Study of the processor and memory power consumption of coupled sparse/dense solvers

Emmanuel Agullo, Marek Felšöci, Amina Guermouche, Hervé Mathieu, Guillaume Sylvand, Bastien Tagliaro

RESEARCH REPORT

N° 9463

February 2022

Project-Team CONCACE, STORM, SED

ISSN 0249-6399 ISRN INRIA/RR--9463--FR+ENG

Study of the processor and memory power consumption of coupled sparse/dense solvers

Emmanuel Agullo*, Marek Felšöci†, Amina Guermouche‡,

Hervé Mathieu§, Guillaume Sylvand¶, Bastien Tagliaro‖

Project-Team CONCACE, STORM, SED

Research Report n° 9463 — February 2022 — 17 pages

* Inria Bordeaux Sud-Ouest (emmanuel.agullo@inria.fr)
† Inria Bordeaux Sud-Ouest (marek.felsoci@inria.fr)
‡ Inria Bordeaux Sud-Ouest, Université de Bordeaux (amina.guermouche@inria.fr)
§ Inria Bordeaux Sud-Ouest (herve.mathieu@inria.fr)
¶ Airbus Central R&T / Inria Bordeaux Sud-Ouest (guillaume.sylvand@airbus.com)
‖ Inria Bordeaux Sud-Ouest (bastien.tagliaro@inria.fr)

RESEARCH CENTRE
BORDEAUX – SUD-OUEST

200 avenue de la Vieille Tour
33405 Talence Cedex

Abstract: In the aeronautical industry, aeroacoustics is used to model the propagation of acoustic waves in air flows enveloping an aircraft in flight. This for instance allows one to simulate the noise produced at ground level by an aircraft during the takeoff and landing phases, in order to validate that the regulatory environmental standards are met. Unlike most other complex physics simulations, the method resorts to solving coupled sparse/dense systems. In a previous study, we proposed two classes of algorithms for solving such large systems on a relatively small workstation (one or a few multicore nodes) based on compression techniques. The objective of this study is to assess whether the positive impact of the proposed algorithms on time to solution and memory usage translates to the energy consumption as well. Because of the nature of the problem, coupling dense and sparse matrices, and the underlying solutions methods, including dense, sparse direct and compression steps, this yields an interesting processor and memory power profile which we aim to analyze in details.

Key-words: power consumption, FEM/BEM coupling, sparse and dense matrices, direct method, parallel solvers, energy_scope

Étude de la consommation de puissance du processeur et de la mémoire des solveurs couplés creux/denses

Résumé : Dans l'industrie aéronautique, l'aéroacoustique est utilisée pour modéliser la propagation d'ondes acoustiques à travers des courants d'air enveloppant un avion en vol. Par exemple, cela permet de simuler le bruit produit au niveau du sol par un avion pendant les phases de décollage et d'atterrissage afin de vérifier si les standards environnementaux réglementaires sont respectés. Contrairement à la plupart des simulations de problèmes physiques complexes, la méthode repose sur la solution de systèmes couplés creux/denses. Dans une précédente étude, nous avons proposé deux classes d'algorithmes pour résoudre ce type de grands systèmes linéaires sur une machine relativement petite (un ou peu de nœuds multi-cœurs) basés sur des techniques de compression. L'objectif de cette étude est de déterminer si l'impact positif de ces algorithmes sur l'utilisation de la mémoire se traduit également dans la consommation énergétique. Vu la nature du problème, le couplage de matrices creuses et denses ainsi que les méthodes de résolution sous-jacentes, y compris les étapes de compression creuse et dense, cela conduit à un profil de consommation de puissance du processeur et de la mémoire très intéressant que nous avons l'intention d'analyser en détails.

Mots-clés : consommation de puissance, couplage FEM/BEM, matrices creuses et denses, méthode directe, solveurs parallèles, energy_scope

Contents

1 Introduction

Aeroacoustics is the study of the coupling between acoustic phenomena and fluid mechanics. In the aeronautical industry, this discipline is used to model the propagation of acoustic waves in air flows enveloping an aircraft in flight. In particular, it allows one to simulate the noise produced at ground level by an aircraft during the takeoff and landing phases. Beyond its technical importance to validate that the regulatory environmental standards are met, it is also both a societal (ensuring noise reduction for alleviating the impact on health) and an economical (designing future planes that can get authorized to access suburban airports) challenge.

The approach adopted in this study consists in considering the airflow around the aircraft as uniform in almost all the space (in a zone modeled by Boundary Elements Method or BEM) except for the jet of the engines where the flow is considered as non-uniform (in a zone modeled by volume Finite Elements Method or FEM). The resulting linear system couples FEM and BEM, and therefore has both dense parts (from BEM) and sparse parts (from FEM). The coupled linear system, called FEM/BEM, has subsequently a singular composition that must be taken into account during its resolution. Moreover, in order to produce a result that is physically realistic, the number of unknowns can be extremely important, which makes the treatment of this system a computational challenge. In a previous study [3], we presented two classes of algorithms, namely the multi-solve and multi-factorization algorithms, for solving such systems. They rely on numerical compression techniques, which allows aircraft designers to process relatively large industrial test cases on a shared-memory workstation.

Until now, the dimensioning variables for such a calculation were the computation time, the consumed RAM and the disk occupation. The study of these quantities allowed us to understand the behavior of the software and to push its limits in order to handle larger cases. The

consideration of carbon footprint issues in industry in general and in computing centers in particular leads the research community and the industry to consider another physical dimension: the power consumption of computations. The objective of this paper is to study the power consumption of the solution of a coupled FEM/BEM linear system, and assess how the energy consumption of the processor and the memory varies with the computation time, the amount of memory used and with the multiple algorithmic choices possible at the solver level. We consider both shared-memory multicore machines and small clusters of such nodes, typically used for relatively large problems in aeroacoustics industry.

