Row Replicated Block Cimmino

Iain Duff [1,2], Philippe Leleux [1], Daniel Ruiz [3], F. Sukru Torun [4]

[1] CERFACS, Toulouse, France
[2] Scientific Computing Dpt., Rutherford Appleton Laboratory, Oxon, England
[3] IRIT - Institut de recherche en informatique de Toulouse, Toulouse, France
[4] Ankara Yildirim Beyazit University, Ankara, Turkey

Technical Report TR-PA-22-51

Publications of the Parallel Algorithms Team

http://www.cerfacs.fr/publication/

Row Replicated Block Cimmino

Iain Duff [1,2], Philippe Leleux [1], Daniel Ruiz [3], F. Sukru Torun [4]

May 23, 2022

Abstract

We study a new technique for reducing the number of iterations of the block Cimmino method by replicating rows in the partitioned system, so that we obtain a non-disjoint partitioning of the rows. Since rows in different partitions that are close to colinear produce a poorly conditioned iteration matrix for the block Cimmino method, row replication can get around this problem. With intelligent replication choices, we can reduce the number of iterations for convergence of the replicated block Cimmino method. The downside is a slight increase of the computational work associated with each partition. In order to find a trade-off between a lower number of iterations and a higher cost per iteration, selecting the proper set of rows for replication is crucial. In this paper, we use graph-based techniques to find good candidates for replication. Since the block Cimmino method can be interpreted as a non-overlapping additive Schwartz method applied to the normal equations, the replication techniques correspond to introducing an overlap between the subdomains defined by the partitions. We show analytically in the case of a two-block partitioning how the replication improves the conditioning of the block Cimmino iteration matrix. We then use challenging 2D PDE problems to show that our algebraic approach targets physically meaningful phenomena on the interface between partitions. Finally, we demonstrate the efficiency of the proposed method in improving the performance of the block Cimmino solver, even with a small amount of replication, on problems from the SuiteSparse Matrix Collection.

Keywords: block Cimmino, hybrid methods, row replication, graph partitioning, overlapping domain decomposition methods

1 Introduction

In this work, we study the block Cimmino method [6], an iterative block row-projection method, for the solution of linear systems of the form

$$Ax = b, \tag{1}$$

where A is a real full rank square sparse matrix of dimension n, and x and b are vectors of size n. We partition the matrix into p mutually disjoint row-blocks as

$$\begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix}, \tag{2}$$

where b is partitioned according to the partitioning of A. The row sizes of A_i and b_i are n_i, $i \in \{1..p\}$, such that $\sum_{i=1}^{p} n_i = n$.

Using this partitioned system, an iteration of block Cimmino consists in projecting the current iterate on the subspaces spanned by each partition then taking a weighted sum of these projections as the new approximate solution. In practice the standard block Cimmino method has been found to converge slowly on many problems [6]. Our work is based on an acceleration of this scheme using a stabilized *block conjugate gradient* algorithm (block-CG) [1] applied to the *symmetric positive definite* (SPD) system

$$Hx = f \text{ with } \begin{cases} H & = \sum_{i=1}^{p} A_i^{+} A_i, \\ f & = \sum_{i=1}^{p} A_i^{+} b_i, \end{cases} \tag{3}$$

where H corresponds to the sum of the projections on the range of A_i^T. We use the notation BC to refer to this accelerated version of the block Cimmino iterations. This scheme is implemented in the ABCD-Solver [1] using MPI parallelism [16]. In this implementation, each partition is distributed to a separate MPI process. Then, at each iteration, all the processes simultaneously compute in parallel the projection associated to their respective partition, through the direct solution of so-called projection systems [16]. The independence between all of the projections is a key point for the hybrid parallelism scheme of the ABCD-Solver. The workload associated with the computation of the projections must be evenly distributed among the MPI processes to get good parallel efficiency. In order to compute the global sum of the local projection vectors, only the common nonzero columns between each pair of partitions, called interconnection columns, must be considered. This distributed sum is then computed in parallel through point-to-point communications between interconnected processes, and the amount of communicated data should be as low as possible.

In [6], the author proves that the spectrum of the iteration matrix H, and thus the convergence of the method, is dependent on the cosines of the principal angles between the subspaces of each partition. The wider the angles are, the faster the convergence. The cosines of the principal angles between $\mathcal{R}(A_i^T)$ and $\mathcal{R}(A_j^T)$ (see [10, pages 329-330]) are successively defined as

$$\begin{aligned} \cos(\Psi_k) &= \frac{u_k^T v_k}{\|u_k\| \|v_k\|} \\ &= \max_{u \in \mathcal{R}(A_i^T)} \max_{v \in \mathcal{R}(A_j^T)} \frac{u^T v}{\|u\| \|v\|} \\ &\quad \text{s.t. } u^T u_p = 0 \text{ and } v^T v_p = 0, \ \ p = 1, \ldots, k-1, \end{aligned} \tag{4}$$

with k varying from 1 to $m_{ij} = \min(\dim(\mathcal{R}({A_i}^T)), \dim(\mathcal{R}({A_j}^T)))$. When we normalise the rows of the matrix in the 2-norm, these cosines are then directly linked to the inner products between rows in separate partitions, also called inter-block inner products.

Here, we will use this link between the spectrum of the iteration matrix and inter-block inner products to improve the convergence of BC with a two-step approach. In the first step, the matrix is partitioned to minimize the sum of inter-block inner product values between row-blocks, while balancing the computational load corresponding to the solution on each block. This in turn leads to an increase in the angles between subspaces spanned by the partitions, and thus to faster convergence of BC. In the second step, starting from the computed partitioning, the convergence

[1] `http://abcd.enseeiht.fr/`

rate of BC is further improved via the application of row replication techniques. These replications lead to a non-disjoint partitioning of the matrix, on which the associated iteration matrix is better conditioned than the original iteration matrix H.

In this paper, we first extend the theory for the block Cimmino method to the use of a non-disjoint partitioning in Section 2. Then, in Sections 2.1 and 2.2, we give geometric and algebraic interpretations which explain why replication helps to accelerate BC. Based on these ideas, in Section 3, we introduce the actual replication techniques based on graph partitioning methods. Finally, in Section 4, we demonstrate the efficiency of this approach on the solution of challenging PDE problems, as well as the parallel solution of matrices from the SuiteSparse Matrix Collection[4].

2 Row-replicated block Cimmino

We consider that the system (1) is partitioned into p blocks of rows, which are now potentially no longer mutually disjoint, contrary to (2). We have the consistent and partitioned system

$$\widetilde{A} = \begin{pmatrix} \widetilde{A}_1 \\ \widetilde{A}_2 \\ \vdots \\ \widetilde{A}_p \end{pmatrix} x = \begin{pmatrix} \widetilde{b}_1 \\ \widetilde{b}_2 \\ \vdots \\ \widetilde{b}_p \end{pmatrix}, \tag{5}$$

where $\widetilde{A}_i$ and $\widetilde{A}_j$ (as well as $\widetilde{b}_i$ and $\widetilde{b}_j$), $i \neq j$, may contain identical rows. These partitions are derived from those in (2), i.e. $\widetilde{A}_i$ contains the rows of A_i, and the additional rows are obtained from a replication technique which we detail in Section 3.

More precisely, we consider that each partition has the structure

$$\widetilde{A}_i = \begin{pmatrix} A_i \\ R_i \end{pmatrix} \in \mathbb{R}^{\widetilde{n}_i \times n}, \quad \widetilde{b}_i = \begin{pmatrix} b_i \\ f_i \end{pmatrix} \in \mathbb{R}^{\widetilde{n}_i}, \quad i \in \{1, \dots, p\}, \tag{6}$$

where R_i (resp. f_i) is a full rank block of $r_i \geq 0$ rows replicated from several other partitions A_j (resp b_j), $j \neq i$. The row size of $\widetilde{A}_i$ is $\widetilde{n}_i = n_i + r_i$, with $\sum_{i=1}^{p} \widetilde{n}_i \geq n$. In other words, R_i is a set of r_i distinct replicated rows which were originally in different partitions than A_i, and f_i is the corresponding part of the right-hand side, such that the system stays consistent, $\widetilde{b} \in \mathcal{R}(\widetilde{A})$. The partitions $\widetilde{A}_i$ do not contain multiple copies of any rows, and thus stay full rank, since A in (1) has full rank. Note that the internal ordering of the rows within A_i, $\widetilde{A}_i$ and R_i does not affect the numerical properties of the block Cimmino method [12].

