 2 StarPU workers, 4 MPI processes with 1 StarPU worker per process on a single miriel node respectively. Blank spaces represent the time spent in computation. The colors represent the actions of the StarPU runtime. Out-of-core has been disabled.

7.7 Coupled FEM/BEM systems

The ultimate goal of this experimental study is to assess the existing implementations involving the direct solvers from the state of the art (see Section 4.2) for the solution of the target coupled FEM/BEM linear systems (see Section 3).

The main and the most computation time and memory consuming steps of the solution process are the computation of the Schur complement followed by the factorization and the solution of the reduced dense Schur complement system (see Section 5).

On one hand, we can proceed using a two-stage implementation scheme (see Section 5.4) combining a sparse solver for the Schur complement computation and a dense solver for the factorization and the solve of the reduced system. On the other hand, we can rely on one single solver to perform both sparse and dense operations using a single-stage implementation scheme (see Section 5.3).

Considering the short pipe test case (see Section 7.2), we benchmark both of the two-stage implementation schemes, multi-solve and multi-factorization (see sections 5.4.1 and 5.4.2), as well as the partially sparse-aware single-stage implementation scheme (see Section 5.3.3). Table 5 lists all of the implementation schemes and the associated solver configurations we evaluate in this section.

Implementation scheme	Type	Solver (couplings)
Multi-solve	two-stage	MUMPS/SPIDO
Multi-factorization	two-stage	MUMPS/SPIDO
Multi-factorization	two-stage	MUMPS/HMAT
Partially sparse-aware	single-stage	HMAT

TABLE 5: Implementation schemes and solvers assessed for the resolution of coupled FEM/BEM linear systems (see Section 3).

Thanks to the previous individual evaluations of each solver on either BEM or FEM linear systems, we are now able to study the performances of the aforementioned methods for solving coupled FEM/BEM linear systems and better understand the results being aware of the possible impact of each of the solvers involved in the computation.

Here, we consider coupled FEM/BEM linear systems counting from 250,000 up to 4,000,000 unknowns where the 126 GiB memory constraint (see Section 7.4.1) allows it. The exact counts of FEM and BEM unknowns are depicted in Table 6. We turn on data compression both for MUMPS and HMAT and always set the low-rank compression threshold ϵ (see Section 6.2) to 10^{-3}. The coefficient matrices are symmetric. Therefore, we rely on the LDL^T factorization in case of SPIDO and MUMPS and on the LL^T factorization in case of HMAT. Finally, all the benchmarks are run on a single miriel node (see Section 7.4.1) using 1 MPI process and all the available cores for OpenMP and MKL threads and StarPU workers in the case of HMAT (see Section 7.3).

7.7.1 Multi-solve

The overall performance of the multi-solve implementation is strongly affected by the value of n_c, the count of right-hand sides the sparse solver, MUMPS for instance, processes at the same time during the Schur complement computation phase (see Section 5.4.1). In other words, higher n_c means less iterations, and therefore less solve operations, in the Schur complement computation loop at line 4 in Algorithm 1 (p. 17). Nevertheless, increasing the value of n_c also leads to higher memory consumption.

Total unknowns	# BEM unknowns	# FEM unknowns
250,000	14,835	235,165
500,000	23,577	476,423
1,000,000	37,169	962,831
1,200,000	41,992	1,158,008
1,400,000	46,482	1,353,518
1,500,000	48,750	1,451,250
2,000,000	58,910	1,941,090
4,000,000	93,593	3,906,407

TABLE 6: Counts of BEM and FEM unknowns in the target coupled FEM/BEM systems according to the overall unknown count.

Here, we run a series of benchmarks of the multi-solve implementation while varying the value of n_c from 32 (default setting of MUMPS [31]) up to 256 and the linear system's unknown count from 250,000 up to 4,000,000.

In Figure 32 we show the computation time of the major factorization phase matching the lines 3 to 9 in Algorithm 1 (p. 17). The factorization phase includes the Schur complement computation step corresponding to lines 3 to 7. Line 10 corresponds to the final solution of the reduced system.

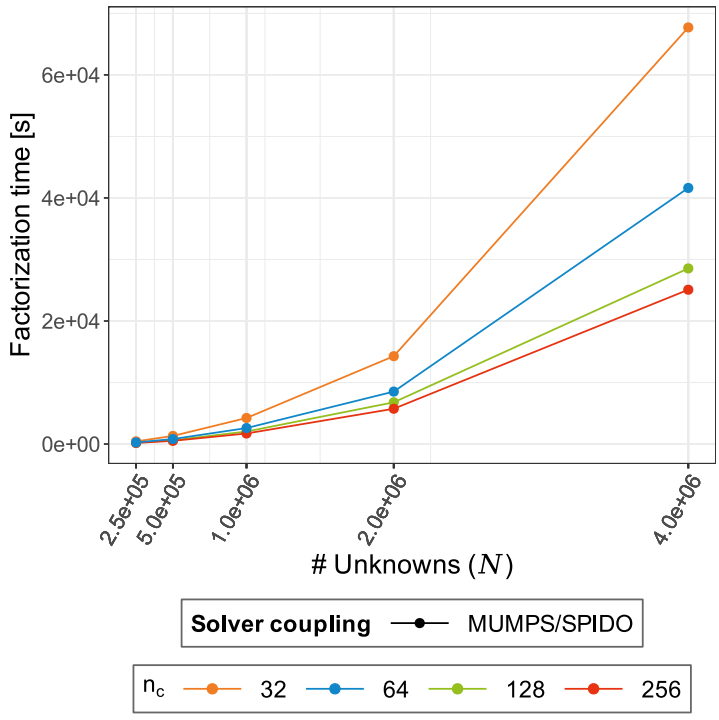

FIGURE 32: Factorization time of the two-stage **multi-solve** implementation scheme (see Section 5.4.1) using the coupling of MUMPS (sparse solver) and SPIDO (dense solver) for various sizes of blocks of columns n_c processed in parallel by MUMPS during the computation of Schur complement. Parallel runs using 1 MPI process, 24 OpenMP and MKL threads on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} for MUMPS. Out-of-core has been disabled.

