



HAL
open science

2D Static Resource Allocation for Compressed Linear Algebra and Communication Constraints

Olivier Beaumont, Lionel Eyraud-Dubois, Mathieu Verite

► **To cite this version:**

Olivier Beaumont, Lionel Eyraud-Dubois, Mathieu Verite. 2D Static Resource Allocation for Compressed Linear Algebra and Communication Constraints. 2020. hal-02900244v1

HAL Id: hal-02900244

<https://inria.hal.science/hal-02900244v1>

Preprint submitted on 15 Jul 2020 (v1), last revised 24 Jul 2020 (v2)

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

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

2D Static Resource Allocation for Compressed Linear Algebra and Communication Constraints

Olivier Beaumont
Inria Bordeaux Sud-Ouest
University of Bordeaux
Talence, France
Olivier.Beaumont@inria.fr

Lionel Eyraud-Dubois
Inria Bordeaux Sud-Ouest
University of Bordeaux
Talence, France
Lionel.Eyraud-Dubois@inria.fr

Mathieu Verite
Inria Bordeaux Sud-Ouest
University of Bordeaux
Talence, France
Mathieu.Verite@inria.fr

Abstract—This paper addresses static resource allocation problems for irregular distributed parallel applications. More precisely, we focus on two classical tiled linear algebra kernels: the *Matrix Multiplication* (MM) and the *LU decomposition* (LU) algorithms on large linear systems. In the context of parallel distributed platforms, data exchanges can dramatically degrade the performance of linear algebra kernels and in this context, compression techniques such as *Block Low Rank* (BLR) compression techniques are good candidates both for limiting data storage on each processor and data exchanges between processors. On the other hand, the use of BLR representation makes the static allocation problem of tiles to processors more complex. Indeed, the load associated to each tile depends on its compression factor, which induces an heterogeneous load balancing problem. In turn, solving this load balancing problem optimally might lead to complex allocation schemes, where the tiles allocated to a given processor are scattered all over the matrix. This in turn induces communication costs, since matrix multiplication and LU decompositions heavily rely on broadcasting operations along rows and columns of processors, so that the communication volume is minimized when the maximal number of different processors on any row and column is minimized. In the fully homogeneous case, *2D Block Cyclic* (BC) allocation solves both load balancing and communication minimization issues simultaneously, but it might lead to bad load balancing in the heterogeneous case. Our goal in this paper is to propose data allocation schemes dedicated to BLR format and to prove that it is possible to obtain good performance on makespan when simultaneously balancing the load and minimizing the maximal number of different processor in any row or column.

Index Terms—Load Balancing, Linear Algebra, Compression, BLR, Block Cyclic, Randomized Algorithms

I. INTRODUCTION

In this paper, we consider the implementation of several linear algebra operations on CPU and GPU platforms where the matrix is described in a compressed form in BLR (Block Low Rank) format. Indeed, many of the time-consuming tasks performed on supercomputers

are linear algebra operations. With the advent of multi-core architectures and massive parallelism, it is therefore particularly interesting to optimize and understand their parallel behavior. In this paper, we consider the problems of Cholesky factorization, LU factorization and the matrix product. Recently, these problems have received particular attention, especially because they are used to validate the behavior of runtime schedulers [3], [9], [32], [37]. For instance, tiled Cholesky factorization has been considered in StarPU [8], [21], SMPSPs [34], and DAGuE [16]. In practice, the kernels that we consider in this paper have been implemented in Dense Linear Algebra state of the art libraries, such as Chameleon [2], DPLASMA [15], PLASMA [17]. Recently, the practical design of good static schedule for heterogeneous resources has been considered in [4] and extensions to sparse matrices [29] have also been proposed.

Today, it is acknowledged that storing and moving data across a distributed platform limit the performance of linear algebra algorithms. In this context, alternative approaches have been proposed to store data, in particular using the opportunity of representing the matrix in a compressed form. For example, the \mathcal{H} -matrix representation introduced in [27] and is now widely used to reduce storage costs and also execution time significantly. For instance, it is known [25] that in the context of LU-factorization, the number of floating point operations can be reduced from $\Theta(2/3n^3)$ to $\Theta(nk^2 \log^2 n)$, where k is a parameter that represents the compression that can be achieved. Recently, these techniques have also been extended to sparse matrices [23], [35].

Complementary approaches, that avoid issues related to complex and irregular data accesses induced by \mathcal{H} -matrix representation, have been proposed. Block Low Rank (BLR) decompositions have been advocated for instance for multi-frontal methods [33] or finite-element

matrices [5]. In BLR decompositions, matrices are represented as regular tiled matrices, except that each block is represented using a low rank decomposition (when possible). Another approach is lattice \mathcal{H} -matrices [38]. The work in [19] follows the same line of research. In all cases, the idea is increase simplicity and to keep a high compression ratio, even at the price of a slight increase in the flop complexity. Indeed, keeping the regular tiled structure makes it possible to use directly runtime schedulers, that have proven their efficiency in making use of heterogeneous resources, at least at the level of a single computing node consisting of one multicore CPU and several accelerators [2].

When considering more distributed platforms, typically consisting in several such nodes, the problem of reducing communications and therefore to compress involved matrices, becomes even more crucial. Indeed, it has been proved [3], [4] that, at the level of a single heterogeneous node, basic dynamic runtime schedulers are such as [8] are in general enough to produce a schedule where communications can be overlapped by computations, provided that the tile size is large enough. This property does not hold true anymore when considering distributed nodes, since moving tiles across the network is a costly operation, both in terms of time and memory. In this context, it has been proved in [2] that in the case of distributed memory platforms, a careful static distribution of the tiles is crucial.

In this context, our general goal in the present paper is to propose static allocation schemes that are adapted to BLR matrices for Matrix Multiplication and LU Factorization kernels in the context of a platform consisting of a set of distributed memory nodes. These static distributions should be able to

(i) balance the load between the different nodes, knowing the relative weight of each tile, related to the rank of its low rank decomposition in the BLR format. Note that in this context, balancing the load is closely related to balancing the memory needs on each node.