We now describe the row-replicated block Cimmino method which finds the solution through successive sums of projections onto the row-blocks of $\widetilde{A}$ as in the classical block Cimmino [6]. From the current iterate $x^{(k)}$, an iteration of the row-replicated block Cimmino is given by

$$\begin{aligned} \delta_i &= \widetilde{A}_i^{+}(\widetilde{b}_i - \widetilde{A}_i x^{(k)}), \quad i = 1, \dots, p, \\ x^{(k+1)} &= x^{(k)} + \omega \sum_{i=1}^{p} \delta_i, \end{aligned} \tag{7}$$

where $\widetilde{A}_i^{+}\widetilde{A}_i = \mathcal{P}_{\mathcal{R}(\widetilde{A}_i^T)}$ is the projection onto $\mathcal{R}(\widetilde{A}_i^T)$, $\widetilde{A}_i^{+}$ is the pseudo-inverse of $\widetilde{A}_i$, and ω is a relaxation parameter. This relaxation parameter is essential to obtain convergence, given that the matrix $\widetilde{A}$ has replicated rows, and thus no longer has full row rank [6]. Since the system stays consistent, starting from any initial solution $x^{(0)}$, the iterations of block Cimmino are guaranteed to converge to the minimum norm solution of (5) if $0 < \omega < 1/\rho(\sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(\widetilde{A}_i^T)})$ [6]. The iteration (7) of block Cimmino can be reformulated as

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \omega \sum_{i}^{p} \widetilde{A}_i^{+}(\widetilde{b}_i - \widetilde{A}_i x^{(k)}), \\ &= \widetilde{Q}_\omega x^{(k)} + \omega \widetilde{\mathcal{F}}, \end{aligned} \tag{8}$$

where $\widetilde{\mathcal{F}} = \sum_{i=1}^{p} \widetilde{A}_i^{+}\widetilde{b}_i$, and $\widetilde{Q}_\omega = I - \omega \widetilde{H}$, with $\widetilde{H}$ defined as

$$\widetilde{H} = \sum_{i=1}^{p} \widetilde{A}_i^{+}\widetilde{A}_i = \sum_{i=1}^{p} \mathcal{P}_{\mathcal{R}(\widetilde{A}_i^T)}. \tag{9}$$

In the following, we refer to $\widetilde{H}$ as the iteration matrix. Looking at the fixed point of the iterations (8), where $\omega > 0$, we obtain the system

$$\widetilde{H}x = \widetilde{\mathcal{F}}, \tag{10}$$

where $\widetilde{H}$ is semi-positive definite, being a sum of symmetric orthogonal projections. Since the system stays consistent, we use a stabilized block-CG following the approach introduced in [1] for full rank matrices.

This approach is identical to what is already used in the ABCD-Solver [1] except that we consider partitions with replicated rows. In this paper, when we refer to the *replicated block Cimmino method*, we also include the use of a block-CG acceleration. Thus, we also denote the replicated block Cimmino method by BC. We now introduce two complementary explanations, one geometric and the other algebraic, for how the replication of rows is able to accelerate the convergence of the block Cimmino method.

2.1 Geometric interpretation as overlapping domain decomposition

In [12, Chapter 2], we propose an interpretation of the block Cimmino row projection method, introduced in Section 1, as a *Domain Decomposition Method* (DDM) where the subdomains are defined by the partitions A_i. In fact, the block Cimmino iterations are equivalent to applying damped block-Jacobi iterations on the normal equations of the partitioned system [6]. The damped block-Jacobi iterations are shown to be an additive Schwartz method *without overlap* in Section 1.2 of [5]. This interpretation gives a natural explanation of how the block Cimmino iterations construct a solution. First, the solution inside the subdomains are computed independently, which corresponds to the local computation of the projection for each partition. Then the information from these local solutions is propagated to the neighbouring subdomains via the interface separating them, which corresponds to the global sum of projections performed on the columns interconnecting the partitions pairwise.

Starting from this interpretation of the classical block Cimmino iterations, we now consider the replicated block Cimmino method, as defined in (7). In this method, a row is no longer associated with a single partition A_i, but can instead be common to several pairwise non-disjoint partitions $\widetilde{A_i}$. Hence, this approach can be naturally interpreted as an overlapping DDM where the overlap between subdomains is defined by the replicated rows.

In the context of systems obtained from discretized PDEs, there is an extensive literature showing how increasing the size of the overlap between subdomains improves the convergence of classical overlapping DDM. For more details, see e.g. [5, 15]. In the context of BC, the partitions, or subdomains, are typically constructed using algebraic methods [14, 3]. Techniques for the algebraic construction of overlapping subdomains were proposed, based on the extensive scan of the system matrix to group strongly coupled degrees of freedom, e.g. in [13], and their positive effect was shown for some classical domain decomposition methods. The replication methods we propose in this paper, see Section 3.3, are also based on the algebraic properties of the matrix, combined with graph partitioning approaches.

Based on this interpretation, we expect that, by introducing carefully chosen replicated rows inside the partitions, we are actually introducing overlaps between subdomains defined by the partitions, and may thus accelerate the convergence of BC.

2.2 Algebraic effect of replication

For the sake of clarity and simplicity, we consider the classical case of a disjoint two-block partitioning to construct the partitions $A_i \in \mathbb{R}^{n_i \times n}$. figure 1a shows a matrix in block tridiagonal form that has been partitioned with two-block partitioning. A two-block partitioning implies that the odd-numbered partitions (resp. even-numbered) are mutually orthogonal. We then reorder the matrix in the form $A = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$, where $B_1 \in \mathbb{R}^{m_1 \times n}$ contains the odd-numbered partitions and $B_2 \in \mathbb{R}^{m_2 \times n}$ contains the even-numbered partitions. It was proved by [6] that, in this case, if $m_{min} = min(m_1, m_2)$, the spectrum of the block Cimmino iteration matrix contains the eigenvalues

$$\begin{aligned} \lambda_k &= 1 + \cos \Psi_k & k &= 1, \ldots, m_{min} \\ \lambda_k &= 1 - \cos \Psi_{k-m_{min}} & k &= m_{min}+1, \ldots, 2m_{min} \;, \\ \lambda_k &= 1 & k &= 2m_{min}+1, \ldots, n \end{aligned} \tag{11}$$

where Ψ_k, $k = 1, \ldots, m_{min}$, are the principal angles between $\mathcal{R}(B_1^T)$ and $\mathcal{R}(B_2^T)$ as defined in (4). A *conjugate gradient* algorithm applied to (3) for such a two-block partitioning would not take more than $2m_{min}$ iterations to converge.

We now apply a row replication technique to the partitioned matrix in order to improve the convergence. Here, we force the replication to respect the two-block partitioning, i.e. no interconnections are introduced between two odd-numbered partitions or between two even-numbered partitions. We obtain the partitions $\widetilde{A}_i$ of size $\widetilde{n}_i = n_i + r_i$, with r_i the number of rows replicated from other partitions inside $\widetilde{A}_i$. figure 1b shows the block tridiagonal example matrix after such a replication. Since the replication respects the two-block partitioning, we obtain the form $\widetilde{A} = \begin{pmatrix} \widetilde{B}_1 \\ \widetilde{B}_2 \end{pmatrix}$. We reorder the two partitions such that

$$\widetilde{B}_1 = \begin{pmatrix} B_1 \\ R \end{pmatrix} \text{ and } \widetilde{B}_2 = \begin{pmatrix} R \\ B_2 \end{pmatrix}, \tag{12}$$

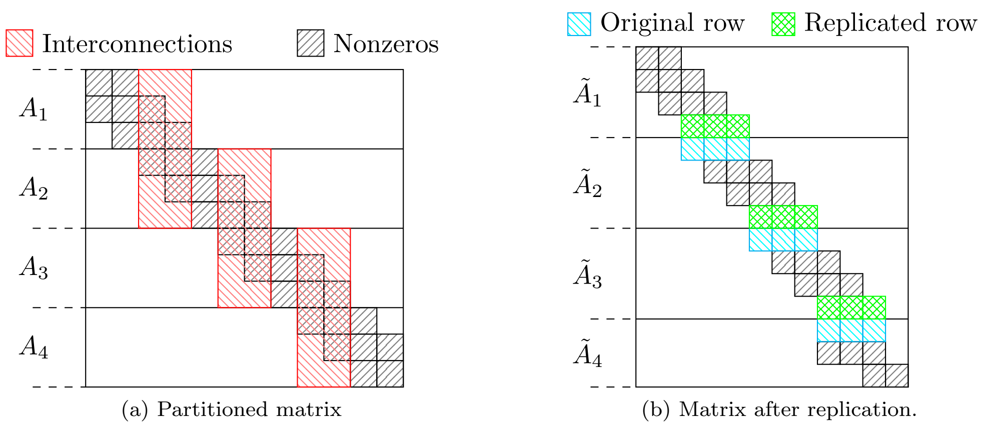


Figure 1: figure 1a shows the interconnection between partitions obtained from a two-block partitioning applied to a block tridiagonal matrix. figure 1b shows the same matrix after a replication preserving the two-block partitioning.

where $R \in \mathbb{R}^{n_r \times n}$ contains the rows shared by the two partitions. Now, let $Q_i = \widetilde{B}_i^T \widetilde{D}_i^{-\frac{1}{2}}$, with $\widetilde{D}_i = \widetilde{B}_i \widetilde{B}_i^T$, be an orthonormal matrix obtained from the QR factorization of $\widetilde{B}_i^T$. Then, the iteration matrix of the replicated BC is

$$\begin{aligned}\widetilde{H} &= \sum_{i=1}^{p} \widetilde{A}_i^+ \widetilde{A}_i \\ &= \sum_{i=1}^{2} \widetilde{B}_i^+ \widetilde{B}_i \\ &= (Q_1, Q_2)(Q_1, Q_2)^T.\end{aligned} \tag{13}$$