The rest of the paper is organized as follows. In Section 2, we further introduce the coupled FEM/BEM systems arising in aeroacoustics and then present the multi-solve and the multi-factorization algorithms from [3] we rely on to solve them. We also discuss related work on the analysis of the power consumption in the context of numerical simulation in general and numerical linear algebra in particular. The hardware, software and instrumentation setup we employ for the present study is introduced in Section 3. The power consumption of the multi-solve and multi-factorization algorithms on a shared-memory node is then analyzed in sections 4 and 5, respectively, before being prolonged in a multi-node context in Section 6. We conclude in Section 7.

2 Background

2.1 Coupled FEM/BEM systems arising in aeroacoustics

We are interested in the solution of very large linear systems of equations $Ax = b$ with the particularity of having both sparse and dense parts. Such systems appear in an industrial context when we couple two types of finite elements methods, namely the volume Finite Element Method (FEM) [22, 16] and the Boundary Element Method (BEM) [11, 25]. This coupling is used to simulate the propagation of acoustic waves around aircrafts (see Fig. 2, left). In the jet flow created by the reactors, the propagation media (the air) is highly heterogeneous in terms of temperature, density, flow, etc. Hence we need a FEM approach to compute acoustic waves propagation in it. Elsewhere, we approximate the media as homogeneous and the flow as uniform, and use BEM to compute the waves propagation. This leads [12, 13] to a coupled sparse/dense FEM/BEM linear system (1) with two groups of unknowns: x_v related to a FEM volume mesh v of the jet flow and x_s related to a BEM surface mesh s covering the surface of the aircraft as well as the outer surface of the volume mesh (see Fig. 2, right). The linear system $Ax = b$ to be solved may be more finely written as:

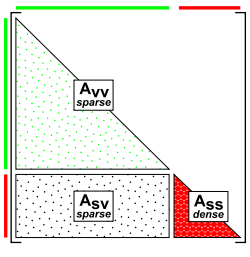

FIGURE 1: Partitionning and sparsity of A in (1).

$$\begin{bmatrix} A_{vv} & A_{sv}^T \\ A_{sv} & A_{ss} \end{bmatrix} \times \begin{bmatrix} x_v \\ x_s \end{bmatrix} = \begin{bmatrix} b_v \\ b_s \end{bmatrix}. \quad (1)$$

In (1), A is a 2×2 block 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 (see Fig. 1).

FIGURE 2: An acoustic wave (blue arrow) emitted by the aircraft's engine, reflected on the wing and crossing the jet flow. Real-life case (left) [23] and a numerical model example (right). The red surface mesh represents the aircraft's surface and the surface of the green volume mesh of the jet flow.

2.2 Multi-solve and multi-factorization algorithms

Various approaches may be employed to solve such coupled sparse/dense systems. In this work, we seek to solve (1) using a direct method. In a sparse context, direct methods are known to possibly consume a lot of memory due to a phenomenon referred to as *fill-in* [15] (zeros of the original matrix A become non-zeros in factorized matrix). On the other hand, when they fit in memory, they are extremely robust from a numerical point of view and represent a must-have in an industrial solution where they are commonly employed to solve moderate to relatively large problems.

In the present study, we consider the multi-solve and multi-factorization algorithms proposed in [3] (see also references therein for a discussion of other approaches of the literature). Their common principle consists in composing existing parallel sparse and dense methods on well chosen submatrices. Their main strength is to rely on state-of-the-art sparse and dense direct solvers and exploit their most advanced features such as compression techniques [17, 5, 21, 2] in an effort to lower the memory footprint and potentially reduce the computation time so as to process larger problems. In this section, we present the main algorithmic steps of both these methods. The objective is neither to motivate them nor to describe them in details (we refer the reader to [3] for that) but to provide a high-level view of the steps and their nature (such as whether they involve dense, sparse or compressed computation). Both methods must assemble the following dense matrix $S = A_{ss} - A_{sv}A_{vv}^{-1}A_{sv}^T$ associated with the A_{ss} block and referred to as the Schur complement.

2.2.1 Multi-solve algorithm.

Most sparse direct solvers do not provide an API to handle coupled sparse/dense systems and can process exclusively sparse systems. The multi-solve approach accommodates with this constraint by delegating only the A_{vv} block to the sparse direct solver. Using the latter, the A_{vv} block is factorized through a so-called *sparse factorization*. The A_{ss} block is handled by the dense direct solver. Because this block may not fully fit in memory, it is split into multiple vertical slices (see Fig. 3) which are assembled one by one, all the processing units tackling the same slice i at the same time. To compute such a slice S_i of A_{ss}, a slice A_{sv_i} is first processed through a *sparse solve* step of the sparse direct solver, yielding a dense temporary slice Y_i. The latter is multiplied by the sparse A_{sv} block. Then, we perform a final assembly $(A_{ss_i} - A_{sv}Y_i)$ to produce the dense S_i slice.

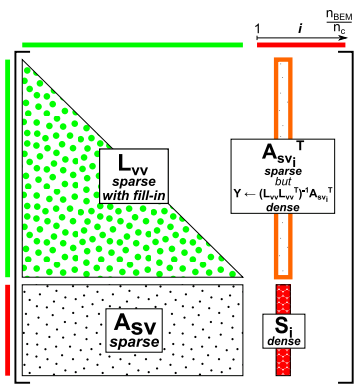

FIGURE 3: *baseline multi-solve.*

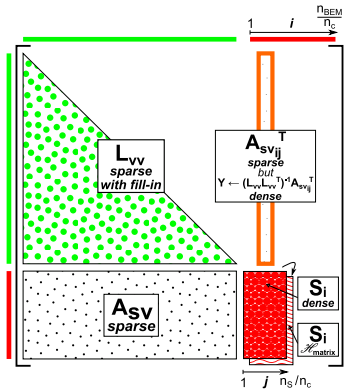

FIGURE 4: *compressed Schur multi-solve.*

In the *baseline multi-solve* case, the block S_i is kept dense. Conversely, in the *compressed Schur multi-solve* variant, it is compressed (through hierarchically low-rank techniques). Note that A_{ss_i} is initially compressed, but this operation implies a recompression of the block at each iteration of the loop on i. This is why this variant allows for computing multiple (typically 4 in the experiments below) slices S_i before compressing and assembling them (see Fig. 4).

