Data Distribution Schemes for Dense Linear Algebra Factorizations on Any Number of Nodes

Olivier Beaumont, Jean-Alexandre Collin, Lionel Eyraud-Dubois, Mathieu Vérité

Data Distribution Schemes for Dense Linear Algebra Factorizations on Any Number of Nodes

Olivier Beaumont
Inria Center of the University of Bordeaux
Bordeaux, France
Olivier.Beaumont@inria.fr

Jean-Alexandre Collin
University of Bordeaux
Bordeaux, France
jean-alexandre.collin@inria.fr

Lionel Eyraud-Dubois
Inria Center of the University of Bordeaux
Bordeaux, France
Lionel.Eyraud-Dubois@inria.fr

Mathieu Vérité
Inria Center of the University of Bordeaux
Bordeaux, France
Mathieu.Verite@inria.fr

***Abstract*—In this paper, we consider the problem of distributing the tiles of a dense matrix onto a set of homogeneous nodes. We consider both the case of non-symmetric (LU) and symmetric (Cholesky) factorizations. The efficiency of the well-known 2D Block-Cyclic (2DBC) distribution degrades significantly if the number of nodes P cannot be written as the product of two close numbers. Similarly, the recently introduced Symmetric Block Cyclic (SBC) distribution is only valid for specific values of P. In both contexts, we propose generalizations of these distributions to adapt them to any number of nodes. We show that this provides improvements to existing schemes (2DBC and SBC) both in theory and in practice, using the flexibility and ease of programming induced by task-based runtime systems like `Chameleon` and `StarPU`.**

***Index Terms*—Dense Linear Algebra, LU factorization, Cholesky Factorization, Task-based Runtime Systems, Data Distribution Schemes**

I. Introduction

We consider the problem of allocating a dense matrix to a set of homogeneous compute nodes to perform linear algebra kernels such as LU and Cholesky factorizations. We assume in the following that the matrix is split into regular tiles (or blocks) having all the same size and that these tiles are allocated to nodes. In the case of 2D distributions, we generally use the owner compute rule, which assigns all updates to the node in charge of the associated tile. The goal is then to find a distribution that (i) is load balanced, i.e., in which the amount of work associated with each node is approximately equal over the factorization, and (ii) minimizes the data exchanges induced by the distribution.

In recent years, a lot of work has been done to understand how to distribute a matrix over a set of heterogeneous computational resources, which is a challenging problem. Optimal solutions for a small number of resources have been found, several sophisticated approximation algorithms have been developed, and efficient low cost heuristics for the larger cases have been proposed. The dual problem, in which the matrix induces heterogeneous costs, for example because it is given in Block Low Rank (BLR) form with a compression level that depends on the numerical properties of the block itself, have also been proposed, in the sparse as well as in the dense case. A review of previous work and existing techniques for those use cases, either heterogeneous resources or workload, is presented in Section II-B.

The most simple problem of allocating a dense matrix on a homogeneous set of nodes has not received much attention in the last few years, probably because the 2DBC (2-Dimensional Block Cyclic) distribution seems to provide an efficient, simple and easy to implement solution in regular algorithms implemented using MPI and collective communications. The 2DBC pattern has indeed several qualities, especially when there are $P = r^2$ resources: (i) it is perfectly balanced since each node appears exactly once, (ii) it allows to minimize the communications since in each row and each column, exactly $\sqrt{P}$ nodes appear, which is optimal (see Section II-A for a survey on the bounds) and (iii) finally, it is a small pattern, the smallest in which each node appears, which makes it a good candidate for matrices with few blocks and problems in which the size of the trailing matrix decreases over time, such as in LU and Cholesky factorizations.

A first challenge to 2DBC was brought by the development of 3D and 2.5D algorithms [1], [2]. These algorithms are based on a 3D representation of dense linear algebra algorithms whose complexity is in $O(N^3)$ where N is the size of the matrix (products, dense factorizations). It has been shown that it is possible to generate a smaller communication volume while keeping perfect load balancing, using 3D distribution schemes more complex than 2DBC. In general, this minimization of communications induces an increase in memory consumption by requiring a replication of some data on several nodes. Nevertheless, the 3D and 2.5D solutions are slice-based and in each slice, a 2DBC distribution is used.

A second challenge to 2DBC has recently been brought in the case of symmetric problems, such as the multiplication of symmetric matrices (SYRK) or the Cholesky factorization (see Section II-A). In this context, it has been shown that it is possible to design specialized data distribution schemes such as SBC [3] (Symmetric Block Cyclic distribution) which generate strictly less communication than 2DBC, even among

2D data distributions. These communication schemes are typically based on a cyclic distribution, but with a larger pattern in which each node can appear several times.

In addition, the development of task based approaches, available in OpenMP (through the `task` directive and the `depend` clause) and in runtime systems like `StarPU` [4], OmpSs [5] or PaRSEC [6], provides significant simplification of the implementation of linear algebra kernels, even in the context of very irregular distributions. Indeed, in practice, it is enough to specify which node is in charge of which tile. From this information, communications are automatically managed by the runtime, with general purpose strategies to overlap as much as possible communications with computations. Thus, it has been demonstrated [3] that significant increased performance can be obtained when using complex data allocation patterns, compared to those obtained with 2DBC, without significant effort for the implementation. Moreover, the use of the runtime allows to avoid synchronizations between the different steps of a LU or Cholesky factorization, and thus allows to obtain much better performance than MPI-based codes that are usually more synchronized due to implementation issues. Details about the task-based execution model and the runtime system, `StarPU`, used in this work are provided in Section II-C.

In this paper, we are interested in another limitation of classical block cyclic distribution, which is shared by 2DBC and SBC and which is related to the number of nodes used to carry out a computation. As mentioned above, the 2DBC grid produced for $P = r^2$ nodes has many qualities (load balancing, communication minimization, pattern size). The situation is more complex when for example 23 nodes are available. A 23×1 grid is very unbalanced and the communications on the rows requires sending messages to 22 other nodes. A classical solution is to use 22 nodes instead (in a 11×2 grid which is still unbalanced) or 21 (7×3 grid, which is better) or even only 20 (5×4 grid). We provide on Figure 1 performance results obtained with the `Chameleon` library [7], built on top of the `StarPU` runtime system [4], for the LU factorization. These results show that as expected, the performance per node (Figure 1) increases when the pattern becomes closer to a square. However, these performance gains per node are limited by the fact that fewer nodes are used, so that all these solutions obtain similar results in terms of time to solution, or equivalently, total performance, as shown on Figure 1. In practice, being able to run a factorization on any number of nodes is of great practical interest. Indeed, for example, the platform on which we are doing our experiments contains 44 nodes each with 36 *Intel Xeon Skylake Gold 6240* cores. Given already present reservations, it is common that the number of available nodes is not of the form $P = r^2$. In general, in this case, users reserve fewer nodes than are actually available, as in the example above, but in the form $P = r \times c$ where r and c are of the same order of magnitude. In the following, we show that it is possible to use any number of nodes without sacrificing on the efficiency per node.

The goal of this paper is to build distribution patterns that

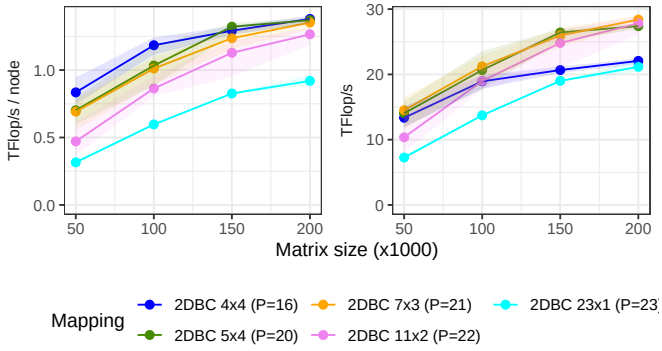


Fig. 1: Sample results for LU factorization of matrix A of size $m \times m$, $m = 50,000$ to $200,000$, distributed using 2DBC with different pattern shapes.

can be used on an arbitrary number of nodes, while keeping good properties in terms of communications. Our approach is to build larger patterns, in which each node may appear several times. This allows us to explore a larger solution space than plain 2DBC, and we propose several new solutions, in both the non-symmetric and symmetric contexts.

We first study the non-symmetric case, where we propose a systematic way to construct, for all possible values of P, a perfectly balanced pattern of size $b(b-1) \times P$, where $b = \lceil (P/\lceil \sqrt{(P)} \rceil) \rceil$ (so that $b(b-1)$ is of order P), in which $\left\lceil \sqrt{P} \right\rceil$ nodes appear in each row and in each column. This pattern is therefore optimal in terms of load balancing (each node appears exactly $P-1$ times in the pattern) and in terms of communications (any pattern on P nodes require at least $\left\lceil \sqrt{P} \right\rceil$ nodes per row and per column). The pattern is larger in general than the 2DBC pattern, but it is valid for all values of P.