From the singular value theory [9, 10], the nonzero spectrum of (13) is then the same as the nonzero spectrum of the block tridiagonal matrix

$$(Q_1, Q_2)^T (Q_1, Q_2) = \begin{pmatrix} I_{\widetilde{n}_1} & Q_1^T Q_2 \\ Q_2^T Q_1 & I_{\widetilde{n}_2} \end{pmatrix}, \tag{14}$$

where the $Q_i^T Q_j$ are matrices whose singular values represent the cosines of the principal angles between the subspaces $\mathcal{R}(\widetilde{B}_1)$ and $\mathcal{R}(\widetilde{B}_2)$, and thus between the subspaces $\mathcal{R}(\widetilde{A}_i)$ and $\mathcal{R}(\widetilde{A}_j)$, as defined in [2]. If we consider (12), we can write the splitting

$$Q_1 = \begin{pmatrix} W_1 & W_R \end{pmatrix} \text{ and } Q_2 = \begin{pmatrix} W_R & W_2 \end{pmatrix}, \tag{15}$$

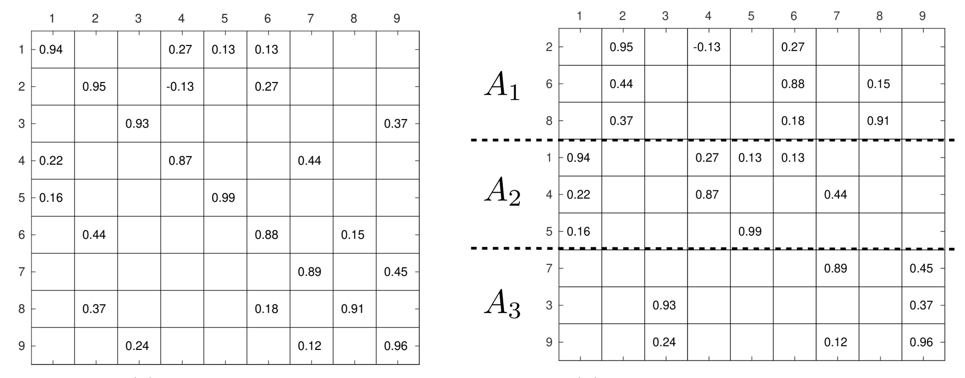
where W_i corresponds to B_i and W_R corresponds to R. Then their scalar product is reduced to

$$Q_1^T Q_2 = \begin{pmatrix} W_1^T \\ W_R^T \end{pmatrix} \begin{pmatrix} W_R & W_2 \end{pmatrix} = \begin{pmatrix} 0 & W_1^T W_2 \\ I_{n_r} & 0 \end{pmatrix}. \tag{16}$$

Since the eigenvalues of $Q_1^T Q_2$ give the cosines of the principal angles between $\mathcal{R}(\widetilde{A}_i)$ and $\mathcal{R}(\widetilde{A}_j)$, we obtain n_r cosines equal to 1 when using the replication technique. In the two-block partitioning case, the iteration matrix $\widetilde{H}$ from (9) has a spectrum similar to that presented in (11). For each replication, an eigenvalue has been shifted to 0 and another one to 2. The issue is then to know the part of the spectrum corresponding to what has not been replicated, i.e. $W_1^T W_2$ in (16).

In conclusion, in the case of a two-block partitioning combined with a replication strategy which respects the two-block form, we expect better convergence of BC thanks to the shifted eigenvalues. We see empirically that replication still shifts eigenvalues for any general partitioning but it is not easy to define what happens to the rest of the spectrum. The links between the subspaces corresponding to the partitions can be visualized as a large net where a row may connect more than two partitions. In this case, a row replication still shifts eigenvalues to 0 and 2. However, it is hard to control what is left in the spectrum of $\widetilde{H}$, since links can appear between previously unlinked partitions. In the worst case, the conditioning of the iteration matrix $\widetilde{H}$ for the replicated block Cimmino method may then be worse than the conditioning of the original iteration matrix H.

We consider the sample matrix from figure 2a that we use to illustrate the proposed method in the rest of this paper. The matrix is a 9×9 sparse unsymmetric matrix with 25 nonzero values and is partitioned into the three sets of rows $\{2, 6, 8\}$, $\{1, 4, 5\}$, and $\{7, 3, 9\}$ as seen in figure 2b. The spectrum of the associated iteration matrix H provides an insight into the convergence rate of block Cimmino. If the eigenvalues are well clustered around one, we expect a good convergence of the underlying block-CG algorithm. figure 3 shows the effects of various replications on the spectrum of $\widetilde{H}$ and its conditioning. Depending on the choice of replicated rows, the spectrum of $\widetilde{H}$ may be clustered around 1, e.g. replicating rows 4 and 7 into partition 3 as in figure 3b lowers the conditioning. Alternatively, e.g. by replicating row 8 into partition 3 as in figure 3c, a poor choice of replicated row can increase the conditioning of $\widetilde{H}$, since the small eigenvalues are unchanged but the largest eigenvalues increase. This shows that the choice of replicated rows is crucial.



	1	2	3	4	5	6	7	8	9
1	0.94			0.27	0.13	0.13			
2		0.95		-0.13		0.27			
3			0.93						0.37
4	0.22			0.87			0.44		
5	0.16				0.99				
6		0.44				0.88		0.15	
7							0.89		0.45
8		0.37				0.18		0.91	
9			0.24				0.12		0.96

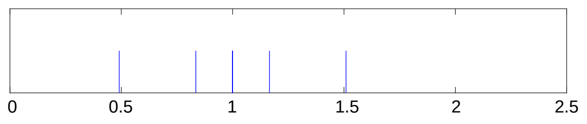
(a) Original matrix

		1	2	3	4	5	6	7	8	9
A_1	2		0.95		-0.13		0.27			
	6		0.44				0.88		0.15	
	8		0.37				0.18		0.91	
A_2	1	0.94			0.27	0.13	0.13			
	4	0.22			0.87			0.44		
	5	0.16				0.99				
A_3	7							0.89		0.45
	3			0.93						0.37
	9			0.24				0.12		0.96

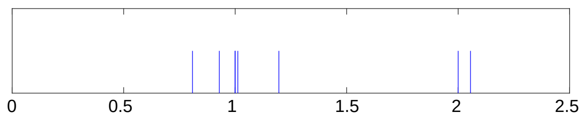
(b) 3-Way partitioned matrix

Figure 2: Sample Matrix before and after partitioning of the rows.

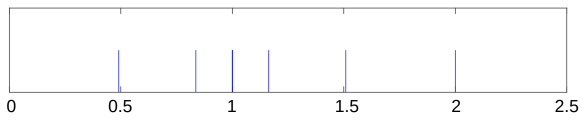
In the next section, we describe two methods to choose these rows based on graph partitioning techniques. In these replication methods, the idea is to target rows with high interconnections, hoping that the gain from replication is higher than the loss from added interconnections. These same rows correspond to the interface between algebraically constructed subdomains that we introduced



(a) Partitioned Matrix from figure 2b. $Cond(H) = 3.08$



(b) Rows 4 and 7 replicated into parts 3 and 2, resp. $Cond(\widetilde{H}) = 2.54$



(c) Row 8 replicated into part 3. $Cond(\widetilde{H}) = 4.07$

Figure 3: Eigenvalue spectrum of H for the partitioned matrix A before and after two choices of replication.

in the previous section.

3 Graph-based approaches for row replication

In this section, we propose two methods of choosing rows for replications. We first obtain an initial partitioning of the matrix, on which we then apply our replications. For this purpose, we use the GRIP row-block partitioning method [14] since it is shown in practice to give the lowest number of iterations for BC for the majority of the test problems, see e.g. [12, Chapter 3] and [14]. Furthermore, the proposed replication methods are based on the *row inner product graph* which is already constructed and used by GRIP. We can thus reuse the same graph, and reduce the preprocessing overhead. In this section, we first give background information on the row inner product graph and the GRIP partitioning, then we describe the two replication methods in detail using a simple example.

3.1 Background

In the *row inner product graph* model [14], a graph $\mathcal{G}(A) = (\mathcal{V}, \mathcal{E})$ is defined where the vertices $\mathcal{V}$ correspond to the rows of A, and the edges $\mathcal{E}$ correspond to the nonzero inner products between the rows of A. In other words, there is a vertex $v_i \in \mathcal{V}$ for each row r_i of A, and there is an edge $(v_i, v_j) \in \mathcal{E}$ for each nonzero inner product between distinct rows r_i and r_j, i.e. if $\langle r_i, r_j \rangle \neq 0$. In $\mathcal{G}(A)$, each edge (v_i, v_j) is assigned the cost $cost(v_i, v_j) = |\langle r_i, r_j \rangle|$, i.e. the absolute value of the corresponding inner product.

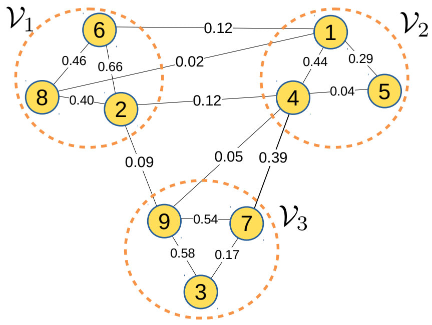
Let $\Pi = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_p\}$ be a mutually disjoint vertex partitioning of $\mathcal{V}$, i.e. we have

$$\begin{gathered} \mathcal{V}_i \cap \mathcal{V}_j = \emptyset, \quad i \neq j, \\ \bigcup_{i=1}^{p} \mathcal{V}_i = \mathcal{V}. \end{gathered} \tag{17}$$

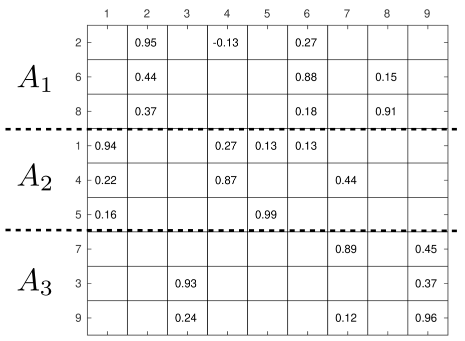
An edge (v_i, v_j) is said to be a *cut-edge* if the vertices v_i and v_j lie in different vertex parts. The cut-edge (v_i, v_j) is said to connect $\mathcal{V}_k$ and $\mathcal{V}_l$ if $v_i \in \mathcal{V}_k$ and $v_j \in \mathcal{V}_k$. The *cutsize* is then defined as the sum of the cut-edges costs. If (v_i, v_j) is a cut-edge then v_i and v_j are called *border vertices*. Each vertex is assigned a weight, here always equal to 1, and the weight of a part is the sum of the vertex weights in that part.

