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

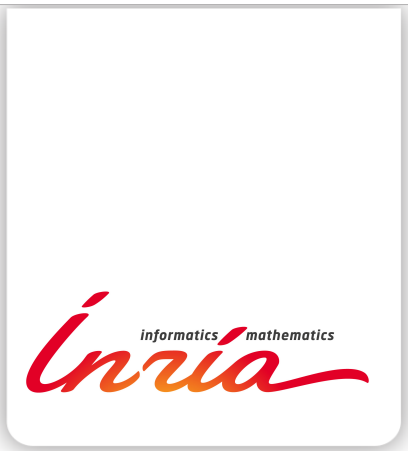
Patrick Amestoy, Sebastien de La Kethulle de Ryhove, Jean-Yves L'Excellent, Gilles Moreau, Daniil V. Shantsev

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

P. R. Amestoy, S. de la Kethulle de Ryhove, J.-Y. L'Excellent, G. Moreau, D. V. Shantsev

RESEARCH REPORT
N° 9220
Novembre 2018
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--9220--FR+ENG



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

P. R. Amestoy*, S. de la Kethulle de Ryhove†, J.-Y. L'Excellent‡, G. Moreau‡, D. V. Shantsev§

Project-Team ROMA

Research Report n° 9220 — Novembre 2018 — 27 pages

* University of Toulouse, INPT, IRIT UMR5505, F-31071 Toulouse Cedex 7, France
† EMGS, Trondheim, Norway
‡ University of Lyon, CNRS, ENS Lyon, Inria, UCBL, LIP UMR5668, F-69342, Lyon Cedex 07, France
§ EMGS, I&I Technology Center, Oslo, Norway

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Abstract: Controlled-source electromagnetic (CSEM) surveying becomes a widespread method for oil and gaz 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).

Key-words: Controlled source electromagnetics (CSEM), Electromagnetic theory, Marine electromagnetics, Numerical modeling, Direct solver, Multiple sparse right-hand sides

Exploitation efficace du creux dans les solveurs directs sur des problèmes d'électromagnétisme 3D à source contrôlée

Résumé : L'électromagnétisme à source controlée (CSEM) est une méthode de plus en plus utilisée pour l'exploration du gaz et du pétrole. L'inversion des données électromagnétiques nécessite souvent la résolution de systèmes d'équations linéaires avec un *grand* nombre de seconds membres *creux*, chacun correspondant à la position d'une source/d'un émetteur dans l'application. Les solveurs creux directs sont très attrayants pour ce type de problèmes, surtout lorsqu'ils sont combinés avec des approximations de rang faible qui réduisent la complexité et le coût de la factorisation. Comme il peut y avoir des milliers de seconds membres, le coût de la phase de résolution devient alors prédominant. Dans cet article, nous montrons qu'il est possible d'exploiter la géométrie du problème et la structure creuse qui apparaît à la fois dans les seconds membres et dans la matrice des solutions et que cela peut avoir un impact important sur les performances des phases de résolutions triangulaires. Pour cela, nous organisons les calculs de manière à minimiser le nombre d'opérations, tout en traitant les seconds membres par blocs qui tiennent en mémoire et qui visent à conserver au mieux le parallélisme lors de la phae de résolution. Nous adaptons aussi les algorithmes de placement et d'équilibrage de charge de la factorisation à la phase de résolution et, motivés par le grand nombre de seconds membres, montrons qu'il est aussi possible d'améliorer la localité des données et l'exploitation des threads au sein de chaque processus MPI. Au final, le temps total de la simulation CSEM peut être divisé par un facteur aux alentours de 3 sur 900 cœurs de calcul pour tous les systèmes linéaires de notre étude (de 3 à 30 millions d'inconnues et de 4 000 à 12 000 RHS).

Mots-clés : Electromagnisme à source contrôlée, Electromagnétisme marin, Simulation numérique, Méthode directe, Seconds membres creux, Seconds membres multiples

1 Introduction

It was demonstrated in 2002 that marine controlled-source electromagnetic (CSEM) method could be used to detect offshore hydrocarbon reservoirs (Ellingsrud et al., 2002). Over years the CSEM method has become an established tool for oil and gas exploration (Constable, 2010), and the technology development keeps going at a high pace (Hanssen et al., 2017). Successful interpretation of the growing volume of geophysical CSEM data, including also land EM data (Streich, 2016), 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 (Avdeev, 2005; Börner, 2010). 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 (Nguyen et al., 2016). 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 RHSs. The system of linear equations then takes the form $\boldsymbol{MX} = \boldsymbol{S}$, where $\boldsymbol{M}$ is a sparse symmetric matrix of size $n \times n$, while the RHS matrix $\boldsymbol{S}$ and solution $\boldsymbol{X}$ are of size $n \times m$. Here the system size n can be up to several millions, while the number of RHSs 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. In this case the cost is proportional to the number of right-hand sides, *i.e.*, scales with the number of EM sources. With direct methods, the matrix $\boldsymbol{M}$ is factored as a product $\boldsymbol{LDL^T}$, where $\boldsymbol{L}$ and $\boldsymbol{D}$ are respectively lower triangular and diagonal matrices. Then, for a given set of right-hand sides $\boldsymbol{S}$, one has to perform the so-called *solve phase* via a forward substitution solving a lower triangular system ($\boldsymbol{LY} = \boldsymbol{S}$), followed by a diagonal resolution ($\boldsymbol{DZ} = \boldsymbol{Y}$) and finally a backward substitution ($\boldsymbol{L^T X} = \boldsymbol{Z}$).

Direct methods are numerically robust and 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 factorization phase can be a bottleneck for large 3D problems since the number of floating-point operations scales as $O(n^2)$ and the number of nonzero entries in the factors is $O(n^{4/3})$ which also defines the complexity of the solve phase. However, in the context of CSEM applications, it has been shown (Shantsev et al., 2017) 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 block low-rank (BLR) format (Amestoy et al., 2017). Thanks to the improvements of the factorization phase due to low rank compression (further improved in (Amestoy et al., 2018) with respect to (Shantsev et al., 2017)), 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 in the context of many right-hand side vectors, 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 solution does not need to be computed in the whole geometrical domain. It was shown in (Gilbert, 1994) that the nonzero structure of $\boldsymbol{Y}$, after the forward substitution, could be predicted beforehand so that one could only target the nonzero variables and thus limit the number of operations. This was also referred to as *tree pruning* in (Slavova, 2009). In the context of computing selected entries of the inverse of a matrix (Amestoy et al., 2015), 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 in the context of the CSEM application.

In the paper we also introduce several new algorithms and techniques to improve the performance of the solve phase for CSEM applications on modern parallel architectures. In a distributed memory parallel environment, it is critical for performance how computational tasks are mapped onto the computer nodes. Mapping controls the equilibration of the work between processors and is driven by factorization phase metrics. We show that using workload metrics from the solve phase can improve the overall performance of the CSEM simulation. Finally, in order to benefit from good 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 for CSEM applications. We consider direct methods based on a multifrontal approach (Duff et al., 2017; Duff & Reid, 1983) even though 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 4, 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 5 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

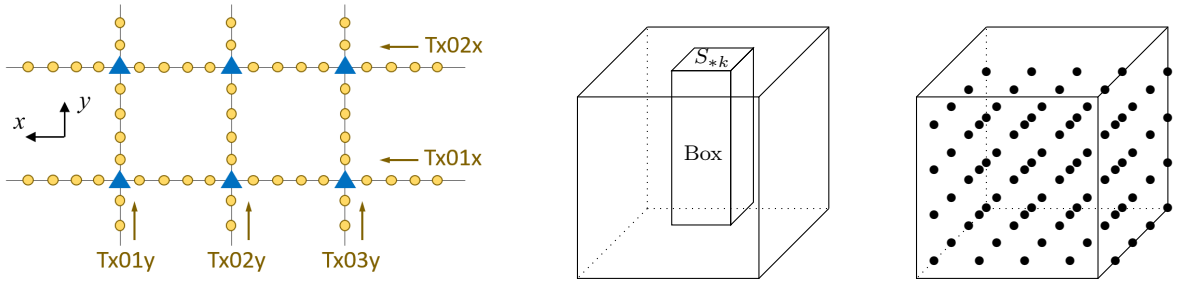
2.1 Finite difference electromagnetic modeling