We then consider the case of distributions tailored to symmetric problems. The SBC distribution (Symmetric Block Cyclic) was introduced recently [3]. It is valid for specific values of P, of the form either $a^2/2$ for an even integer a, or $a(a-1)/2$ for any integer a. SBC induces a communication volume lower by a factor of $\sqrt{2}$ than 2DBC, but however remains within a factor of $\sqrt{2}$ of the lower bound [8]. In this paper, we propose an extension of this symmetric distribution to all possible values of P. Unlike for the non-symmetric case, we failed to build an elegant, optimal and valid distribution scheme for all P. Instead, we propose GCR&M, a greedy-based heuristic that builds an efficient pattern for a given P and a pattern size $S \times S$. The solutions found are not optimal, but yield better performance than SBC on average, with the additional benefit that GCR&M is able to provide solutions for all values of P.

The rest of this paper is organized as follows. In Section II, we survey the literature on lower bounds on communications induced by different linear algebra kernels, on task based runtime systems and on data distributions in the dense case and the sparse case, with homogeneous or heterogeneous

resources. In Section III we introduce the model and notations used to evaluate the quality of data distributions and guide the design of more efficient ones, regarding total running time. In Section IV, we show how to build, in the non-symmetric case, a generic data distribution scheme that is valid for any number of nodes, while providing an optimal overall communication volume. An experimental evaluation using the `Chameleon` library shows that the theoretical reduction of the volume of communications induced by our distributions actually translates into a reduction of the execution time of the factorization operation. In Section V, we consider the symmetric case, and we propose an algorithm that, despite not being optimal, allows us to build efficient distribution schemes for any number of nodes. Finally, we provide concluding remarks in Section VI.

II. Related work

A. Lower Bounds on Communications

Research work regarding theoretical communication lower bounds for linear algebra operations deals both with the *two-levels memory* setting, where the machine considered features one fast and limited memory of size M and one slow unlimited memory, and the parallel setting. All bounds obtained for sequential schedule in the first setting can then be extended to the parallel case assuming that data is fairly distributed among all nodes, *i.e.* $M = \mathcal{O}(\frac{m^2}{P})$ for a matrix of size $m \times m$.

The first results on communications lower bound were obtained for classical matrix multiplication, $C = A \cdot B$, by Hong and Kung in [9] in the *two-levels memory* setting: the volume of communications between the two levels of memory is $\Omega(\frac{mnk}{\sqrt{M}})$ where matrices A, B and C are of respective size $m \times k$, $k \times n$, $m \times n$. This result was extended to the parallel case by Irony *et al.* [10]: assuming a fair distribution of data among nodes, the minimum volume of communications per node to perform matrix multiplication is $\Omega(\frac{m^2}{\sqrt{P}})$. They proved that, under the same assumption, using the 2DBC distribution and the *owner computes* rule, as implemented in the `ScaLAPACK` library [11], is asymptotically optimal. Furthermore, the volume of communication per node is actually minimum, equals to $\frac{2m^2}{\sqrt{P}}$, if P is a perfect square. The results regarding the lower bounds were then extended to Cholesky factorization in [12] and later to almost all direct linear algebra applications in both dense and sparse case in [13]: for such operations, the minimum volume of communication required is $\Omega(\frac{\#\text{ arithmetic operations}}{\sqrt{M}})$.

More recent approaches manage to provide an explicit coefficient for the dominant term of lower bound formulas. In [14] Olivry *et al.* develop a tool called `IOLB`, that performs automatic DAG analysis to derive lower bounds on communications. It improves bounds for several kernels in the two-levels memory setting:

- $\frac{mnk}{\sqrt{M}}$ for GEMM operation ($C = A \cdot B$ with A $m \times k$ and B $k \times n$)
- $\frac{1}{2}\frac{m^2n}{\sqrt{M}}$ for SYRK operation ($C = A \cdot A^T$ with A $m \times n$)
- $\frac{1}{3}\frac{m^3}{\sqrt{M}}$ for LU factorization ($A = L \cdot U$ with A $m \times m$)
- $\frac{1}{6}\frac{m^3}{\sqrt{M}}$ for Cholesky factorization ($A = L \cdot L^T$ with A $m \times m$)

Another work from Kwasniewski *et al.* [2] use explicit enumeration of data reuse to derive an improved bound for LU factorization: it requires at least $\frac{2}{3}\frac{m^3}{\sqrt{M}}$ communications for a matrix of size $m \times m$. They also present a parallel algorithm, `COnfLUX`, based on 2DBC distribution. Under the assumption of fair distribution of data among nodes, this algorithm is optimal: it generates no more than $\frac{m^2}{\sqrt{P}} + \mathcal{O}(\frac{m^2}{P})$ communications per node.

Regarding symmetric kernels, improved bounds for the *two-levels memory* setting are presented in [8]: the authors show that SYRK operation requires at least $\frac{1}{\sqrt{2}}\frac{m^2n}{\sqrt{M}}$ and Cholesky $\frac{1}{3\sqrt{2}}\frac{m^3}{\sqrt{M}}$. Sequential algorithms matching those bound are also presented. In the parallel setting, an improved distribution based on the same ideas called SBC is developed in [3] which take advantage of the symmetry of the input to reduces the communications. This cyclic distribution ensures a good load balancing and is shown to outperform the 2DBC distribution for Cholesky factorization and SYRK operation. However it does not match the communication lower bounds and is only available for specific values of the number of nodes: $P = \frac{r^2}{2}$ or $P = \frac{r(r-1)}{2}$.

Simultaneous development was carried out to deal with the case where extra memory is available. Then multiple replications of the input matrix can be used. The 3D version of 2DBC distribution has been proven optimal regarding communications for matrix multiplications in [10]. In [15] Solomonik *et al.* detail the continuum between 2D and 3D distributions and present 2.5D algorithms for matrix multiplication and LU factorization, both based on replicated 2DBC distributions, that are asymptotically optimal. `COnfLUX` and `COnfCHOX` [2] can also make use of data replication and be extended to 2.5D versions.

B. Heterogeneous Resources

The problem of designing alternative placement strategies has been considered in the case of heterogeneous resources. In this framework, in the most general situation, we are given P resources of relative speeds $v_1, \ldots, v_P$ and the goal is to allocate tiles to these different resources in a way that balances the load (each node receives a number of tiles proportional to its speed) and minimizes the volume of communications.

To minimize communications in the context of matrix product, the original matrix is partitioned into P rectangles, whose areas are proportional to the relative speeds of processors, and such that the sum of all rectangle perimeters is minimal. Indeed, using Cannon-type algorithms for multiplying matrices, the volume of data exchanged by a node at each step is proportional to its perimeter. This approach was initially proposed in [16] and [17]. It led to developments in two orthogonal directions, one concerned with solving the optimal problem for a small number of resources [18]–[20] and the other concerned with the design of approximation

algorithms [21]–[23]. A survey on the results obtained with these different approaches has been proposed in [24].

Nevertheless, partitioning algorithms focus on the case of the product of matrices and do not consider symmetric problems and factorizations. Moreover, because of the additional difficulty introduced by heterogeneity, they usually have to rely on approximation algorithms that perform worse in the homogeneous case than 2DBC distributions.

A very similar problem arises when trying to balance heterogeneous workloads among identical resources. Such situations occur when considering compressed or sparse matrices. In those case, the 2DBC distribution is again the *de facto* standard although several improvements are actively developed to overcome its limitations regarding load balancing. Those solutions are based on alternative distributions [25], making use of mixed-precision arithemtic [26] or data sparsity [27].

C. Task-based Execution Model

To perform linear algebra operations in a parallel and distributed way, several recent libraries rely on the task-based execution model, provided by OpenMP (through the `task` directive and the `depend` clause) and by runtime systems such as `StarPU` [4], `OmpSs` [5] or PaRSEC [6]. In this model, the application is responsible for allocating the computations while the management of data and task dependencies during the execution is delegated to a dedicated runtime system. This provides a high level of abstraction to the end user and flexibility to the libraries developed according to this model. At the platform level, only the distribution of data among nodes must be provided. This allows to easily implement any data distribution, as irregular as it may be. Besides, during the execution, it removes the need of synchronizations used to ensure the correctness of the algorithm. This allows a better overlap of inter-node communications with computations and results in higher performance.