In the *graph partitioning problem*, the objective is to partition $\mathcal{V}$ into p disjoint parts such that the cutsize is minimized and part weights are balanced. In the partitioner GRIP, the row inner product graph is partitioned by using the state-of-the-art partitioning tool METIS [11]. The partitioning Π obtained in GRIP is used to determine the row-blocks of A, where the vertices in $\mathcal{V}_i$ determine the rows in A_i [14]. The partitioning objective is then to minimize the sum of inner products between different row-blocks, while maintaining a balance on the number of rows among row-blocks.

We consider again the test matrix A given in figure 2a. figure 4a shows a 3-way vertex partitioning Π obtained by GRIP which exploits the row inner product graph $\mathcal{G}(A)$. In $\mathcal{G}(A)$, there are 9 vertices and 15 edges that correspond to 15 nonzero row inner products. For instance, there is an edge (v_1, v_4) with $cost(v_1, v_4) = 0.44$ since $|\langle r_1, r_4 \rangle| = (0.94 \times 0.22 + 0.27 \times 0.87) = 0.44$. However, $\mathcal{G}(A)$ does not contain (v_1, v_2) since $|\langle r_1, r_2 \rangle| = (0.27 \times -0.13 + 0.13 \times 0.27) = 0$. figure 4b shows a permuted 3-way row-block partitioning of the matrix A where the row-blocks are defined by Π. With this row–block partitioning, the row inner product values between row–blocks are minimized and a good load balance is maintained by having three rows in each part.



(a) Graph $\mathcal{G}(A)$ with a 3-way partitioning Π



		1	2	3	4	5	6	7	8	9
A_1	2		0.95		-0.13		0.27			
	6		0.44				0.88		0.15	
	8		0.37				0.18		0.91	
A_2	1	0.94			0.27	0.13	0.13			
	4	0.22			0.87			0.44		
	5	0.16				0.99				
A_3	7							0.89		0.45
	3			0.93						0.37
	9			0.24				0.12		0.96

(b) Sample matrix with rows permuted according to Π

Figure 4: Row-block partitioning of the test matrix from figure 2, computed by the GRIP method.

3.2 Row replication

Once the vertex partitioning Π has been computed using the GRIP algorithm, we expect to have a good row–block partitioning of A in the sense that the eigenvalues of H have a good clustering around one, leading to good convergence of the block Cimmino method. Although most edges with a high cost should be kept inside the parts of the graph after the partitioning, some edges may still remain between the parts, i.e. in the cut of Π. This may be caused by several factors such as a strong load balance constraint in the partitioning, some vertices having a high degree, or the graph partitioning tool getting stuck in a local optimum.

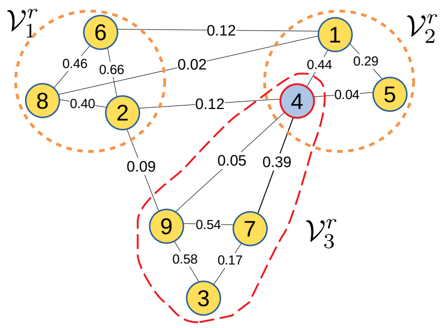
If edges with high costs remain in the cut of Π, then it may affect the conditioning of H adversely, resulting in slower convergence than expected. The replication schemes that we propose in Section 3.3 aim to reduce the negative effects of these cut-edges on the eigenvalues of H.

Given a graph $\mathcal{G}(A) = (\mathcal{V}, \mathcal{E})$, we define a *replicated partitioning* of $\mathcal{G}(A)$ as any partitioning $\Pi^r = \{\mathcal{V}_1^r, \mathcal{V}_2^r, \ldots \mathcal{V}_p^r\}$ where the vertex parts are not necessarily pairwise disjoint, i.e.

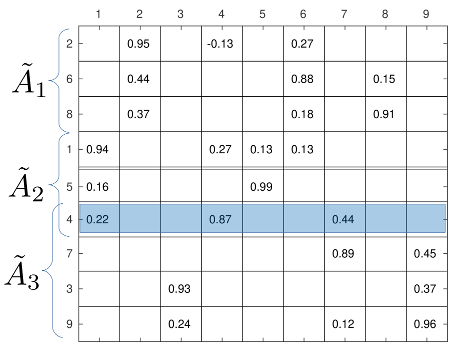
$$\begin{aligned} &\mathcal{V}_i^r \cap \mathcal{V}_j^r \neq \emptyset \text{ for some } i \neq j, \\ &\bigcup_{i=1}^{p} \mathcal{V}_i^r = \mathcal{V}. \end{aligned} \tag{18}$$

In the matrix representation, the replication of vertices corresponds to replicating rows between partitions. Since a cut-edge (v_i, v_j) in Π corresponds to a row couple r_i and r_j which adversely affects the angles between the subspaces spanned by the corresponding row-blocks, the row replication aims to improve the spectrum of the iteration matrix $\widetilde{H}$ of the replicated BC.

figure 5 shows an example of replication, giving overlapping parts, applied to the partitioned matrix in figure 4b. In figure 5a, v_4 is replicated into $\mathcal{V}_3^r$ from $\mathcal{V}_2$. Thus, the edges (v_4, v_7) and (v_4, v_9) do not contribute to the cutsize after the replication of v_4. On the other hand, after replicating v_4 into $\mathcal{V}_3^r$, a new cut-edge (v_4, v_2) appears, which connects $\mathcal{V}_3^r$ and $\mathcal{V}_1^r$. figure 5b shows the corresponding replication in the matrix representation. Here, row r_4 is replicated into $\tilde{A}_3$, which incurs an overlapping partitioning by having r_4 in both $\tilde{A}_2$ and $\tilde{A}_3$.



(a) Replicated partitioning Π^r of graph $\mathcal{G}(A)$, where v_4 is replicated to $\mathcal{V}_3^r$



		1	2	3	4	5	6	7	8	9
$\tilde{A}_1$	2		0.95		-0.13		0.27			
	6		0.44				0.88		0.15	
	8		0.37				0.18		0.91	
$\tilde{A}_2$	1	0.94			0.27	0.13	0.13			
	5	0.16				0.99				
$\tilde{A}_2$, $\tilde{A}_3$	4	0.22			0.87			0.44		
$\tilde{A}_3$	7							0.89		0.45
	3			0.93						0.37
	9			0.24				0.12		0.96

(b) Partitioned Matrix according to Π^r

Figure 5: Row replication applied to the test matrix from figure 4.

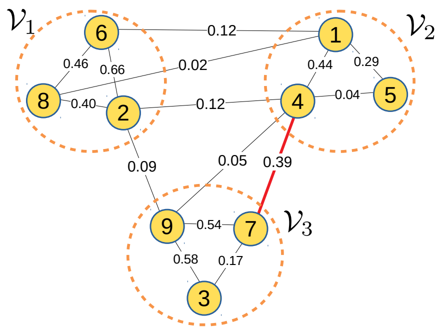
3.3 The replication methods

We propose two graph-based replication methods using the graph $\mathcal{G}(A)$.

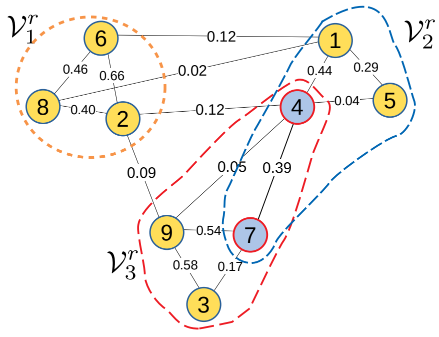
3.3.1 Duplication method (DM)

In this method, the resulting vertex partitioning Π of $\mathcal{G}(A)$ is used directly to determine the candidates for replication. The method successively chooses the cut-edge with highest cost, and replicates the corresponding border vertices into the connected parts. The replication selection continues until the total number of replicated vertices reaches a predefined limit, or all the border vertices are replicated.

figure 6 shows the graphs before and after replication of two vertices. figure 6a shows the sample partitioning Π which is given to DM as an input. As shown in figure 6a, the red edge (v_4, v_7) is the cut-edge with the highest cost and, therefore, border vertices v_4 and v_7 are selected for replication. figure 6b shows the replicated partitioning Π^r after v_4 and v_7 are replicated into $\mathcal{V}_3^r$ and $\mathcal{V}_2^r$, respectively. With those replications, the edges (v_4, v_7) and (v_4, v_9) are no longer cut-edges. Note that after this replication, the cut-edge (v_4, v_2) now links $\mathcal{V}_3^r$ and $\mathcal{V}_1^r$, but we do not take this information into account in DM.