2.2.2 Multi-factorization algorithm.

The multi-factorization algorithm is based on a more advanced usage of sparse direct methods consisting in delegating also the management of the dense A_{ss} block to the sparse direct solver. Only supported by a few fully-featured sparse direct solvers, this functionality (referred to as Schur) has the advantage of efficiently handling off-diagonal blocks thanks to the advanced combinatorial (such as management of the fill-in), numerical (such as low-rank compression) and computational (such as level-3 BLAS usage) features of modern sparse direct solvers when processing the off-diagonal A_{sv}^T and A_{sv} sparse-dense coupling parts (see [3] for more details). The computation of the Schur complement S in the *baseline multi-factorization* algorithm is not anymore computed by vertical slices but tile-wise. Computing a tile S_{ij} (see Fig. 5) amounts to form a temporary *non-symmetric* (except when $i = j$) submatrix W from A_{vv}, A_{sv_i} and $A_{sv_j}^T$ and call a *sparse factorization+Schur* step on W relying on the Schur feature of the sparse direct solver. This call returns the Schur complement block $X_{ij} = -A_{sv_i}(L_{vv}U_{vv})^{-1}A_{sv_j}^T$ associated with W. To determine S_{ij}, we perform a final assembly $(A_{ss_{ij}} + X_{ij})$. Due to a current limitation in the API of modern sparse direct solver (see extended discussion in [3]), the *sparse factorization+Schur* step involving W implies a re-factorization of A_{vv} in W at each iteration, although it does not change during the computation. Furthermore, the API of modern fully-featured sparse direct solvers (see once again discussion in [3]) only allows to retrieve the Schur complement itself as a non compressed dense matrix, even if compression occurs within the rest of processing.

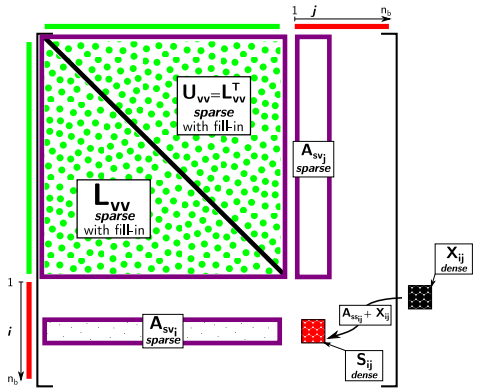

FIGURE 5: *baseline multi-factorization.*

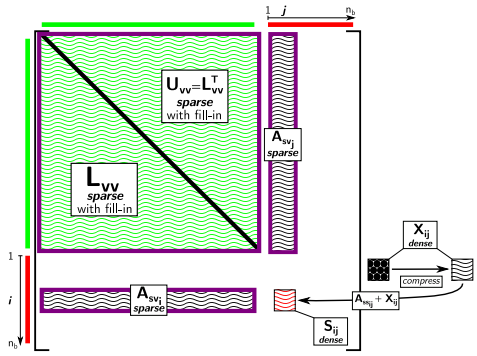

FIGURE 6: *compressed Schur multi-factorization.*

In the *compressed Schur multi-factorization* variant (see Fig. 6), we compress the X_{ij} Schur block into a temporary compressed matrix as soon as the sparse solver returns it. Hence, the final assembly step becomes a compressed assembly $A_{ss_{ij}} \leftarrow A_{ss_{ij}} + \texttt{Compress}(X_{ij})$. Like in the case of *compressed Schur multi-solve*, this operation implies a recompression of the initally compressed $A_{ss_{ij}}$.

2.3 Related work on studies of the power consumption of numerical algorithms

While many studies like [14, 24] analyze the power and energy consumption of various applications on different architectures, fewer studies focus on dense or sparse solvers. [10] presents the energy consumption of OpenMP runtime systems on three dense linear algebra kernels. [4] and [9] focus on sparse solvers. While the former studies the behavior of the Conjugate Gradient method on different CPU-only architectures, the latter focuses on sparse solvers on heterogeneous architectures. In [8], the authors present an energy and performance study of state-of-the-art sparse matrix solvers on GPUs. Note that many studies have been conducted regarding the improvement of the energy consumption of sparse or dense linear algebra algorithms [7, 1]. In this work, we focus on the analysis of an application coupling both sparse and dense techniques.

3 Experimental setup

We conducted an energy consumption study of the previously discussed algorithms for solving larger coupled sparse/dense FEM/BEM linear systems such as defined in (1). The multi-solve and multi-factorization algorithms (see Section 2.2) are implemented on top of the coupling of the sparse direct solver MUMPS [6] with either the proprietary scalapack-like dense direct solver SPIDO (for the *baseline* variants) or the hierarchical low-rank $\mathcal{H}$-matrix compressed solver HMAT [19] (for the *compressed* variants). In the following, we thus refer to these *baseline* and *compressed* couplings as to MUMPS/SPIDO and MUMPS/HMAT, respectively. For the experiments, we used a short pipe test case (see Fig. 7) yielding linear systems close enough to those arising from real life models (see Fig. 2) while relying on a reproducible example

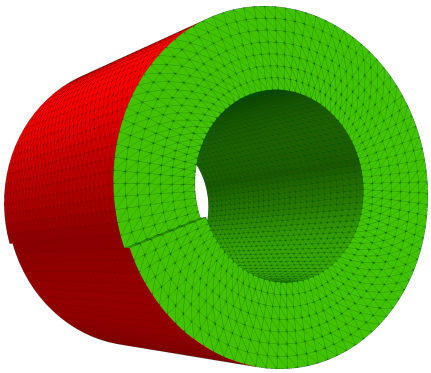

FIGURE 7: Short pipe test case (length: 2 m, radius: 4 m) with BEM surface mesh in red and FEM volume mesh in green.

(`https://gitlab.inria.fr/solverstack/test_fembem`) available for the scientific community. MUMPS and HMAT both provide low-rank compression and expose a precision parameter ϵ set to 10^{-3}. Low-rank compression in the sparse solver MUMPS is enabled for all the benchmarks presented in this paper. The solver test suite is compiled with GNU C Compiler (gcc) 9.4.0, OpenMPI 4.1.1, Intel(R) MKL library 2019.1.144, and MUMPS 5.2.1.

