Comparison of coupled solvers for FEM/BEM linear systems arising from discretization of aeroacoustic problems

Emmanuel Agullo, Marek Felšöci, Guillaume Sylvand

Comparison of coupled solvers for FEM/BEM linear systems arising from discretization of aeroacoustic problems

Emmanuel Agullo, Marek Felšöci, Guillaume Sylvand

Centre de recherche Inria Bordeaux Sud-Ouest
200, avenue de la Vieille Tour
33405 Talence
emmanuel.agullo@inria.fr, marek.felsoci@inria.fr, guillaume.sylvand@airbus.com

---

Résumé
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 work, we propose and compare a set of implementation schemes relying on the coupling of the open-source sparse direct solver MUMPS with the proprietary direct solvers from Airbus Central R&T, i.e. the scalapack-like dense solver SPIDO and the hierarchical $\mathcal{H}$-matrix compressed solver HMAT. For this preliminary study, we limit ourselves to a single 24-core computational node.

Mots-clés : aeroacoustics, finite elements, boundary elements, solver comparison, coupled FEM/BEM linear systems

---

1. Introduction

In the industrial context of Airbus, we are interested in models of aeroacoustic phenomena such as the propagation of acoustic waves emitted by an aircraft on take-off, landing and taxiing (see Figure 1, left). Such physical models are typically expressed using partial differential equations. Prior to computing the numerical model, an approximation of its original physical expression is made over a limited domain using a suitable discretization technique.

FIGURE 1 – An acoustic wave (blue arrow) emitted by the aircraft's engine, reflected on the wing and crossing the jet of exhaust gas. Real-life case (left) [22] and a numerical model example (right). The red 2D mesh represents the aircraft's surface and the surface of the green 3D cylinder volume mesh representing the exhaust flow.

Aeroacoustic waves may interfere with various media. In our model, we take into account the air around the aircraft, considered as a homogeneous medium, and jet exhaust flow, considered as heterogeneous. The discretization relies on a coupling of two different techniques, the Finite Elements Method (FEM) [10, 14, 19, 26] applied to the exhaust flow and the Boundary Elements Method (BEM) [9, 20, 23] applied to the surfaces (see Figure 1, right). This leads to a coupled FEM/BEM linear system having two groups of unknowns, x_s associated to a surface mesh s resulting from BEM and x_v associated to a volume mesh v resulting from FEM (see Equation 1). Ultimately, we target systems counting around 10^9 FEM-related unknowns and 10^6 BEM-related unknowns. Note that, R_1 and R_2 denote respectively the first and the second row of the linear system.

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

Here, A is a 2×2 symmetric coefficient matrix where A_{vv} is a large sparse submatrix representing the action of the volume part on itself, A_{ss} is a smaller dense submatrix representing the action of the exterior surface on itself, A_{sv} is a sparse submatrix representing the action of the volume part on the exterior surface and A_{vs} is the transpose of A_{sv} and represents the action of the exterior surface on the volume part (see Figure 2).

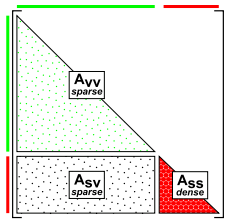

FIGURE 2 – Internal dimensions and sparsity of A in Equation 1.

In this work, we focus on a direct method for solving the FEM/BEM linear system in Equation 1. We distinguish two categories of implementation schemes for this method, single-stage and two-stage. The single-stage schemes rely on a single solver which takes the entire coefficient matrix A and the right-hand side on input and gives directly the final solution of the whole system on output. Ideally, the solver should be aware of the different sparsity levels of A and capable of applying appropriate sparse or dense operations. The two-stage schemes are based on a coupling of a sparse direct solver and a dense direct solver. Our goal here is to present and compare the two-stage schemes between them. In this case, we consider the coupling of the open-source sparse direct solver MUMPS [7, 12] with either the scalapack-like dense direct solver SPIDO or the hierarchical $\mathcal{H}$-matrix compressed solver HMAT. Both are proprietary solvers from Airbus Central R&T. However, an open-source sequential implementation of HMAT is available as HMAT-OSS [18, 2].