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

$$\nabla \times \nabla \times \boldsymbol{E} - i\omega\mu\bar{\boldsymbol{\sigma}}\boldsymbol{E} = i\omega\mu\boldsymbol{J}, \tag{1}$$

where $\boldsymbol{E}$ is the electric field, $\bar{\boldsymbol{\sigma}}$ 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, E_z at each grid point and can be approximated by solving linear systems of the form $\boldsymbol{M}\boldsymbol{X} = \boldsymbol{S}$, where $\boldsymbol{M}$ is a sparse matrix of order $n = 3N$ and $\boldsymbol{S}$ results from the right hand-sides in Equation (1). $\boldsymbol{M}$ can easily be made symmetric. Let us now discuss the properties of RHSs and 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 seafloor 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 Figure 1(a), where receiver locations are indicated with blue circles. Yellow circles along the towlines indicate transmitter



(a) Sources layout (b) Boxing. (c) Sampling.

Figure 1: (a) Schematic example of a CSEM survey with locations of receivers (blue triangles) and transmitter positions (brown circles). (b) Entries of the final solution are those inside the "box" centered around the corresponding source S_{*k}. (c) Entries of the final solution correspond to a subset of nodal points uniformly distributed in the domain.

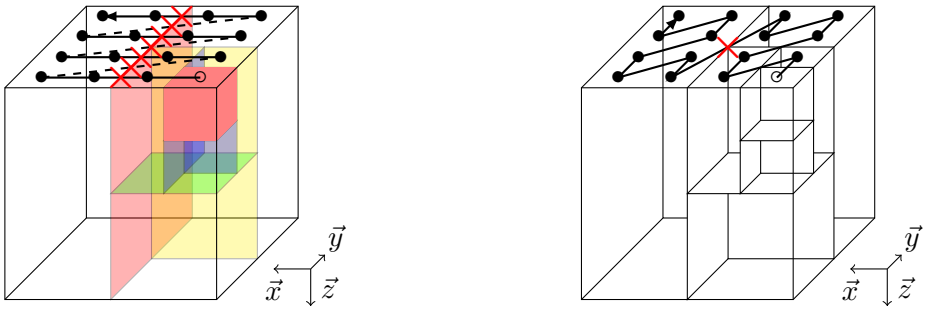
locations: it is usually assumed that distinct transmitter positions are spaced by 100 m. Since the transmitter is moving, while seabed receivers are fixed, the number of transmitters n_t is much larger (by 1–2 orders of magnitude) than the number of receivers n_r.

The number of right-hand sides is determined by the inversion algorithm used to analyze CSEM data. The gradient-based (BFGS) scheme (Zach et al., 2008) 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 (Nguyen et al., 2016). 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 ~ 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 defining order of the columns in $\boldsymbol{X}$ usually reflects the transmitter trajectory. In this article the ordering of transmitter positions obeys the following simple rule, see Figure 1(a). We start with towline Tx01x, and there go over all transmitter positions in the $\vec{x}$ direction, see Figure 1(a). Then we switch to towline Tx02x and follow the same ordering, and so on until we reach the last $\vec{x}$-directed towline. Then we switch to $\vec{y}$-directed towlines, starting with Tx01y, and for each of them go over all transmitter positions in the direction of $\vec{y}$ 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-Newtom scheme, one also has to handle right-hand sides related to receiver positions. However their number is much smaller since $n_t \gg n_r$, and therefore we have not included them in the analysis below for the sake of simplicity.

Only a subset of entries in the solution $\boldsymbol{X}$ 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 Figure 1(b). The box excludes the top of the domain: the air layer since its resistivity is known, and also the water layer since water conductivity is usually measured during the CSEM survey. The EM fields decay with



(a) Initial source order. (b) Optimized source order.

Figure 2: (a) A computational 3D domain where sources are marked with • or ∘ and connected with a line representing the transmitter trajectory. A hierarchical domain decomposition with 2D plane-shaped separators is applied, the 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.

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 × 25 km. All the regions beyond this box, in particular, the perfectly matched layers at the edges, are excluded from the solve phase. Depending on the problem, the box may represent around one half of the whole computational domain.

Reducing the number of inversion parameters can make the Gauss-Newton inversion faster. 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 $\boldsymbol{X}$ 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. 1(c), where only every 20th or every 100th point is included into the subset.

In the next paragraphs we give some preliminary hints on how sparsity described above can be used to reduce the solver complexity.

2.2 Impact of the source structure

In this section we focus on the forward substitution, and on the sparsity in the source vectors $\boldsymbol{S}$. The discussion on exploiting sparsity in the backward substitution relies on similar ideas but is postponed to Section 5.

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 (George, 1973) where the mesh is divided into subdomains and separators. A separator can be defined as a set of nodal points the removal of which divides the domain (or subdomains) into two balanced and *disjoint* subdomains. In a regular 3D grid such as the one represented in Figure 2, the separator shapes are planes with normals in the $\vec{x}, \vec{y}$ or $\vec{z}$ directions. In Figure 2(a), we represented with (red, yellow, green, ...) colors the successive *top separators* that led to the subdomain corresponding to the red subcube in the top 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 subdomains. In other words, for the computation of $\boldsymbol{Y}$, we have to consider only the given subdomain and *top separators*. In Figure 2(a), the contribution of the first source is represented by its subdomain (red cube), top separators (colored planes), while all other parts

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

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	H3	$114 \times 114 \times 74$	2,885,112	12.9	8000	7.5	10 (1%)	34 (4%)	806 (**95%**)	850
water	H17	$214 \times 214 \times 127$	17,448,276	12.9	8000	6	56 (1%)	378 (8%)	4133 (**91%**)	4567
SEAM	S21	$181 \times 160 \times 237$	20,590,560	12.9	12340	9.5	68 (1%)	476 (6%)	7819 (**93%**)	8363
DayBreak	DB30	$230 \times 422 \times 102$	29,700,360	12.9	3914	7.6	106 (2%)	765 (15%)	4246 (**83%**)	5117

of the domain will stay out of its area of influence. Section 3 is dedicated to the exploitation of this feature.

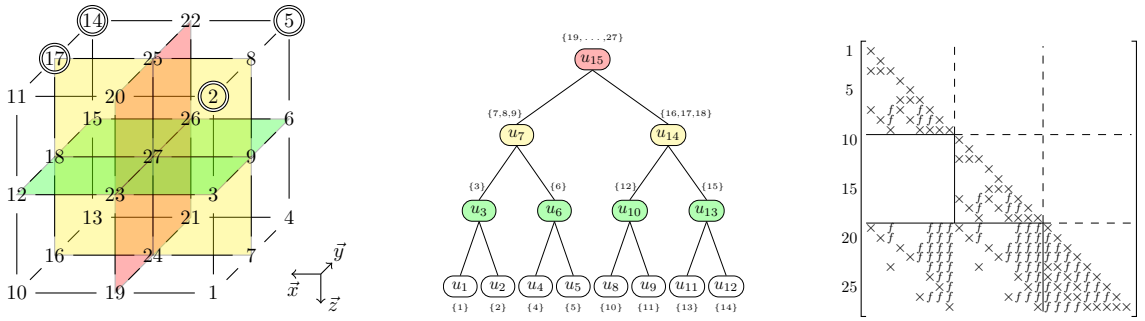
Furthermore, we will show that the initial ordering of sources is not optimal with respect to the operation counts, especially if one aims at processing simultaneously many sources. A simplified example of initial ordering is depicted in Figure 2(a) showing the source locations (symbols) and their connecting line, "transmitter trajectory". The important message here is that the transmitter trajectory crosses the top separator multiple times. We will show that the optimal ordering will be such that the transmitter trajectory has the smallest possible number of crossings of top separators. The transmitter trajectory displayed in Figure 2(b) will be shown in Section 3 to possess 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 finite-difference Yee grids with a uniform core 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 (Shantsev et al., 2017). 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 (Hiner et al., 2015).

