



HAL
open science

On Exploiting Sparsity of Multiple Right-Hand Sides in Sparse Direct Solvers

Patrick Amestoy, Jean-Yves L 'Excellent, Gilles Moreau

► **To cite this version:**

Patrick Amestoy, Jean-Yves L 'Excellent, Gilles Moreau. On Exploiting Sparsity of Multiple Right-Hand Sides in Sparse Direct Solvers. [Research Report] RR-9122, ENS de Lyon; INRIA Grenoble - Rhone-Alpes. 2017, pp.1-26. hal-01649244v1

HAL Id: hal-01649244

<https://inria.hal.science/hal-01649244v1>

Submitted on 27 Nov 2017 (v1), last revised 5 Dec 2017 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Exploitation de seconds-membres creux et multiples dans les solveurs creux directs

Patrick Amestoy, Jean-Yves L'Excellent, Gilles Moreau

**RESEARCH
REPORT**

N° 9122

Novembre 2017

Project-Team ROMA

ISSN INRIA/RR--9122--FR+ENG

ISSN 0249-6399



Exploitation de seconds-membres creux et multiples dans les solveurs creux directs

Patrick Amestoy*, Jean-Yves L'Excellent†, Gilles Moreau‡

Équipe-Projet ROMA

Rapport de recherche n° 9122 — Novembre 2017 — 26 pages

Résumé : Le coût des résolutions triangulaires des solveurs creux directs est parfois critique. Ce coût peut dépasser celui de la factorisation dans les applications qui nécessitent la résolution de plusieurs milliers de seconds membres. Cette étude se concentre sur les cas où les seconds membres sont multiples, creux et n'ont pas tous la même structure.

Étant donnée la factorisation $A = LU$ d'une matrice creuse A , l'étude met surtout l'accent sur la résolution du premier système triangulaire $LY = B$, où L est triangulaire inférieure. Dans ce type de problèmes, le creux dans la matrice B peut être exploité de deux manières. Premièrement, on évite le calcul de certaines lignes, ce qui correspond à élaguer certains nœuds de l'arbre d'élimination (qui représente les dépendances entre les calculs de la résolution). Deuxièmement, on réduit, pour chaque nœud, le calcul sur des sous-ensembles de colonnes de B plutôt que sur la matrice complète. Dans ce cas, un problème combinatoire doit être résolu afin de trouver une permutation des colonnes de B .

S'appuyant d'abord sur l'algorithme de dissection emboîtée appliqué à un domaine régulier, un premier algorithme est proposé pour contruire une permutation des colonnes de B . Puis une nouvelle approche permet de poursuivre la réduction du nombre d'opérations grâce à la création de blocs. Pour préserver la flexibilité de l'implémentation ainsi que l'efficacité des opérations de type BLAS 3, un nombre minimal de groupe est créé. Inspirés d'abord par des observations géométriques, ces nouveaux algorithmes ont été étendus algébriquement pour n'utiliser que des informations provenant de la structure des seconds membres et des arbres d'élimination. Ils permettent ainsi une convergence rapide vers le nombre minimal d'opérations. Les résultats expérimentaux démontrent le gain obtenu par rapport à d'autres approches classiques. Enfin, les applications et extensions possibles de ce travail sont présentées.

Mots-clés : Algèbre linéaire creuse, matrices creuses, méthode directe, seconds membres creux et multiples

* University of Toulouse, INPT and IRIT laboratory, France

† University of Lyon, Inria and LIP laboratory, France

‡ University of Lyon, Inria and LIP laboratory, France

30 **Introduction.** We consider the direct solution of sparse systems of linear equations

31 (1)
$$AX = B$$

32 where A is an $n \times n$ sparse matrix with a symmetric structure and B is an $n \times m$ matrix of right-
33 hand sides (RHS). When A is decomposed under the form $A = LU$ with a sparse direct method [7],
34 *e.g.*, the multifrontal method [8], the solution can be obtained by forward and backward triangular
35 solves involving L and U . In this study, we are interested in the situation where not only A is
36 sparse, but also B , with the columns of B possibly having different structures, and we focus on the
37 efficient solution of the forward system

38 (2)
$$LY = B$$

39 where the unknown Y and the right-hand side B are $n \times m$ matrices. We will see in this study
40 that the ideas developed for Equation (2) are indeed more general and can be applied in a broader
41 context. In particular, they can be applied to the backward substitution phase, in situations where
42 the system $UX = Y$ must be solved for a subset of the entries in X [3, 17]. In direct methods, the
43 dependencies of the computations for factorization and solve operations can be represented by a
44 tree [13], which plays a central role and expresses parallelism between tasks. The factorization phase
45 is usually the most computationally intensive phase but depending on the number of columns m in
46 B , the cost of the solve phase may also be significant. For example, electromagnetism, geophysics
47 or imaging applications lead to problems with sparse multiple right-hand sides [19, 20] for which
48 the solution phase can be significantly more costly than the factorization phase [1, 16]. Such
49 applications motivate the algorithms presented in this study.

50 A sparse RHS is characterized by its set of non-zeros and it is worth considering a RHS as sparse
51 when doing so improves performance compared to the dense case. The exploitation of RHS sparsity
52 (later extended to reduce computations when only a subset of the solution is needed [17, 19]) was
53 formalised by Gilbert [10] and Gilbert and Liu [11], who showed that the structure of the solution Y
54 from Equation (2) can be predicted from the structures of L and B . From this structure prediction,
55 one can design mechanisms to reduce computation. In particular, *tree pruning* suppresses nodes in
56 the elimination tree involving only computations on zeros. When solving a problem with multiple
57 RHS, the preferred technique is usually to process all the RHS in one shot. The subset of the
58 elimination tree to be traversed is then the union of the different pruned trees (see Section 2.1).
59 However, when the RHS have different structures, this means that extra operations on zeros have to
60 be performed. In order to limit these extra operations, several approaches may be applied. The one
61 that minimizes the number of operations consists in processing the RHS columns one by one, each
62 time with a different pruned tree. However, such an approach is not practical and leads to a poor
63 arithmetic intensity (*e.g.*, it does not exploit level 3 BLAS [6]). Another approach, in the context of
64 blocks of RHS with a predetermined number of columns in each block, consists in finding heuristics
65 to determine which columns to include within which block. The objective function to minimize
66 might be the volume of accesses to the factor matrices [3], or the number of operations [18]. When
67 memory allows it, large sets of columns, possibly the whole set of m columns, may be processed in
68 one shot. Thanks to the different sparsity structure of each column of B , it is then possible to work
69 on less than m columns at most nodes in the tree, as explained in Section 2.2. Such a mechanism
70 has been introduced in the context of the parallel computation of entries of the inverse [4], where
71 at each node, computations are performed on a contiguous interval of RHS columns.

72 After a description of these mechanisms with illustrative examples, one contribution of this
73 work is to propose algorithms that optimize the exploitation of *column intervals* at each node. A

74 geometrical intuition motivates a new approach to obtain a permutation of the columns of B that
 75 significantly reduces computation during (2) with respect to previous work. The algorithm is first
 76 introduced for a nested dissection ordering and for a regular mesh, then generalized to arbitrary
 77 elimination trees. Computation can then be further reduced by dividing the RHS into blocks.
 78 However, instead of enforcing a constant number of columns per block, our objective is to minimize
 79 the number of blocks created. If $\Delta_{min}(B)$ represents the number of operations to solve (2) when
 80 processing the RHS columns one-by-one, we show on real applications that our blocking algorithm
 81 can approach $\Delta_{min}(B)$ within a tolerance of 1% while creating a small number of blocks. Please
 82 note that RHS sparsity limits the amount of tree parallelism because only a few branches are
 83 traversed in the elimination tree. Therefore, whenever possible, our heuristics also aim at choosing
 84 the approach that maximizes tree parallelism.

85 This paper is organized as follows. Section 1 presents the general context of our study and
 86 Section 2 exposes the classical *tree pruning* technique together with the notion of *node intervals*
 87 where different intervals of columns may be processed at each node of the tree. In Section 3,
 88 we introduce a new permutation to reduce the size of such intervals and thus limit the number
 89 of operations, first using geometrical considerations for a regular nested dissection ordering, then
 90 with a pure algebraic approach that can then be applied in a general case and for arbitrary right-
 91 hand sides. We call it the *Flat Tree* algorithm because of the analogy with the ordering that
 92 one would obtain when “flattening” the tree. In Section 4, an original blocking algorithm is then
 93 introduced to further improve the flat tree ordering. It aims at defining a limited amount of blocks
 94 of right-hand sides to minimize the number of operations while preserving parallelism. Section 5
 95 gives experimental results on a set of matrices coming from two geophysics applications relying on
 96 Helmholtz or Maxwell equations. Section 6 discusses adaptations of the nested dissection algorithm
 97 to further decrease computation and Section 7 shows why this work has a broader scope than solving
 98 Equation (2) and presents possible applications.

99 **1. Nested dissection, sparse direct solvers and triangular solve.** In sparse direct
 100 solvers, the order in which the variables of a sparse matrix A are eliminated has a strong im-
 101 pact on the number of operations for the factorization, on the size of the factor matrices, and on
 102 the cost of the solve phase. We first illustrate the use of nested dissection on a regular mesh [9]
 103 with the example of Figure 1.

104 First, the nested dissection algorithm consists in dividing domains with separators. The regular
 105 $3 \times 3 \times 3$ domain shown in Figure 1(a) is first divided by a 3×3 constant- x plane separator (variables
 106 $\{19, \dots, 27\}$) into two even subdomains. Each subdomain is then divided recursively (constant- y
 107 plane separators $\{16, 17, 18\}$ and $\{7, 8, 9\}$, etc.). By reordering the matrix with separators num-
 108 bered after the subdomains, large blocks of zeros appear which limit the amount of computation
 109 (Figure 1(b)). The order of variables inside separators is similar to [9, Appendix].

110 Second, the elimination tree [13] represents dependencies between computations: it is processed
 111 from bottom to top during the factorization and forward elimination, and from top to bottom
 112 during the backward substitution. The elimination tree may be compressed thanks to the use of
 113 supernodes, leading to a tree identical to the *separator tree* of Figure 1(c) when choosing supernodes
 114 identical to the separators resulting from the nested dissection algorithm¹. This separator tree will
 115 simply be referred to as *tree* in the rest of this paper. We note that the order in which tree nodes

¹Note that in this example, identifying supernodes to separators leads to relaxed supernodes: although some sparsity exists in the interaction of u_7 and u_0 (and u_{112} and u_0), the interaction is considered dense and few computations on zeros are performed to benefit from larger blocks.

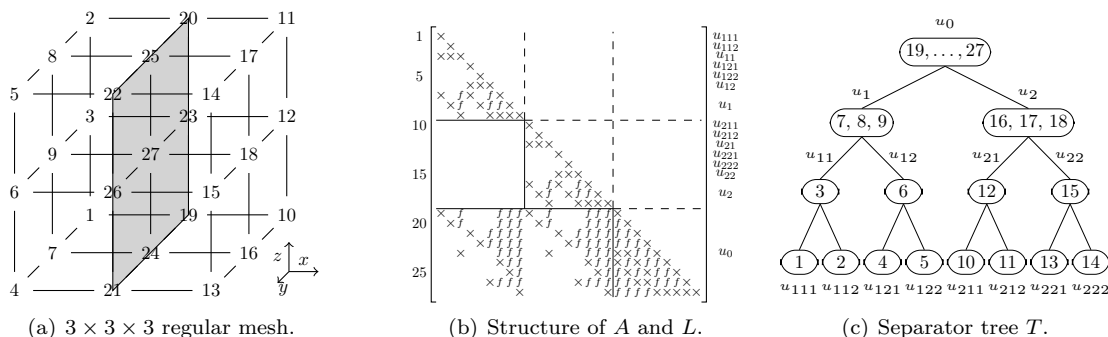


FIGURE 1. (a) a 3D regular mesh based on a 7-point stencil; mesh nodes are numbered according to the nested dissection algorithm. (b) corresponding matrix with initial nonzeros (\times) in the lower triangular part of a symmetric matrix A and fill-in (f) in the L factor. (c) separator tree; numbers inside nodes represent the set of variables to be eliminated at each node; above and below each node is the label.

116 are processed represented on the right of the matrix ($u_{111}, u_{112}, u_{11}, u_{121}, \dots, u_0$) is a postordering:
 117 nodes in any subtree are processed consecutively.

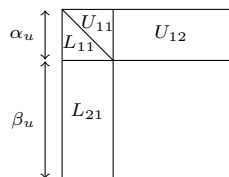


FIGURE 2. Structure of the factors associated to a node u of the tree.

118 Third, considering a single RHS b and the sparse matrix factorization $A = LU$, the solution of
 119 the triangular systems $Ly = b$ (forward elimination) and $Ux = y$ (backward substitution) can be
 120 obtained with block operations at each node of the tree. Figure 2 represents the L and U factors
 121 restricted to a given node u of the tree T , where the diagonal block is formed of the two lower
 122 and upper triangular matrices L_{11} and U_{11} , and the update matrices are L_{21} and U_{12} . The α_u
 123 variables are the ones of node (or separator) u , and the β_u variables correspond to the nonzero rows
 124 in the off-diagonal parts of the L factor restricted to node u (Figure 1(b)), that have been gathered
 125 together. For example, node u_1 from Figure 1 corresponds to separator $\{7, 8, 9\}$, so that L_{11} and
 126 U_{11} are of order $\alpha_{u_1} = 3$ and there are $\beta_{u_1} = 9$ update variables $\{19, \dots, 27\}$, so that L_{21} is of size
 127 9×3 (and U_{12} is of size 3×9).

