On the use of block low rank preconditioners for primal domain decomposition methods

Christophe Bovet, Théodore Gauthier, Pierre Gosselet

Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

On the use of block low rank preconditioners for primal domain decomposition methods

Christophe Bovet[1] | **Théodore Gauthier**[1,2] | **Pierre Gosselet**[2]

[1]Onera – The French Aerospace Lab, F-92322 Châtillon, France

[2]LaMcube, Univ. Lille / CNRS / Centrale Lille, F-59000, Lille, France

Correspondence
Christophe Bovet.
Email: christophe.bovet@onera.fr

Summary

This article investigates the use of the block low rank (BLR) factorization, recently proposed in the MUMPS solver, to define efficient and cheap preconditioners for primal domain decomposition methods, such as the Balancing Domain Decomposition method (BDD) and its adaptive multipreconditioned variant. To be scalable, these methods are equipped with an augmentation projector built from the local preconditioners nullspaces. The determination of these nullspaces is a complex task in the case of ill conditioned system, the use of block low rank compression makes this task even more complex as MUMPS' automatic detection no longer works properly. Two alternatives based on incomplete factorization with a well-chosen Schur complement are proposed. Also, the first massively parallel implementation of the adaptive multipreconditioned BDD solver (AMPBDD) is introduced. The performance of the methods is assessed with two weak scalability studies on problems up to 24576 cores and about 790 millions of unknowns, on the Sator and Topaze supercomputers. BLR preconditioning proves to be an interesting strategy both in terms of memory usage and time to solution for reasonably conditioned problems.

KEYWORDS
Domain decomposition, adaptive multipreconditioning, block low rank factorizations

1 | INTRODUCTION

In the last decade, non-overlapping domain decomposition methods have reached a high level of maturity, with sophisticated robustification techniques, and high performance implementations. Even though these questions are still the object of intense research, another question of interest is the ability to derive less numerically demanding variants of the methods which result in better performance in practice on sufficiently regular problems.

Dual methods (FETI[1], AMPFETI[2,3,4]) provide a zoology of preconditioners with variable quality and computational cost, allowing them to adapt to the conditioning of the system to be solved. However, they are less suited for the simulation of fracture problems, such as damage and crack propagation, due to their high sensitivity to the computation of the nullspace of local stiffness operators. A wrong estimation of these kernels leads to the divergence of the Krylov solver and/or the FETI system not being equivalent to the initial one anymore. The FETI-DP method[5] only partially solves this issue. Indeed, this approach enforces the continuity between subdomains of certain (generalized) degrees of freedom (like corner nodes or averages on faces/edges), leading to all Neumann problems being well-posed, without local nullspaces. But in the presence of propagating cracks, subdomains may split into several pieces which may generate internal mechanism not eliminated by the kinematic constraints.

Regarding this point, primal methods (BDD[6], BDDC[7]) exhibit greater robustness because their operator is based on Dirichlet interface condition. Initially, they only propose a Neumann preconditioner that in general use the full factorization of local Schur complements. The memory footprint of these factorizations can be a limiting factor to exploit modern supercomputers since the memory-per-core tends to decrease. For instance, the Milan nodes of the

new supercomputer, "Topaze", at TGCC (one french very large computational center), provide only 2Go of RAM per core.

In recent years, direct solvers such as MUMPS and Pastix have introduced compression techniques to reduce the memory footprint of factorization. For example, MUMPS utilizes low-rank blocks (BLR) to perform approximate solutions, thereby expediting the factorization and substitution steps. These new features offer the possibility of creating cost-effective preconditioners for the BDD method. However, a balance must be found between the compression ratio and the convergence speed of the iterative solver. The question of inexact preconditioners for domain decomposition methods has been investigated for a long time [8,9,10,11], in particular in the BDDC context [12,13]. As Dohrmann showed [12], inexact solves, made at the subdomain level or at the coarse grid level, must preserve the local operator nullspace to remain scalable.

The coarse problem of the BDD method is based on the nullspace of the local Neumann operator. The exact estimation of the nullspace is not mandatory for BDD since it comes into play at the preconditioner level. Nevertheless, the approximation caused by the compression process may remove the local nullspace and thus prevent scalability. Alternative coarse grid mechanisms have to be proposed in order to compensate this loss such as multipreconditioning and/or low energy modes.

The paper is structured as follows: the block low rank factorization is briefly introduced in Section 2. The primal domain decomposition is reminded in Section 3 and the use of BLR factorization to build the preconditioner is emphasized. The specific point of nullspace and generalized inverse computation is discussed in Section 4, then Section 5 provides scalability results on academic examples. Section 6 concludes the paper.

2 | BLOCK LOW-RANK METHODS IN A NUTSHELL

This section briefly introduces multifrontal sparse direct solvers [14] and block low-rank factorization [15] (BLR). In the context of domain decomposition methods, sparse direct solvers are used to solve local linear systems, i.e. linear systems defined at the subdomains level. The BLR factorization is only used for the preconditioner, in order to reduce the memory footprint of these local factorizations (see section 3.2).