The RHSs are generated by listing all transmitter positions using the ordering indicated in Figure 2(a). 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 spacing, which resulted in 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 $D(\boldsymbol{S})$ 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 (Amestoy et al., 2018) on the CALMIP supercomputer EOS (https://www.calmip.univ-toulouse.fr/), which is a BULLx DLC system composed of 612



(a) $3 \times 3 \times 3$ regular mesh. (b) Assembly/separators tree. (c) Structure of $\boldsymbol{L}$.

Figure 3: (a) A 3D regular mesh based on a 7-point stencil; each node is numbered according to the nested dissection algorithm following a postorder. Nodes marked by double circles are elementary sources modelling two stacked sources from the CSEM application. (b) Resulting separators or assembly tree; numbers in brackets show the sets of variables to be eliminated at each node. (c) Corresponding matrix with initial nonzeros ($\times$) in A and fill-in (f) in L.

computing nodes, each composed of two Intel Ivybridge processors with 10 cores (total 12 240 cores) running at 2.8 GHz, with 64 GBytes of memory per node. As mentioned earlier, the introduction of low-rank approximations has significantly reduced the factorization time (Shantsev et al., 2017), 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 subsection, 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.

2.4 Solve phase algorithms

We first describe the algorithms used to solve the linear systems $\boldsymbol{MX} = \boldsymbol{S}$, where $\boldsymbol{M} = \boldsymbol{LDL}^T$, 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.

$\boldsymbol{L}$ is a unit lower triangular sparse matrix of order n whereas $\boldsymbol{S}$ 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 can be written as $\boldsymbol{LY} = \boldsymbol{S}$.

We assume that $Y_{*k} \leftarrow S_{*k}$ for each column k so that all our algorithms can be expressed only in terms of modifications of Y_{*k}. The first version of our forward algorithm is a scalar two-loop algorithm limited to nonzero entries in $\boldsymbol{L}$.

$$
\begin{array}{l}
Y_{*k} \leftarrow \boldsymbol{L^{-1}} Y_{*k} \quad \text{(Scalar two-loop algorithm)} \\
\qquad \textbf{for } j = 1, \ldots, n-1 \\
\qquad\qquad \textbf{for } i \geq j+1 \text{ such that } l_{ij} \neq 0 \\
\qquad\qquad\qquad y_{ik} \leftarrow y_{ik} - l_{ij} y_{jk}
\end{array}
\tag{2}
$$

The algorithm described in (2) exploits the fact that the diagonal of $\boldsymbol{L}$ is the identity, and that $\boldsymbol{L}$ is sparse, *i.e.*, many of the l_{ij} entries are zero. Based on the example of Figure 3, we explain in the following how sparsity in $\boldsymbol{L}$ 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.

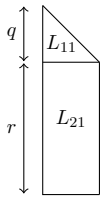


Figure 4: Structure of the factor associated with a node from the separator tree.

Figure 3(a) provides 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 Figure 3(c). For the right-hand side matrix $\boldsymbol{S}$, 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 $\vec{z}$-plane 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 and then 2-5-17-14, which is similar to the ordering shown in Figure 1(a) and convenient for illustrative purposes. The matrix $\boldsymbol{S}$ has the structure depicted in Figure 5(a). We will reuse this matrix of sources in Section 3.

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 $m_{ij} = 0$ turning into an entry $l_{ij} \neq 0$ in the factor matrix $\boldsymbol{L}$). Initially, the process builds a separator that divides the domain into two disjoint and *independent* subdomains, see Figure 3(a). 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 Figure 3(b). The two subtrees characterize the aforementioned independence between the variables of both subdomains which is reflected by the empty square in the structure of $\boldsymbol{L}$ corresponding to rows $[10, 18]$ and columns $[1, 9]$ in Figure 3(c). From the algorithm corresponding to Equation (2), the computation of the components of $\boldsymbol{Y}$ inside each subdomain will then be independent from each other. For $i \in [10, 18]$ and $j \in [1, 9]$ we have $l_{ij} = 0$ so that, for any k, and for any $i_1 \in [1, 9]$ and $i_2 \in [10, 18]$ component $y_{i_1 k}$ does not depend on component $y_{i_2 k}$. The separator tree 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 $\boldsymbol{L}$ factor, as illustrated in Figure 4. Each front 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 $\boldsymbol{Y}$ or $\boldsymbol{X}$ 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 $\boldsymbol{Y}_1 \leftarrow \boldsymbol{L}_{11}^{-1}\boldsymbol{Y}_1$ and $\boldsymbol{Y}_2 \leftarrow \boldsymbol{Y}_2 - \boldsymbol{L}_{21}\boldsymbol{Y}_1$, whereas the backward substitution is a top-down process which performs, for each front, the block operations $\boldsymbol{X}_1 \leftarrow \boldsymbol{X}_1 - \boldsymbol{L}_{21}^T\boldsymbol{X}_2$ and $\boldsymbol{X}_1 \leftarrow \boldsymbol{L}_{11}^{-T}\boldsymbol{X}_1$. Here, $\boldsymbol{Y}_1, \boldsymbol{Y}_2, \boldsymbol{X}_1, \boldsymbol{X}_2$ are partial matrices of $\boldsymbol{Y}$ and $\boldsymbol{X}$ containing only the variables corresponding to the $q+r$ rows 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 Figure 4), operations reported in Equation (2) can be performed on dense matrices and the condition "$l_{ij} \neq 0$" is replaced by "i in the structure of the factors at node $u(j)$", as will be indicated in Equation (3). Furthermore, in the context of sparse RHSs y_{*k} might remain equal to zero so that Equation (2) should

perform the update of y_{ik} only for nonzero entries y_{jk}, Equation (2) becomes:

$$\begin{array}{l} \boldsymbol{Y} \leftarrow \boldsymbol{L^{-1}Y} \quad \text{(Nodal algorithm)} \\ \quad \textbf{for } j = 1, \ldots, n-1 \\ \quad\quad \textbf{for } i \text{ in the structure of the factors at node } u(j), i > j \\ \quad\quad\quad \textbf{if } (y_{jk} \neq 0) \quad y_{ik} \leftarrow y_{ik} - l_{ij} y_{jk} \end{array} \tag{3}$$

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 $\boldsymbol{DZ} = \boldsymbol{Y}$ 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

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 L_{*j} was not needed and the two-loop algorithm (Equation 2) could be simplified. Furthermore, since $\boldsymbol{S}$ is sparse, some elements y_{jk} in Equation (3) may remain equal to zero. Similarly one would like to avoid systematic testing for $y_{jk} \neq 0$ at each update of the nodal algorithm (Equation 3). 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 ($\boldsymbol{LY} = \boldsymbol{S}$), but the same ideas can be applied to the backward substitution ($\boldsymbol{L}^T\boldsymbol{X} = \boldsymbol{Z}$) when a partial solution is needed, as will be discussed in Section 5.