(ii) minimize the volume of communications between nodes and minimize the number of communications between nodes. Since we consider applications where most of the communications are based on broadcasts among the nodes that share the same row or the same column of tiles, we will bound the maximal number of participating nodes in each row/column.

In all cases, the achieved makespan, *i.e.* the time to completion, will be used as a metric to evaluate the quality of data distribution schemes.

The rest of paper is organized as follows. In Section II,

we review the related work related to packing problems with additional constraints, heterogeneous partitioning problems arising in linear algebra and practical solutions used in the case of compressed matrices. Then, we formalize the platform model and the linear algebra kernels that are used throughout the paper in Section III. The strategies to minimize makespan are described in Section IV and experimental results are given in Section V. Then, Section VI is dedicated to the detailed analysis of Block Cyclic based solutions. A last, concluding remarks and perspectives are proposed in Section VII.

II. RELATED WORK

A. Heterogeneous Load Balancing in Linear Algebra

The dual problem of the one we address in this paper, *i.e.* how to allocate dense homogeneous matrices to heterogeneous resources while minimizing communications, has been considered in the literature. For example, in the context of matrix multiplication, the problem of partitioning a matrix between heterogeneous resources has been modeled by Kalinov et al. [31] as the problem of partitioning a square into fixed area rectangles while minimizing the sum of the perimeters of these rectangles. Since then, numerous works have successively improved the approximation ratio of this NP-Complete problem, which are summarized in a recent survey paper [10]. In a more formal way, but in a homogeneous framework, the problem of communication minimization associated with the matrix product, considered as a cube instead of a square, has been considered by [28] and in the development of communication avoiding algorithms [36].

A closely related problem is the one of allocating a sparse matrix to a set of homogeneous processors in order to balance the load between processors while keeping a regular allocation structure has been introduced by F. Manne et al. under the name of rectilinear partitioning or generalized block distribution in [26] and is still the object of an active literature [39]. In rectilinear partitioning, the problem consist in partitioning the rows and the columns into groups, the intersection between a group of columns and a group of rows being in turn assigned to a given processor. In this context, the main goal is to build groups such as the load (*i.e.* the number of non-zero elements allocated to a processor) is well balanced.

B. Cholesky Factorization with BLR Format

Numerical Linear Algebra techniques are at the core of many intensive scientific applications requiring large linear system resolution. In this context, state-of-the-art methods are based on using efficient linear algebra

kernels under task-based runtime systems for scheduling. In [7] the authors detail techniques to perform the Cholesky factorization of matrix which off-diagonal tiles are compressed; the underlying linear system arises from Boundary Elements Methods (BEM) applied to wave propagation modeling. The proposed implementation for a distributed memory platform makes use of plain 2D block cyclic method to allocate tiles to processors, which is expected to balance the workload associated to each processor while reducing the number of processors involved in each data transfer, thus allowing a better overlap of communications by computations.

2D block cyclic method has been proven to achieve optimal load balancing between resources in the dense case [14] and is the only allocation rule implemented in historical ScaLAPACK library. In the field of weather forecast, maximum-likelihood estimation method requires the resolution of large scale linear systems. In [1] the authors propose a Cholesky decomposition of a matrix whose tiles are compressed using BLR format. Since full rank diagonal tiles are associated to longer work time, they are treated specifically to ensure better load balancing between processors, in distributed platforms implementation. The proposed hybrid tiles to processors allocation method, detailed in [18], is a round robin allocation for diagonal tiles and a 2D block cyclic for off diagonal ones.

C. Bin Packing Problems

The optimization problem considered in this paper can be considered as a bin packing problem, where tiles of the matrix are items to be packed into the different processors, considered as bins. Considerations on limiting communications lead use to include additional constraints on the feasible packings, with a limit on the number of processors allotted to a given row or column. This can be seen as a generalization of some related literature on bin packing: bin packing with class constraints [22], [30], in which there is a limit to the number of classes allowed in a bin, and bin packing with minimum color fragmentation [12], in which the number of bins containing a given class is limited. Since we consider both rows and columns, our case is a two-dimensional version of these problems, however the techniques developed in these papers can not be generalized to our context.

III. MODEL AND NOTATIONS

A. Notations

Let us consider a matrix A of size $N \times N$, divided in n^2 tiles of size $n_b \times n_b$ where $n = N/n_b$. We

consider that n_b is fixed, typically so that the granularity of resulting tasks is large enough for a computing node consisting of several GPUs, and small enough so as to be able to balance the load.

For any $(i, j) \in \llbracket 1; N \rrbracket^2$ the tile in position (i, j) will be denoted by $A_{i,j}$. Following BLR format, we assume that each tile is represented as a (low) rank matrix, *i.e.* that $\forall i, j, \exists n_{i,j} \in \mathbb{N}^*$ and two rectangular matrices $U_{i,j}, V_{i,j}$ of respective size $n_b \times n_{i,j}$ and $n_{i,j} \times n_b$ such that $A_{i,j} \approx U_{i,j}V_{i,j}$.

The ratio $d_{i,j} = \frac{n_{i,j}}{n_b}$ can be seen as the density of the tile in position (i, j) and is denoted by $d[i, j]$. In what follows, if the cost of an operation on a dense full rank tile is C , we assume that the cost of the associated operation on $A_{i,j}$ is $d[i, j]C$, except in the simulations where we will consider the actual costs.

Concerning the computing platform, we assume that it consists into P identical (homogeneous) processors, that can in turn consist in several resources such as accelerators. Our goal is to allocate the set of tiles $A_{i,j}$ onto these P distributed memory processors while minimizing data exchanges between processors. On each given processor, we assume that a runtime scheduler such as StarPU [8], [21] is used to actually map elementary tasks to resources. Note that once tile $A_{i,j}$ is allocated to processor P_k , then P_k will be responsible for all the tasks that modify the value of tile $A_{i,j}$ and that P_k will receive the other input tiles if it does not hold them locally. This assumption concerning the organization of computations is classic and all libraries such as Chameleon [2] or DPLASMA [15] rely on the same model. For both LU factorization (LU) or Matrix Multiplication (MM), we seek to find a static allocation of tiles to resources that minimizes the total makespan of the kernel.

