BLOCK LOW-RANK MATRICES WITH SHARED BASES: POTENTIAL AND LIMITATIONS OF THE BLR^2 FORMAT*

CLEVE ASHCRAFT†, ALFREDO BUTTARI‡, AND THEO MARY§

Abstract. We investigate a special class of data sparse rank-structured matrices that combine a flat block low-rank (BLR) partitioning with the use of shared (called nested in the hierarchical case) bases. This format is to $\mathcal{H}^2$ matrices what BLR is to $\mathcal{H}$ matrices: we therefore call it the BLR^2 matrix format. We present algorithms for the construction and LU factorization of BLR^2 matrices, and perform their cost analysis—both asymptotically and for a fixed problem size. With weak admissibility, BLR^2 matrices reduce to block separable matrices (the flat version of HBS/HSS). Our analysis and numerical experiments reveal some limitations of BLR^2 matrices with weak admissibility, which we propose to overcome with two approaches: strong admissibility, and the use of multiple shared bases per row and column.

Key words. Data sparse matrices, block low-rank matrices, block separable matrices, hierarchical matrices, LU factorization, numerical linear algebra.

AMS subject classifications. 65F05, 65G50.

1. Introduction.

Data sparse matrices possess an off-diagonal block low-rank structure: most of their off-diagonal blocks B can be accurately approximated by low-rank matrices: $B \approx XY^T$, with $\text{size}(X) + \text{size}(Y) \ll \text{size}(B)$. This property can be exploited to reduce the number of entries required to store them and the floating-point operations (flops) required to perform standard linear algebra computations on them.

Many different rank-structured matrix formats have been proposed in the literature to exploit this property. The most general format is the hierarchical $\mathcal{H}$ matrix format [10, 11, 20, 21]. Hierarchical formats can be classified based on two main criterions: whether they use weak or strong admissibility, and whether they use nested bases or not.

Weakly admissible formats directly approximate all off-diagonal blocks as low-rank matrices, regardless of their rank, whereas strongly admissible formats recursively partition the off-diagonal blocks until their rank becomes small enough. $\mathcal{H}$ matrices use strong admissibility; with weak admissibility, they reduce to a format called HODLR [8].

The so-called nested bases structure consists of two main components. First, all blocks on the same block-row or block-column share the same X or Y basis, respectively. Second, these low-rank bases are nested across the levels of the hierarchy, that is, the basis at a given level is implicitly represented by the bases at the lower levels. $\mathcal{H}^2$ [11] matrices add to $\mathcal{H}$ these nested bases for an even more compact representation. The HSS [15,33] and HBS [18] formats are weakly admissible versions of $\mathcal{H}^2$ matrices.

In the recent years, non-hierarchical formats based on a flat partitioning have gained attention: the block low-rank (BLR) format [3] has been successfully employed to accelerate many computational science applications, notably in sparse direct methods [7,29], geosciences [4,31], climate modeling [1,12], boundary integral equations [2], and many others [26, 28, 32]. The theoretical and numerical properties of the BLR

*Version of March 15, 2021.

†Ansys (cleve.ashcraft@ansys.com).

‡IRIT, Université de Toulouse, CNRS, Toulouse, France (alfredo.buttari@irit.fr).

§Sorbonne Université, CNRS, LIP6, Paris, France (theo.mary@lip6.fr).

format are well understood [5, 6, 22, 24], and its performance on modern parallel computers has been heavily optimized [7, 9, 13, 16, 23, 29, 30]. The BLR format is usually used with strong admissibility since, unlike hierarchical formats, doing so is not any more complex than using weak admissibility. Furthermore, BLR does not use any nested bases structure. The BLR format can therefore be seen as a flat version of the $\mathcal{H}$ format.

Figure 1.1 summarizes the data sparse matrix formats presented so far. This taxonomy reveals an unexplored corner: flat formats exploiting a nested bases structure. Indeed, surprisingly, even though the BLR format has been extensively studied, there have been no real attempts to introduce nested bases in BLR matrices. Note that since flat formats possess no hierarchy, low-rank bases cannot be *nested*. However, they can still be *shared* across all blocks on any given block-row or block-column. We refer to such a structure as *shared bases*. In a nutshell, if BLR is a flat version of $\mathcal{H}$ matrices, BLR with shared bases becomes a flat version of $\mathcal{H}^2$ matrices: we therefore propose to call such a format BLR^2.

Gillman et al. [18] describe the flat version of their HBS format, which they call block separable (BS), but it is only presented as a first step towards HBS, and is not studied in its own right. Given the success of the BLR format in a wide range of computational science applications, we believe that it is timely and worthy to dedicate a specific investigation of flat data sparse formats exploiting shared bases. This article represents a first attempt at introducing a shared bases structure into BLR matrices. We now describe more precisely our contributions.

1.1. Our contributions. With weak admissibility, the BLR^2 format is essentially equivalent to the BS format of Gillman et al. [18]. As mentioned, this format was only thought of as being a first step towards HBS, and therefore was not investigated in its own right. We therefore begin, in sections 3 through 7, by revisiting the BS format with a dedicated study. In particular:

- We describe a new algorithm to transform a BLR matrix into a BLR^2 one (Algorithm 3.2), which can be used to build a BLR^2 representation of a dense matrix at a lower cost than existing algorithms that skip the intermediate BLR step.
- The existing algorithms to solve linear systems $Ax = b$ where A is under BLR^2 form are based on an inversion scheme that uses the Sherman–Morrison–Woodbury formula [18]. We propose instead new algorithms based on LU factorization and triangular substitution (Algorithms 4.1 and 4.2).
- We perform detailed cost analyses of the BLR^2 format, both for storage and flops for their LU factorization. We first investigate their asymptotic cost, obtaining reduced asymptotic costs in $O(n^{4/3})$ for storage and $O(n^{9/5})$ for flops. This asymptotic analysis is however only valid under the assumption that the shared bases have low enough rank, so we also perform a non-asymptotic analysis, obtaining precise conditions (6.1) and (6.2) for the BLR^2 format to achieve storage and flops gains over the BLR one, respectively.

Based on this study, we propose two new improvements to BLR^2 matrices aiming at reducing the rank of the shared bases.

- We first introduce strong admissibility, that is, we move from a flat version of HBS/HSS to a flat version of $\mathcal{H}^2$. We show how the rank of the shared bases can be significantly reduced by removing only a small number of high-rank blocks from the shared bases.
- We further generalize the BLR^2 representation by not using a unique shared

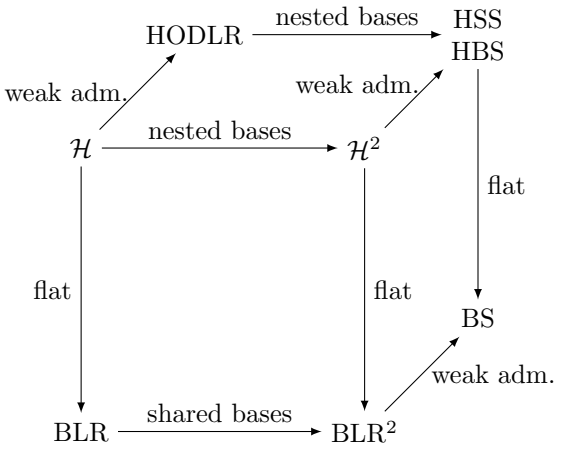


Fig. 1.1: A taxonomy of data sparse matrix formats.

basis per row and column, but multiple bases. We propose several strategies to select groups of blocks that share the same basis.

Throughout the article, we illustrate the analysis and discussion with experiments on a range of matrices coming from various applications. Our experimental results highlight the potential and limitations of the BLR^2 format.

The outline of this article follows the above list of contributions. After introducing BLR and BLR^2 matrices in section 2, we present BLR^2 construction algorithms in section 3 and BLR^2 LU factorization and solution algorithms in section 4. We then perform asymptotic and non-asymptotic cost analyses in sections 5 and 6. Based on these results, we discuss the potential and limitations of the block separable format (BLR^2 with weak admissibility) in section 7. We then propose two directions to overcome these limitations: we propose introducing strong admissibility in section 8 and we investigate the use of multiple shared bases in section 9. We finally provide our conclusions in section 10.

2. Preliminaries. We present BLR and BLR^2 matrices in sections 2.1 and 2.2. We also discuss our experimental setting and our notations (sections 2.3 and 2.4).

2.1. BLR matrices. Consider a dense matrix A partitioned into $b \times b$ blocks A_{ij}. A BLR representation separates the matrix as $A = D + H$, where D is block diagonal (with $D_{ii} = A_{ii}$) and where all blocks of H are low rank:

$$H_{ij} = \underbrace{X_{ij}}_{b \times r_{ij}} \underbrace{Y_{ij}^T}_{r_{ij} \times b}. \tag{2.1}$$

where the dimension r_{ij} corresponds to the rank of the H_{ij} block. We refer to matrices X_{ij} and Y_{ij} as *bases*.