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

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) of $\boldsymbol{Y}$, referred to as *horizontal* sparsity;

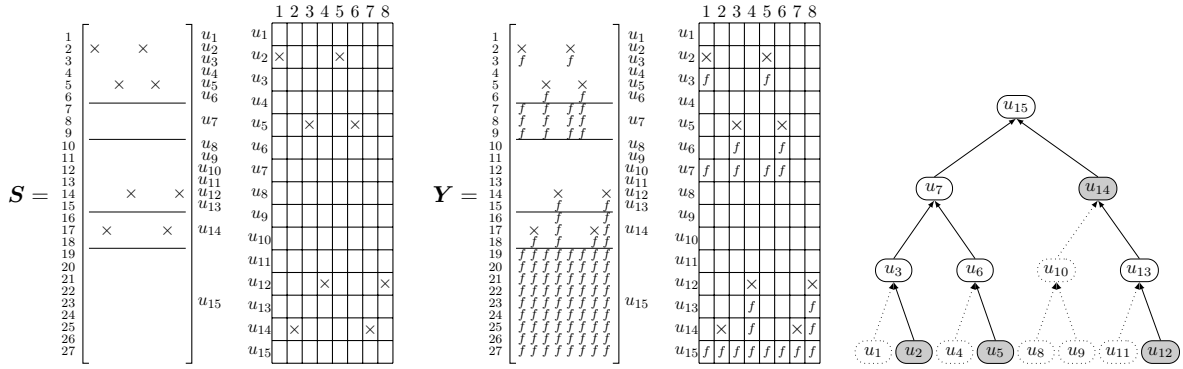
finally, find a suitable column *ordering* to improve the performance of horizontal sparsity.

We describe in the following each step and also relate it to geometric/applicative interpretations.

Vertical sparsity

Exploiting vertical sparsity in the forward substitution makes use of the properties proved in (Gilbert, 1994) and was also formulated in terms of paths using the tree structure in (Gilbert & Liu, 1993). We explain in this section how it relates to our application and to the position of the sources in the CSEM application.

In Figure 2(a) 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 $\boldsymbol{S}$, the aforementioned contribution



(a) Structure of $\boldsymbol{S}$ (b) Structure of $\boldsymbol{Y} = (\boldsymbol{L})^{-1}\boldsymbol{S}$ (c) Pruned tree $T_p(\boldsymbol{S})$

Figure 5: (a) Structure of right-hand side matrix $\boldsymbol{S}$ (associated with sources described in Figure 3) and its node representation, 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, active nodes, *i.e.* nodes with sources, are shadowed.

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

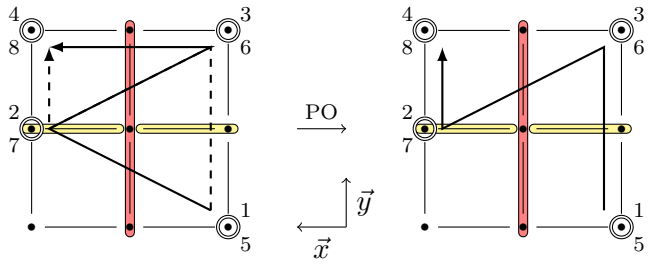
Figure 5(a) represents a matrix $\boldsymbol{S}$ composed of 8 right-hand sides associated with 8 sources (Figure 3) placed at nodes 2, 17, 5, 14, 2, 5, 17, 14, in this precise order. In our simplified model and for the sake of clarity, we considered a single nonzero element per source. To simplify the figure, we also provide a compact representation of matrix $\boldsymbol{S}$ 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 $\boldsymbol{S}$, *i.e.*, each node $u(i)$ for which there exists a column index k such that $s_{ik} \neq 0$, will be called an *active node*. Active nodes have been filled in the separator tree represented in Figure 5(c) corresponding to our simplified model.

As the solve algorithm proceeds, new nonzero entries (so called fill-in) with respect to the original entries of $\boldsymbol{S}$ appear in $\boldsymbol{Y}$. Given the initial nonzero structure of $\boldsymbol{S}$, (Gilbert, 1994) and (Gilbert & Liu, 1993) showed that it is possible to predict the nonzero structure of $\boldsymbol{Y}$. In our context, (Gilbert & Liu, 1993, Theorem 2.1) can be translated into:

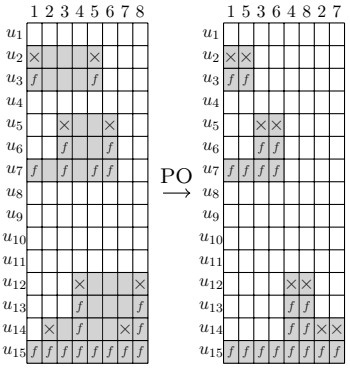
Theorem 1. *When solving $\boldsymbol{L}Y_{*k} = S_{*k}$, the structure of the vector Y_{*k} is given by the union of the variables in nodes on paths in the tree T from the set of* active nodes *of S_{*k} up to the root.*

As a consequence, a component y_{ik} will be different from zero if and only if $s_{ik} \neq 0$ or there exists an $s_{jk} \neq 0$ such that either $u(j) = u(i)$ or $u(j)$ is a descendant of $u(i)$ in T. Equation (3) is only computed for variables j belonging to such nodes. This a priori knowledge gives the possibility to *prune* nodes from the separator tree, the process is referred to as *tree pruning* in (Slavova, 2009). As an example, take S_{*1} from Figure 5(a) 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 Figure 2(a).

Furthermore, to enhance the performance of the solve phase, computation should be done on multiple columns at the same time. In doing so, one can benefit from the use of BLAS 3 operations (Dongarra et al., 1990) that can often 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 $\boldsymbol{S}$ as one block, it is noted $T_p(\boldsymbol{S})$ and shown in Figure 5(c). Therefore, Equation (3)



(a) Initial column ordering (1-2-3-4-5-6-7-8, left, corresponding to nodal points 2-17-5-14-2-5-17-14 in Figure 3(a)) and Postorder (1-5-3-6-4-8-2-7, right).



(b) Initial (left) and Postorder (right) orderings.

Figure 6: (a) A 2D section from Figure 3(a) located on the plane containing the source positions. The numbering of nodal points has been omitted for clarity. (b) Illustration of horizontal sparsity with node intervals and influence of the ordering of columns on sparsity.

becomes:

$$\begin{array}{l} \boldsymbol{Y} \leftarrow \boldsymbol{L^{-1}Y} \quad \text{(Pruned tree nodal algorithm)} \\ \quad \textbf{for } Y_{j*} \neq 0, 1 \leq j \leq n-1 \text{ (i.e., variable } j \text{ belongs to the pruned tree)} \\ \quad\quad \textbf{for } i \text{ in the structure of the factors at node } u(j), i > j \\ \quad\quad\quad Y_{i*} \leftarrow Y_{i*} - l_{ij} Y_{j*} \end{array} \tag{4}$$

where Y_{j*} is the j-th row of $\boldsymbol{Y}$ and $Y_{j*} \neq 0$ means that *at least one* of its component is different from 0. In Figure 5(c), each pruned node in $T_p(\boldsymbol{S})$ corresponds to an empty row in $\boldsymbol{Y}$, this is why sparsity is exploited *vertically*.

Horizontal sparsity and column ordering

Equation (4) assumes that all columns are processed at each node of the pruned tree. However, sources do not all have the same structure and thus it is possible to further exploit sparsity by reducing the number of columns on which Equation (4) is applied. This will be referred to as *horizontal sparsity*. To do so, we define the set of active columns and explain how and why the notion of node intervals combined with column ordering introduced for computing selective entries of the inverse of a matrix (Amestoy et al., 2015) can be effective in our context to reduce the number of operations. We illustrate these aspects in Figure 6 on the same simplified example with 8 sources as in the previous section.