The document is organized as follows : in Section 2, we explain the direct solution method for the target linear system ; in Section 3, we detail the two-stage implementation schemes ; in Section 4, we describe our experimental evaluation of the latter and present the comparative results. We conclude in Section 5.

2. Direct solution of FEM/BEM systems

The first step of a direct solution of Equation 1 consists in reducing the problem on boundaries and simplifying the system to solve. Based on its first row R_1, we express x_v as :

$$x_v = A_{vv}^{-1}(b_v - A_{vs}x_s)\ . \tag{2}$$

Then, substituting x_v in R_2 by Equation 2 yields in R_2 a reduced system without x_v. This represents one step of Gaussian elimination, i.e. subtracting $A_{sv}A_{vv}^{-1}$ times R_1 from R_2 :

$$\begin{matrix} R_1 \\ R_2 \leftarrow R_2 - A_{sv}A_{vv}^{-1} \times R_1 \end{matrix} \begin{bmatrix} A_{vv} & A_{vs} \\ 0 & \underbrace{A_{ss} - A_{sv}A_{vv}^{-1}A_{vs}}_{S} \end{bmatrix} \times \begin{bmatrix} x_v \\ x_s \end{bmatrix} = \begin{bmatrix} b_v \\ b_s - A_{sv}A_{vv}^{-1}b_v \end{bmatrix}. \tag{3}$$

The expression $A_{ss} - A_{sv}A_{vv}^{-1}A_{vs}$ that appeared in R_2 in Equation 3 is often referred to as the Schur complement [24] noted S and associated with the partitioning, v and s, of the variables in the system (see Section 1). When solving the problem numerically, we factorize A_{vv} into a product of matrices making the system easier to solve rather than actually performing the inverse A_{vv}^{-1}. In summary, we have to compute S and solve the reduced Schur complement system matching R_2 in Equation 3 after eliminating x_v :

$$Sx_s = b_s - A_{sv}A_{vv}^{-1}b_v \; . \tag{4}$$

Once we have computed x_s, we use its value to determine x_v according to Equation 2.

3. Two-stage implementation schemes and related work

The principle of a two-stage implementation scheme is to use a sparse direct solver to compute the Schur complement S (see Equation 3) in the first stage and then, call a dense solver to factorize S and solve the reduced system (see Equation 4) in the second stage. Some sparse direct solvers, such as MUMPS, expose in their Application Program Interface (API) a Schur complement computation function. Provided with A_{vv}, A_{vs}, A_{sv} and A_{ss} they can compute S and yield the result as a dense matrix in A_{ss}. Although the sparse solver supports out-of-core computation [1], S still has to be entirely stored in RAM. In terms of memory constraints, this quickly becomes a significant limiting factor in solving large systems. 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 [13, 16, 25, 15, 21, 17]. Here, we rely on an algorithm allowing us to store S out-of-core, i.e. on disk, and keep using the community solvers. The idea is to compute the Schur complement S block-wise and load into memory only the block necessary for the current computation step. We propose two variants of such an algorithm, multi-solve and multi-factorization.

3.1. Multi-solve

In this approach, we compute the Schur complement S by blocks of columns (see Figure 3). 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 the definition of S in Equation 3, S_i is a block of n_c columns of S defined as :

$$S_i = A_{ss_i} - \underbrace{A_{sv}\overbrace{(L_{vv}L_{vv}^T)^{-1}A_{sv_i}^T}^{Y}}_{Z} . \tag{5}$$

The multi-solve algorithm (see Algorithm 1 in the appendix) begins by the LL^T factorization of A_{vv} (line 3) into the product of matrices $L_{vv}L_{vv}^T$ where L_{vv} is a lower triangular matrix and L_{vv}^T its transpose. Then, we loop (line 4) over the blocks of A_{sv}^T and A_{ss} to compute all the blocks S_i of S while overwriting A_{ss}. The first step of this computation (line 5) is a triangular solve done by a sparse solver for determining $Y = (L_{vv}L_{vv}^T)^{-1}A_{sv_i}^T$. It is possible to benefit from the sparsity of the right-hand side matrix $A_{sv_i}^T$ during the solve operation [6]. However, independently from the sparsity of the input right-hand side, the resulting Y is a **dense matrix** [12] with only a few zero entries. Also, the block $A_{sv_i}^T$ needs to be stored explicitly (see Figure 3). The number of columns n_c in S_i depends on n_{BEM}, the count of unknowns associated with the formulation of

1. Using out-of-core feature the solver can store on disk (out of the core memory) the parts of matrices not being actively used for computation, reduce memory consumption and process bigger problems with the same amount of memory.