According to the results, reducing the number of solves in the Schur complement computation loop by increasing the value of n_c allows for lower execution times. In other words, the higher

the value of n_c, the more right-hand sides can be processed in parallel by the sparse solver. Nevertheless, choosing n_c as high as possible is not the aim here. While increasing its value from 32 to 64 translates into a significant reduction of the computation time, the effect is noticeably smaller in case of higher values of n_c, e. g. between 128 and 256. Growing n_c also means that more memory space needs to be allocated during each loop iteration (see Figure 33). However, the amount of computation time we can save is more important than the increase of RAM consumption, especially in case of larger systems with more than 1,000,000 unknowns.

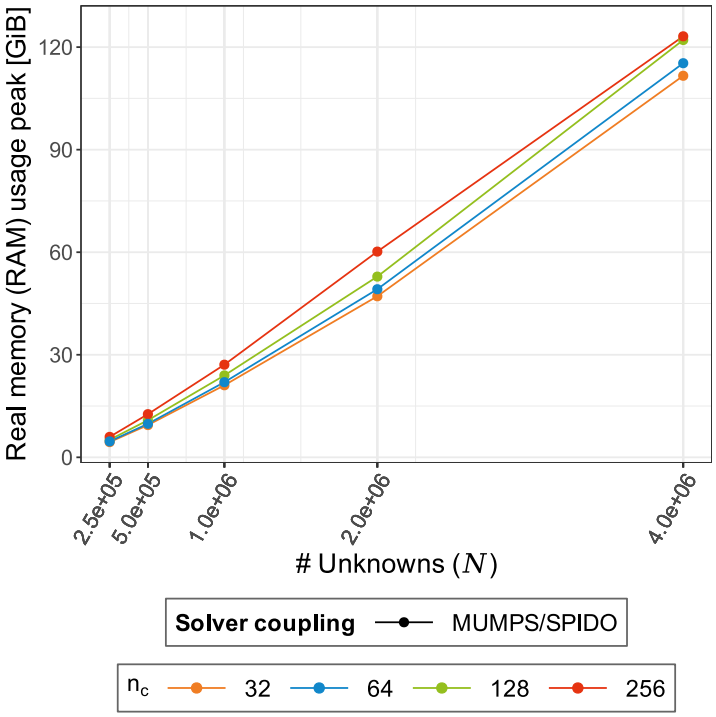

FIGURE 33: Real memory (RAM) usage peaks of the two-stage **multi-solve** implementation scheme (see Section 5.4.1) using the coupling of MUMPS (sparse solver) and SPIDO (dense solver) for various sizes of blocks of columns n_c processed in parallel by MUMPS during the computation of Schur complement. Parallel runs using 1 MPI process, 24 OpenMP and MKL threads on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} for MUMPS. Out-of-core has been disabled.

Finally, Figure 34 illustrates the RAM usage of the multi-solve scheme in function of time. After the initial assembly phase, the RAM consumption rises and remains relatively stable during the entire Schur complement computation step.

7.7.2 Multi-factorization

In the case of the multi-factorization implementation, n_b indicates the count of blocks per row and column of the dense sub-matrix A_{ss} and it largely influences the computation time of the Schur complement (see Section 5.4.2). n_b actually determines the number of calls to the Schur complement computation routine `CreateSchurComplement`. The higher it is relatively to the number of BEM unknowns in the coupled linear system, the more factorizations of the sub-matrix A_{vv} are performed inside of the loop at line 3 in Algorithm 2 (p. 18). Lot of factorization operations can strongly degrade the performance.

Here, we run a series of benchmarks with n_b varying between 1 to 12 blocks per row or column of A_{ss}. The linear system's unknown count N goes from 250,000 up to 1,000,000. We reach the