Our experiments were carried on the PlaFRIM platform (`https://plafrim.fr/`) where we used the `miriel` computing nodes equipped with 2 × 12-core Haswell Intel(R) Xeon(R) E5-2680 v3 @ 2.5 GHz with a Thermal Design Power (TDP) of 120 W, 128 GiB (5.3 GiB/core) RAM bank @ 2933 MT/s, an OmniPath 100 Gbit/s and a 10 Gbit/s Ethernet network links. Note that TDP represents the average power, in watts, the processor dissipates when operating at base frequency with all cores active under an Intel-defined, high-complexity workload. Finally, Turbo-Boost and Hyperthreading are disabled on all the nodes in order to ensure the reproducibility of the experiments.

We measure the energy consumption of our application with `energy_scope`. It is a software package dedicated to performing energy measurements and identifying energy profiles of HPC codes [20]. It consists of an acquisition and energy statistics delivering module running on the cluster and a post-processing and data analysis module running on a dedicated server. Measurements are performed at a user-defined frequency on both the processor and the RAM. We monitor also RAM usage and flop rate. To measure RAM usage, we use a Python script relying on the `/proc/[PID]/statm` file (the resident size field). Finally, for the measurement of the flop rate, we use the `likwid` [18] software tool. All the data have been acquired at a 1 Hz frequency. We assessed the potential overhead of using these three software probes on our application by running the tests with and without using the software probes. Results show an overhead under 5%.

4 Study of the multi-solve algorithm in shared-memory

At first, we study the evolution of power consumption (in Watts) with respect to execution time of the multi-solve algorithm (see Section 2.2) on a single computational node. We consider both the baseline MUMPS/SPIDO and further compressed MUMPS/HMAT coupling assessed on a coupled FEM/BEM linear system of 1,000,000 unknowns in total.

Regarding the *baseline multi-solve* algorithm relying on the MUMPS/SPIDO coupling, we set the size n_c of the $A_{sv_j}^T$ and S_i slices to 256 columns. For the *compressed Schur multi-solve* relying on the MUMPS/HMAT coupling, the size of the S_i and the size of the $A_{sv_j}^T$ slices are handled by two different parameters, n_S and n_c respectively. In this case, we set n_c to 256 and n_S to 1,024 columns so that compression is delayed until four slices involving $A_{sv_j}^T$ are completed. The choice of these values is motivated by our previous study of the multi-solve algorithm in [3]. In Fig. 8, for each solver coupling, the two plots at the top show the evolution of the CPU and RAM power consumption as well as the flop rate for the multi-solve algorithm. Then, the two other plots at the bottom of the figure, show the corresponding RAM usage evolution. The labels mark the alpha and the omega, i.e. the beginning and the end, of the most important computation phases.

In the case of multi-solve, the Schur complement computation dominates the execution time as well as the RAM usage. From the point of view of the computational intensity, illustrated by the flop rate, it is the opposite. The factorization phase of the dense Schur complement matrix is very short for the MUMPS/HMAT coupling thanks to the usage of low-rank compression. Nevertheless, as of the MUMPS/SPIDO coupling, this phase consists of a dense factorization which is a computationally intensive operation. Indeed, in this case, we reach the flop rate peak.

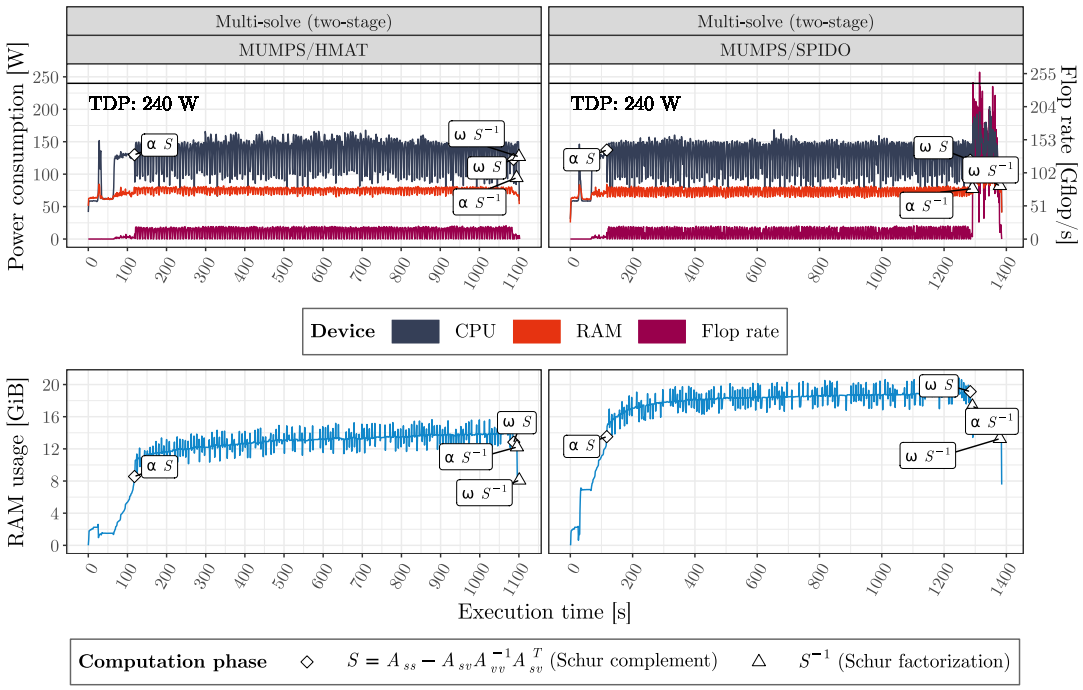

FIGURE 8: Power consumption, flop rate and RAM usage evolution of **multi-solve** for the MUMPS/SPIDO and the MUMPS/HMAT couplings on a FEM/BEM system with 1,000,000 unknowns. Parallel runs using 24 threads on single `miriel` node. α and ω mark the beginning and the end of a computation phase.