(a) Partitioning Π where the highest cut-edge cost is highlighted



(b) Replicated partitioning Π^r after replication of v_4 and v_7

Figure 6: Graph illustration of before and after replication of two vertices 4 and 7 for the partitioned graph of the sample matrix. This is one step of the DM replication applied to the test matrix from figure 2.

DM Algorithm: All pairs of border vertices $v_i \in \mathcal{V}_k$ and $v_l \in \mathcal{V}_z$ connected by a cut-edge (v_i, v_j), where $k \neq l$, are kept in a max-heap, where $cost(v_i, v_j)$ is used as the key value. After generating the heap, the edge with the highest cost from the top of the heap is extracted, and the corresponding vertices are replicated into the respective parts. The replication of a row into its own part is prevented to avoid making the row-blocks rank deficient. For example, v_i is replicated into $\mathcal{V}_z^r$ unless v_i is already a member of $\mathcal{V}_z^r$, and similarly for v_j into $\mathcal{V}_y^r$. The algorithm stops when the predefined replication ratio is reached, i.e. the number of replicated rows reaches a certain percentage of the matrix size, or the heap is empty.

3.3.2 Gain-based replication (GR)

The second method takes the additional cost of the newly introduced cut-edges into account after replication. In GR, we define a replication gain for each border vertex to estimate the reduction in the cut after replication of a vertex. The replication *gain* of replicating $v_i \in \mathcal{V}_y$ into $\mathcal{V}_z$ is the sum of the costs of all cut-edges connecting v_i to $\mathcal{V}_z$ minus the sum of all cut-edges that are connecting v_i to other parts than $\mathcal{V}_y$ and $\mathcal{V}_z$. Formally, for each vertex-part pair $(v_i, \mathcal{V}_z)$, where vertex $v_i \in \mathcal{V}_y$ is connected to $\mathcal{V}_z$ $(z \neq y)$, the gain is calculated as

$$gain(v_i, \mathcal{V}_z) = \sum_{v_j \in \mathcal{V}_z} cost(v_i, v_j) - \sum_{v_k \in \mathcal{V} \setminus (\mathcal{V}_y \cup \mathcal{V}_z)} cost(v_i, v_k). \tag{19}$$

Then, the vertex with highest gain is selected for replication. The replication continues until the total number of replicated vertices reaches a predefined limit.

table 1 gives the replication gains of the border vertices of the sample graph $\mathcal{G}(A)$ shown in figure 4a. In the table, there are two rows for each cut-edge, resulting in a total of 12 rows for 6 cut-edges. Each row shows how much gain is obtained in the cut, if that vertex is replicated into the specified part. In the third column of the table, we show the detailed computation of the replication gain using (19). For instance, the gain of the pair $(v_4, \mathcal{V}_3)$ is calculated as follows: in figure 6a, v_4 connects three cut-edges, two of which connect to $\mathcal{V}_3$, namely (v_4, v_7) and (v_4, v_9). The sum of costs of these edges is 0.44 (i.e. 0.39+0.05). However, the cut-edge (v_4, v_2) connects to the different part $\mathcal{V}_1$, and $cost(v_4, v_2) = 0.12$ must be subtracted. Thus, the replication of v_4 into $\mathcal{V}_3$ has a gain of $0.44 - 0.12 = 0.32$.

Table 1: Replication gains of border vertices sorted in decreasing order.

Vertex	Part	Calculation (Eq. (19))	Replication gain
v_7	$\mathcal{V}_2$	0.39	0.39
v_4	$\mathcal{V}_3$	$0.39 + 0.05 - 0.12$	0.32
v_1	$\mathcal{V}_1$	$0.12 + 0.02$	0.14
v_6	$\mathcal{V}_2$	0.12	0.12
v_9	$\mathcal{V}_1$	$0.09 - 0.05$	0.04
v_2	$\mathcal{V}_2$	$0.12 - 0.09$	0.03
v_8	$\mathcal{V}_2$	0.02	0.02
v_2	$\mathcal{V}_3$	$0.09 - 0.12$	-0.03
v_9	$\mathcal{V}_2$	$0.05 - 0.09$	-0.04
v_4	$\mathcal{V}_1$	$0.12 - 0.39 - 0.05$	-0.34

GR Algorithm: In the GR algorithm, a max-heap contains all the vertex-part couples $(v_i, \mathcal{V}_z)$ with their replication gain as the key value, where each pair corresponds to a border vertex $v_i \in \mathcal{V}_y$ connected to a separate part $\mathcal{V}_z$ $(z \neq y)$. Then, the algorithm chooses the couple with the highest gain and replicates the vertex into the specified part. For our sample graph, the replication of v_7 to $\mathcal{V}_2$ has the highest replication gain as seen in table 1. The algorithm stops when a predefined replication ratio is reached.

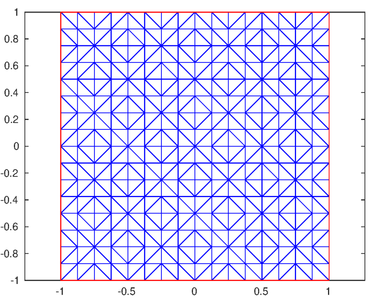
4 Numerical experiments

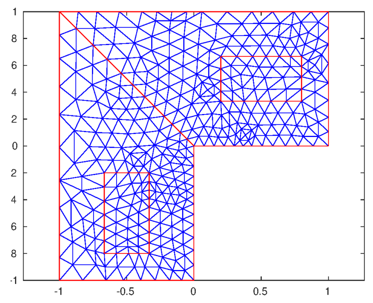
In this section, we assess the potential of the row replication techniques to accelerate the convergence of the block Cimmino method (BC). First, we consider linear systems arising from the discretization of challenging 2D PDE problems [7]. We emphasize the interpretation of our algebraic approach as an overlapping domain decomposition method [5] and show its ability to capture physically meaningful effects. Then we show parallel experiments on matrices from the SuiteSparse Matrix Collection[2] [4] with the replication method implemented inside the MPI-OpenMP parallel ABCD-Solver[3] code [16]. All runs are performed using the default block size of 4 for the block-CG in the solver.

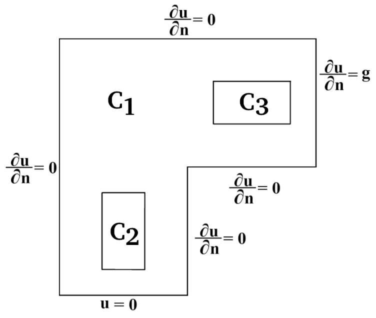
4.1 Effect of the replication techniques on PDE problems

We are interested here in the solution of discretized 2D PDE problems, inspired from [7]. These problems are defined on either the square domain $\Omega_\square = (-1,1) \times (-1,1)$, or an L-shaped domain $\Omega_\mathbf{L}$ built as the union of the three smaller square domains $(-1,0) \times (-1,0)$, $(-1,0) \times (0,1)$, and $(0,1) \times (0,1)$.

On these domains, we define a grid by applying several uniform refinement steps to an initial grid. figures 7a and 7b show the grids obtained with 2 refinement levels for $\Omega_\square$ and $\Omega_\mathbf{L}$, respectively. Concerning $\Omega_\mathbf{L}$, the initial grid splits the domain into several subdomains, and in particular two rectangular patches, $\mathbf{C_2}$ and $\mathbf{C_3}$, on which the PDE problem has specific properties.


(a) $\Omega_\square$ grid 2


(b) $\Omega_\mathbf{L}$ grid 2



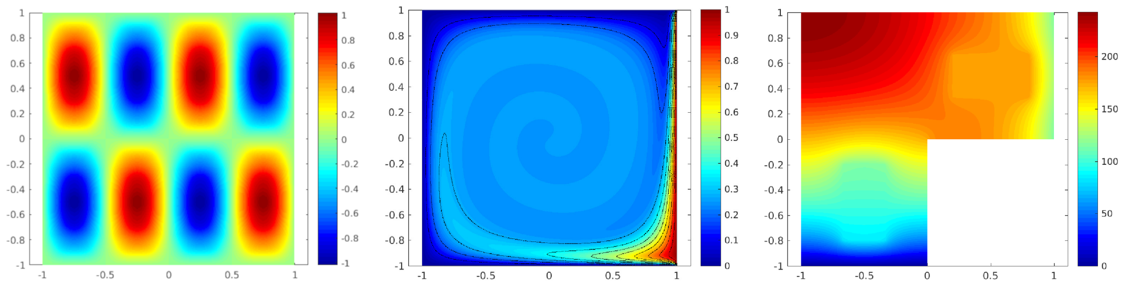
$\frac{\partial u}{\partial n} = 0$
C_1
C_3
$\frac{\partial u}{\partial n} = g$
$\frac{\partial u}{\partial n} = 0$
$\frac{\partial u}{\partial n} = 0$
C_2
$\frac{\partial u}{\partial n} = 0$
$u = 0$

(c) Diffusivity: $c_1 = 1$, $c_2 = 10$, $c_3 = 10^4$

Figure 7: *(a, b)* Structured grid after two levels of refinement on the initial grid. *(c)* Diffusivity and boundary conditions for the heterogeneous `diffusion` problem.

On these domains, we consider the three PDE problems:

1. a `Helmholtz` problem on $\Omega_\square$, with the wave number 40, see figure 8a,
2. a `convection-diffusion` problem with recirculating wind on $\Omega_\square$, see figure 8b,
3. a heterogeneous `diffusion` problem on $\Omega_\mathbf{L}$, based on diffusivity and boundary conditions defined as in figure 7c, see the corresponding solution in figure 8c.