The representation (2.1) can be computed by several methods. The singular value decomposition of H_{ij} yields the optimal rank r_{ij}, but requires $O(b^3)$ flops. Throughout this article, we will rather assume the use of a cheaper method requiring only $O(b^2 r_{ij})$ flops. Many such methods exist, such as rank-revealing QR factorizations, adaptive cross approximations, randomized methods, etc.

The ranks r_{ij} are in general very different but, to simplify the notation and the analysis, we will use a single quantity r to denote them. In the cost analyses where r appears, r should formally be defined as the maximum of all r_{ij}, although more realistic estimates are obtained by taking r to be the average rank.

We also recall in Algorithm 2.1 how to compute the LU factorization of a BLR matrix.

Algorithm 2.1 Standard BLR factorization algorithm.

{**Input:** a BLR matrix $A = D + H$ as defined by (2.2).}
{**Output:** the BLR LU factors $A = LU$.}
for $k = 1$ **to** p **do**
  $A_{kk} = L_{kk}U_{kk}$.
  for $i = k + 1$ **to** p **do**
    $Y_{ik} = U_{kk}^{-T} Y_{ik}$.
    $X_{ki} = L_{kk}^{-1} X_{ki}$.
    for $j = k + 1$ **to** p **do**
      $A_{ij} \leftarrow A_{ij} - X_{ik}(Y_{ik}^T X_{kj})Y_{kj}^T$.
    end for
  end for
end for

To summarize, a BLR matrix $A = D + H$ is represented by

$$D_{ii} = A_{ii} \quad (H_{ii} = 0), \tag{2.2a}$$

$$H_{ij} = X_{ij}Y_{ij}^T, \quad i \neq j. \tag{2.2b}$$

2.2. Block separable matrices. The BLR^2 format is based on the idea of finding a common (or joint) basis for all blocks in a given block-row or block-column of H, namely

$$H_{ij} = \underbrace{X_i}_{b\times s_i}\ \underbrace{C_{ij}}_{s_i\times t_j}\ \underbrace{Y_j^T}_{t_j\times b}. \tag{2.3}$$

We moreover require X_i and Y_j to have orthonormal columns (hereinafter **OC** matrices). For a given block-row i, all blocks H_{ij} share the same left basis X_i and, for a given block-column j, all blocks H_{ij} share the same right basis Y_j. Therefore, we call X_i and Y_j *shared bases*.

The dimensions s_i and t_j are the ranks of the shared bases, and are different for each i and j. For the simplicity of the notation and analysis, we will again use a single quantity s to denote the ranks of the shared bases.

The matrices C_{ij}, which we call *coupling matrices*, play a very important role. Indeed, because the shared bases are **OC**, C_{ij} contains all the norm and spectrum information of the block H_{ij}: it has the same singular values and therefore the same rank r_{ij}. Crucially, C_{ij} is thus a low-rank matrix that can be represented as

$$C_{ij} = \underbrace{\Phi_{ij}}_{s_i\times r_{ij}}\ \underbrace{\Psi_{ij}^T}_{r_{ij}\times t_j}. \tag{2.4}$$

As soon as $s_i t_j \geq (s_i + t_j)r_{ij}$, that is, $r \leq s/2$, storing C_{ij} under this form is more efficient than in full form.

Table 2.1: List of test matrices used in this article.

Name	Matrix size n	Block size b	Application
P64	4096	128	Poisson
P160	25600	256	Poisson
GaAsH6	6232	128	Quantum chemistry
nd24k	7785	128	2D/3D
Fault_639	7983	128	Contact mechanics
nlpkkt80	14080	256	Optimization
Serena	15552	256	Structural
Cube_Coup_dt0	21072	256	Coupled consolidation
Geo_1438	13254	256	Geomechanical

To summarize, a BLR^2 matrix $A = D + H$ is represented by

$$D_{ii} = A_{ii} \quad (H_{ii} = 0), \tag{2.5a}$$
$$H_{ij} = X_i C_{ij} Y_j^T, \quad i \neq j, \tag{2.5b}$$
$$C_{ij} = \Phi_{ij} \Psi_{ij}^T. \tag{2.5c}$$

2.3. Experimental setting. All our numerical experiments were carried out with MATLAB R2018b. To test the algorithms, we have used the matrices listed in Table 2.1. These matrices are Schur complements of sparse matrices, (corresponding to the root separator of the sparse factorization), all of which come from the SuiteSparse collection [17], except P64 and P160, which are 3D Poisson problems with Schur complement of dimension 64^2 and 160^2. We unsymmetrize all matrices for the sake of simplicity.

2.4. Notations. For convenience, we recall here the main notations used in this article. We consider an $n \times n$ matrix A partitioned into $b \times b$ blocks. We denote by r the ranks of the local blocks and by s the ranks of the shared bases. We also define $p = n/b$. We refer to computing the low-rank form of a block $XY^T \approx B$ as *compression* and to restoring an approximate full-rank form $\tilde{B} = XY^T$ as *decompression*; by the same token we call (de)compression of a matrix the operation of (de)compressing all the blocks it is made of.

3. Construction algorithms. Given a dense matrix, what is the best method to build its BLR^2 representation? In this section we compare two different construction algorithms. The simplest approach, presented in section 3.1, is Algorithm 3.1, which builds the BLR^2 representation by directly compressing the dense matrix. In section 3.2, we propose instead Algorithm 3.2, which first builds a BLR representation before transforming it into a BLR^2 one. We show in section 3.3 that the intermediate BLR construction leads to lower cost.

3.1. Dense $\rightarrow$ BLR^2 construction algorithm. Algorithm 3.1 proceeds in three steps. First, the shared bases X_i of each block-row are computed (line 4). (Starting with the shared bases of the block-columns is equivalent). At this point, each block H_{ij} can be represented as $X_i B_{ij}^T$, where X_i is **OC**. Then, each shared basis Y_j is computed by agglomerating and compressing all B_{ij} matrices on block-column

j together (line 7). Finally, the third step is to compute a low-rank representation of the coupling matrices (line 11) which, as noted before, have usually low rank.

Algorithm 3.1 Dense→BLR^2 construction algorithm.

1: {**Input:** a dense matrix A.}
2: {**Output:** the BLR^2 matrix $A = D + H$ as defined by (2.5).}
3: **for** $i = 1$ **to** p **do**
4: Compress $[H_{i1} \cdots H_{ip}] = X_i[B_{i1}^T \cdots B_{ip}^T]$ where X_i is **OC**.
5: **end for**
6: **for** $j = 1$ **to** p **do**
7: Compress $[B_{1j} \cdots B_{pj}] = Y_j[C_{1j}^T \cdots C_{pj}^T]$ where Y_j is **OC**.
8: **end for**
9: **for** $i = 1$ **to** p **do**
10: **for** $j = 1$ **to** p **do**
11: Compress $C_{ij} = \Phi_{ij}\Psi_{ij}^T$.
12: **end for**
13: **end for**

Algorithm 3.2 Dense→BLR→BLR^2 construction algorithm.

1: {**Input:** a dense matrix A.}
2: {**Output:** the BLR^2 matrix $A = D + H$ as defined by (2.5).}
3: **for** $i = 1$ **to** p **do**
4: **for** $j = 1$ **to** p **do**
5: Compress $H_{ij} = X_{ij}Y_{ij}^T$ where X_{ij} and Y_{ij} are **OC**.
6: **end for**
7: **end for**
8: **for** $j = 1$ **to** p **do**
9: Compress $[Y_{1j} \cdots Y_{pj}] = Y_j[E_{1j}^T \cdots E_{pj}^T]$ where Y_j is **OC**.
10: **end for**
11: **for** $i = 1$ **to** p **do**
12: **for** $j = 1$ **to** p **do**
13: Orthonormalize $E_{ij} = F_{ij}\Psi_{ij}^T$ where Ψ_{ij} is **OC**.
14: **end for**
15: **end for**
16: **for** $i = 1$ **to** p **do**
17: Compress $[X_{i1}F_{i1} \cdots X_{ip}F_{ip}] = X_i[\Phi_{i1} \cdots \Phi_{ip}]$ where X_i is **OC**.
18: **end for**

3.2. Dense → BLR → BLR^2 construction algorithm. Next we describe Algorithm 3.2, which is based on the observation that a BLR representation can be transformed into a BLR^2 one. The basic idea is to first compute a BLR representation (line 5, yielding blocks H_{ij} under the form $X_{ij}Y_{ij}^T$, where X_{ij} is **OC**), and then to obtain each shared basis Y_j from the compression of the agglomerated Y_{ij} bases on block-column j (line 9). Similarly, the shared bases X_i could be obtained from agglomerating the X_{ij} basis on each block-row. However, there are two key points to make the algorithm effective.