128 Starting with $y \leftarrow b$, at each node u of the tree T , the active components of y at u can be
 129 gathered into two temporary dense vectors y_1 of size α_u and y_2 of size β_u , so that the triangular
 130 solve can be written:

131 (3)
$$y_1 \leftarrow L_{11}^{-1} y_1,$$

132 and the update operation can be written:

133 (4)
$$y_2 \leftarrow y_2 - L_{21} y_1.$$

134 y_1 and y_2 can then be scattered back into y ; y_2 will be used at higher levels in the tree until the
 135 root is processed and y contains the solution of $Ly = b$. Because the matrix blocks in Figure 2 are
 136 considered dense, there are $\alpha_u(\alpha_u - 1)$ arithmetic operations for the triangular solution (3) and
 137 $2\alpha_u\beta_u$ operations for the update operation (4), leading to a total number of operations to process
 138 the whole tree T

$$139 \quad (5) \quad \Delta = \sum_{u \in T} \mathcal{F}_u,$$

140 where $\mathcal{F}_u = \alpha_u \times (\alpha_u - 1 + 2\beta_u)$ is the number of arithmetic operations (divisions, additions,
 141 multiplications) at node u . In general, the number of operations increases a lot towards the top of
 142 the tree.

143 **2. Exploitation of sparsity in right-hand sides.** In this section, we review approaches
 144 to exploit sparsity in B when solving the triangular system (2) [11, 4, 17]. The first one, called
 145 tree pruning and explained in Section 2.1, consists in pruning the nodes of the tree where only
 146 computations on zeros are performed. The second one, presented in Section 2.2, goes further by
 147 working on different sets of RHS columns at each node of the tree.

148 **2.1. Tree pruning.** Consider a non-singular $n \times n$ matrix A with a nonzero diagonal, and
 149 its directed graph $G(A)$ (with an edge from i to j if $a_{ij} \neq 0$) [13]. Given a vector b , let us
 150 define its structure as its set of nonzeros, $\text{struct}(b) = \{i, b_i \neq 0\}$, and $\text{closure}_A(b)$ as the smallest
 151 subset of vertices of $G(A)$ including $\text{struct}(b)$ without incoming edges. Gilbert [10, Theorem 5.1]
 152 characterizes the structure of the solution of $Ax = b$ by the relation $\text{struct}(A^{-1}b) \subseteq \text{closure}_A(b)$,
 153 with equality in case there is no numerical cancellation. In our context of triangular systems,
 154 ignoring such cancellation, $\text{struct}(L^{-1}b) = \text{closure}_L(b)$ is also the set of vertices reachable from
 155 $\text{struct}(b)$ in $G(L^T)$, where edges have been reversed [11, Theorem 2.1]. Finding these reachable
 156 vertices can be done using the transitive reduction of $G(L^T)$, which is a tree (the elimination tree)
 157 when L results from the factorization of a matrix with symmetric (or symmetrized) structure.

158 Since we work with a tree T with possibly more than one variable to eliminate at each node,
 159 let us define V_b as the set of nodes in T including at least one element of $\text{struct}(b)$. The structure
 160 of $L^{-1}b$ can be obtained by following paths from the nodes of V_b upto the root and these will be
 161 the only nodes needed to compute $L^{-1}b$. The tree consisting of this set of nodes is what we call
 162 the *pruned tree* for b , and we note it $T_p(b)$. Thanks to the pruned tree, the number of operations
 163 Δ from Section 1 is now a function of b and we redefine it as:

$$164 \quad (6) \quad \Delta(b) = \sum_{u \in T_p(b)} \mathcal{F}_u.$$

165 **EXAMPLE 1.** Take a right-hand side vector b with nonzeros at positions 4, 13, and 21. The
 166 corresponding tree nodes are given by $V_b = \{u_{121}, u_{221}, u_0\}$ (see Figure 3). Following the paths from
 167 nodes in V_b to the root results in the pruned tree of Figure 3(b). Compared to $\Delta = 288$ in the case
 168 of a dense right-hand side, here $\Delta(b) = 228$ ($\mathcal{F}_{u_{121}} = \mathcal{F}_{u_{221}} = 6, \mathcal{F}_{u_{12}} = \mathcal{F}_{u_{22}} = 12, \mathcal{F}_{u_2} = \mathcal{F}_{u_1} =$
 169 $60, \mathcal{F}_{u_0} = 72$).

170 We now consider the case of multiple RHS (Equation (2)), where RHS columns may have
 171 different structures. We denote by B_i the columns of B , for $1 \leq i \leq m$. Instead of solving
 172 the m linear systems one after the other with each pruned tree $T_p(B_i)$, one generally prefers to
 173 favor matrix-matrix computations for performance reasons. For that, a first approach consists in

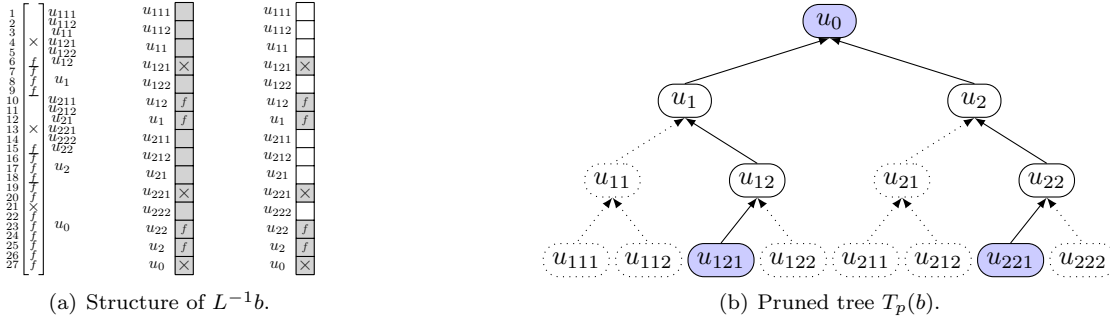


FIGURE 3. Illustration of Example 1. (a) Structure of $L^{-1}b$ with respect to matrix variables (left) and to tree nodes (middle and right). \times corresponds to original nonzeros and f to fill-in. In the dense case (middle) and the sparse case (right), grey parts of $L^{-1}b$ are the ones involving computation. (b) Pruned tree $T_p(b)$: pruned nodes and edges are represented with dotted lines and nodes in V_b are filled.

174 considering $V_B = \bigcup_{1 \leq i \leq m} V_{B_i}$, the union of all nodes in T with at least one nonzero from matrix
 175 B , and the pruned tree $T_p(B) = \bigcup_{1 \leq i \leq m} T_p(B_i)$ containing all nodes in T reachable from nodes
 176 in V_B . In that case, triangular and update operations from (3) and (4) become matrix operations
 177 $Y_1 \leftarrow L_{11}^{-1}Y_1$ and $Y_2 \leftarrow Y_2 - L_{21}Y_1$, at each node of the tree. The number of operations can then
 178 be defined as:

179 (7)
$$\Delta(B) = m \times \sum_{u \in T_p(B)} \mathcal{F}_u.$$

180 EXAMPLE 2. Figure 4(a) shows a RHS matrix $B = [\{B_{11,1}\}, \{B_{6,2}\}, \{B_{13,3}\}, \{B_{10,4}\}, \{B_{2,5}\}]$ in
 181 terms of original variables (1 to 27) and in terms of tree nodes ($V_B = \{u_{212}, u_{12}, u_{221}, u_{211}, u_{112}\}$).
 182 In Figure 4(a), \times corresponds to original nonzeros in B and f corresponds to “fill-in” that appears
 183 in $L^{-1}B$ during the forward elimination on the nodes that are on the paths from nodes in V_B to the
 184 root (see Figure 4(b)). We have $\Delta(B) = 5 \times 264 = 1320$ and $\Delta(B_1) + \Delta(B_2) + \dots + \Delta(B_5) = 744$.

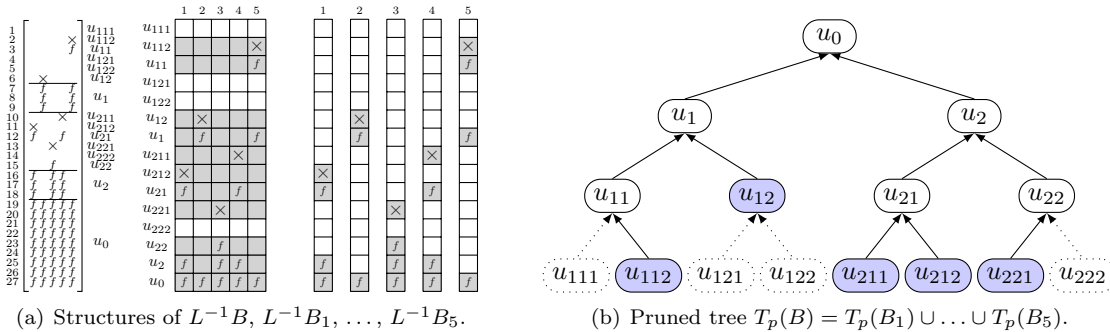


FIGURE 4. Illustration of multiple RHS and tree pruning corresponding to Example 2. Grey parts of $L^{-1}B$ (resp. of $L^{-1}B_i$) are the ones involving computations when RHS are processed in one shot (resp. one by one).

185 At this point, we exploit sparsity in the tree but perform extra operations by considering globally
 186 $T_p(B)$ instead of each individual pruned tree $T_p(B_i)$. In other words, we only exploit the *vertical*

187 *sparsity* of matrix B . Processing B by smaller blocks of columns would further reduce the number
 188 of operations at the cost of more traversals of the tree and a smaller arithmetic intensity, with a
 189 minimal number of operations $\Delta_{min}(B) = \sum_{i=1,m} \Delta(B_i)$ reached when B is processed column by
 190 column, as in Figure 4(a)(right). We note that performing this minimal number of operations while
 191 traversing the tree only once (and thus accessing the L factor only once) from leaves to root would
 192 require performing complex and costly data manipulations at each node u with copies and indirect
 193 accesses to work only on the active columns of B at u . We present in the next section a simpler
 194 approach which consists in exploiting the notion of intervals of columns at each node $u \in T_p(B)$.
 195 This approach to exploit what we call *horizontal sparsity* in B was introduced in another context [4].

196 **2.2. Exploitation of column intervals at each node.** Given a matrix B , we associate to
 197 a node $u \in T_p(B)$ its set of active columns

198 (8)
$$Z_u = \{j \in \{1, \dots, m\} \mid u \in T_p(B_j)\}.$$

199 The interval $[\min(Z_u), \max(Z_u)]$ includes all active columns, and its length is

200
$$\theta(Z_u) = \max(Z_u) - \min(Z_u) + 1.$$

201 Z_u is sometimes defined for an ordered subset R of the RHS columns in B , in which case we use
 202 the notations $Z_u|_R$, and $\theta(Z_u|_R)$. Note that for u in $T_p(B)$, Z_u is non-empty and $\theta(Z_u)$ is different
 203 from 0. In this approach, operations (3) and (4) are performed on the $\theta(Z_u)$ contiguous columns
 204 $[\min(Z_u), \max(Z_u)]$ instead of the m original columns of B , leading to

205 (9)
$$\Delta(B) = \sum_{u \in T_p(B)} \mathcal{F}_u \times \theta(Z_u).$$

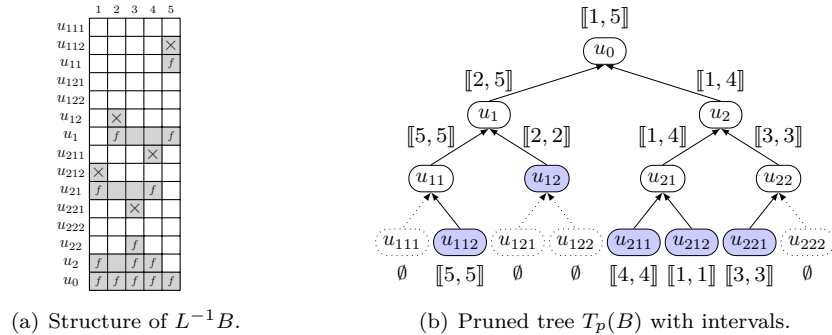


FIGURE 5. Column intervals corresponding to Example 3: in grey (a) and above/below each node (b).

206 **EXAMPLE 3.** In Example 2, there are nonzeros in columns 1 and 4 at node u_{21} so that $Z_{u_{21}} =$
 207 $\{1, 4\}$ (see Figure 5). Instead of performing the solve operations on all 5 columns at node u_{21} , we
 208 limit the computations to the $\theta(Z_{u_{21}}) = 4$ columns of interval $[1, 4]$ (and to a single column at, e.g.,
 209 node u_{221}). Overall, $\Delta(B)$ is reduced from 1320 to 948 (while $\Delta_{min} = 744$).

210 It is clear from Example 3 that $\theta(Z_u)$ and $\Delta(B)$ strongly depend on the order of the columns
 211 in B . In Section 3, we formalize the problem of permuting the columns of B and propose a new
 212 heuristic to find such a permutation. In Section 4, we further decrease the number of operations
 213 by identifying and extracting “problematic” columns.

