Computational Geosciences (2019) 23:1237–1258
https://doi.org/10.1007/s10596-019-09883-y

ORIGINAL PAPER

Efficient use of sparsity by direct solvers applied to 3D controlled-source EM problems

Patrick R. Amestoy[1,2] · Sébastien de la Kethulle de Ryhove[3,4] · Jean-Yves L'Excellent[2,5] · Gilles Moreau[5] · Daniil V. Shantsev[6]

Received: 10 January 2019 / Accepted: 20 August 2019 / Published online: 13 September 2019

Abstract

Controlled-source electromagnetic (CSEM) surveying becomes a widespread method for oil and gas exploration, which requires fast and efficient software for inverting large-scale EM datasets. In this context, one often needs to solve sparse systems of linear equations with a *large* number of *sparse* right-hand sides, each corresponding to a given transmitter position. Sparse direct solvers are very attractive for these problems, especially when combined with low-rank approximations which significantly reduce the complexity and the cost of the factorization. In the case of thousands of right-hand sides, the time spent in the sparse triangular solve tends to dominate the total simulation time, and here we propose several approaches to reduce it. A significant reduction is demonstrated for marine CSEM application by utilizing the sparsity of the right-hand sides (RHS) and of the solutions that results from the geometry of the problem. Large gains are achieved by restricting computations at the forward substitution stage to exploit the fact that the RHS matrix might have empty rows (*vertical sparsity*) and/or empty blocks of columns within a non-empty row (*horizontal sparsity*). We also adapt the parallel algorithms that were designed for the factorization to solve-oriented algorithms and describe performance optimizations particularly relevant for the very large numbers of right-hand sides of the CSEM application. We show that both the operation count and the elapsed time for the solution phase can be significantly reduced. The total time of CSEM simulation can be divided by approximately a factor of 3 on all the matrices from our set (from 3 to 30 million unknowns, and from 4 to 12 thousands RHSs).

Keywords Controlled-source electromagnetics (CSEM) · Marine electromagnetics · Numerical modeling · Direct solver · Multiple sparse right-hand sides

Mathematics Subject Classification (2010) 15A06 · 15A23 · 65F05 · 65F50 · 65Y05 · 65Z05 · 68U20 · 68W10 · 78A25 · 86-04 · 86-08 · 86A20 · 86A22

1 Introduction

It was demonstrated in 2002 that marine controlled-source electromagnetic (CSEM) method could be used to detect offshore hydrocarbon reservoirs [15]. Over the years, the CSEM method has become an established tool for oil and gas exploration [11], and the technology development keeps going at a high pace [19]. Successful interpretation of the growing volume of geophysical CSEM data, including also land EM data [29], requires efficient large-scale 3D electromagnetic (EM) modeling algorithms.

Among various approaches to handle 3D EM problems, the most popular is to solve a sparse linear system of frequency-domain Maxwell equations built using finite-difference or finite-element methods [8, 10]. Recent applications of the Gauss-Newton inversion algorithm to large-scale marine CSEM problems indicate that it is very efficient and will likely become the standard inversion approach in the nearest future [23]. The Gauss-Newton method requires that the linear system is solved for all transmitter positions in a given survey, often resulting in several thousands of right-hand sides. The system of linear equations then takes the form $\mathbf{MX} = \mathbf{S}$, where $\mathbf{M}$ is a sparse symmetric matrix of size $n \times n$, while the RHS matrix $\mathbf{S}$

✉ Jean-Yves L'Excellent
Jean-Yves.L.Excellent@mumps-tech.com

Extended author information available on the last page of the article.

and solution matrix $\mathbf{X}$ are of size $n \times m$. Here the system size n can be up to several millions, while the number of right-hand sides m is up to several thousands.

Such systems can be solved with iterative methods which in general are relatively cheap in terms of memory and computational requirements, but may have convergence issues [8, 10]. In comparison with direct solvers, their cost increases quickly with the number of right-hand sides, i.e., with the number of EM sources. With direct methods, the matrix $\mathbf{M}$ is factored as a product $\mathbf{LDL^T}$, where $\mathbf{L}$ and $\mathbf{D}$ are respectively lower triangular and diagonal matrices. Then, for a given set of right-hand sides $\mathbf{S}$, one has to perform the so-called *solve phase* via a forward substitution solving a lower triangular system ($\mathbf{LY} = \mathbf{S}$), followed by a diagonal resolution ($\mathbf{DZ} = \mathbf{Y}$) and finally a backward substitution ($\mathbf{L^T X} = \mathbf{Z}$).

Direct methods are numerically robust. They are well suited to multi-source simulations since once the matrix factorization is performed, only the solve phase needs be applied on all the right-hand side vectors. The complexity of the sparse matrix factorization phase can be a bottleneck for large 3D problems, for which the number of floating-point operations scales as $O(n^2)$ (see [16]). The number of nonzero entries in the factors, which also defines the complexity of the solve phase, scales as $O(n^{4/3})$. However, in the context of CSEM applications, it has been shown [27] that using block low-rank (BLR) format and related approximations significantly improves the performance of the direct approach and reduces the factorization complexity from $O(n^2)$ to $O(n^{4/3})$ for a simple BLR format [1]. Sparse direct methods rely on a sequence of partial factorizations of dense matrices F referred to as fronts in the remainder of this paper. Although the matrix F cannot in general be expressed as the low-rank product of two matrices, the matrix F can be reordered and partitioned in a simple flat 2D blocking format. Using this so-called BLR format, it has been proved in [1] that in many applications (e.g., those coming from the discretization of elliptic partial differential equations) the off-diagonal blocks of each F matrix can be approximated by the product of two matrices of low rank. This low-rank representation is then used to reduce the complexity of the factorization. A very nice feature of this approach is that the accuracy of the approach relies on a unique numerical threshold, so-called ϵ_{BLR}, given by the user that controls the accuracy of each low-rank representation.

Thanks to the improvements of the factorization phase due to low-rank compression (further improved in [2] with respect to [27]), and since CSEM modeling involves thousands of right-hand sides, the time needed to perform the complete CSEM simulation becomes largely dominated by the solve phase of the direct solver. Indeed, the complexity of the solve phase, $O(m \times n^{4/3})$, becomes significant compared to that of a low-rank factorization.

To improve the performance of the solve phase with many right-hand sides, we first explain how to exploit the sparse structure of the CSEM source matrix, which is essentially defined by positions of transmitters in the 3D domain. We also exploit the solution sparsity, i.e., the fact that the solution does not need to be computed in the whole geometrical domain. It was shown in [17] that the nonzero structure of the matrix $\mathbf{Y}$ resulting from the forward substitution can be predicted, and exploited to limit the number of operations. This was also referred to as *tree pruning* in [28]. In the context of computing selected entries of the inverse of a matrix [4], the notion of intervals combined to an appropriate column permutation of the right-hand sides was introduced. We will explain how such techniques can be applied or adapted to the context of CSEM simulations.

In this paper, we also introduce several new algorithms and techniques to improve the performance of the solve phase on modern parallel architectures. In a distributed memory parallel environment, the performance of direct solvers strongly depends on how the computational tasks are mapped onto the computer nodes. Mapping algorithms control the equilibration of the work between processors and are typically driven by metrics from the factorization phase. We show that using workload metrics from the solve phase improves the overall performance of the CSEM simulation. Finally, to enhance arithmetic intensity and parallelism, large blocks of right-hand sides must be processed simultaneously. For this approach to be efficient, we show that locality of computations should be improved during the solve phase, especially when several threads are used within each distributed memory process (MPI process). This corresponds to an hybrid distributed-multithreaded setting, well adapted to the clusters of multicore processors that we target in this type of applications.

This paper is organized as follows. In Section 2, we describe our frequency-domain finite-difference EM modeling approach in the context of nested dissection, and focus on the structure of the right-hand sides. We also describe the test problems and emphasize the cost of the solve phase of a direct solver based on a multifrontal approach [13, 14]. The proposed algorithms are more general and could also be applied to other sparse direct methods. A brief background on direct solvers, with a focus on the solve phase is then provided. In Section 3, we explain how the sparse structure of the right-hand sides (RHS) may influence the solve phase and can be used to reduce the amount of computations. In Section 5, we discuss parallel aspects of the solve phase. First, we propose strategies to balance the workload for the solve phase. Then,

we show that RHS sparsity and parallelism are contradictory objectives and propose ways to group RHS columns together to recover some parallelism. While RHS sparsity can only be exploited during the forward substitution, we show in Section 4 that the same ideas can be transposed to the backward substitution, leading to further computational gains due to solution sparsity. Section 6 studies and illustrates the effects of each of the proposed algorithms on the performance of the solve phase. The global results are summarized in Section 6.3, also showing a comparison of the direct approach and a conventional iterative approach.

2 Background and motivations

2.1 Finite-difference electromagnetic modeling

The frequency-domain Maxwell equations in the conductive earth in a presence of a current source $\mathbf{J}$ can be approximated as follows:

$$\nabla \times \nabla \times \mathbf{E} - i\omega\mu\bar{\sigma}\mathbf{E} = i\omega\mu\mathbf{J}, \quad (1)$$

where $\mathbf{E}$ is the electric field, $\bar{\sigma}$ is the conductivity tensor, μ is the magnetic permeability, and ω is the frequency. Using finite differences on a grid of size $N = N_x \times N_y \times N_z$ corresponding to the discretization of the physical domain, the electric field has three components E_x, E_y, and E_z at each grid point and can be approximated by solving linear systems of the form $\mathbf{MX} = \mathbf{S}$, where $\mathbf{M}$ is a sparse matrix of order $n = 3N$ and $\mathbf{S}$ results from the right-hand side in Eq. 1. $\mathbf{M}$ can easily be made symmetric. Let us now discuss the properties of the RHSs and of the solution that result from the geometry of marine CSEM surveys, and the inversion approaches used to analyze CSEM data.

The CSEM receivers are typically placed at the sea-floor in a regular grid with 1–3-km spacing, while the transmitter is towed above the receiver lines. To achieve illumination of subsurface with different transmitter orientations, two orthogonal directions of towlines are often chosen. A schematic picture of such a CSEM survey outline is presented in Fig. 1a, where receiver locations are indicated with triangles. Circles along the towlines indicate transmitter locations: it is usually assumed that distinct transmitter positions are spaced by 100 m. Since the transmitter is moving, while seabed receivers are fixed, the number of transmitters n_t is much larger (by 1–2 orders of magnitude) than the number of receivers n_r.

The number of right-hand sides is determined by the inversion algorithm used to analyze CSEM data. The

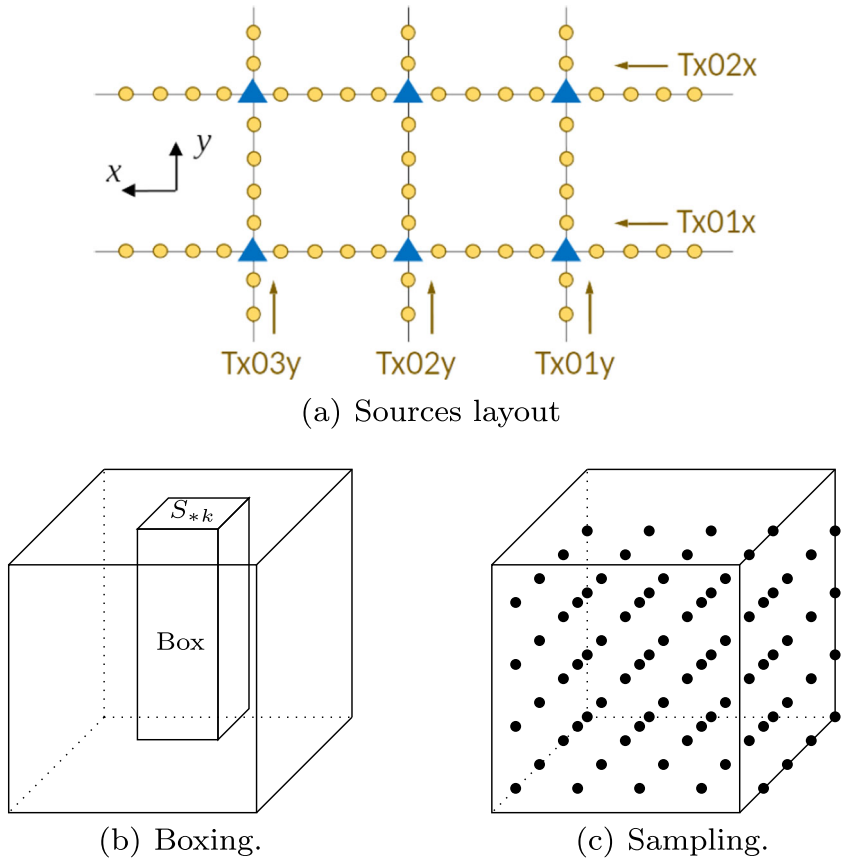


Fig. 1 **a** Schematic example of a CSEM survey with locations of receivers (triangles) and transmitter positions (circles). **b** Entries of interest in the solution are those inside a "box" centered around the corresponding source S_{*k}. **c** Entries of the solution may also be limited to a subset of nodal points uniformly distributed in the domain

gradient-based (BFGS) scheme [31] is relatively cheap: at each iteration one needs to solve a linear system of equations for source terms placed only at the receiver positions (due to reciprocity). In this paper, we will however focus on the more powerful Gauss-Newton scheme that is expected to soon become the prevailing inversion method [23]. In the Gauss-Newton method, the sources should also be placed at each transmitter position, i.e., the total number of RHSs is becoming much larger since $n_t \gg n_r$. Note also that the transmitter is usually towed within 30–100 m above the seafloor. Therefore, all RHSs (due to both transmitters and receivers) belong to a narrow depth interval near the seafloor—the property we shall utilize later in the article.