The first key point is not to compress the X_{ij} bases on their own, because they are **OC**. Indeed, the Y_{ij} bases contain all the information on the spectrum of the

local block H_{ij}, and it is important to exploit this information in the compressions of *both* shared bases. To do so, we must include in the computation of the X_i bases the weight information contained in matrices E_{ij} obtained from the compression of the Y_j bases (line 9). For example, agglomerating matrices $X_{ij}E_{ij}$ on block-row i and compressing them together would yield

$$[X_{i1}E_{i1} \cdots X_{ip}E_{ip}] = X_i[C_{i1} \cdots C_{ip}]$$

, where X_i is **OC** and where it remains to compress the C_{ij} coupling matrices.

However, the second keypoint is that it is more efficient to orthonormalize matrices E_{ij} first, as this allows to obtain the coupling matrices C_{ij} directly under their low-rank form $\Phi_{ij}\Psi_{ij}^T$. Indeed, assume E_{ij} is orthonormalized as $E_{ij} = F_{ij}\Psi_{ij}^T$ (line 13), where Ψ_{ij} is **OC** (by means of an LQ factorization, for example). Then, by compressing the agglomerated $X_{ij}F_{ij}$ (line 17), we obtain both the X_i shared bases and the Φ_{ij} local bases of the coupling matrices C_{ij}.

3.3. Cost analysis. Let us now compare the costs of the two construction algorithms 3.1 and 3.2. We begin with Algorithm 3.1.

- Line 4: computing the shared basis of any given $b \times n$ block-row requires $O(nbs)$ flops, and so the total cost of compressing all p block-rows is $O(n^2 s)$.
- Line 7 requires p compressions of $b \times O(ps)$ matrices of rank s, amounting to $O(nps^2)$ flops.
- Line 11 requires p^2 compressions of $s \times s$ matrices of rank r, amounting to $O(p^2 s^2 r)$ flops.
- Total: $O(n^2 s + nps^2 + p^2 s^2 r) = O(n^2 s)$. The overall cost is dominated by the cost $O(n^2 s)$ of the first step because $ps \leq pb = n$.

For Algorithm 3.1, we thus recover the $O(n^2 s)$ cost mentioned in [18, p. 226]. We now turn to Algorithm 3.2.

- Line 5 requires p^2 compressions of $b \times b$ blocks of rank r, amounting to $O(n^2 r)$ flops.
- Line 9 performs p compressions of $b \times O(pr)$ matrices of rank s, which require $O(npsr)$ flops.
- Line 13 carries out p^2 orthonormalizations (LQ factorizations) of $r \times s$ matrices, which require $O(p^2 sr^2)$ flops.
- Line 17 first requires p^2 products of cost $O(br^2)$, for a total of $O(npr^2)$. Then, p compressions of $b \times O(pr)$ matrices of rank s are performed, which require $O(npsr)$ flops.
- Total: $O(n^2 r + npsr + p^2 sr^2 + npr^2) = O(n^2 r)$. The overall cost is dominated by the cost $O(n^2 r)$ of the first step.

The cost of Algorithm 3.2 is therefore roughly a factor s/r smaller than that of Algorithm 3.1, ignoring lower order terms. We conclude from this cost analysis that using an intermediate BLR construction leads to a reduced cost.

We illustrate this conclusion in Table 3.1, where we experimentally compare the construction costs of each algorithm on matrix P64 for varying ε. The comparison shows that Algorithm 3.2 outperforms Algorithm 3.1, and that this trend becomes stronger as ε increases due to a larger s/r ratio. Note that this trend might be explained by the fact that s cannot exceed b, and so as ε decreases, s can stop increasing whereas r continues to grow. In the rest of this article, we will use Algorithm 3.2.

4. LU factorization and solution algorithms. In this section we explain how to compute the BLR^2 LU factorization, that is, how to decompose a BLR^2 matrix A

Table 3.1: Construction costs (in GigaFlops) of Algorithms 3.1 and 3.2 on matrix P64 (see section 2.3), with varying ε.

	Algorithm 3.1	Algorithm 3.2	ratio	average s/r
$\varepsilon = 10^{-12}$	12.66	5.44	2.3	5.7
$\varepsilon = 10^{-8}$	9.53	2.13	4.5	10.2
$\varepsilon = 10^{-4}$	5.03	0.37	13.5	22.8

into a lower triangular BLR^2 matrix L and an upper triangular BLR^2 matrix U such that $A = LU$ (Algorithm 4.1 described in section 4.1). We also explain how to exploit these BLR^2 LU factors to solve a linear system $Ax = b$ (Algorithm 4.2 in section 4.2) and present backward error results in section 4.3.

Other algorithms to invert BLR^2 matrices or solve a linear system involving a BLR^2 matrix have been proposed. In particular, Gillman et al. [18] propose an inversion scheme based on the Sherman–Morrison–Woodbury formula. However, to our knowledge, the algorithms presented here are the first of the literature to exploit an LU factorization for BLR^2 matrices, similarly to LU-based algorithms for other matrix formats, such as BLR (Algorithm 2.1, [5]), $\mathcal{H}$ [14, 19], HSS [15], and $\mathcal{H}^2$ [25].

4.1. LU factorization algorithm. Given a BLR^2 matrix A as defined in (2.5), Algorithm 4.1 computes two lower and upper triangular BLR^2 matrices L and U satisfying $A = LU$ (see (4.6) below), by performing the following operations.

At each step k, we first compute the LU factorization of the diagonal block $A_{kk} = L_{kk}U_{kk}$ (line 8). Then, we perform the triangular solves $L_{ik} = A_{ik}U_{kk}^{-1}$ and $U_{kj} = L_{kk}^{-1}A_{kj}$. Here, the crucial observation is that all the blocks $A_{ik} = X_iC_{ik}Y_k^T$ in the k-th block-column share the same right basis Y_k, so that only one triangular solve $V_k = U_{kk}^{-T}Y_k$ (line 9) suffices. Similarly, to compute U_{kj} we only need to perform one triangular solve $W_k = L_{kk}^{-1}X_k$ (line 10).

Finally, the last step is the update of the blocks of the trailing submatrix:

$$A_{ij} \leftarrow A_{ij} - A_{ik}A_{kj}, \tag{4.1}$$

for all $i, j > k$. Here again the crucial observation is that all $A_{ik} = X_iC_{ik}V_k^T$ share the same right basis V_k and all $A_{kj} = W_kC_{kj}Y_j^T$ share the same left basis W_k, so that we actually only need to compute one product, $Z_k = V_k^TW_k$ (line 11). The update (4.1) then becomes

$$X_iC_{ij}Y_j^T \leftarrow X_iC_{ij}Y_j^T - X_iC_{ik}Z_kC_{kj}Y_j^T, \tag{4.2}$$

$$= X_i\big(C_{ij} - C_{ik}Z_kC_{kj}\big)Y_j^T. \tag{4.3}$$

Thus, (4.3) shows that we only need to update the coupling matrices (line 14).

The update of the diagonal blocks A_{kk} is a special case, since A_{kk} is not represented under low rank form. The updates $A_{kk} \leftarrow A_{kk} - A_{k\ell}A_{\ell k}$ for all $\ell < k$ can be efficiently accumulated by noticing that

$$A_{kk} - \sum_{\ell<k} A_{k\ell}A_{\ell k} = A_{kk} - X_k\left(\sum_{\ell<k} C_{k\ell}Z_\ell C_{\ell k}\right)Y_k^T. \tag{4.4}$$

Thus we can accumulate the products $C_{k\ell}Z_\ell C_{\ell k}$ in a temporary workspace C_{kk} (initialized to zero on line 4) and only perform the decompression $X_kC_{kk}Y_k^T$ once, at the beginning of step k (line 7).

Note that since the coupling matrices usually have low rank, the update of the coupling matrices on line 14 takes the form

$$C_{ij} - C_{ik}Z_kC_{kj} = \Phi_{ij}\Psi_{ij}^T - \Phi_{ik}\big(\Psi_{ik}^T Z_k \Phi_{kj}\big)\Psi_{kj}^T. \tag{4.5}$$

We thus recover the standard form of a low-rank matrix update, which amounts to adding together two low-rank matrices and recompressing the result to avoid rank growth [5], [26, Chap. 3].

Algorithm 4.1 BLR^2 LU factorization algorithm.