In this paper, we use the `Chameleon` [7] linear algebra library which provides tiled implementations of many classical dense operations, including LU and Cholesky factorizations. `Chameleon` follows the task-based execution model: the computations are abstracted as a Directed Acyclic Graph (DAG) of tasks whose dependencies are handled automatically by the underlying runtime system. The distribution of tasks to nodes follows the *owner computes rule*, *i.e.* each node performs all the tasks modifying the tiles it owns. For a given data distribution, when performing an operation, the `Chameleon` library submits the corresponding tasks to the runtime system along with the access mode (read and/or write) associated to each input and output tile. From this description of data dependencies, the runtime system then infers task dependencies and the necessary inter-node communications. Though it can be backed by several runtime systems, we use the `Chameleon` library in combination with `StarPU`. At the level of one node, `Chameleon` and `StarPU` apply dynamic scheduling strategies to distribute the tasks among workers, and are thus able to manage heterogeneous ressources (CPU cores and GPUs). Each elementary task assigned to a node is performed sequentially by a worker (a CPU core or a GPU).

The current implementation of `Chameleon` library does not make use of complex collective communication schemes: each inter-node communication uses a point-to-point MPI communication operation, *i.e.* each tile is sent to its destination as a separate message. Hence, the total number of messages is proportional to the communication volume. Besides, as computations and communications take place asynchronously, the latency is overlapped with computations. Therefore, we use the volume of communication rather than the total number of messages as the value of interest when analyzing data distributions.

III. Model and Notations

In this work, we design unusual and irregular distribution patterns, with the goal to build patterns that reduce the communication volume for LU and Cholesky factorizations. In this section, we analyze the communication volume induced by an arbitrary pattern when replicated onto the entire matrix to define a *communication cost* metric for an arbitrary pattern.

Let us assume that a $m \times m$ matrix A is distributed by following a pattern $\mathcal{G}$, of dimension $r \times c$. To avoid any ambiguity, we use *tile* to denote a position in the matrix, and *cell* to denote a position in a pattern. A pattern completely defines the data distribution of the matrix: for all $(i, j) \in \{1, \ldots, m\}^2$, the tile at position (i, j) is owned by the node which is present in the cell $(i \mod r, j \mod c)$ in $\mathcal{G}$. Then the set of nodes in a row or column of A is the same as the set of nodes in the corresponding row or column of $\mathcal{G}$.

A. Case of LU factorization

To complete the analysis, we denote by (i) x_i the number of different nodes on row $1 \leq i \leq r$ and by (ii) y_j the number of different nodes on column $1 \leq j \leq c$.

First, let us analyze the communications scheme in the case of the LU factorization $A = L \cdot U$ where A is $m \times m$ and L and U are respectively lower and upper triangular matrices. We assume that A is distributed according to a cyclic distribution using P nodes. Its values are overwritten by L and U during the operation. An illustration with a 2DBC distribution with $m = 12$ and $P = 6$ is depicted on the left of Figure 2. The replicated pattern is of size $r \times c$, with $(r, c) = (2, 3)$.

Let us analyze the data dependencies during iteration ℓ of the right looking variant of the algorithm: the factorized tile (ℓ, ℓ) is needed to perform solve operations (TRSM) on all tiles on column ℓ and row ℓ; then the update operation (GEMM) on tile (i, j) with $(i, j) \in \{\ell + 1, \ldots, m\}^2$ requires both tiles (i, ℓ) and (ℓ, i). With the *owner computes* rule, this means that each node owning a tile in column ℓ of A sends it to all other nodes on the same row to the right, and each node owning a tile in row ℓ sends it to all other nodes on the same column below, as illustrated by the black zones on the left of Figure 2.

There are x_i different nodes on a row i of the pattern (respectively y_j on a column j) and one among them is the sender. Hence, for each replicated pattern, the volume

of communications generated is $\underbrace{\sum_{i=1}^{r} x_i - 1}_{\text{row-wise}} + \underbrace{\sum_{j=1}^{c} y_j - 1}_{\text{column-wise}}$. The pattern is replicated $\frac{m-\ell}{r}$ times vertically and $\frac{m-\ell}{c}$ times horizontally over the trailing submatrix of A. Assuming that $\ell \leqslant m - \max(r, c)$ at least one full pattern appears both vertically and horizontally. Therefore the total number of communications at iteration ℓ is $Q_l^{\text{LU}} = (m-\ell)\big(\frac{1}{r}\sum_{i=1}^{r}(x_i - 1) + \frac{1}{c}\sum_{j=1}^{c}(y_j - 1)\big)$. If we denote by $\bar{x}$ and $\bar{y}$ the average values of x_i and y_j, so that $\bar{x} = \frac{1}{r}\sum_{i=1}^{r} x_i$, and the total volume of communication over the complete factorization is given by

$$Q^{\text{LU}}(\mathcal{G}) = \frac{m(m+1)}{2}\big(\bar{x} + \bar{y} - 2\big) \tag{1}$$

As soon as $\ell > m - \max(r, c)$, the trailing matrix of A becomes smaller than a full pattern, either vertically or horizontally. If $\ell > m - r$, the term of the first sum in Equation 1 is not x_i but the number of different nodes on row i of the pattern allocated to the last $m - \ell$ columns of matrix A, which is smaller than x_i, and similarly if $\ell > m - c$ for y_j. Equation 1 therefore overestimates the row-wise communications for the last c iterations, respectively column-wise communications for the last r iterations, because the domains where elementary operations are performed shrinks as iterations progress. Besides, partial pattern replication may occur at the edges of the matrix if m is not a multiple of r (and/or m is not a multiple of c) which would change one term of each sum in Equation 1. In either case, it modifies the non-dominant parts of the total communication estimate and can therefore be neglected.

B. Case of Cholesky factorization

The case of Cholesky decomposition is depicted on the right of Figure 2. Since A is symmetric, only half of it is needed; we assume that it is the lower triangular part. In this case, during iteration ℓ, tiles of column ℓ are sent to all nodes along the same row to the right, *and* along the column with the same index. We denote such a set of tiles as a *colrow*:

Definition 1 (colrow): given a square pattern or a square matrix, we denote as *colrow* i the union of the column i and of the row i of the pattern or matrix.

In addition, for a square pattern $\mathcal{G}$, we denote by z_i the number of nodes belonging to colrow i of $\mathcal{G}$.

In the case of Cholesky factorization, tiles are sent along a colrow instead of a row or a column. Furthermore, when the pattern $\mathcal{G}$ is square, a colrow of the matrix always corresponds to exactly one colrow of the pattern. If this is the case, the previous reasoning to compute the number of communications at a given iteration remains valid if applied with the number of different nodes in a colrow, *i.e.* z_i for colrow i of the pattern. The number of communications generated at a given iteration for this kernel is therefore given by

$$Q^{\text{Chol}}(\mathcal{G}) = \frac{m(m+1)}{2}(\bar{z} - 1). \tag{2}$$

The same restrictions apply as for LU factorization regarding domain shrinking and partial model replications.

C. Communication cost metric

In both cases (Equations 1 and 2), neither the multiplicative factor $\frac{m(m+1)}{2}$ nor the additive -1 depend on the pattern. In the following, we thus aim at designing patterns $\mathcal{G}$ that minimize the following metric $T(\mathcal{G})$, which we call *communication cost*

$$T(\mathcal{G}) = \begin{cases} \bar{x} + \bar{y} & \text{for LU factorization} \\ \bar{z} & \text{for Cholesky factorization, if } r = c \end{cases}$$

In addition, in order to enforce a good load balancing of the computations, we require that the patterns are *balanced*, *i.e.* that all nodes appear the same number of times in the pattern. Note that contrarily to the 2DBC pattern, we allow a node to appear several times in the pattern.

IV. Generalized 2DBC

We introduce G-2DBC (for Generalized 2DBC), which extends the original 2DBC distribution to any number of nodes, maintaining the good properties in terms of communication volume of the square 2DBC distribution with $P = r^2$ nodes. First, we show in Section IV-A how to build the pattern. Then, in Section IV-B, we prove some properties associated with G-2DBC. We then provide a comparative evaluation of the cost of G-2DBC and 2DBC in terms of communication in Section IV-C. Finally experimental results illustrating the performance of G-2DBC are presented in Section IV-D.