BEM in the global linear system, and the amount of available RAM. If n_c is too small, a great number of solve operations are performed on a small number of right-hand sides 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 RAM.

3.2. Multi-factorization

In this case, the A_{ss} submatrix receiving the Schur complement S is split into square blocks of equal size (see Figure 4). Let us note A_{sv_i} a block of n_b rows of A_{sv} and $A_{sv_j}^T$ a block of n_b columns of A_{sv}^T. Then, based on the definition of S in Equation 3, S_{ij} is a block of n_b rows and columns of S such as :

$$S_{ij} = A_{ss_{ij}} - A_{sv_i}(L_{vv}U_{vv})^{-1}A_{sv_j}^T \ . \tag{6}$$

In the loop (line 2) of the multi-factorization algorithm (see Algorithm 2 in the appendix), we construct a submatrix W from A_{vv}, A_{sv_i} and $A_{sv_j}^T$:

$$W \leftarrow \begin{bmatrix} A_{vv} & A_{sv_j}^T \\ A_{sv_i} & 0 \end{bmatrix} \tag{7}$$

Then, we call the `SchurComplement` function on W (line 3) to compute the associated block S_{ij} of S relying on the Schur complement computation function provided by the sparse solver, e. g. MUMPS. Here, we decompose A_{vv} using the LU factorization as W is not symmetric except when $i = j$. This yields a **duplicated storage** of A_{vv}, A_{sv_i} and $A_{sv_j}^T$ in W (see Figure 4).

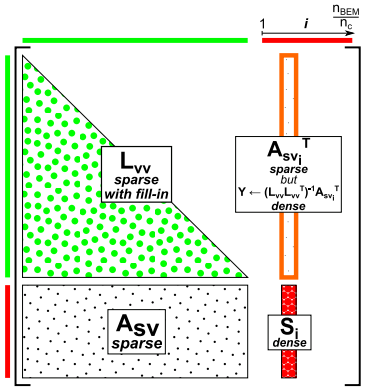

FIGURE 3 – Computation loop of S in multi-solve (see Algorithm 1). $A_{sv_i}^T$ is explicitly stored and Y is a **dense matrix**.

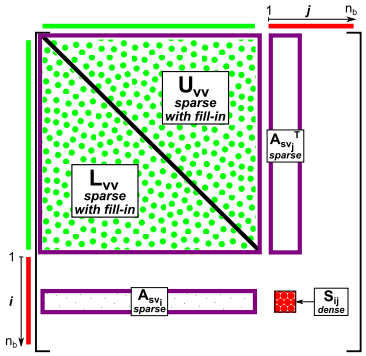

FIGURE 4 – Computation loop of S in multi-factorization (see Algorithm 2). Constructing W requires a **duplicated storage** of A_{vv}, A_{sv_i}, $A_{sv_j}^T$.

n_b represents the count of blocks per row or column of A_{ss}. The `SchurComplement` function operating on W implies a re-factorization of A_{vv} in W at each iteration, yet it does not change during the computation. Consequently, the more blocks A_{ss} is split into, the more superfluous factorizations of A_{vv} are performed. In Section 4, we show that too much blocks can substantially degrade the performance of the multi-factorization scheme. Similarly, too small values of n_c in the case of multi-solve (see Section 3.1) 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 both methods are designed to compute the Schur complement S, then solve the reduced system (see Section 2), but differ only in the way they make use of the available API provided by the community solvers. In the Airbus Central R&T solver framework, both variants are implemented using MUMPS as sparse solver, and SPIDO or HMAT as dense solver.