1: {**Input:** a BLR^2 matrix $A = D + H$ as defined by (2.5).}
2: {**Output:** the triangular factors L and U satisfying (4.6).}
3: **for** $k = 1$ **to** p **do**
4: Initialize $C_{kk} = 0$.
5: **end for**
6: **for** $k = 1$ **to** p **do**
7: $A_{kk} \leftarrow A_{kk} + X_k C_{kk} Y_k^T$.
8: $A_{kk} = L_{kk}U_{kk}$.
9: $V_k = U_{kk}^{-T} Y_k$.
10: $W_k = L_{kk}^{-1} X_k$.
11: $Z_k = V_k^T W_k$.
12: **for** $i = k + 1$ **to** p **do**
13: **for** $j = k + 1$ **to** p **do**
14: $C_{ij} \leftarrow C_{ij} - C_{ik}Z_kC_{kj}$.
15: **end for**
16: **end for**
17: **end for**

Overall, we have therefore computed lower and upper triangular matrices L and U, respectively, that satisfy

$$A = LU, \tag{4.6a}$$

$$L_{ij} = X_i C_{ij} V_j^T, \quad V_j = U_{jj}^{-T} Y_j, \quad i > j, \tag{4.6b}$$

$$U_{ij} = W_i C_{ij} Y_j^T, \quad W_i = L_{ii}^{-1} X_i, \quad i < j. \tag{4.6c}$$

A key observation is that the shared bases X_i and Y_j of the original matrix A are preserved throughout the factorization and become the left bases of L and the right bases of U, respectively. Moreover, since the right bases of L are given by $V_j = U_{jj}^{-T} Y_j$, and similarly the left bases of U are given by $W_i = L_{ii}^{-1} X_i$, V_j and W_i have the same rank as Y_j and X_i, respectively. Therefore, Algorithm 4.1 has the notable property that it does lead not any rank growth in the shared bases. Only the coupling matrices exhibit rank growth.

However, we now have two BLR^2 matrices (L and U) and thus four shared bases, and so the storage for the shared bases has doubled. Since this can represent a significant increase of the overall storage, a possible strategy is to only store explicitly the original bases X_i and Y_j. Indeed, using (4.6b) and (4.6c), the new bases V_j and W_i can be implicitly applied at the price of an additional triangular solve.

4.2. LU solution algorithm. Algorithm 4.2 describes how to solve a linear system $Ax = b$ given the BLR^2 LU factorization $A = LU$ defined by (4.6). The algorithm is based on standard forward ($Ly = b$) and backward ($Ux = y$) substitutions,

which are given by the formulas

$$y_i \leftarrow L_{ii}^{-1}\left(b_i - \sum_{j=1}^{i-1} L_{ij}y_j\right), \quad i = 1\colon p. \tag{4.7}$$

$$x_i \leftarrow U_{ii}^{-1}\left(y_i - \sum_{j=i+1}^{p} U_{ij}y_j\right), \quad i = p\colon 1. \tag{4.8}$$

Exploiting the BLR^2 structure of the LU factors to reduce the cost of these substitutions is however not completely straightforward: simply replacing L_{ij} and U_{ij} by $X_iC_{ij}V_j^T$ and $W_iC_{ij}Y_j^T$ in the above equations is not sufficient.

Algorithm 4.2 BLR^2 LU solution algorithm.

1: {**Input:** BLR^2 LU factors L and U defined (4.6) and a right-hand side b.}
2: {**Output:** the solution x to the system $LUx = b$.}
3: Forward substitution $Ly = b$
4: $y \leftarrow b$.
5: **for** $j = 1$ **to** p **do**
6: $\quad t \leftarrow 0$.
7: $\quad$ **for** $i = 1$ **to** $j - 1$ **do**
8: $\quad\quad t \leftarrow t + C_{ij}z_j$.
9: $\quad$ **end for**
10: $\quad y_j \leftarrow y_j - X_jt$.
11: $\quad y_j \leftarrow L_{jj}^{-1}y_j$.
12: $\quad z_j = V_j^Ty_j$.
13: **end for**
14: Backward substitution $Ux = y$
15: $x \leftarrow y$.
16: $z \leftarrow 0$.
17: **for** $i = p$ **to** 1 **by** -1 **do**
18: $\quad t \leftarrow 0$.
19: $\quad$ **for** $j = i + 1$ **to** p **do**
20: $\quad\quad t \leftarrow t + C_{ij}z_j$.
21: $\quad$ **end for**
22: $\quad x_i \leftarrow x_i - W_it_i$.
23: $\quad x_i \leftarrow U_{ii}^{-1}x_i$.
24: $\quad z_i \leftarrow Y_ix_i$.
25: **end for**

For example, in the case of the forward substitution ($Ly = b$), we obtain

$$y_i \leftarrow L_{ii}^{-1}\left(b_i - \sum_{j=1}^{i-1} X_i(C_{ij}(V_j^Ty_j))\right) \tag{4.9}$$

$$= L_{ii}^{-1}\left(b_i - X_i\sum_{j=1}^{i-1} C_{ij}(V_j^Ty_j)\right), \tag{4.10}$$

so a first obvious improvement is to mutualize the multiplication with X_i over all terms in the sum by accumulating them in some vector t of b elements (line 8) before

Table 4.1: Backward error (4.11) obtained by BLR and BLR^2 LU factorizations to solve the linear system $Ax = b$, for matrix P64 and $x = [1 \dots 1]^T$, with different ε.

$\varepsilon = 10^{-4}$		$\varepsilon = 10^{-8}$		$\varepsilon = 10^{-12}$	
BLR	BLR^2	BLR	BLR^2	BLR	BLR^2
6.79e-05	6.82e-05	8.64e-09	8.69e-09	2.98e-13	3.00e-13

the product with X_i (line 10). More subtly, the products $V_j^T y_j$ can also be mutualized over the steps i, at the price of an extra vector z of n elements to store the results (line 12).

The case of backward substitution is similar.

4.3. Backward error analysis. The error analysis from [22] proves that solving a linear system $Ax = b$ by BLR LU factorization yields a computed solution $\widehat{x}$ satisfying the backward error bound

$$\frac{\|A\widehat{x} - b\|}{\|A\|\|\widehat{x}\| + \|b\|} = O(\varepsilon). \tag{4.11}$$

Adapting this analysis to the BLR^2 LU factorization is outside our scope, but we expect the bound (4.11) to be preserved, since Algorithms 4.1 and 4.2 essentially perform the same operations as the BLR LU factorization and solution algorithms, except that they exploit the shared bases structure of the matrix.

For reference, in Table 4.1 we experimentally compare the backward error obtained by solving the linear system $Ax = b$ by BLR and BLR^2 LU factorization, for a variety of matrices A and where $x = [1 \dots 1]^T$ is the vector of ones (and where b is computed as the matrix–vector product Ax). The BLR^2 error is almost consistently larger than the BLR one, but only by a very small amount. We therefore conclude that the introduction of shared bases has a negligible impact on the stability and accuracy of the solution.

5. Asymptotic cost analysis. In many problems of interest, the ranks r of the local blocks can be shown to be small, increasing slowly or not at all with the problem size n and the block size b. As a consequence, the BLR factorization can significantly reduce the asymptotic costs in both storage and flops. Specifically, under the assumption that r is a small constant, the analysis of [5] obtains costs in $O(n^{3/2})$ for storage and in $O(n^2)$ for flops, which both have a much weaker dependence on n than their counterparts for full dense matrices ($O(n^2)$ storage and $O(n^3)$ flops).

It is not clear whether these costs still apply to the BLR^2 factorization or if they can be even further improved. The main question to elucidate is the dependence of the shared bases ranks s on the block size b. Gillman et al. [18] give the estimate $s \sim \log b$ for one-dimensional boundary integral equations, and under this assumption, prove a $O(n^{9/5})$ cost for their proposed inversion scheme based on the Sherman–Morrison–Woodbury formula. Note that for general matrices with arbitrary s, the complexity of this inversion scheme scales as s^3 [18].

We now perform an asymptotic cost analysis of the BLR^2 format in the same spirit as the above references. We assume $r = O(b^\alpha)$ and $s = O(b^\beta)$, for $\alpha, \beta \in [0, 1]$. In particular, if both r and s are small constants (that is, $\alpha = \beta = 0$), we obtain asymptotic costs in $O(n^{4/3})$ for storage and in $O(n^{9/5})$ flops for Algorithm 4.1 (thus recovering the same asymptotic cost as the inversion scheme of [18]).

5.1. Storage. The number of entries required to store a BLR^2 matrix can be computed as the sum of three terms:

- The diagonal full-rank blocks: $O(pb^2)$ entries.
- The shared bases: $O(ns)$ entries.
- The coupling matrices: $O(p^2sr)$ entries.

Since $s \leq b$, the overall storage is therefore in $O(pb^2+p^2sr) = O(nb+n^2b^{\alpha+\beta-2})$, which is minimized when $b = O(n^{1/(3-\alpha-\beta)})$. For this choice of block size, the resulting storage cost is

$$\text{Storage} = O(n^{\frac{4-\alpha-\beta}{3-\alpha-\beta}}). \tag{5.1}$$

Therefore, if both r and s are small constants ($\alpha = \beta = 0$), we obtain a storage cost in $O(n^{4/3})$, which is significantly lower than that of the BLR format (in $O(n^{3/2})$). For reference, the two-level MBLR storage cost is $O(n^{4/3})$ [6]. If, on the contrary, s grows linearly with b ($\beta = 1$), we recover an asymptotic cost for BLR^2 that is identical to the BLR one.