Fig. 9 is a zoom of Fig. 8 between the execution times 498 and 525 s, i.e. within the Schur complement computation phase. In this figure, we can observe cycles of high and low power consumption and flop rate. As of the RAM usage, the cycles are also present but less noticeable. Based on the labels, we can see that these cycles are almost entirely due to the *sparse solve* operation involved in the computation of each of the Schur complement blocks S_i (see Section 2.2).

Fig. 10 finally compares the total energy consumption (in Joules), the total execution time and the peak RAM usage, once again of both variants (baseline MUMPS/SPIDO and further compressed MUMPS/HMAT) of the multi-solve algorithm. We consider three coupled FEM/BEM systems of a total of 1,000,000, 3,000,000 and 5,000,000 unknowns, respectively. The results confirm, as one may expect, that the energy consumption, the execution time as well as the peak RAM usage rise with increasing size of the linear system. Moreover, the results show that the compressed MUMPS/HMAT variant (the one that also performs compression within the Schur) consumes less energy, in addition to being faster. This is an interesting result from an industrial point of view, further motivating the usage of low-rank compression techniques.

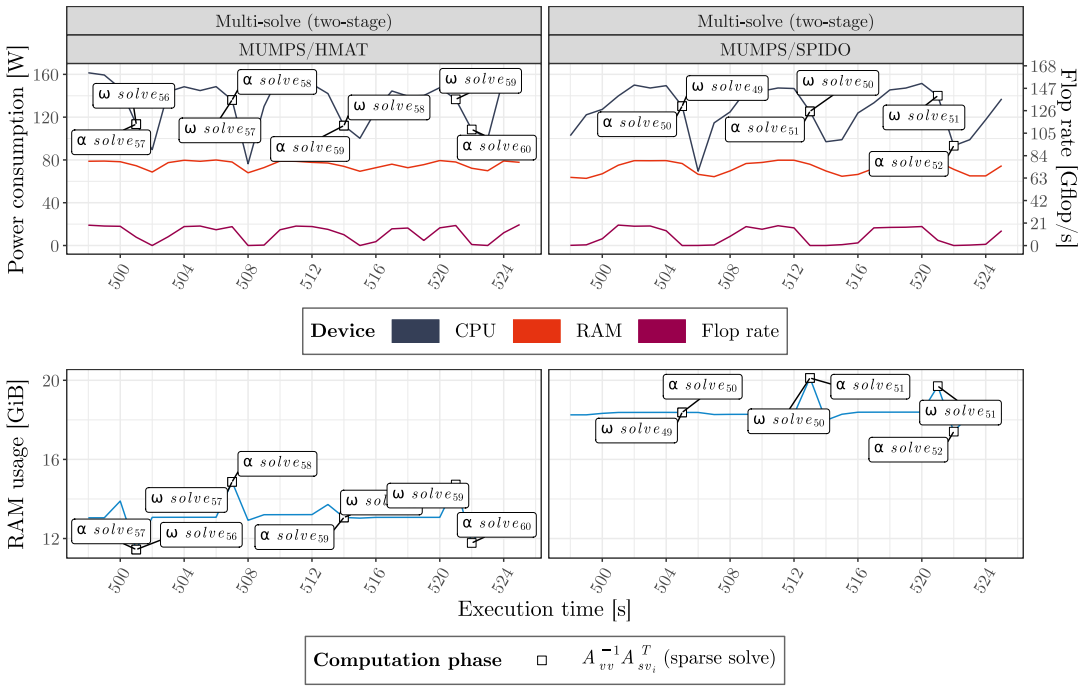

FIGURE 9: Zoom on Fig. 8 between the execution times 498 and 525 s. The Y-axis has been adapted to the extremum values in the selected time span.

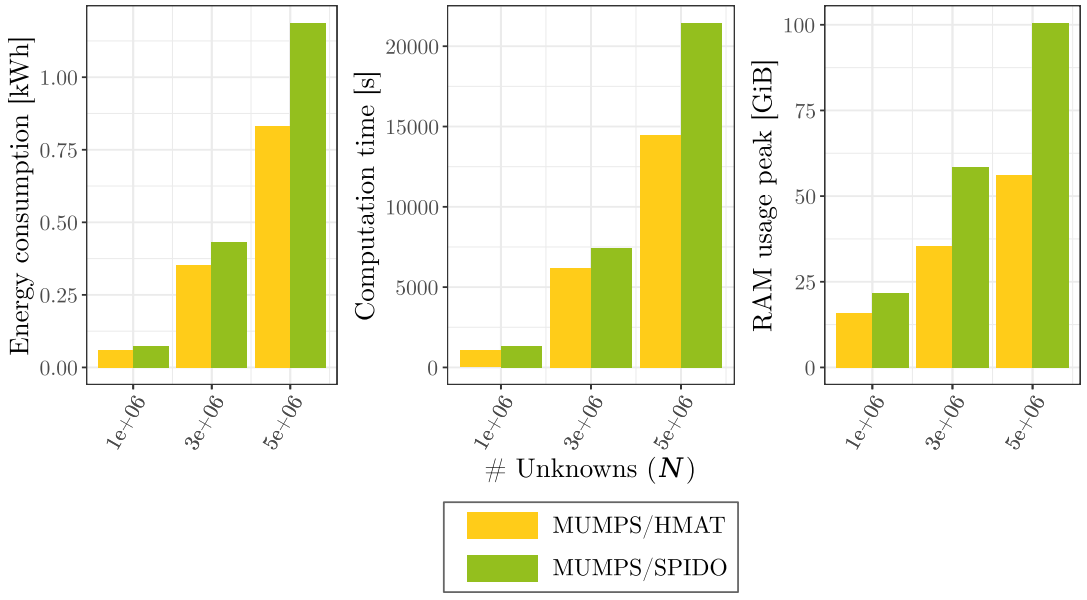

FIGURE 10: Total energy consumption, computation time and peak RAM usage of **multi-solve** for the MUMPS/SPIDO and the MUMPS/HMAT couplings on FEM/BEM systems with 1,000,000, 3,000,000 and 5,000,000 unknowns. Parallel runs using 24 threads on single `miriel` node.

