

IMPROVING THE COMPLEXITY OF BLOCK LOW-RANK
FACTORIZATIONS WITH FAST MATRIX ARITHMETIC*CLAUDE-PIERRE JEANNEROD†, THEO MARY‡, CLÉMENT PERNET§, AND
DANIEL S. ROCHE¶

Abstract. We consider the LU factorization of an $n \times n$ matrix A represented as a block low-rank (BLR) matrix: most of its off-diagonal blocks are approximated by matrices of small rank r , which reduces the asymptotic complexity of computing the LU factorization of A down to $\mathcal{O}(n^2r)$. Even though lower complexities can be achieved with hierarchical matrices, the BLR format allows for a very simple and efficient implementation. In this article, our aim is to further reduce the BLR complexity without losing its nonhierarchical nature by exploiting fast matrix arithmetic, that is, the ability to multiply two $n \times n$ full-rank matrices together for $\mathcal{O}(n^\omega)$ flops, where $\omega < 3$. This is not straightforward: simply accelerating the intermediate operations performed in the standard BLR factorization algorithm does not suffice to reduce the quadratic complexity in n , because these operations are performed on matrices whose size is too small. To overcome this obstacle, we devise a new BLR factorization algorithm that, by recasting the operations so as to work on intermediate matrices of larger size, can exploit more efficiently fast matrix arithmetic. This new algorithm achieves an asymptotic complexity of $\mathcal{O}(n^{(\omega+1)/2}r^{(\omega-1)/2})$, which represents an asymptotic improvement compared with the standard BLR factorization as soon as $\omega < 3$. In particular, for Strassen's algorithm, $\omega \approx 2.81$ yields a complexity of $\mathcal{O}(n^{1.904}r^{0.904})$. Our numerical experiments are in good agreement with this analysis.

Key words. fast matrix arithmetic, block low-rank matrices, linear algebra

AMS subject classifications. 15A23, 65F05

DOI. 10.1137/19M1255628

1. Introduction. Efficiently computing the solution of linear systems of equations is a fundamental part of many scientific computing applications. Let us refer to such a system as

$$(1.1) \quad Ax = y,$$

where $A \in \mathbb{R}^{n \times n}$ is a dense matrix, $x \in \mathbb{R}^n$ is the unknown vector, and $y \in \mathbb{R}^n$ is the right-hand side vector. This article focuses on solving (1.1) with direct approaches based on Gaussian elimination, which seek to factorize the matrix under the form $A = LU$. Throughout the article, we assume that A is generic enough so that its LU factorization exists and that numerical pivoting is not needed for stability. The asymptotic complexity of the traditional LU factorization algorithm is $\mathcal{O}(n^3)$ floating-point

*Received by the editors April 10, 2019; accepted for publication (in revised form) by M. Tůma October 14, 2019; published electronically November 26, 2019.

<https://doi.org/10.1137/19M1255628>

Funding: The work of the authors was supported by the Engineering and Physical Sciences Research Council grant EP/P020720/1 and the OpenDreamKit Horizon 2020 European Research Infrastructures grant 676541. The opinions and views expressed in this publication are those of the authors and not necessarily those of the funding bodies.

†Univ Lyon, Inria, CNRS, ENS de Lyon, Université Claude Bernard Lyon 1, LIP UMR 5668, F-69007 LYON, France (claude-pierre.jeannerod@inria.fr).

‡School of Mathematics, The University of Manchester, Manchester M13 9PL, UK (theo.mary@manchester.ac.uk).

§Univ. Grenoble Alpes, CNRS, Grenoble INP (Institute of Engineering Univ. Grenoble Alpes), LJL, 38000 Grenoble, France (clement.pernet@univ-grenoble-alpes.fr).

¶United States Naval Academy, Annapolis, MD (roche@usna.edu).

operations (flops), which makes computing the LU factorization the computational bottleneck of solving (1.1).

In many applications (e.g., Schur complements arising from the discretization of elliptic partial differential equations), the matrix A has been shown to have a low-rank property: most of its off-diagonal blocks can be approximated by low-rank matrices [9]. This property can be exploited to reduce the asymptotic complexity of LU factorization. A particularly simple low-rank matrix format is a flat, two-dimensional (2D) blocking of the matrix known as block low-rank (BLR) format [2]. As proved in [3], the asymptotic complexity of the LU factorization of a BLR matrix with off-diagonal blocks of rank r is reduced down to $\mathcal{O}(n^2r)$. More complex formats based on a hierarchical partitioning of the matrix have been proposed to achieve a (quasi-)linear complexity in n . For example, the \mathcal{H} format [21] leads to a factorization with complexity $\mathcal{O}(nr^2 \log^2 \frac{n}{r})$. The logarithmic factor can be avoided by using a so-called nested basis structure, which corresponds to the \mathcal{H}^2 and HSS formats [21, 39].

On the other hand, LU factorization algorithms based on fast matrix multiplication have been devised to reduce the $\mathcal{O}(n^3)$ complexity down to $\mathcal{O}(n^\omega)$ with $\omega < 3$. For example, Strassen's algorithm [38] leads to $\omega = \log_2 7 \approx 2.81$, while the algorithm currently achieving the lowest complexity is that of Le Gall [30] ($\omega \approx 2.37$).

Even though the BLR format achieves a higher theoretical complexity than hierarchical formats, its simplicity and flexibility make it easy to use in the context of a general purpose, algebraic solver [6, 33, 4, 36]. Due to its nonhierarchical nature, the BLR format is particularly efficient on parallel computers [6, 35, 14, 1]. An important question is therefore how to further reduce the $\mathcal{O}(n^2r)$ BLR complexity without sacrificing its simplicity and parallelism. For example, a possible idea is to extend the BLR format to use a small number of levels of hierarchy (MBLR format [5]), therefore losing as little parallelism as possible while achieving an asymptotic complexity reduction. In this work, we are rather interested in reducing the BLR factorization complexity *without changing its nonhierarchical nature*. To this aim, we seek to exploit fast matrix arithmetic within the BLR factorization. We investigate whether this is possible and what asymptotic gain can be expected.

Pernet and Storjohann [32] investigate the use of fast matrix arithmetic within the weakly admissible \mathcal{H} format (commonly referred to as HODLR, or RRR in their paper), analyzing various fast \mathcal{H} -matrix operations, which include the solution of linear systems (via matrix inversion). Since the standard \mathcal{H} complexity is already linear in n , their goal is to reduce the asymptotic dependence on r . They show that exploiting fast matrix arithmetic to accelerate each intermediate product in the \mathcal{H} factorization can reduce its asymptotic complexity to $\mathcal{O}(nr^{\omega-1} \log^2 \frac{n}{r})$.

In the BLR case, however, the complexity $\mathcal{O}(n^2r)$ is linear in r ; it is its quadratic dependence on n that we seek to reduce. As we will show, accelerating each intermediate product separately as in [32] does not yield a satisfying result in the BLR case: it does not reduce the asymptotic dependence on n , but only on r . To overcome this obstacle, we devise a new BLR factorization algorithm that successfully exploits fast matrix arithmetic to reduce the asymptotic complexity of the BLR factorization. The main idea is to recast the operations so as to work on intermediate matrices of larger size, which allows for a more efficient use of fast matrix arithmetic.

The rest of this article is organized as follows. We first provide some preliminary material in section 2, which covers some definitions and tools that we use in the following sections. Then, we describe in section 3 the standard BLR factorization algorithm and show that by incorporating fast matrix arithmetic its asymptotic complexity is only reduced from $\mathcal{O}(n^2r)$ to $\mathcal{O}(n^2r^{\omega-2})$. To achieve a better asymp-

totic reduction, we propose a new BLR factorization algorithm in section 4 that uses fast matrix arithmetic more efficiently; we prove that its asymptotic complexity is $\mathcal{O}(n^{(\omega+1)/2}r^{(\omega-1)/2})$, which is subquadratic in n as soon as $\omega < 3$. We support this theoretical complexity analysis with some numerical experiments in section 5. Our concluding remarks are provided in section 6.

Throughout this article, the rank of a matrix may refer either to its exact rank or its numerical rank at some given accuracy ε , depending on the application at hand. The numerical rank of matrix A at accuracy ε is defined as the smallest integer r_ε such that there exists a rank- r_ε matrix \tilde{A} such that $\|A - \tilde{A}\| \leq \varepsilon \|A\|$.

Throughout the article, n, r, b, p are positive integers such that $r \leq b \leq n$ and $p = n/b$. Specifically, n is the number of rows and columns of A , r is the maximal value of the rank of the low-rank blocks of the LU factors of A , b is the block size, and p is the number of blocks per row and column. We can think of n and r as being fixed, while b (and thus also p) can be adjusted. Note that, even though we assume that the matrices are over the real field \mathbb{R} , our results carry over to the complex field \mathbb{C} or to any other arbitrary field.