B. Matrix Multiplication

Let us consider a square matrix A of $N \times N$ tiles, where the size of tile $A_{i,j}$ is $n_b \times n_b$ and let us assume that we want to compute $C = AA^T$. In order to compute C , we rely of a variant of Canon's algorithm [24] adapted to low rank representation. More specifically, the algorithm consists in N stages. During stage k , all block matrix multiplications of type $C_{i,j} + = A_{i,k}A_{k,j}$ are performed. In a distributed implementation, the data distribution for A and C are the same and in terms of computations, at stage k , the processor in charge of $A_{i,k}$ tile for any value i broadcasts $A_{i,k}$ to the processors that are in charge of $C_{i,j}$ (we will denote them as the processors of its row of processors) and the processor in charge of $A_{k,j}$ tile for any value j broadcasts $A_{k,j}$ to the

processors that are in charge of $C_{i,j}$ (we will denote them as the processors of its column of processors). Therefore, at each stage, exactly one broadcast takes place in each row and each column of processors.

C. LU Factorization

Let us consider a square matrix A of $N \times N$ tiles, where the size of tile $A_{i,j}$ is $n_b \times n_b$. We consider the textbook right looking variant of LU decomposition described in [24]. The algorithm consist in N stages and each step can itself be decomposed in 4 steps. All operations are performed in-place and each operation therefore replaces one of its entries by its output. For the sake of simplicity, we will denote the block in position i,j of the resulting matrix by $A_{i,j}$, even if it is not a block of the initial input matrix A . During stage k , the tiled LU decomposition algorithm (i) computes the LU decomposition of the tile in position k,k using GETRF LAPACK operation, (ii) then updates the tiles in column k (with L_k) and the tiles in row k (with U_k), using the factors L_k and U_k (produced by the previous GETRF operation) and TRSM LAPACK operation and then (iii) updates the remaining blocks of A in the lower right corner with respect to the block row and column panels (just computed with TRSM operations) using GEMM LAPACK operation. The set of operations, together with their dependences, are depicted in Figure 1. Let us first remark that only the edges depicted in Figure 1 correspond to actual data dependencies, and that above presentation, based on stages, is only used for the sake of simplicity. Indeed, one of the level 1 GEMM operation (the one that correspond to the lower right corner) does not have to be performed before the GETRF and and the TRSMs of stage 2, even on this small example. If we now consider a distributed setting, in terms of communications, all operations correspond to broadcast operations. Indeed, at each stage, the block produced by the GETRF operation is broadcasted along the k th row of processors and the the k th column of processors. Then, the blocks produced by the TRSM operations are then broadcasted, either in their respective row or column of processors.

D. Constraints

Regarding the communication model, we assume that the bandwidth of the links between processors is not large enough for purely dynamic allocation strategies to be able to overlap communications and computations. Therefore, it is crucial to minimize the overall volume of communications. In the applications described in Section III-B and Section III-C, communications consist

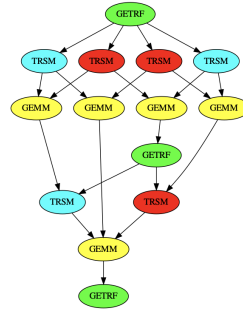


Fig. 1: Task graph of the LU decomposition for a 3×3 matrix

in broadcast operations performed on the processors allocated to the same rows and columns of the matrix. In this context, a crucial metric for performance is to minimize the number of different processors in the same row or column. Indeed, let us consider a broadcast operation of a message of size S performed on a row of size N , but with only n_p different processors in the same row. Then, the broadcast requires the transmission of exactly $(n_p - 1)S$ bits (S bits per unique processor) and the duration can even be reduced to order $\log n_p$ when using a diffusion tree. It is therefore crucial to minimize n_p in order to minimize the volume of data transmitted over the network.

In the 2D block-cycling case with P processors, the solution consists in creating a virtual grid of $\sqrt{P} \times \sqrt{P} = P$ processors, which define \sqrt{P} row and column types. Then, the different row (resp. column) types are allocated in a round robin fashion to the rows and columns of the matrix. Considering the total load \mathcal{L} , such allocation scheme leads to perfect load balancing between processors if the processing costs of the different tasks are homogeneous since (i) if \sqrt{P} and N/\sqrt{P} are integers, then each processor receives a load equal to \mathcal{L}/P (ii) a solution in which each row and each column would have strictly less than \sqrt{P} processors cannot achieve perfect load balancing.

In the case where the workloads associated to the different tiles are heterogeneous, as in the context of this paper where compression is used, the solution is more complex. Indeed, if the workload of each tile is weighted, trying to balance the loads among processors (which is a hard NP-difficult problem) is likely to yield an allocation such that some rows or columns featuring all processors. This would in turn induce very important communication costs. Conversely, it is expected that a

solution in which one would impose to have exactly \sqrt{P} processors per row and column, especially if one imposes a cyclic round robin distribution of rows and columns, may lead to a significant load imbalance. Hence, to limit the overhead due to communications while maintaining a good load balance throughout the computation, constraints are added to limit the number of different on each row and column. More specifically, a feasible allocation is a one to one tile to resource assignment, denoted as *mappings* M , such that no more than $\alpha\sqrt{P}$ different processors appear on any row and column of M . The $\alpha \in [1; +\infty[$ parameter controls the tightness of communication restriction; $\alpha = 1$ being the basic *2D block cyclic* solution used for homogeneous case. In the following, larger values of α ($\alpha = 1.1, 1.5, 2$ or 3) will be considered.