A. Pattern Construction

In order to approximate a square pattern, we first build a quasi-square pattern with P cells (if $\exists r,\ P = r^2$) or slightly more. We use the following definitions:

$$a = \left\lceil \sqrt{P} \right\rceil, \qquad b = \left\lceil \frac{P}{a} \right\rceil, \qquad \text{and } c = ab - P.$$

By construction, we can first observe that $0 \leq c < a$. Indeed, $\frac{P}{a} \leq b < \frac{P}{a} + 1$ so that $P \leq ab < P + a$ and $0 \leq c < a$. We can now define an incomplete $b \times a$ pattern $\mathcal{IP}$ which contains the elements from 1 to P and whose c elements of the last row are left undefined. An example for $P = 10$ is given on the left of Figure 3.

From $\mathcal{IP}$, we now build $b - 1$ different $b \times a$ patterns $\mathcal{P}_i$, for $1 \leq i \leq b - 1$. For a given $1 \leq i \leq b - 1$, the pattern $\mathcal{P}_i$ is a copy of $\mathcal{IP}$, where the undefined elements are replaced with the last c elements of row i in $\mathcal{IP}$. These c elements are thus present twice in $\mathcal{P}_i$.

Additionally, we also build a last pattern of size $b \times (a - c)$, denoted $\mathcal{LP}$, which consists of the $a - c$ first columns of $\mathcal{IP}$.

Finally, the complete G-2DBC pattern $\mathcal{P}$ has size $b(b-1) \times P$, and is built as follows. The first b rows of $\mathcal{P}$ contain $b - 1$ copies of $\mathcal{P}_1$, followed by a copy of $\mathcal{LP}$. This leads to a total of $a * (b - 1) + a - c = ab - c = P$ columns. The following rows of $\mathcal{P}$ are built in the same way, successively using copies of $\mathcal{P}_2, \ldots, \mathcal{P}_{b-1}$. The result pattern for $P = 10$ can be seen on the right of Figure 3.

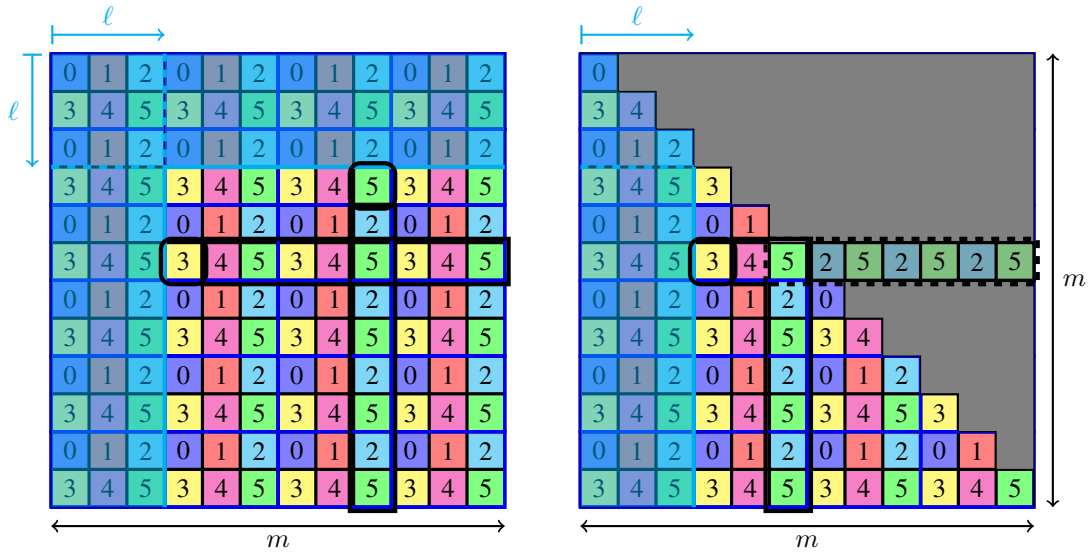


Fig. 2: Communication scheme using 2DBC distribution for $m = 12$, using $P = 6$ nodes laid out as a 2×3 pattern. **Left:** LU factorization; solid black rectangles highlight two tiles of A sent by nodes 3 and 5 at iteration $\ell = 3$ and the corresponding sets of receiver nodes. **Right:** Cholesky factorization; one tile of A is sent by node 3 at iteration $\ell = 3$ to receiver nodes in its *colrow*, as highlighted by the solid black shape. This communication scheme comes from the symmetry of A as illustrated by the dashed black rectangle.

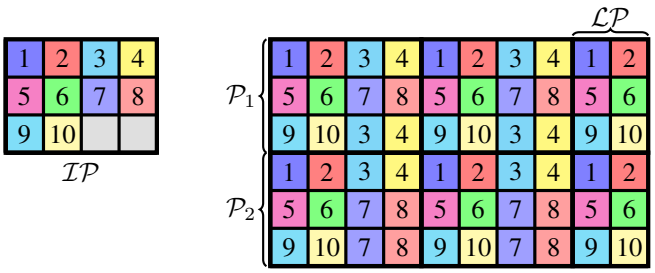


Fig. 3: Example of the G-2DBC pattern for $P = 10$, thus $a = 4$, $b = 3$ and $c = 2$. **Left:** incomplete pattern $\mathcal{IP}$. **Right:** full G-2DBC pattern.

B. Pattern Properties

We first show that the pattern $\mathcal{P}$ is well balanced: each node appears exactly $b(b-1)$ times.

Lemma 1: In the G-2DBC pattern $\mathcal{P}$ defined above, each node is assigned to exactly $b(b-1)$ cells.

Proof: We distinguish the nodes in $\mathcal{LP}$ and the others.

(i) Nodes in $\mathcal{LP}$ appear exactly once in $\mathcal{LP}$ and in each of the patterns $\mathcal{P}_i$. Furthermore, the pattern $\mathcal{P}$ contains exactly $b-1$ copies of $\mathcal{LP}$ and $b-1$ copies of each $\mathcal{P}_i$. Hence, a node in $\mathcal{LP}$ appears $b(b-1)$ times $\mathcal{P}$.

(ii) Let us now consider a node that does not appear in $\mathcal{LP}$, and denote by u its row in $\mathcal{IP}$. This node appears in exactly two cells of $\mathcal{P}_u$, and in exactly one cell in each $\mathcal{P}_i$ for $i \neq u$. Therefore, this node appears in exactly $2(b-1)+(b-2)(b-1) = b(b-1)$ cells in $\mathcal{P}$. □

Let us now analyze the communication cost $T(\mathcal{P})$ of this pattern. For this purpose, let us compute the number of different nodes that appear in each row and each column of $\mathcal{P}$. By construction, in each row of $\mathcal{P}$, exactly a different nodes are present, so that $\bar{x} = a$. In terms of columns, the situation differs depending on whether the column is part of $\mathcal{LP}$ or not. A column of $\mathcal{LP}$ contains exactly b different nodes, there are $a-c$ such columns, and the complete pattern contains b copies of each of these columns (one as a column of $\mathcal{LP}$, and $b-1$ as copies of the $\mathcal{P}_i$). A column not in $\mathcal{LP}$ contains $b-1$ different nodes, since the c undefined cells of $\mathcal{IP}$ are filled with nodes already present in the column. There are c such columns, and they are copied $b-1$ times in pattern $\mathcal{P}$.

In total, the pattern $\mathcal{P}$ has P columns, so that the average number of nodes per column in $\mathcal{P}$ is

$$\bar{y} = \frac{1}{P}\Big(\overbrace{b \cdot (a-c) \cdot b}^{\text{columns in } \mathcal{LP}} + \overbrace{(b-1) \cdot c \cdot (b-1)}^{\text{columns not in } \mathcal{LP}}\Big).$$

We can thus provide a bound on the total cost of the G-2DBC pattern.

Lemma 2: For any P, the Generalized 2DBC pattern $\mathcal{P}$ with P nodes has a total cost $T(\mathcal{P})$ bounded by $2\sqrt{P} + \frac{2}{\sqrt{P}}$.

Proof: By definition, $T(\mathcal{P}) = \bar{x} + \bar{y}$. The above considerations yield $\bar{x} = a$ and $\bar{y} = \frac{1}{P}(b^2(a-c) + (b-1)^2 c)$. We start by

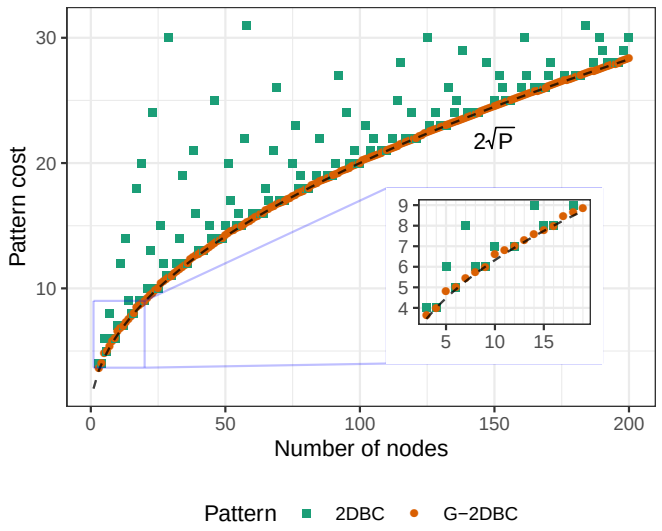


Fig. 4: Total cost T of G-2DBC and the best 2DBC for varying values of P.