2. Preliminaries.

2.1. Definitions: FR, LR, and BLR matrices. We shall denote any $b \times b$ matrix as full-rank (FR) if it has rank larger than r and as low-rank (LR) otherwise. Note that a matrix of rank k such that $r < k < b$ is therefore considered FR even though it does not have full rank. This is because it is treated *as if* it had full-rank, and thus, as far as complexity is concerned, we need not distinguish rank- b and rank- k matrices when $r < k < b$. A LR matrix can be represented in the compressed form XY^T for some $X, Y \in \mathbb{R}^{b \times r}$, and can thus be stored using only $2br$ entries (instead of the usual b^2); we refer to this quantity as the *size* of the matrix.

Let $A \in \mathbb{R}^{n \times n}$ be partitioned into p^2 blocks A_{ij} of dimensions $b \times b$. Then A is said to be a *BLR matrix* if all its blocks are LR, except for $\mathcal{O}(1)$ FR blocks in each block-row and block-column. (Formally, we should say that A is an (n, b, r) -BLR matrix, but this will in general be clear from the context.) The LR blocks are given in their compressed form $A_{ij} = X_{ij}Y_{ij}^T$.

A BLR matrix A therefore consists of $\mathcal{O}(p)$ FR blocks and no more than p^2 LR blocks. Hence its size is bounded by $\mathcal{O}(pb^2 + p^2br)$, that is, recalling that $p = n/b$,

$$\text{size}(A) = \mathcal{O}(nb + n^2r/b).$$

For $1 \leq \beta \leq n$, the function $f_{n,r}(\beta) := n\beta + n^2r/\beta$ is minimum when $f'_{n,r}(\beta) = n - n^2r/\beta^2 = 0$, or, equivalently, when $\beta^2 = nr$. Therefore, a choice of block size which minimizes the size bound of BLR matrices is about \sqrt{nr} :

$$b = \Theta(\sqrt{nr}) \quad \Rightarrow \quad \text{size}(A) = \mathcal{O}(n^{3/2}r^{1/2}).$$

BLR matrices typically arise in applications such as the resolution of discretized partial differential equations. In this context, a block represents the interaction between two physical subdomains σ and τ : diagonal blocks correspond to self interactions and are FR; off-diagonal blocks may be FR or LR depending on whether they correspond to near-field or far-field interactions (that is, on how far away the subdomains are from each other). This is formalized by the so-called admissibility condition [21, section 4.2.3]:

$$(2.1) \quad \text{The block } \sigma \times \tau \text{ is admissible} \quad \Leftrightarrow \quad \eta \text{ dist}(\sigma, \tau) \geq \min(\text{diam}(\sigma), \text{diam}(\tau)),$$

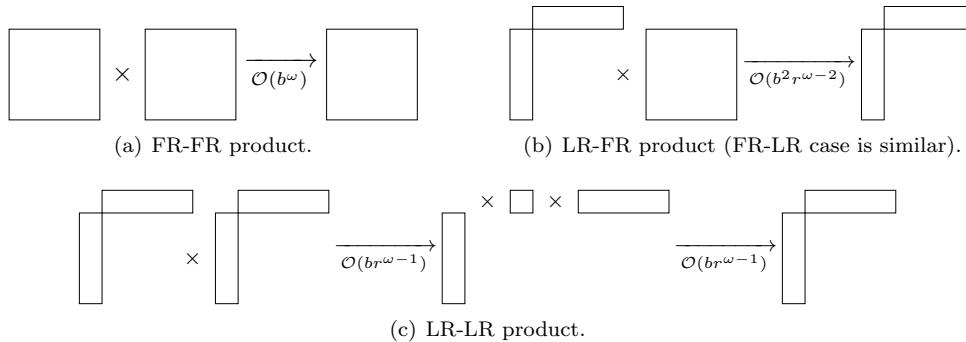


FIG. 1. *Different types of product with FR and LR blocks.*

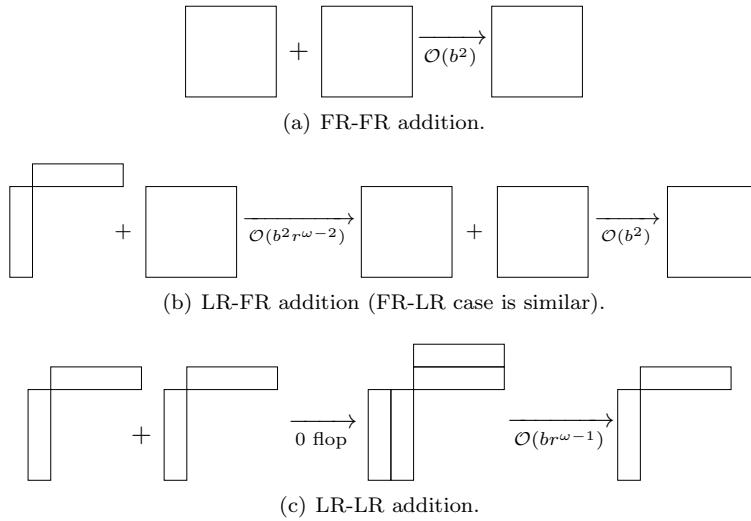
where $\eta > 0$ is some fixed parameter and where an ‘‘admissible’’ block means it can be represented by an LR matrix. A consequence of this geometric admissibility definition is that the union of several admissible blocks remains admissible. In algebraic terms, this means that independent LR blocks can be agglomerated in a single block that is also LR. Throughout the paper, we assume this algebraic property to hold. We point out that a sufficient (though not necessary) condition for this assumption to hold is that the matrix is quasiseparable [19].

2.2. Fast matrix arithmetic with FR and LR blocks. Figure 1 illustrates the four possible matrix products between FR and LR blocks. The product of two FR blocks yields in general an FR block and costs $\mathcal{O}(b^\omega)$ flops. The product of an FR and an LR block yields an LR block and costs $\mathcal{O}(b^2r^{\omega-2})$ flops. Finally, the product of two LR blocks yields an LR block and costs $\mathcal{O}(br^{\omega-1})$ flops.

Figure 2 illustrates the case of addition. Two FR blocks can be added together for $\mathcal{O}(b^2)$ flops, yielding an FR block. Adding an LR block with an FR block requires decompressing the LR one, that is, to convert it back to its dense representation, thus requiring $\mathcal{O}(b^2r^{\omega-2})$ flops and yielding an FR block. Finally, the addition of two LR blocks can be computed simply by concatenating their (transposed) low-rank factors X and Y together. However, the resulting representation is not LR anymore, since it is a rank- $2r$ representation. As mentioned in the previous section, in our context the result of the addition of two LR blocks is assumed to be an LR matrix; in this case, it must then be recompressed so as to compute a rank- r representation. This recompression can be done by any rank-revealing factorization.

Throughout the paper, we assume that the cost of computing a rank-revealing factorization of an $m \times n$ matrix of rank r is in $\mathcal{O}(mnr^{\omega-2})$ (which, for a $b \times r$ matrix, yields a cost in $\mathcal{O}(br^{\omega-1})$). Such algorithms are well-known in the context of exact linear algebra [37, 28]. On the other hand, to the best of our knowledge, there does not exist any article discussing this specific question in the numerical linear algebra case. This can, however, be achieved with randomized algorithms [22], which are able to compute LR representations using only matrix–matrix multiplication and non-rank-revealing QR factorization, of which fast versions are known (such as the one described in [16, section 4.1]). Since, to our knowledge, such an algorithm is not described anywhere in the literature, let us briefly sketch it. Given an $m \times n$ matrix A and a target rank r , a LR approximation of A can be computed in the following three steps [22]:

1. Sample the columns of A : $S \leftarrow A\Omega$, where Ω is an $n \times r$ random Gaussian matrix.

FIG. 2. *Different types of addition with FR and LR blocks.*TABLE 1
Cost of each type of product and addition.

	FR-FR	LR-FR	FR-LR	LR-LR
Products	$\mathcal{O}(b^\omega)$	$\mathcal{O}(b^2 r^{\omega-2})$	$\mathcal{O}(b^2 r^{\omega-2})$	$\mathcal{O}(br^{\omega-1})$
Additions	$\mathcal{O}(b^2)$	$\mathcal{O}(b^2 r^{\omega-2})$	$\mathcal{O}(b^2 r^{\omega-2})$	$\mathcal{O}(br^{\omega-1})$

2. Orthogonalize S , e.g., via QR factorization: $Q \leftarrow \text{orth}(S)$. Q approximates the range of A , that is, $A \approx QQ^T A$.
3. Compute $Y \leftarrow A^T Q$ and set $X \leftarrow Q$.

Then XY^T is an LR approximation of A . We do not delve into finer algorithmic details such as the choice of Ω , the oversampling, the choice of orthogonalization method, or how to adapt this fixed-rank algorithm to the fixed-accuracy case. We believe that further investigating fast numerical rank-revealing factorizations and comparing the performance and accuracy of different algorithms would be of interest, but is outside the scope of this article.