The right-hand sides are usually very sparse since they describe a source term that is localized in space. A point transmitter is often represented by placing source terms at $2 \times 2 \times 2 = 8$ nearest nodes in 3D problems, i.e., a RHS will have only 8 nonzero elements. In marine CSEM, a horizontal electric-dipole transmitter is often an extended antenna of $\sim$ 300 m length, rather than a point. In that case, the number of nonzero elements will be slightly larger (e.g., 16 or 24), but this complication will have only minor effect on our results, thus for the sake of simplicity we shall stick to considering point sources with 8 nonzero elements in RHSs.

The initial ordering of RHSs (also defining the order of the columns in $\mathbf{X}$) usually reflects the transmitter trajectory. In this article, the ordering of transmitter positions obeys the following simple rule (see Fig. 1a). We start with towline Tx01x, and there go over all transmitter positions in the $\mathbf{x}$ direction (see Fig. 1a). Then we switch to towline Tx02x and follow the same ordering, and so on until we reach the last $\mathbf{x}$-directed towline. Then we switch to $\mathbf{y}$-directed towlines, starting with Tx01y, and for each of them go over all transmitter positions in the direction of $\mathbf{y}$-axis. As we shall see below, this continuous ordering of RHSs is not optimal for the solver performance, and considerable gains can be achieved by appropriate reorderings. Strictly speaking, in the Gauss-Newton scheme, one also has to handle right-hand sides related to receiver positions. However, their number is much smaller. Therefore, for the sake of simplicity, we have not included them in the analysis below.

Only a subset of entries in the solution $\mathbf{X}$ usually needs to be computed when inverting marine CSEM data. For each column k and source vector S_{*k}, only the entries in a box with a square section and centered around S_{*k} are needed (see Fig. 1b). The box excludes the top of the domain since it corresponds to the air layer for which the resistivity is known. The box also excludes the water layer since the water conductivity is usually measured during the CSEM survey. The EM fields decay with increasing offset between transmitter and receiver and eventually drop below the noise level. We shall assume that the maximum offset for CSEM data is 12.5 km and therefore the lateral extent of the box will be 25 km $\times$ 25 km. All the regions beyond this box, in particular, the perfectly matched layers at the edges, can be excluded from the computed solution. Depending on the problem, the box may represent approximately one half of the whole computational domain.

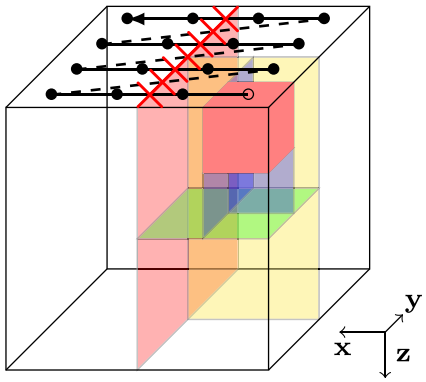
The CSEM Gauss-Newton inversion is an ill-posed problem. It therefore requires strong regularization that typically favors smooth solutions. Another way to make inversion more robust as well as faster is to reduce the number of inversion parameters. Since the CSEM method resolution decreases with depth, it is common to use fewer inversion parameters in deeper formation layers. As a result, the solution $\mathbf{X}$ in some regions may be required for a coarser sampling than the grid used to build the system matrix. In Section 6.1.2, we assume that the solution could be required on a uniformly distributed subset of nodal points (see Fig. 1c), where only every 20th or every 100th point is included into the subset.

2.2 Impact of the source structure

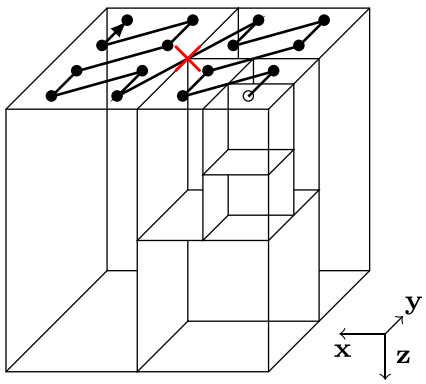
In this section, we relate the sparsity in the source vectors $\mathbf{S}$ to geometric properties of the underlying application and provide some preliminary intuitions on how sparsity will be used to reduce the solver complexity. The exploitation of sparsity in the solve phase will then be the object of Sections 3 and 4, for the forward and backward substitution, respectively.

The application of direct solvers to linear EM systems built on finite-difference methods is often illustrated by a hierarchical domain decomposition based on nested dissection [16], where the mesh is divided into subdomains and separators. A separator can be defined as a set of nodal points which splits the domain (or a given subdomain) into two balanced and *disjoint* subdomains. In a regular 3D grid such as the one represented in Fig. 2, the separator shapes are planes with normals in the $\mathbf{x}-$, $\mathbf{y}$-, or $\mathbf{z}$-directions. In Fig. 2a, we illustrate using different colors the separator planes that define the red cube in the upper right corner of the domain.

Since the source term is geometrically localized, all nonzero elements of a source vector usually belong to the same subdomain. The nested dissection creates independence between disjoint subdomains. Hence, the source contribution during the forward substitution will not affect any other subdomain. In other words, the computation of a column of $\mathbf{Y}$ for a given source is only concerned by the associated subdomain and the *top separators*. In



(a) Initial source order.



(b) Optimized source order.

Fig. 2 **a** A computational 3D domain where sources (marked with • or ∘) are connected with a line representing the transmitter trajectory. A hierarchical domain decomposition with 2D plane-shaped separators is applied. The top separators leading to the red subcube containing the first source (∘) are marked with different colors. The red crosses show where the transmitter trajectory crosses the main separator. **b** The line connecting source positions indicates a new source ordering that minimizes the crossings of top separators

Fig. 2a, the contribution of the first source is represented by its subdomain (red cube) and the top separators (colored planes), while all other parts of the domain will remain out of its area of influence. Section 3.1 is dedicated to the exploitation of this feature.

Furthermore, we will show that the initial ordering of the sources is not optimal with respect to the operation counts, especially if one aims at processing many sources simultaneously. A simplified example of initial ordering is depicted in Fig. 2a, showing the source locations (circles) and the transmitter trajectory (line connecting the sources). One important remark here is that the transmitter trajectory crosses the top separator many times. In Section 3.2, we will show that the optimal ordering will be such that the transmitter trajectory has the smallest possible number of crossings of the top separators. The transmitter trajectory of Fig. 2b indeed possesses most of the properties of the theoretically optimal solution.

2.3 Characteristics of the models and computing environment

Our study is based on realistic anisotropic earth resistivity models characteristic for marine CSEM applications. The models are discretized using the finite-difference Yee grid [30] that places all electric and magnetic field components at different positions within a grid cell to ensure central differences, and hence higher accuracy. The grid had a uniform core in the middle, and growing cell sizes at the model edges and an air layer on top. Properties of system matrices and right-hand sides resulting from these discretizations are summarized in Table 1.

The matrices H3, H17, and S21 are described in detail in [27]. All matrices have a regular global structure that depends on the grid shape. Their nonzero structure typically looks like diagonals at distances $3 \times N_x$ and $3N_x \times N_y$ from the main diagonal, for a grid of dimension (N_x, N_y, N_z). The two H-matrices are based on a half-space 1 Ωm model with a 100-m water layer and a pizza-box resistor of 100 Ωm. The H3 matrix is based on a coarser grid, with cell sizes (in the central part) double of those used for the H17 matrix. The S21 matrix is obtained from the SEAM (SEG Advanced Modeling Corporation) Phase 1 resistivity model representative of the Gulf of Mexico: it has a rough bathymetry, hydrocarbon reservoirs, and salt bodies. The DB30 matrix is built from a resistivity model corresponding to a CSEM survey "Daybreak" acquired in Alaminos Canyon, Gulf of Mexico [20]. As an illustration of the structure of the matrix that will be processed by the direct solver, we show in Fig. 3 the nonzero structure of matrix H3 permuted with a nested dissection ordering.

The RHSs are generated by listing all transmitter positions using the ordering indicated in Fig. 2a. For example, for the SEAM S21 matrix, the survey layout suggested 36 towlines, 40 km long, in one direction, and 29 towlines, 35 km long in the orthogonal direction. The distance between towlines was 1 km. We downsampled the transmitter positions to 200 m of spacing, which resulted in $36 \times 201 + 29 \times 176 = 12340$ RHSs. RHSs for the daybreak matrix were given by the real survey that included 12 towlines of 60 km length and 2 km apart, and 2 orthogonal towlines of 30 km length, 4 km apart. For each system, the number of right-hand sides m reaches several thousands and their density $D(\mathbf{S})$ is below 10 nonzeros per column.

We also report in Table 1 the analysis, factorization, and solve times of the sparse direct solver MUMPS using BLR compression [2] at precision $\epsilon_{\text{BLR}} = 10^{-7}$ on the CALMIP supercomputer EOS (www.calmip.univ-toulouse.fr), which is a BULLx DLC system composed of 612 computing nodes, each composed of two Intel Ivybridge processors with 10 cores (total 12 240 cores) running at 2.8 GHz, with 64 GB of memory per node. As mentioned earlier, the

Table 1 Characteristics of the systems of equations $\boldsymbol{MX} = \boldsymbol{S}$

Model	System	Grid shape	Matrix $\boldsymbol{M}(n \times n)$		RHS $\boldsymbol{S}(m \times n)$		Timings in seconds (percentage of total time)			
		$N_x \times N_y \times N_z$	n	$D(\boldsymbol{M})$	m	$D(\boldsymbol{S})$	T_a	T_f	T_s	T_{total}
Shallow water	H3	$114 \times 114 \times 74$	2,885,112	12.9	8000	7.5	10 (1 %)	34 (4 %)	806 (*95 %*)	850
SEAM	H17	$214 \times 214 \times 127$	17,448,276	12.9	8000	6	56 (1 %)	378 (8 %)	4133 (*91 %*)	4567
DayBreak	S21	$181 \times 160 \times 237$	20,590,560	12.9	12340	9.5	68 (1 %)	476 (6 %)	7819 (*93 %*)	8363
	DB30	$230 \times 422 \times 102$	29,700,360	12.9	3914	7.6	106 (2 %)	765 (15 %)	4246 (*83 %*)	5117

Here $n = 3N_x \times N_y \times N_z$ is the order of $\boldsymbol{M}$, m is the number of columns of the right-hand side matrix $\boldsymbol{S}$, $D(\boldsymbol{M})$ and $D(\boldsymbol{S})$ are the average numbers of nonzeros per column for $\boldsymbol{M}$ and $\boldsymbol{S}$, respectively. The resolution times for the different phases of a sparse direct solver using 90 MPI processes and 10 threads per MPI process are also reported: T_a for analysis, T_f for factorization, T_s for solve, and $T_{\text{total}} = T_a + T_f + T_s$ for the entire resolution

Italicized entries emphasize the high relative cost of the solve phase

introduction of low-rank approximations has significantly reduced the factorization time [27], and the initial solve time T_s (not using the work presented in this paper) has become predominant. Note that the solve phase was performed by blocks of size BLK = 1024 for H3 and BLK = 512 for H17, S21, and DB30. Using larger blocks was not possible as the memory required to process all right-hand sides in one shot would have exceeded the available memory.

In the following section, we give some background on the solve phase of sparse direct solvers, before explaining in Section 3 how to take advantage of the right-hand side sparsity resulting from the geometrical structure of CSEM applications.