E. Evaluation Metric

The objective is to find a feasible allocation M that minimizes the makespan of the target kernel (LU factorization or Matrix Multiplication). However, finding a closed form formula to evaluate the performance of an allocation, that would take into account the communication costs, the evolution of the ranks of the tiles, the contentions on the communication links and the capability of overlapping communications and computations is impossible in practice. Therefore, the evaluation of the makespan denoted by $\mathfrak{R}(M)$ in Section V is performed in this paper using either direct experiments (for computational costs) or fine grain simulations using SimGrid.

Nevertheless, in order to build the allocations, we will rely on the following surrogate metrics in order to evaluate the expected performance of an allocation and to guide the heuristics.

(i) $\mathfrak{B}(M)$ is associated to load balancing. For a given allocation M , $\mathfrak{B}(M)$ denotes the overall load allocated to any processor assuming that the processors are never idle and that communications can always be overlapped with computations. We denote by $\bar{C}[i, j]$ the overall cost of all the tasks associated to the position $[i, j]$ and $\mathfrak{B}(M)$ is then given by $\mathfrak{B}(M) = \max_{p \in \{1, \dots, P\}} \left(\sum_{\substack{(i,j) \in \llbracket 1; N \rrbracket^2 \\ M[i,j]=p}} \bar{C}[i, j] \right)$.

(ii) $\mathfrak{S}_{\text{BASIC}}(M)$ denotes the maximum (simulated) makespan, assuming that communications can always be overlapped with computations but taking into account the idle time that can be induced by task dependencies. $\mathfrak{S}_{\text{BASIC}}(M)$ value is calculated using `simu-exe` algorithm.

(iii) $\mathfrak{S}_{\text{SIMGRID}}(M)$ denotes the maximum (simulated) makespan, using the realistic communication model provided by SimGRID [20] and taking into account the idle time that can be induced by task dependencies.

IV. DATA DISTRIBUTION SCHEMES

In this section, we review some data distribution schemes, the objective of which is both to balance the computation load between the different resources throughout the computation and to minimize the number of processors participating in the same row or column of the matrix to minimize communications. In Section IV-A, we describe the 2D Block Cyclic algorithm (BC) that is classically used in the case of homogeneous cost tiles. In Section IV-B, we consider a natural extension of the 2D Block Cyclic algorithm, called Extended 2D Block Cyclic (BCE). Variants of this scheme have been proposed in [13] for example, but we propose here a formalization of the algorithm and an optimal algorithm to solve the associated optimization problem. Finally, we propose a randomized strategy in Section IV-C, that does not rely on a repeated pattern.

A. 2D Block Cyclic

Plain *2D block cyclic* is a straightforward and extensively used method in numerical algebra kernels to perform the allocation in the more regular dense case. Tiles are allocated to processors following a rectangular pattern, subsequently called *grid* and denoted G . This pattern is then repeated over the whole matrix. The grid of size $r \times c$ is almost square: typically $r = c - 1$ and r the smallest integer such that $P \geq r * c$. It is filled using all processors in a round-robin fashion. This shape choice ($r = c - 1$) is expected to ensure the variety of processors allocated to diagonal tiles. In the following, *2D block cyclic* method is denoted as BC. In the homogeneous case, BC is known to have all desirable properties, provided that r is close to \sqrt{P} . Indeed, in this case, (i) there are only (close to) optimal number \sqrt{P} of different processors on each row and column and (ii) the load associated to each processor is expected to be almost well balanced, since each processor is allocated $\lfloor \frac{N^2}{P} \rfloor$ (or $\lceil \frac{N^2}{P} \rceil$) homogeneous tiles. In the heterogeneous case we consider in this paper, (i) still holds under the same conditions on P but (ii) might not hold true anymore, since tiles have different ranks and therefore different costs. Nevertheless, if $\frac{N^2}{P}$ is large enough, the load associated to each processor will be the summation of a large number of individual tiles, and the resulting overall load balancing between processors might be acceptable due to the law of large numbers.

We will analyze these observations in more details in Section VI.

B. Extended 2D Block Cyclic Distribution

In order to balance the load between the processors, one possible idea is to relax condition (i) a bit so as to better control the load allocated to each processor. This strategy has already been proposed in [13] but with a non-optimal computation algorithm. To do this, we allow the use of approximately $\alpha\sqrt{P}$ processors per row and per column, for a small value of α , typically $\alpha \leq 3$. We then construct a BC distribution using a grid of $\alpha^2 P$ virtual processors. The last step is to associate each virtual processor with a real one. This last step can typically be performed using any bin packing algorithm, either an optimal one based on an Integer Linear Program or using a classical approximation algorithm (details can be found in companion research report [11]). Typically, if $\alpha = \sqrt{2}$, each real processor is expected to receive the load from two virtual processors. For instance, one processor will receive the cumulative load of the most and least loaded virtual processor, which is expected to lead to a better load balancing. Again, these observations will be analyzed in more details in Section VI.

C. Random Subsets Algorithm

Both BC and BCE impose a very regular pattern for allocating tiles to processors. This very regular pattern has the advantage of allowing direct control of the number of processors allocated to each row and each column and of easily implementing the algorithm for allocating tiles to the processors. On the other hand, this very regular pattern (i) imposes constraints on P because of the size of the pattern and (ii) imposes an imperfect solution to the load balancing problem because of the pattern's own constraints, although BCE is expected to limit this issue somewhat.

The idea behind Random Subset (RS) is to get away from the regular pattern constraint, using a randomized strategy. To do this, we start by determining Q sets of $\alpha\sqrt{P}$ processors, which will correspond to the different rows of the matrix. Each set R_k corresponds to a list of $\alpha\sqrt{P}$ processors, which will be the ones to appear on the row i of the matrix if it is allocated to R_k . In practice, each of the Q sets of rows is generated by choosing $\alpha\sqrt{P}$ processors randomly and uniformly, without replacement.

The idea is then to perform the same task on the columns by determining sets C_l , $1 \leq l \leq Q$ of processors that may appear in the same column. To do this, the procedure is very close to the one used for the

rows and for each set C_l , a list of $\alpha\sqrt{P}$ processors is generated in a uniform random way. However, we then check that this column is compatible with all the sets of rows, i.e. that the intersection between this column and each of the rows is not empty. If not, this column is rejected and the process continues until Q compatible columns are available.