5.2. LU factorization. Algorithm 4.1 calls several kernels whose cost is given as follows:

- LU factorization of the diagonal blocks (line 8): $O(pb^3)$ flops.
- Triangular solves (lines 9 and 10): $O(pb^2s)$ flops.
- Computing the product Z_k (line 11): $O(pbs^2)$ flops.
- Update of the coupling matrices (line 14): $O(p^3s^2r)$ flops.
- Update of the diagonal blocks (line 7): $O(pb^2s)$ flops.

The total cost of the factorization is the sum of the above costs, and is thus in $O(pb^3 + p^3s^2r) = O(nb^2 + n^3b^{\alpha+2\beta-3})$. This expression is minimized for a block size $b = O(n^{2/(5-\alpha-2\beta)})$, which yields the total cost

$$\text{LU flops} = O(n^{\frac{9-\alpha-2\beta}{5-\alpha-2\beta}}). \tag{5.2}$$

For $\alpha = \beta = 0$, we obtain a cost in $O(n^{9/5})$, which represents a significant improvement with respect to the cost of the classical BLR factorization, $O(n^2)$, and is identical to the cost of the inversion scheme proposed in [18]. For $\beta = 1$ (and $\alpha = 0$), we obtain a cost in $O(n^{7/3})$, which is thus worse than the BLR cost. In fact, to obtain a BLR^2 cost no worse than the BLR one, we must have $\beta \leq (1 + \alpha)/2$.

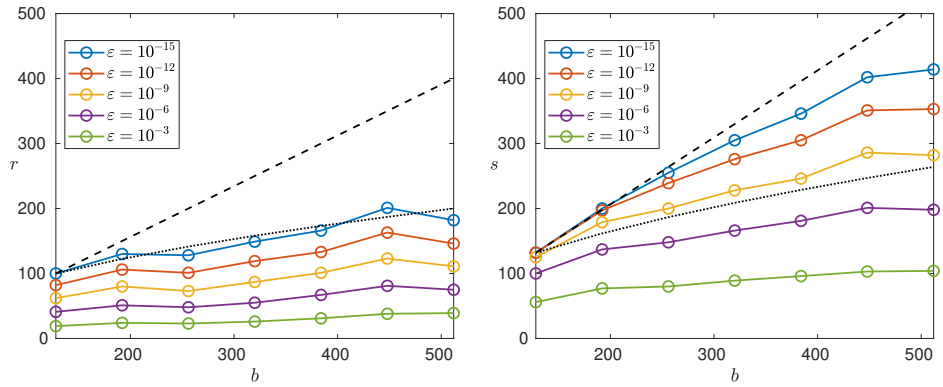
5.3. Experimental assessment. In order to assess what asymptotic gains can be expected from the BLR^2 format, we experimentally estimate the values of α and β by investigating the dependence of the ranks r and s on the block size b. We report the results for the P64 matrix in Figure 5.1 (other matrices exhibit similar trends).

Figure 5.1a shows that the ranks of the local blocks r (which are also equal to the ranks of the coupling matrices) are indeed quite small and grow only slowly with the block size b, at a rate of at most $\sqrt{b}$. In contrast, Figure 5.1b shows that the ranks of the shared bases s are much larger, and grow with b at a faster rate, unless ε is large.

These experiments suggest that we are not likely to obtain much asymptotic improvement from the use of the BLR^2 format over the BLR one, and motivates the development of the non-asymptotic analysis of the next section.

6. Non-asymptotic cost analysis. We next perform a non-asymptotic cost analysis to get further insight on the conditions under which we can expect the BLR^2 format to achieve gains with respect to the BLR one.

6.1. Storage. Compared with the BLR representation, in the BLR^2 case we replace the $b \times b$ blocks of rank r ($2p^2br$ entries in total) by smaller, $s \times s$ coupling



(a) Maximal r.

(b) Maximal s.

Fig. 5.1: Maximal ranks r and s for varying block size b and ε. The dashed and dotted black lines indicate a growth proportional to b and $\sqrt{b}$, respectively. The P64 matrix is used (see section 2.3).

matrices of corresponding rank ($2p^2sr$ entries). Therefore, the BLR^2 storage is less than the BLR storage as soon as this gain is large enough to compensate the extra storage needed for the shared bases ($2ns$ entries). In other words, we have the condition

$$2p^2(b-s)r \geq 2ns \quad \Leftrightarrow \quad \left(\frac{b^2}{nr}+1\right)s \leq b \quad \Leftrightarrow \quad \frac{s}{b} \leq 1 - \frac{b^2}{rn+b^2}. \tag{6.1}$$

The BLR storage is minimized for a block size in $\Theta(\sqrt{nr})$ [5]. For such a block size, the condition reduces to $cs \leq b$, where $c > 1$ is a fixed constant. In practice c is typically a small constant, but the conclusion is nevertheless that s must be significantly smaller than b for BLR^2 to outperform BLR.

6.2. LU factorization. We now compare the costs of Algorithm 4.1 and that of the standard BLR factorization (Algorithm 2.1). We analyze the costs of the operations required at any given step k, and write $p_k = p - k$ the number of off-diagonal blocks in the trailing submatrix at step k.

- Lines 8 (Alg. 4.1) and 4 (Alg. 2.1) both perform the LU factorization of the diagonal block A_{kk} and have the same cost ($2b^3/3$ flops).
- The cost of the triangular solves is reduced thanks to the use of shared bases: at step k, Alg. 4.1 (lines 9 and 10) only requires $2b^2s$ flops, while Alg. 2.1 (lines 6 and 7) requires $2b^2p_kr$ flops. This yields a first gain of $2b^2(p_kr - s)$ flops (**gain 1**).
- The update of the coupling matrices (line 14 of Alg. 4.1) requires two extra operations: computing Z_k (line 11), for $2bs^2$ flops (**extra cost 1**), and multiplying Z_k with either C_{ik} or C_{kj}, for $2p_k^2s^2r$ flops (**extra cost 2**).
- Once these extra operations have been performed, the update of the coupling matrices reduces to a low-rank matrix update $C_{ij} \leftarrow C_{ij} - VW$, where all matrices involved are of rank at most r. This is therefore a similar operation to the updates in the BLR factorization (line 9 of Alg. 2.1), the difference being the latter involve $b \times b$ matrices instead of $s \times s$ ones. Therefore the difference

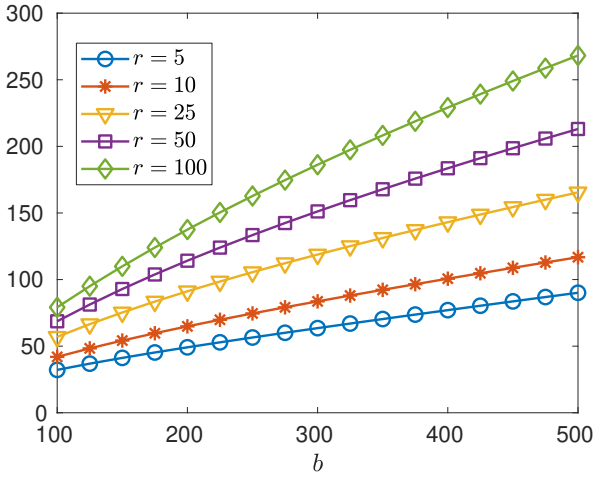


Fig. 6.1: Largest value of s such that condition (6.2) is satisfied, for $p_k = 10$, $c = 6$, and varying b and r.

in cost between these operations leads to another gain of $cp_k^2 r^2(b-s)$ flops (**gain 2**), where c is a constant whose precise value depends on the specific recompression algorithm used. Since at least two matrix–matrix products are needed, $c \geq 4$.

In summary, at step k Algorithm 4.1 performs less flops than Algorithm 2.1 if

$$2b^2(p_k r - s) + cp_k^2 r^2(b-s) - 2bs^2 - 2p_k^2 s^2 r > 0. \tag{6.2}$$

The first two terms on the left-hand side of this equation are clearly positive since $s \leq \min(b, p_k r)$, and correspond to **gains 1 and 2** listed above. The question is therefore whether these gains are large enough to compensate the (negative) third and fourth terms associated with **extra costs 1 and 2**. The answer obviously depends on the precise value of each of the involved constants.