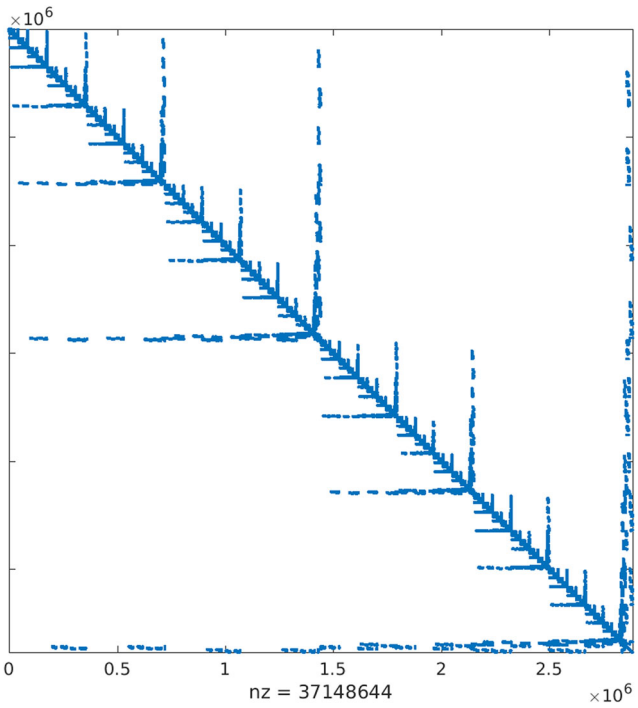


Fig. 3 Nonzero structure of matrix H3, permuted with a nested dissection ordering

2.4 Solve phase algorithms

We first describe the algorithms used to solve the linear systems $\mathbf{MX} = \mathbf{S}$, where $\mathbf{M} = \mathbf{LDL}^T$, from an algebraic point of view. We also explain how they can be interpreted and correlated to the structural and geometrical properties of CSEM applications.

$\mathbf{L}$ is a unit lower triangular sparse matrix of order n, whereas $\mathbf{S}$ is an $n \times m$ matrix of right-hand sides. As mentioned earlier, the first part of the solve algorithm consists in performing the forward substitution, which consists in solving the system $\mathbf{LY} = \mathbf{S}$.

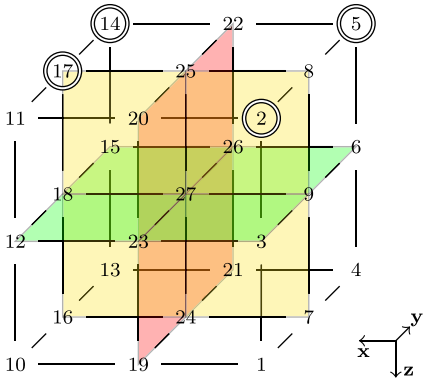
For each column k, the first version of our forward substitution algorithm is a scalar two-loop algorithm limited to nonzero entries in $\mathbf{L}$.

Algorithm 1 Scalar two-loop algorithm.

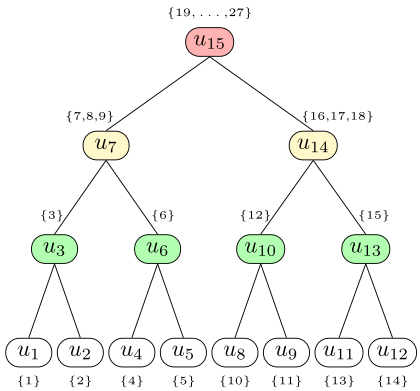
$Y_{*k} \leftarrow \mathbf{L}^{-1} Y_{*k}$
for $j = 1, \ldots, n-1$
  for $i \geq j+1$ such that $l_{ij} \neq 0$
    $y_{ik} \leftarrow y_{ik} - l_{ij} y_{jk}$

Algorithm 1 assumes that Y_{*k} has initially been set to S_{*k}. This way, it is expressed only in terms of modifications of Y_{*k}. The algorithm exploits the fact that the diagonal of $\mathbf{L}$ is the identity, and that $\mathbf{L}$ is sparse, i.e., many of the l_{ij} entries are zero. Based on the example of Fig. 4, we explain in the following how sparsity in $\mathbf{L}$ can be exploited in a more efficient way by limiting *a priori* the iterations on the i loop and will reformulate the algorithm to illustrate it.

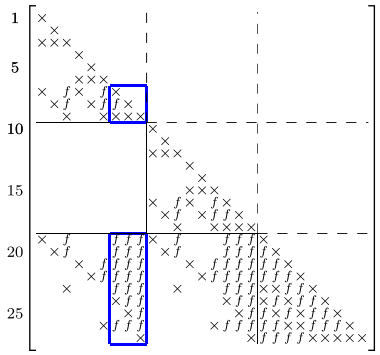
Figure 4a gives a simplified version of the CSEM application, where we consider a tiny $3 \times 3 \times 3$ grid, with one degree of freedom for each nodal point and used a 7-point stencil to represent the mesh. The corresponding matrix is represented in Fig. 4c. For the right-hand side matrix **S**, we consider only eight sources with a single nonzero per source. The sources are placed at nodal points 2, 5, 14, and 17, which all belong to the same **z**-plane. This is to illustrate the fact that the transmitter trajectory along the seafloor is usually quite horizontal. The initial ordering of the sources is assumed to be 2-17-5-14, followed by 2-5-17-14. This ordering matches the one shown in Fig. 1a and is convenient for illustrative purposes. The matrix **S** has the structure depicted in Fig. 6a. We will use this matrix of sources again in Section 3.



(a) $3 \times 3 \times 3$ regular mesh.



(b) Assembly/separator tree.



(c) Structure of $\boldsymbol{L}$.

Fig. 4 **a** A 3D regular mesh based on a 7-point stencil; each node is numbered according to the nested dissection algorithm following a postordering. Each node marked by double circles is an elementary source modeling two stacked sources. **b** Resulting separator tree; numbers in brackets show the sets of variables to be eliminated at each node. **c** Nonzero entries in **L**: $(\times)$ corresponds to entries already nonzero in **M** and (f) to new entries (fill-in) resulting from the factorization of **M**. Surrounded by two blue boxes are the entries concerned when processing node u_7 of the separator tree

In Section 2.1, we described the nested dissection process as a hierarchical domain decomposition. Performed prior to the factorization, it can also be regarded as a special numbering of nodal points to reduce *fill-in* (an initial $m_{\mathrm{ij}} = 0$ turning into an entry $l_{ij} \neq 0$ in the factor matrix **L**). Initially, the process builds a separator that divides the domain into two disjoint and *independent* subdomains, see Fig. 4a. It numbers the variables of each subdomain consecutively and the variables of the separator last. This can also be expressed as a root node u_{15} (separator) and two subtrees (subdomains) in the separator tree of Fig. 4b. The two subtrees characterize the aforementioned independence between the variables of both subdomains which is reflected by the empty square in the structure of **L** corresponding to rows $[10, 18]$ and columns $[1, 9]$ in Fig. 4b. From Algorithm 1, the computation of the components of **Y** inside each subdomain will then be independent from each other. For $i \in [10, 18]$ and $j \in [1, 9]$ we have $l_{\mathrm{ij}} = 0$ so that, for any k, and for any $i_1 \in [1, 9]$ and $i_2 \in [10, 18]$ component $y_{i_2 k}$ does not depend on component $y_{i_1 k}$. The separator tree thus also characterizes the parallelism of the solve phase. The nested dissection process is reproduced recursively on both subdomains, preserving the mentioned properties.

In the context of the multifrontal method, each node of the separator tree may be represented by a dense matrix called front which is used to compute a part of the **L** factor, as illustrated in Fig. 4c for node u_7 and in Fig. 5. Each front

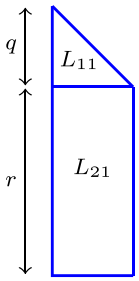


Fig. 5 Structure of the factor associated with a node from the separator tree

is associated with two sets of variables: the q variables of the separator (also called fully summed variables), which are used to compute entries of the $\mathbf{Y}$ or $\mathbf{X}$ solutions; and the r off-diagonal variables (or non fully summed variables), which are used to compute contributions. Data computed at each node will be used by the parent (resp. children) fronts in case of forward (resp. backward) substitution. More precisely, the forward substitution is a bottom-up process which performs, for each front, the two block operations $\mathbf{Y}_1 \leftarrow \mathbf{L}_{11}^{-1}\mathbf{Y}_1$ and $\mathbf{Y}_2 \leftarrow \mathbf{Y}_2 - \mathbf{L}_{21}\mathbf{Y}_1$, whereas the backward substitution is a top-down process which performs, for each front, the block operations $\mathbf{X}_1 \leftarrow \mathbf{X}_1 - \mathbf{L}_{21}^{T}\mathbf{X}_2$ and $\mathbf{X}_1 \leftarrow \mathbf{L}_{11}^{-T}\mathbf{X}_1$. Here, $\mathbf{Y}_1, \mathbf{Y}_2, \mathbf{X}_1$, and $\mathbf{X}_2$ are partial matrices of $\mathbf{Y}$ and $\mathbf{X}$. They only contain the rows of $\mathbf{Y}$ and $\mathbf{X}$ corresponding to the $q + r$ rows of the front, and subscript 1 (resp. 2) is associated with the q (resp. r) variables in the diagonal (resp. off-diagonal) part of the front. As follows from the properties of the separator tree, if two fronts belong to different subtrees, the computations at those fronts can be done in parallel.

We will use the notation $u(j)$ to denote the node of the separator tree containing variable j. We have, for example, $u(14) = u_{12}$, or $u(25) = u_{15}$. Thanks to the compact representation of the structure of the factors at each node (see Fig. 5), operations reported in Algorithm 1 can be performed on dense matrices. The condition "$l_{ij} \neq 0$" can then be replaced by "i in the structure of the factors at node $u(j)$," as will be indicated in Algorithm 2. Furthermore, in the context of sparse RHSs, some entries of Y_{*k} might remain equal to zero. Therefore, Algorithm 1 should only perform the update of y_{ik} for nonzero entries y_{jk}, leading to Algorithm 2.

Algorithm 2 Nodal algorithm.

$Y_{*k} \leftarrow \mathbf{L}^{-1}Y_{*k}$
for $j = 1, \ldots, n-1$
  for i in the structure of the factors at node $u(j)$, $i > j$
    if$(y_{jk} \neq 0)$ $y_{ik} \leftarrow y_{ik} - l_{ij}y_{jk}$

Similar to Algorithm 1, we assumed in Algorithm 2 that Y_{*k} has been set to S_{*k} for each column k prior to executing the algorithm. Note that in our example, the numbering of the node identifiers $u_1, u_2, \ldots, u_{15}$ obeys the following *postordering* rule: all nodes in any subtree are numbered consecutively and precede the number for the root of the subtree. Moreover, any subtree of T rooted at node u (which we denote as $T[u]$), corresponds to a subdomain created through the nested dissection. For example, $T[u_7]$ corresponds to the subdomain on the right of the first separator (u_{15}) and is composed of the variables {1,2,3,4,5,6,7,8,9}. Note also that the resolution of the diagonal system $\mathbf{DZ} = \mathbf{Y}$ can be performed in-between the forward and the backward substitutions or can be combined with one of these phases by computing each component as $z_{ik} = y_{ik}/d_{ii}$.

3 Exploiting RHS sparsity during the forward substitution

In the previous section, we have shown that thanks to the knowledge of the frontal matrix structure at each node of the separator tree, testing nonzero entries in the rows i of column $\mathbf{L}_{*j}$ was not needed and Algorithm 1 could be simplified. Furthermore, since $\mathbf{S}$ is sparse, some elements y_{jk} in Algorithm 2 remain equal to zero. Similarly, one would like to avoid such systematic testing for $y_{jk} \neq 0$ at each update of the nodal algorithm (Algorithm 2). For efficiency, we also want to perform operations on a block of columns and thus to *a priori* identify blocks of columns sharing the same structure and allowing simultaneous operations.

We describe the graph structure that needs to be introduced and exploited to avoid systematic testing and relate this structure to the geometric properties of the CSEM application. We focus in this section on the forward substitution ($\mathbf{LY} = \mathbf{S}$), but the same ideas can be applied to the backward substitution ($\mathbf{L}^T\mathbf{X} = \mathbf{Z}$) when a partial solution is needed, as will be discussed in Section 4.