4. Experimental study

Within this preliminary study, we compare to each other the performance of the two-stage implementation schemes, multi-solve and multi-factorization, for solving the target coupled FEM/BEM linear system in Equation 1. Moreover, for each scheme, we consider both of the available solver couplings, i.e. using MUMPS as sparse solver and either HMAT or SPIDO as dense solver. In this section, we use the notations MUMPS/HMAT and MUMPS/SPIDO respectively to refer to these couplings. Our experiments rely on a short pipe test case (see Figure 6 in the appendix) yielding linear systems close enough to those arising from real life models (see Figure 1) while providing the scientific community with a reproducible example [5]. Note that MUMPS and HMAT both provide an out-of-core feature which we disable in our experiments. Also, these solvers rely on data compression and expose a precision parameter ϵ set to 10^{-3} for all the computations. We design the test case so as to know the expected result in advance. This way, we can determine the relative error of the computed solution. The relative error of the results we obtained was approximately 10^{-3}, which is a satisfactory accuracy in our context. We do not report further on that in the following.

We conduct our experiments on a single node of `miriel` on the PlaFRIM [3] high-performance computing platform. A `miriel` node has a total of 24 processor cores running each at 2.5 GHz and 126 GiB of RAM (see Figure 7 in the appendix). As for the software, we use the Airbus Central R&T solver test suite `test_FEMBEM` [5]. Table 1 in the appendix details the version and commit numbers of some of the core packages of our experimental software environment. We consider coupled FEM/BEM linear systems with N, the total unknown count, ranging from 250,000 up to 4×10^6. Table 2 in the appendix details the proportions of FEM and BEM unknowns for each value of N. In addition, we evaluate multiple configurations for each implementation scheme. Regarding multi-solve, we vary the number of columns n_c in a Schur complement block S_i (see Section 3.1). In case of MUMPS/HMAT, the value of n_c ranges from 512 to 4,096 columns. As of MUMPS/SPIDO, it varies between 32 and 256. When it comes to multi-factorization, we vary the count n_b of Schur complement blocks S_{ij} per row or column of the submatrix A_{ss} (see Section 3.2). Regardless the solver coupling, the tested values of n_b are between 1 and 12 blocks per row or column.

In Figure 5, for each solver coupling, we show the best computation times of multi-solve and multi-factorization among all of the evaluated configurations. The worst performing variant seems to be multi-factorization using SPIDO as dense solver. Using HMAT instead, we can benefit from data compression and lower the memory footprint. This allows us to consider fewer Schur complement blocks and consequently speed-up the computation by reducing the number of sparse factorizations performed by the routine `CreateSchurComplement` (see Section 3.2). Up to $N = 10^6$, this configuration can even outperform the multi-solve scheme. However, in case of larger systems, the memory constraint prevents us from lowering n_b enough despite using data compression. It may seem the compression is necessary for multi-factorization to be competitive at all. However, in the current 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 m such that $16m^2 = \frac{2^{32}}{2} - 1$ allowing m to be at most 11,585. This restricts the size of a Schur complement block in the case of the MUMPS/SPIDO coupling. Consequently, we can not always consider n_b as low as possible even if there is enough of memory available. Although we would not be able to lower n_b as much as in the case of the MUMPS/HMAT coupling due to the significantly hihger memory consumption of SPIDO compared to HMAT, a better performance for the MUMPS/SPIDO coupling might be achieved without this limitation. The multi-solve im-

plementation is not the best performing one, but its lower memory requirements allow us to process systems with up to 4×10^6 unknowns in the case of MUMPS/SPIDO. We have recently implemented multi-solve using HMAT as dense solver too.

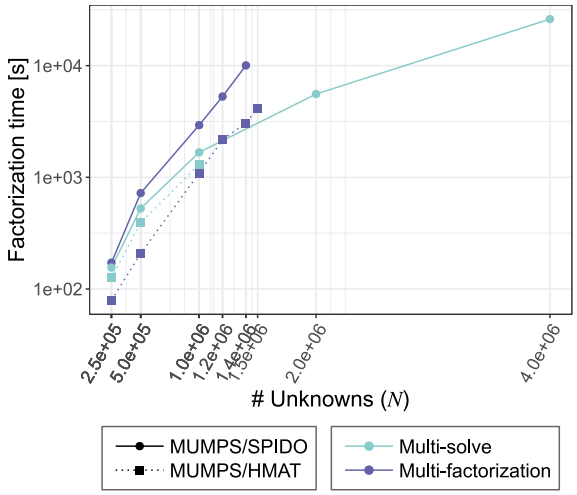

FIGURE 5 – Best computation times of **multi-solve** and **multi-factorization** for both of the solver couplings MUMPS/HMAT and MUMPS/SPIDO. Parallel runs using 24 threads on single `miriel` node.