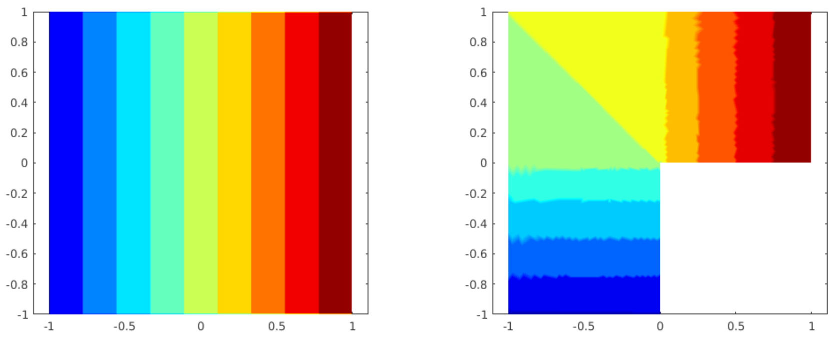
(a) `Helmholtz` (b) `convection-diffusion` (c) `diffusion`

Figure 8: Shape of the solution for the three PDE problems.

The three PDE problems are discretized on these grids using P1 finite elements. To apply our approach based on the BC method, we define a partitioning using the geometry of the domains. The solution to these linear systems is known to be challenging for linear solvers due to either heterogeneous coefficients in the domain, or to high frequencies that are difficult to capture on a coarse grid in the case of `Helmholtz` problems, or due to strong non-ellipticity with a dominant convection effect.

Small PDE problems with two-block partitioning

We first consider matrices obtained from the 3 discretized PDE problems above with 3 levels of refinement, obtaining systems of size 9.2×10^2 and 1.4×10^3 for the problems defined on $\Omega_\square$ and $\Omega_\mathbf{L}$, respectively. Here, we partition the system using the geometry so that we obtain a two-block partitioning for the matrix, see figure 9. As in Section 2.2, each row-block in the matrix is thus interconnected to only the previous and next row-blocks, in the sense that they share non-zero columns in the sparse structure.



(a) $\Omega_\square$ ($p = 9$) (b) $\Omega_\mathbf{L}$ ($p = 10$)

Figure 9: Two-block partitioning into p subdomains based on the geometry.

[2]`https://sparse.tamu.edu/`

[3]`http://abcd.enseeiht.fr/`

We now apply the replication methods DM and GR from Section 3.3, with an additional step which ensures that each replication respects the two-block partitioning structure. For each problem, and each replication method, we increase the number of replicated rows starting from 0 up to a number equivalent to the size of the original matrix. The results in terms of convergence of BC (or its replicated version) are displayed in figure 10. We observe for both DM and GR replication methods that increasing the number of replications steadily accelerates the convergence. In fact, in the case of the `diffusion` problem, we get up to a 50% and 70% decrease in the number of iterations for convergence using DM and GR, respectively, and the better performance of GR reflects the greater choice of replicable rows available to it.

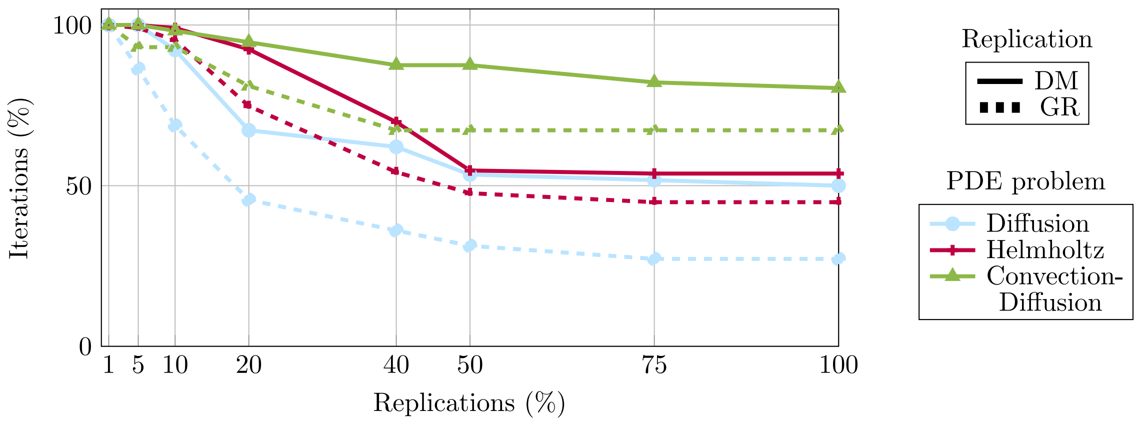


Figure 10: Effect of the replication on the convergence of BC for the three small PDE problems with two-block partitioning. The number of iterations is given relative to the case without replication. The number of replications is given as a percentage of the matrix size.

However, the application of a step to check that each replication respects the two-block partitioning is crucial. In figure 11, we include results for the `Helmholtz` problem when the replication is not forced to respect the two-block partitioning. For the case of the DM replication, although not respecting the two-block partitioning structure can increase the number of iterations above that with no replication (e.g. with a 10% replication ratio), the difference stays negligible. On the contrary, in the case of the GR replication, starting from a 20% replication ratio the number of iterations for convergence starts to increase dramatically. This increase can be explained by interconnections between previously unlinked partitions which hinders the convergence.

Small PDE problems with generic partitioning

In an actual application, the two-block partitioning of such PDE problems would not be realistic to handle. We now partition the same small systems using a more generic partitioning based on the geometry of the problem, see figure 12.

figure 13 displays for each problem and each replication method the finite elements corresponding to the replicated rows. In this figure, the color indicates the number of replications for each row (blue≡0, light blue≡1, red≡2). The color values are interpolated between finite elements, which explains the apparent color gradient between different areas. As expected, we observe that the

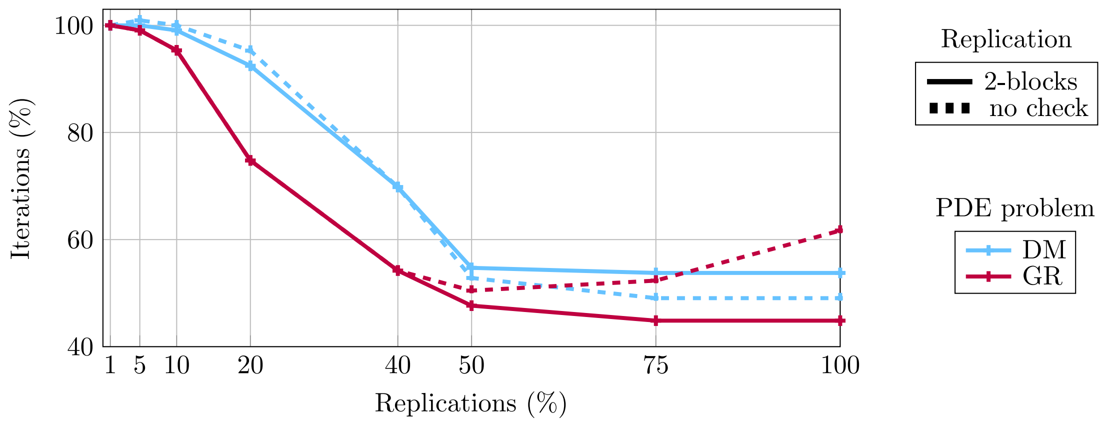
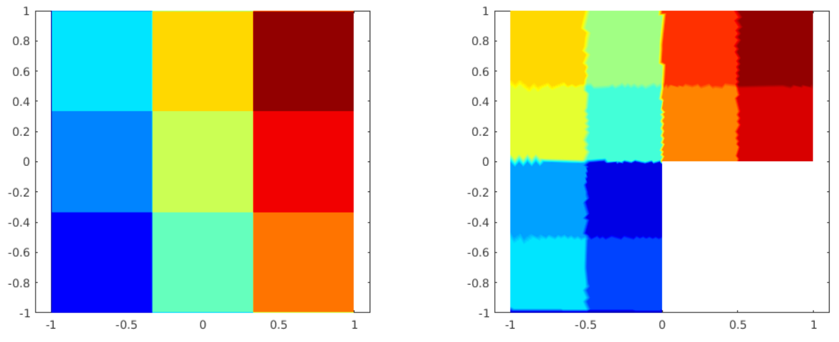


Figure 11: Effect of the replication on the convergence of BC for the small `Helmholtz` problem with two-block partitioning. The 2 replication methods, DM and GR, are applied with and without respecting the two-block partitioning. The number of iterations is given relative to the case without replication. The number of replications is given as a percentage of the matrix size.



(a) $\Omega_{\Box}$ $(p = 9)$ (b) $\Omega_{\mathbf{L}}$ $(p = 12)$

Figure 12: Generic partitioning into p subdomains based on the geometry.

replications are located around the interface between subdomains. Through the row replication, we are introducing an overlap between subdomains. This is particularly clear with the GR method where most of the interface nodes are replicated. In fact, by increasing the amount of replications, we would observe that the whole interface becomes replicated.