214 **3. Permutation of right-hand side's columns.** In Section 2.2, we observed that *horizontal*
 215 *sparsity* can be exploited thanks to column intervals at each node of the tree. The number of
 216 operations to solve (2) then depends on the permutation σ of the columns in B . The minimization
 217 problem can be expressed as:

$$(10) \quad \text{Find a permutation } \sigma \text{ of } \{1, \dots, m\} \text{ that minimizes } \Delta(B, \sigma) = \sum_{u \in T_p(B)} \mathcal{F}_u \times \theta(\sigma(Z_u)),$$

218 where $\sigma(Z_u) = \{\sigma(i) \mid i \in Z_u\}$, and
 $\theta(\sigma(Z_u))$ is the length of the permuted interval $[\min(\sigma(Z_u)), \max(\sigma(Z_u))]$.

219 Although it might be possible to solve the global problem using general linear optimization algo-
 220 rithms, we propose cheaper heuristics based on the tree structure. In order to do so, we first define
 221 the notion of node optimality.

222 **DEFINITION 4.** *Given a node u in $T_p(B)$, and a permutation σ of $\{1, \dots, m\}$, we say that we*
 223 *have node optimality at u , or that σ is u -optimal if and only if $\theta(\sigma(Z_u)) = \#Z_u$, where $\#Z_u$ is*
 224 *the cardinal of Z_u . Said differently, $\sigma(Z_u)$ is a set of contiguous columns.*

225 Remark that $\theta(\sigma(Z_u)) - \#Z_u$ is the number of explicit zeros (or padded zeros) on which extra
 226 computation is done and is equal to 0 if σ is u -optimal.

227 **EXAMPLE 5.** *Take the RHS structure of Figure 5(a) and the identity permutation. For node u_0 ,*
 228 *we have $\#Z_{u_0} = \#\{1, 2, 3, 4, 5\} = 5 = \theta(Z_{u_0})$ so that we have node optimality for u_0 . However, this*
 229 *is not the case for nodes u_1 or u_2 for which the numbers of padded zero columns are $\theta(Z_{u_1}) - \#Z_{u_1} =$*
 230 *2 and $\theta(Z_{u_2}) - \#Z_{u_2} = 1$, respectively.*

231 Our aim is thus to find a permutation σ that reduces the difference $\theta(\sigma(Z_u)) - \#Z_u$, corre-
 232 sponding to the gray parts in Figure 5(a). We first present a permutation based on a postordering
 233 of $T_p(B)$, then expose our new heuristic, which targets node optimality in priority at the nodes
 234 near the top of the tree.

235 **3.1. The Postorder permutation.** In Figure 1, the sequence $[u_{111}, u_{112}, u_{11}, u_{121}, u_{122},$
 236 $u_{12}, u_1, u_{211}, u_{212}, u_{21}, u_{221}, u_{222}, u_{22}, u_2, u_0]$ used to order the matrix follows a postordering: any
 237 subtree contains a set of consecutive nodes in that sequence. This postordering is also the basis to
 238 permute the columns of B :

239 **DEFINITION 6.** *Consider a postordering of the tree nodes $u \in T$, and a RHS matrix $B =$*
 240 $[B_j]_{j=1\dots m}$ *where each column B_j is represented by one of its associated nodes $u(B_j) \in V_{B_j}$ (see*
 241 *below). B is said to be postordered if and only if: $\forall 1 \leq j_1 < j_2 \leq m$, we have either $u(B_{j_1}) = u(B_{j_2})$*
 242 *or $u(B_{j_1})$ appears before $u(B_{j_2})$ in the postordering. In other words, the order of the columns B_j*
 243 *is compatible with the order of the associated representative nodes $u(B_j)$.*

244 The postordering has been applied [3, 17, 18] to group together in regular chunks RHS columns with
 245 “nearby” pruned trees, thereby limiting the accesses to the factors or the amount of computation.
 246 It was also experimented together with node intervals [4] to RHS with a single nonzero per column,
 247 although it was then combined with an interleaving mechanism for parallel issues.

248 In Figure 6(a), each RHS only has one initial nonzero and the representative nodes for each
 249 column are u_{212} , u_{12} , u_{221} , u_{211} , and u_{112} , respectively. The initial natural ordering of the columns
 250 (INI) induces computation on explicit zeros represented by gray empty cells (see also Example 3
 251 and Figure 5 where $\Delta(B) = \Delta(B, \sigma_{\text{INI}}) = 948$ and $\Delta_{\text{min}}(B) = 744$). On the other hand, the
 252 postorder permutation, σ_{PO} , reorders the columns of B so that the order of their representative

253 nodes u_{112} , u_{12} , u_{211} , u_{212} , u_{221} is compatible with the postordering and avoids computations on
 254 explicit zeros. In this case, there are no gray empty cells (Figure 6(a)) and $\Delta(B, \sigma_{P0}) = \Delta_{min}(B)$.
 255 More generally, it can be shown that the postordering induces no extra computations for RHS with
 256 a single nonzero per column [4].

257 For applications with multiple nonzeros per RHS, each column B_j may correspond to a set V_{B_j}
 258 with more than one node, among which a representative node should be chosen. We describe here
 259 two strategies. The first strategy called PO_1 chooses as representative node the one corresponding
 260 to the first nonzero found in B_j (in the natural order associated to the physical problem). The
 261 second one, called PO_2, chooses as representative node in V_{B_j} the one that appears first in the
 262 sequence of postordered nodes of the tree. A comparison of the two postorders with the initial
 263 natural order is provided in Table 1, for four problems presented in Table 2 of Section 5. Note
 264 that the natural order depends on the physical context of the application and has some geometrical
 properties. Table 1 shows that the choice of the representative node has a significant impact on the

TABLE 1
 Comparison of the number of operations ($\times 10^{13}$) between postorder strategies PO_1 and PO_2. The four test cases are issued from Table 2.

Δ	INI	PO_1	PO_2	Δ_{min}
H0	.086	.076	.070	.050
H3	2.48	1.69	1.47	.95
5Hz	.44	.44	.36	.22
7Hz	1.46	1.48	1.21	.69

265 number of operations. The superiority of PO_2 over PO_1 is clear and is larger when the number of
 266 nonzeros per RHS column is large (problems 5Hz and 7Hz). Indeed, PO_1 is even worse than the
 267 initial order on problem 7Hz.
 268

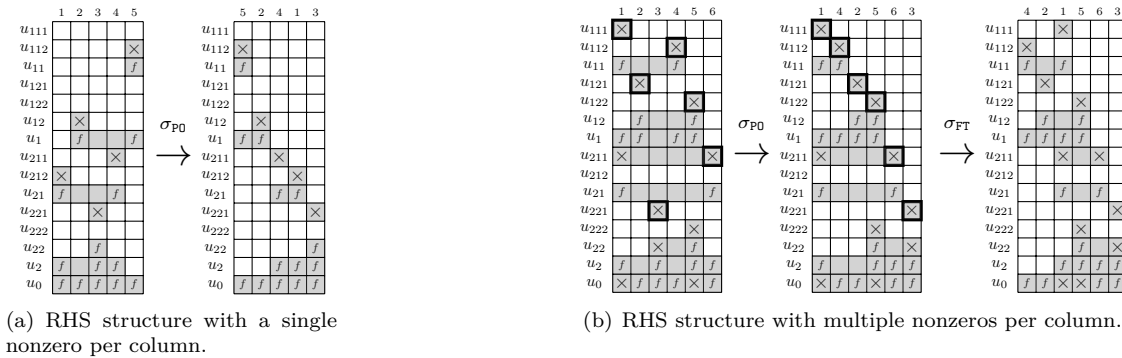


FIGURE 6. Illustration of the permutation σ_{P0} based on a postordering strategy on two RHS with (a) a single initial nonzero per column (Example 2), and (b) multiple nonzeros per column (Example 7).

269 EXAMPLE 7. Let $B = [B_1, B_2, B_3, B_4, B_5, B_6] = [\{B_{1,1}, B_{10,1}, B_{19,1}\}, \{B_{4,2}\}, \{B_{13,3}, B_{15,3}\},$
 270 $\{B_{2,3}\}, \{B_{5,4}, B_{14,4}, B_{22,4}\}, \{B_{10,5}\}]$ be the RHS represented in Figure 6(b). In terms of tree nodes,
 271 we have: $V_{B_1} = \{u_{111}, u_{211}, u_0\}$, $V_{B_2} = \{u_{121}\}$, etc. Because the rows of B have already been
 272 permuted according to the postordering of the tree, the representative nodes for strategies PO_1

273 and $P0_2$ are in both cases the nodes u_{111} , u_{121} , u_{221} , u_{112} , u_{122} , u_{211} (cells with a bold con-
 274 tour), for columns $B_1, B_2, B_3, B_4, B_5, B_6$ respectively. The postorder permutation yields $\sigma_{P0}(B) =$
 275 $[B_1, B_4, B_2, B_5, B_6, B_3]$, which reduces the number of gray cells and the volume of computation with
 276 respect to the original column ordering: $\Delta(B) = 1368$ becomes $\Delta(B, \sigma_{P0}) = 1242$. Computations on
 277 padded zeros still occur, for example at nodes u_{211} and u_{21} where $\theta(\sigma_{P0}(Z_{u_{211}})) = \theta(\sigma_{P0}(Z_{u_{21}})) = 5$
 278 whereas $\#Z_{u_{211}} = \#Z_{u_{21}} = 2$. In Figure 6(b), we represented another permutation σ_{FT} that will be
 279 discussed in the next section and that leads to $\Delta(B, \sigma_{FT}) = 1140$.

280 Remark that the quality of σ_{P0} depends on the original postordering. In Example 7, if u_{111} and
 281 u_{112} were exchanged in the original tree postordering, B_1 and B_4 would be swapped, and Δ would
 282 be further improved. A possible drawback of the postorder permutation is also that, since the
 283 position of a column is based a single representative node, some information on the RHS structure
 284 is unused. We now present a more general and powerful heuristic.

285 **3.2. The Flat Tree permutation.** With the aim of satisfying node optimality (see Def-
 286 inition 4), we present another algorithm to compute the permutation σ by first illustrating its
 287 geometrical properties and then extending it to only rely on algebraic properties.

288 **3.2.1. Geometrical illustration.** We first introduce some notations. In the example of
 289 Section 1, the variables of a separator u are the ones of the corresponding node u in the tree T . We
 290 use the same approach to represent a domain: for $u \in T$, the domain associated with u is defined
 291 by the subtree rooted at u and is noted $T[u]$. The set of variables in $T[u]$ then corresponds to a
 292 subdomain created during the nested dissection algorithm. As an example, the initial 2D domain
 293 in Figure 7(a) (left) is $T[u_0]$ and its subdomains created by dividing it with u_0 are $T[u_1]$ and $T[u_2]$.
 294 In the following, $T[u]$ will equally refer to a subdomain or a subtree. Figure 7 shows several types
 295 of RHS with different positions and nonzero structures. For the sake of simplicity, we assume here
 296 that the nonzeros in a RHS column correspond to a set of geometrically contiguous nodes in the
 297 domain, as represented in the 2D domain of Figure 7(a)(left). We also assume a regular domain for
 298 which a perfect nested dissection has been performed. For instance, all separators are in the same
 299 direction at each level of the tree.

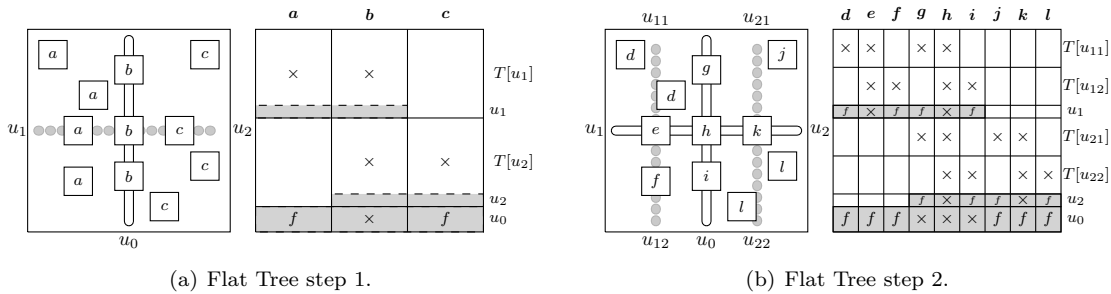


FIGURE 7. A first illustration of the “flat tree” permutation on a 2D domain. In (a) and (b), the figure on the left represents a partitioned 2D domain with different types of RHS, and the one on the right the partial structures of the permuted matrix of RHS. \times or f in a rectangle indicate the presence of nonzeros in the corresponding submatrix, parts of the matrix filled in grey are fully dense and blank parts only contain zeros.