It appears that using HMAT, we can speed up the computations but there is still a limitation in our implementation preventing us from running cases with $N > 10^6$ independently from the memory constraint. As of multi-solve, reducing the number of Schur complement blocks impacts each of the couplings differently. For MUMPS/SPIDO, it means solve operations with more right-hand sides treated simultaneously by MUMPS, hence improving performance, although not to the same extent as in multi-factorization. Moreover, increasing the block size too much would prevent us from solving systems with up to 4×10^6 unknowns due to the memory constraint. For MUMPS/HMAT, we need to take into account an additional computational cost of transforming the Schur complement blocks into $\mathcal{H}$-matrices. To minimize this cost, we use substantially bigger blocks, i.e. blocks of 2,048 columns compared to 256 in case of MUMPS/SPIDO. For now, setting n_c to values bigger than 2,048 seems causing MUMPS to produce incorrect results and does not allow us to further increase the block size regardless the memory constraint.

5. Conclusion

We have proposed and evaluated the multi-solve and multi-factorization two-stage implementation schemes for solving the target FEM/BEM linear system using both of the available solver couplings relying on MUMPS as sparse solver and on HMAT or on SPIDO as dense solver. According to our findings, replacing the usage of SPIDO by the HMAT solver based on the low-rank data compression can speed-up the computation in case of both implementation schemes. As of the overall comparison between the two schemes, we are not able to make a clear statement for now. Multi-solve using the coupling with SPIDO allowed us to process the largest coupled systems we have considered. On the other hand, due to the current limitations in our implementation of multi-solve with HMAT as dense solver, we could not confirm yet that this variant performs better on systems with more than 10^6 unknowns in total.

In the future, we should address the implementation limitations, perform a more complex study of the two-stage implementation schemes and try to establish a criterion determining when to prefer multi-solve over multi-factorization and vice-versa. Then, we shall the study of the impact of out-of-core computation. Eventually, we shall extend our benchmarks to multiple computational nodes. We also plan to include other solver alternatives such as qr_mumps [11, 4] (version 3 now includes a sparse Gaussian elimination) and further explore the possibilities of the alternative single-stage implementation schemes relying on one single solver optimized for both sparse and dense matrix operations.

Bibliographie

1. GNU C Compiler. – `https://gcc.gnu.org/`.
2. hmat-oss, a hierarchical matrix C/C++ library including a LU solver. – `https://github.com/jeromerobert/hmat-oss`.
3. PlaFRIM : Plateforme fédérative pour la recherche en informatique et mathématiques. – `https://plafrim.fr/`.
4. qr_mumps, a software package for the solution of sparse, linear systems on multicore computers. – `http://buttari.perso.enseeiht.fr/qr_mumps/`.
5. 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`.
6. Amestoy (P.), L'Excellent (J.-Y.) et Moreau (G.). – On exploiting sparsity of multiple right-hand sides in sparse direct solvers. *SIAM Journal on Scientific Computing*, vol. 41, n1, 2019, pp. A269–A291.
7. Amestoy (P. R.), Duff (I. S.) et L'Excellent (J.-Y.). – MUMPS multifrontal massively parallel solver version 2.0. 1998.
8. Augonnet (C.), Thibault (S.) et Namyst (R.). – *StarPU : a Runtime System for Scheduling Tasks over Accelerator-Based Multicore Machines*. – Rapport de recherche nRR-7240, INRIA, mars 2010.
9. Banerjee (P. K.) et Butterfield (R.). – *Boundary element methods in engineering science*. – McGraw-Hill London, 1981, volume 17.
10. Brenner (S.) et Scott (R.). – *The mathematical theory of finite element methods*. – Springer Science & Business Media, 2007, volume 15.
11. Buttari (A.). – Fine-Grained Multithreading for the Multifrontal QR Factorization of Sparse Matrices. *SIAM Journal on Scientific Computing*, vol. vol. 35, nn° 4, 2013, pp. pp. 323–345.
12. CERFACS, ENS Lyon, INPT(ENSEEIHT)-IRIT, Inria, Mumps Technologies, Université de Bordeaux. – *MUltifrontal Massively Parallel Solver (MUMPS) User's guide*, 2020.
13. De Coninck (A.), Kourounis (D.), Verbosio (F.), Schenk (O.), De Baets (B.), Maenhout (S.) et Fostier (J.). – 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*, pp. 747–750, 2015.
14. Ern (A.) et Guermond (J.-L.). – *Theory and practice of finite elements*. – Springer Science & Business Media, 2013, volume 159.
15. Ganesh (M.) et Morgenstern (C.). – 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*, vol. 307, 2016, pp. 183–203. – 1st Annual Meeting of SIAM Central States Section, April 11–12, 2015.
16. Genes (M. C.). – *Parallel application on high performance computing platforms of 3D BEM/FEM based coupling model for dynamic analysis of SSI problems*, p. 205–216. – CIMNE, 2013.
17. Kocak (S.) et Akay (H.). – Parallel schur complement method for large-scale systems on distributed memory computers. *Applied Mathematical Modelling*, vol. 25, n10, 2001, pp. 873–886.
18. Lizé (B.). – *Résolution Directe Rapide pour les Éléments Finis de Frontière en Électromagnétisme et Acoustique : $\mathcal{H}$-Matrices. Parallélisme et Applications Industrielles.* – Thèse de PhD, Université Paris 13, 2014.
19. Raviart (P.) et Thomas (J.). – A mixed finite element method for 2-nd order elliptic problems. *In : Mathematical Aspects of Finite Element Methods*, éd. par Galligani (I.) et Magenes