The costs of the different types of products and additions are summarized in Table 1.

3. The standard BLR factorization algorithm.

3.1. Algorithm description. We describe in Algorithm 3.1 the standard BLR factorization algorithm. It is simply a blocked LU algorithm where the off-diagonal blocks of the matrix may be either FR or LR.

The algorithm consists of three main kernels. First, at each step k , we compute the LU factorization of the diagonal block A_{kk} (Factor kernel, line 2). Then, we perform a triangular solve with these $L_k U_k$ factors (Solve kernel, line 4) for all $i > k$. If A_{ki} is an FR block, we compute $A_{ki} \leftarrow L_k^{-1} A_{ki}$. However, if A_{ki} is LR, we replace it by its LR representation $X_{ki} Y_{ki}^T$, which amounts to computing $X_{ki} \leftarrow L_k^{-1} X_{ki}$. Thus only X_{ki} needs to be updated. Similarly, $A_{ik} \leftarrow A_{ik} U_k^{-1}$ becomes $Y_{ik} \leftarrow U_k^{-T} Y_{ik}$ if A_{ik} is LR. Finally, we update the blocks A_{ij} of the trailing submatrix (Update kernel,

Algorithm 3.1 Standard BLR factorization.

Input: a BLR matrix $A \in \mathbb{R}^{n \times n}$ partitioned into p^2 blocks A_{ij} .
Output: A overwritten by its BLR LU factors.

```

1: for  $k = 1$  to  $p$  do
2:   Factor:  $A_{kk} = L_k U_k$  { $A_{kk}$  is FR}
3:   for  $i = k + 1$  to  $p$  do
4:     Solve:  $A_{ki} \leftarrow L_k^{-1} A_{ki}$  and  $A_{ik} \leftarrow A_{ik} U_k^{-1}$  { $A_{ki}$  and  $A_{ik}$  can be FR or LR}
5:   end for
6:   for  $i = k + 1$  to  $p$  do
7:     for  $j = k + 1$  to  $p$  do
8:       Update:  $A_{ij} \leftarrow A_{ij} - A_{ik} A_{kj}$  { $A_{ij}$ ,  $A_{ik}$ , and  $A_{kj}$  can be FR or LR}
9:     end for
10:   end for
11: end for
```

TABLE 2
Total number of each type of product and subtraction in the Update kernel.

	FR-FR	LR-FR	FR-LR	LR-LR
Products	$\mathcal{O}(p)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^3)$
Subtractions	$\mathcal{O}(p)$	$\mathcal{O}(p)$	$\mathcal{O}(p^2)$	$\mathcal{O}(p^3)$

line 8). We first compute the product $A_{ik} A_{kj}$, which takes one of the four possible forms described in Figure 1, and then subtract the result from A_{ij} , which can also take one of four possible forms, as described in Figure 2.

3.2. Complexity analysis with fast matrix arithmetic. We now derive the asymptotic complexity of Algorithm 3.1 by analyzing each kernel separately.

The Factor kernel costs $\mathcal{O}(b^\omega)$ per step, and thus in total

$$(3.1) \quad \text{cost(Factor)} = \mathcal{O}(pb^\omega).$$

The cost of the Solve kernel depends on whether A_{ik} is an FR or LR block: it is $\mathcal{O}(b^\omega)$ if A_{ik} is FR and $\mathcal{O}(b^2 r^{\omega-2})$ if A_{ik} is LR. Since there are only $\mathcal{O}(1)$ FR blocks per block-row and block-column, the total cost of the Solve kernel is

$$(3.2) \quad \text{cost(Solve)} = \mathcal{O}(pb^\omega + p^2 b^2 r^{\omega-2}).$$

Finally, the Update kernel consists of products and differences of matrices that can be either FR or LR. At each step k , since there are only $\mathcal{O}(1)$ FR blocks in each block-column and block-row, there are $\mathcal{O}(p^2)$ LR-LR products, $\mathcal{O}(p)$ LR-FR or FR-LR products, and $\mathcal{O}(1)$ FR-FR products. The FR-FR products yield an FR result and therefore lead to $\mathcal{O}(1)$ LR-FR and FR-FR subtractions. The other three types of products yield an LR result; since there are only $\mathcal{O}(p)$ FR blocks in the trailing submatrix, this leads to $\mathcal{O}(p)$ FR-LR subtractions and $\mathcal{O}(p^2)$ LR-LR subtractions. Each of these numbers must be multiplied by the number of steps p , yielding the numbers summarized in Table 2. Summing the cost of each type of operation, we obtain the total cost of the Update kernel:

$$(3.3) \quad \text{cost(Update)} = \mathcal{O}(pb^\omega + p^2 b^2 r^{\omega-2} + p^3 br^{\omega-1}).$$

Note that it is assumed that each intermediate LR-LR subtraction yields an intermediate LR matrix, that is, a matrix which can be recompressed into a rank- r representation. This is a very common and reasonable assumption, which can be theoretically proved for some application domains such as integral equations or discretized partial differential equations [9, 3]. Indeed, for these applications the BLR property is preserved through inversion and LU factorization, that is, the final ranks of the blocks of the LU factors are known to be at most r .

The following theorem summarizes this complexity analysis, which generalizes the result of [3] by incorporating the exponent ω of fast matrix arithmetic.

THEOREM 3.1 (complexity of fast BLR factorization: standard algorithm). *Let $A \in \mathbb{R}^{n \times n}$ be a BLR matrix partitioned into p^2 blocks of order b . For a choice of block size $b \in [\Theta(b_{\min}), \Theta(b_{\max})]$, where $b_{\min} = \sqrt{nr}$ and $b_{\max} = n^{1/(\omega-1)}r^{(\omega-2)/(\omega-1)}$, the asymptotic complexity of factorizing A via Algorithm 3.1 is*

$$\text{cost(Alg. 3.1)} = \mathcal{O}(n^2 r^{\omega-2}).$$

Proof. Summing the costs of the Factor, Solve, and Update kernels in (3.1), (3.2), and (3.3), we obtain

$$\mathcal{O}(pb^\omega + p^2 b^2 r^{\omega-2} + p^3 br^{\omega-1}) \subset \mathcal{O}(nb^{\omega-1} + n^2 r^{\omega-2} + n^3 r^{\omega-1}/b^2).$$

In the last sum, the second term dominates the first if $b \leq b_{\max} = n^{1/(\omega-1)}r^{(\omega-2)/(\omega-1)}$, whereas it dominates the third if $b \geq b_{\min} = \sqrt{nr}$. Since $b_{\min} \leq b_{\max}$ holds because $r \leq n$ and $\omega \leq 3$, the interval $[b_{\min}, b_{\max}]$ is not empty, and any choice of block size lying in $[\Theta(b_{\min}), \Theta(b_{\max})]$ makes the second term dominant and yields the result. \square

Exploiting fast matrix arithmetic can thus reduce the asymptotic complexity of the standard BLR factorization algorithm from $\mathcal{O}(n^2 r)$ to $\mathcal{O}(n^2 r^{\omega-2})$. With $\omega = 3$, we recover the $\mathcal{O}(n^2 r)$ complexity proved in [3]; with $\omega = 2$, we obtain $\mathcal{O}(n^2)$. This result is underwhelming, because we have only achieved an asymptotic reduction of the dependence on r , but not on n . Since the rank r is in general much smaller than n , this approach fails to significantly improve the complexity of the factorization.

Our aim is now to devise a new BLR factorization algorithm which can exploit more efficiently fast matrix arithmetic, and in particular which achieves a sub-quadratic complexity in n . More precisely, in the next section, we design a new algorithm whose complexity is $\mathcal{O}(n^{(\omega+1)/2} r^{(\omega-1)/2})$.

Note that if the compressed representation of the blocks of the input matrix is not readily available and has to be explicitly computed, there will be an overhead cost of $\mathcal{O}(b^2 r^{\omega-2})$ flops for each block and thus $\mathcal{O}(n^2 r^{\omega-2})$ flops in total. The asymptotic cost of the compression is therefore the same as that of the standard BLR factorization, which means that no further improvement can be expected. However, in many applications the compressed form of the blocks can be directly obtained or cheaply computed (for example, via a fast matrix-vector product [22]) and can therefore be considered to be of negligible cost.

4. A new BLR factorization algorithm. The main drawback of the standard BLR algorithm is that the products involving LR matrices are of too small granularity to allow for an efficient use of fast matrix arithmetic. The goal of this section is to reformulate the algorithm so that these products are performed on matrices of larger granularity. This new algorithm achieves an improved complexity of $\mathcal{O}(n^{(\omega+1)/2} r^{(\omega-1)/2})$ for a block size b in $\Theta(\sqrt{nr})$.