300 *Sketch of the algorithm.* The Flat Tree algorithm relies on the evaluation of the position of
 301 each RHS column compared to separators of the nested dissection algorithm. The name Flat Tree

comes from the fact that, given a parent node with two child subtrees in the separator tree T , the algorithm orders first RHS columns included in the left subtree, then RHS columns associated to the parent (because they intersect both subtrees), and finally, RHS columns included in the right subtree. Figure 7(a) shows the first step of the algorithm: it starts with the root separator u_0 which divides $T = T[u_0]$ into $T[u_1]$ and $T[u_2]$. The initial RHS columns may be *identified* by three different types noted a , b and c according to their positions and nonzero structures. An RHS column is of type a when its nonzero structure is included in $T[u_1]$, c when it is included in $T[u_2]$ and b when it is *divided* by u_0 . First, we group the RHS according to their type (a , b , or c) with respect to u_0 which leads to the creation of submatrices/subsets of RHS columns noted \mathbf{a} , \mathbf{b} and \mathbf{c} . Second, we make sure to place \mathbf{b} between \mathbf{a} and \mathbf{c} . We thus achieve operation reduction by guaranteeing node optimality at u_1 and u_2 : since all RHS in \mathbf{a} and \mathbf{b} have at least one nonzero in $T[u_1]$, u_1 belongs to the pruned tree of all of them, hence the dense area filled in gray in the RHS structure. The same is true for \mathbf{b} and \mathbf{c} and u_2 . In Figure 7(a)(right), B is permuted as $[\mathbf{a}, \mathbf{b}, \mathbf{c}]$ ($[\mathbf{c}, \mathbf{b}, \mathbf{a}]$ would also be possible), so that \mathbf{a} and \mathbf{b} , and \mathbf{b} and \mathbf{c} , are contiguous. Thus, $\theta(Z_{u_1}) = \#Z_{u_1}$, $\theta(Z_{u_2}) = \#Z_{u_2}$ and we have u_1 - and u_2 -optimality. The algorithm then proceeds recursively on each newly created submatrix (see Figure 7(b)), meaning that only *local* node optimality can be obtained. First, $\mathbf{d}, \mathbf{e}, \mathbf{f}$ (resp. $\mathbf{j}, \mathbf{k}, \mathbf{l}$) form the subset of RHS of submatrix \mathbf{a} (resp. \mathbf{c}) based on their position/type with respect to u_1 (resp. u_2). Second, thanks to the fact that u_1 and u_2 are perfectly aligned, they can be combined to form a single separator that subdivides the RHS of \mathbf{b} into three subsets \mathbf{g}, \mathbf{h} and \mathbf{i} , see Figure 7(b). During this second step, we have the permuted B matrix $[\mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{i}, \mathbf{j}, \mathbf{k}, \mathbf{l}]$. The complete permutation is then obtained by applying the algorithm recursively on each subset until the tree is fully processed or the RHS sets contain a single RHS.

This draws the outline of the algorithm introduced with geometrical considerations. The permutation is determined only based on the position of each right-hand side with respect to the current separator(s). However, the algorithm relies on strong assumptions regarding the ordering algorithm and the RHS structure. Without them, it is difficult or impossible to discriminate RHS columns in some cases, for example when they are separated by several separators. In order to overcome those limitations and enlarge the application field, we now extend these geometrical considerations with a more general approach.

3.2.2. Algebraic approach. Let us consider the columns of matrix $B = [B_1, B_2, \dots, B_m]$ as an initially unordered *set* of RHS columns that we note $R_B = \{B_1, B_2, \dots, B_m\}$. A subset of the columns of B is then denoted by $R \subset R_B$ and a generic element of R (one of the columns $B_j \in R$) is noted r . A permuted matrix or submatrix of B can be expressed as an ordered *sequence* of RHS columns with square brackets. For example, for two subsets of columns R and R' , $[R, R']$ denotes a sequence of RHS columns, in which the RHS from the subset R are ordered before the RHS from R' , without the order of the RHS inside R or R' to be necessarily defined. We found this framework of RHS sets and subsets simpler and better adapted to formalize our algebraic algorithm than matrix notations with complex index notations. We recall that T is the tree and that, for $r \in R_B$, $T_p(r)$ is the pruned tree of r , as defined in Section 2.1. We now characterize the geometrical position of one RHS using the notion of *pruned layer*: for a given depth d in the tree, and for a given RHS r , we define the *pruned layer* $L_d(r)$ as the set of nodes at depth d in the pruned tree $T_p(r)$. In the example of Figure 7(a), $L_1(r) = \{u_1\}$ for all $r \in \mathbf{a}$, $L_1(r) = \{u_2\}$ for all $r \in \mathbf{c}$, and $L_1(r) = \{u_1, u_2\}$ for all $r \in \mathbf{b}$. The notion of pruned layers allows to formally identify sets of right-hand sides with common characteristics in the tree without any geometrical information. This is formalized and generalized by Definition 8.

347 DEFINITION 8. Let $R \subset R_B$ be a set of RHS, and let U be a set of nodes at depth d of the tree
 348 T . We defined $R[U] = \{r \in R \mid L_d(r) = U\}$ as the subset of RHS with pruned layer U .

349 We have for example, see Figure 7: $R[\{u_1\}] = \mathbf{a}$, $R[\{u_2\}] = \mathbf{c}$ and $R[\{u_1, u_2\}] = \mathbf{b}$ at depth $d = 1$.

350 The algebraic algorithm is a recursive algorithm depicted by Algorithm 1. Its arguments are
 351 R , a set of RHS and d , the current depth. Initially, $R = R_B = R[u_0]$ where u_0 is the root of
 352 the tree T , and $d = 0$. At each step of the recursion, the algorithm builds the distinct pruned
 353 layers $U_i = L_{d+1}(r)$ for the RHS r in R . Then, instead of finding a permutation σ minimizing
 354 $\sum_{u \in T_p(R_B)} \mathcal{F}_u \times \theta(\sigma(Z_u))$ (10) it orders the $R[U_i]$ by considering the *restriction* of the problem (10)
 355 to R and to nodes at depth $d+1$ of $T_p(R)$. Furthermore, with the assumption that T is balanced, all
 356 nodes at a given level of $T_p(R)$ are of comparable size. \mathcal{F}_u may thus be assumed *constant* per level
 357 and needs not be taken into account in our minimization problem. The algorithm is thus a greedy
 358 top-down algorithm, where at each step a local optimization problem is solved. This way, priority
 359 is given to the top layers of the tree, which are in general more critical because factor matrices are
 360 larger.

Algorithm 1 Flat Tree

```

procedure FLATTREE( $R, d$ )
  1) Build the set of children  $C(R)$ 
  1.1) Identify the distinct pruned layers (pruned layer = set of nodes)
   $\mathcal{U} \leftarrow \emptyset$ 
  for all  $r \in R$  do
     $\mathcal{U} \leftarrow \mathcal{U} \cup \{L_{d+1}(r)\}$ 
  end for
  1.2)  $C(R) = \{R[U] \mid U \in \mathcal{U}\}$ 
  2) Order children  $C(R)$  as  $[R[U_1], \dots, R[U_{\#C(R)}]]$ :
  return [FLATTREE( $R[U_1], d + 1$ ), ..., FLATTREE( $R[U_{\#C(R)}], d + 1$ )]
end procedure

```

361 The recursive structure of the algorithm can be represented by a recursion tree that we note
 362 T_{rec} . It is defined as follows: each node R of T_{rec} represents a set of RHS, $C(R)$ denotes the set
 363 of children of R and the root is R_B . By construction of Algorithm 1, $C(R)$ is a partition of R ,
 364 *i.e.*, $R = \dot{\bigcup}_{R' \in C(R)} R'$ (disjoint union). Please note that all $r \in R$ such that $L_{d+1}(r) = \emptyset$ belong to
 365 $R[\emptyset]$, which is also included in $C(R)$. In this particular case, $R[\emptyset]$ can be added at either extremity
 366 of the current sequence without introducing extra computation and the recursion stops for those
 367 right-hand sides, as will be illustrated in Example 9.

368 With such a construction, each leaf of T_{rec} contains RHS with indistinguishable nonzero struc-
 369 ture, and keeping them contiguous in the final permutation avoids introducing extra computations.
 370 Assuming that for each $R \in T_{rec}$ the children $C(R)$ are ordered, this induces an ordering of all the
 371 leaves of the tree, which defines the final RHS sequence. It now remains to define how the set of
 372 children $C(R)$ is built at each step and how children are ordered:

373 1) *Building the set of children..* The set of children of R in the recursion tree is built by first
 374 identifying the pruned layers U of all RHS $r \in R$. The different pruned layers are stored in \mathcal{U}
 375 and we have for example (Figure 7, first step of the algorithm), $\mathcal{U} = \{\{u_1\}, \{u_2\}, \{u_1, u_2\}\}$. Using
 376 Definition 8, we then define $C(R) = \{R[U] \mid U \in \mathcal{U}\}$, which forms a partition of R . One important
 377 property is that all $r \in R[U]$ have the same nonzero structure at the corresponding layer so that

378 numbering them contiguously prevent the introduction of extra computation.

379 2) *Ordering the children.* At each depth d of the recursion, the ordering of the children
 380 results from the resolution of the local optimization problem consisting in finding a sequence
 381 $[R[U_1], \dots, R[U_{\#C(R)}]]$ such that the size of the intervals is minimized for all nodes u at depth
 382 $d+1$ of the pruned tree $T_p(R)$. As mentioned earlier, when solving this local optimization problem,
 383 the RHS order inside each $R[U_i]$ has no impact on the size of the intervals (it will only impact lower
 384 levels). For any node u in $T_p(R)$ such that $depth(u) = d+1$, the size of the interval is then:

$$385 \quad \theta(Z_u|_R) = \max(Z_u|_R) - \min(Z_u|_R) + 1 = \sum_{i=i_{\min}(u)}^{i_{\max}(u)} \#R[U_i]$$

386 where $Z_u|_R$ is the set of permuted indices representing the active columns restricted to R , and
 387 $i_{\min}(u)$ (resp. $i_{\max}(u)$) is the first (resp. last) index i such that $u \in U_i$: $i_{\min}(u) = \min\{i \in$
 388 $\{1, \dots, \#C(R)\} \mid u \in U_i\}$ (resp. $i_{\max}(u) = \max\{i \in \{1, \dots, \#C(R)\} \mid u \in U_i\}$),

389 *Proof.* In the sequence $[R[U_1], \dots, R[U_{\#C(R)}]]$, $\min(Z_u|_R)$ (resp. $\max(Z_u|_R)$) corresponds to
 390 the index of the first (resp. last) column in $R[U_{i_{\min}}]$ (resp. $R[U_{i_{\max}}]$). Since all columns from
 391 $R[U_{i_{\min}}]$ to $R[U_{i_{\max}}]$ are numbered consecutively, we have the desired result. \square

392 Therefore, the resolution of (10) is approximated by the minimization of the following cost function
 393 (sum of the interval sizes for each node at depth $d+1$):

$$394 \quad (11) \quad \text{cost}([R[U_1], \dots, R[U_{\#C(R)}]]) = \sum_{\substack{u \in T_p(R) \\ depth(u)=d+1}} \sum_{i=i_{\min}(u)}^{i_{\max}(u)} \#R[U_i]$$

395 To build the ordered sequence $[R[U_1], \dots, R[U_{\#C(R)}]]$, we use a greedy algorithm that starts
 396 with an empty sequence, then, at each step $k \in \{1, \dots, \#C(R)\}$, we insert a RHS set $R[U]$ picked
 397 randomly in $C(R)$ at the position that minimizes (11) on the current sequence. To do so, we simply
 398 start from one extremity of the sequence of size $k-1$ and compute (11) for the new sequence
 399 of size k for each possible position $0 \dots k$; if several positions lead to the same minimal cost, the
 400 first one encountered is chosen. In case u -optimality is obtained for each node u considered, then
 401 the permutation is said to be *perfect* and the cost function is minimal, locally inducing no extra
 402 operations on those nodes.

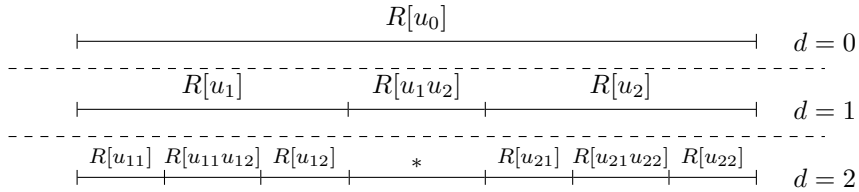


FIGURE 8. Representation of a layered sequence built by the Flat Tree algorithm on a binary tree. Sets with empty pruned layers have not been represented but could be added at the extremity of the concerned sequence (e.g., right after $R[u_2]$ for a RHS included in u_0). With the geometric assumptions corresponding to Figure 7, one would have $*$ = $R[u_{11}u_{21}]$, $R[u_{11}u_{12}u_{21}u_{22}]$, $R[u_{12}u_{22}]$. Without such assumptions, the sequence $*$ is more complex.

403 Figure 8 shows the recursive structure of the RHS sequence after applying the algorithm on a
 404 binary tree. We refer to this representation as the *layered sequence*. For simplicity, the notation for

405 pruned layers has been reduced from, e.g., $\{u_1\}$ to u_1 , and from $\{u_1, u_2\}$ to u_1u_2 . From the recursion
 406 tree point of view, $R[u_1], R[u_1u_2], R[u_2]$ are the children of $R[u_0]$ in T_{rec} , $R[u_{11}], R[u_{11}u_{12}], R[u_{12}]$
 407 the ones of $R[u_1]$, etc.