Making efficient use of the sparsity in the RHS matrix is a three-step process described in the next subsections:

- Firstly, exploit sparsity within the columns of the sources (i.e., detecting empty rows), referred to as *vertical* sparsity
- Secondly, exploit sparsity within the rows (i.e., detecting nonzero blocks within non-empty rows), referred to as *horizontal* sparsity
- Finally, find a suitable column *ordering* to improve the performance of horizontal sparsity

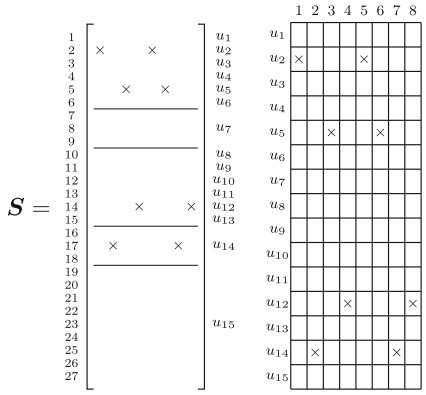
3.1 Vertical sparsity

Exploiting vertical sparsity is related to the sparsity of the source vectors in the CSEM application, which will leave many rows in $\mathbf{Y}$ empty. It makes use of the properties proved in [17] and was also formulated in terms of paths using the tree structure in [18].

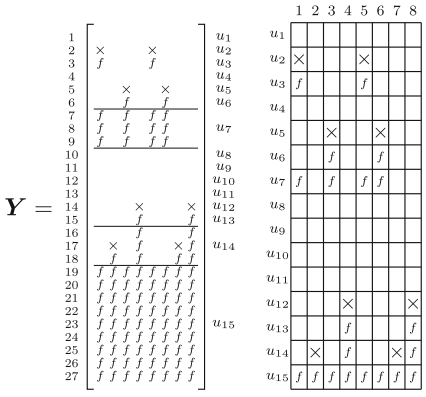
In Fig. 2a of Section 2.1, we illustrated the fact that the contribution of each source is limited to its local subdomain (the red subcube) and to the top separators. For a given source, or equivalently a column S_{*k} of the RHS matrix $\mathbf{S}$, the aforementioned contribution corresponds to nonzero

components of Y_{*k}. This observation provides guidance on how to sweep the separator tree. In the following, we explain how the separator tree can be used to efficiently characterize nonzero entries in $\mathbf{Y}$ so that the loop on index j can be set up *a priori* without need for any checks to restrict the subset of indices.

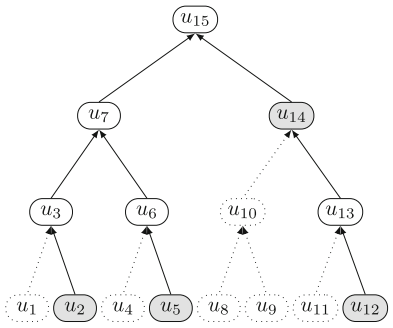
Figure 6a represents a matrix $\mathbf{S}$ composed of 8 right-hand sides associated with 8 sources (Fig. 4) placed at nodes



(a) Structure of $\boldsymbol{S}$



(b) Structure of $\boldsymbol{Y} = \boldsymbol{L}^{-1}\boldsymbol{S}$



(c) Pruned tree $T_p(\boldsymbol{S})$

Fig. 6 **a** Structure of the RHS matrix $\boldsymbol{S}$ (associated with the sources shown in Figure 4) and its node representation (right); nonzeros are represented with $\times$. **b** Structure of $\boldsymbol{Y}$ after the forward elimination; fill-in is represented with f (left and right). **c** Corresponding pruned tree; pruned nodes are dotted, and active nodes, i.e., nodes with sources, are shadowed

2, 17, 5, 14, 2, 5, 17, and14, in this precise order. In our simplified model and for the sake of clarity, we considered a single nonzero element per source. To simplify the figure, we also provide a compact representation of matrix $\mathbf{S}$ where each row corresponds to the set of variables from a node of the tree. Finally, each node whose set of variables includes at least one nonzero from matrix $\mathbf{S}$, i.e., each node $u(i)$ for which there exists a column index k such that $s_{\mathrm{ik}} \neq 0$, will be called an *active node*. Active nodes have been filled in the separator tree represented in Fig. 6c corresponding to our simplified model.

As the solve algorithm proceeds, new nonzero entries (so-called fill-in) with respect to the original entries of $\mathbf{S}$ appear in $\mathbf{Y}$. Given the initial nonzero structure of $\mathbf{S}$, [17] and [18] showed that it is possible to predict the nonzero structure of $\mathbf{Y}$. In our context, [18, Theorem 2.1] can be translated into the following:

Theorem 1 *When solving $\boldsymbol{L}Y_{*k} = S_{*k}$, the structure of the vector Y_{*k} is given by the union of the variables in nodes on paths in the tree T from the set of active nodes of S_{*k} up to the root.*

As a consequence, a component y_{ik} will be different from zero if and only if $s_{\mathrm{ik}} \neq 0$ or there exists an $s_{\mathrm{jk}} \neq 0$ such that either $u(j) = u(i)$ or $u(j)$ is a descendant of $u(i)$ in T. The update in Algorithm 2 is only applied for variables j belonging to such nodes. This a priori knowledge gives the possibility to *prune* nodes from the separator tree. This process is referred to as *tree pruning* in [28]. As an example, take S_{*1} from Fig. 6a with $s_{2,1} \neq 0$ and $u(2) = u_2$. Then every nonzero component of Y_{*1} belongs to nodes that are on the path from u_2 to u_{15}. This algebraic perspective translates into the geometrical interpretation illustrated in Fig. 2a.

Furthermore, to enhance the performance, computation should be done on multiple columns at the same time. In doing so, one can benefit from the use of BLAS 3 operations [12] that can almost reach the peak performance of a processor. Theorem 1 is then applied for the union of the set of *active nodes* of each column (see Section 2.3). The tree resulting from the pruning process is called the pruned tree and, if we consider the whole matrix $\mathbf{S}$ as one block, it is noted $T_p(\mathbf{S})$ and shown in Fig. 6c. Therefore, Algorithm 2 can be replaced by Algorithm 3.

Algorithm 3 Pruned tree nodal algorithm.

$\mathbf{Y} \leftarrow \mathbf{L}^{-1}\mathbf{Y}$
for $Y_{j*} \neq 0, 1 \leq j \leq n-1$ (i.e., $u(j) \in T_p(\mathbf{S})$)
  for i in the structure of the factors at node $u(j), i > j$
    $Y_{i*} \leftarrow Y_{i*} - l_{ij} Y_{j*}$

In this algorithm, Y_{j*} is the jth row of $\mathbf{Y}$ and $Y_{j*} \neq 0$ means that *at least one* of its component is different from 0. In practice, Algorithm 3 is implemented by processing $T_p(\mathbf{S})$ from bottom to top and performing the BLAS 3 operations $\mathbf{Y}_1 \leftarrow \mathbf{L}_{11}^{-1}\mathbf{Y}_1$ and $\mathbf{Y}_2 \leftarrow \mathbf{Y}_2 - \mathbf{L}_{21}\mathbf{Y}_1$ at each node, using the notation of Section 2.4. In Fig. 6c, each pruned node corresponds to an empty row in $\mathbf{Y}$. This is why sparsity is said to be exploited *vertically*.

3.2 Horizontal sparsity and column ordering

Algorithm 3 assumes that all columns are processed at each node of the pruned tree. However, sources do not share the same structure and thus it is possible to further exploit sparsity by reducing the number of columns on which Algorithm 3 is applied. This will be referred to as *horizontal sparsity*. To do so, we introduce the notion of active columns and node intervals. We then explain why column orderings introduced for computing selective entries of the inverse of a matrix [4] can be effective in our context to reduce the number of operations. We illustrate these aspects in Fig. 7 on the same simplified example with 8 sources.

The subset of columns of $\mathbf{S}$ that possesses at least one nonzero element at a given node u is called the set of *active columns* at node u. For example for node u_5, corresponding to row u_5 in Fig. 7b (left), there are only two active columns: 3 and 6. Ideally, one would like to operate only on these two columns, but this would either require a complex data reorganization, or the computation of the columns one after the other. The latter would not be efficient since processing a block of columns simultaneously is much faster that processing them one by one. What can be done at no extra reorganization cost is to consider a *subinterval of columns* including the first and the last indices of the active columns at that node. The intervals are thus defined for each node of the separator tree. In Algorithm 3 and for the computation of component Y_{i*}, $*$ is replaced by the interval defined for node $u(i)$. For example, the active columns for node u_5 are 3 and 6; thus, the interval at node u_5 only includes four columns: 3, 4, 5, and 6, rather than all 8 columns. With intervals, we reduce computation on columns and thus exploit *horizontal* sparsity.

Clearly, the size of the intervals is influenced by the ordering of the columns. The idea is to order successively

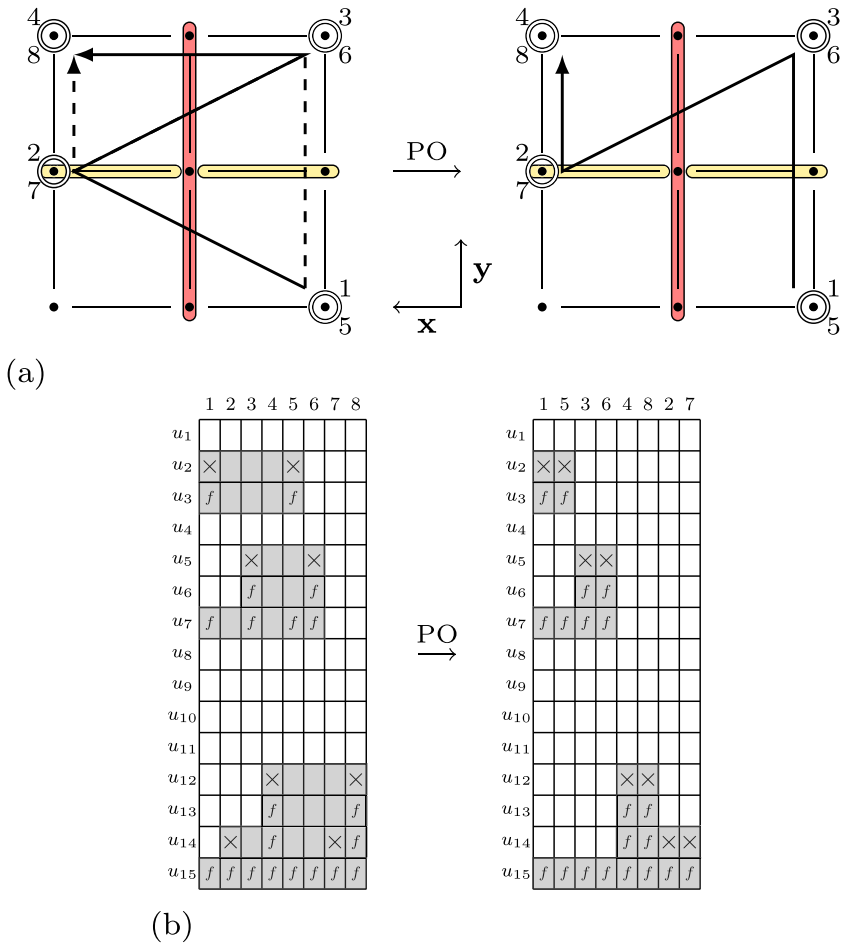


Fig. 7 **a** A 2D horizontal top-layer section from Fig. 4a, located on the plane containing the source positions. Colored lines are the traces of the separators of Fig. 4a. The numbering of nodal points has been omitted for clarity. On the left: the initial column ordering (1-2-3-4-5-6-7-8) corresponding to nodal points 2-17-5-14-2-5-17-14 in Fig. 4a. The plain and dashed arrows refer to the transmitter trajectories, firstly in the x direction and secondly in the y direction. On the right: the postorder (PO) column ordering (1-5-3-6-4-8-2-7). The plain arrow indicates the modified transmitter trajectory. **b** Illustration of horizontal sparsity with node intervals and influence of the column ordering on sparsity. The structure of $\mathbf{Y}$ with initial (left) and postorder (right) column orderings is shown

columns with close initial nonzero structure or, equivalently, to limit the crossing of top separators as was mentioned in relation to Fig. 2b.