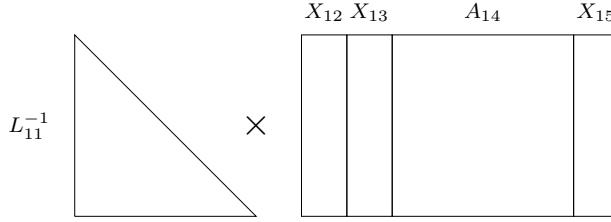


FIG. 3. Illustration of the accumulated Solve defined in (4.1), with $k = 1$ and $p = 5$, and assuming A_{12} , A_{13} , and A_{15} are LR whereas A_{14} is FR.

The key idea is to accumulate independent operations together to increase their granularity. We proceed kernel by kernel, proposing accumulation schemes for the solve, product, and subtraction operations. The first two are new, whereas the last has already been proposed in [6] under the name “low-rank updates accumulation” (LUA). The key difficulty is that we must combine all these accumulation schemes together in a single algorithm in order to achieve subquadratic complexity—in particular, we emphasize that the LUA strategy alone would not improve the complexity.

The cost of the Factor kernel is $O(pb^\omega)$ [13], which, for $b = \Theta(\sqrt{nr})$, gives the target complexity $\mathcal{O}(n^{(\omega+1)/2}r^{(\omega-1)/2})$.

We analyze the Solve kernel in section 4.1, and then consider the Update kernel in sections 4.2 (products) and 4.3 (subtractions). The new algorithm that results from the changes proposed in this section is described in Algorithm 4.1. We explain all the changes made to the standard BLR algorithm step by step in the next three subsections. Then, we analyze the asymptotic complexity of this new algorithm in section 4.4. Finally, we discuss some implementation aspects in section 4.5.

4.1. Accumulated solve kernel. At step k , we must perform the triangular solves $A_{ki} \leftarrow L_k^{-1}A_{ki}$ for all FR blocks A_{ki} and $X_{ki} \leftarrow L_k^{-1}X_{ki}$ for all LR blocks $A_{ki} = X_{ki}Y_{ki}^T$.

Rather than performing these solves separately, we can perform them all together on the accumulated matrices A_{ki} and X_{ki} . We compute

$$(4.1) \quad \begin{bmatrix} W_{k+1}^{(k)} & \cdots & W_p^{(k)} \end{bmatrix} \leftarrow L_k^{-1} \begin{bmatrix} W_{k+1}^{(k)} & \cdots & W_p^{(k)} \end{bmatrix},$$

where $W_i^{(k)} = A_{ki}$ if A_{ki} is FR, and $W_i^{(k)} = X_{ki}$ otherwise. Similarly, we also accumulate the matrices Y_{ik} to compute

$$(4.2) \quad \begin{bmatrix} V_{k+1}^{(k)} \\ \vdots \\ V_p^{(k)} \end{bmatrix} \leftarrow \begin{bmatrix} V_{k+1}^{(k)} \\ \vdots \\ V_p^{(k)} \end{bmatrix} U_k^{-1},$$

where $V_i^{(k)} = A_{ik}$ if A_{ik} is FR, and $V_i^{(k)} = Y_{ik}^T$ otherwise. This accumulated Solve is illustrated in Figure 3 and used at line 4 of Algorithm 4.1.

We now analyze the asymptotic cost of this new kernel. At each step k we must perform a triangular solve with a $b \times b$ block on a matrix of dimensions $b \times \mathcal{O}(b + pr)$, which therefore costs $\mathcal{O}(b^{\omega-1}(b + pr))$. Summing over the p steps, we obtain a total cost of

$$(4.3) \quad \text{cost(Accumulated Solve)} = \mathcal{O}(pb^\omega + p^2b^{\omega-1}r).$$

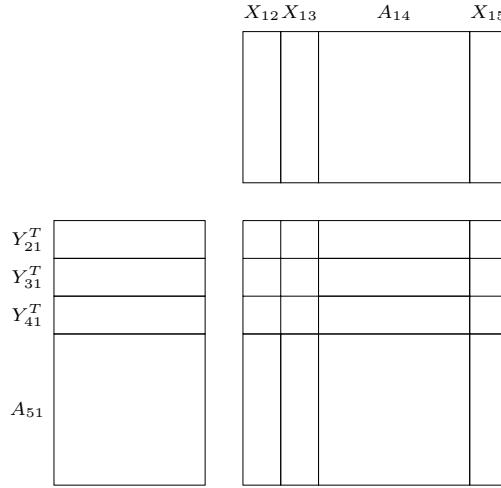


FIG. 4. Illustration of the accumulated product defined in (4.4), with $k = 1$ and $p = 5$, and assuming A_{14} and A_{51} are FR blocks.

Compared with the second term in the cost (3.2) of the standard Solve kernel, the second term in (4.3) is smaller by a factor $(b/r)^{3-\omega}$. Thus, if $\omega = 3$, the asymptotic cost of the Solve kernel is naturally unchanged, since the computations being performed have not been modified but only rearranged. In contrast, with $\omega < 3$, the accumulated Solve kernel leads to an improved asymptotic complexity thanks to the increased granularity of the operations.

4.2. Accumulated LR-FR, FR-LR, and LR-LR products. Now, we devise a new Update kernel. In this section, we first focus on the product operations $P_{ij}^{(k)} = A_{ik}A_{kj}$. We then turn to the subtractions $A_{ij} \leftarrow A_{ij} - P_{ij}^{(k)}$ in the next section.

At step k , we must compute the products $P_{ij}^{(k)} = A_{ik}A_{kj}$ for $i, j = k+1 : p$. Instead of computing each of these $\mathcal{O}(p^2)$ products separately, we can accumulate them as follows. We begin by computing

$$(4.4) \quad \begin{bmatrix} Z_{k+1,k+1}^{(k)} & \cdots & Z_{k+1,p}^{(k)} \\ \vdots & \ddots & \vdots \\ Z_{p,k+1}^{(k)} & \cdots & Z_{pp}^{(k)} \end{bmatrix} \leftarrow \begin{bmatrix} V_{k+1}^{(k)} \\ \vdots \\ V_p^{(k)} \end{bmatrix} \begin{bmatrix} W_{k+1}^{(k)} & \cdots & W_p^{(k)} \end{bmatrix},$$

where $V_i^{(k)} = A_{ik}$ if A_{ik} is FR, and $V_i^{(k)} = Y_{ik}^T$ otherwise, and $W_j^{(k)} = A_{kj}$ if A_{kj} is FR, and $W_j^{(k)} = X_{kj}$ otherwise. Then, we deduce $P_{ij}^{(k)}$ from $Z_{ij}^{(k)}$ as follows: if A_{ik} and A_{kj} are both FR, then $P_{ij}^{(k)}$ is FR and equal to $Z_{ij}^{(k)}$; otherwise, $P_{ij}^{(k)}$ is LR and its LR representation is given by $X_{ik}Z_{ij}^{(k)}$ if only A_{ik} is LR, by $Z_{ij}^{(k)}Y_{kj}^T$ if only A_{kj} is LR, and by $X_{ik}Z_{ij}^{(k)}Y_{kj}^T$ if both A_{ik} and A_{kj} are LR. This accumulated product is illustrated in Figure 4 and used at line 5 of Algorithm 4.1.

Let us now analyze the asymptotic cost of this accumulated product. Each product (4.4) has dimensions $\mathcal{O}(pr+b) \times b \times \mathcal{O}(pr+b)$, and therefore costs $\mathcal{O}((pr+b)^2 b^{\omega-2})$. Summing over the p steps, we obtain a total cost of

$$(4.5) \quad \text{cost(Accumulated Product)} = \mathcal{O}(p^3 b^{\omega-2} r^2 + p^2 b^{\omega-1} r + pb^\omega).$$

The cost of the FR-FR products remains unchanged and equal to $\mathcal{O}(pb^\omega)$. The cost of the FR-LR and LR-FR products is reduced from $\mathcal{O}(p^2b^2r^{\omega-2})$ to $\mathcal{O}(p^2b^{\omega-1}r)$, that is, by a factor $(b/r)^{3-\omega}$, just like for the new Solve cost (4.3). Similarly, the cost of the LR-LR products is reduced from $\mathcal{O}(p^3br^{\omega-1})$ to $\mathcal{O}(p^3b^{\omega-2}r^2)$, by again the same factor. Thus, just as for the new Solve kernel, if $\omega = 3$, the asymptotic cost of the products is unchanged, but with $\omega < 3$, an asymptotic improvement is achieved.

4.3. Accumulated FR-LR and LR-LR subtractions. From Tables 1 and 2, it is clear that the total cost of the FR-FR and LR-FR subtractions, namely, in $\mathcal{O}(pb^2r^{\omega-2})$, is small enough so that we do not have to worry about them. We can therefore simply perform these subtractions separately, as in the standard BLR algorithm. This corresponds to lines 10 (FR-FR) and 12 (LR-FR) of Algorithm 4.1.