5 Study of the multi-factorization algorithm in shared-memory

We now consider the multi-factorization algorithm. For the 1,000,000 unknowns test case (Fig. 11), we have set the number of Schur complement block rows and columns n_b to 3 for both baseline and further compressed solver couplings. As for multi-solve, in the case of multi-factorization, the execution time is dominated by the Schur complement computation phase. With $n_b = 3$, we have a total of 6 Schur complement blocks S_{ij} to compute. We can identify the moment of computation of each S_{ij} thanks to the apparent cycles of high and low power consumption and flop rate and especially of RAM usage. Here, the Schur complement computation phase again consumes most of the RAM but is not the most computationally intensive part of the algorithm. The peak power consumption and flop rates are met during the dense factorization of S in case of the baseline MUMPS/SPIDO coupling.

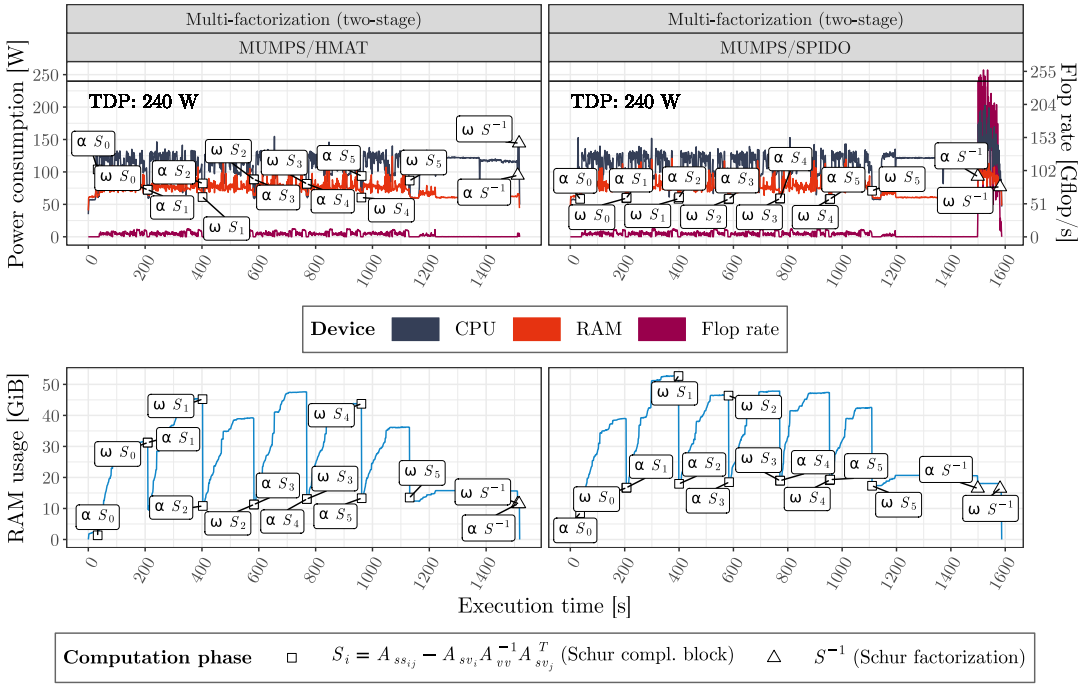

FIGURE 11: Power consumption, flop rate and RAM usage evolution of **multi-factorization** for the MUMPS/SPIDO and the MUMPS/HMAT couplings on a FEM/BEM system with 1,000,000 unknowns. Parallel runs using 24 threads on single `miriel` node. α and ω mark the beginning and the end of a computation phase.

Fig. 12 compares the total energy consumption (in Joules), the total execution time and the peak RAM usage of the multi-factorization algorithm with problems of 1,000,000, 1,500,000 and 2,000,000 total unknowns. We can draw the same conclusion as in the case of multi-solve.

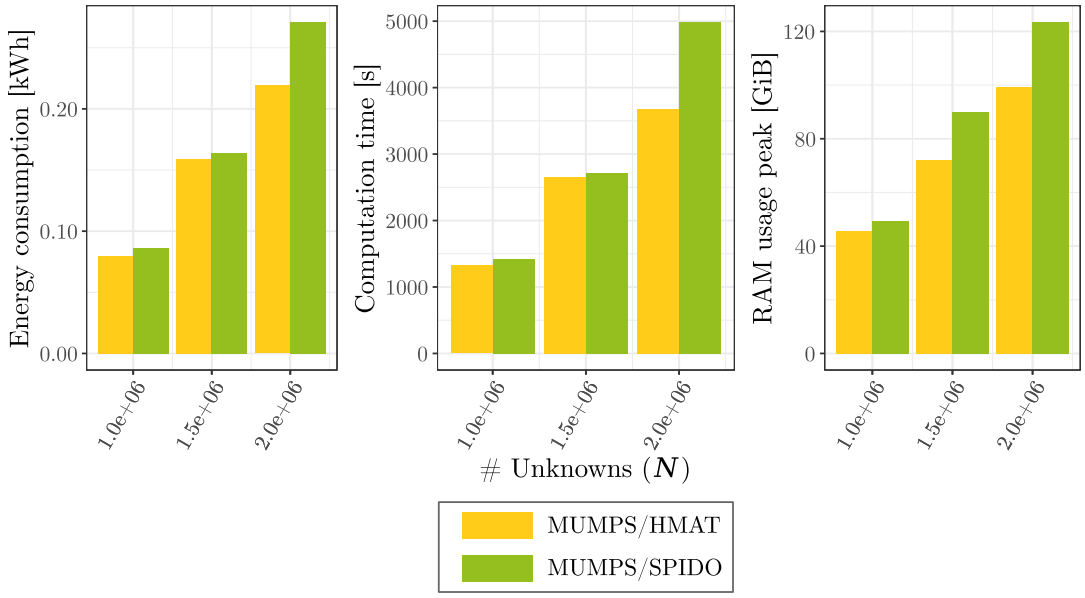

FIGURE 12: Total energy consumption, computation time and peak RAM usage of **multi-factorization** for the MUMPS/SPIDO and the MUMPS/HMAT couplings on FEM/BEM systems with 1,000,000, 1,500,000 and 2,000,000 unknowns. Parallel runs using 24 threads on single `miriel` node.