In [11], we show how to choose the value of Q to ensure that this process is fast. Typically, it is easy to determine the probability of success of each random column draw as a function of Q so as to ensure a constant probability (say at least 1/3) that each column draw will be successful.

Once the Q sets of rows and Q sets of columns are drawn, these rows and columns are randomly and uniformly distributed to the different rows and columns of the matrix. By construction, each tile at the intersection of a row and a column can be associated with a non-empty set (because of the column selection procedure) of candidate processors.

To allocate the tiles of the matrix to the processors, we proceed as follows: all the tiles with a single candidate processor are allocated to this processor. Then the tiles are considered by decreasing weight and each tile is, in turn, given to the one of its candidate processors with the smallest load at that moment. Details and formal algorithm can be found in [11].

Compared to BC and BCE, RS is expected to produce better load balancing. On the other hand, the distribution of tiles allocated to a processor is not a priori regular in the matrix, contrarily to pattern based allocations, which is likely to cause load balancing problems over time, for factorizations such as LU in which the set of available tasks is not available from the beginning.

V. EXPERIMENTS

A. Test Cases

The purpose of this study is to investigate and to compare different data distribution strategies to balance the load and to minimize the makespan for different kernels of linear algebra when the matrices are compressed using the BLR format. It is difficult to make available matrices corresponding to actual electromagnetism problems and in this study we will therefore rely on synthetic matrices, which nevertheless reproduce the main characteristics of the actual matrices, and which allow us to validate the approaches over a larger range of parameters and to ensure the reproducibility of the experiments. A matrix is thus characterized, in what follows, by its size (in number of blocks) and by the data of $d_{i,j}$, where $d_{i,j}$

represents the density of tile $A_{i,j}$ (i.e. the rank of $A_{i,j}$ is $r \times n_b$).

The generation of test cases therefore requires the definition of a density distribution of the tiles according to their position in the matrix and in particular on their distance from the diagonal. The following process is used.

(i) All diagonal tiles are associated with $d = 1$ which means that diagonal tiles are assumed to be non-compressible (full rank).

(ii) $\forall(i, j), d_{i,j} = \max(\min(v(i, j) + \gamma, 1), 0)$ where $v(i, j)$ follows an exponential distribution as a function of the distance to the diagonal $v(i, j) = \exp^{-\frac{\delta}{2} * (\frac{i-j}{N-1})^2}$ and γ is a noise that follows a normal distribution $\gamma \sim \mathcal{N}(0, \frac{1}{20})$.

(iii) In addition to diagonal tiles, a number η of tiles are assumed to be full rank. They are located at uniformly randomly in the matrix, and η follows a normal distribution $\eta \sim \mathcal{N}(\sqrt{N}, \frac{\sqrt{N}}{2})$.

B. Preliminary Results

The performance of the three strategies described in Section IV has been evaluated on numerous generated linear systems for several values of (N, P, α) parameters (results for different values of δ can be found in [11] but results are very similar) and different criteria: maximum load (dots) and simulated execution time without communications (triangles) and maximum execution time with communications (squares). All displayed results are normalized against a trivial lower bound: if W denotes the overall work, the load of each processor and the makespan are divided by W/P . A subset of numerical results can be found below and more are available (with consistent results) in [11].

Results show that both *BCE* and *RS* lead to better solutions than plain *block cyclic* method in specific regions of the parameters space: (details). A meta-strategy based on selecting the solution with minimum maximum load among all three methods is therefore a significant improvement over *block cyclic* and over each method alone. Such a strategy provides a good quality *mapping* for almost any combination of parameters.

Results for $\delta = 8$ (different values for δ can be found in [11]) are depicted for Matrix Multiplication in Figure 2 and LU factorization in Figure 3. In both figures, we choose values for P such that $\exists r, r(r-1) = P$ so that we can concentrate on the load balancing itself and not on rounding errors. Note that this situation favors BC (red) and BCE (purple) with respect to RS (green). All displayed results are normalized with respect to

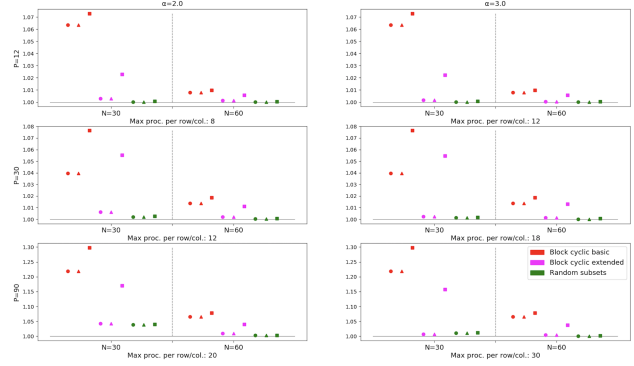


Fig. 2: Results for Matrix Multiplication

ideal load balancing where all processors would receive exactly the same load.

From Figure 2, we can draw some interesting conclusions. Since the matrix product consists of a set of independent tasks, there is in general no difference between pure load balancing (the dots) and simulations without communications (the triangles). When $\alpha = 1$, i.e. when using a minimum number of processors per row and per column, the difference with simulations with communication (the squares) is not significant but it becomes so as soon as α becomes larger ($\alpha = 2$ or 3), especially in the case of BCE. For the matrix product, the best heuristic is clearly RS for all values of P and N , provided that α is large enough. This observation is true even though the chosen P values benefit BC and BCE. With $\alpha = 3$, for all the configurations tested here, the value of the makespan with communication is indeed always less than 1% of the value that would be obtained without communication and with optimal load balancing.