We must, however, work on the FR-LR and LR-LR subtractions, which, in the standard BLR algorithm, cost $\mathcal{O}(p^2b^2r^{\omega-2})$ and $\mathcal{O}(p^3br^{\omega-1})$, respectively. In both situations, we have an LR matrix $P_{ij}^{(k)}$ which we seek to subtract to matrix A_{ij} , which is either FR or LR. We recall that $P_{ij}^{(k)}$ is of the form $X_{ik}Z_{ij}^{(k)}$, $Z_{ij}^{(k)}Y_{kj}^T$, or $X_{ik}Z_{ij}^{(k)}Y_{kj}^T$, depending on whether it is the result of an LR-FR, FR-LR, or LR-LR product, respectively. We define K_{ij}^{LR} as the set of indices $k < \min(i, j)$ such that $P_{ij}^{(k)}$ is the result of an LR-LR product and K_{ij}^{FR} as the set of indices $k < \min(i, j)$ such that $P_{ij}^{(k)}$ is the result of an LR-FR or FR-LR product.

Let us first analyze the cost associated with the products $P_{ij}^{(k)}$ such that $k \in K_{ij}^{FR}$. Since there can only be $\mathcal{O}(1)$ FR blocks on the i th block-row and j th block-column, the K_{ij}^{FR} set has only $\mathcal{O}(1)$ elements. Therefore, there are $\mathcal{O}(p)$ FR-LR and $\mathcal{O}(p^2)$ LR-LR subtractions associated with these products, which cost $\mathcal{O}(pb^2r^{\omega-2} + p^2br^{\omega-1})$ flops. These two terms are, respectively, dominated by $\mathcal{O}(pb^\omega)$ and $\mathcal{O}(p^2b^{\omega-1}r)$ (which appear in (4.5), for example); we conclude that the operations associated with the products $P_{ij}^{(k)}$ such that $k \in K_{ij}^{FR}$ are of negligible cost and can be performed separately as in the standard BLR algorithm. This corresponds to lines 14 through 25 of Algorithm 4.1.

We can therefore focus on the products $P_{ij}^{(k)}$ such that $k \in K_{ij}^{LR}$. The key to improving the asymptotic cost of the subtractions is to accumulate all these $P_{ij}^{(k)}$ matrices together over k , that is, to compute

$$P_{ij} = \sum_{k \in K_{ij}^{LR}} P_{ij}^{(k)}.$$

All $P_{ij}^{(k)}$ such that $k \in K_{ij}^{LR}$ are of the form $X_{ik}Z_{ij}^{(k)}Y_{kj}^T$. Consequently,

$$(4.6) \quad P_{ij} = [X_{i1} \ \cdots \ X_{im}] \begin{bmatrix} Z_{ij}^{(1)} \\ & \ddots \\ & & Z_{ij}^{(m)} \end{bmatrix} [Y_{1j} \ \cdots \ Y_{mj}]^T,$$

where we have assumed for readability that indices 1 and $m = \min(i, j) - 1$ belong to the set K_{ij}^{LR} .

This accumulated sum, which is illustrated in Figure 5, corresponds to the LUA strategy introduced in [6] to improve the performance of the BLR factorization by increasing the granularity of the BLAS operations.

Let us now analyze the cost of the subtractions associated with this accumulated sum of products P_{ij} . There are $\mathcal{O}(p)$ FR blocks A_{ij} from which we must subtract P_{ij}

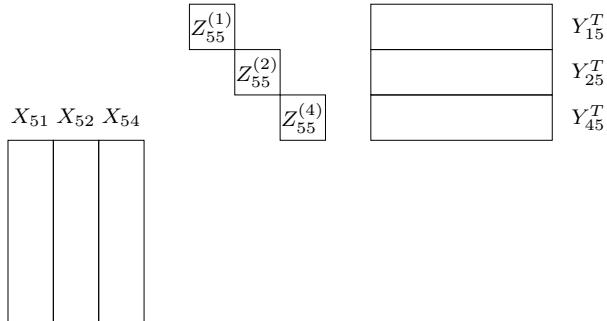


FIG. 5. Illustration of the accumulated sum of products $P_{ij}^{(k)}$ defined in (4.6), with $i = j = 5$, and assuming that $K_{ij}^{LR} = \{1, 2, 4\}$.

(FR-LR subtractions); they require us to decompress P_{ij} . Each decompression first requires multiplying the $r \times r$ matrices $Z_{ij}^{(k)}$ to either side of (4.6). Defining $q_{ij} \leq p$ as the cardinality of K_{ij}^{LR} , these products cost $\mathcal{O}(q_{ij}br^{\omega-1}) \subset \mathcal{O}(pbr^{\omega-1})$ flops. This yields an LR matrix P_{ij} expressed as a product of dimensions $b \times \mathcal{O}(pr) \times b$, and whose decompression costs $\mathcal{O}(pb^{\omega-1}r)$ flops if $b \leq pr$, and $\mathcal{O}(p^{\omega-2}b^2r^{\omega-2})$ otherwise. Since $r \leq b$, the cost $\mathcal{O}(pbr^{\omega-1})$ is dominated by the other costs. Finally, subtracting the decompressed P_{ij} to A_{ij} costs a negligible $\mathcal{O}(b^2)$ flops, and the total cost of the FR-LR subtractions is therefore

$$(4.7) \quad \text{cost(FR-LR subtractions)} = \mathcal{O}\left(\max(p^2b^{\omega-1}r, p^{\omega-1}b^2r^{\omega-2})\right).$$

When $\omega < 3$, we have thus reduced the standard cost $\mathcal{O}(p^2b^2r^{\omega-2})$ of the FR-LR subtractions. These accumulated FR-LR subtractions correspond to line 31 of Algorithm 4.1. Note that to ensure that we have accumulated all possible matrices to be subtracted, A_{ij} is only updated if it belongs to the next panel, that is, if $i = k + 1$ or $j = k + 1$. This is known as a Crout factorization [34].

The other $\mathcal{O}(p^2)$ A_{ij} blocks are LR and require an LR-LR subtraction with P_{ij} ; in this case, we must recompress P_{ij} . Because P_{ij} is an accumulated sum of products, many different recompression strategies are possible; several of them are described and analyzed in [31, Chapter 3]. A thorough discussion of these strategies is outside our scope; our focus is to derive an algorithm that efficiently exploits fast matrix arithmetic. There are two key ingredients to achieve a low recompression cost. The first is to exploit the fact that, by assumption (see section 2.1), the outer matrices in (4.6) are LR. The second ingredient is to compute the LR representation of these outer matrices incrementally rather than from scratch at each step. Indeed, assume that an LR basis VW^T of $[X_{i1} \dots X_{im}]$ is known, where $i > j$. Then, writing $m' = m + 1$ (note that we actually have $m' = j$), the matrix $[X_{i1} \dots X_{im'}]$ is given by

$$(4.8) \quad [V \ X_{im'}] \begin{bmatrix} W^T & 0 \\ 0 & I \end{bmatrix}$$

and an LR representation can therefore be obtained simply by recompressing $[V \ X_{im'}]$ for $\mathcal{O}(br^{\omega-1})$ flops. Overall the outer matrices in (4.6) can thus be recompressed for $\mathcal{O}(p^2br^{\omega-1})$ flops. Defining $VW^T = [X_{i1} \dots X_{im}]$ and $V'W'^T = [Y_{1j} \dots Y_{jm}]$ as their LR bases, it remains to multiply the middle $Z_{ij}^{(k)}$ matrices to either side of the

expression

$$P_{ij} = V W^T \begin{bmatrix} Z_{ij}^{(1)} \\ \ddots \\ Z_{ij}^{(m)} \end{bmatrix} V' W^T,$$

which costs $O(pr^\omega + br^{\omega-1})$ flops per block and thus $O(p^3r^\omega + p^2br^{\omega-1})$ in total. Having computed an LR representation of $P_{ij} = X_{P_{ij}}Y_{P_{ij}}^T$, we obtain a rank- $2r$ representation of $A_{ij} - P_{ij}$ by accumulating $X_{P_{ij}}$ with X_{ij} and $-Y_{P_{ij}}$ with Y_{ij} , as performed at line 33 of Algorithm 4.1. We must finally recompress this representation to obtain a rank- r representation (line 34), requiring again $\mathcal{O}(p^2br^{\omega-1})$ flops. In the end, the total cost of the LR-LR subtractions is

$$(4.9) \quad \text{cost(LR-LR subtractions)} = \mathcal{O}(p^3r^\omega + p^2br^{\omega-1}).$$

4.4. New algorithm and its complexity. Now that we have devised new kernels for each of the operations performed in the factorization, we can bring together all these improvements. Algorithm 4.1 summarizes the final form of the new algorithm. Note that the computation of the products (section 4.2) is done in a right-looking fashion, whereas the recompressions associated with the subtractions (section 4.3) are performed in a left-looking fashion.