6 Multi-node study

We now consider a platform composed of four computational nodes and assess a coupled system of 2,000,000 total unknowns.

6.1 Multi-solve algorithm

Fig. 13 shows the processor and RAM power consumption over all four monitored nodes for both multi-solve variants. The results show that the power consumption in a distributed parallel test case evolves similarly to the single node test case (see Section 4). Similarly, the RAM usage follows the pattern of the single node test case.

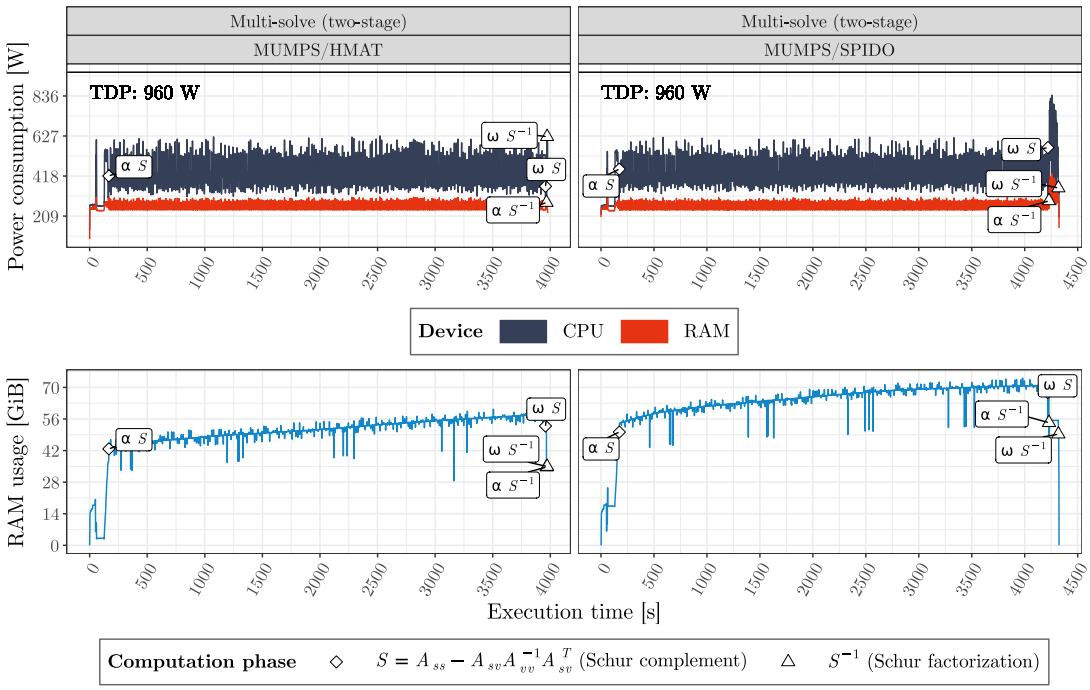

FIGURE 13: Power consumption, flop rate and RAM usage evolution of **multi-solve** for the MUMPS/SPIDO and the MUMPS/HMAT couplings on a FEM/BEM system with 2,000,000 unknowns. Distributed parallel runs using a total of 96 threads and 4 `miriel` nodes. α and ω mark the beginning and the end of a computation phase.

6.2 Multi-factorization algorithm

Fig. 14 shows the behavior of the multi-factorization algorithm. The evolution of RAM usage of MUMPS/HMAT follows the pattern of the single node test case (see Section 5). Regarding the power consumption, the high and low cycles corresponding to the computation of different Schur complement blocks differ considerably compared to the single-node test case. At the beginning of each cycle, there is a peak but the consumption falls down long before the end of the *sparse factorization+Schur* step. This may indicate an under-optimized usage of this routine in a parallel distributed environment. Interestingly, this study allowed us to potentially identify a bottleneck in our usage of the distributed-memory parallelization of a key component of our solver stack, which we had not identified in previous performance-only studies we conducted, showing the interest of such multi-metric profiles beyond the assessment of the overall energy consumption.