The postordering rule introduced in Section 2.4 was used to order the nodes of the elimination tree. It also leads to a postordering of all the variables. The permutation used in this study will exploit this postordering of the variables and is built as follows. For a column k of $\mathbf{S}$, we define $u_{\text{rep}}(k)$ as the node among the active nodes for column S_{*k} ($\{u(i), s_{\text{ik}} \neq 0\}$) that appears first in the postordering of the tree. We have for example $u_{\text{rep}}(1) = u_2$ and $u_{\text{rep}}(2) = u_{14}$ in Figs. 6a (and 7b, left), and call $u_{\text{rep}}(k)$ the *representative* node of column k. Now $\mathbf{S}$ is said to be *postordered* if and only if: $\forall k_1, k_2, 1 \leq k_1 < k_2 \leq m$, $u_{\text{rep}}(k_1)$ appears before (or is identical to) $u_{\text{rep}}(k_2)$ in the postordering. In other words, the order of the columns S_{*k} and the postordering of their representative nodes $u_{\text{rep}}(k)$ are compatible.

In Section 2.1, we mentioned that the transmitter trajectory should minimize the number of crossings of top separators. In Fig. 7b, the postorder trajectory is also interpreted in terms of nonzero structure of $\mathbf{S}$ and $\mathbf{Y}$. Namely, it corresponds to ordering the columns of $\mathbf{S}$ in such a way that two successive columns have similar nonzero structure (in $\mathbf{Y}$). This was not the case with the initial transmitter trajectory of Fig. 7a, which resulted in large interval sizes for rows u_7 and u_3 (for example) (see Fig. 7b (left)). The postorder heuristic addresses this problem (as shown in Fig. 7b (right)). It is here optimal since the gray areas representing the intervals no longer include zero entries. Note that, for the purpose of our illustration, we have considered sources with only one nonzero entry and that in this case the postorder heuristic has been shown to be optimal [4]. In our CSEM application, each source has more than one entry per column, and thus possibly more than one active node, hence the definition of u_{rep}.

Tree pruning, node intervals, and a suitable column ordering exploit the sparsity of the application to reduce the amount of computations in the solve phase. However, this is done at the expense of reduced parallelism. Section 5 shows how to still efficiently exploit parallelism in the solve phase, even when dealing with sparse right-hand sides.

4 Exploiting sparsity during the backward substitution

During the backward phase ($\mathbf{L}^T\mathbf{X} = \mathbf{Z}$), the nonzero structure of $\mathbf{Z}$ results from the operations performed during the forward substitution. It is preserved for $\mathbf{Y}$ since $\mathbf{DZ} = \mathbf{Y}$ and $\mathbf{D}$ is diagonal. When the matrix $\mathbf{M}$ is irreducible, which is the case in the CSEM application, the elimination tree is a real tree and not a forest. Theorem 1 then states that the variables of $\mathbf{Y}$ belonging to the root node of the separator tree will be nonzero, independently of the position of the sources. The backward substitution processes the $\mathbf{L}$ matrix in a backward way or, equivalently, into a top-down traversal of the separator tree. As a result, all the nodes in the tree are reached and need to be processed during the backward phase. Since the tree is unique, this translates back into the fact that $\mathbf{Z}$ is dense and that sparsity in the sources $\mathbf{S}$ does not result in any reduction of the number of operations in the backward phase.

However, the sparsity of the solution can result from the properties of the physical problem, typically when only part of the solution is valuable and needs to be computed. As explained in Section 2.1 and illustrated in Fig. 1, boxing and/or regular sampling can be used to select a subset of valuable entries. How sparsity can be exploited during the backward substitution is explained below.

Given a valuable entry x_{ik} in the solution, the computations that contribute to updating x_{ik} can be characterized, similarly to the forward phase, by Theorem 1. Only nodes in the path from the root node to node $u(i)$ need be considered to compute x_{ik}. This property was proved in a more general context in [26, Lemma 2.2]. In other words and from a geometric perspective, if one assumes that i belongs to the filled subdomain of Fig. 2a then the variables involved in the computation of x_{ik} will correspond to the colored separators and part of the filled subdomain. As a consequence, the process of tree pruning introduced in Section 3.1 can be applied to the backward substitution. The exploitation of horizontal sparsity also remains unchanged and the computation of a suitable column ordering inside each block follows the same rule, namely "columns with similar structure of valuable entries should be kept close in the column ordering."

First, sources close to each other have highly overlapping boxes of valuable entries. Their representative nodes in the separator tree are thus close to each other. Second, in the case of regular sampling of the entries in the solution, we have no locality property to preserve since all the space is regularly covered by the solution. Thus, the representative nodes of the sources can also be used for the boxes; therefore, the column ordering chosen during the forward phase can be used during the backward phase.

The valuable entries in each column of $\mathbf{X}$ are thus defined as a sampled set of variables in a box around the corresponding source location. It should be noted that this numerical sparsification of $\mathbf{X}$ is quite moderate compared to the extreme sparsity of the sources $\mathbf{S}$. Thus, $\mathbf{X}$ is a much denser matrix with less geometrically localized nonzero variables than $\mathbf{S}$. As illustrated in Section 6, the impact of exploiting sparsity will thus be smaller for the backward step than for the forward step.

5 Improving the parallel aspects of the solve algorithms

In this section, we first explain the differences between the factorization and the solve phases in terms of parallel algorithms. We then show how the blocks of sparse RHS can be defined and how the solve phase can be adapted to improve the available parallelism.

5.1 Differences between the factorization and the solve algorithms

The factorization and the solve algorithms have different properties in terms of parallelism and load balancing. Although in practice we apply a BLR factorization, we consider in this section full-rank metrics because they are the basis for the mapping and scheduling algorithms we use [5]. We recall that, on the one hand, *tree parallelism* is represented by the separator tree (two nodes from different subtrees can be processed independently, as explained in Section 2.4). On the other hand, large nodes of the separator tree offer an additional potential for parallelism. This will be referred to as *node parallelism*. Moreover, on a dense matrix of order n, the complexity in terms of number of operations of the factorization and solve phases, respectively $O(n^3)$ and $O(n^2)$, is quite different. With nested dissection, the size of the separators and thus the size of the frontal matrices increases as we get closer to the top of the tree. Most of the computation thus occurs near the top of the tree and this is emphasized for the factorization (with respect to the solve). This effect is illustrated in Fig. 8, which compares the distribution of computations in the separator tree for both phases. At a level where 50 % of the computation is completed for the solve phase, only 20 % is completed for the factorization. This indicates that the solve phase has more potential to exploit tree parallelism than the factorization phase.

Tree pruning limits the number of branches of the separator tree that can be computed independently. Thus, tree parallelism and tree pruning introduced to exploit RHS sparsity are two conflicting objectives. A classical approach to balance the workload between the processors during the factorization is to use a *proportional mapping* [25]. The algorithm starts from the root node and proceeds recursively down the tree. At each node of the tree the current list of processors is partitioned among its children according to the load of each child subtree. This is referred to as strict proportional mapping and illustrated in Fig. 9a. It can be adapted or relaxed in order to allow for dynamic mapping and scheduling decisions, or to reduce memory usage [5]. If the whole set $\mathbf{S}$ is considered, and if the set of sources is separated by the top level separators, then the width of the pruned tree $T_p(\mathbf{S})$ may be large enough to cover most of the tree and almost fully benefit from tree parallelism.

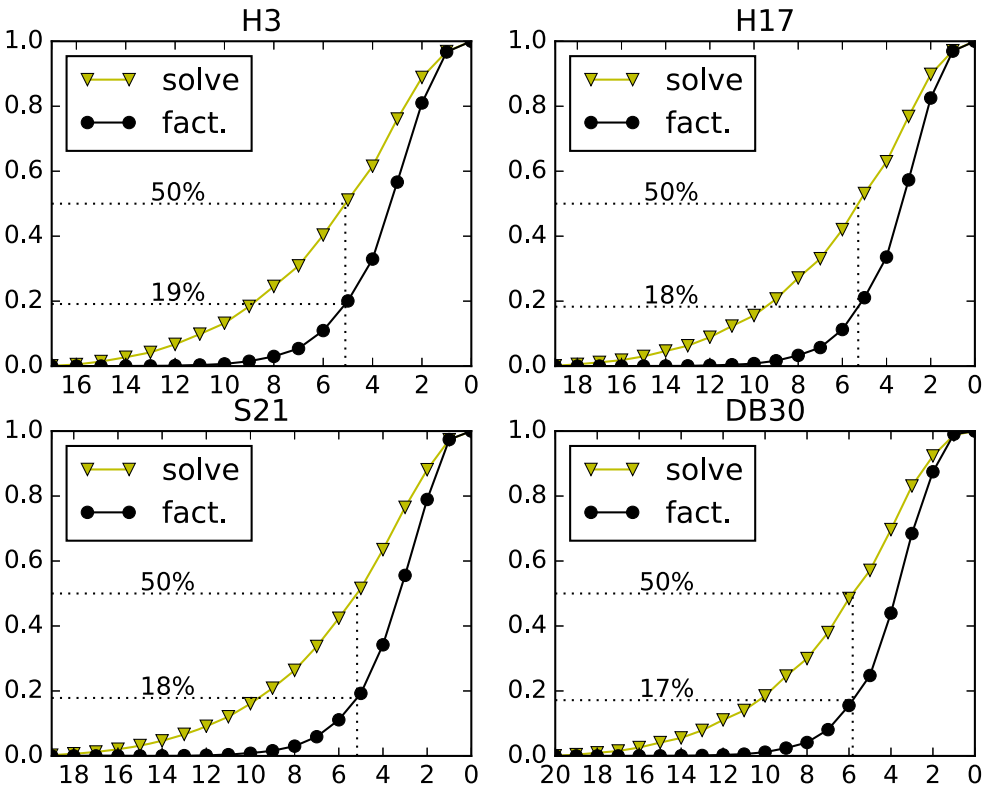
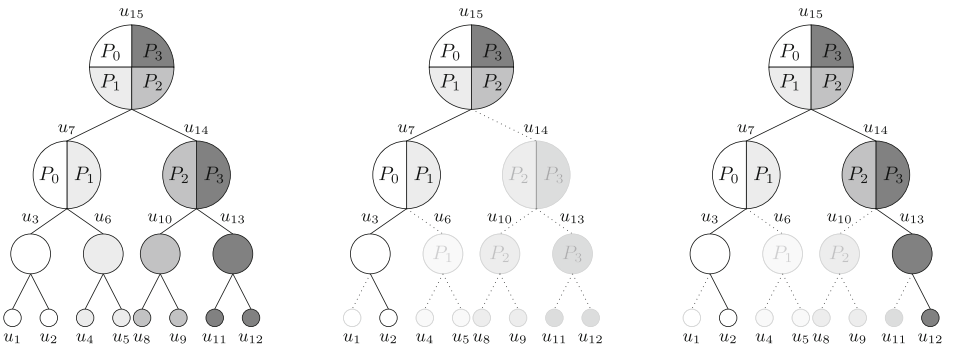


Fig. 8 Normalized operation count of the solve and factorization phases as a function of the level in the separator tree, with level (root) = 0. The nested dissection ordering has been used



(a) T with 4 processors. (b) $T_p(\boldsymbol{S}_1)$ with $\boldsymbol{S}_1 = (S_{*1}, S_{*5})$. (c) $T_p(\boldsymbol{S}_2)$ with $\boldsymbol{S}_2 = (S_{*1}, S_{*4})$.

Fig. 9 Proportional mapping and comparison of tree coverage between three blocks of right-hand sides based on the example from Figs. 4 and 6. **a** Dense RHS (all the tree is covered); **b** set of two close sources; **c** set of two distant sources

However, because of the memory constraints mentioned in Section 2.3, the RHSs are processed by blocks of limited size (BLK), potentially reducing the width and parallelism of the pruned tree. In this context, it is important to define these blocks to limit the loss of tree parallelism introduced by the exploitation of RHS sparsity.