The following theorem analyzes the asymptotic complexity of Algorithm 4.1.

THEOREM 4.1 (complexity of fast BLR matrix multiplication: new algorithm). *Let $A \in \mathbb{R}^{n \times n}$ be a BLR matrix partitioned into p^2 blocks of order b . For the choice of block size $b = \Theta(\sqrt{nr})$, the asymptotic complexity of factorizing A via Algorithm 4.1 is*

$$\text{cost(Alg. 4.1)} = \mathcal{O}(n^{(\omega+1)/2}r^{(\omega-1)/2}).$$

Proof. The total cost is obtained by summing the previously computed costs (4.3), (4.5), (4.7), and (4.9). Using $p = n/b$, $r \leq b$, and $2 \leq \omega$, the term $\mathcal{O}(p^2br^{\omega-1})$ in (4.9) is dominated by the term $\mathcal{O}(p^2b^{\omega-1}r)$ in all the other costs. Similarly, the term $\mathcal{O}(p^3r^\omega)$ in (4.9) is dominated by the term $\mathcal{O}(p^3b^{\omega-2}r^2)$ in (4.5). In the end, we are left with the four terms

$$\mathcal{O}(pb^\omega) + \mathcal{O}(p^2b^{\omega-1}r) + \mathcal{O}(p^{\omega-1}b^2r^{\omega-2}) + \mathcal{O}(p^3b^{\omega-2}r^2).$$

For $b = \Theta(\sqrt{nr})$, each of these terms is in $\mathcal{O}(n^{(\omega+1)/2}r^{(\omega-1)/2})$. \square

For $\omega = 3$ (classical matrix multiplication), we recover the standard $\mathcal{O}(n^2r)$ complexity from [3]. For smaller ω , the asymptotic complexity of the BLR factorization is reduced. In particular, for Strassen's exponent $\omega = \log_2 7$, we obtain a complexity of $\mathcal{O}(n^{1.904}r^{0.904})$.

For $\omega = 2$, the complexity matches its lower bound given by $\text{size}(A) = \mathcal{O}(n^{3/2}r^{1/2})$. For this reason, we believe the complexity achieved by Algorithm 4.1 to be satisfying.

4.5. Practical considerations. Even though the high performance implementation of Algorithm 4.1 is outside our scope, we nevertheless discuss some practical aspects regarding it. These concern the potential for parallelism, the volume of data copies, and the storage overhead of the algorithm.

To achieve the highest performance on multicore architectures, the standard BLR algorithm performs individual operations concurrently, as described in [6]. In contrast, in the new algorithm, we sacrifice this concurrency in order to accumulate these separate operations together; this, however, significantly increases the granularity of

Algorithm 4.1 New BLR factorization.

Input: a BLR matrix $A \in \mathbb{R}^{n \times n}$ partitioned into p^2 blocks A_{ij} .

Output: A overwritten by its BLR LU factors.

```

1: Initialize empty matrices  $X_{P_{ij}}$  and  $Y_{P_{ij}}$  for all  $i, j$ 
2: for  $k = 1$  to  $p$  do
3:   Factor:  $A_{kk} = L_k U_k$ 
4:   Solve: Perform the accumulated solves (4.1) and (4.2)
5:   Product: Compute  $Z_{ij}^{(k)}$  via the accumulated product (4.4)
6:   for  $i = k + 1$  to  $p$  do
7:     for  $j = k + 1$  to  $p$  do
8:       if  $A_{ik}$  and  $A_{kj}$  are FR then
9:         if  $A_{ij}$  is FR then
10:           $A_{ij} \leftarrow A_{ij} - Z_{ij}^{(k)}$ 
11:        else
12:           $A_{ij} \leftarrow X_{ij} Y_{ij}^T - Z_{ij}^{(k)}$ 
13:        end if
14:      else if  $A_{ik}$  is LR and  $A_{kj}$  is FR then
15:        if  $A_{ij}$  is FR then
16:           $A_{ij} \leftarrow A_{ij} - X_{ik} Z_{ij}^{(k)}$ 
17:        else
18:           $X_{ij} \leftarrow [X_{ij} \quad X_{ik}]$  and  $Y_{ij} \leftarrow [Y_{ij} \quad -(Z_{ij}^{(k)})^T]$ 
19:        end if
20:      else if  $A_{ik}$  is FR and  $A_{kj}$  is LR then
21:        if  $A_{ij}$  is FR then
22:           $A_{ij} \leftarrow A_{ij} - Z_{ij}^{(k)} Y_{kj}^T$ 
23:        else
24:           $X_{ij} \leftarrow [X_{ij} \quad Z_{ij}^{(k)}]$  and  $Y_{ij} \leftarrow [Y_{ij} \quad -Y_{kj}]$ 
25:        end if
26:      else
27:        Update the LR representation  $X_{P_{ij}} Y_{P_{ij}}^T$  of  $P_{ij}$  as described in (4.8)
28:      end if
29:      if  $i = k + 1$  or  $j = k + 1$  then
30:        if  $A_{ij}$  is FR then
31:           $A_{ij} \leftarrow A_{ij} - X_{P_{ij}} Y_{P_{ij}}^T$ 
32:        else
33:           $X_{ij} \leftarrow [X_{ij} \quad X_{P_{ij}}]$  and  $Y_{ij} \leftarrow [Y_{ij} \quad -Y_{P_{ij}}]$ 
34:           $X_{ij} Y_{ij}^T \leftarrow \text{Recompress}(X_{ij} Y_{ij}^T)$ 
35:        end if
36:      end if
37:    end for
38:  end for
39: end for

```

the operations, and it is thus quite likely that the new algorithm would be faster than the standard one, even without considering the use of fast matrix arithmetic.

One drawback of accumulating the operations together is that it sometimes requires some data copies. The accumulated Solve kernel requires copying the entire current panel, unless the LR representations of the blocks are directly built in the

required accumulated format (see Figure 3). Fortunately, the accumulated product can then be performed without any extra copy, because the accumulated result of the Solve kernel can be directly reused under this format. Finally, the result of the product must sometimes be copied back to the appropriate location depending on the situation. The result of FR-FR products must always be immediately added and thus copied; the LR-FR and FR-LR products also require a copy to accumulate the result to the LR representation of the target block A_{ij} , unless A_{ij} is FR. (In this case, the result is decompressed and this can be done in place.) Finally, the result of LR-LR products never needs to be copied, since it is then compressed, which can be done in place. Since the number of LR-LR products is asymptotically dominant, we expect the volume of data copies required by the new algorithm to remain limited.

Finally, another drawback of the new algorithm is its storage overhead. Indeed, the result of LR-LR, FR-LR, and LR-FR products is computed and then accumulated, but not immediately consumed. For large matrices, the dimensions of the accumulated matrices, which are $b \times pr$, may become too important to be stored. When this happens, it suffices to process the already accumulated results (by recompressing or decompressing them, depending on whether A_{ij} is LR or FR, respectively) and then start accumulating new product results again. From a practical point of view, we only require a reasonably large enough storage overhead to obtain large enough granularities to efficiently exploit fast matrix arithmetic.

In conclusion, even though the high performance, parallel implementation of Algorithm 4.1 is a complex matter that is outside our scope and that will require further research, we believe that the reasons described above give reasonable hope that the new algorithm could achieve actual time gains for large enough matrices.

5. Numerical experiments. In this section we measure experimentally the number of flops required to factorize a sequence of matrices of increasing size, with the standard and new BLR factorization algorithms (Algorithms 3.1 and 4.1), both with and without the use of fast matrix arithmetic.

5.1. Experimental setting. We have developed a MATLAB code that implements both Algorithms 3.1 and 4.1. All the experiments were performed on the brunch computer from the LIP laboratory (ENS de Lyon).

To compute the low-rank form of the blocks, we perform a truncated QR factorization with column pivoting (that is, a truncated version of LAPACK's `geqp3` routine [7]). We stop the factorization after an accuracy of ε has been achieved. In the following experiments, we have set $\varepsilon = 10^{-14}$, which yields a residual of the same order.

To validate our theoretical complexity results, we use a Poisson problem, which is a classical choice to analyze the complexity of low-rank solvers. Matrix A is symmetric positive definite and generated from a 7-point finite-difference discretization of equation

$$\Delta u = f$$

on a 3D domain of size $N = k \times k \times k$ with Dirichlet boundary conditions, with k varying from 64 to 224. We compute the BLR factorization of the matrices corresponding to the root separator of the nested dissection partitioning [20], which are dense matrices of order $n = k^2$. Thus, n varies from $64^2 = 4096$ to $224^2 = 50176$. For the Poisson problem, it is proved in [10] that $r = \mathcal{O}(1)$; therefore, in the following, we aim at validating the asymptotic behavior in n , which should be subquadratic for our new algorithm.