The subset of columns of $\boldsymbol{S}$ that possesses at least one nonzero element at a given node $u \in T$ is called the set of *active columns* at node u. For example for node u_5, corresponding to row u_5 in Figure 6(b) (left), there are only two active columns: 3 and 6. Ideally, one would like to operate only on these two columns which would require complex data reorganizations or the computation of the columns one after the other. This would not be efficient since processing simultaneously a block of columns is 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 Equation (4) 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 includes only 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 columns with close initial nonzero structure or, equivalently, to limit the crossing of top separators as was mentioned in relation to Figure 2(b). The permutation used in this study is called a *postorder* and is built as follows: let the tree be numbered following a postordering, as in Section 2.4. For a column k of $\boldsymbol{S}$, we define $u_{rep}(k)$ as the node among the active nodes for column S_{*k} ($\{u(i), s_{ik} \neq 0\}$) that appears first in the postordering of the tree. We have for example $u_{rep}(1) = u_2$ and $u_{rep}(2) = u_{14}$ in Figure 5(a) (and 6(b), left), and call $u_{rep}(k)$ the *representative* node of column k.

Now $\boldsymbol{S}$ is said to be *postordered* if and only if: $\forall k_1, k_2, 1 \leq k_1 < k_2 \leq m$, $u_{rep}(k_1)$ appears before (or is identical to) $u_{rep}(k_2)$ in the postordering. In other words, the order of the columns S_{*k} and the postordering of their representative nodes $u_{rep}(k)$ are compatible.

In Section 2.1, we said that the postorder trajectory should minimize the number of crossing of top separators. We see in Figure 6(b) that the postorder trajectory can also be interpreted in terms of nonzero structure of $\boldsymbol{S}$ and $\boldsymbol{Y}$. Namely, it corresponds to ordering the columns of $\boldsymbol{S}$ in such a way that two successive columns have similar nonzero structure (in $\boldsymbol{Y}$). Indeed, the initial transmitter trajectory, see Figure 6(a), first implies a "superposition" of non-successive sources in $\boldsymbol{S}$. Thus, columns with close positions were not initially close in the column ordering. This resulted in large interval sizes, see rows u_7 and u_3 in Figure 6(b) (left). The postorder heuristic addresses this problem, see Figure 6(b) (right) and is optimal in this case since the gray areas do not include zero entries anymore. 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 (Amestoy et al., 2015). On our CSEM application, each source has more than one entry per column, 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. The next section shows how to still exploit parallelism efficiently in the solve phase, even when dealing with sparse right-hand sides.

4 Improving the parallel aspects of the solve algorithms

In this section, we first explain the differences between the factorization and the solve phase 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.

4.1 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 (Amestoy et al., 2006). 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. Computation is thus concentrated near the top of the tree and this is more true for the factorization than for the solve. This effect is illustrated in Figure 7, which compares the distribution of computations in the separator tree for both phases. At a depth where 50% of the computation is completed for the solve phase, only 20% is completed for the factorization.

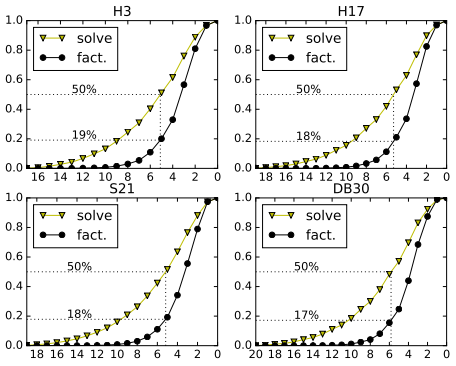
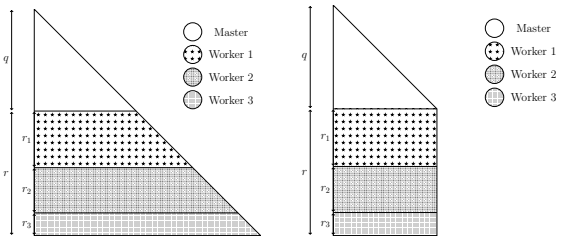


Figure 7: Normalized operation count of the solve and factorization phases as a function of the depth in the separator tree, with $depth(root) = 0$. The nested dissection ordering has been used.



(a) Front during factorization (triangular frontal matrix).

(b) Front during solve (factors).

Figure 8: Mapping of the rows of a front to balance the workload of the factorization between processors.

This indicates that the solve phase will benefit more from tree parallelism than the factorization phase.

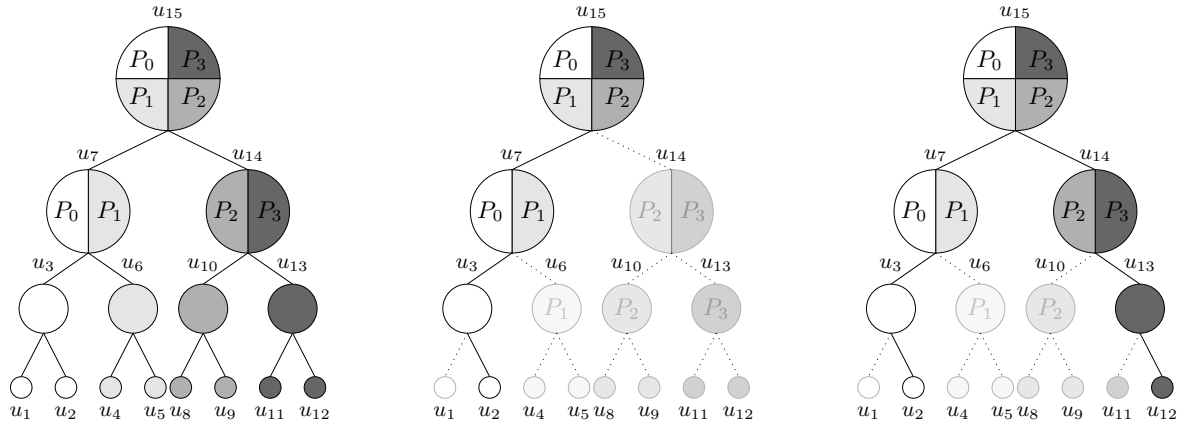
Tree pruning limits the number of branches of the separator tree that can be computed independently, thus tree parallelism and tree pruning introduced to exploit RHS sparsity are two conflicting objectives. A classical approach to balance the workload between the processors during the factorization is to use a *proportional mapping* (Pothen & Sun, 1993). Starting from the root node to which all processors are allocated and going down the tree, at each level of the tree the list of processors of the current node is partitioned between its sons according to the load of each of the subtrees rooted at each son. This is referred to as strict proportional mapping and is illustrated in Figure 9(a). It can be adapted or relaxed in order to allow for dynamic mapping and scheduling decisions, or to reduce memory usage (Amestoy et al., 2006). If the whole set $\boldsymbol{S}$ is considered, and if the set of sources is separated by the top level separators, then the width of the pruned tree $T_p(\boldsymbol{S})$ may be large enough to cover most of the tree and almost fully benefit from tree parallelism. However, because of the memory constraints mentioned in Section 2.3, the solve phase is generally processed by blocks of limited size (BLK), potentially reducing the width and parallelism of the pruned tree. In this context, it is important to decide how these blocks can be created to minimize the loss of tree parallelism introduced by the use 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 Figure 8 where q is the size of the separator and r is the number of rows to be updated. At each node the first r variables are factorized so that the number of operations is

$$W^f(q,r) = W^f_m(q) + W^f_w(q,r) = \frac{1}{6}(2q^3 + 3q^2 - 5q) + qr(q + r + 1), \tag{5}$$

where $W^f_m(q) = \frac{1}{6}(2q^3 + 3q^2 - 5q)$ corresponds to the operation count on the master processor and $W^f_w(q,r) = qr(q + r + 1)$ to the operation count on the workers. As shown in Figure 8(a), more rows must then be associated to the processors that appear first in the front. Note that we also want to adjust relative size of q and r to balance the workload between the master and each worker by splitting nodes of the separator tree (Amestoy et al., 2001, 2014).

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.



(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 $\boldsymbol{S}_2 = (S_{*1}, S_{*4})$.