Furthermore, at each node of the separator tree, a symmetric frontal matrix is partially factored. For frontal matrices associated with large separators near the top of the tree, the proportional mapping assigns several processors and the workload of the factorization is divided between a master and several workers. This is illustrated in Fig. 10 where q is the size of the separator and r is the number of rows to be updated. At each node, the first q variables are factorized. The number of operations at each front can be expressed as the sum of the operations performed on the master $W_m^f(q)$ and on the workers $W_w^f(q,r)$ as follows:

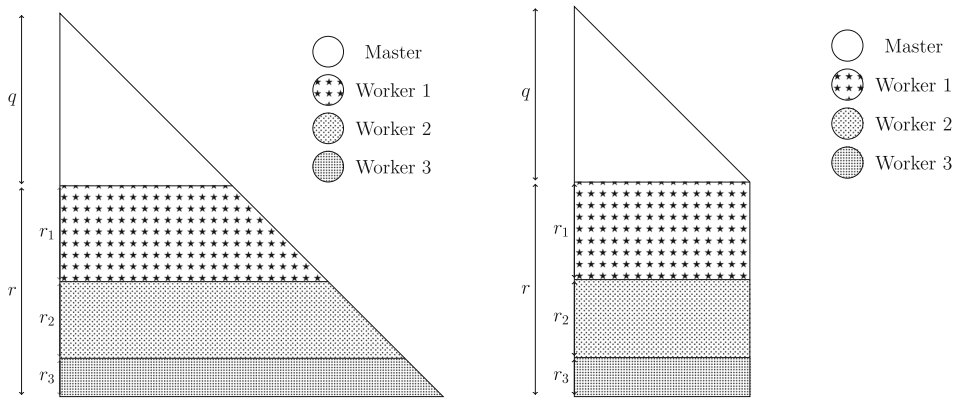
$$W^f(q,r) = W_m^f(q) + W_w^f(q,r), \tag{2}$$

where $W_m^f(q) \approx \frac{1}{3}q^3$ corresponds to the cost to factor a dense matrix of order q and $W_w^f(q,r) = \mathrm{qr}(q+r+1)$ to the cost to update the trailing r rows of the front. As shown in Fig. 10a, more rows must then be mapped onto the processors that appear first in the front. Note that we also want to adjust the relative sizes of q and r to balance the workload between the master and each worker by splitting nodes of the separator tree [3, 7].

In CSEM applications, where the solve phase becomes predominant, we need to drive our algorithms with metrics related to the solve phase, as described in the following subsection.

5.2 Improving algorithms for the solve phase

Because of memory constraints, the columns of $\mathbf{S}$ need be processed by blocks of limited size BLK. In the scheme presented in Section 2.3, the columns of $\mathbf{S}$ are processed using the initial ordering and then only a subpart of the



(a) Front during factorization (triangular frontal matrix). (b) Front during solve (factors).

Fig. 10 Mapping of the rows of a front to balance the workload of the factorization between processors

domain is covered by the partial transmitter trajectory within each block. Large subdomains, or subtrees, will be pruned from the separator tree, limiting the number of operations but also leading to a significant loss of tree parallelism. Figure 9 a and c illustrates this property with two sets $\mathbf{S}_1$ and $\mathbf{S}_2$, containing close and distant sources, respectively.

To improve the tree coverage, one can select non-contiguous columns from matrix $\mathbf{S}$. They will better cover the physical domain because the transmitter follows a regular trajectory. Furthermore, to increase the efficiency of BLAS kernels, each block of BLK columns consists of a set of sub-blocks of constant size equally distributed onto the transmitter trajectory. To do so, for a given sub-block size whose size is related to the BLAS-3 performance kernel, one can compute a constant gap to provide a good trajectory coverage and thus a good separator tree coverage. We then apply a postordering permutation within each block to maximize the effect of horizontal sparsity.

Moreover, node parallelism has an important role in the performance of the solve phase. As shown before, Fig. 10b illustrates the distribution of data among processors when the work is balanced for the factorization. At each step of the solve phase (forward or backward substitution), two operations are performed for each nonzero entry in the $\mathbf{L}$ factor. The number of operations performed at each node is expressed as the sum of the operations performed on the master $W_m^{\text{fwd}}(q)$ and on the workers $W_w^{\text{fwd}}(q, r$ as follows:

$$W^{\text{fwd}}(q, r) = W_m^{\text{fwd}}(q) + W_w^{\text{fwd}}(q, r), \tag{3}$$

where $W_m^{\text{fwd}}(q) = q(q-1)$ and $W_w^{\text{fwd}} = 2\text{rq}$. To balance W_w^{fwd} among workers, data need to be mapped so that all workers possess the same number of rows. For that, we also switched off the dynamic schedulers from the factorization that lead to irregular partitions with a dynamic choice of the workers at each node. Instead, we use a strict static proportional mapping of the processors in the tree. This strategy will be referred to as S-RowDistrib.

Furthermore, to balance the work between the master and each worker, we aim at splitting nodes in the separator tree so that $W_m^{\text{fwd}}(q) \approx W_w^{\text{fwd}}(q, r_i)$, where r_i, the number of rows of each worker is equal to r divided by the number of workers. This strategy is referred to as S-Split.

In summary, the optimizations above aim at favoring tree and node parallelism during the solve phase when dealing with either sparse or dense RHSs. Concerning the optimizations specific to sparse RHSs, we focused on the forward substitution but they also apply to the backward substitution, for the same geometrical reasons discussed in Section 4, and with the same choice of blocks. In Section 6, we also experiment with another optimization of the solve phase regarding locality of data access and multithreading. This was motivated by the need to process large blocks of columns to improve tree coverage and tree parallelism.

6 Performance analysis in a parallel context

We analyze the impact of exploiting RHS sparsity and using parallel solve-aware strategies on the performance of the solve phase in a parallel environment. We also present global resolution times showing that the relative weight of the solve phase has significantly decreased compared to the initial results from Table 1.

A perfect nested dissection ordering has been chosen for all the following results, which were obtained using the MUMPS solver [3, 5]. We note T_f, T_s, T_{fwd}, T_{root}, and T_{bwd} and the times to perform the factorization, solve, forward substitution, solve on the root node (through ScaLAPACK [9]), and backward substitution, respectively.

6.1 Exploiting sparsity

We first study vertical and horizontal sparsity, and show the impact of the choice of the columns and their order on parallelism. In particular, we will demonstrate the effect on tree parallelism of using non-contiguous columns within each block. We consider the forward substitution in Section 6.1.1 and the backward substitution in Section 6.1.2.

6.1.1 The forward substitution

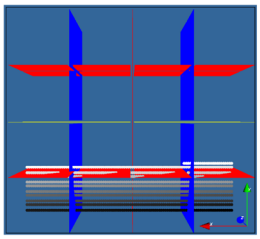
We first report in Table 2 the performance in terms of number of operations and time for solution of the proposed algorithms (dense, vertical, horizontal sparsity, and postordering of RHS columns) on the system H3 on 1024 contiguous columns of RHS. In the column "dense," the RHSs are still provided sparse, but sparsity is ignored during the computations. The resulting number of operations and runtime are thus similar to those one would obtain with dense RHSs. As expected from theory

Table 2 Number of operations (OPS$\times 10^{10}$) for the forward elimination for 1024 contiguous RHSs of system H3

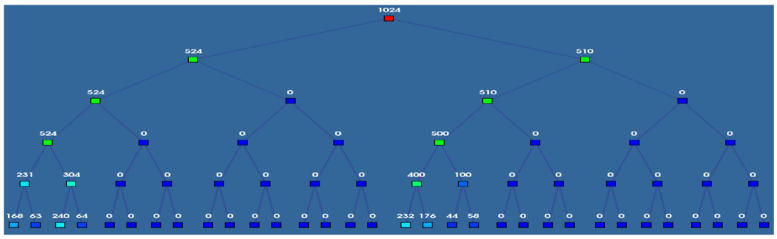
OPS ($\times 10^{10}$)		Dense	Vertical sparsity	Horiz. sparsity	Horiz. sparsity and postorder
Contiguous	First 1024 ($\boldsymbol{S}_1$)	951	270	225	149
	Last 1024 ($\boldsymbol{S'}_1$)	951	232	190	151

Table 3 Times (seconds) for the forward elimination for 1024 contiguous RHSs of system H3, with 32 MPI and 1 thread per MPI

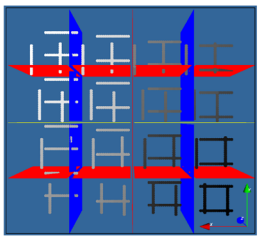
T_{fwd} (s)		Dense	Vertical sparsity	Horiz. sparsity	Horiz. sparsity and postorder
Contiguous	First 1024 ($\boldsymbol{S}_1$)	170	112	85	60
	Last 1024 ($\boldsymbol{S'}_1$)	170	114	87	69



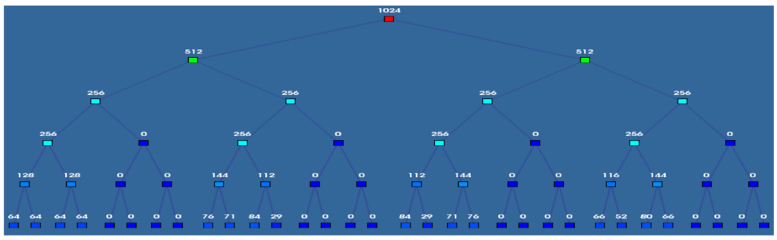
(a) Separators and set of 1024 contiguous sources ($\boldsymbol{S}_1$).



(b) $T_p(\boldsymbol{S}_1)$ with active columns.



(c) Separators and set of 1024 non contiguous sources ($\boldsymbol{S}_2$).



(d) $T_p(\boldsymbol{S}_2)$ with active columns.

Fig. 11 Geometrical and algebraic RHS distribution for two subsets of 1024 columns for system H3. **a** , **c** Represent top views of the geometrical domain for, respectively, 1024 contiguous RHS in natural order and 1024 non-contiguous RHS sets of 16 columns with a gap of 109 columns permuted using a postordering. The color gradient indicates the index of the column (source) in the (possibly reordered) set of RHS columns. **b**, **d** The respectively corresponding top 6 layers of the separator tree with, for each node, the number of active columns, as defined in Section 3.2

Table 4 The same as Table 2, but for non-contiguous RHSs

OPS ($\times 10^{10}$)		Dense	Vertical sparsity	Horiz. sparsity	Horiz. sparsity and postorder
Non-contiguous	16 cols, gap 109 ($\boldsymbol{S}_2$)	951	428	306	125
	32 cols, gap 218 ($\boldsymbol{S'}_2$)	951	427	302	132

Table 5 The same as Table 3, but for non-contiguous RHSs

T_{fwd} (s)		Dense	Vertical sparsity	Horiz. sparsity	Horiz. sparsity and postorder
Non-contiguous	16 cols, gap 109 ($\boldsymbol{S}_2$)	170	111	98	30
	32 cols, gap 218 ($\boldsymbol{S'}_2$)	169	107	96	31

Table 6 Number of operations (OPS $\times 10^{10}$) for the backward substitution for 1024 non-contiguous RHS of system H3

OPS ($\times 10^{10}$)	Samp.	Dense	Vertical sparsity	Horiz. sparsity	Horiz. sparsity and postorder
16 cols, gap 109	20	951	876	702	636
(S_2)	100	951	876	701	635
32 cols, gap 218	20	951	876	703	626
(S'_2)	100	951	876	702	625

Except for column "Dense," only a subset of the solution is computed with coarse solution vector sampling applied

(compare columns 3 and 4 of Table 2), using vertical sparsity significantly reduces the number of operations with respect to the dense case. Adding horizontal sparsity and postordering the columns further reduces the number of operations. However, as shown in Table 3, this operation reduction is not fully converted into time reduction.

We illustrate with Fig. 11 the conflicting objectives of vertical sparsity and performance and explain how to address this issue. With the initial order of the columns in $\mathbf{S}$, the 1024 first ones (set $\mathbf{S}_1$) are located at the low y part of the horizontal plane containing the sources (see Fig. 11a). They appear in the order described in Fig. 2a. From an algebraic point of view, the effect of tree pruning (see Fig. 11b) is that $T_p(\mathbf{S}_1)$ is quite narrow (many branches have no active columns). On the contrary, choosing 1024 non-contiguous columns spreads the RHSs in the domain. This is illustrated in Fig. 11c with the set $\mathbf{S}_2$, consisting of subsets of 16 columns in $\mathbf{S}$ separated by gaps of 109 columns. A first consequence of such a distribution is a wider pruned tree, with more nodes in $T_p(\mathbf{S}_2)$ than in $T_p(\mathbf{S}_1)$—compare Fig. 11b, d.

As a consequence, when only vertical sparsity is used, one can expect a larger number of operations with $\mathbf{S}_2$ than with $\mathbf{S}_1$ (compare columns "Vertical sparsity" of Tables 2 and 4). However, it is also interesting to observe that horizontal sparsity combined with a postordering of the RHS columns recovers this increase in the number of operations (compare last columns of Tables 2 and 4). We discuss/explain it in the following.

In Section 3.2, we explained that sources closely located in the geometrical domain needed to be close in the column ordering to reduce the operation count. The color gradient from Fig. 11c illustrates the effect of postordering the columns: sources that belong to the same subdomain become close with respect to the column ordering. This property explains why the efficiency of horizontal sparsity is increased even more when a postordering of the columns is applied. Indeed, for set $\mathbf{S}_1$, we have a 17 % reduction in the number of operations with horizontal sparsity, reaching 45 % when postordering is applied. With non-contiguous columns (set $\mathbf{S}_2$), the operation reduction due to horizontal sparsity and postordering reaches 71 % (see Table 4). Thus, even in case of non-contiguous columns, the number of operations is comparable (even slightly smaller) to the case of contiguous columns (compare the last columns of Tables 2 and 4). Non-contiguous columns also expose the forward step to more parallelism. Thus, the time for the forward step with contiguous columns (already divided by a factor of three with respect to dense RHS processing—compare the last and the third columns of Table 3) is further divided by a factor of two (last column in Table 5).