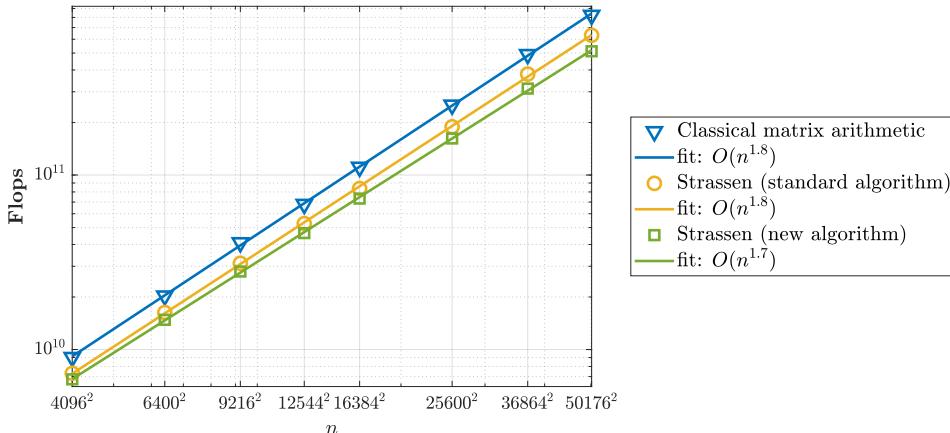


FIG. 6. Experimentally computed complexities for the standard and new BLR factorization algorithms, depending on whether classical or fast matrix arithmetic is used.

To do so, we experimentally estimate the asymptotic complexities by means of the least-squares estimation of the coefficients $\{\beta_i\}_i$ of a regression function f such that $X_{fit} = f(n, \{\beta_i\}_i)$ fits the observed data X_{obs} , where X_{obs} denotes the measured flop count for the BLR factorization. We use the following regression function:

$$X_{fit} = e^{\beta_1^*} n^{\beta_2^*} \quad \text{with} \quad \beta_1^*, \beta_2^* = \arg \min_{\beta_1, \beta_2} \| \log X_{obs} - \beta_1 - \beta_2 \log n \|^2.$$

5.2. Complexity experimental analysis. We report in Figure 6 the computed complexities for the standard and new BLR factorization algorithms, depending on whether classical or fast matrix arithmetic is used. In the case of fast matrix arithmetic, we use Strassen's algorithm, for which $\omega = \log_2 7 \approx 2.81$.

For classical matrix arithmetic, both algorithms achieve the same complexity $\mathcal{O}(n^{1.8})$ (blue triangle curve). With fast matrix arithmetic (using Strassen's algorithm), the standard algorithm (yellow circle curve) fails to reduce the asymptotic complexity of the factorization, achieving only a constant gain. In contrast, the new algorithm (green square curve) achieves a reduced asymptotic complexity of $\mathcal{O}(n^{1.7})$. For the largest problem, using Strassen's matrix arithmetic with the standard algorithm yields a 23% gain (with respect to classical matrix arithmetic) and a 38% gain with the new algorithm.

In all three cases, the complexities computed experimentally are better than their respective theoretical bounds ($\mathcal{O}(n^2)$ for the first two curves and $\mathcal{O}(n^{1.9})$ for the third) by a factor of the order of $n^{0.2}$.

These numerical results confirm that the new BLR factorization algorithm exploits fast matrix arithmetic more efficiently and achieves a better asymptotic complexity than the standard algorithm.

These experiments only consider the reduction of the BLR factorization cost in terms of flops. Investigating how to translate this theoretical reduction into gains in computing time is outside our scope. We note recent advances on the high performance implementation of Strassen's algorithm [15, 8, 12, 18, 27, 26], as well as on reducing the constant hidden in the \mathcal{O} of the complexity of fast matrix algorithms [29, 11], which allow for significant time gains even for moderately large n .

TABLE 3

Backward error $\|Ax - y\|/(\|A\|\|x\| + \|y\|)$ for solving the linear system $Ax = y$, where x is the all-ones vector and A is a BLR matrix corresponding to the root separator of a Poisson problem of size 64^2 .

	$\varepsilon = 10^{-14}$		$\varepsilon = 10^{-8}$	
	Classical	Strassen	Classical	Strassen
Algorithm 3.1	4.61e-15	4.64e-15	1.56e-08	1.56e-08
Algorithm 4.1	4.55e-15	4.44e-15	1.56e-08	1.56e-08

5.3. Numerical error analysis. We now analyze experimentally the accuracy of the BLR factorization by measuring the numerical error produced depending on the choice of factorization algorithm, matrix arithmetic, and low-rank truncation threshold ε .

We measure the backward error

$$\|Ax - y\|/(\|A\|\|x\| + \|y\|)$$

for solving the linear system $Ax = y$, where A is the BLR matrix corresponding to the 64^2 Poisson problem and y is the sum of the columns of A , so that the solution x is known to be the all-ones vector. IEEE double precision (fp64) is used.

We report in Table 3 the backward error for both Algorithms 3.1 and 4.1 using either classical or Strassen matrix arithmetic, for two choices of low-rank truncation threshold, $\varepsilon = 10^{-14}$ and $\varepsilon = 10^{-8}$.

It is clear from the results that the error is strongly correlated to ε and almost does not depend on the choice of factorization algorithm or matrix arithmetic. Intuitively, this is not surprising: while the overall error for the BLR factorization depends on both the floating-point precision and the low-rank truncation threshold, we can expect the truncation errors to dominate the floating-point ones if ε is large enough. Note that a rigorous proof of this intuition is the subject of ongoing research [25].

6. Conclusion.

6.1. Summary. BLR dense matrices of order n with blocks of rank at most r can be factorized classically in $\mathcal{O}(n^2r)$ flops. Even though lower complexities can be achieved with hierarchical formats, the BLR format allows for a particularly simple and efficient implementation within a parallel, general-purpose, algebraic solver. For this reason, in this article, we have investigated how to exploit fast matrix arithmetic within the BLR factorization in order to reduce its asymptotic complexity without losing its nonhierarchical nature and thus all the benefits associated with it.

The standard BLR factorization algorithm involves many intermediate operations on matrices of small rank $r \ll n$. If each of these operations is accelerated with fast matrix arithmetic separately, the quadratic dependence on n remains: the complexity is only reduced from $\mathcal{O}(n^2r)$ to $\mathcal{O}(n^2r^{\omega-2})$, where $\omega < 3$ corresponds to the complexity $\mathcal{O}(n^\omega)$ of multiplying two $n \times n$ matrices.

To improve upon this first underwhelming result, we have proposed a new BLR factorization algorithm that accumulates these low-rank intermediate matrices together to perform operations on matrices of larger granularity. This allows for a more efficient use of fast matrix arithmetic and reduces the asymptotic complexity to $\mathcal{O}(n^{(\omega+1)/2}r^{(\omega-1)/2})$. This gives a theoretical bound of $\mathcal{O}(n^{3/2}r^{1/2})$ for $\omega = 2$ which corresponds to the complexity of storing a BLR matrix, and is thus linear in the size of the input. Moreover, this complexity already represents a significant improvement

for practical values of ω , such as $\omega = \log_2 7$ (Strassen's algorithm), for which the complexity is $\mathcal{O}(n^{1.904}r^{0.904})$.

Recall that the complexity of the \mathcal{H} -matrix factorization is reduced from $\mathcal{O}(nr^2)$ to $\mathcal{O}(nr^{\omega-1})$ thanks to the use of fast matrix arithmetic (ignoring logarithmic factors for simplicity). Interestingly, the ratio between the BLR and \mathcal{H} complexities is thus also reduced with the use of fast matrix arithmetic, from $\mathcal{O}(n/r)$ to $\mathcal{O}((n/r)^{(\omega-1)/2})$. Therefore, fast matrix arithmetic can partially bridge the asymptotic complexity gap between flat and hierarchical low-rank matrix formats.

6.2. Perspectives. A possible drawback of fast matrix arithmetic is its larger error bounds in floating-point arithmetic [23], [24, section 23.2]. Interestingly, this problem may be less important in the BLR setting. Indeed, as experimentally observed in section 5.3, the accuracy of the BLR factorization depends on both floating-point and low-rank truncation errors, but the latter appear to dominate the former when the low-rank truncation threshold ε is large enough. Since the use of fast matrix arithmetic only concerns floating-point errors, its impact on the numerical behavior of the BLR factorization may be limited. A rigorous error analysis proving this valuable property is the subject of ongoing research [25].