P	2DBC		G-2DBC	
	dim.	T	dim.	T
16	4x4	8		
20	5x4	9		
21	7x3	10		
22	11x2	13		
23	23x1	23	20x23	9.261
30	6x5	11		
31	31x1	31	30x31	11.194
35	7x5	12	30x35	11.857
36	6x6	12		
39	13x3	16	30x39	12.615

(a) LU factorization

P	SBC		GCR&M	
	dim.	T	dim.	T
21	7x7	6		
23			22x22	6.045
28	8x8	7		
31			31x31	7.065
32	8x8	8		
35			15x15	7.4
36	9x9	8		
39			27x27	7.926

(b) Cholesky factorization

TABLE I: Dimensions and cost of the patterns used for the experimental evaluation. The rows highlighted in grey represent the experimental test cases.

bounding $\bar{y}$, using $ab - c = P$:

$$\bar{y} = \frac{b^2a - 2bc + c}{P} = b\left(1 - \frac{c}{P}\right) + \frac{c}{P} = \frac{(P+c)(P-c)}{aP} + \frac{c}{P} \leqslant \frac{P}{a} + \frac{c}{P}.$$

$$= \frac{b(ab-c) - cb + c}{P}$$

we now replace $b = \frac{P+c}{a}$:

$$= \frac{P^2 - c^2}{aP} + \frac{c}{P}$$

Let us write $a = \sqrt{P} + \mu$, where $0 \leq \mu < 1$. Then,

$$\frac{\mu^2}{a} = \frac{(a-\sqrt{P})^2}{a} = a - 2\sqrt{P} + \frac{P}{a}, \qquad \text{hence} \qquad a + \frac{P}{a} = 2\sqrt{P} + \frac{\mu^2}{a}$$

We can now bound $T(\mathcal{P}) = \bar{x} + \bar{y} \leqslant a + \frac{P}{a} + \frac{c}{P}$:

$$T(\mathcal{P}) \leqslant 2\sqrt{P} + \frac{\mu^2}{a} + \frac{c}{P} \leqslant 2\sqrt{P} + \frac{2}{\sqrt{P}}$$

The last inequality holds because $\mu < 1$, $a \geq \sqrt{P}$, and $c \leq a - 1 \leq \sqrt{P}$. □

As a comparison, the square 2DBC pattern for $P = p^2$ obtains a cost exactly equal to $2\sqrt{P}$. In addition, we can remark that if $c = 0$ (*i.e.* if $P = p^2$ or if $P = p(p+1)$), the G-2DBC pattern reduces to the standard 2DBC pattern.

C. Evaluation of G-2DBC

We provide on Figure 4 a representation of the cost $T(\mathcal{P})$ of the G-2DBC pattern, compared to the standard 2DBC pattern. For each value of P, we display the cost of the best available 2DBC pattern, using all possible ways to write P as $P = rc$, and the cost of the corresponding G-2DBC pattern. As we can see, the cost of G-2DBC closely follows the $2\sqrt{P}$ value, and allows to significantly improve the volume of communications over 2DBC for many values of P.

D. Experimental Performance Evaluation of G-2DBC

We now present sample results of experiments to test out the connection between the patterns cost and the actual performance of the associated cyclic distribution. We perform LU factorization on four test cases, using $P = 23, 31, 35$ and 39 nodes. Apart from the case $P = 35$, they correspond to situations where there is no satisfying 2DBC pattern using all the resources: to limit the cost of communications, we must use a 2DBC distribution with fewer nodes. On the other hand, for those situations, G-2DBC provides solutions using all the available nodes with a significantly lower communication cost.

For each case ($P = 23, 31, 35, 39$), we compare the execution time with the G-2DBC distribution using all nodes and one or several 2DBC distributions using a reasonably similar number of nodes, smaller or equal to P. The dimensions and communication cost of the patterns used are detailed in Table Ia. We observe the global and per-node performance in terms of GFlop/s. Figures 5 and 6 show the performance evolution according to the matrix size for the test cases $P = 23$ and $P = 39$. To illustrate the performance of both method in terms of strong scaling, sample results for all values of P and a matrix of size $N = 200,000$ can be seen on Figure 7a.

This experiment was made with the `Chameleon` library version 1.1.0, backed by the `StarPU` runtime version 1.3.8. `Chameleon` relies on the *Intel MKL 2020* implementation of `BLAS` routines and makes use of `Open MPI` version 4.0.3. In our experimental setting, we bound one MPI process per node.

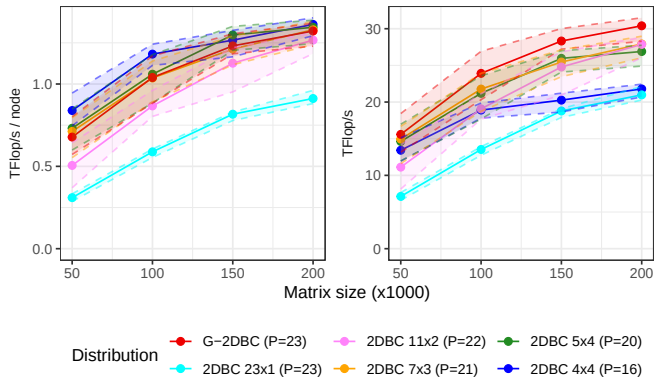


Fig. 5: Results for LU factorization using a maximum of $P = 23$ nodes.

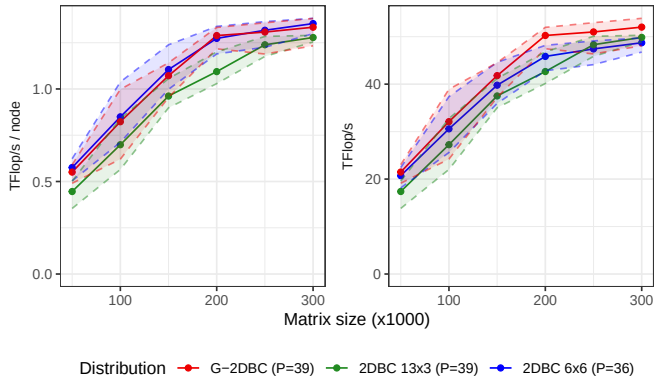
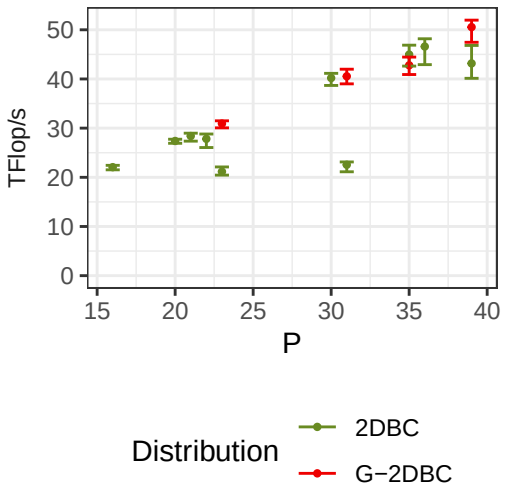


Fig. 6: Results for LU factorization using a maximum of $P = 39$ nodes.

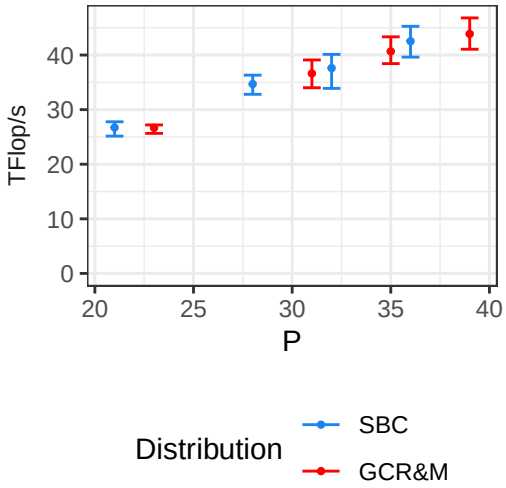
One `StarPU` instance is also running on each node, and uses one core to perform the scheduling and another core to handle the MPI communications. All the operations are performed in double precision on a cluster of 44 nodes, each with 36 *Intel Xeon Skylake Gold 6240* cores, for a total of 1512 cores. The nodes are connected with a 100Gb/s OmniPath network.