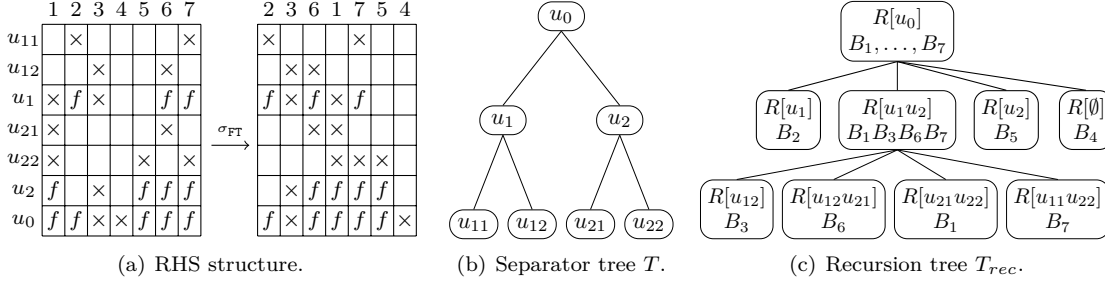


FIGURE 9. Small example with (a) a RHS structure, (b) the corresponding tree and (c) the recursion tree built by the application of the Flat Tree algorithm on a set of 7 right-hand sides.

408 EXAMPLE 9. Although we still use a binary tree in this example, we make no assumption on
 409 the RHS structure, on the domain, or on the ordering. Let $B = [B_1, B_2, B_3, B_4, B_5, B_6, B_7]$ be
 410 a RHS matrix with the nonzero structure presented in Figure 9(a). Initially, every RHS belongs
 411 to the domain $T[u_0]$ so that $R[u_0] = \{B_1, B_2, B_3, B_4, B_5, B_6, B_7\}$. At depth 1, the set of pruned
 412 layers corresponding to $R[u_0]$ is $\mathcal{U} = \{u_1, u_1u_2, u_2, \emptyset\}$ and the RHS partition is composed of the
 413 sets $R[u_1], R[u_2], R[u_1u_2]$ and $R[\emptyset]$. Then, $C(R[u_0]) = \{R[u_1], R[u_1u_2], R[u_2]\}$ and the recursion
 414 tree shown in Figure 9(c) is built from top to bottom. As can be seen in the non-permuted RHS
 415 structure, $R[\emptyset] = B_4$ at depth 1 induces extra operations at nodes descendant of u_0 , which disappear
 416 when placing $R[\emptyset]$ at one extremity of the sequence. We choose to place it last, and the ordered
 417 sequence obtained by the algorithm is $[R[u_1], R[u_1u_2], R[u_2], R[\emptyset]]$ ($[R[u_2], R[u_1u_2], R[u_1], R[\emptyset]]$ is
 418 also possible). A recursive call is done on each of the identified sets. We only focus on $R = R[u_1u_2]$
 419 since $R[u_1]$, $R[u_2]$ and $R[\emptyset]$ contain a single RHS which needs not be further ordered. At this
 420 stage, the set of pruned layers is $\mathcal{U} = \{u_{21}u_{22}, u_{12}, u_{12}u_{21}, u_{11}u_{22}\}$. It appears that the sequence
 421 $[R[U_1], R[U_2], R[U_3], R[U_4]]$, where $U_1 = u_{12}$, $U_2 = u_{12}u_{21}$, $U_3 = u_{21}u_{22}$, and $U_4 = u_{11}u_{22}$ is a
 422 perfect sequence which gives local optimality. However, taking the problem globally, we see that
 423 $\theta(Z_{u_{11}}) \neq \#Z_{u_{11}}$. The final sequence obtained is $[B_2, B_3, B_6, B_1, B_7, B_5, B_4]$.

424 The algebraic algorithm simplifies the assumptions that were made in the geometrical approach.
 425 Indeed, the nonzero structure of each RHS does not need to be geometrically localized anymore,
 426 and we can now address irregular problems and orderings that yield non-binary trees. We compared
 427 both approaches on the test cases H0, H3, 5Hz and 7Hz, described in Table 2, and observed an
 428 average gain of 7% on the value of Δ with the algebraic approach with respect to the geometrical
 429 one. The algebraic approach thus takes better into account the irregularity of real problems and will
 430 be used in all our experiments. Nevertheless, computations on explicit zeros (for example zero rows
 431 in column f and subdomain $T[u_{11}]$ in Figure 7(b)), may still occur. This will also be illustrated
 432 in Section 5, where $\Delta(B, \sigma_{FT})$ is 32% larger than $\Delta_{min}(B)$ in the worst case. In Section 4, the
 433 Blocking algorithm is introduced to further reduce $\Delta(B, \sigma_{FT})$.

434 **4. Toward an optimal solution using group creation.** In this section, we identify the
 435 causes of the remaining extra operations and provide an efficient blocking algorithm to reduce

436 them efficiently while creating a small number of groups. The algorithm relies on a property of
 437 independence of right-hand sides that is first illustrated, and then formalized.

438 **4.1. Objectives and first illustration of independence property.** The use of blocking
 439 techniques may fulfill different objectives. In terms of operation count, optimality ($\Delta_{min}(B)$) is
 440 obtained when processing the columns of B one by one, which implies the creation of m groups.
 441 However, this requires processing the tree m times and will typically lead to a poor arithmetic
 442 intensity (and likely a poor performance). On the other hand, the algorithms of Section 3 only use
 443 one block, which enables a higher arithmetic intensity but leads to extra operations. In the dense
 444 case, blocking techniques are also used to improve the arithmetic intensity by working on blocks
 445 of vectors instead of individual columns, for example. In the sparse RHS case, blocking strategies
 446 with regular blocks of columns have been associated to tree pruning to either limit the access to
 447 the factors [3], or limit the number of operations [18]. They were either based on a reordering
 448 of the columns or on hypergraph models. In this section, to give as much flexibility as possible to
 449 the underlying algorithms and avoid unnecessary constraints, our objective is to create a minimal
 450 number of (possibly large) groups while reducing the number of extra operations by a given amount.
 451 In particular, we allow groups to be irregular and assume node intervals are exploited within each
 452 block.

453 On the one hand, in the same way as variables in two different domains are independent, two
 454 RHS or two sets of RHS included in two different domains exhibit interesting properties, as can be
 455 observed for sets $\mathbf{a} \in T[u_1]$ and $\mathbf{c} \in T[u_2]$ from Figure 7(a). It implies that no extra operations
 456 are introduced between them: $\Delta([\mathbf{a}, \mathbf{c}]) = \Delta(\mathbf{a}) + \Delta(\mathbf{c})$. We say that \mathbf{a} and \mathbf{c} are *independent sets*
 457 and can thus be associated together. On the other hand, a set of RHS intersecting a separator
 458 (such as set \mathbf{b}) exhibits some zeros and nonzeros in rows common to their adjacent RHS sets (\mathbf{a}
 459 and \mathbf{c}) which will likely introduce extra computation. For example, one can see in Figure 7(b) that
 460 $\Delta([\mathbf{a}, \mathbf{b}]) = \Delta([\mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}, \mathbf{i}]) > \Delta([\mathbf{d}, \mathbf{e}, \mathbf{f}]) + \Delta([\mathbf{g}, \mathbf{h}, \mathbf{i}]) = \Delta(\mathbf{a}) + \Delta(\mathbf{b})$ and that $\Delta([\mathbf{a}, \mathbf{b}, \mathbf{c}]) >$
 461 $\Delta(\mathbf{a}) + \Delta(\mathbf{b}) + \Delta(\mathbf{c})$. We say that \mathbf{b} is a set of *problematic RHS*. Another example is the one of
 462 Figure 6 (right), where extracting the problematic RHS B_1 and B_5 from $[B_4, B_2, B_1, B_5, B_6, B_3]$
 463 suppresses all extra operations: $\Delta([B_4, B_2, B_6, B_3]) + \Delta([B_1, B_5]) = \Delta_{min} = 1056$.

464 To give further intuition on the Blocking algorithm, consider the RHS structure of Figure 7(b).
 465 Problematic RHS \mathbf{e} and \mathbf{k} in $[\mathbf{d}, \mathbf{e}, \mathbf{f}, \mathbf{j}, \mathbf{k}, \mathbf{l}]$ can be extracted to form two groups $[\mathbf{e}, \mathbf{k}]$ and $[\mathbf{d}, \mathbf{f}, \mathbf{j}, \mathbf{l}]$.
 466 The situation is slightly more complicated for $[\mathbf{g}, \mathbf{h}, \mathbf{i}]$, where \mathbf{h} indeed intersects two separators,
 467 u_1 and u_2 . In this case, \mathbf{h} should be extracted to form the groups $[\mathbf{g}, \mathbf{i}]$ and $[\mathbf{h}]$. We note that the
 468 amount of extra operations will likely be much larger when the separator intersected is high in the
 469 tree.

470 Situations where no assumption on the RHS structure is made are more complicated and require
 471 a more general approach. For this, we formalize the notion of *independence*, which will be the basis
 472 for our blocking algorithm.

473 **4.2. Algebraic formalization and first blocking algorithm.** In this section, we give a
 474 first version of the Blocking algorithm and show that it provides a sufficient condition to group
 475 together sets of RHS without introducing extra computation. Furthermore, we assume the matrix
 476 B to be flat tree ordered and the recursion tree T_{rec} to be built and ordered. Using the notations
 477 of Definition 8, we first give an algebraic definition of the independence property between two sets
 478 of RHS:

479 **DEFINITION 10.** *Let U_1, U_2 be two sets of nodes at a given depth of a tree T , and let $R[U_1], R[U_2]$*
 480 *be the corresponding sets of RHS. $R[U_1], R[U_2]$ are said to be independent if and only if $U_1 \cap U_2 = \emptyset$.*

481 With Definition 10, we are now able to formally identify independent sets and we will show formally
 482 why they can be associated together. For example, take $\mathbf{a} = R[u_1]$ and $\mathbf{c} = R[u_2]$ from Figure 7(a),
 483 $R[u_1]$ and $R[u_2]$ are independent and $\Delta([R[u_1], R[u_2]]) = \Delta(R[u_1]) + \Delta(R[u_2])$. On the contrary,
 484 when $R[U_1], \dots, R[U_n]$ are not pairwise independent, the objective is to group together independent
 485 sets of RHS, while forming as few groups as possible. In terms of graphs, this problem is equivalent
 486 to a classical coloring problem, where $R[U_1], \dots, R[U_n]$ are the vertices and an edge exists between
 487 $R[U_i]$ and $R[U_j]$ if and only if $U_i \cap U_j \neq \emptyset$. Several heuristics exist for this problem, and each
 color will correspond to one group. The Blocking algorithm as depicted in Figure 10 consists of a

Algorithm 2 Blocking algorithm

```

for  $d = 0$  to  $d_{max}$  do
   $j \leftarrow 0$  /* #groups at depth  $d + 1$  */
  for all groups  $g_i^d$  at depth  $d$  do
     $(g_{j+1}^{d+1} \dots g_{j+k}^{d+1}) \leftarrow \text{BUILDGROUPS}(g_i^d, d+1)$ 
    /*  $k$  new groups have been created */
     $j \leftarrow j + k$ 
  end for
end for

```

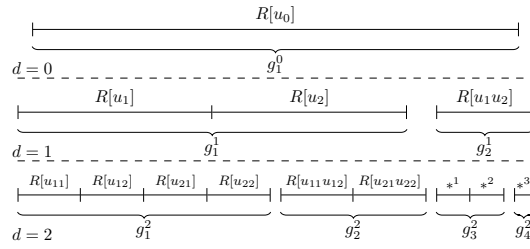


FIGURE 10. A first version of the Blocking algorithm (left). It is illustrated (right) on the layered sequence of Figure 8. With the geometric assumptions of Figure 7, $*^1 = R[u_{11}u_{21}]$, $*^2 = R[u_{12}u_{22}]$, and $*^3 = R[u_{11}u_{12}u_{21}u_{22}]$.

488 top-down traversal of T_{rec} where at each depth d and for each intermediate group g_i^d , the central
 489 procedure BUILDGROUPS is called. Notice that any group g_i^d verifies the following properties: (i)
 490 g_i^d can be represented by a sequence $[R[U_1], \dots, R[U_n]]$; and (ii) the sequence respects the flat tree
 491 order of T_{rec} . Then, BUILDGROUPS($g_i^d, d+1$) first builds the sets of RHS at depth $d+1$, which are
 492 exactly the children of the $R[U_j] \in g_i^d$ in T_{rec} . Second, BUILDGROUPS($g_i^d, d+1$) solves the coloring
 493 problem mentioned earlier on these RHS sets and builds the k groups $(g_{j+1}^{d+1}, \dots, g_{j+k}^{d+1})$.

494 In the example of Figure 10(right), there is initially a single group $g_1^0 = [R[u_0]]$ with one
 495 set of RHS. This group may be expressed as the ordered sequence $[R[u_1]R[u_1u_2]R[u_2]]$, since
 496 $C(R[u_0]) = \{R[u_1], R[u_1u_2], R[u_2]\}$. g_1^0 does not satisfy the independence property at depth 1
 497 because $u_1 \cap u_1u_2 \neq \emptyset$ or $u_2 \cap u_1u_2 \neq \emptyset$. BUILDGROUPS($g_1^0, 1$) yields $g_1^1 = [R[u_1], R[u_2]]$ and
 498 $g_2^1 = [R[u_1u_2]]$. The algorithm proceeds on each group until a maximal depth d_{max} is reached:
 499 $(g_1^2, g_2^2) = \text{BUILDGROUPS}(g_1^1, 2)$, $(g_3^2, g_4^2) = \text{BUILDGROUPS}(g_2^1, 2)$, etc. To illustrate the interest of
 500 property (ii) on groups, let us take sets $\mathbf{d} = R[u_{11}]$, $\mathbf{f} = R[u_{12}]$, $\mathbf{j} = R[u_{21}]$ and $\mathbf{l} = R[u_{22}]$ from
 501 Figure 7(b). One can see that $\Delta([\mathbf{d}, \mathbf{f}, \mathbf{j}, \mathbf{l}]) = \Delta(\mathbf{d}) + \Delta(\mathbf{f}) + \Delta(\mathbf{j}) + \Delta(\mathbf{l}) < \Delta([\mathbf{d}, \mathbf{j}, \mathbf{f}, \mathbf{l}])$. Com-
 502 pared to $[\mathbf{d}, \mathbf{f}, \mathbf{j}, \mathbf{l}]$ which respects the global flat tree ordering and ensures u_1 - and u_2 -optimality,
 503 $[\mathbf{d}, \mathbf{j}, \mathbf{f}, \mathbf{l}]$ does not and thus increases $\theta(Z_{u_1})$ and $\theta(Z_{u_2})$.