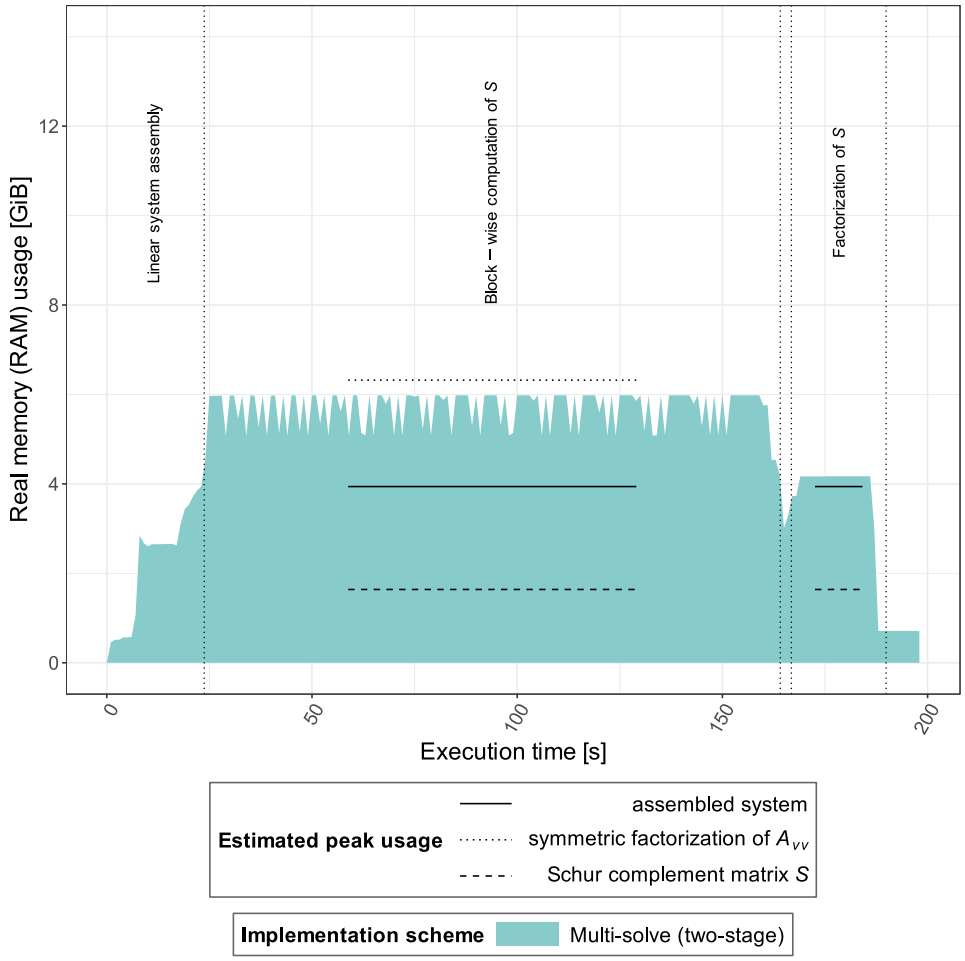

FIGURE 34: Evolution of the real memory (RAM) usage during the execution of the two-stage **multi-solve** implementation scheme (see Section 5.4.1) using the coupling of MUMPS (sparse solver) and SPIDO (dense solver) on a coupled FEM/BEM linear system having 250,000 unknowns with the size of blocks of columns n_c set to 256. Parallel run using 1 MPI process, 24 OpenMP and MKL threads on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} for MUMPS. Out-of-core has been disabled.

memory limit before the end of computation for $N > 1,400,000$ when using SPIDO as dense solver and for $N > 1,500,000$ when using HMAT. In Figure 35, we show the computation times of the factorization phase matching the lines 3 to 9 in Algorithm 2 (p. 18). The factorization phase includes the Schur complement computation step corresponding to lines 3 to 6. Line 10 corresponds to the final solution of the reduced system.

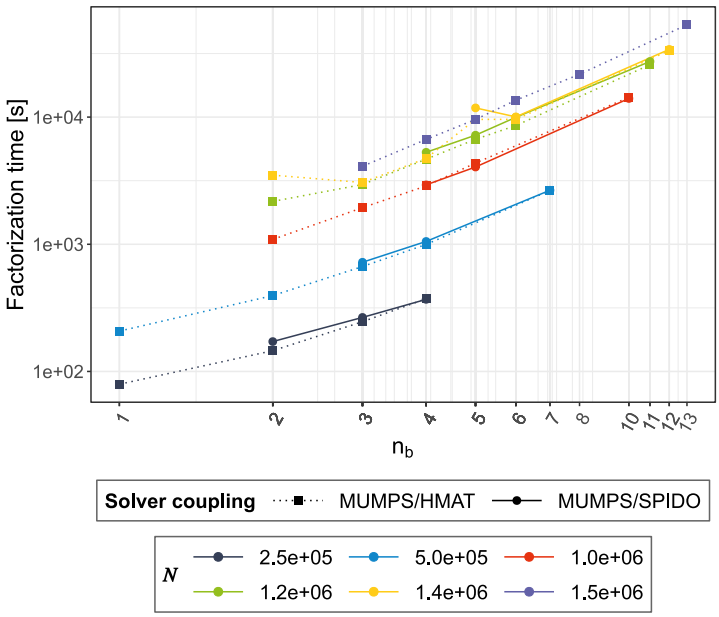

FIGURE 35: Factorization time of the two-stage **multi-factorization** (see Section 5.4.2) implementation using the coupling of MUMPS (sparse solver) and SPIDO or HMAT (dense solvers) for various counts of blocks n_b of S used during the computation of the Schur complement. Parallel runs using 1 MPI process, 24 OpenMP and MKL threads and StarPU workers on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} in case of MUMPS and HMAT. Out-of-core has been disabled.

According to the results, lower values of n_b and consequently a lower number of calls to the Schur complement computation routine greatly improve the performance of the computation in terms of execution time for both solver couplings. On the other hand, using less but larger blocks represents considerably higher memory requirements (see Figure 36). For example, in the case of the MUMPS/HMAT coupling on a system with 1,000,000 unknowns we were unable to use only 1 block as the latter did not fit into RAM. For MUMPS/SPIDO, using 1 block was not possible at all and on a system having 1,000,000 unknowns the lowest value of n_b was 4. Regarding the MUMPS/SPIDO coupling, there is another limitation besides the amount of available memory. In the implementation of SPIDO, the total size of a block in bytes is encoded using a 32-bit integer type. In double complex arithmetic, the size of one element is 16 bytes which limits the count of rows and columns of one block to n such that $16n^2 = \frac{2^{32}}{2} - 1$ allowing n to be at most $11,585$.