Experiments have been performed on randomly generated matrices whose sizes range from $50,000$ to $300,000$. They are divided into tiles of size 500×500, which is the smallest size for which individual cores perform kernels with enough efficiency so that single node performance is close to the peak. For each matrix size, 5 experiments have performed. Each dot represents the average result over those 5 runs while the shaded zone represents the minimum and maximum range of values.

In both test cases on Figures 5 and 6, we observe that G-2DBC distribution consistently achieves a higher raw performance for all tested matrix sizes. More precisely, for $P = 23$, 2DBC using all the nodes performs poorly because of the tall and narrow shape of the pattern used, 23×1, which induces many communications. Its global throughput is even smaller than the version using only 16 nodes laid out as a perfect square, though the trend seems to reverse for larger



(a) LU factorization



(b) Cholesky factorization

Fig. 7: Strong-scaling results using an input matrix of size $N \times N$, $N = 200,000$.

matrix sizes. On the other hand, G-2DBC shows better global performance for all matrix sizes, which immediately translates into smaller running time. On a per-node basis, its performance is comparable to the 2DBC distribution using 21 nodes with a 7×3 pattern. The observations are quite similar for $P = 39$: despite using all the available nodes, the performance of the second 2DBC distribution is hindered by the rectangular shape of its 13×3 pattern which generates many communications. As in the previous case, it is less efficient than its square-pattern counterpart using 36 nodes. G-2DBC consistently achieves the highest throughput of all three distributions for all matrix sizes and manages to reach the same per-node efficiency than 2DBC with a 6×6 patterns while using roughly 10% more resources.

On Figure 7a, we can analyze the performance for both distributions for a fixed matrix size ($N = 200,000$), when

the number of nodes used varies. We can make similar observations: as expected, 2DBC performs poorly when forced to use $23, 31$ and, to a lesser extent, 39 nodes. On the contrary, in those cases, G-2DBC solutions reach significantly higher performance. In the case $P = 35$ where the 2DBC pattern is reasonably square, we can observe that G-2DBC distribution manages to achieve roughly the same performance.

As a conclusion, those experiments illustrate that G-2DBC distribution enables to efficiently use all available resources in parallel while not sacrificing on the number of inter-node communications thanks to its almost periodic and square pattern. Indeed, even if the G-2DBC pattern is significantly larger (22×23 for G-2DBC with $P = 23$ against 4×5 for 2DBC for $P = 20$), it is in fact itself quasi-periodic since it is built from 20 quasi-copies of the 5×5 incomplete pattern $\mathcal{IP}$ defined in Section IV-A. The workload between the processors in the trailing matrix remains very well balanced, even if the pattern is larger. This allows to achieve higher global performance than 2DBC distribution for specific values of the number of nodes P and in turn translates into faster global execution of the operation.

V. Data Distributions for Symmetric Matrices

In this section, let us consider the symmetric case and search for patterns to generalize the SBC distribution. A mentioned above, in the symmetric case it is necessary to use square patterns, of size $r = c$. Having a balanced pattern of size r imposes a strong constraint on r: since there are r^2 cells to distribute among P nodes, a balanced pattern would require that r^2 is a multiple of P. To soften this constraint, we use a property of the diagonal cells of the pattern: each diagonal cell belongs to a unique colrow. It can thus be assigned to any node on its colrow without changing the communication cost, and its different replicas on the complete matrix might even be assigned to different nodes as long as these nodes belong to the colrow of the pattern. It is thus possible to design incomplete patterns, where the diagonal cells remain undefined, and assign them only when the pattern is replicated on the complete matrix. This can be done greedily, by successively assigning undefined tiles to the least loaded node among those present in the colrow. This is a generalization of the *extended* version of SBC (see Section III-C in [8]).

Still, there are limitations to the possible values of r. The load balancing procedure above can only be performed successfully if each node is assigned at most $\frac{r^2}{P}$ cells, otherwise one node would be assigned too many tiles even without receiving any undefined tile. The number of non-diagonal cells to assign is $\frac{r(r-1)}{P}$, so that at least one node receives $\left\lceil \frac{r(r-1)}{P} \right\rceil$ cells. Thus, a balanced pattern exist if and only if

$$\left\lceil \frac{r(r-1)}{P} \right\rceil \leq \frac{r^2}{P}. \tag{3}$$

A. Greedy ColRow & Matching Algorithm

Our greedy algorithm, Greedy ColRow & Matching (GCR&M), has two phases. The first phase assigns colrows to nodes, trying to balance the load between them. In this process however, it may happen that the colrow assignment allows several nodes to be assigned the same cell. Then, the second phase of the greedy algorithm applies a matching algorithm to perform the assignment of cells to the nodes.

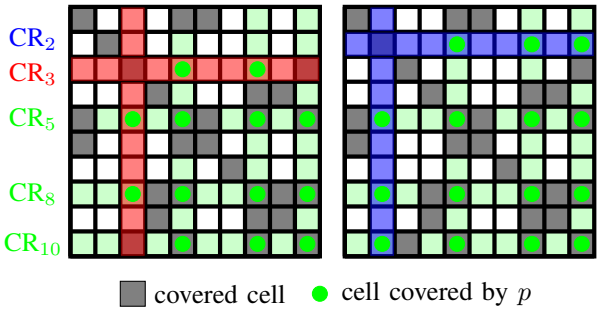


Fig. 8: First phase of GCR&M algorithm, line 8: node p was already assigned colrows 5, 8 and 10. Colrow 2 is preferred over colrow 3 for the next assignment to p, since it covers more new cells.

Algorithm 1: Greedy ColRow & Matching Algorithm

Input: Number of nodes P, Pattern Size r

1 $\mathcal{U} \leftarrow \{(i,j) | 1 \leq i \neq j \leq r\}$
2 **for** $i \leftarrow 1 \dots r$ **do**
3 $\quad \mathcal{A}[i \mod P] \leftarrow \{i\}$
4 **while** $\mathcal{U} \neq \emptyset$ **do**
5 $\quad p \leftarrow$ least loaded node
6 $\quad$ **for** each colrow r **do**
7 $\quad\quad \mathcal{C}[r] \leftarrow \mathcal{U} \cap \{(r,i) | i \in \mathcal{A}[p]\}$
8 $\quad b \leftarrow \text{argmax}_r(\text{Card}(\mathcal{C}[r]))$ *(tie-break: lowest usage)*
9 $\quad \mathcal{A}[p] \leftarrow \mathcal{A}[p] \cup \{b\}$
10 $\quad$ remove $\mathcal{C}[b]$ from $\mathcal{U}$
11 Compute a matching between all cells and $\left\lfloor \frac{r(r-1)}{P} \right\rfloor$ duplicates per node
12 Compute a matching between unassigned cells and 1 duplicate per node
13 **if** unassigned cells $c = (i,j)$ remain **then**
14 $\quad$ Assign (i,j) to the least loaded node p s.t. $\mathcal{A}[p]$ contains i or j

a) First phase: The first phase of the GCR&M algorithm computes an assignment $\mathcal{A}$ of colrows to nodes, so that $\mathcal{A}[p]$ is the set of colrows on which node p can appear. A cell (i,j) is *covered* by a node p if p can appear on this cell, *i.e.* i and j are both in $\mathcal{A}[p]$. In Algorithm 1, the set $\mathcal{U}$, defined line 1, contains all *uncovered* cells.

The algorithm starts by assigning one node to each colrow, in a round robin way if $P < r$, as shown line 3. The rest of the assignment is performed with a greedy procedure: as long as an uncovered tile remains, we assign an additional colrow to the least loaded node. The colrow is chosen so as to maximize the number of newly covered cells (see line 8, and

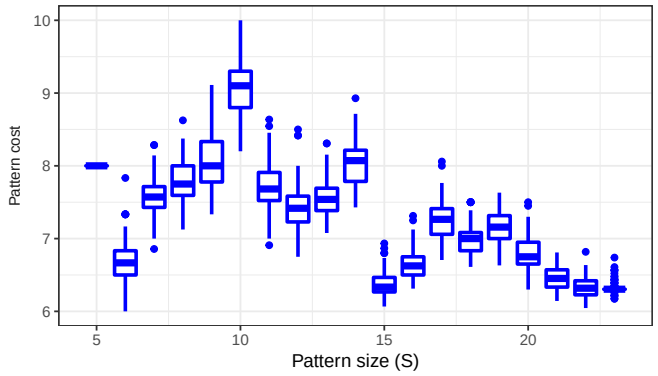


Fig. 9: Effect of random choices and pattern size for $P = 23$.