504 Furthermore, Algorithm 2 ensures the following property, which shows that the independent
 505 sets of RHS grouped together do not introduce extra operations.

506 **PROPERTY 11.** For any group $g^d = [R[U_1], \dots, R[U_n]]$ created through Algorithm 2 at depth d ,
 507 we have $\Delta([R[U_1], \dots, R[U_n]]) = \sum_{i=1}^n \Delta(R[U_i])$.

508 *Proof.* For $d \geq 1$, let $g^d = [R[U_i^d]_{i=1, \dots, n^d}]$ be a group at depth d created through Algorithm 2
 509 (we use superscripts d in this proof to indicate the depth without ambiguity). Let us split nodes
 510 above (A) and below (B) layer d in the pruned tree $T_p(g^d)$. The number of operations to process
 511

512 g^d is:

$$513 \quad (12) \quad \Delta(g^d) = \sum_{u \in T_p(g^d)} \mathcal{F}_u \times \theta(Z_u|_{g^d}) = \overbrace{\sum_{u \in A} \mathcal{F}_u \times \theta(Z_u|_{g^d})}^{\Delta_A} + \overbrace{\sum_{u \in B} \mathcal{F}_u \times \theta(Z_u|_{g^d})}^{\Delta_B},$$

514 where $A = \{u \in T_p(g^d) \mid \text{depth}(u) < d\}$ and $B = \{u \in T_p(g^d) \mid \text{depth}(u) \geq d\}$.

515 (i) We first consider the term Δ_B . Let $B_i = \{u \in T_p(R[U_i^d]) \mid \text{depth}(u) \geq d\}$. Thanks to the
516 independence property of the $R[U_i^d]$ forming g^d , the pruned layers U_i^d in T are disjoint and since
517 T is a tree, we have $B_i \cap B_j = \emptyset$ for all $i \neq j$. Hence, $B = \dot{\bigcup}_{i=1}^{n^d} B_i$, where $\dot{\bigcup}$ denotes the disjoint
518 union. Therefore,

$$519 \quad \Delta_B = \sum_{u \in \dot{\bigcup}_{i=1}^{n^d} B_i} \mathcal{F}_u \times \theta(Z_u|_{[R[U_j^d]_{j=1, \dots, n^d}]}) = \sum_{i=1}^{n^d} \sum_{u \in B_i} \mathcal{F}_u \times \theta(Z_u|_{[R[U_j^d]_{j=1, \dots, n^d}]}).$$

520 We recall that a RHS r is said to be active at node u if $u \in T_p(r)$. In the inner sum, the only possible
521 active RHS in B_i are the ones that belong to $R[U_i^d]$ (independence of the $R[U_j^d]$), so that for all
522 $u \in B_i$, we have $\theta(Z_u|_{[R[U_j^d]_{j=1, \dots, n^d}]}) = \theta(Z_u|_{R[U_i^d]})$. Therefore, $\Delta_B = \sum_{i=1}^{n^d} \sum_{u \in B_i} \mathcal{F}_u \times \theta(Z_u|_{R[U_i^d]})$.

523 (ii) We now consider the term Δ_A . Similarly to (i), we define $A_i = \{u \in T_p(R[U_i^d]) \mid \text{depth}(u) < d\}$.
524 We have $A = \bigcup_{i=1}^{n^d} A_i$ but the union is no longer disjoint. Let $T_{rec}(g^d)$ be the restriction to g^d of
525 the recursion tree T_{rec} associated to the flat-tree algorithm applied to R_B (see Section 3.2.2 for the
526 definition of T_{rec}). $T_{rec}(g^d)$ is obtained by excluding at each node of T_{rec} the right-hand sides that
527 are not part of g^d , then by pruning all empty nodes. We also restrict Definition 8 to g^d and thus
528 note $R[U] = \{r \in g^d \mid L_d(r) = U\}$. In particular, the root of $T_{rec}(g^d)$ is $R[u_0] = g^d$.

529 By construction of Algorithm 2 (Figure 10), we know that any layer at depth $d' < d$ of the
530 group g^d consists of independent sets $R[U_j^{d'}]$ of RHS. Therefore, $\forall u \in A, \exists! R[U] \in T_{rec}(g^d)$ such
531 that $u \in U$. This means that the only active columns at node u are those in this unique $R[U]$
532 and, since the RHS in $R[U]$ are all contiguous in g^d thanks to the global flat tree ordering, we have
533 $\theta(Z_u|_{R[U]}) = \theta(Z_u|_{g^d}) = \#R[U]$.

534 Furthermore, by construction of the recursion tree (children nodes form a partition of each
535 parent node), the RHS in $R[U]$ are the ones in the disjoint union of $R[U_i^d] \subset R[U]$, the sets
536 of right-hand sides at layer d that are descendants of $R[U]$ in $T_{rec}(g^d)$. Therefore, $\#R[U] =$
537 $\sum_{R[U_i^d] \subset R[U]} \#R[U_i^d]$. Furthermore, since the $R[U_i^d]$ such that $R[U_i^d] \subset R[U]$ are contiguous sets in
538 g^d and are all active at node u , we also have $\theta(Z_u|_{R[U_i^d]}) = \#R[U_i^d]$. It follows:

$$539 \quad \theta(Z_u|_{g^d}) = \sum_{R[U_i^d] \subset R[U]} \theta(Z_u|_{R[U_i^d]}).$$

540 We define $\delta_i(u) = 1$ if $R[U_i^d] \subset R[U]$ (with $R[U]$ derived from u as explained above), and
541 $\delta_i(u) = 0$ otherwise. The condition $R[U_i^d] \subset R[U]$ means that u is an ancestor of U_i^d nodes in T .
542 Thus, $\delta_i(u) = 1$ for $u \in A_i$ and $\delta_i(u) = 0$ for $u \notin A_i$. We can thus write $\sum_{R[U_i^d] \subset R[U]} \theta(Z_u|_{R[U_i^d]}) =$
543 $\sum_{i=1}^{n^d} \delta_i(u) \theta(Z_u|_{R[U_i^d]})$ and redefine Δ_A as:

$$544 \quad \Delta_A = \sum_{u \in A} \mathcal{F}_u \times \theta(Z_u|_{g^d}) = \sum_{u \in A} \mathcal{F}_u \times \sum_{i=1}^{n^d} \delta_i(u) \theta(Z_u|_{R[U_i^d]})$$

$$\begin{aligned}
545 \quad &= \sum_{i=1}^{n^d} \sum_{u \in A} \mathcal{F}_u \times \delta_i(u) \theta(Z_u |_{R[U_i^d]}) \\
546 \quad &= \sum_{i=1}^{n^d} \sum_{u \in A_i} \mathcal{F}_u \times \theta(Z_u |_{R[U_i^d]}). \\
547
\end{aligned}$$

548 Joining the terms Δ_A and Δ_B , we finally have:

$$549 \quad \Delta(g^d) = \Delta_A + \Delta_B = \sum_{i=1}^{n^d} \sum_{u \in B_i} \mathcal{F}_u \times \theta(Z_u |_{R[U_i]}) + \sum_{i=1}^{n^d} \sum_{u \in A_i} \mathcal{F}_u \times \theta(Z_u |_{R[U_i]}) = \sum_{i=1}^{n^d} \Delta(R[U_i]) \quad \square$$

550 Interestingly, Property 11 can be used to prove, in the case of a single nonzero per RHS, the
551 optimality of the Flat Tree permutation.

552 **COROLLARY 12.** *Let R_B be the initial set of RHS such that $\forall r \in R_B, \#V_r = 1$. Then the Flat*
553 *Tree permutation is optimal: $\Delta(R_B) = \Delta_{min}(R_B)$.*

554 *Proof.* Since $\forall r \in R_B, \#V_r = 1$, $T_p(r)$ is a branch of T . Indeed, V_r is a singleton and $T_p(r)$ is
555 built by following the path in T from V_r up to the root. As a consequence, any set of RHS $R[U]$
556 built through the Flat Tree algorithm will be represented by a pruned layer U containing a single
557 node u . Thus, at each step of the algorithm, the RHS sets identified by the Flat Tree algorithm
558 are all independent from each other. In case Algorithm 2 is applied, a unique group R_B is then
559 kept until the bottom of the tree. Blocking is thus not needed and Property 11 applies at each
560 level of the flat tree recursion. $\Delta(R_B)$ is thus equal to the sum of the $\Delta(R[U])$ for all leaves $R[U]$
561 of the recursion tree T_{rec} . Since $\Delta(R[U]) = \Delta_{min}(R[U])$ on those leaves (all RHS in $R[U]$ involve
562 the exact same nodes and operations), we conclude that $\Delta(R_B) = \Delta_{min}(R_B)$. \square

563 This proof is independent of the specific ordering of the children at step 2 of Algorithm 1. The
564 corollary is therefore more general: any recursive top-down ordering based on keeping together at
565 each layer the RHS with an identical pruned layer is optimal, as long as the pruned layers identified
566 at each layer are independent.

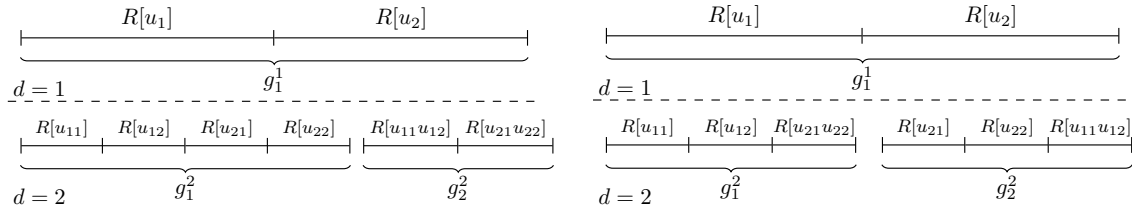


FIGURE 11. Two strategies to build groups: CRITPATHBUILDGROUPTS (left) and REGBUILDGROUPTS (right).

567 Back to the BUILDGROUPS function and the coloring problem, we want to mention that the
568 solution may not be unique. Even on the simple example of Figure 10, there are several ways to
569 define groups, as shown in Figure 11 for g_1^1 : both strategies satisfy the independence property and
570 minimize the number of groups. The first strategy, CRITPATHBUILDGROUPTS, tends to create a large
571 group g_1^2 and a smaller one (g_2^2). In each group the computations on the tree nodes are expected to be
572 well balanced because all branches of the tree rooted at u_0 might be covered by the RHS (assuming

573 thus a reasonably balanced RHS distribution over the tree). The choice of CRITPATHBUILDGROUPS
 574 can be driven by tree parallelism considerations, namely, the limitation of the sum of the operation
 575 count on the *critical paths* of all groups. The second one, REGBUILDGROUPS, tends to balance the
 576 sizes of the groups created but may create more unbalance regarding the repartition of the work
 577 over the tree.

578 We note that for a given depth, the application of BUILDGROUPS on *all* groups may lead to a
 579 too rapid and unnecessary increase of the number of groups. Furthermore, the enforcement of the
 580 independence property during the BUILDGROUPS operation may require the creation of more than
 581 two groups. In the next section we add features to minimize the number of groups created using
 582 greedy heuristics and propose our final version of the Blocking algorithm.

583 **4.3. A greedy approach to minimize the number of groups.** The final greedy blocking
 584 algorithm is given by Algorithm 3. Compared to Algorithm 2, it adds the group selection, limits the
 585 number of groups during BUILDGROUPS to two, and stops when a given tolerance on the amount
 586 of extra operations is reached.

587 First, instead of stepping into and incrementing the current depth of each group, as shown
 588 in Algorithm 2, the group selection consists in choosing among the current groups the one that
 589 introduces the most extra computation, that is, the one for which $\Delta(g) - \Delta_{min}(g)$ is maximal.
 590 This implies that groups that are split might have been created at different depths and we use a
 591 superscript to indicate the depth d at which a group was split, as in the notation g_0^d .

592 Second, enforcing the independence property inside all the groups created may be inappropriate
 593 for some problems, because the number of groups may increase much more than needed. Instead
 594 of a coloring problem, BUILDGROUPS then just looks inside the sets of g_0^d for a maximal group of
 595 independent sets of RHS at depth $d + 1$, noted g_{imax}^{d+1} , and leaves the other sets in another group g_c^d ,
 596 for which the depth remains equal to d . The latter may thus consist of dependent sets that may be
 597 subdivided later if needed². Rather than an exact algorithm to determine g_{imax} , we use a greedy
 598 heuristic that forms a *maximal* independent set.

599 Finally, we define μ_0 as the tolerance of extra operations authorized. With a typical value
 600 $\mu_0 = 1.01$, the algorithm stops when the number of extra operations is within 1% of the minimal
 601 number of operations, Δ_{min} . When the algorithm stops, G contains the final set of groups.