In summary, the MUMPS/HMAT coupling performs better than MUMPS/SPIDO. The difference in term of computation time for the same value of n_b is small but at the end, the possibility of considering larger blocks during the Schur complement computation in case of MUMPS/HMAT significantly lowers its computation time compared to MUMPS/SPIDO.

Finally, Figure 37 illustrates the RAM usage of the multi-factorization scheme in function of time. Following the initial assembly phase, we can observe three peaks representing the calls to the Schur complement computation routine for each of the blocks of the corresponding matrix S. Note that the important amout of data duplication related to the multi-factorization scheme

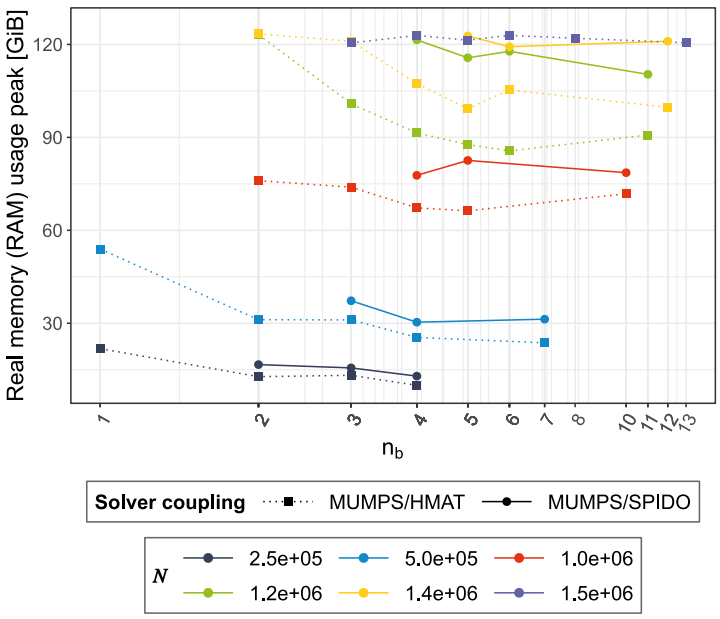

FIGURE 36: Real memory (RAM) usage peaks of the two-stage **multi-factorization** implementation scheme (see Section 5.4.2) using the coupling of MUMPS (sparse solver) and SPIDO or HMAT (dense solver) for various counts of blocks n_b of S used during the computation of the Schur Schur complement. Parallel runs using 1 MPI process, 24 OpenMPI and MKL threads on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} for MUMPS. Out-of-core has been disabled.

(see Section 5.4.2) largely contributes to the high memory consumption during the computation of the Schur complement blocks.

7.7.3 Partially sparse-aware scheme and overall comparison

At the end, we run a series of benchmarks putting in action only one single solver to solve the target coupled system (see Section 5.3). Here, we consider the partially sparse-aware single-stage implementation scheme (see Section 5.3.3) with HMAT used for both sparse and dense operations. Note that the coefficient matrix A is split into three separate hierarchical sub-matrices A_{vv}, A_{sv} and A_{ss}. The linear system's unknown count ranges from 250,000 up to 2,000,000. Beyond this size, we reach the memory limit before the end of computation.

In Figure 38 we compare the best factorization times (including the Schur complement computation and the factorization of the reduced Schur complement system) of the two-stage multi-solve and multi-factorization coupled solver schemes to the single-stage partially sparse-aware scheme implemented using HMAT.

The worst performing variant seems to be the multi-factorization implementation using SPIDO as dense solver. The MUMPS/SPIDO multi-solve implementation is not the best performing one, but its lower memory consumption allows it to process bigger linear systems compared to the other approaches, with up to 4,000,000 unknowns. Then, the MUMPS/HMAT multi-factorization implementation seems to be the fastest one at the beginning and even outperforms multi-solve on systems with up to 1,000,000 unknowns. However, with an increasing unknown count, i.e. $N \geq 1,000,000$, the single-stage sparse-aware scheme implemented with HMAT becomes the fastest one.