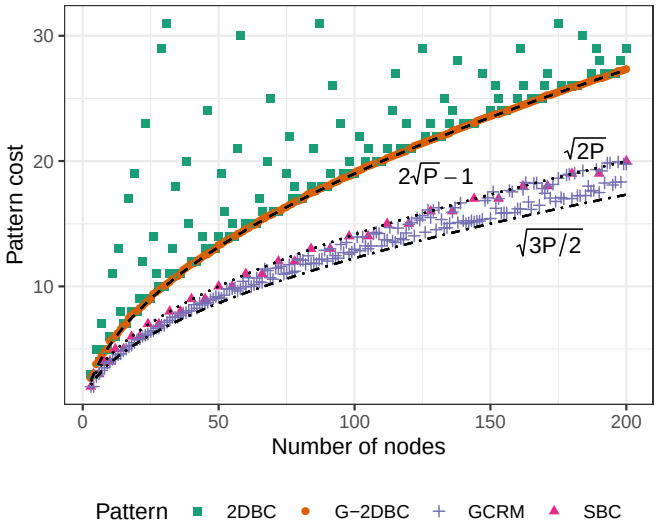


Fig. 10: Total cost T of the symmetric and non-symmetric patterns for varying values of P.

the example provided in Figure 8). In case of equality, the least used colrow is selected (the one which appears in the lowest number of $\mathcal{A}[p]$s). Further ties are broken randomly. When a colrow is selected, the set of covered cells is updated.

b) Second phase: The first phase of the algorithm provides an assignment of colrows to the nodes, and could also be used to obtain an assignment of cells. For example, one could assign a cell to the first node that covers it. However, this does not guarantee a good load balancing. To solve this, the second phase of the GCR&M algorithm uses a bipartite graph matching approach. We build a bipartite graph with all cells on one side, and k copies of each node on the other side, where $k = \left\lfloor \frac{r(r-1)}{P} \right\rfloor$. There are edges between a cell and all copies of all nodes that cover this cell. A bipartite matching algorithm is used to perform an assignment of cells to nodes.

The reasoning behind this choice of k is the following: the total number of copies of nodes is lower than the number of cells, so that in a perfect matching all nodes receive k cells. Using $k' = \left\lceil \frac{r(r-1)}{P} \right\rceil$ would result in more copies of nodes than cells, so that all cells would be assigned. However, there would be no guarantee on the load balance between nodes: it could happen that many nodes receive k' cells, and some nodes receive few or even zero cells.

Hence, this first matching assigns k cells per node but may leave some cells unassigned. Another matching procedure is performed next, between all unassigned cells and a single copy per node. This ensures that all nodes have at least $\left\lfloor \frac{r(r-1)}{P} \right\rfloor$ and at most $\left\lceil \frac{r(r-1)}{P} \right\rceil$ cells. If some cells remain unassigned after both matching procedures, each remaining cell is assigned greedily to the least loaded node p that can cover it by adding only one colrow to $\mathcal{A}(p)$ (see line 14).

B. Evaluation of GCR&M

GCR&M is defined in Algorithm 1 for a given pattern size r; it depends on random choices made when breaking ties. Our evaluation takes into account these two parameters in the following way. For each value of P, we apply Algorithm 1 for all values of $r \leq 6\sqrt{P}$ which satisfy Equation 3. The algorithm is run 100 times with different random seeds. For each value of P, we keep the pattern with the lowest cost. These thresholds ($6\sqrt{P}$ and the number of re-executions) are sufficient in practice to obtain good patterns.

The effect of the pattern size on the quality of the solution is presented on Figure 9, for $P = 23$. This figure shows the pattern cost, computed with Equation 2, as a function of r and for different random choices. We can see that it is not trivial to find the best value of r; in particular, a larger pattern size does not always give better results. We can also notice that for a fixed pattern size r, the random choices made when breaking ties have a significant impact on the quality of the pattern. Further studies would be necessary to better understand these phenomena. However, since the patterns depend only on the number of processors, and typically not on the size of the matrix, obtaining the set of points of Figure 9 for a given value of P can be done once and for all; furthermore it only takes a few seconds on a laptop. The results presented below for GCR&M are therefore based on such an enumeration of random choices and pattern sizes.

We now perform a theoretical evaluation of GCR&M by providing on Figure 10 a comparison of the cost of all patterns for the symmetric case. As before, for each value of P the plot shows the cost of the best pattern of each type for this value of P. Note that for 2DBC and G-2DBC, the symmetric cost is equal to the non-symmetric cost minus 1: indeed, the number of nodes on a colrow is the sum of the number of nodes on some column and on some row, minus 1 for the intersection. The plot shows the SBC pattern, whose cost grows as $\sqrt{2P}$ for its basic version, and $\sqrt{2P} - 0.5$ for the extended version. The GCR&M algorithm obtains patterns with a cost either similar to SBC, or even lower in many cases, with a lower limit that we empirically observe to be $\sqrt{3P/2}$.

We can provide an intuition for this limit. Let us consider a regular pattern in which each node is present on v colrows,

and is assigned l non-diagonal cells. For example, on the SBC pattern, $v = 2$ and $l = 2$. If such a regular pattern contains P nodes, then its size r satisfies $r(r-1) = Pl$, so that $r \sim \sqrt{Pl}$. In addition, since each node is present on v colrows, we can compute the total number of nodes present on the colrows: $\sum_i z_i = Pv$. Thus, $\bar{z} \sim \frac{Pv}{\sqrt{Pl}} = \frac{v}{\sqrt{l}}\sqrt{P}$.

As mentioned, for SBC we have $v = 2$. The next possible value for v is $v = 3$, in which case the maximum possible number of non-diagonal cells is $v(v-1) = 6$. A regular pattern with these values would have a communication cost $T \sim \frac{3}{\sqrt{6}}\sqrt{P} = \sqrt{\frac{3P}{2}}$. The results obtained by GCR&M show that it is able to produce patterns where most of the nodes obtain an assignment of cells with a similar efficiency as what can be done with $v = 3$.

C. Experimental Performance Evaluation of GCR&M

Like for the LU factorization, we present an experimental evaluation of the solutions provided by GCR&M algorithm. We consider the Cholesky factorization in the same four test cases with a total number of available nodes $P = 23, 31, 35$ and 39. To the best of our knowledge, since the Cholesky factorization is a symmetric operation, the best static allocation strategy regarding the number of communications is SBC. In all four cases, since there exists no SBC distribution using all the available nodes, it is necessary to use fewer nodes. The number of nodes, pattern size and associated communication costs used for this experiment are gathered in Table Ib. All experiments are carried out using the same experimental setting as for LU factorization, described in Section IV-D.

Sample results for the test cases $P = 31$ and $P = 35$ are illustrated on Figures 11 and 12, which report the performance for different input matrix sizes. In the case $P = 31$, the best SBC distribution uses 28 nodes laid out in a 8×8 pattern. We can observe that the raw performance using GCR&M distribution using all 31 nodes is higher than its SBC counterpart for all tested matrix sizes. GCR&M distribution achieves up to 11% higher throughput than SBC for the largest matrix size $N = 300,000$. Regarding the performance per node, the results of GCR&M distribution is however slightly worse than SBC, though the gap between the two strategies reduces as large matrix sizes are considered.

Similar trends can be observed on Figure 12 for the case $P = 35$: in this configuration, the SBC distribution uses 32 nodes. We can notice that the communication cost of the GCR&M pattern is significantly smaller than that of the SBC pattern, 7.4 compared to 8, as detailed in Table Ib. Using all available nodes while generating 7.5% less communications, the GCR&M distribution yields better raw performance than its SBC counterpart for all tested matrix sizes.

Figure 7b shows performance results of both methods for all considered values of P for a matrix size $N = 200,000$. We can observe that the strong scaling trends of performance for both strategies are very similar. This is actually a relevant indicator of the interest of GCR&M allocation strategy as it allows to fill the gap between two neighboring SBC distributions while achieving the expected performance. For example, GCR&M

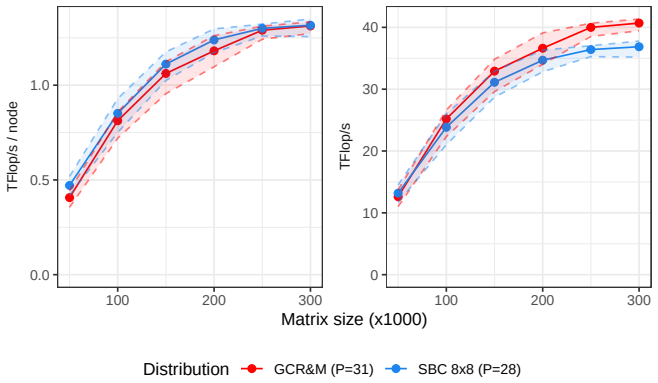


Fig. 11: Results for Cholesky factorization using a maximum of $P = 31$ nodes.