Figure 9: Proportional mapping and comparison of tree coverage between three blocks of right-hand sides based on the example from Figures 3 and 5. (a) Dense RHS (all the tree is covered); (b) set of two close sources; (c) set of two distant sources.

4.2 Improving algorithms for the solve phase

Because of memory constraints, the columns of $\boldsymbol{S}$ need be processed by blocks of limited size BLK. In the scheme presented in Section 2.3, the columns of matrix $\boldsymbol{S}$ are processed using the initial ordering and then only a subpart of the 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. Figures 9(b) and 9(c) illustrate this property with two sets $\boldsymbol{S}_1$ and $\boldsymbol{S}_2$, containing close and distant sources, respectively.

To improve the tree coverage, one can select non-contiguous columns from matrix $\boldsymbol{S}$. They will better cover the physical domain because the transmitter follows a regular trajectory. Furthermore, since we also want the efficiency of BLAS-3 kernels, we propose to select each block of BLK columns such that it is composed 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 mention that within each block, we still apply a postordering permutation to maximize the effect of horizontal sparsity.

Moreover, node parallelism has an important role in the performance of the solve phase. As shown before, Figure 8(b) illustrates the distribution of data among processors when the work is balanced for the factorization. Considering the solve phase, the number of operations performed during the forward substitution (or the backward substitution) at each node is:

$$W^{fwd}(q,r) = W_m^{fwd}(q) + W_w^{fwd}(q,r) = q(q-1) + 2rq, \tag{6}$$

where $W_m^{fwd}(q) = q(q-1)$ and $W_w^{fwd} = 2rq$. As a consequence, to balance W_w^{fwd} among workers, data need to be reorganized 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 $W_m^{fwd}(q) \approx W_w^{fwd}(q, r_i)$, 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, as discussed in Section 5. In Section 6, we also experiment with another optimization of the solve phase regarding locality of data access and multithreading, which was motivated by the need to process large blocks of columns to improve tree coverage and tree parallelism.

5 Exploiting sparsity during the backward substitution

During the backward phase ($\boldsymbol{L}^T\boldsymbol{X} = \boldsymbol{Z}$), the nonzero structure of $\boldsymbol{Z}$ results from the operations performed during the forward substitution. It is preserved for $\boldsymbol{Y}$ since $\boldsymbol{DZ} = \boldsymbol{Y}$ and $\boldsymbol{D}$ is diagonal. When the matrix $\boldsymbol{M}$ is irreducible, which is the case in the CSEM application, the variables of $\boldsymbol{Y}$ belonging to the root node of the separator tree will be nonzero, independently of the position of the sources. The backward substitution processes the $\boldsymbol{L}$ matrix in a backward way which translates 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. This translates back into the fact that $\boldsymbol{Z}$ is dense and that sparsity in the sources $\boldsymbol{S}$ does not result in any reduction of the number of backward-phase operations.

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 Figure 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}. In other words and from a geometric perspective, if one assumes that i belongs to the filled subdomain of Figure 2(a) 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 can be applied to the backward substitution (Rouet, 2012, Lemma 2.2). 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 and their representative nodes in the separator tree are 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 and therefore, the column ordering chosen during the forward phase can be used during the backward phase and the choice of the blocks from Section 4.2 can be reused.

The valuable entries in each column of $\boldsymbol{X}$ 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 $\boldsymbol{X}$ is quite moderate compared to the extreme sparsity of the sources $\boldsymbol{S}$. Thus, $\boldsymbol{X}$ is a much denser matrix with less geometrically localized nonzero variables than $\boldsymbol{S}$. As illustrated in the next section, one should thus expect a smaller impact of exploiting sparsity during the backward step than during the forward step.

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

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 (Amestoy et al., 2001, 2006). We list T_f, T_s, T_{fwd}, T_{root}, T_{bwd} – the times to perform the factorization, solve, forward substitution, solve on the root node (through ScaLAPACK (Blackford et al., 1997)) and backward substitution, respectively.

6.1 Exploiting sparsity

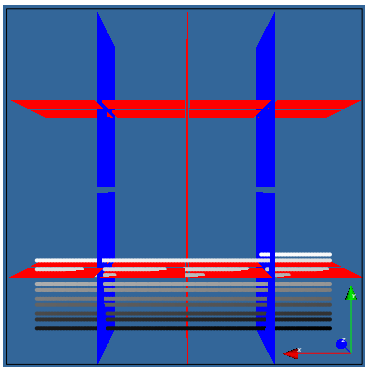
We first study vertical and horizontal sparsity, and show the impact of the choice of the columns and their order on parallelism. We consider the forward substitution in Section 6.1.1 and the backward substitution in Section 6.1.2.

6.1.1 The forward substitution

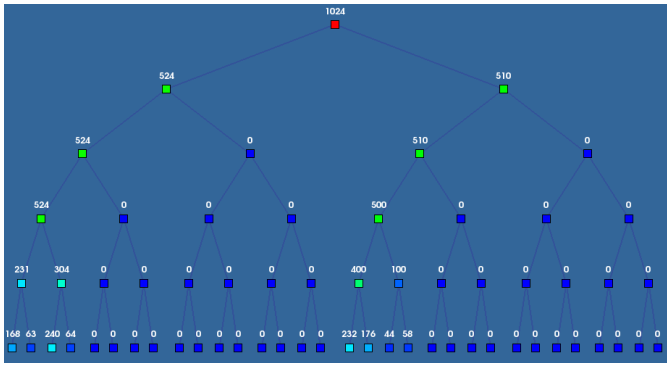
We first report in Table 2 the performance in terms of number of operations and time for solution of the proposed algorithms (dense, vertical, horizontal sparsity and postorder reordering of RHS columns) on the system H3 on 1024 contiguous columns of RHS. We mention that in the column "dense", the RHSs are still provided sparse, but sparsity is ignored so that the resulting number of operations and runtime are the same as one would have with dense RHSs. As expected from the theory (compare columns 3 and 4 of Table 2) using vertical sparsity significantly reduces the number of operations with respect to processing dense RHS. Adding horizontal sparsity and postordering of the columns further reduces the number of operations. However, as shown in Table 3, this operation reduction is not fully converted into time reduction and most notably for the operations reduction due to vertical sparsity.

Let us illustrate with Figure 10 the conflicting objectives of vertical sparsity and performance and explain how to address this issue. With the initial order of the columns in $\boldsymbol{S}$, the 1024 first ones (set $\boldsymbol{S}_1$) are located at the low y part of the horizontal plane containing the sources, see Figure 10(a), and appear in the order described in Figure 2(a). From an algebraic point of view, the effect of tree pruning (see Figure 10(b)) is that $T_p(\boldsymbol{S}_1)$ is quite narrow (many branches have no active columns). On the contrary, choosing 1024 non-contiguous columns spreads the RHSs in the domain, as illustrated in Figure 10(c) with the set $\boldsymbol{S}_2$ consisting of sets of 16 columns in $\boldsymbol{S}$ separated by 109 columns. The first consequence of such a distribution is a wider pruned tree. Indeed, comparing $T_p(\boldsymbol{S}_1)$ and $T_p(\boldsymbol{S}_2)$ from Figures 10(b) and 10(d), we see that the top subdomains are not filled with sources from $\boldsymbol{S}_1$ so that the pruned tree $T_p(\boldsymbol{S}_1)$ contains less nodes than $T_p(\boldsymbol{S}_2)$.