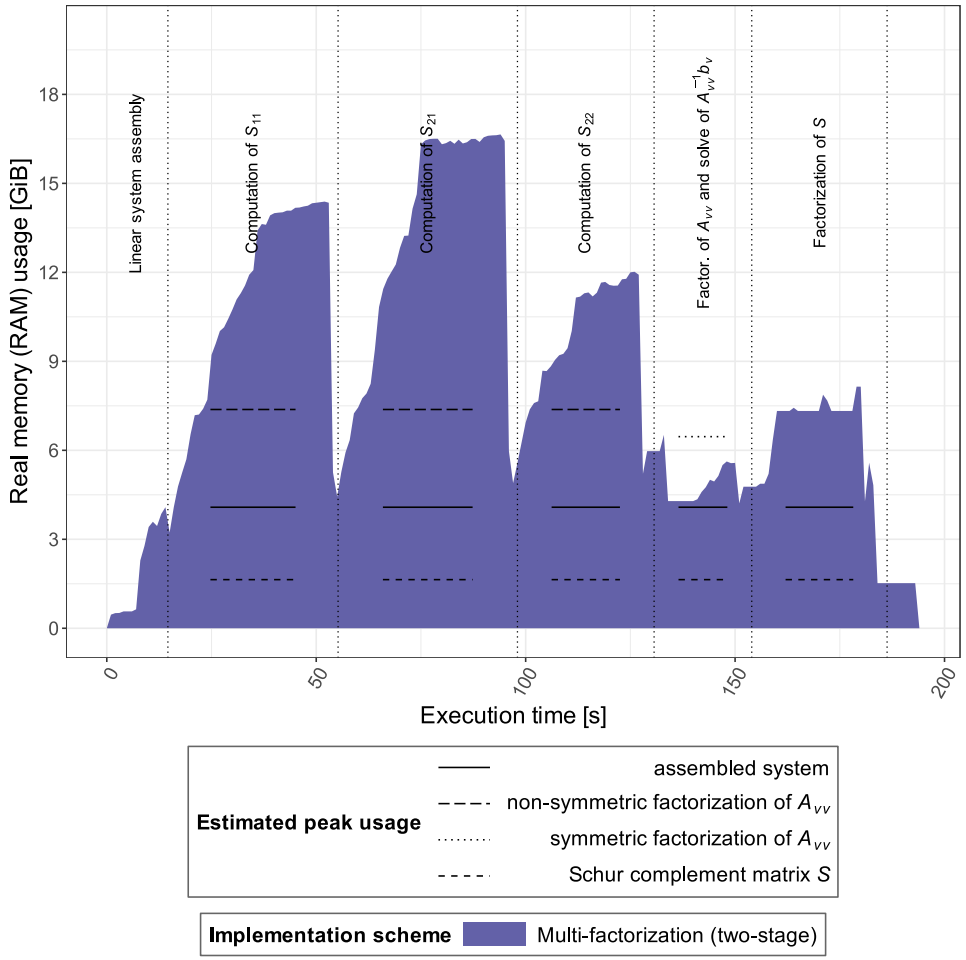

FIGURE 37: Evolution of the real memory (RAM) usage during the execution of the two-stage **multi-factorization** implementation scheme (see Section 5.4.2) using the coupling of MUMPS (sparse solver) and SPIDO (dense solver) on a coupled FEM/BEM linear system having 250,000 unknowns with the number of blocks n_b of S set to 2. Parallel run using 1 MPI process, 24 OpenMP and MKL threads on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} for MUMPS. Out-of-core has been disabled.

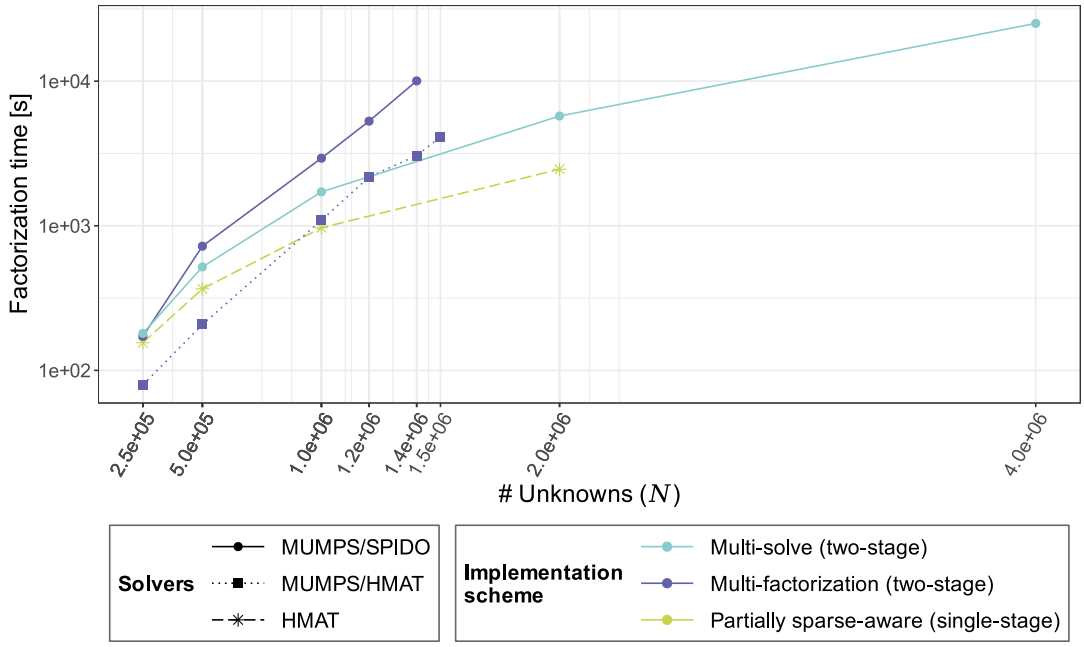

FIGURE 38: Best computation times of the two-stage **multi-solve** and the **multi-factorization** implementations (see Section 5.3) compared to the single-stage **partially sparse-aware** scheme (see Section 5.3) using HMAT on coupled FEM/BEM systems. Parallel runs using 1 MPI process, 24 OpenMP and MKL threads and StarPU workers on single miriel node with the low-rank compression threshold ϵ (see Section 6.2) set to 10^{-3} in case of MUMPS and HMAT. Out-of-core has been disabled.

8 Conclusion

In this report, we have presented our preliminary study on the performance of selected sparse and dense direct solvers committed to solve coupled FEM/BEM linear systems arising from the discretization of aeroacoustic problems relying on the Finite Elements Method (FEM) and Boundary Elements Method (BEM).