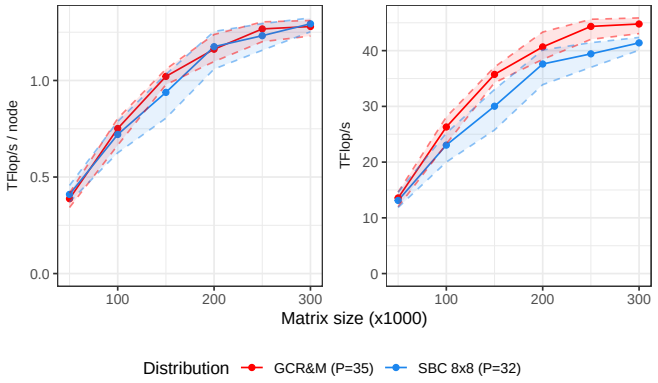


Fig. 12: Results for Cholesky factorization using a maximum of $P = 35$ nodes.

performance for $P = 35$ is almost what could be expected if SBC allocations were available for any number of nodes and would evolve linearly with P.

In this section, we present results only for specific values of P to showcase the interest of GCR&M method. However, as shown on Figure 10, the procedure detailed in Algorithm 1 generates patterns that achieve a communication cost close to or better than SBC for any number of nodes.

VI. Conclusion

In this paper, we propose improved distribution schemes for performing the LU and Cholesky factorizations on homogeneous distributed nodes. In the non-symmetric case, we propose a generalized version of the standard Block-Cyclic pattern, which can be applied to any number of nodes while retaining the communication efficiency of square 2D Block-Cyclic patterns. In the symmetric case, we propose GCR&M, a greedy heuristic which also produces patterns for any number of nodes. These patterns can be seen as a generalization of the recently introduced Symmetric Block-Cyclic pattern, and achieve an even better communication efficiency in many cases. An experimental evaluation shows that these patterns can be used within a task-based linear algebra library such as

`Chameleon`, to achieve improved performance when using all possible available nodes.

This work opens several perspectives. The question of whether it is possible to find an explicit description of an efficient pattern in the symmetric case (instead of relying on a heuristic) remains open. It would also be interesting to assess how large a pattern needs to be to obtain good communication efficiency, or to explore the tradeoff between pattern size and communication efficiency. Note that the computational cost of this pattern is not a problem, since we do not need to compute the patterns at each execution. On the contrary, one could imagine to provide in a database containing, for each possible value of P, a very efficient pattern for the symmetric case. Another avenue of research could be to extend these results to the case of heterogeneous nodes.

ACKNOWLEDGEMENT

This work was partially supported by the Euro-HPC project Towards EXtreme scale Technologies and Accelerators for euROhpc hw/Sw Supercomputing Applications for exascale (TEXTAROSSA), Horizon 2020 Program for Research and Innovation, Project ID: 956831, and by the SOLHARIS project (ANR- 19-CE46-0009) which is operated by the French National Re- search Agency (ANR).

Experiments presented in this paper were carried out using the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see https://www.plafrim.fr).

REFERENCES

[1] D. Irony and S. Toledo, "Trading replication for communication in parallel distributed-memory dense solvers," *Parallel Processing Letters*, vol. 12, no. 01, pp. 79–94, 2002.

[2] G. Kwasniewski, T. Ben-Nun, A. N. Ziogas, T. Schneider, M. Besta, and T. Hoefler, "On the parallel I/O optimality of linear algebra kernels: Near-optimal LU factorization," in *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2021.

[3] O. Beaumont, P. Duchon, L. Eyraud-Dubois, J. Langou, and M. Vérité, "Symmetric Block-Cyclic Distribution: Fewer Communications Leads to Faster Dense Cholesky Factorization," in *SC 2022 - Supercomputing*, Dallas, Texas, United States, Nov. 2022.

[4] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures," *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, vol. 23, pp. 187–198, 2011.

[5] A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, and J. Planas, "OmpSs: a proposal for programming heterogeneous multi-core architectures," *Parallel processing letters*, 2011.

[6] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Hérault, and J. J. Dongarra, "Parsec: Exploiting heterogeneity to enhance scalability," *Computing in Science and Engineering*, vol. 15, no. 6, pp. 36–45, 2013.

[7] E. Agullo, O. Aumage, M. Faverge, N. Furmento, F. Pruvost, M. Sergent, and S. P. Thibault, "Achieving high performance on supercomputers with a sequential task-based programming model," *IEEE Transactions on Parallel and Distributed Systems*, 2017.

[8] O. Beaumont, L. Eyraud-Dubois, J. Langou, and M. Vérité, "I/O-optimal algorithms for symmetric linear algebra kernels," in *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, New York, USA, 2022, p. 423–433.

[9] J. Hong and H. T. Kung, "I/O complexity: The red-blue pebble game," in *STOC '81*, 1981.

[10] D. Irony, S. Toledo, and A. Tiskin, "Communication lower bounds for distributed-memory matrix multiplication," *J. Parallel Distrib. Comput.*, vol. 64, no. 9, pp. 1017–1026, Sep. 2004.

[11] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, 1997.

[12] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, "Communication-optimal parallel and sequential Cholesky decomposition," *SIAM Journal on Scientific Computing*, vol. 32, no. 6, pp. 3495–3523, 2010.

[13] ——, "Minimizing communication in numerical linear algebra," *SIAM J. Matrix Anal. Appl.*, vol. 32, pp. 866–901, 2011.

[14] A. Olivry, J. Langou, L.-N. Pouchet, P. Sadayappan, and F. Rastello, "Automated derivation of parametric data movement lower bounds for affine programs," in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 808–822.

[15] E. Solomonik and J. Demmel, "Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms," in *EuroPar*. Springer, 2011.

[16] A. Kalinov and A. Lastovetsky, "Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers," *Journal of Parallel and Distributed Computing*, vol. 61, no. 4, pp. 520–535, 2001.

[17] O. Beaumont, V. Boudet, F. Rastello, Y. Robert *et al.*, "Partitioning a square into rectangles: NP-completeness and approximation algorithms," *Algorithmica*, vol. 34, no. 3, pp. 217–239, 2002.

[18] B. A. Becker and A. Lastovetsky, "Towards data partitioning for parallel computing on three interconnected clusters," in *ISPDC'07*. IEEE, 2007, pp. 39–39.

[19] A. DeFlumere and A. Lastovetsky, "Optimal data partitioning shape for matrix multiplication on three fully connected heterogeneous processors," in *HeteroPar*, Porto, Portugal, 25 August 2014.

[20] T. Malik and A. Lastovetsky, "Towards optimal matrix partitioning for data parallel computing on a hybrid heterogeneous server," *IEEE Access*, vol. 9, pp. 17 229–17 244, 2021.

[21] H. Nagamochi and Y. Abe, "An approximation algorithm for dissecting a rectangle into rectangles with specified areas," *Discrete applied mathematics*, vol. 155, no. 4, pp. 523–537, 2007.

[22] A. Fügenschuh, K. Junosza-Szaniawski, and Z. Lonc, "Exact and approximation algorithms for a soft rectangle packing problem," *Optimization*, vol. 63, no. 11, pp. 1637–1663, 2014.

[23] O. Beaumont, L. Eyraud-Dubois, and T. Lambert, "A new approximation algorithm for matrix partitioning in presence of strongly heterogeneous processors," in *IPDPS*, 2016.

[24] O. Beaumont, B. A. Becker, A. DeFlumere, L. Eyraud-Dubois, T. Lambert, and A. Lastovetsky, "Recent advances in matrix partitioning for parallel computing on heterogeneous platforms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 1, pp. 218–229, 2018.

[25] Q. Cao, Y. Pei, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, "Extreme-scale task-based cholesky factorization toward climate and weather prediction applications," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, New York, NY, USA, 2020.

[26] S. Abdulah, Q. Cao, Y. Pei, G. Bosilca, J. Dongarra, M. G. Genton, D. E. Keyes, H. Ltaief, and Y. Sun, "Accelerating geostatistical modeling and prediction with mixed-precision computations: A high-productivity approach with parsec," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 964–976, 2022.

[27] Q. Cao, R. Alomairy, Y. Pei, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, "A framework to exploit data sparsity in tile low-rank cholesky factorization," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 414–424.