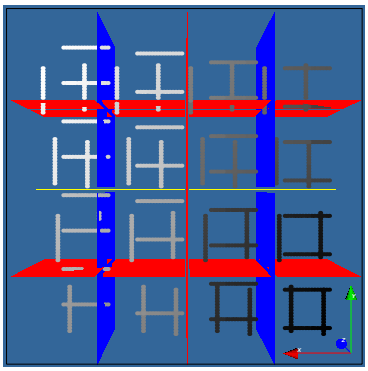
As a consequence, when only vertical sparsity is used, one can expect a larger number of operations with $\boldsymbol{S}_2$ than with $\boldsymbol{S}_1$ (compare columns "Vertical sparsity" of Tables 2 and 4). However it is also interesting to observe that the exploitation of horizontal sparsity *combined*



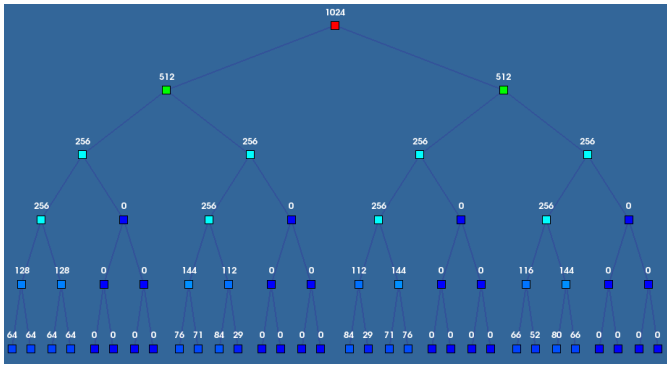
(a) Separators and set of 1024 contiguous sources ($\boldsymbol{S}_1$).



(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.

Figure 10: Geometrical and algebraic RHS distribution for two subsets of 1024 columns for system $H3$. (a) and (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 postorder. The color gradient indicates the index of the column (source) in the (possibly reordered) set of RHS columns. (b) and (d) are respectively the corresponding top 6 layers of the separator tree with, for each node, the number of active columns, as defined in Section 3.

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-	16 cols, gap 109 ($\boldsymbol{S}_2$)	951	428	306	125
Contiguous	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-	16 cols, gap 109 ($\boldsymbol{S}_2$)	170	111	98	30
Contiguous	32 cols, gap 218 ($\boldsymbol{S'}_2$)	169	107	96	31

with a postordering of the columns of RHS 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, 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 Figure 10(c) 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 $\boldsymbol{S}_1$, we have a 17% reduction in the number of operations with horizontal sparsity, reaching 45% when postordering is applied. With non-contiguous columns, the operation reduction due to horizontal sparsity and postordering reaches 71% (see Table 4).Thus, even in the case of non-contiguous columns, the number of operations is comparable (even slightly smaller) than with contiguous columns (compare last columns of Tables 2 and 4). Non-contiguous columns also expose the forward step to more parallelism and thus the time for the forward step with contiguous columns (already divided by a factor of three with respect to dense RHS processing, compare last and third columns of Table 3) is further divided by a factor of two (see 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 substitution, avoiding any RHS permutation between the forward and the backward phases. We only show results with non-contiguous sets of columns which enable, as in the forward phase, to better exploit parallelism. To measure the time and the number of operations, we exploit the fact that performing the backward substitution ($\boldsymbol{L}^T\boldsymbol{X} = \boldsymbol{Z}$) while computing only a subset of the entries of the solution $\boldsymbol{X}$ is equivalent in terms of operations, computation kernels used and parallelism, to performing the forward substitution $\boldsymbol{LY} = \boldsymbol{X}$ exploiting the sparsity of the right-hand side $\boldsymbol{X}$. 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 more moderate gains due to sparsity compared to the forward phase. Indeed, the model of Figure 1(b) and the RHS distribution of Figure 10(c) show that most of the domain is concerned by the solve phase and thus most of the nonzero structure will be concerned. Hence, 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 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 $\boldsymbol{X}$ brings an approximate 1.35x gain on the time for the backward substitution. These gains are significant and will be included in the global results of Section 6.3.

Table 6: Number of operations (OPS$\times 10^{10}$) for the backward substitution for 1024 non-contiguous RHS of system H3. Except for column "Dense", only a subset of the solution is computed with coarse solution vector sampling applied.

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

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

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

Table 8: Effect of different mapping strategies on the time for the factorization and solve phases for H3, on EOS, with 1024 RHS (set $\boldsymbol{S}_2$), using 32 MPI processes and 1 thread per MPI process.

Mapping strategy	Fact. time T_f	Forward sparsity	Solve time T_s	Solve time $T_{fwd}+T_{root}+T_{bwd}$
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

6.2 Improvement through load-balancing and multithreading

Load-balancing is performed 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 Figure 11 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 Figure 7 that the solve phase offers a greater potential for exploiting tree parallelism than the factorization phase. We see in Figure 11 that at depth five, when 50% of the computation is performed with dense RHS, only 23% of the computation is performed using horizontal sparsity and only 7% when all optimizations are used. 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 4.2.

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 seconds down to 19 seconds, showing a significantly larger relative gain than the one obtained during the (dense) backward substitution, from 167 down to 139 seconds. This is coherent with the observation reported in Figure 11 that node parallelism is more critical when sparsity is

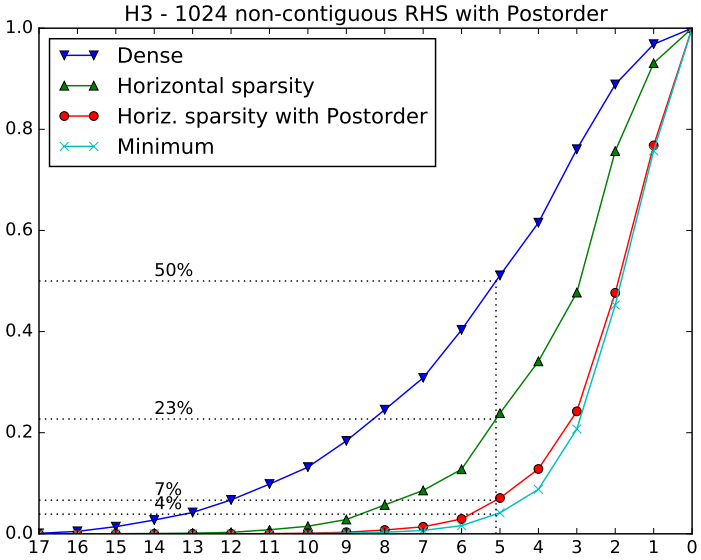


Figure 11: Normalized accumulation of operations by levels in the separator tree for the forward elimination with 1024 non-contiguous RHS permuted in postorder ($\boldsymbol{S}_2$) for H3. `Minimum` corresponds to the minimal number of operations, if all operations on zeros were avoided.

Table 9: Effect of locality and multithreading optimizations (ThreadOpt) on T_s (seconds) to process the RHS columns $\boldsymbol{S}_2$ of system H3. T_s is reported as a 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.

# 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

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,

Table 10: Time (seconds) of the analysis, factorization and solve phases on 90 MPI × 10 threads **with** and **without** all the improvements described in the study. Dense timings of the backward substitution have been divided by 1.35, see Section 6.1.2. The numbers in parenthesis indicate the percentage of T_{total}.

Statistics with improvements				
	H3	H17	S21	DB30
T_a	10 (4%)	56 (3%)	68 (2%)	106 (8%)
T_f	31 (11%)	380 (24%)	434 (16%)	510 (37%)
T_s	233 (85%)	1163 (73%)	2284 (82%)	755 (55%)
T_{fwd}	73 (27%)	289 (18%)	759 (27%)	184 (13%)
T_{root}	14 (5%)	190 (12%)	326 (12%)	80 (6%)
T_{bwd}	146 (53%)	684 (43%)	1199 (43%)	491 (36%)
T_{total}	274	1599	2786	1371
Statistics without improvements (see details in Table 1)				
T_{total}	850	4567	8363	5117

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 $\boldsymbol{S}_2$ 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 seconds.

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 (Shantsev et al., 2017) which compared the direct approach to an iterative one.