In the case of the DM method, the resulting overlapping is harder to interpret. By looking at the shape of the solution expected for the PDE problems, we observe some similarities. Our interpretation is that the DM method targets some physical effects of the PDE problem inside the overlapping. This means that by targeting the cut-edges with highest cost, the replication method focuses on strongly connected degrees of freedom (DOFs) from separate subdomains in the problem, and these DOFs correspond to nodes affected by the same physical effect in the PDE problem. In the case of the `convection-diffusion` problem, the overlapping resulting from DM follows the

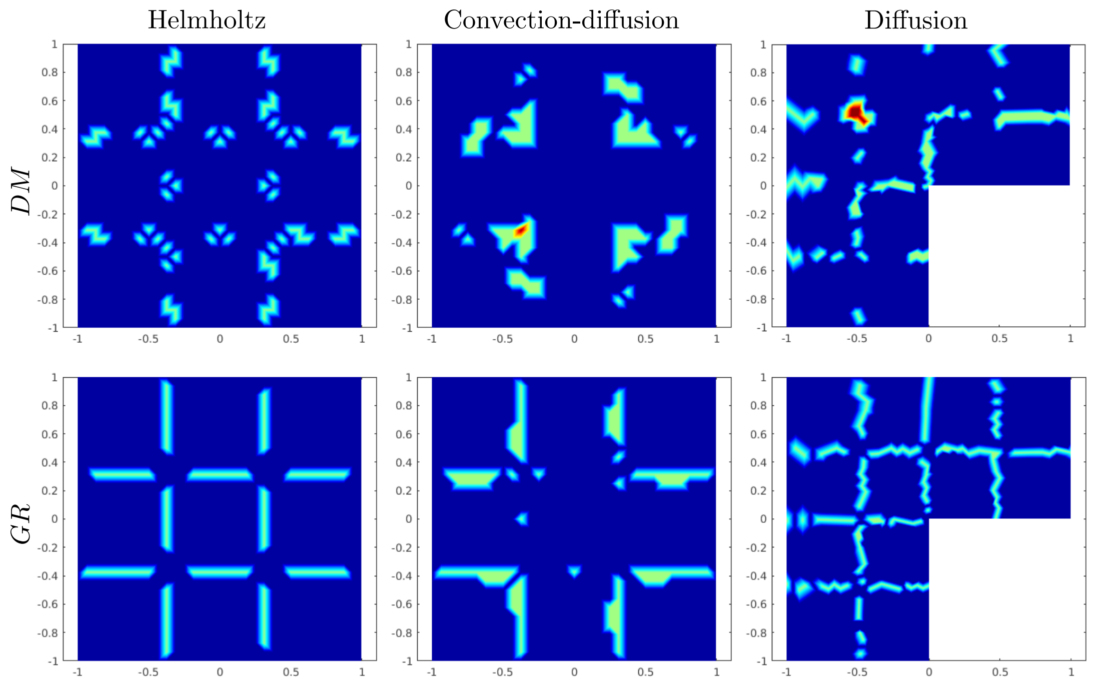


Figure 13: Location of the elements corresponding to replicated rows for the three discretized PDE problems. Both methods, DM and GR, use a number of replications up to 10% of the matrix size. The color indicates the number of replications: blue≡0, light blue≡1, red≡2.

recirculating wind surface lines around the interface between subdomains defined by the partitions. In the case of the `Helmholtz` problem, the overlapping is located on elements separated by fixed lengths which correspond to vibrations from the `Helmholtz` equation. figure 14 shows the result of the same replication method DM applied to the `Helmholtz` problem, with a wave number divided by four. The space separating the replications increases, and their form flattens, which suggests that the replications follow the wave form of the Helmholtz equation, supporting our previous interpretation.

Large PDE problems with generic partitioning

We now run the replicated BC method on larger systems to assess the effect of replicated rows on the convergence. We consider matrices of size $8.7{\times}10^5$ and $6.5{\times}10^5$ for the three PDE problems defined on $\Omega_\square$ and $\Omega_{\mathbf{L}}$, respectively, using the grids obtained from five refinement steps. figure 15 displays the number of iterations for convergence, relative to the number of iterations without replication, depending on the amount of replication. We vary the amount of replication from 0 (no replication) to 20% of the matrix size. In the case of the DM method, the replication of rows is overall positive, and we observe up to a 70% decrease in the number of iterations for the `diffusion` problem, and 50% for the `Helmholtz` and `convection-diffusion` problems. Only in the case of the `Helmholtz problem`, does the number of iterations first increase by around 10% then there is a dramatic decrease until 10% of replicated rows. However, in the case of GR, while the convergence is faster than using DM up to 5% replications, we then observe a steady increase in the number of iterations

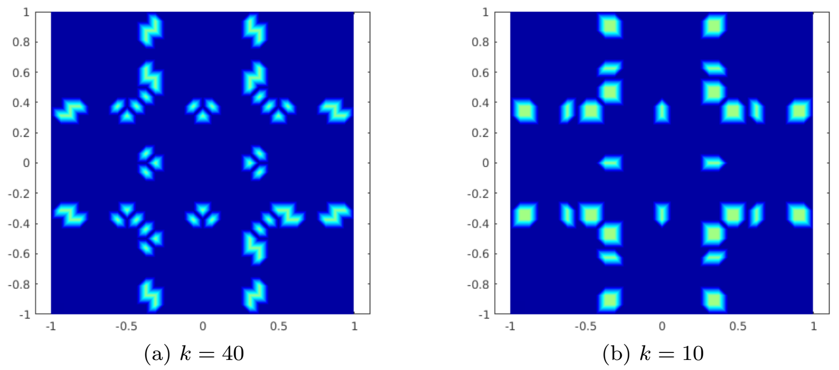


Figure 14: Location of the elements corresponding to replicated rows, using DM, for the `Helmholtz` problem depending on its wave number k. The color indicates the number of replications: blue≡0, light blue≡1, red≡2.

when increasing the amount of replication. This slower convergence has also been observed by the domain decomposition community, for example when solving the discretized Helmholtz problem [8, Section 2.3].

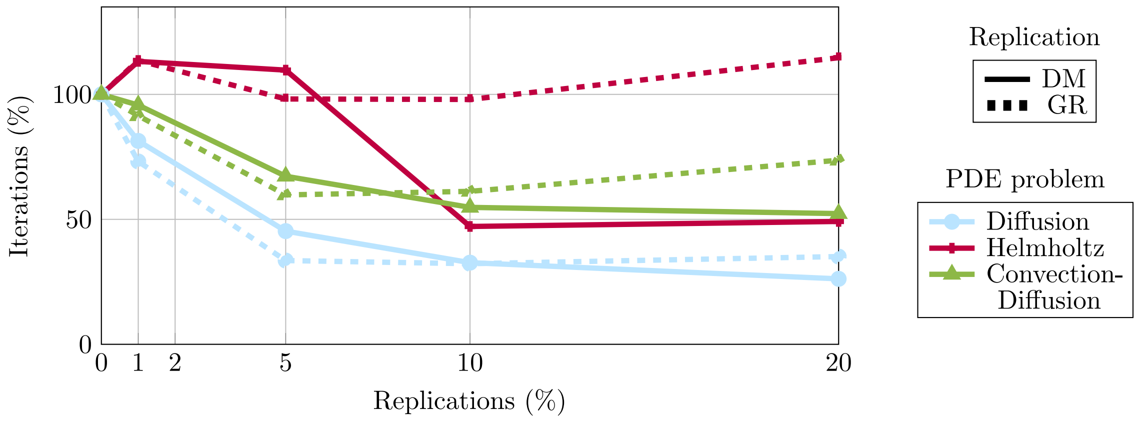


Figure 15: Effect of the replication on the convergence of the BC for the three PDE problems. The number of iterations is given relative to the case without replication. The number of replications is given as a percentage of the matrix size.

4.2 Parallel performance of the ABCD-Solver with replication

The block Cimmino method is intended for the solution of unsymmetric systems and, combined with the GRIP partitioner and our replication methods, is a purely algebraic approach. In this section, we are interested in the behaviour of the replicated BC, as implemented in the parallel

ABCD-Solver[4], when applied to the matrices from table 2, extracted from the SuiteSparse Matrix Collection[5] [4]. These matrices were selected in order to represent the different behaviours we encountered in practice when running the replicated BC.

Table 2: Characteristics of the test matrices. **m**: the size of the matrix. **elts per row**: the number of nonzero entries per row. **#Parts**: the number of partitions used.

Matrix	**m** ($\times 10^6$)	**elts per row**	**#Parts**	**Application**
`atmosmodl`	1.49	6.93	256	Computational Fluid Dynamics
`circuit5M_dc`	3.52	4.22	512	Circuit Simulation
`Goodwin_127`	0.18	32.38	32	Computational Fluid Dynamics
`memchip`	2.71	4.93	32	Circuit Simulation
`ss`	1.65	21.03	256	Semiconductor Process

Our experiments are performed on the cluster Kraken[6] from CERFACS. Kraken is a cluster with 6660 Skylake Intel Xeon Gold cores at 2.3GHz, for a theoretical peak performance of 490 TFLOPS/s. Each of its 185 compute nodes is a 2-socket system with 96 GB memory, where the 18 cores of each processor constitute a separate NUMA (non-uniform memory access) domain. Kraken uses the Intel OmniPath interconnect.