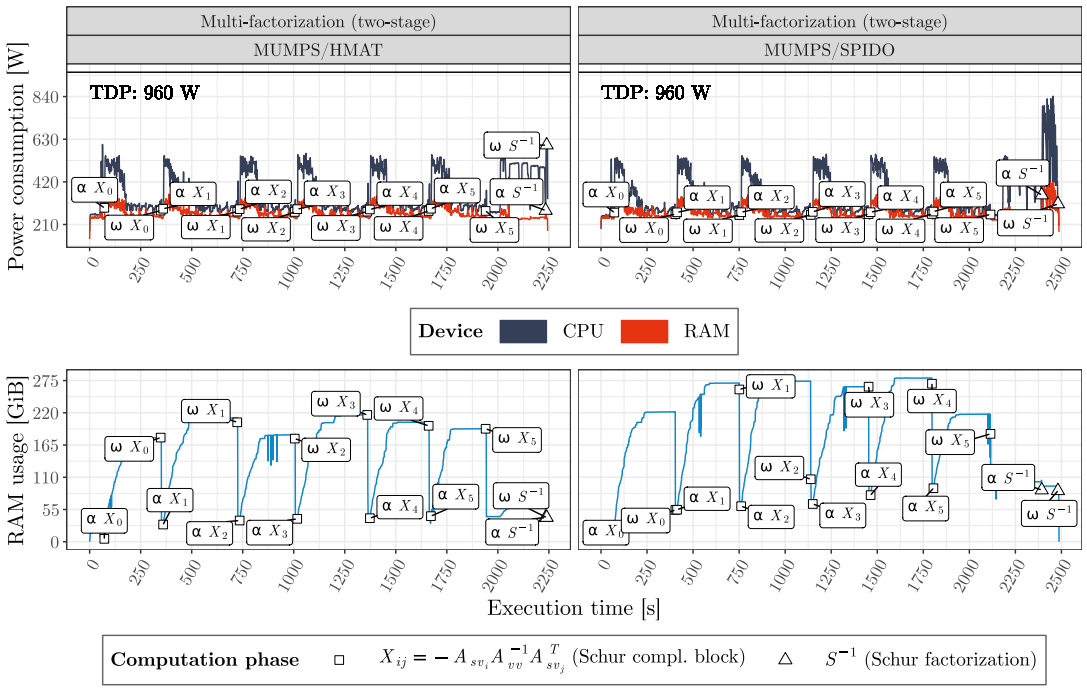

FIGURE 14: Power consumption, flop rate and RAM usage evolution of **multi-factorization** for the MUMPS/SPIDO and the MUMPS/HMAT couplings on a FEM/BEM system with 2,000,000 unknowns. Distributed parallel runs using a total of 96 threads and 4 `miriel` nodes. α and ω mark the beginning and the end of a computation phase.

7 Conclusion and perspectives

We have studied the energetic profile of a complex HPC application, involving dense, sparse, and compressed operations. The study confirmed that the further compressed algorithms (MUMPS/HMAT) are also worth from an energetic perspective. The profiles of the processor and memory power together with the memory usage and flop rate allowed us to have a more comprehensive understanding of the behavior of the application, up to the point that we identified a potential improvement in our usage of the Schur functionality of the sparse direct solver.

Acknowledgements

This work was supported by the 'Projet Région Nouvelle-Aquitaine 2018-1R50119 « HPC scalable ecosystem »'.

References

[1] S. Abdelfattah, A. Haidar, S. Tomov, and J. Dongarra, *Analysis and Design Techniques towards High-Performance and Energy-Efficient Dense Linear Solvers on GPUs*, IEEE Transactions on Parallel and Distributed Systems, 29 (2018), pp. 2700–2712.

[2] E. Agullo, A. Falco, L. Giraud, and G. Sylvand, *Vers une factorisation symbolique hiérarchique de rang faible pour des matrices creuses*, in Conférence d'informatique en Parallélisme, Architecture et Système (ComPAS'17), Sophia Antipolis, France, June 2017.

[3] E. Agullo, M. Felšöci, and G. Sylvand, *Direct solution of larger coupled sparse/dense linear systems using low-rank compression on single-node multi-core machines in an industrial context*, Research Report 9453 (accepted at IPDPS 2022), Inria Bordeaux Sud-Ouest, Feb. 2022.

[4] J. I. Aliaga, H. Anzt, M. Castillo, J. C. Fernández, G. León, J. Pérez, and E. S. Quintana-Ortí, *Unveiling the Performance-Energy Trade-off in Iterative Linear System Solvers for Multithreaded Processors*, Concurrency and Computation : Practice and Experience, 27 (2015), p. 885–904.

[5] 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.

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

[7] H. Anzt, J. Dongarra, G. Flegar, N. J. Higham, and E. S. Quintana-Ortí, *Adaptive precision in block-Jacobi preconditioning for iterative sparse linear system solvers*, Concurrency and Computation: Practice and Experience, 31 (2019), p. e4460. e4460 cpe.4460.

[8] H. Anzt, S. Tomov, and J. Dongarra, *Energy Efficiency and Performance Frontiers for Sparse Computations on GPU Supercomputers*, in Proceedings of the Sixth International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM '15, New York, NY, USA, 2015, Association for Computing Machinery, p. 1–10.

[9] H. Anzt, S. Tomov, and J. Dongarra, *On the performance and energy efficiency of sparse linear algebra on GPUs*, The International Journal of High Performance Computing Applications, 31 (2017), pp. 375–390.

[10] J. ao Vicente Ferreira Lima, I. Raïs, L. Lefèvre, and T. Gautier, *Performance and energy analysis of OpenMP runtime systems with dense linear algebra algorithms*, The International Journal of High Performance Computing Applications, 33 (2019), pp. 431–443.

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

[12] F. Casenave, *Méthodes de réduction de modèles appliquées à des problèmes d'aéroacoustique résolus par équations intégrales*, PhD thesis, Université Paris-Est, 2013.

[13] 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.

[14] J. Charles, W. Sawyer, M. F. Dolz, and S. Catalán, *Evaluating the Performance and Energy Efficiency of the COSMO-ART Model System*, Comput. Sci., 30 (2015), p. 177–186.

[15] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct methods for sparse matrices*, Oxford University Press, 2017.

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

[17] P. Ghysels, X. Li, F.-H. Rouet, S. Williams, and A. Napov, *An Efficient Multicore Implementation of a Novel HSS-Structured Multifrontal Solver Using Randomized Sampling*, SIAM Journal on Scientific Computing, 38 (2015).

[18] T. Gruber, J. Eitzinger, G. Hager, and G. Wellein, *LIKWID*. `https://doi.org/10.5281/zenodo.5752537`, 12 2021. This research has been partially funded by grants: BMBF 01IH13009 and BMBF 01IH16012C.

[19] B. Lizé, *Résolution Directe Rapide pour les Éléments Finis de Frontière en Électromagnétisme et Acoustique : $\mathcal{H}$-Matrices. Parallélisme et Applications Industrielles.*, PhD thesis, Université Paris 13, 2014.

[20] H. Mathieu, *Energy Scope: a tool for measuring the energy profile of HPC and AI applications*. `https://jcad2021.sciencesconf.org/data/Herve_Mathieu_energy_scope.pdf`, 2021.

[21] G. Pichon, E. Darve, M. Faverge, P. Ramet, and J. Roman, *Sparse supernodal solver using block low-rank compression: Design, performance and analysis*, Journal of Computational Science, 27 (2018), pp. 255–270.

[22] 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.

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

[24] L. Solis-Vasquez, D. Santos-Martins, A. Koch, and S. Forli, *Evaluating the Energy Efficiency of OpenCL-accelerated AutoDock Molecular Docking*, in 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), 2020, pp. 162–166.

[25] 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.

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