The conclusions that can be drawn from Figure 3 are quite similar, except that the dependencies between tasks generate a difference between dots and triangles. For the rest, we can notice that load balancing is less strong for LU than for MM when using BC (red). This situation is related to the fact that the load associated with the tiles is less homogeneous (the weight of a tile increases with i and j), which makes load balancing more difficult. In this context, both BCE and RS are strategies of choice for data distribution and both give excellent results. Nevertheless, the performance degradation between pure load balancing and makespan without communications is more important in the case of RS than in the case of BCE. This can be explained by the fact that the tiles assigned to a given processor are always distributed regularly by construction with BCE, whereas

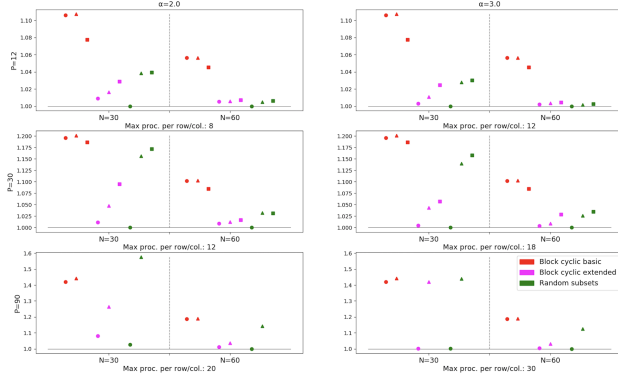


Fig. 3: Results for LU factorization

this property is not automatically met with RS, which can lead, at certain instants, to an overload for one of the processors, which in turn can slow down the whole process. For example, as soon as $\alpha = 3$, BCE returns allocations that are within 5% of the lower bound. In all cases, we can observe that BCE’s behavior is very robust, as soon as α is greater than two. BCE generally performs at least as well as RS, which is a more expensive and sophisticated heuristic. In Section VI, our goal is to understand the performance of BCE from a more specific study of load balancing.

VI. ANALYSIS OF BCE ALGORITHM

A. Why does BCE balance the load well?

In this section, our objective is to understand and analyze the behavior of BC and BCE when the tile weight distribution corresponds to the case of the matrix multiplication (Figure 4). In this case, the tile weights in the matrix depend only on their distance to the diagonal. The case of LU factorization is shown in the companion research report [11] but does not exhibit any significant difference. We consider here different sizes of matrices ($N = 30, 60, 90$), different values of the number of processors ($P = 12, 30, 90$) and different values of $\alpha = 1, 2, 3$, where $\alpha\sqrt{P}$ is the maximum number of different processors that can be present on the same row or column. $\alpha = 1$ corresponds to BC, whereas $\alpha = 2, 3$ correspond to two different variants of BCE.

Our goal is to measure the quality of the load balancing between the processors for pattern based distributions BC and BCE. Blue points are associated to grid tiles, *i.e.* the overload of each virtual processor, whereas red points are associated to actual processors. When $\alpha = 1$ red and blue are the same since the numbers of virtual

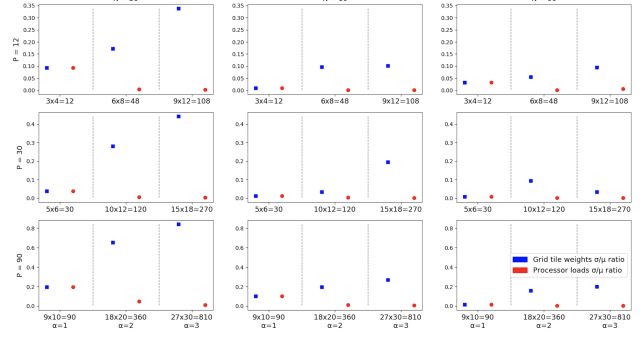


Fig. 4: BCE load balancing with MM tiles

and real processors are the same whereas the number of virtual processors is α^2 times higher in general. Each point represents the dispersion of the load of virtual processors (blue points) or real processors (red points), where the dispersion of processor loads is defined as

$$\frac{\text{standard deviation of the distribution of loads}}{\text{mean value of the distribution of loads}}$$

so that a lower value represents a better load balancing (a ratio of 0 corresponds to perfect load balancing).

The first conclusion that can be drawn is related to the observation of the blue dots on any cell for $\alpha = 1, 2, 3$. We can notice that the first phase of tile aggregation following the regular pattern is efficient enough to limit the dispersion and that the dispersion is much greater when the number of virtual processors is increased (and therefore the number of tiles allocated to each virtual processor is decreased). The second conclusion is related to the observation of the relative values of the blue and red dots when $\alpha = 2, 3$. It shows that the second phase of packing virtual processors into BCE is very efficient. Indeed, we can observe that the dispersion is almost null as soon as we group the virtual processors by groups of 4 ($\alpha = 2$) or 9 ($\alpha = 3$) virtual processors.

The intuition is as follows. If we use for example $\alpha = \sqrt{2}$, we will obtain $2P$ virtual processors, whose respective loads are already rather homogeneous since each virtual processor corresponds to the aggregation of a certain number of tiles. In this case, the packing algorithm results in the most loaded k -th virtual processor being matched with the least loaded k -th virtual processor inside the real k -th processor, resulting in a very homogeneous distribution of weights. We have tried to establish this result theoretically but this result is in fact notoriously difficult. Indeed, even if we assume that the initial weight distribution is known and simple

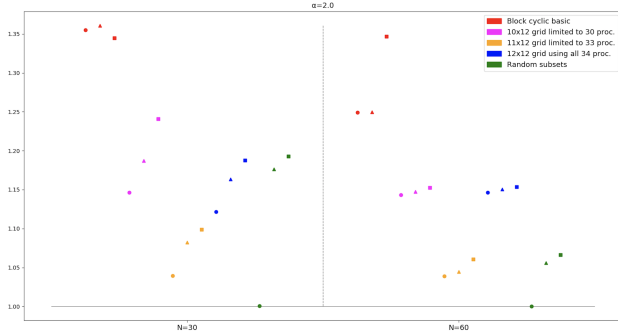


Fig. 5: Comparison of different strategies with $P = 34$

(uniform or normal for example), then the distribution of the mean value and standard deviation for the k -th element are only known by approximate and non-closed formulas [6]. This is why we relied on simulations to establish these properties.