The first part of the experiments was dedicated to the evaluation of the direct solvers SPIDO and HMAT on the solution of purely dense BEM systems. We have observed that the hierarchical matrix structure and the low-rank compression in HMAT provides the solver with an important performance advantage over SPIDO, both in terms of computation time and memory consumption. On the other hand, on a single computation node, HMAT scales well only if it relies exclusively on StarPU worker parallelism. In this case, StarPU is more adaptive to workload appearing on the fly compared to statically mapped MPI processes. SPIDO scales well in all parallel configurations.

The second part was aimed to evaluate the direct solvers MUMPS and HMAT on the solution of purely sparse FEM systems. The results showed a noticeably better performance of MUMPS compared to HMAT on sparse matrices. The implementation limitations, including the prototype of Nested Dissection, in HMAT which was primarily meant for dense matrices does not allow the solver to process as large linear systems as MUMPS yet. HMAT also consumes more memory and computation time while considering the same hardware. MUMPS scales well for all parallel configurations. The scalability of HMAT on sparse matrices presents the same tendencies as on dense matrices.

We have evaluated existing implementation schemes involving the solvers from the state of

the art for the solution of FEM/BEM linear systems in the third and last part of the study. According to our findings, the worst performing two-stage implementation scheme was the multi-factorization variant using MUMPS as sparse and SPIDO as dense solver. Followed the multi-solve implementation using the same couple of solvers. On the other hand, the latter was able to process the largest coupled systems we have considered. Replacing the usage of SPIDO by HMAT providing the low-rank compression allowed in multi-factorization for a significant improvement of performance. This configuration can even outperform multi-solve up to a given size of the linear system. Eventually, the single-stage partially sparse-aware scheme implemented with HMAT turns out to be the fastest one, at least for systems with up to 2,000,000 unknowns.

As of the overall comparison between the different two-stage schemes, we are not able to make a clear statement yet. We need to address multiple issues prior to being able to conduct a more complex memory-aware study of the two-stage schemes, i. e. the limitation on the maximum size of Schur complement blocks in case of multi-factorization relying on SPIDO as dense solver. Moreover, we are currently working on the implementation of multi-solve for the MUMPS/HMAT coupling. The performance advantage that may be reached using HMAT also motivates our future work on the ideal sparse-aware single-stage implementation scheme in an effort to further speed-up the solution of coupled systems and ensure a better control over out-of-core computations. In addition, we plan to include other sparse solver alternatives such as qr_mumps [28, 9] (version 3 now includes a sparse Gaussian elimination).

References

[1] *(g)enerate (c)ompute (v)alidate (b)enchmark, a Python 3 module aiming at facilitating non-regression, validation and benchmarking of simulation codes.* `https://github.com/felsocim/gcvb`.

[2] *ggplot2, a system for declaratively creating graphics.* `https://ggplot2.tidyverse.org/`.

[3] *GNU C Compiler.* `https://gcc.gnu.org/`.

[4] *GNU Guix software distribution and transactional package manager.* `https://guix.gnu.org`.

[5] *hmat-oss, a hierarchical matrix C/C++ library including a LU solver.* `https://github.com/jeromerobert/hmat-oss`.

[6] *Intel(R) Math kernel library.* `https://software.intel.com/content/www/us/en/develop/tools/math-kernel-library.html`.

[7] *OpenMPI: A high performance message passing library.* `https://www.open-mpi.org/`.

[8] *PlaFRIM: Plateforme fédérative pour la recherche en informatique et mathématiques.* `https://plafrim.fr/`.

[9] *qr_mumps, a software package for the solution of sparse, linear systems on multicore computers.* `http://buttari.perso.enseeiht.fr/qr_mumps/`.

[10] *Quick start user's guide for slurm.* `https://slurm.schedmd.com/quickstart.html`.

[11] *slurm workload manager.* `https://slurm.schedmd.com`.

[12] *test_FEMBEM, a simple application for testing dense and sparse solvers with pseudo-FEM or pseudo-BEM matrices.* `https://gitlab.inria.fr/solverstack/test_fembem`.

[13] *The R project for statistical computing.* `https://www.r-project.org/`.

[14] E. Agullo, M. Felšöci, and G. Sylvand, *A comparison of selected solvers for coupled FEM/BEM linear systems arising from discretization of aeroacoustic problems: literate and reproducible environment*, Rapport technique RT-0513, Inria, June 2021.

[15] G. Alleon, *Résolution de grands problèmes d'électromagnétisme sur calculateurs parallèles*, PhD thesis, 2000. Thèse de doctorat dirigée par Petiton, Serge Informatique Paris 6 2000.

[16] P. Amestoy, J.-Y. L'Excellent, and G. Moreau, *On exploiting sparsity of multiple right-hand sides in sparse direct solvers*, SIAM Journal on Scientific Computing, 41 (2019), pp. A269–A291.

[17] P. R. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker, *Improving multifrontal methods by means of block low-rank representations*, SIAM Journal on Scientific Computing, 37 (2015), pp. A1451–A1474.

[18] P. R. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM Journal on Matrix Analysis and Applications, 17 (1996), pp. 886–905.

[19] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent, *MUMPS multifrontal massively parallel solver version 2.0*, (1998).

[20] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, et al., *LAPACK Users' guide*, vol. 9, SIAM, 1999.

[21] C. Augonnet, S. Thibault, and R. Namyst, *StarPU: a Runtime System for Scheduling Tasks over Accelerator-Based Multicore Machines*, Rapport de recherche RR-7240, INRIA, Mar. 2010.