Unlike the condition on storage (6.1), (6.2) is too complicated to obtain a simple rule on how small s should be compared to b. We therefore turn to numerical examples to get further insight. Given a matrix of fixed size n (50000 in our experiments), we summed the left-hand side of Equation (6.2) over $p_k = 1\colon n/b$. For values of b between 100 and 500, Figure 6.1 plots the largest value of s such that this quantity is positive (c is set to 6). Expectedly, as b increases larger and larger s can satisfy condition (6.2). More interestingly, the same trend arises as the rank r increases, which is due to the fact that the gain terms in the condition grow faster with r than the extra cost terms. Most importantly, Figure 6.1 clearly shows that, unless r is very large, s needs to be significantly smaller than b for any gain to be possible: for example, for $b = 200$ and $r = 10$, (6.2) is satisfied for $s \leq 66$.

6.3. Experimental assessment. The conclusions from both the storage and flops analysis are similar: the BLR^2 format can outperform the BLR one only when the rank of the shared bases s is small with respect to the block size b. We now assess experimentally to what extent this condition is satisfied for our test problems.

Figure 6.2 compares the storage and flops required by the BLR and BLR^2 representations for a variety of matrices and for different threshold ε. Unfortunately, in

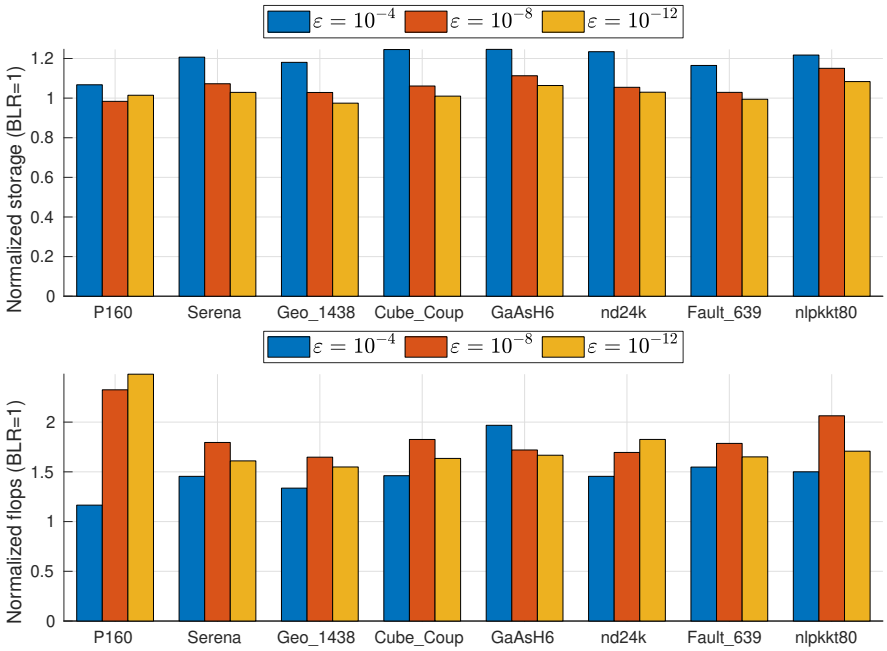


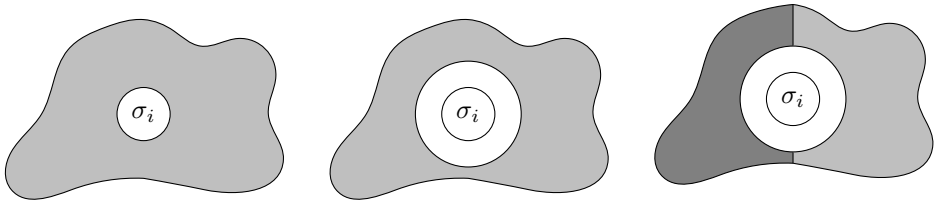
Fig. 6.2: Weakly admissible BLR^2 storage and flops for LU factorization, normalized by that of the BLR format, for different matrices and varying ε.

almost all cases, the BLR^2 format requires slighly more storage than the BLR one, and a much higher flop count. This indicates that, for many problems, the rank of the shared bases is simply not small enough for the weakly admissible BLR^2 format to be beneficial.

7. Discussion: potential and limitations of BLR^2 vs. BLR matrices. The analysis and experiments of the previous sections suggest that block separable matrices (BLR^2 matrices with weak admissibility) are not competitive with BLR matrices, neither in terms of storage nor flops to factorize them. For the BLR^2 representation to be competitive with the BLR one, the rank of the shared bases s should be significantly smaller than what we have observed it to be for several real problems. Furthermore the situation does not seem to improve significantly as the matrix size increases.

This is perhaps one of the reasons why block separable matrices have rarely been considered in the literature. In fact, Gillman et al. [18] only introduce block separable matrices as a first step towards the HBS representation, which recursively applies the block separable format to the coupling matrices C_{ij}. Employing a hierarchical approach is indeed one possible strategy to attenuate the issue that the rank of the shared bases is too large, since this rank becomes smaller at each level of the hierarchy. However, the fundamental issue persists, because it is related to the use of weak admissibility: for many problems, the matrix formed of the entire off-diagonal part of any block-row or block-column has quite a large rank.

Indeed, geometrically, the shared basis of block-row number i must represent the interaction of a domain σ_i with all other domains, near and far, and in every direction.



(a) Weak admissibility. (b) Strong admissibility. (c) Multiple bases (and strong admissibility).

Fig. 7.1: Illustration of the geometric interpretation of different approaches to compress a given block-row corresponding to subdomain σ_i. The gray area of the domain is compressed: a shared basis of the corresponding part of the block-row is computed. In the right figure, the light and dark gray areas use different shared bases.

The interaction between two widely separated domains is weak, and only requires a few of the shared basis vectors. The interaction between two nearby domains is strong, and requires a good number of the shared basis vectors. The two interactions between a domain and two others, the same distance away from the first, but located on opposite sides of each other, may not use many shared basis vectors in common.

This geometric intuition motivates alternative approaches that only force the shared basis structure on a subset of the blocks of each block-row and block-column. In the next two sections, we investigate two such approaches, whose geometric interpretation is illustrated in Figure 7.1.

- The first approach (section 8) introduces strong admissibility in the BLR^2 representation. Near interactions are not compressed, that is, blocks whose rank is too large are not included in the shared bases (Figure 7.1b). The BLR^2 format then essentially reduces to a flat (non-hierarchical) version of $\mathcal{H}^2$ matrices.
- The second approach (section 9) employs multiple shared bases per block-row/block-column: blocks are grouped together in subsets who each use a different shared basis. Geometrically, this means that interactions with domains in opposite directions are compressed separately (Figure 7.1c).

Both approaches however share one drawback. By losing the weakly admissible structure, factorizing the matrix becomes a significantly more complicated endeavor. Indeed, the good properties that allowed for an elegant LU factorization algorithm (Algorithm 4.1) are lost. The main issue is that blocks belonging to a given shared basis will receive updates from blocks outside this shared basis (either because they are in BLR form, or because they belong to a different shared basis). This leads to rank growth in the shared bases, and requires expensive recompressions to control the growth. Overall, we therefore envision BLR^2 being more effective at reducing storage and operations whose cost is proportional to the storage, such as matrix–vector products or triangular solves. This makes applications where such operations are the bottleneck an ideal target: for example, data sparse matrices can be used in the context of iterative solvers [27], or in the context of direct solvers with many right-hand sides [4, 31]. In the next two sections, we therefore focus on the storage costs of the methods.

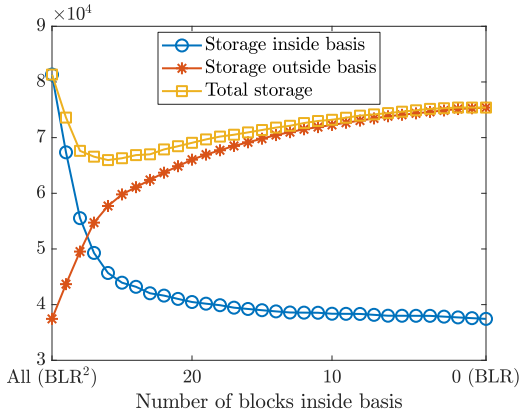


Fig. 8.1: Storage for block-row number 10 of matrix P64, depending on the number blocks inside the shared basis (under BLR^2 form) and outside of it (under BLR form). We have used $\varepsilon = 10^{-8}$.

8. Improving BLR^2 matrices with strong admissibility. To illustrate why strong admissibility may help improving the BLR^2 compression, we begin with an example. We focus on a given block-row of matrix P64 (specifically, block-row number 10), which is composed of 31 off-diagonal blocks. If we try to compute a left basis for the entire block-row, as we do in the weakly admissible case, we obtain a shared basis of rank $s = 104$. Representing this block-row in BLR^2 form then requires $81,328$ entries, which is more than the $75,400$ entries required by the BLR form. Clearly, this is due to s being too large.