As a first step, we run the block Cimmino method on all the matrices from table 2 using 30 MPI processes (2 processes per node) with an amount of replication between 0 and 20% of the matrix size. All results were obtained as the average from using 10 different seeds in the GRIP partitioner [14] and running each configuration 5 times.

figure 16 shows the total execution time of the replicated BC, relative to the case without replication, depending on the amount of replication. Very similar to the results obtained with the PDE problems, we observe that the GR method gives better results in terms of convergence compared to DM for up to 2% replication. Then, GR starts to stagnate and eventually the convergence worsens for large amounts of replication. In the case of DM, we get an improvement in the total execution time up to 70% for the matrix `memchip`, and 60% for the other matrices. The matrix `circuit5M_dc` is a special case where the replicated BC systematically shows a larger execution time. For this matrix, convergence is obtained with BC in only 7 iterations whether or not we are applying replication. Thus, the extra cost induced by the replication cannot be amortized.

We now choose for each matrix and replication technique the best amount of replication in terms of execution time, and provide detailed execution times for specific parts of the parallel solver. The results are reported in table 3. In order to compute each projection, the linear system for each partition is solved using the direct solver MUMPS[7], see e.g. [16]. With respect to the factorization of these systems, we give the imbalance ratio computed as the longest factorization time for a projection system minus the shortest time divided by the average time. Once each process has computed its local projection, the distributed sum must be computed. These sums are performed using point-to-point communications between processes, restricted to the interconnections between their respective partitions. We thus give in table 3 the increase in the amount of communication, i.e. the number of shared columns between partitions, relative to the case without replication. Then, together with the number of iterations, we display the total execution time for the factorization and

[4]`http://abcd.enseeiht.fr/`
[5]`https://sparse.tamu.edu/`
[6]`https://cerfacs.fr/les-calculateurs-du-cerfacs/`
[7]`http://mumps.enseeiht.fr/`

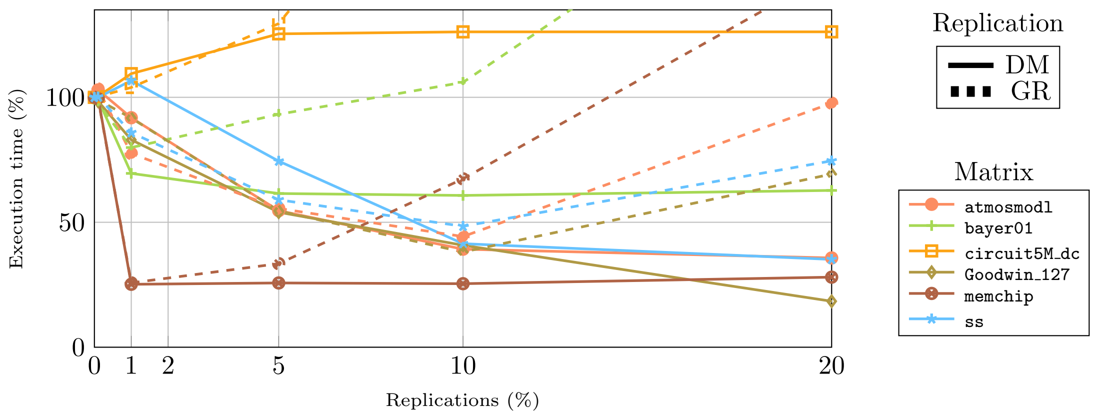


Figure 16: Effect of the replication on the ABCD-Solver. The execution time is given relative to the case without replication. The number of replications is given as a percentage of the matrix size.

the *block conjugate gradient* (block-CG) iterations, used in BC as introduced in Section 1. Finally, "sum proj. avg." is the timing to compute the local projections and their global sum (which is included in block-CG) averaged over the iterations.

Table 3: Impact of the replication on the efficiency of the ABCD-Solver. For each replication method, the best replication amount in terms of total runtime is chosen.

Matrix	**Replication**		**Facto.**	**Comm.**	**Iterations**	**Execution time (s)**		
	method	amount(%)	**imbalance**	**increase**		facto.	block-CG	sum proj. avg.
`atmosmodl`	None		0.66	1.00	280	20.28	154.37	0.46
	DM	20	0.68	3.17	63	24.43	**38.74**	0.49
	GR	10	0.67	2.76	89	23.60	53.62	0.48
`circuit5M_dc`	None		0.50	1.00	7	3.15	**13.47**	1.86
	DM	0.1	0.50	1.00	7	3.15	13.52	1.87
	GR	0.1	0.50	1.00	7	3.15	13.51	1.86
`Goodwin_127`	None		0.23	1.00	3072	0.55	68.70	0.02
	DM	20	0.24	4.11	419	0.34	**12.40**	0.02
	GR	20	0.34	4.55	519	0.38	47.60	0.09
`memchip`	None		0.58	1.00	211	2.24	249.55	1.09
	DM	1	0.54	2.90	49	2.09	**60.65**	1.14
	GR	1	0.59	2.86	51	2.25	62.85	1.09
`ss`	None		0.51	1.00	502	53.45	366.65	0.63
	DM	20	0.50	3.88	85	73.46	**77.11**	0.85
	GR	10	0.53	2.38	161	67.18	135.37	0.79

As expected, we observe that the amount of communication as well as the time to compute the sum of projections always increases when we include replications. In fact, the common nonzero columns between partitions, or interconnections, cause extra communications in the ABCD-Solver for the computation of the sum of projections, as explained in Section 1. Then, when we replicate a row inside a partition, all the columns corresponding to the entries in this row are interconnected with the partition from which the row originates. All the interconnections that were not originally present are then added, which increases the total number of communications. Additionally, while the imbalance of the factorizations can increase or decrease depending on the matrix, we observe longer execution times for this step in the case of the matrices `atmosmodl` and `ss`. This is natural since, through the replication, we add rows to some of the partitions. However, in the case of the matrices `circuit5M_dc`, `Goodwin_127`, and `memchip`, the execution time for factorization stays very similar or even smaller compared to the non-replicated case. These small variations could be caused by instabilities of the supercomputer.

The most important observation is that the execution time of the block-CG iterations, which is the main part of the computation in these cases, directly follows the decrease in the number of iterations. As observed before, we obtain an improvement of the execution time for the block-CG between 75% and 82% in most cases. In fact, the only case where the replicated BC method does worse in terms of execution time is `circuit5M_dc` where the number of iterations does not change. Finally, thanks to its greater effect in accelerating the convergence of BC, the DM replication method is best overall in terms of execution time.

5 Conclusion

We have proposed an iterative construction for overlapping between subdomains using graph techniques. This construction is based on a partitioning of the matrix obtained using the GRIP partitioner, which respects the numerical values of entries in the normal equations. Replication techniques are applied to accelerate the convergence of the block Cimmino method by shifting to 1.0 some cosines of principal angles between subspaces spanned by the partitions.

Through this replication, we have demonstrated an effective improvement of convergence, even for problems coming from challenging discretized PDEs. We also demonstrate an improvement of up to 82% in the resulting execution time of the ABCD-Solver.

For some matrices, the replication can still slow down the convergence. In particular, the GR method is very sensitive to interconnections between partitions appearing from the row replications. The simplest method, DM, is thus the most robust approach for accelerating the convergence of the block Cimmino method.

Bibliography

References

[1] M. Arioli, I. S. Duff, D. Ruiz, and M. Sadkane, *Block lanczos techniques for accelerating the block cimmino method*, SIAM Journal on Scientific Computing, 16 (1995), pp. 1478–1511.

[2] r. Björck and G. H. Golub, *Numerical methods for computing angles between linear subspaces*, Mathematics of computation, 27 (1973), pp. 579–594.

[3] Ü. V. Çatalyürek and C. Aykanat, *Patoh (partitioning tool for hypergraphs)*, in Encyclopedia of Parallel Computing, Springer, 2011, pp. 1479–1487.

[4] T. A. Davis and Y. Hu, *The university of florida sparse matrix collection*, ACM Transactions on Mathematical Software (TOMS), 38 (2011), pp. 1–25.

[5] V. Dolean, P. Jolivet, and F. Nataf, *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation*, SIAM, 2015.

[6] T. Elfving, *Block-iterative methods for consistent and inconsistent linear equations*, Numerische Mathematik, 35 (1980), pp. 1–12.

[7] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Finite elements and fast iterative solvers: With applications in incompressible fluid dynamics*, Oxford University Press., Oxford, 2014.

[8] O. G. Ernst and M. J. Gander, *Why it is difficult to solve helmholtz problems with classical iterative methods*, Numerical analysis of multiscale problems, (2012), pp. 325–363.

[9] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis, 2 (1965), pp. 205–224.

[10] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 4, JHU Press, 2013.

[11] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM Journal on scientific Computing, 20 (1998), pp. 359–392.

[12] P. Leleux, *Hybrid direct and iterative solvers for sparse indefinite and overdetermined systems on future exascale architectures*, PhD thesis, INP Toulouse, FAU Erlangen-Nürnberg, 2021.

[13] B. F. Smith, *Domain decomposition methods for partial differential equations*, in Parallel Numerical Algorithms, Springer, 1997, pp. 225–243.

[14] F. Torun, M. Manguoglu, and C. Aykanat, *A novel partitioning method for accelerating the block cimmino algorithm*, SIAM Journal on Scientific Computing, 40 (2018), pp. C827–C850.

[15] A. Toselli and O. Widlund, *Domain decomposition methods-algorithms and theory*, vol. 34, Springer Science & Business Media, 2006.

[16] M. Zenadi, *The solution of large sparse linear systems on parallel computers using a hybrid implementation of the block Cimmino method*, PhD thesis, EDMITT, 2013.