Algorithm 3 Blocking algorithm

```

G ← {RB}, Δmin ← Δmin(RB), Δ ← Δ(RB)
while Δ/Δmin > μ0 do
  Select g0d such that Δ(g0d) - Δmin(g0d) = maxg∈G (Δ(g) - Δmin(g))           ▷ Group selection
  (gimaxd+1, gcd) ← BUILDMAXINDEPSET(g0d, d+1)
  G ← G ∪ {gimaxd+1, gcd} \ {g0d}
  Δ ← Δ - Δ(g0d) + Δ(gimaxd+1) + Δ(gcd)
end while

```

602 **5. Experimental results.** In this section, we report on the operation count Δ resulting from
 603 the proposed permutation and blocking algorithms, measured in terms of number of operations
 604 during the forward elimination (Equation (2)). Our experiments are performed on a set of 3D

²In case g_c^d consists of independent sets and is selected, the exact same sets will be used for g_c^{d+1} , which will then only be subdivided at depth $d + 2$.

TABLE 2

Characteristics of the $n \times n$ matrix A and $n \times m$ matrix B for different test cases. $D(A) = \text{nnz}(A)/n$ and $D(B) = \text{nnz}(B)/m$ represent the average number of nonzeros per column of A and B , respectively.

application	matrix	$n(\times 10^6)$	$D(A)$	sym	m	$D(B)$
seismic modeling	5Hz	2.9	24	no	2302	567
	7Hz	7.2	25	no	2302	486
	10Hz	17.2	26	no	2302	486
electromagnetism modeling	H0	.3	13	yes	8000	9.8
	H3	2.9	13	yes	8000	7.5
	H17	17.4	13	yes	8000	6
	H116	116.2	13	yes	8000	6
	S3	3.3	13	yes	12340	19.7
	S21	20.6	13	yes	12340	9.5
	S84	84.1	13	yes	12340	8.6
	D30	29.7	23	yes	3914	7.6

TABLE 4

Theoretical tree parallelism according to the strategy used (ND ordering).

S	DEN	TP	RAN	INI	PO	FT
5Hz	8.60	3.91	3.88	3.11	2.39	2.54
7Hz	8.92	3.97	3.94	3.02	2.25	2.48
10Hz	9.10	4.04	4.02	2.96	2.30	2.30
H0	5.88	2.11	2.11	1.75	1.51	1.45
H3	5.99	3.22	3.21	2.47	2.02	2.11
H17	6.32	3.34	3.32	2.54	2.00	1.97
H116	7.92	3.63	3.61	2.75	2.05	2.02
S3	6.12	2.84	2.83	2.18	1.73	1.61
S21	6.30	2.56	2.46	1.85	1.49	1.47
S84	8.01	2.41	2.38	1.90	1.53	1.52
D30	8.50	4.73	4.70	2.86	2.05	2.56

TABLE 3

Number of operations ($\times 10^{13}$) during the forward elimination ($LY = B$) according to the strategy used (ND ordering).

Δ	DEN	TP	RAN	INI	PO	FT	Δ_{min}
5Hz	1.73	.74	.74	.44	.36	.28	.22
7Hz	5.94	2.54	2.52	1.46	1.21	.92	.69
10Hz	20.62	9.01	8.92	4.78	3.85	2.87	2.26
H0	0.39	0.11	0.11	0.086	0.070	0.057	0.050
H3	7.19	3.33	3.31	2.48	1.47	1.26	0.95
H17	81.34	37.15	36.97	27.52	10.41	10.21	10.12
H116	990.02	448.31	445.91	327.89	123.79	121.76	120.68
S3	13.36	4.98	4.91	3.73	2.65	2.17	1.71
S21	156.20	49.04	48.07	35.42	25.73	22.53	19.43
S84	983.48	286.57	282.70	222.59	161.87	138.56	118.51
D30	71.60	39.78	39.38	19.49	10.93	10.21	7.31

TABLE 5

Impact of the number of groups NG on the normalized operation count, until Δ_{NG}/Δ_{min} becomes smaller than the tolerance $\mu_0 = 1.01$ (ND ordering).

Δ_{NG}/Δ_{min}	FT	NG= 2	NG= 3	NG= 4	NG= 5
5Hz	1.283	1.111	1.001	x	x
7Hz	1.321	1.116	1.002	x	x
10Hz	1.269	1.029	1.002	x	x
H0	1.148	1.029	1.010	1.002	x
H3	1.329	1.068	1.027	1.005	x
H17	1.009	x	x	x	x
H116	1.009	x	x	x	x
S3	1.275	1.120	1.045	1.012	1.003
S21	1.160	1.037	1.015	1.003	x
S84	1.169	1.041	1.015	1.002	x
D30	1.397	1.082	1.058	1.024	1.004

605 regular finite difference problems coming from seismic and electromagnetism modeling [1, 16], for
 606 which the cost of the solve phase is critical. The characteristics of the corresponding matrices and
 607 associated RHS are presented in Table 2. In both applications, the nonzeros of each RHS correspond
 608 to a small set of close points, near the top of the 3D grid corresponding to the physical domain,
 609 and there is some overlap between RHS. Except in Section 5.3, a geometric nested dissection (ND)
 610 algorithm is used to reorder the matrix.

611 **5.1. Impact of the Flat Tree algorithm.** We first introduce the terminology used to denote
 612 the different strategies developed in this study and that impact the number of operations Δ . DEN
 613 represents the dense case, where no optimization is used to reduce Δ , and TP means tree pruning.
 614 When column intervals are exploited at each tree node, we denote by RAN, INI, PO and FT the
 615 random, initial ($\sigma = id$), Postorder (σ_{PO}) and Flat Tree (σ_{FT}) permutations, respectively.

616 The improvements brought by the different strategies are presented in Table 3. Compared to
 617 the dense case, TP divides Δ by at least a factor 2. In the case column intervals are exploited at each
 618 node, the large gap between RAN and INI shows that the original column order holds geometrical
 619 properties. FT behaves better than INI and PO and gets reasonably close to Δ_{min} . Overall, FT

620 provides a 13% gain on average over P0. However, the gain on Δ decreases from 25% on the 10Hz
 621 problem to 1% on the H116 problem. This can be explained by the fact that B is denser for the
 622 seismic applications than for the electromagnetism applications (see Table 2). Indeed, the sparser
 623 B , the closer we are from a single nonzero per RHS in which case both FT and P0 are optimal.

624 Second, we evaluate the impact of exploiting RHS sparsity on tree parallelism. For this, we
 625 report in Table 4 the maximal theoretical speed-up S that can be reached using tree parallelism
 626 only (node parallelism is also needed, for example on the root). It is defined as $S = \frac{\Delta}{\Delta_{cp}}$ where
 627 Δ_{cp} is the number of operations on the critical path of the tree. We observe that tree parallelism
 628 is significantly smaller than in the dense case. This is because the depth of the pruned tree $T_p(B)$
 629 is similar to the one of the original tree (some nonzeros of B appear in general in the leaves), while
 630 the tree effectively processed is pruned and thus the overall amount of operations is reduced. For
 631 the same reason, S is smaller for test cases where $nnz(B)/m$ is small. For the 5Hz, 7Hz, and 10Hz
 632 problems which have more nonzeros per column of B , besides decreasing the operation count more
 633 than the other strategies, FT exhibits equivalent or even better tree parallelism than P0. For such
 634 matrices, where $D(B) = nnz(B)/m$ is large, FT balances the work on the tree better than P0 and
 635 reduces the work on the critical path more than the total work. Overall, FT reduces the operation
 636 count better than any other strategy and has good parallel properties.

637 **5.2. Impact of the Blocking algorithm.** First, we show that the Blocking algorithm de-
 638 creases the operation count Δ while creating a limited number of groups. Second, we discuss and
 639 justify with parallelism arguments our clustering strategies illustrated in Figure 11.

640 In Table 5, we represent the value of $\frac{\Delta_{NG}}{\Delta_{min}}$ depending on the number of groups created. x means
 641 that the Blocking algorithm stopped earlier because the condition $\Delta_{NG}/\Delta_{min} \leq \mu_0$ was reached,
 642 where the tolerance μ_0 for Algorithm 3 is set to 1.01. Computing from Table 5 the ratio of extra
 643 operations reduction $1 - \frac{\Delta_{NG} - \Delta_{min}}{\Delta_1 - \Delta_{min}}$ for NG groups created, we observe an average reduction of
 644 74% of the extra operations when $NG = 2$, *i.e.*, when only two groups are created. Table 5 also
 645 shows that Δ_{NG} reaches very quickly a value close to Δ_{min} and thus we confirm the expectation
 646 from Section 4.1 that right-hand sides responsible for most extra operations were those intersecting
 647 a separator high in the tree.

TABLE 6
 Sum of critical paths' operations ($\times 10^{13}$) for two grouping strategies when three groups are created.

$\sum_g \Delta_{cp}(g)$	5Hz	7Hz	10Hz	H0	H3
CRITPATHBUILDGROUP	.092	.30	1.00	.037	.50
REGBUILDGROUP	.12	.43	1.58	.044	.72

648 In Table 6, we report the sum of operation counts on the critical paths Δ_{cp} over all groups cre-
 649 ated using CRITPATHBUILDGROUP and REGBUILDGROUP strategies, when the number of groups
 650 created is three, leading to Δ close to Δ_{min} , see column “NG=3” of Table 5. In this case, the
 651 total number of operations Δ during the forward solution phase on all groups is equal whether we
 652 use CRITPATHBUILDGROUP or REGBUILDGROUP. Tree parallelism is thus a crucial discriminant
 653 between both strategies, and we indeed observe in Table 6 that CRITPATHBUILDGROUP effectively
 654 limits the length of critical paths over the three groups created, justifying its use.

655 **5.3. Experiments with other orderings.** As mentioned earlier, several orderings [14, 12, 2]
 656 may be used to order the unknowns of the original matrix, thanks to the algebraic nature of our
 657 Flat Tree and Blocking algorithms. Although local ordering methods (AMD, AMF as provided by

TABLE 7

Operation count $\Delta(\times 10^{13})$ for permutation strategies *P0* and *FT*, and number of groups *NG* required to reach $\frac{\Delta_{NG}}{\Delta_{min}} \leq 1.01$ for blocking strategies *REG* and *BLK*. Different orderings (*AMD*, *AMF*, *SCOTCH*, *METIS*) are used.

orderings	AMD					AMF					SCOTCH					METIS				
	σ	PO	REG	FT	BLK	Δ_{min}	PO	REG	FT	BLK	Δ_{min}	PO	REG	FT	BLK	Δ_{min}	PO	REG	FT	BLK
5Hz	1.44	53	1.36	4	1.25	.75	51	.87	7	.68	.47	328	.32	3	.25	.43	230	.30	3	.24
7Hz	5.03	38	4.44	4	4.35	15.29	18	17.82	12	14.89	1.60	287	1.14	3	.86	1.42	230	1.08	3	.82
10Hz	19.34	38	19.79	3	15.27	96.86	18	99.07	11	82.13	5.86	287	4.21	3	3.05	4.67	230	3.44	3	2.53
H0	.54	533	.53	4	.47	.12	333	.12	5	.0910	.0728	499	.0627	3	.0548	.0774	615	.0668	3	.0569
H3	134.54	380	105.98	5	9.07	101.43	63	133.97	17	9.26	2.19	615	1.76	5	1.18	1.95	533	1.53	5	1.12
H17	183.03	266	225.66	7	135.94	467.06	173	558.31	50	395.74	22.79	380	18.97	5	12.58	21.49	242	16.80	4	12.33
H116	2244.71	1	2244.71	1	2244.71	39383.4	1	39383.6	1	39383.4	290.45	109	224.04	4	153.16	263.72	78	215.21	4	157.00
S3	20.88	725	17.85	6	15.14	20.71	184	24.83	10	17.78	4.54	771	3.41	5	2.72	3.24	771	2.64	5	2.09
S21	392.57	685	348.87	5	310.63	1141.45	493	1352.32	77	830.51	50.91	492	39.55	4	31.73	34.31	223	28.53	5	24.86
S84	3025.30	352	2847.53	5	2501.38	38664.7	725	45346.4	213	30076.9	289.14	286	228.31	4	193.36	207.39	171	174.43	4	150.93
D30	115.37	111	121.29	8	94.51	1015.16	139	1279.55	75	825.21	16.72	156	12.78	5	8.77	15.52	144	12.98	5	8.61

658 the MUMPS package³) are known not to be competitive with respect to algebraic nested dissection-
659 based approaches such as SCOTCH⁴ or METIS⁵ on large 3D problems, we include them in order
660 to study how the Flat Tree and Blocking algorithms behave in general situations.

661 First, an important aspect of using other orderings is that they often produce much more
662 irregular trees, leading to a large number of pruned layers to sequence. The FT permutation reduces
663 the operation count significantly with SCOTCH and METIS, for which we observe an average 31%
664 and 26% reduction compared to the P0 permutation. Gains are also obtained with AMD for most
665 test cases. However, FT does not perform well with AMF. This can be explained by the fact that
666 the former produces too irregular trees which do not fit well with the design of the FT strategy.