We now sort the 31 blocks by decreasing rank, and start removing them one by one outside the shared basis. For example, the first block has rank 48. If we take it out, that is, if we compute a BLR^2 representation of the 30 remaining blocks, the rank of the shared bases decreases from 104 to 80. The number of entries required to store the entire block-row (30 blocks in BLR^2 form and 1 block in BLR form) is now only $73,600$. Figure 8.1 shows the evolution of the storage as we continue taking more blocks out of the shared basis. The optimal storage of $65,966$ entries is attained when the first 4 blocks of highest rank are stored in BLR form, and the remaining 27 are under BLR^2 form. This reduces the rank of the shared basis to $s = 31$.

We can generalize this idea to the entire matrix by setting some parameter $r_{\max}$ that defines the maximal rank for a block to be included in the shared bases. Any block of rank larger than $r_{\max}$ thus remains in BLR form.

We analyze in Figure 8.2 the storage for the strongly admissible BLR^2 format depending on the choice of $r_{\max}$. Standard BLR corresponds to $r_{\max} = 0$ (none of the blocks are in the shared bases) and weakly admissible BLR^2 corresponds to $r_{\max} = \infty$ (all blocks, regardless of their rank, are included in the shared bases). The figure shows that setting $r_{\max}$ to in-between values can lead to significant storage savings, reducing the BLR storage by at least 15% and up to 30% overall.

These results indicate that strong admissibility, by significantly reducing the rank of the shared bases, leads to a new BLR^2 format that requires less storage than BLR.

9. Improving BLR^2 matrices with multiple shared bases. The second approach that we explore to reduce the rank of the shared bases in the BLR^2 format

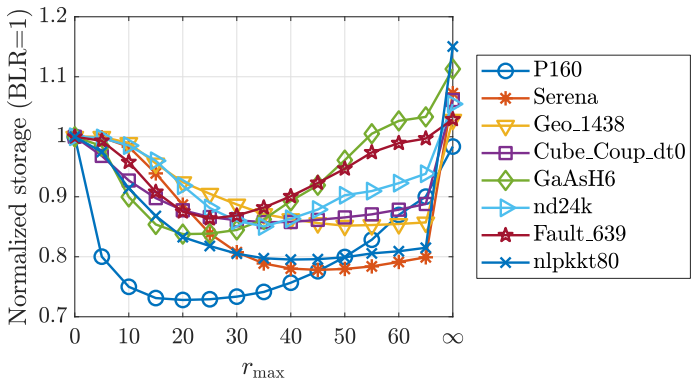


Fig. 8.2: Storage for strongly admissible BLR^2 format depending on the maximal rank value $r_{\max}$ allowed in the shared bases. Storage is normalized with respect to BLR format ($r_{\max} = 0$). Weakly admissible BLR^2 corresponds to $r_{\max} = \infty$. We have used $\varepsilon = 10^{-8}$.

is the use of multiple shared bases for a given block-row/block-column. That is, we replace the representation (2.3) by

$$H_{ij} = X_i^{(q)} C_{ij} Y_j^{(q)}, \tag{9.1}$$

for a given base number q. This approach is based on the idea that, while an entire block-row may not possess a strong shared rank structure, carefully selected parts of it may.

Interpreting this approach geometrically brings further insight. While any given subdomain (corresponding to some block-row) usually has a strong interaction with the rest of the domain (the off-diagonal part of the block-row has high rank), it may have much weaker interaction with a localized part of the domain. Note that this observation remains valid with strong admissibility: using a unique shared basis per block-row amounts to compress the interaction between a given subdomain and the rest of the domain, except for a ring around the subdomain (which is taken out based on *distance*). Instead, using multiple bases amounts to compressing different parts of the domain separately (based not only on distance but also *direction*). This geometric interpretation is illustrated in Figure 7.1.

How should one choose blocks that are grouped together and share the same basis? We distinguish two approaches.

- Regular $t \times t$ block-groups: we group t^2 adjacent blocks forming a $t \times t$ block-group pattern.
- HODLR groups with ℓ levels: similarly to the HODLR representation, we partition the matrix into

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

(where the A_{ij} are of the same size) and group all blocks belonging to A_{12} and A_{21} together. We then apply this approach recursively to A_{11} and A_{22}, up to ℓ levels. On the last level, off-diagonal blocks of A_{11} and A_{22} are also grouped together.

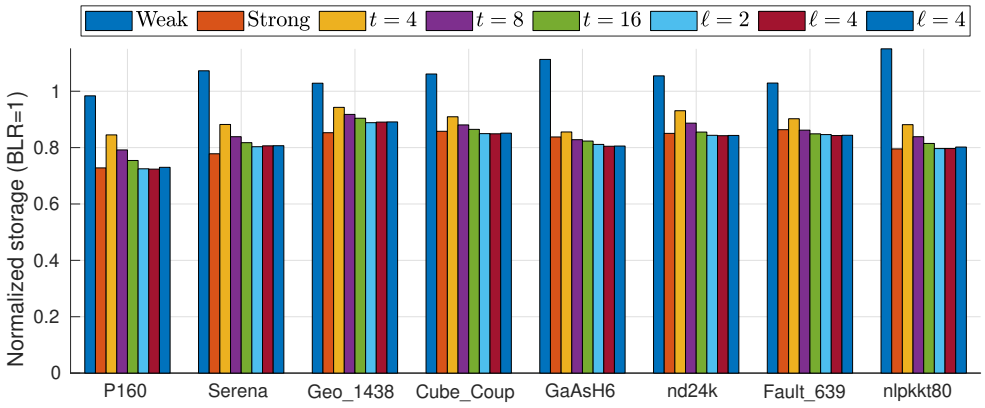


Fig. 9.1: Storage for various BLR^2 variants, normalized by that of BLR. For each matrix, the first two bars correspond to BLR^2 with weak and strong admissibility, respectively. The remaining six bars correspond to the use of multiple shared bases with different strategies to group blocks that share a basis together: either $t \times t$ groups or HODLR groups with ℓ levels. We have used $\varepsilon = 10^{-8}$. The strongly admissible variant (second bar) uses the optimal choice of $r_{\max}$ identified in Figure 8.2 The multiple base variants also use strong admissibility, with fixed $r_{\max} = 30$.

We compare these two approaches with standard BLR and both weakly and strongly admissible BLR^2 in Figure 9.1. Both multiple base approaches achieve significant storage gains with respect to both the weakly admissible BLR^2 format and the BLR format, up to 30% reduction overall. While both approaches are therefore useful to improve BLR^2 compression, the HODLR grouping strategy consistently outperforms the regular block grouping one. The storage for multiple base BLR^2 is in all cases very similar to that of the strongly admissible BLR^2 approach described in section 8.

A connection can be drawn between the variant using regular $t \times t$ block-groups and the lattice-$\mathcal{H}$ [34] and MBLR [6] formats. These two formats also partition the matrix using a regular block partitioning. Then, for each block (or lattice), the lattice-$\mathcal{H}$ format uses the $\mathcal{H}$ format and MBLR recursively uses the (M)BLR format. The approach discussed in this section uses the BLR^2 format, and others are possible. In fact, an interesting direction for future research is to investigate a general data sparse format that partitions the matrix into blocks each using a potentially different representation (BLR, BLR^2, $\mathcal{H}$, $\mathcal{H}^2$, ...).

10. Conclusions. Can a nested bases-like structure be used within flat data sparse matrix representations, such as the block low-rank (BLR) format? We have attempted to answer this question. Unlike hierarchical formats, the low-rank bases cannot be nested, since there is only one level; however, they can still be shared across the block-rows and block-columns. We have proposed such a format, that we call BLR^2 in reference to the $\mathcal{H}^2$ representation of which it is essentially a flat version. We have described algorithms to build BLR^2 matrices, to compute their LU factorization, and to use them for the solution of linear systems.

We have carried out costs analyses, both in an asymptotic and non-asymptotic sense, that show that, if the shared bases are of sufficiently low rank, the BLR^2

format is more compact than the standard BLR one and should allow for significant storage gains. However, with weak admissibility (that is, all off-diagonal blocks are compressed and included in the shared bases), our experiments on a limited set of problems suggest that the rank of the shared bases might be too large in practice to lead to significant gains.

Motivated by this observation, we have proposed two alternative approaches to reduce the rank of the shared bases. The first approach switches from weak to strong admissibility, leaving blocks of too large rank outside the shared bases. The second approach uses multiple shared bases for each block-row and block-column: only subsets of blocks, grouped together based on different strategies, share their bases. With either of these two approaches, we have shown that the BLR^2 format can achieve significant storage gains with respect to the BLR format, up to a 30% reduction.

In future work, we will explore the application of these techniques to the iterative solution of linear systems, which relies on matrix–vector products, and whose cost could therefore be reduced by using the BLR^2 representation.