B. Limitations of BC and BCE solutions

For Figures 2 and 3, the number of processors P used the experiments have been carefully chosen to be written as $r(r-1)$ so that the rectangular pattern allows all processors to be used. This is not the case for arbitrary values of P and we are faced with the problem of finding a rectangular grid $r \times c$, with (i) r and c close together so that the grid is not too long and the number of different processors per row or column is not too large, and (ii) $rc \leq P$ but close to P so as not to waste resources. In practice, these two constraints are not easy to fulfill and their simultaneous implementation naturally leads to choosing a number of processors that is smaller than the number of processors actually available. In this case, the RS strategy proves to be very efficient, because it is agnostic of any property on P and its efficiency does not depend on the value of P .

For example, let us consider the situation described in Figure 5 where the number of processors is 34 and the maximum number of processors that can appear in the same row or column is 12 (which corresponds to $\alpha = 2$). In the case of BCE, several pattern sizes are possible. First of all, one can choose a 12×12 grid, which leads to 144 virtual processors to be mapped onto 34 real processors, thus 4 or 5 virtual processors. With the law of large numbers, the virtual processors have quite similar loads, so there is a risk of load imbalance. Another option is to create a grid of size 11×12 , so $132 (= 33 \times 4)$ virtual processors that we can allocate onto 33 real processors (we sacrifice 1) by giving each

real processor 4 virtual processors. We can also make a 12×12 grid and allocate only 30 real processors. One BCE intrinsic difficulty is that all these solutions are reasonable and it is not easy a priori to arbitrate between them. In practice, Figure 5 shows that for our parameters, the best is to use a 11×12 grid and 33 processors. On the other hand, the RS solution does not require any special adjustment and leads to a solution that is still quite significantly superior to BCE solutions in terms of load balancing. In the case of LU factorization, the 11×12 nevertheless provides a slightly better makespan. It is therefore a more flexible, generic and efficient solution in the general case, if one wishes to use all available resources.

VII. CONCLUSION AND PERSPECTIVES

We consider the problem of data distribution in the context of compressed matrices in BLR format for linear algebra kernels such as the matrix product and LU factorization. In the compressed case, the load associated with each tile is no longer homogeneous more sophisticated strategies than the classical 2D Block Cyclic (BC) have to be implemented in order to balance the load throughout the computation. We propose and analyze two new strategies. The BCE strategy is an extension of BC, which allows better load balancing at the cost of a slightly higher, but controlled, number of processors per row and per column. The RS strategy is a randomized strategy that generates row and column types before performing the allocation. Both of these strategies significantly improve the results compared to BC. BCE proves to be particularly efficient for certain P values, which we were able to analyze and justify using order statistics. RS, on the other hand, has the advantage of giving very interesting results regardless of the value of P . This work opens several perspectives. First of all, one can try to improve the RS strategy so that it better takes into account the specific dependencies of the application's tasks, in order to balance the load throughout the computation. It may also be possible to hybridize the different strategies by adding a randomized component to BCE so that it can run on any number of processors.

REFERENCES

- [1] ABDULAH, S., LTAIEF, H., SUN, Y., GENTON, M. G., AND KEYES, D. E. Geostatistical modeling and prediction using mixed precision tile cholesky factorization. *HIPC* (2019), 152–162.