6.1.2 The backward substitution

We analyze in Tables 6 and 7 the impact of computing only a subset of the solution on the operation count and on the execution time, respectively. In these tables, the postorder used is identical to the one from the forward

Table 7 Estimated times (seconds) for the backward substitution for 1024 non-contiguous RHS of system H3, with 32 MPI and 1 thread per MPI

T_{bwd} (s)	Samp.	Dense	Vertical sparsity	Horiz. sparsity	Horiz. sparsity and postorder
16 cols, gap 109	20	169	160	141	127
(S_2)	100	170	160	140	131
32 cols, gap 218	20	169	160	137	127
(S'_2)	100	170	160	141	127

Except for column "Dense," only a subset of the solution is computed with coarse solution vector sampling applied

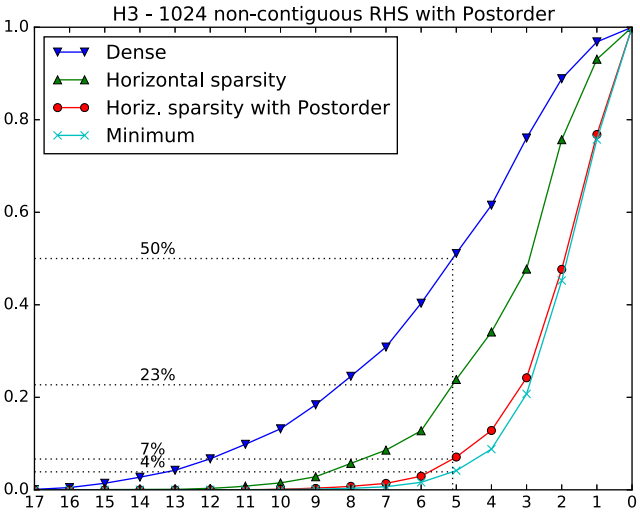


Fig. 12 Normalized accumulation of operations by levels in the separator tree for the forward elimination with 1024 non-contiguous RHS permuted in postorder (S_2) for H3. `Minimum` corresponds to the minimal number of operations, if all operations on zeros were avoided

substitution, avoiding any RHS permutation between the forward and the backward phases. We only show results with non-contiguous sets of columns since this enables, as in the forward step, a better exploitation of tree parallelism. To measure the time and the number of operations, we exploit the fact that performing the backward substitution ($\mathbf{L}^T\mathbf{X} = \mathbf{Z}$) while computing only a subset of the entries of the solution $\mathbf{X}$ is equivalent in terms of operations, computation kernels used, and parallelism, to performing the forward substitution $\mathbf{LY} = \mathbf{X}$ exploiting the sparsity of the right-hand side $\mathbf{X}$. All options to exploit sparsity developed for the forward phase could then be used to analyze the potential of exploiting sparsity during the backward step.

The density of the solution is such that one should expect much smaller gains due to sparsity for the backward substitution than for the forward substitution. Indeed, Fig. 1 b shows that the box around a source where the solution needs to be found is quite large. Furthermore, to make the forward phase most efficient, the RHSs have been combined in blocks, as illustrated in Fig. 11c. The union of all boxes associated with a given block of sources will cover an even bigger part of the physical domain. This means that the union of the nonzero structures of the corresponding solutions will cover most of the domain. Hence, as shown in Table 6, the ratio of operations between the dense and the vertical strategies is close to one. We also observe that the performance using coarse solution vector sampling is not affected when the number of degrees of freedom on which the solution is computed decreases from 1 over 20 to 1 over 100 (from sampling 20 to 100). Although the coarse solution vector sampling can be useful to reduce the volume of data corresponding to the solution, it indeed only affects vertical and horizontal sparsity in the lowest levels of the tree, which constitute only a minor part of the computation. Overall, the exploitation of sparsity in the computed entries of $\mathbf{X}$ brings an approximate 1.35× gain on the time for the backward substitution. This gain is significant but is for the moment based on running the forward substitution, so that we decided not to include it in the global results of Section 6.3.

6.2 Improvement through load balancing and multithreading

Load balancing is necessary at several levels. In this section, we first evaluate the impact of sparsity on tree parallelism, and then show the importance of pushing forward node parallelism through balanced workload between workers and between the master and the workers. We mention that sparsity is not exploited for the backward substitution in this section and we report the actual times obtained for the dense backward substitution.

In Fig. 12, we analyze the relation between tree parallelism and exploitation of right-hand side sparsity during the forward substitution. Note that for the dense RHS case we have shown in Fig. 8 that the solve phase offers a greater potential for exploiting tree parallelism than the factorization phase. We see in Fig. 12 that when reaching the fifth level of the tree, 50 % of the computation is performed with dense RHS. When exploiting horizontal sparsity, only 23 % of the computation is performed

Table 8 Effect of different mapping strategies on the time for the factorization and solve phases for H3, on EOS, with 1024 RHS (set S_2), using 32 MPI processes and 1 thread per MPI process

Mapping strategy	Fact. time	Forward sparsity	Solve time	
	T_f		T_s	$T_{\text{fwd}} + T_{\text{root}} + T_{\text{bwd}}$
Standard MUMPS solver	203	No	346	170 + 9 + 167
		Yes	206	30 + 9 + 167
With S-RowDistrib	203	No	306	153 + 9 + 144
		Yes	174	24 + 9 + 144
With S-RowDistrib and S-Split	197	No	295	147 + 9 + 139
		Yes	167	19 + 9 + 139

Table 9 Effect of locality and multithreading optimizations (THREADOPT) on T_s (seconds) to process the RHS columns S_2 of system H3

# Threads	THREADOPT	T_f	T_s	
			BLK =16	BLK =1024
1	Off	197	272	167
	On	197	271	136
10	Off	48	106	79
	On	48	87	28

T_s is reported as a function of the block size BLK (standard or large) and of the number of threads (1 or 10) per MPI process, for 32 MPI processes. Sparsity is exploited during the forward elimination phase only

at that level, decreasing to 7% when horizontal sparsity is combined with a postordering of the columns. This translates into an important loss of tree parallelism that confirms the increasing relative weight of node parallelism. Table 8 gathers results for the different strategies introduced in Section 5.2.

We first observe that, thanks to the efficient use of sparsity during the forward step, the solve phase is dominated by the backward step. However, balancing the workload between workers through equal distribution of rows (S-ROWDISTRIB strategy in Table 8), and balancing the workload between master and workers (S-SPLIT strategy) significantly improves the performance of the solve phase. The forward substitution time with sparse RHS decreases from 30 s down to 19 s, showing a significantly larger relative gain than the one obtained during the (dense) backward substitution, from 167 down to 139 s. This is coherent with the observation reported in Fig. 12 that node parallelism is more critical when sparsity is exploited.

We now consider the performance of the solve phase in an hybrid MPI-OpenMP environment, where multiple threads are used within each MPI process. In general, sparse direct solvers are used on a limited number of right-hand sides, and processing several of them together (e.g., 16 or 32) leads to better arithmetic intensity and performance thanks to the use of BLAS 3 operations at each node of the separator tree, for which one can rely on multithreaded BLAS libraries. However, CSEM applications have a much larger number of sparse right-hand sides and larger blocks of right-hand sides (ideally all of them if memory was not an issue) are needed to cover the tree and benefit from sufficient tree parallelism (see Section 6.1.1). In this context, and especially in a multithreaded environment, efficient data manipulations at each node of the tree and data locality become critical to efficiently exploit the caches and the memory bandwidth of the processors. We have thus worked on improving locality and on multithreading memory-bound operations in both the forward and backward solve phases: arrange nested loops to match the storage of right-hand sides and intermediate solutions, introduce new OpenMP directives, improve data locality, and suppress intermediate storage whenever possible.

Table 9 reports the impact of these improvements on locality and multithreading (noted THREADOPT) on the solve time for the system H3 with the 1024 non-contiguous RHSs corresponding to the set $\mathbf{S}_2$ used in Sections 6.1.1 and 6.1.2. The block size (BLK) defines the number of right-hand sides treated in one shot. We see that with a block size of 16, the improvement due to better data locality is nonexistent with one thread and relatively limited with 10 threads. However, with a block size of 1024 (i.e., when all RHSs of S_2 are processed in one shot), the impact of these optimizations motivated by the CSEM sparse RHS context become very large, as T_s decreases from 79 to 28 s.

We end the study with results on several test matrices that combine all techniques introduced previously.

6.3 Global resolution times

We now summarize the new results obtained on the set of systems presented in Table 1, and show that the work described in this article has a large impact on the global resolution times. We also relate the results to previous work [27] which compared the direct approach with an iterative one.

The experimental environment is the one described in Section 2.3. Runs are performed on the EOS machine on 90 MPI $\times$10 threads, hence a total of 900 cores. The solve phase is performed using a blocking parameter BLK equal to 1024 columns for system H3, and to 512 for the larger systems H17, S21, and DB30. Compared to Table 1, each block now consists of non-contiguous columns in order to encourage tree parallelism, and the columns within each block are postordered. The blocks are thus defined in a way similar to $\boldsymbol{S}_2$ or $\boldsymbol{S'}_2$ in Section 6.1.1, but they are shifted in order to cover the entire RHS set. The root node of the separator tree uses ScaLAPACK for both the factorization and the solve phases (as in Table 1).

We report in Table 10 the detailed times for the solve phase, as well as the time for the analysis and factorization

Table 10 Time (seconds) of the analysis, factorization, and solve phases on 90 MPI × 10 threads *with* and *without* the improvements described in the study

Statistics with improvements				
	H3	H17	S21	DB30
T_a	10 (3 %)	56 (3 %)	68 (2 %)	106 (7 %)
T_f	31 (10 %)	380 (21 %)	434 (14 %)	510 (33 %)
T_s	284 (87 %)	1402 (76 %)	2704 (84 %)	927 (60 %)
T_{fwd}	73 (22 %)	289 (16 %)	759 (24 %)	184 (12 %)
T_{root}	14 (4 %)	190 (10 %)	326 (10 %)	80 (5 %)
T_{bwd}	197 (61 %)	923 (50 %)	1619 (50 %)	663 (43 %)
T_{total}	325	1838	3206	1543
Statistics without improvements (see details in Table 1)				
T_{total}	850	4567	8363	5117

Timings of the backward substitution do not include potential gains reported in Section 6.1.2. The numbers in parenthesis indicate the percentage of T_{total}

phases when the improvements described in this article are applied. In Section 6.1.2, we have explained how sparsity of the solution can be efficiently exploited during the backward substitution. Based on an assumption of sparsity of the solution motivated by the application, estimated gains were computed. This on-going and promising work should be further studied and validated in the context of a real simulation and has thus not be included in the global gains reported in this section.

Whereas the solve time represented between 83 and 95 % of the complete resolution time in Table 1, its weight now only represents between 60 and 87 % of the resolution time. The solve time has indeed been divided by a factor between 2.8 (for H3) and 4.6 (for DB30). We note that the factorization times have slightly varied between Tables 1 and 10. Although not expected, the modified mapping described in Section 5.1 also improves T_f. Overall, the time for the entire resolution has been divided by a factor between 2.6 (for S21) and 3.3 (for DB30).

Finally, we would like to compare the performance of our improved direct solver with an iterative multigrid solver. This iterative solver is a complex biconjugate-gradient-type solver used in combination with a multigrid preconditioner and a block Gauss-Seidel smoother, see [24] for more details. It works on smaller parts of the domain depending on the sources to be processed (see discussion below). The tolerance threshold was set as $||\mathbf{MX} - \mathbf{S}||/||\mathbf{S}|| < 10^{-6}$. An algebraic multigrid preconditioner was used and each coarsening step was carried out by coarsening the previous grid by a factor of 2 in each direction. There were 5 coarsening steps in total, while the system at the coarsest grid was solved with a direct solver.