REFERENCES

[1] K. Akbudak, H. Ltaief, A. Mikhalev, and D. Keyes, *Tile low rank Cholesky factorization for climate/weather modeling applications on manycore architectures*, in High Performance Computing, J. M. Kunkel, R. Yokota, P. Balaji, and D. Keyes, eds., Cham, 2017, Springer International Publishing, pp. 22–40.

[2] N. A. Al-Harthi, R. M. Alomairy, K. Akbudak, R. Chen, H. Ltaief, H. Bagci, and D. E. Keyes, *Solving acoustic boundary integral equations using high performance tile low-rank lu factorization*, High Performance Computing, (2020), https://doi.org/10.1007/978-3-030-50743-5_11, http://hdl.handle.net/10754/663212.

[3] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker, *Improving multifrontal methods by means of block low-rank representations*, SIAM J. Sci. Comput., 37 (2015), pp. A1451–A1474, https://doi.org/10.1137/120903476.

[4] P. R. Amestoy, R. Brossier, A. Buttari, J.-Y. L'Excellent, T. Mary, L. Métivier, A. Miniussi, and S. Operto, *Fast 3D frequency-domain full waveform inversion with a parallel Block Low-Rank multifrontal direct solver: application to OBC data from the North Sea*, Geophysics, 81 (2016), pp. R363–R383, https://doi.org/10.1190/geo2016-0052.1, http://personalpages.manchester.ac.uk/staff/theo.mary/doc/GEO16.pdf.

[5] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, *On the complexity of the Block Low-Rank multifrontal factorization*, SIAM J. Sci. Comput., 39 (2017), pp. A1710–A1740, https://doi.org/10.1137/16M1077192.

[6] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, *Bridging the gap between flat and hierarchical low-rank matrix formats: The multilevel block low-rank format*, SIAM J. Sci. Comput., 41 (2019), pp. A1414–A1442, https://doi.org/10.1137/18M1182760, https://doi.org/10.1137/18M1182760.

[7] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, *Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures*, ACM Trans. Math. Software, 45 (2019), pp. 2:1–2:26.

[8] A. Aminfar, S. Ambikasaran, and E. Darve, *A fast block low-rank dense solver with applications to finite-element matrices*, Journal of Computational Physics, 304 (2016), pp. 170–188.

[9] O. Beaumont, L. Eyraud-Dubois, and M. Verite, *2D Static Resource Allocation for Compressed Linear Algebra and Communication Constraints*. working paper or preprint, July 2020, https://hal.inria.fr/hal-02900244.

[10] M. Bebendorf, *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, vol. 63 of Lecture Notes in Computational Science and Engineering (LNCSE), Springer-Verlag, 2008.

[11] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Engineering analysis with boundary elements, 27 (2003), pp. 405–422, https://doi.org/10.1016/S0955-7997(02)00152-2.

[12] Q. Cao, Y. Pei, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, *Extreme-scale task-based cholesky factorization toward climate and weather pre-*

diction applications, in Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '20, New York, NY, USA, 2020, Association for Computing Machinery, https://doi.org/10.1145/3394277.3401846, https://doi.org/10.1145/3394277.3401846.

[13] Q. Cao, Y. Pei, T. Herault, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, *Performance analysis of tile low-rank cholesky factorization using parsec instrumentation tools*, in 2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools), 2019, pp. 25–32.

[14] R. Carratalá-Sáez, S. Christophersen, J. I. Aliaga, V. Beltran, S. Börm, and E. S. Quintana-Ortí, *Exploiting nested task-parallelism in the h-lu factorization*, Journal of Computational Science, 33 (2019), pp. 20–33.

[15] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ulv decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[16] A. Charara, D. Keyes, and H. Ltaief, *Tile low-rank GEMM using batched operations on GPUs*, in Euro-Par 2018: Parallel Processing, M. Aldinucci, L. Padovani, and M. Torquati, eds., Cham, 2018, Springer International Publishing, pp. 811–825, https://doi.org/10.1007/978-3-319-96983-1_57.

[17] T. A. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1–1:25, https://doi.org/10.1145/2049662.2049663, http://doi.acm.org/10.1145/2049662.2049663.

[18] A. Gillman, P. Young, and P.-G. Martinsson, *A direct solver with $\mathcal{O}(N)$ complexity for integral equations on one-dimensional domains*, Frontiers of Mathematics in China, 7 (2012), pp. 217–247.

[19] L. Grasedyck, R. Kriemann, and S. Le Borne, *Domain decomposition based $\mathcal{H}$-lu preconditioning*, Numerische Mathematik, 112 (2009), pp. 565–600.

[20] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108, https://doi.org/http://dx.doi.org/10.1007/s006070050015.

[21] W. Hackbusch, *Hierarchical matrices : algorithms and analysis*, vol. 49 of Springer series in computational mathematics, Springer, Berlin, 2015, https://doi.org/10.1007/978-3-662-47324-5.

[22] N. J. Higham and T. Mary, *Solving block low-rank linear systems by LU factorization is numerically stable*, MIMS EPrint 2019.15, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Sept. 2021, http://eprints.maths.manchester.ac.uk/2752/. To appear in IMA. J. Numer. Anal.

[23] A. Ida, H. Nakashima, and M. Kawai, *Parallel hierarchical matrices with block low-rank representation on distributed memory computer systems*, in Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, HPC Asia 2018, New York, NY, USA, 2018, ACM, pp. 232–240, https://doi.org/10.1145/3149457.3149477, http://doi.acm.org/10.1145/3149457.3149477.

[24] C.-P. Jeannerod, T. Mary, C. Pernet, and D. Roche, *Exploiting fast matrix arithmetic in block low-rank factorizations*, SIAM J. Matrix Anal. Appl., (2019), http://personalpages.manchester.ac.uk/staff/theo.mary/doc/FMA_BLR.pdf. Submitted.

[25] M. Ma and D. Jiao, *Accuracy directly controlled fast direct solution of general $\mathcal{H}^2$-matrices and its application to solving electrodynamic volume integral equations*, IEEE Transactions on Microwave Theory and Techniques, 66 (2018), pp. 35–48, https://doi.org/10.1109/TMTT.2017.2734090.

[26] T. Mary, *Block Low-Rank Multifrontal Solvers: Complexity, Performance, and Scalability*, PhD thesis, Université de Toulouse, Toulouse, France, Nov. 2017, http://personalpages.manchester.ac.uk/staff/theo.mary/doc/thesis.pdf.

[27] S. Ohshima, I. Yamazaki, A. Ida, and R. Yokota, *Optimization of hierarchical matrix computation on GPU*, in Supercomputing Frontiers, R. Yokota and W. Wu, eds., Springer International Publishing, Cham, 2018, pp. 274–292, https://doi.org/10.1007/978-3-319-69953-0_16.

[28] G. Pichon, *On the use of low-rank arithmetic to reduce the complexity of parallel sparse linear solvers based on direct factorization techniques*, Ph.D. thesis, Université de Bordeaux, Nov. 2018, https://hal.inria.fr/tel-01953908/.

[29] G. Pichon, E. Darve, M. Faverge, P. Ramet, and J. Roman, *Sparse supernodal solver using block low-rank compression: Design, performance and analysis*, Journal of Computational Science, 27 (2018), pp. 255–270, https://doi.org/https://doi.org/10.1016/j.jocs.2018.06.007, http://www.sciencedirect.com/science/article/pii/S1877750317314497.

[30] M. Sergent, D. Goudin, S. Thibault, and O. Aumage, *Controlling the memory subscription of distributed applications with a task-based runtime system*, in 2016 IEEE International

Parallel and Distributed Processing Symposium Workshops (IPDPSW), vol. 00, May 2016, pp. 318–327, https://doi.org/10.1109/IPDPSW.2016.105.

[31] D. V. Shantsev, P. Jaysaval, S. de la Kethulle de Ryhove, P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary, *Large-scale 3D EM modeling with a Block Low-Rank multifrontal direct solver*, Geophys. J. Int., 209 (2017), pp. 1558–1571, https://doi.org/10.1093/gji/ggx106, http://personalpages.manchester.ac.uk/staff/theo.mary/doc/GJI17.pdf.

[32] C. Weisbecker, *Improving multifrontal solvers by means of algebraic block low-rank representations*, PhD thesis, Institut National Polytechnique de Toulouse, Oct. 2013, http://ethesis.inp-toulouse.fr/archive/00002471/.

[33] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Fast algorithms for hierarchically semiseparable matrices*, Numerical Linear Algebra with Applications, 17 (2010), pp. 953–976.

[34] I. Yamazaki, A. Ida, R. Yokota, and J. Dongarra, *Distributed-memory lattice $\mathcal{H}$-matrix factorization*, Int. J. High Performance Computing Applications, 33 (2019), pp. 1046–1063, https://doi.org/10.1177/1094342019861139.