- [2] AGULLO, E., AUMAGE, O., FAVERGE, M., FURMENTO, N., PRUVOST, F., SERGENT, M., AND THIBAUT, S. P. Achieving high performance on supercomputers with a sequential task-based programming model. *IEEE Transactions on Parallel and Distributed Systems* (2017).
- [3] AGULLO, E., BEAUMONT, O., EYRAUD-DUBOIS, L., HERMANN, J., KUMAR, S., MARCHAL, L., AND THIBAUT, S. Bridging the gap between performance and bounds of cholesky factorization on heterogeneous platforms. In *IPDPSW* (2015), IEEE, pp. 34–45.
- [4] AGULLO, E., BEAUMONT, O., EYRAUD-DUBOIS, L., AND KUMAR, S. Are static schedules so bad? a case study on cholesky factorization. In *IPDPS* (2016), IEEE, pp. 1021–1030.
- [5] AMINFAR, A., AMBIKASARAN, S., AND DARVE, E. A fast block low-rank dense solver with applications to finite-element matrices. *Journal of Computational Physics* 304 (2016), 170–188.
- [6] ARNOLD, B. C., BALAKRISHNAN, N., AND NAGARAJA, H. N. *A first course in order statistics*. SIAM, 2008.
- [7] AUGONNET, C., GOUDIN, D., KUHN, M., LACOSTE, X., NAMYST, R., AND RAMET, P. A hierarchical fast direct solver for distributed memory machines with manycore nodes. Research report, CEA/Total; Université de Bordeaux, Oct. 2019.
- [8] AUGONNET, C., THIBAUT, S., NAMYST, R., AND WACRENIER, P.-A. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009 23* (Feb. 2011), 187–198.
- [9] BADIA, R. M., HERRERO, J. R., LABARTA, J., PÉREZ, J. M., QUINTANA-ORTÍ, E. S., AND QUINTANA-ORTÍ, G. Parallelizing dense and banded linear algebra libraries using SMPs. *Concurrency and Computation: Practice and Experience* 21, 18 (2009), 2438–2456.
- [10] BEAUMONT, O., BECKER, B. A., DEFLUMERE, A., EYRAUD-DUBOIS, L., LAMBERT, T., AND LASTOVETSKY, A. Recent advances in matrix partitioning for parallel computing on heterogeneous platforms. *IEEE Transactions on Parallel and Distributed Systems* 30, 1 (2018), 218–229.
- [11] BEAUMONT, O., LIONEL, E.-D., AND VERITE, M. 2D Static Resource Allocation strategies for load balancing in for Compressed Linear Algebra and Communication Constraints. working paper or preprint, July 2020.
- [12] BERGMAN, D., CARDONHA, C., AND MEHRANI, S. Binary decision diagrams for bin packing with minimum color fragmentation. In *CPAIOR* (2019), Springer, pp. 57–66.
- [13] BEZIAU, P. Data distribution strategies for cholesky decomposition. In *Compas* (2019).
- [14] BLACKFORD, L. S., CHOI, J., CLEARY, A., D’AZEVEDO, E., DEMMEL, J., DHILLON, I., DONGARRA, J., HAMMARLING, S., HENRY, G., PETITET, A., STANLEY, K., WALKER, D., AND WHALEY, R. C. *ScaLAPACK Users’ Guide*. Society for Industrial and Applied Mathematics, 1997.
- [15] BOSILCA, G., BOUTEILLER, A., DANALIS, A., FAVERGE, M., HAIDAR, A., HERAULT, T., KURZAK, J., LANGOU, J., LEMARINIER, P., LTAIEF, H., LUSZCZEK, P., YARKHAN, A., AND DONGARRA, J. Flexible development of dense linear algebra algorithms on massively parallel architectures with dplasma. In *IPDPSW* (May 2011), pp. 1432–1441.
- [16] BOSILCA, G., BOUTEILLER, A., DANALIS, A., HERAULT, T., LEMARINIER, P., AND DONGARRA, J. DAGuE: A generic distributed dag engine for high performance computing. *Parallel Computing* 38, (1-2) (2012), 37–51.
- [17] BUTTARI, A., LANGOU, J., KURZAK, J., AND DONGARRA, J. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing* 35, 1 (2009), 38 – 53.
- [18] CAO, Q., PEI, Y., HERAULT, T., AKBUDAK, K., MIKHALEV, A., BOSILCA, G., LTAIEF, H., KEYES, D., AND DONGARRA, J. Performance analysis of tile low-rank Cholesky factorization using parsec instrumentation tools. In *ProTools* (2019), ACM.
- [19] CARRATALÁ-SÁEZ, R., FAVERGE, M., PICHON, G., SYLVAND, G., AND QUINTANA-ORTÍ, E. Tiled algorithms for efficient task-parallel h-matrix solvers. In *PDSEC* (2020).
- [20] CASANOVA, H., LEGRAND, A., AND QUINSON, M. Simgrid: A generic framework for large-scale distributed experiments. In *Tenth International Conference on Computer Modeling and Simulation (uksim 2008)* (2008), IEEE, pp. 126–131.
- [21] COJEAN, T., GUERMOUCHE, A., HUGO, A., NAMYST, R., AND WACRENIER, P.-A. Resource aggregation for task-based cholesky factorization on top of modern architectures. *Parallel Computing* 83 (2019), 73–92.
- [22] EPSTEIN, L., IMREH, C., AND LEVIN, A. Class constrained bin packing revisited. *Theoretical Computer Science* (2010).
- [23] FAVERGE, M., PICHON, G., RAMET, P., AND ROMAN, J. On the use of h-matrix arithmetic in pastix: a preliminary study. In *Workshop on Fast Solvers, Toulouse, France* (2015).
- [24] GOLUB, G. H., AND VAN LOAN, C. F. *Matrix computations*, vol. 3. JHU press, 2012.
- [25] GRASEDYCK, L., AND HACKBUSCH, W. Construction and arithmetics of h-matrices. *Computing* 70, 4 (2003), 295–334.
- [26] GRIGNI, M., AND MANNE, F. On the complexity of the generalized block distribution. In *International Workshop on Parallel Algorithms for Irregularly Structured Problems* (1996), Springer, pp. 319–326.
- [27] HACKBUSCH, W. A sparse matrix arithmetic based on h-matrices. *Computing* 62, 2 (1999), 89–108.
- [28] IRONY, D., TOLEDO, S., AND TISKIN, A. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.* 64, 9 (Sept. 2004), 1017–1026.
- [29] JACQUELIN, M., ZHENG, Y., NG, E., AND YELICK, K. An asynchronous task-based fan-both sparse cholesky solver. *arXiv preprint arXiv:1608.00044* (2016).
- [30] JANSEN, K., LASSOTA, A., AND MAACK, M. Approximation algorithms for scheduling with class constraints. *arXiv preprint arXiv:1909.11970* (2019).
- [31] KALINOV, A., AND LASTOVETSKY, A. Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers. *Journal of Parallel and Distributed Computing* 61, 4 (2001), 520–535.
- [32] KURZAK, J., LTAIEF, H., DONGARRA, J., AND BADIA, R. M. Scheduling dense linear algebra operations on multicore processors. *Concurrency and Computation: Practice and Experience* 22, 1 (2010), 15–44.
- [33] MARY, T. *Block Low-Rank multifrontal solvers: complexity, performance, and scalability*. PhD thesis, 2017.
- [34] PÉREZ, J. M., BADIA, R. M., AND LABARTA, J. A flexible and portable programming model for SMP and multi-cores. Tech. rep., Barcelona Supercomputing Center, 2007.
- [35] PICHON, G., DARVE, E., FAVERGE, M., RAMET, P., AND ROMAN, J. Sparse supernodal solver using block low-rank compression: Design, performance and analysis. *Journal of computational science* 27 (2018), 255–270.
- [36] SOLOMONIK, E., AND DEMMEL, J. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *EuroPar* (2011), Springer.
- [37] SONG, F., YARKHAN, A., AND DONGARRA, J. Dynamic task scheduling for linear algebra algorithms on distributed-memory multicore systems. In *SC’09* (2009).
- [38] YAMAZAKI, I., IDA, A., YOKOTA, R., AND DONGARRA, J. Distributed-memory lattice h-matrix factorization. *The International Journal of High Performance Computing Applications* 33, 5 (2019), 1046–1063.
- [39] YAŞAR, A., AND ÇATALLYÜREK, Ü. V. Heuristics for symmetric rectilinear matrix partitioning. *arXiv preprint arXiv:1909.12209* (2019).