667 Second, we evaluate the Blocking algorithm (BLK) and compare it with a regular blocking
668 algorithm (REG) based on the P0 permutation, that divides the initial set of columns into regular
669 chunks of columns. Table 7 shows that the number of groups required to reach $\frac{\Delta}{\Delta_{min}} \leq 1.01$ is
670 much smaller for BLK than for REG in all cases. Our Blocking algorithm is very efficient with
671 most orderings except AMF, where the number of groups created is high (but still lower than REG).

672 All preceding results confirmed that using the combination of the flat tree permutation and the
673 Blocking algorithm, we are able to approach Δ_{min} within a sufficiently small tolerance. In the next
674 section, we try to influence the ordering of the matrix to decrease Δ_{min} .

675 **6. Guided Nested Dissection.** At the moment, as soon as at least one RHS nonzero is
676 present in a tree node, we considered in Section 1 that all operations involving the factors of that
677 node are performed. A smaller granularity of sparsity (inner node sparsity) could be exploited by
678 ordering last the indices of a front corresponding to RHS nonzeros. Because of fill-in, this is only
679 useful for leaf nodes of the pruned tree $T_p(B)$. In general, those leaf nodes may not be very high
680 in the tree, in which case there is not much gain to expect. Given an ordering and a tree, one may
681 think of artificially moving the unknowns corresponding to RHS nonzeros higher in the tree with on
682 one side, a smaller pruned tree, but on the other side, an increase of the factor size and of the fronts
683 receiving those extra unknowns. Better, one may try to guide the nested dissection ordering in order
684 to include as many nonzeros of B within separators during the top-down nested dissection and be
685 able to prune larger subtrees. This will however involve a significant extra cost for applications
686 where each RHS contains several contiguous nodes in the grid, *e.g.*, form a small parallelepiped.

³<http://mumps.enseiht.fr/>

⁴<http://www.labri.fr/perso/pelegrin/scotch/>

⁵<http://glaros.dtc.umn.edu/gkhome/metis/metis/overview>

687 For such applications, the geometry of the RHS nonzeros could however be exploited. A first idea
 688 consists in avoiding problematic RHS by choosing separators that do not intersect RHS nonzeros.
 689 Although this idea could for example be tested by adding edges between RHS nonzeros before
 690 applying SCOTCH or METIS, this does not appear to be so useful in our case, where we observed
 691 much overlap between successive RHS. Another idea, when all RHS are localized in a specific area
 692 of the domain, consists in shifting the separators from the nested dissection to insulate the RHS
 693 in a small part of the domain. Such a modification of the ordering yields an unbalanced tree in
 694 which the RHS nonzeros appear at the smaller side of the tree, improving the efficiency of tree
 695 pruning and resulting in a reduction of Δ_{min} , and thus Δ . This so called *guided* nested dissection
 696 was implemented and tested on the set of test cases shown in Table 8, where we observe that
 697 the number of operations Δ_{min} is decreased, as expected. Since the factor size has also increased
 698 significantly (except on H0, which is very small), one may need to find a trade-off in order to avoid
 699 increasing too much the cost of the factorization.

TABLE 8
 Number of operations ($\times 10^{13}$) during the forward elimination and factor size ($\times 10^9$) for the original (ND) and
 for the guided (GND) nested dissection.

Matrices	5Hz		7Hz		10Hz		H0		H3	
Strategy	ND	GND	ND	GND	ND	GND	ND	GND	ND	GND
Δ_{min}	.22	.19	.69	.62	2.26	1.99	.050	.025	.95	.81
factor size	3.72	5.18	12.8	19.7	44.8	73.4	.24	.37	4.50	5.57

700 **7. Applications and related problems.** We illustrate the scope of this work by present-
 701 ing applications where our contributions can be applied. In applications requiring only part of
 702 the solution, one can show that the tools presented in Section 2 can be applied to the backward
 703 substitution ($UX = Y$), which involves similar mechanisms as the forward elimination [17, 19].
 704 The backward substitution traverses the tree nodes from top to bottom so that the interval mech-
 705 anism is reversed, *i.e.*, the interval from a parent node includes the intervals from its children
 706 and the properties of local optimality are preserved. If the structure of the partial solution re-
 707 quested differs from the RHS structure, another call to the Flat Tree algorithm must then be
 708 performed to optimize the number of operations. Exploiting sparsity also in the backward step can
 709 for instance be useful in some augmented approaches [20] to deal with small matrix updates with-
 710 out complete refactoring, and in some 3D EM geophysics applications [16]. Another application
 711 of this work is the computation of Schur complements, where instead of truncating a factoriza-
 712 tion of the whole system $\begin{pmatrix} A & C \\ B & D \end{pmatrix}$, one exploits the factorization of A to use triangular solves with
 713 sparse RHS. Taking the symmetric case where $C = B^T$, the Schur complement S can be written
 714 $S = D - BA^{-1}B^T = D - B(LL^T)^{-1}B^T = D - (L^{-1}B^T)^T(L^{-1}B^T)$, as in the PDSLIn solver [18].
 715 Since B is sparse, $B' = L^{-1}B^T$ can be computed thanks to the algorithms developed in this article
 716 before computing the sparse product $B'^T B'$.

717 To conclude this discussion, we comment on related problems and algorithms. We have seen
 718 that the Blocking algorithm is closely related to graph algorithms like coloring and maximum
 719 independent set. Concerning the minimization problem (10) which we addressed with the Flat Tree
 720 algorithm, it can also be regarded globally: using the structure of $L^{-1}B$, the problem then consists
 721 in finding a permutation of the columns that minimizes the sum of the intervals weighted with \mathcal{F}_\square .
 722 This interval minimization problem is similar to a sparse matrix profile reduction problem [5, 15].
 723 As mentioned at the beginning of Section 4.1 hypergraph models have also been used in the context

724 of blocking algorithms, with different constraints and objectives compared to ours [3, 18]. Modeling
 725 $L^{-1}B$ as an hypergraph might lead to other heuristics than the Flat Tree algorithm using some
 726 variants of hypergraph partitioning, although dense parts in $L^{-1}B$ might need special treatment.
 727 One advantage of our permutation and blocking algorithms is that, instead of tackling the problem
 728 globally, they decompose the problem into easier subproblems with low complexity by making use
 729 of the separator tree T as much as possible, thereby exploiting the fact that $L^{-1}B$ has a very special
 730 structure closely related to the tree.

TABLE 9

Time (s) of the forward elimination according to the strategy used on a single Intel Xeon core @2.3GHz.

Times	DEN	TP	INI	PO	TP	BLK
H0	673.5	156.2	120.8	95.7	78.1	65.4
5Hz	1291.0	472.3	274.3	224.1	180.0	138.6

731 **Conclusion.** We introduced permutation and blocking algorithms to further improve the tree
 732 pruning [11, 17] and the node interval [4] algorithms introduced in previous work. A first main
 733 contribution of this article is to provide a “flat tree” algorithm to permute right-hand sides in order
 734 to reduce the cost of the forward elimination. As a second contribution, we introduced a Blocking
 735 algorithm that further decreases this cost by adequately choosing groups of right-hand sides that can
 736 be processed together. Although both algorithms are based on geometrical observations, they are
 737 designed with an algebraic approach, giving a general scope to this work. Notions of node optimality
 738 and RHS independence were introduced and formalized, together with theoretical properties to
 739 provide insight and to support the proposed algorithms. Experimental results on real test cases
 740 confirmed the effectiveness of both the *Flat Tree* and the *Blocking* algorithms. Compared to a
 741 Postorder-based permutation, the Flat Tree permutation showed an average (resp. maximum) gain
 742 of 13% (resp. 25%) on the total operation count with a nested dissection ordering, and interesting
 743 parallel properties. Moreover, results with the Blocking algorithm validate our approach since only a
 744 handful of groups is created compared to several hundreds when using a regular blocking technique.
 745 Finally, Table 9 shows that in a sequential setting, time reduction follows operation reduction on
 746 the smallest of our two sets of problems. A detailed performance analysis in multithreaded and
 747 distributed environments is out of the scope of this study and will be the object of future work.

748 **Acknowledgements.** We thank EMGS and SEISCOPE for providing the test cases, and F.-
 749 H. Rouet for his comments on a previous version of this paper. This work was partially supported
 750 by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "In-
 751 vestissements d’Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency
 752 (ANR), and by the MUMPS Consortium.

753

REFERENCES

- 754 [1] P. R. AMESTOY, R. BROSSIER, A. BUTTARI, J.-Y. L’EXCELLENT, T. MARY, L. MÉTIVIER, A. MINIUSSI, AND
 755 S. OPERTO, *Fast 3D frequency-domain full waveform inversion with a parallel Block Low-Rank multifrontal*
 756 *direct solver: application to OBC data from the North Sea*, Geophysics, 81 (2016), pp. R363 – R383.
 757 [2] P. R. AMESTOY, T. A. DAVIS, AND I. S. DUFF, *Algorithm 837: AMD, an approximate minimum degree ordering*
 758 *algorithm*, ACM Transactions on Mathematical Software, 33(3) (2004), pp. 381–388.

- 759 [3] P. R. AMESTOY, I. S. DUFF, J.-Y. L'EXCELLENT, Y. ROBERT, F.-H. ROUET, AND B. UÇAR, *On computing*
760 *inverse entries of a sparse matrix in an out-of-core environment*, SIAM Journal on Scientific Computing,
761 34 (2012), pp. A1975–A1999.
- 762 [4] P. R. AMESTOY, I. S. DUFF, J.-Y. L'EXCELLENT, AND F.-H. ROUET, *Parallel computation of entries of A^{-1} ,*
763 SIAM Journal on Scientific Computing, 37 (2015), pp. C268–C284.
- 764 [5] M. W. BERRY, B. HENDRICKSON, AND P. RAGHAVAN, *Sparse matrix reordering schemes for browsing hypertext,*
765 Lecture notes in applied mathematics, 32 (1996), pp. 99–124.
- 766 [6] J. J. DONGARRA, J. DU CROZ, S. HAMMARLING, AND I. S. DUFF, *A set of level 3 basic linear algebra subpro-*
767 *grams*, ACM Trans. Math. Softw., 16 (1990), pp. 1–17.
- 768 [7] I. S. DUFF, A. M. ERISMAN, AND J. K. REID, *Direct Methods for Sparse Matrices*, Oxford University Press,
769 London, 1986.
- 770 [8] I. S. DUFF AND J. K. REID, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM
771 Transactions on Mathematical Software, 9 (1983), pp. 302–325.
- 772 [9] J. A. GEORGE, *Nested dissection of a regular finite-element mesh*, SIAM Journal on Numerical Analysis, 10
773 (1973), pp. 345–363.
- 774 [10] J. R. GILBERT, *Predicting structure in sparse matrix computations*, SIAM Journal on Matrix Analysis and
775 Applications, 15 (1994), pp. 62–79.
- 776 [11] J. R. GILBERT AND J. W. H. LIU, *Elimination structures for unsymmetric sparse LU factors*, SIAM Journal
777 on Matrix Analysis and Applications, 14 (1993), pp. 334–352.
- 778 [12] G. KARYPIS AND K. SCHLOEGEL, *ParMetis: Parallel Graph Partitioning and Sparse Matrix Ordering Library*
779 *Version 4.0*, University of Minnesota, Department of Computer Science and Engineering, Army HPC
780 Research Center, Minneapolis, MN 55455, U.S.A., Aug. 2003. Users' manual.
- 781 [13] J. W. H. LIU, *The role of elimination trees in sparse factorization*, SIAM Journal on Matrix Analysis and
782 Applications, 11 (1990), pp. 134–172.
- 783 [14] F. PELLEGRINI, *SCOTCH and LIBSCOTCH 5.0 User's guide*, Technical Report, LaBRI, Université Bordeaux I,
784 2007.
- 785 [15] J. K. REID AND J. A. SCOTT, *Reducing the total bandwidth of a sparse unsymmetric matrix*, SIAM Journal on
786 Matrix Analysis and Applications, 28 (2006), pp. 805–821.
- 787 [16] D. V. SHANTSEV, P. JAYSAVAL, S. DE LA KETHULLE DE RYHOVE, P. R. AMESTOY, A. BUTTARI, J.-Y.
788 L'EXCELLENT, AND T. MARY, *Large-scale 3D EM modeling with a Block Low-Rank multifrontal direct*
789 *solver*, Geophysical Journal International, 209 (2017), pp. 1558–1571.
- 790 [17] TZ. SLAVOVA, *Parallel triangular solution in the out-of-core multifrontal approach for solving large sparse linear*
791 *systems*, Ph.D. dissertation, Institut National Polytechnique de Toulouse, Apr. 2009.
- 792 [18] I. YAMAZAKI, X. S. LI, F.-H. ROUET, AND B. UÇAR, *On Partitioning and Reordering Problems in a Hierarchi-*
793 *cally Parallel Hybrid Linear Solver*, in 2013 IEEE 27th International Parallel and Distributed Processing
794 Symposium Workshops & PhD Forum (IPDPSW), Cambridge, MA, United States, May 2013, IEEE, IEEE
795 Computer Society.
- 796 [19] Y.-H. YEUNG, J. CROUCH, AND A. POTHEN, *Interactively cutting and constraining vertices in meshes using*
797 *augmented matrices*, ACM Trans. Graph., 35 (2016), pp. 18:1–18:17.
- 798 [20] Y. H. YEUNG, A. POTHEN, M. HALAPPANAVAR, AND Z. HUANG, *AMPS: an augmented matrix formulation*
799 *for principal submatrix updates with application to power grids*, SIAM Journal on Scientific Computing,
800 (2017). Accepted for publication.



**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