To compare the two solvers, we shall revisit results of our previous work ([27] Table 5) where both solvers were tested on the S21 matrix with two sets of sparse RHSs with sizes 968 and 3784 (much smaller than the 12340 RHSs of the experiments reported in Table 10). As in [27], the direct solver job was executed on 900 cores of EOS computer, while for an iterative solver each of 968 (or 3784) jobs was sent to a single core, all runtimes were summed up and divided by 900 (cores). The conclusion of the previous work was that the iterative solver is always better because of the slow solve phase of the direct solver that required around one second per RHS. After the improvements described in the present paper, the conclusions change dramatically. As we see from Table 11, for the moderate number of right-hand sides (< 1000), the two solvers show similar performance. However, for several thousands of RHSs, which is typical for Gauss-Newton iterations, the direct solver demonstrates a superior speed.

It is worth noting that in a standard setup of the Gauss-Newton inversion of CSEM data, the nonzero structure of the system matrix $\mathbf{M}$ as well as the number and positions of sources remain the same for all iterations. The total number of iterations in one inversion is usually between 10 and 30. Some of the proposed procedures, e.g., RHS reorderings, can therefore be reused for all iterations. On the other hand, if one uses the same system matrix $\mathbf{M}$ for all RHSs, some savings can be achieved also for iterative solvers. In particular, it is possible to treat many RHSs simultaneously as the closeness between RHS and parallel implementation considerations may benefit the convergence of block iterative methods [19, 21]. As a result, the scaling between the runtime of iterative solvers and the RHS number may be weaker than the linear scaling assumed above. However, in practice, the system matrices for the iterative solver are based only on a relatively small part of the physical model centered around the source, i.e., they are different for different sources. Therefore, achieving savings here is not trivial, though one could consider taking into

Table 11 Extrapolation of the total resolution time (seconds) for S21 on 900 cores of the EOS machine

Number of RHSs	BLR solver ($\epsilon_{\text{BLR}} = 10^{-7}$)				Iterative solver
	T_a	T_f	T_s (*)	T_{total}	(**)
968	68	*434*	*212*	*714*	803
3784	68	*434*	*829*	*1331*	3141

Comparison of direct and iterative solvers. (*) T_s is extrapolated from the measured value of 3206 seconds for 12340 RHSs, reported in Table 10. (**) The iterative solver was executed on a single core and a perfect speedup is assumed

Italicized entries emphasize the most important data in the table

account the overlapping parts of the physical domain when assembling the system matrices. Finally, in the case of BLR factorizations, an approach with several different matrices **M** depending on the considered parts of the physical domain could also be envisaged. This would however reduce the number of RHSs to be processed for each factorization, and we expect less compression due to BLR because of the smaller matrix sizes.

7 Concluding remarks

We have shown how known properties of 3D CSEM problems can be used to significantly improve performance of direct solvers at the solve phase. The demonstrated improvements are twofold: first, we reduced the computational load through the exploitation of sparsity in RHS and solution; second, we highlighted properties of the solve phase used to drive parallel algorithms for modern parallel architectures.

On the one hand, thanks to the sparse structure of EM sources, we have been able to limit the amount of computations during the forward substitution of the solve phase. For a system with 3 million unknowns and thousands of RHSs, the resulting gains in the operation count for the forward substitution is a factor of $\sim 7.6\times$ leading to a runtime reduction by a factor of $\sim 5.7\times$. These gains have been achieved by exploiting both horizontal and vertical sparsity and by reordering the columns of **S**. Furthermore, we showed that it is also possible to reduce the time of the backward step by exploiting the sparsity of the solution. Indeed, in marine CSEM applications, the solution entries belonging to the air, water, and distant parts of formation are usually of little interest. The results should be applicable to linear systems arising in other physical problems on finite-difference or finite-element grids as long as the sources are localized in space, thus leading to very sparse RHS. We mention that the exploitation of RHS sparsity has been further studied in [6], indicating that the operation count may be further decreased. However, the performance of the corresponding algorithms in a parallel context remains to be analyzed and optimized and their application is out of the scope of this article.

On the other hand, we redesigned parallelization approaches to optimize the solve phase since it becomes the most critical step for CSEM forward problem in the case of very large number of RHSs. By using solve phase metrics to better balance work and data in a parallel environment and by improving multithreading settings for large numbers of RHSs, we achieve an additional time reduction for both the forward and backward substitutions.

In the end, the overall time reduction for the solve phase is between a $2.8\times$ and $4.6\times$ factor for the tested CSEM problems. This makes direct methods very competitive against conventional iterative methods, especially for problems with numerous RHSs occurring, e.g., in the Gauss-Newton inversion.

It is also worth noting that combining the improvements on the solve phase with the use of block low-rank (BLR) approximation to speedup the factorization phase makes the modern direct solver much more powerful in essentially all respects that it used to be a few years back. Moreover, since the solve phase often remains its most computationally intensive part, an interesting direction for future work would be to exploit the BLR format of the factors also during the solve phase, as this further decreases the number of operations during the solve phase and reduces the memory footprint during factorization, which is especially critical for large-scale problems. Although a first implementation has been developed [22] to achieve these objectives, much work is still needed to optimize its performance in a parallel MPI-OpenMP environment.

Funding information This work was partially supported by the MUMPS consortium and by LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR).

References

1. Amestoy, P.R., Buttari, A., L'Excellent, J.Y., Mary, T.: On the complexity of the block low-rank multifrontal factorization. SIAM J. Sci. Comput. **39**(4), A1710–A1740 (2017). https://doi.org/10.1137/16M1077192
2. Amestoy, P.R., Buttari, A., L'Excellent, J.Y., Mary, T.: Performance and scalability of the block low-rank multifrontal factorization on multicore architectures. ACM Trans. Math. Softw. **45**(1), 2:1–2:26 (2019). https://doi.org/10.1145/3242094
3. Amestoy, P.R., Duff, I.S., Koster, J., L'Excellent, J.Y.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM J Matrix Anal Appl **23**(1), 15–41 (2001)
4. Amestoy, P.R., Duff, I.S., L'Excellent, J.Y., Rouet, F.H.: Parallel computation of entries of A^{-1}. SIAM J. Sci. Comput. **37**(2), C268–C284 (2015)
5. Amestoy, P.R., Guermouche, A., L'Excellent, J.Y., Pralet, S.: Hybrid scheduling for the parallel solution of linear systems. Parallel Comput. **32**(2), 136–156 (2006)
6. Amestoy, P.R., L'Excellent, J.Y., Moreau, G.: On exploiting sparsity of multiple right-hand sides in sparse direct solvers. SIAM J. Sci. Comput. **41**, A269–A291 (2019)
7. Amestoy, P.R., L'Excellent, J.Y., Rouet, F.H., Sid-Lakhdar, W.M.: Modeling 1D distributed-memory dense kernels for an asynchronous multifrontal sparse solver. In: High Performance Computing for Computational Science, VECPAR 2014 - 11th International Conference, Eugene, Oregon, USA, June 30 - July 3, 2014, Revised Selected Papers, pp. 156–169 (2014)
8. Avdeev, D.B.: Three-dimensional electromagnetic modelling and inversion from theory to application. Surv. Geophys. **26**(6), 767–799 (2005). https://doi.org/10.1007/s10712-005-1836-x

9. Blackford, L.S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.C.: ScaLAPACK users' guide. SIAM Press (1997)
10. Börner, R.U.: Numerical modelling in geo-electromagnetics: advances and challenges. Surv. Geophys. **31**(2), 225–245 (2010). https://doi.org/10.1007/s10712-009-9087-x
11. Constable, S.: Ten years of marine CSEM for hydrocarbon exploration. Geophysics **75**(5), 75A67–75A81 (2010). https://doi.org/10.1190/1.3483451
12. Dongarra, J.J., Du Croz, J., Duff, I.S., Hammarling, S.: Algorithm 679: a set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. **16**, 1–17 (1990)
13. Duff, I.S., Erisman, A.M., Reid, J.K. Direct Methods for Sparse Matrices, 2nd edn. Oxford University Press, London (2017)
14. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear systems. ACM Trans. Math. Softw. **9**, 302–325 (1983)
15. Ellingsrud, S., Eidesmo, T., Johansen, S., Sinha, M.C., MacGregor, L.M., Constable, S.: Remote sensing of hydrocarbon layers by seabed logging (SBL): results from a cruise offshore Angola. Lead. Edge **21**(10), 972–982 (2002). https://doi.org/10.1190/1.1518433
16. George, J.A.: Nested dissection of a regular finite-element mesh. SIAM J. Numer. Anal. **10**(2), 345–363 (1973)
17. Gilbert, J.R.: Predicting structure in sparse matrix computations. SIAM J. Matrix Anal. Appl. **15**, 62–79 (1994)
18. Gilbert, J.R., Liu, J.W.H.: Elimination structures for unsymmetric sparse LU factors. SIAM J. Matrix Anal. Appl. **14**, 334–352 (1993)
19. Hanssen, P., Nguyen, A.K., Fogelin, L.T.T., Jensen, H.R., Skaro, M., Mittet, R., Rosenquist, M., Suilleabhain, L.O., van der Sman, P.: The next generation offshore CSEM acquisition system, pp. 1194–1198. Society of Exploration Geophysicists. https://doi.org/10.1190/segam2017-17725809.1 (2017)
20. Hiner, M., Martinez, Y., Sun, S.: Delineating salt bodies with 3D CSEM technology. In: Salt Challenges in Hydrocarbon Exploration, SEG Annual Meeting Post-convention Workshop. New Orleans (2015)
21. Lötstedt, P., Nilsson, M.: A minimal residual interpolation method for linear equations with multiple right-hand sides. SIAM J. Sci. Comput. **25**(6), 2126–2144 (2004)
22. Mary, T.: Block low-rank multifrontal solvers: complexity, performance, and scalability. PhD thesis, Université de Toulouse (2017)
23. Nguyen, A.K., Nordskag, J.I., Wiik, T., Bjorke, A.K., Boman, L., Pedersen, O.M., Ribaudo, J., Mittet, R.: Comparing large-scale 3D Gauss-Newton and BFGS CSEM inversions, pp. 872–877. Society of Exploration Geophysicists (2016). https://doi.org/10.1190/segam2016-13858633.1
24. Plessix, R.E., Darnet, M., Mulder, W.A.: An approach for 3D multisource, multifrequency CSEM modeling. Geophysics **72**(5), SM177–SM184 (2007)
25. Pothen, A., Sun, C.: A mapping algorithm for parallel sparse Cholesky factorization. SIAM J. Sci. Comput. **14**(5), 1253–1257 (1993)
26. Rouet, F.H.: Memory and performance issues in parallel multifrontal factorizations and triangular solutions with sparse right-hand sides. PhD thesis, Institut National Polytechnique de Toulouse (2012)
27. Shantsev, D., Jaysaval, P., de la Kethulle de Ryhove, S., Amestoy, P.R., Buttari, A., L'Excellent, J.Y., Mary, T.: Large-scale 3-D EM modeling with a Block Low-Rank multifrontal direct solver. Geophys. J. Int. **209**(3), 1558–1571 (2017)
28. Slavova, Tz.: Parallel triangular solution in the out-of-core multifrontal approach for solving large sparse linear systems. Ph.D. dissertation, Institut National Polytechnique de Toulouse (2009). Available as CERFACS Report TH/PA/09/59
29. Streich, R.: Controlled-source electromagnetic approaches for hydrocarbon exploration and monitoring on land. Surv. Geophys. **37**(1), 47–80 (2016). https://doi.org/10.1007/s10712-015-9336-0
30. Yee, K.: Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. IEEE Trans. Antennas Propag. **14**(3), 302–307 (1966)
31. Zach, J., Bjorke, A., Storen, T., Maao, F.: 3D inversion of marine CSEM data using a fast finite-difference time-domain forward code and approximate Hessian-based optimization. In: SEG Technical Program Expanded Abstracts 2008, pp. 614–618 (2008). https://doi.org/10.1190/1.3063726

Affiliations

Patrick R. Amestoy[1,2] · Sébastien de la Kethulle de Ryhove[3,4] · Jean-Yves L'Excellent[2,5] · Gilles Moreau[5] · Daniil V. Shantsev[6]

Patrick R. Amestoy
Patrick.Amestoy@mumps-tech.com

Sébastien de la Kethulle de Ryhove
delaketh@gmail.com

Gilles Moreau
Gilles.Moreau@ens-lyon.fr

Daniil V. Shantsev
dshantsev@emgs.com

1 University Toulouse, INPT, IRIT UMR5505, Toulouse, France

2 Present address: Mumps Technologies, Lyon, France

3 EMGS, Trondheim, Norway

4 Present address: Kongsberg Defence & Aerospace, Asker, Norway

5 University Lyon, CNRS, ENS Lyon, Inria, UCBL, LIP UMR5668, Lyon, France

6 EMGS, I&I Technology Center, Oslo, Norway