As mentioned in the introduction, the MBLR format [5] is a multilevel extension of the BLR matrices. The idea is to recursively partition the full-rank blocks of the matrix into BLR matrices, up to a fixed number of levels ℓ . Investigating whether and by how much the $\mathcal{O}(n^{(\ell+3)/(\ell+1)}r^{2\ell/(\ell+1)})$ asymptotic complexity of the MBLR factorization can be reduced with fast matrix arithmetic could be the subject of future work. However, one difficulty lies with the size of the blocks, which becomes smaller and smaller at lower levels of the hierarchy, thus making it more challenging to efficiently exploit fast matrix arithmetic.

Finally, the ideas behind the new algorithm presented in section 4 raise interesting perspectives independently of the use of fast matrix arithmetic. As mentioned in section 4.5, the accumulated kernels are likely to improve the performance of the BLR factorization by increasing the granularity of its BLAS operations, in the same spirit as the LUA strategy proposed in [6], or, more generally, as batched BLAS computations [17]. This could be of particular interest when targeting architectures benefitting from large granularities of computations, such as GPU accelerators.

REFERENCES

- [1] K. AKBUDAK, H. LTAIEF, A. MIKHALEV, A. CHARARA, A. ESPOSITO, AND D. KEYES, *Exploiting data sparsity for large-scale matrix computations*, in Proceedings of Euro-Par 2018: Parallel Processing, M. Aldinucci, L. Padovani, and M. Torquati, eds., Springer, Berlin, 2018, pp. 721–734, https://doi.org/10.1007/978-3-319-96983-1_51.
- [2] 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.
- [3] P. 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>.
- [4] P. R. AMESTOY, R. BROSSIER, A. BUTTARI, J.-Y. L'EXCELLENT, T. MARY, L. MÉTIVIER, A. MINIUSI, 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>.
- [5] 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>.

- [6] 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.
- [7] E. ANDERSON, Z. BAI, C. BISCHOF, L. S. BLACKFORD, J. DEMMEL, J. DONGARRA, J. DUROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, AND D. SORENSEN, *LAPACK Users' Guide*, 3rd ed., SIAM, Philadelphia, 1999.
- [8] G. BALLARD, J. DEMMEL, O. HOLTZ, B. LIPSHITZ, AND O. SCHWARTZ, *Communication-optimal parallel algorithm for Strassen's matrix multiplication*, in Proceedings of the Twenty-fourth Annual ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'12, ACM, New York, 2012, pp. 193–204, <https://doi.org/10.1145/2312005.2312044>.
- [9] M. BEBENDORF, *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems*, Lect. Notes Comput. Sci. Eng. 63, Springer, Berlin, 2008.
- [10] M. BEBENDORF AND W. HACKBUSCH, *Existence of \mathcal{H} -matrix approximants to the inverse FE-matrix of elliptic operators with L^∞ -coefficients*, Numer. Math., 95 (2003), pp. 1–28.
- [11] G. BENIAMINI AND O. SCHWARTZ, *Faster matrix multiplication via sparse decomposition*, in Proceedings of the 31st ACM Symposium on Parallelism in Algorithms and Architectures, SPAA'19, ACM, New York, 2019, ACM, pp. 11–22, <https://doi.org/10.1145/3323165.3323188>.
- [12] A. R. BENSON AND G. BALLARD, *A framework for practical parallel fast matrix multiplication*, SIGPLAN Not., 50 (2015), pp. 42–53, <https://doi.org/10.1145/2858788.2688513>.
- [13] J. R. BUNCH AND J. E. HOPCROFT, *Triangular factorization and inversion by fast matrix multiplication*, Math. Comp., 28 (1974), pp. 231–236.
- [14] A. CHARARA, D. KEYES, AND H. LTAIEF, *Tile low-rank GEMM using batched operations on GPUs*, in Proceedings of Euro-Par 2018: Parallel Processing, M. Aldinucci, L. Padovani, and M. Torquati, eds., Springer, Berlin, 2018, pp. 811–825.
- [15] P. D'ALBERTO, M. BODRATO, AND A. NICOLAU, *Exploiting parallelism in matrix-computation kernels for symmetric multiprocessor systems: Matrix-multiplication and matrix-addition algorithm optimizations by software pipelining and threads allocation*, ACM Trans. Math. Software, 38 (2011), pp. 2:1–2:30, <https://doi.org/10.1145/2049662.2049664>.
- [16] J. DEMMEL, I. DUMITRIU, AND O. HOLTZ, *Fast linear algebra is stable*, Numer. Math., 108 (2007), pp. 59–91.
- [17] J. DONGARRA, S. HAMMARLING, N. J. HIGHAM, S. D. RELTON, P. VALERO-LARA, AND M. ZOUNON, *The design and performance of batched BLAS on modern high-performance computing systems*, Procedia Comput. Sci., 108 (2017), pp. 495–504, <https://doi.org/10.1016/j.procs.2017.05.138>.
- [18] J.-G. DUMAS, T. GAUTIER, C. PERNET, J.-L. ROCH, AND Z. SULTAN, *Recursion based parallelization of exact dense linear algebra routines for Gaussian elimination*, Parallel Comput., 57 (2016), pp. 235–249, <https://doi.org/10.1016/j.parco.2015.10.003>.
- [19] Y. EIDELMAN AND I. GOHBERG, *On a new class of structured matrices*, Integral Equations Operator Theory, 34 (1999), pp. 293–324, <https://doi.org/10.1007/BF01300581>.
- [20] A. GEORGE, *Nested dissection of a regular finite-element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.
- [21] W. HACKBUSCH, *Hierarchical Matrices: Algorithms and Analysis*, Springer Ser. Comput. Math. 49, Springer, Berlin, 2015, <https://doi.org/10.1007/978-3-662-47324-5>.
- [22] N. HALKO, P.-G. MARTINSSON, AND J. TROPP, *Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions*, SIAM Rev., 53 (2011), pp. 217–288, <https://doi.org/10.1137/090771806>.
- [23] N. J. HIGHAM, *Exploiting fast matrix multiplication within the level 3 BLAS*, ACM Trans. Math. Software, 16 (1990), pp. 352–368, <https://doi.org/10.1145/98267.98290>.
- [24] N. J. HIGHAM, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002, <https://doi.org/10.1137/1.9780898718027>.
- [25] 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, 2019, <http://eprints.maths.manchester.ac.uk/2730/>.
- [26] J. HUANG, L. RICE, D. A. MATTHEWS, AND R. A. VAN DE GEIJN, *Generating families of practical fast matrix multiplication algorithms*, in Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2017, pp. 656–667, <https://doi.org/10.1109/IPDPS.2017.56>.
- [27] J. HUANG, T. M. SMITH, G. M. HENRY, AND R. A. VAN DE GEIJN, *Strassen's algorithm reloaded*, in Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16, IEEE Press, Piscataway, NJ, 2016, pp. 59:1–59:12.

- [28] C.-P. JEANNEROD, C. PERNET, AND A. STORJOHANN, *Rank-profile revealing Gaussian elimination and the CUP matrix decomposition*, J. Symbolic Comput., 56 (2013), pp. 46–68.
- [29] E. KARSTADT AND O. SCHWARTZ, *Matrix multiplication, a little faster*, in Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA’17, ACM, New York, 2017, pp. 101–110, <https://doi.org/10.1145/3087556.3087579>.
- [30] F. LE GALL, *Powers of tensors and fast matrix multiplication*, in Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ISSAC ’14, ACM, New York, 2014, pp. 296–303, <https://doi.org/10.1145/2608628.2608664>.
- [31] T. MARY, *Block Low-Rank Multifrontal Solvers: Complexity, Performance, and Scalability*, Ph.D. thesis, Université de Toulouse, 2017.
- [32] C. PERNET AND A. STORJOHANN, *Time and space efficient generators for quasiseparable matrices*, J. Symbolic Comput., 85 (2018), pp. 224–246, <https://doi.org/10.1016/j.jsc.2017.07.010>.
- [33] G. PICHON, E. DARVE, M. FAVERGE, P. RAMET, AND J. ROMAN, *Sparse supernodal solver using block low-rank compression: Design, performance and analysis*, J. Comput. Sci., 27 (2018), pp. 255–270, <https://doi.org/10.1016/j.jocs.2018.06.007>.
- [34] W. H. PRESS, S. A. TEUKOLSKY, W. T. VETTERLING, AND B. P. FLANNERY, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, Cambridge University Press, New York, 2007.
- [35] M. SERGENT, D. GOUDIN, S. THIBAULT, AND O. AUMAGE, *Controlling the memory subscription of distributed applications with a task-based runtime system*, in Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016, pp. 318–327, <https://doi.org/10.1109/IPDPSW.2016.105>.
- [36] 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>.
- [37] A. STORJOHANN, *Algorithms for Matrix Canonical Forms*, Ph.D. thesis, Swiss Federal Institute of Technology–ETH, 2000.
- [38] V. STRASSEN, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.
- [39] J. XIA, S. CHANDRASEKARAN, M. GU, AND X. S. LI, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 17 (2010), pp. 953–976.