[22] T. Banachiewicz, *Principes d'une nouvelle technique de la méthode des moindres carrés; méthode de résolution numérique des équations linéaires, du calcul des déterminants et des inverses et de réduction des formes quadratiques*, Bull. Inter. Acad. Polon. Sci., Sér. A, (1938), pp. 393–404.

[23] P. K. Banerjee and R. Butterfield, *Boundary element methods in engineering science*, vol. 17, McGraw-Hill London, 1981.

[24] S. Brenner and R. Scott, *The mathematical theory of finite element methods*, vol. 15, Springer Science & Business Media, 2007.

[25] U. Brink, M. Kreienmeyer, and E. Stein, *Different methodologies for coupled bem and fem with implementation on parallel computers*, in Boundary Element Topics, W. L. Wendland, ed., Berlin, Heidelberg, 1997, Springer Berlin Heidelberg, pp. 317–337.

[26] D. Brunner, M. Junge, and L. Gaul, *A comparison of fe–be coupling schemes for large-scale problems with fluid–structure interaction*, International Journal for Numerical Methods in Engineering, 77 (2009), pp. 664 – 688.

[27] J. R. Bunch and L. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, Mathematics of computation, (1977), pp. 163–179.

[28] A. Buttari, *Fine-Grained Multithreading for the Multifrontal QR Factorization of Sparse Matrices*, SIAM Journal on Scientific Computing, vol. 35 (2013), pp. pp. 323–345.

[29] F. Casenave, *Aéroacoustique, couplage BEM-FEM 3D pour la simulation du bruit rayonné par un turboréacteur*, tech. rep., 2010.

[30] F. Casenave, A. Ern, and G. Sylvand, *Coupled BEM–FEM for the convected Helmholtz equation with non-uniform flow in a bounded domain*, Journal of Computational Physics, 257 (2014), pp. 627–644.

[31] CERFACS, ENS Lyon, INPT(ENSEEIHT)-IRIT, Inria, Mumps Technologies, Université de Bordeaux, *MUltifrontal Massively Parallel Solver (MUMPS) User's guide*, 2020.

[32] A.-L. Cholesky, *Note sur une méthode de résolution des équations normales provenant de l'application de la méthode des moindres carrés à un syteme d'équations linéaires en nombre inférieur à celui des inconnues*, Bull. Géodésique, 2 (1924), pp. 67–77.

[33] A. De Coninck, D. Kourounis, F. Verbosio, O. Schenk, B. De Baets, S. Maenhout, and J. Fostier, *Towards parallel large-scale genomic prediction by coupling sparse and dense matrix algebra*, in 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, 2015, pp. 747–750.

[34] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Transactions on Mathematical Software (TOMS), 9 (1983), pp. 302–325.

[35] ——, *The multifrontal solution of unsymmetric sets of linear equations*, SIAM Journal on Scientific and Statistical Computing, 5 (1984), pp. 633–641.

[36] A. Ern and J.-L. Guermond, *Theory and practice of finite elements*, vol. 159, Springer Science & Business Media, 2013.

[37] L. Euler, *Vollständige Anleitung zur Algebra: von den verschiedenen Rechnungsarten, Verhältnissen und Proportionen. Zweyter Theil*, Auf Kosten C. F. Schiermann, 1771.

[38] A. Falco, *Combler l'écart entre H-Matrices et méthodes directes creuses pour la résolution de systèmes linéaires de grandes tailles*, thèse de doctorat, Université de Bordeaux, June 2019.

[39] M. Ganesh and C. Morgenstern, *High-order fem–bem computer models for wave propagation in unbounded and heterogeneous media: Application to time-harmonic acoustic horn problem*, Journal of Computational and Applied Mathematics, 307 (2016), pp. 183–203. 1st Annual Meeting of SIAM Central States Section, April 11–12, 2015.

[40] V. Garcia Pinto, L. Mello Schnorr, L. Stanisic, A. Legrand, S. Thibault, and V. Danjean, *A visual performance analysis framework for task-based parallel applications running on hybrid clusters*, Concurrency and Computation: Practice and Experience, 30 (2018), p. e4472. e4472 cpe.4472.

[41] M. C. Genes, *Parallel application on high performance computing platforms of 3D BEM/FEM based coupling model for dynamic analysis of SSI problems*, CIMNE, 2013, p. 205–216.

[42] A. George, *Nested dissection of a regular finite element mesh*, SIAM Journal on Numerical Analysis, 10 (1973), pp. 345–363.

[43] A. George, J. W. Liu, and E. Ng, *Computer solution of sparse linear systems*, Academic, Orlando, (1994).

[44] D. Goldberg, *What every computer scientist should know about floating-point arithmetic*, ACM Computing Surveys (CSUR), 23 (1991), pp. 5–48.

[45] J. F. Grcar, *Mathematicians of Gaussian elimination*, Notices of the AMS, 58 (2011), pp. 782–792.

[46] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part I: Introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108. 10.1007/s006070050015.

[47] W. Hackbusch, *Hierarchical matrices: Algorithms and analysis*, vol. 49, Springer, 2015.

[48] B. Heise, *Parallel Solvers for coupled FEM-BEM equations with applications to non-linear magnetic field problems*, Vieweg+Teubner Verlag, Wiesbaden, 1995, pp. 73–85.

[49] B. Heise and M. Kuhn, *Parallel solvers for linear and nonlinear exterior magnetic field problems based upon coupled fe/be formulations*, Computing, 56 (1996), pp. 237–258.