(E.), pp. 292–315. – Springer Berlin Heidelberg, 1977.

20. Sauter (S. A.) et Schwab (C.). – *Boundary Element Methods*, pp. 183–287. – Berlin, Heidelberg, Springer Berlin Heidelberg, 2011.
21. Schauer (M.), Roman (J. E.), Quintana-Ortí (E. S.) et Langer (S.). – Parallel computation of 3-d soil-structure interaction in time domain with a coupled fem/sbfem approach. *Journal of Scientific Computing*, vol. 52, n2, Aug 2012, pp. 446–467.
22. Sebaso. – Jet engine airflow during take-off. – `https://commons.wikimedia.org/wiki/File:20140308-Jet_engine_airflow_during_take-off.jpg`.
23. Wang (A.), Vlahopoulos (N.) et Wu (K.). – Development of an energy boundary element formulation for computing high-frequency sound radiation from incoherent intensity boundary conditions. *Journal of Sound and Vibration*, vol. 278, n1-2, 2004, pp. 413 – 436.
24. Zhang (F.). – *The Schur complement and its applications*. – Springer Science & Business Media, 2006, volume 4.
25. Zhang (P.), Wu (T.) et Finkel (R.). – Parallel computation for acoustic radiation in a subsonic nonuniform flow with a coupled fem/bem formulation. *Engineering Analysis with Boundary Elements*, vol. 23, n2, 1999, pp. 139–153.
26. Zienkiewicz (O.) et Taylor (R.). – *The finite element method*. – McGraw-hill London, 1977, volume 3.

Appendix

Algorithm 1: Multi-solve algorithm for computing the Schur complement S based on Equation 5 and solving the target system (see Equation 1).

Function `Multi-solve` (A, b) **:**
  ▷ USING A SPARSE SOLVER :
  $A_{vv} \leftarrow LL^T$-`Factorization` (A_{vv})
  for $i = 1$ **to** n_{BEM}/n_c **do**
    ▷ Using the i^{th} block of columns of A_{sv}^T as right-hand side :
    $Y \leftarrow$ `TriangularSolve` $(A_{vv}, A_{sv_i}^T)$
    $Z \leftarrow A_{sv} \times Y$ ▷ GEMM
    $A_{ss_i} \leftarrow A_{ss_i} - Z$ ▷ AXPY
  $b_v \leftarrow$ `TriangularSolve` (A_{vv}, b_v)
  ▷ USING A DENSE SOLVER :
  $A_{ss} \leftarrow LL^T$-`Factorization` (A_{ss})
  $x_s \leftarrow$ `TriangularSolve` $(A_{ss}, b_s - A_{sv}b_v)$

Algorithm 2: Multi-factorization algorithm for computing the Schur complement S based on Equation 6 and solving the target system (see Equation 1).