The experimental environment is the one described in Section 2.3. Runs are performed on the EOS machine on 90 MPI ×10 threads, hence a total of 900 cores, and 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 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

Table 11: Extrapolation of the total resolution time (seconds) for S21 on 900 cores of the EOS machine. Comparison of direct and iterative solvers.

Number of RHSs	BLR solver ($\epsilon = 10^{-7}$)				Iterative solver
	T_a	T_f	T_s	T_{total}	
968	68	**434**	**170**	**672**	803
3784	68	**434**	**670**	**1172**	3141

analysis and factorization phases when all the improvements described in this article are applied. To take into account the results of Section 6.1.2 showing that sparsity can be efficiently exploited during the backward substitution, we have also divided the times for the dense backward substitution by 1.35.

Whereas the solve time represented between 83% and 95% of the complete resolution time in Table 1, its weight now only represents between 55% and 85% of the resolution time. The solve time has indeed been divided by a factor between 3.4 (for H3) and 5.7 (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 4.1 also improves T_f. Overall, the time for the entire resolution has been divided by a factor between 2.8 (for S21) and 3.7 (for DB30).

Finally, we would like to compare the performance of our improved direct solver with an iterative multigrid solver. For that purpose we shall revisit results of our previous work (Shantsev et al., 2017, table 5) where both solvers were tested on the S21 matrix with two sets of sparse RHSs with sizes 968 and 3784. The conclusion of the previous work was that the iterative solver is always better because of 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.

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 certain properties of the solve phase 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 matrix with 3 million unknowns and thousands of RHSs, the resulting gains in the operation count for forward substitution is a factor of ∼7.6x leading to a run-time reduction by a factor of ∼5.7x. These gains have been achieved by exploiting both horizontal and vertical sparsity and by reordering the columns of $\boldsymbol{S}$. Similarly, we reduced the time of the backward step by exploiting the sparsity of the solution that appears because in marine CSEM applications 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.

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, with all the improvements included, the overall time reduction for the solve phase is between a 3.4x and 5.7x factor for the tested CSEM problems. This makes direct methods very competitive against conventional iterative methods, especially for problem 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 speed up 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 (Mary, 2017) to achieve these objectives, much work is still needed to optimize its performance in a parallel MPI-OpenMP environment.

Acknowledgement

This work was partially supported 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

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

Amestoy, P. R., Guermouche, A., L'Excellent, J.-Y., & Pralet, S., 2006. Hybrid scheduling for the parallel solution of linear systems, *Parallel Computing*, **32**(2), 136–156.

Amestoy, P. R., L'Excellent, J.-Y., Rouet, F.-H., & Sid-Lakhdar, W. M., 2014. 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.

Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., & Rouet, F.-H., 2015. Parallel computation of entries of A^{-1}, *SIAM Journal on Scientific Computing*, **37**(2), C268–C284.

Amestoy, P. R., Buttari, A., L'Excellent, J.-Y., & Mary, T., 2017. On the complexity of the Block Low-Rank multifrontal factorization, *SIAM Journal on Scientific Computing*, **39**(4), A1710–A1740.

Amestoy, P. R., Buttari, A., L'Excellent, J.-Y., & Mary, T., 2018. Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures, *ACM Transactions on Mathematical Software*, To appear.

Avdeev, D. B., 2005. Three-dimensional electromagnetic modelling and inversion from theory to application, *Surveys in Geophysics*, **26**(6), 767–799.

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., 1997. *ScaLAPACK Users' Guide*, SIAM Press.

Börner, R.-U., 2010. Numerical modelling in geo-electromagnetics: Advances and challenges, *Surveys in Geophysics*, **31**(2), 225–245.

Constable, S., 2010. Ten years of marine CSEM for hydrocarbon exploration, *GEOPHYSICS*, **75**(5), 75A67–75A81.

Dongarra, J. J., Du Croz, J., Duff, I. S., & Hammarling, S., 1990. Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms., *ACM Transactions on Mathematical Software*, **16**, 1–17.

Duff, I. S. & Reid, J. K., 1983. The multifrontal solution of indefinite sparse symmetric linear systems, *ACM Transactions on Mathematical Software*, **9**, 302–325.

Duff, I. S., Erisman, A. M., & Reid, J. K., 2017. *Direct Methods for Sparse Matrices, Second Edition*, Oxford University Press, London.

Ellingsrud, S., Eidesmo, T., Johansen, S., Sinha, M. C., MacGregor, L. M., & Constable, S., 2002. Remote sensing of hydrocarbon layers by seabed logging (sbl): Results from a cruise offshore angola, *The Leading Edge*, **21**(10), 972–982.

George, J. A., 1973. Nested dissection of a regular finite-element mesh, *SIAM Journal on Numerical Analysis*, **10**(2), 345–363.

Gilbert, J. R., 1994. Predicting structure in sparse matrix computations, *SIAM Journal on Matrix Analysis and Applications*, **15**, 62–79.

Gilbert, J. R. & Liu, J. W. H., 1993. Elimination structures for unsymmetric sparse LU factors, *SIAM Journal on Matrix Analysis and Applications*, **14**, 334–352.

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., 2017. *The next generation offshore CSEM acquisition system*, pp. 1194–1198, Society of Exploration Geophysicists.

Hiner, M., Martinez, Y., & Sun, S., 2015. Delineating salt bodies with 3D CSEM technology , in *Salt Challenges in Hydrocarbon Exploration, SEG Annual Meeting Post-convention Workshop, New Orleans, 2015*.

Mary, T., 2017. *Block Low-Rank multifrontal solvers: complexity, performance, and scalability*, PhD thesis, Université de Toulouse.

Nguyen, A. K., Nordskag, J. I., Wiik, T., Bjorke, A. K., Boman, L., Pedersen, O. M., Ribaudo, J., & Mittet, R., 2016. *Comparing large-scale 3D Gauss-Newton and BFGS CSEM inversions*, pp. 872–877, Society of Exploration Geophysicists.

Pothen, A. & Sun, C., 1993. A mapping algorithm for parallel sparse Cholesky factorization, *SIAM Journal on Scientific Computing*, **14(5)**, 1253–1257.

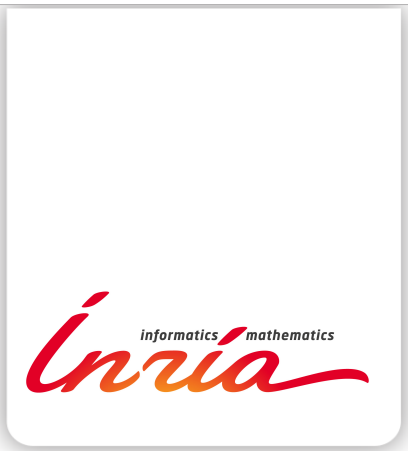
Rouet, F.-H., 2012. *Memory and performance issues in parallel multifrontal factorizations and triangular solutions with sparse right-hand sides*, PhD thesis, Institut National Polytechnique de Toulouse.

Shantsev, D., Jaysaval, P., de la Kethulle de Ryhove, S., Amestoy, P. R., Buttari, A., L'Excellent, J.-Y., & Mary, T., 2017. Large-scale 3D EM modeling with a Block Low-Rank multifrontal direct solver, *Geophysical Journal International*, **209**(3), 1558–1571.

Slavova, Tz., 2009. *Parallel triangular solution in the out-of-core multifrontal approach for solving large sparse linear systems*, Ph.D. dissertation, Institut National Polytechnique de Toulouse, Available as CERFACS Report TH/PA/09/59.

Streich, R., 2016. Controlled-source electromagnetic approaches for hydrocarbon exploration and monitoring on land, *Surveys in Geophysics*, **37**(1), 47–80.

Zach, J., Bjorke, A., Storen, T., & Maao, F., 2008. 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.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée

655 avenue de l'Europe Montbonnot

38334 Saint Ismier Cedex

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

ISSN 0249-6399