[50] P. Hénon, P. Ramet, and J. Roman, *PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems*, Parallel Computing, 28 (2002), pp. 301–321.

[51] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, no. 48, SIAM, 2002.

[52] Inria, *Parallel Sparse matriX package (PaStiX) User's guide*, 2013.

[53] N. Kamiya and H. Iwase, *Bem and fem combination parallel analysis using conjugate gradient and condensation*, Engineering Analysis with Boundary Elements, 20 (1997), pp. 319–326.

[54] N. Kamiya, H. Iwase, and E. Kita, *Parallel computing for the combination method of bem and fem*, Engineering Analysis with Boundary Elements, 18 (1996), pp. 223–229. Parallel BEM and Related Methods and Algorithms.

[55] D. E. Knuth, *Literate programming*, Comput. J., 27 (1984), p. 97–111.

[56] S. Kocak and H. Akay, *Parallel schur complement method for large-scale systems on distributed memory computers*, Applied Mathematical Modelling, 25 (2001), pp. 873–886.

[57] M. Kuhn, *Fem-bem coupling and parallel multigrid solvers for 3d magnetic field problems*, in European Congress on Computational Methods in Applied Sciences and Engineering ECCOMAS Barcelona, 2000, pp. 1–12.

[58] J.-Y. L'Excellent, *Multifrontal Methods: Parallelism, Memory Usage and Numerical Aspects*, habilitation à diriger des recherches, École normale supérieure de Lyon - ENS LYON, Sept. 2012.

[59] J. W. Liu, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Transactions on Mathematical Software (TOMS), 11 (1985), pp. 141–153.

[60] ——, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM review, 34 (1992), pp. 82–109.

[61] B. Lizé, *Solveur direct haute performance*, tech. rep., EADS IW/École centrale Paris, 2009.

[62] E. Ng and P. Raghavan, *Performance of greedy ordering heuristics for sparse Cholesky factorization*, SIAM Journal on Matrix Analysis and Applications, 20 (1999), pp. 902–914.

[63] R. Perera and E. Alarcon, *Parallel algorithms for fe/be coupling*, 1998.

[64] V. G. Pinto, L. Stanisic, A. Legrand, L. M. Schnorr, S. Thibault, and V. Danjean, *Analyzing dynamic task-based applications on hybrid platforms: An agile scripting approach*, in 2016 Third Workshop on Visual Performance Analysis (VPA), 2016, pp. 17–24.

[65] P. Raviart and J. Thomas, *A mixed finite element method for 2-nd order elliptic problems*, in Mathematical Aspects of Finite Element Methods, I. Galligani and E. Magenes, eds., vol. 606 of Lecture Notes in Mathematics, Springer Berlin Heidelberg, 1977, pp. 292–315.

[66] V. Rischmuller, M. Haas, S. Kurz, and W. Rucker, *3d transient analysis of electromechanical devices using parallel bem coupled to fem*, IEEE Transactions on Magnetics, 36 (2000), pp. 1360–1363.

[67] E. Rothberg and S. C. Eisenstat, *Node selection strategies for bottom-up sparse matrix ordering*, SIAM Journal on Matrix Analysis and Applications, 19 (1998), pp. 682–695.

[68] Y. Saad, *Iterative methods for sparse linear systems*, SIAM, 2003.

[69] S. A. Sauter and C. Schwab, *Boundary Element Methods*, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 183–287.

[70] M. Schauer, J. E. Roman, E. S. Quintana-Ortí, and S. Langer, *Parallel computation of 3-d soil-structure interaction in time domain with a coupled fem/sbfem approach*, Journal of Scientific Computing, 52 (2012), pp. 446–467.

[71] A. Schwarzenberg-Czerny, *On matrix factorization and efficient least squares solution.*, Astronomy and Astrophysics Supplement Series, 110 (1995), p. 405.

[72] Sebaso, *Jet engine airflow during take-off.* `https://commons.wikimedia.org/wiki/File:20140308-Jet_engine_airflow_during_take-off.jpg`.

[73] T. Steinmetz, N. Godel, G. Wimmer, M. Clemens, S. Kurz, and M. Bebendorf, *Efficient symmetric fem-bem coupled simulations of electro-quasistatic fields*, IEEE Transactions on Magnetics, 44 (2008), pp. 1346–1349.

[74] L. Trefethen and D. Bau, *Numerical linear algebra*, Miscellaneous Bks, Society for Industrial and Applied Mathematics, 1997.

[75] Université de Bordeaux, CNRS (LaBRI UMR 5800), Inria, *StarPU Handbook*, 2020.

[76] A. Wang, N. Vlahopoulos, and K. Wu, *Development of an energy boundary element formulation for computing high-frequency sound radiation from incoherent intensity boundary conditions*, Journal of Sound and Vibration, 278 (2004), pp. 413 – 436.

[77] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM Journal on Algebraic Discrete Methods, 2 (1981), pp. 77–79.

[78] F. Zhang, *The Schur complement and its applications*, vol. 4, Springer Science & Business Media, 2006.

[79] P. Zhang, T. Wu, and R. Finkel, *Parallel computation for acoustic radiation in a subsonic nonuniform flow with a coupled fem/bem formulation*, Engineering Analysis with Boundary Elements, 23 (1999), pp. 139–153.

[80] O. Zienkiewicz and R. Taylor, *The finite element method*, vol. 3, McGraw-hill London, 1977.

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour

33405 Talence Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399