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` $(A_{ss}, b_s - A_{sv}b_v)$

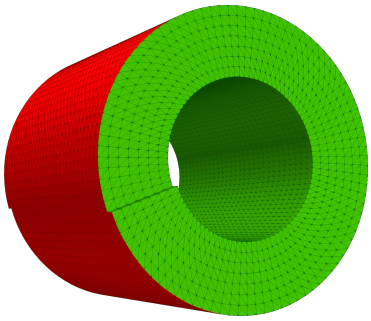

FIGURE 6 – Short pipe test case (length : 2 m, radius : 4 m) with BEM surface mesh in red and FEM volume mesh in green.

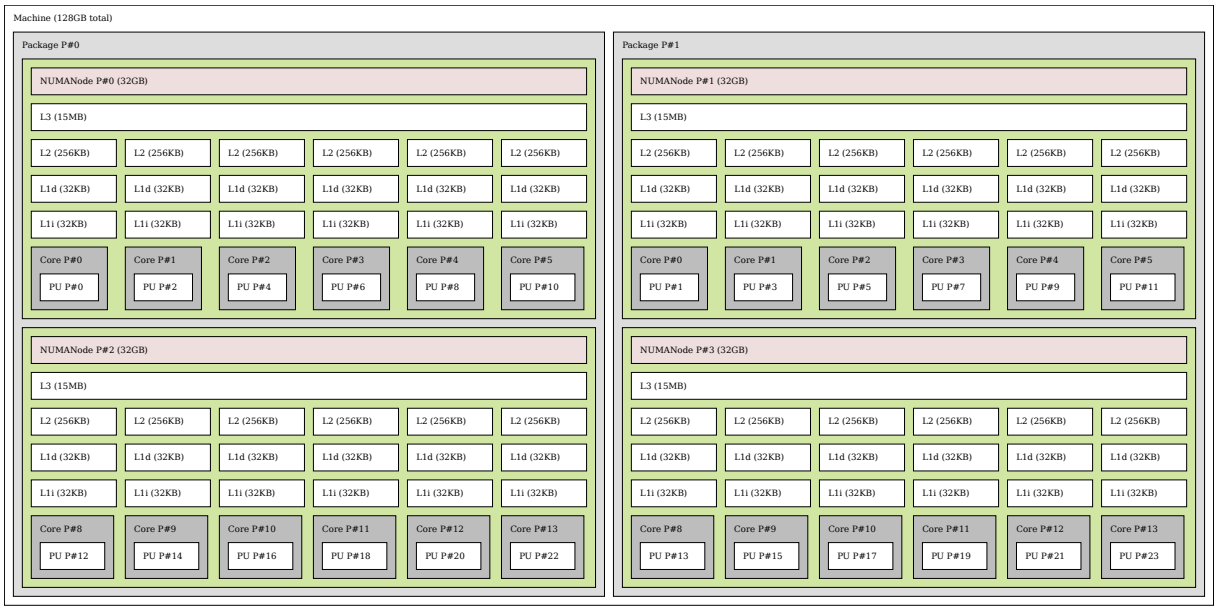

FIGURE 7 – Hardware configuration scheme of miriel nodes determined using the Linux Shell command `lstopo`.

Software	Commit/Version	Branch	Licensing
HMAT	912aa93	master	proprietary
MPF	432b53b	mf/devel	proprietary
SCAB (`test_FEMBEM`)	0c09d1f	mf/devel	proprietary
gcc [1]	9.3.0	N/A	open-source
OpenMPI	4.1.0	N/A	open-source
Intel(R) MKL	2019.1.144	N/A	proprietary
MUMPS [7]	5.2.1	N/A	open-source
StarPU [8]	1.3.7	N/A	open-source

TABLE 1 – Commit and version numbers of selected packages used in the experimental software environment of the study.

Total unknowns (N)	# BEM unknowns (n_{BEM})	# 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 2 – Counts of BEM and FEM unknowns in the target coupled FEM/BEM systems according to the overall unknown count.