For simplicity, we will consider the Cholesky factorization $\boldsymbol{L}^\top \boldsymbol{L}$ of a symmetric positive definite matrix $\boldsymbol{A} = (a_{ij})$. The general method is illustrated on the small matrix proposed by Liu [14]:

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & & & & & & & \\ & a_{22} & & & & & & \\ & & a_{33} & & & & & \\ & a_{42} & & a_{44} & & & \textit{sym.} & \\ & & a_{53} & & a_{55} & & & \\ & a_{62} & & & a_{65} & a_{66} & & \\ a_{71} & & & & & & a_{77} & \\ a_{81} & & a_{83} & a_{84} & a_{85} & & a_{87} & a_{88} \\ a_{91} & & & a_{94} & & a_{96} & a_{97} & & a_{99} \end{bmatrix}, \quad \boldsymbol{L} = \begin{bmatrix} \ell_{11} & & & & & & & \\ & \ell_{22} & & & & & & \\ & & \ell_{33} & & & & & \\ & \ell_{42} & & \ell_{44} & & & & \\ & & \ell_{53} & & \ell_{55} & & & \\ & \ell_{62} & & \ell_{64} & \ell_{65} & \ell_{66} & & \\ \ell_{71} & & & & & & \ell_{77} & \\ \ell_{81} & & \ell_{83} & \ell_{84} & \ell_{85} & \ell_{86} & \ell_{87} & \ell_{88} \\ \ell_{91} & & & \ell_{94} & & \ell_{96} & \ell_{97} & \ell_{98} & \ell_{99} \end{bmatrix}.$$

First, a symbolic Gaussian elimination is performed to obtain the structure of $\boldsymbol{L}$ accounting for the fill-in but not the lucky cancellations (filled-in entries appeared in red in the matrix $\boldsymbol{L}$). This structure is then used to build the Elimination Tree whose vertices are pivots to be eliminated (grouped together in practice but not in Figure 1 for simplicity). To each node n of the elimination tree is associated a dense Frontal Matrix or Front $\boldsymbol{F}_n$ constructed by assembling the entries of $\boldsymbol{A}$ corresponding to the future non zero-entries of $\boldsymbol{L}$ for the rows of the current pivots with the contributions of the previously eliminated pivots thanks to Update Matrices ($\boldsymbol{U}_m$) of the children (m) of n and an add-extension operation $\leftrightarrow$. Then a partial factorization is realized to obtain the final entries of $\boldsymbol{L} = (\ell_{ij})$ corresponding to the pivots of the current node and their contribution as an Update Matrix to the Frontal Matrices in the upper nodes. The tree is parsed from bottom to top and the factorization is complete once the root has been reached and treated.

These multifrontal methods have several advantages:

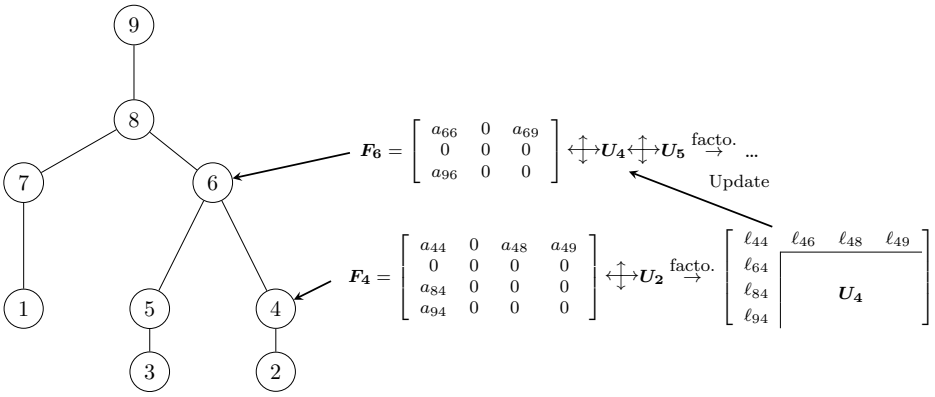


FIGURE 1 Following Liu[14], a case of a $\boldsymbol{L}^\top \boldsymbol{L}$ factorization denoting $\boldsymbol{L} = (\ell_{ij})$, the tree is explored in parallel from bottom to top.

- Different branches of a tree are independent, making it possible to handle computations in parallel.
- Frontal Matrices are dense which allows optimized dense factorization kernels.
- Update Matrix can be stored and applied at different steps allowing various strategies to manage memory.
- Sparsity of data can be further exploited by compressing the Front in low-rank format as explained below.

Before explaining block low-rank formats, low-rank matrices are defined. Let ϵ_p be a strictly positive threshold. Let n and k be non-negative integers such that $k \leqslant n$. Let $\boldsymbol{M}$ be a $n \times n$ matrix. The numerical rank of $\boldsymbol{M}$ at precision ϵ_p is k if and only if k is the lowest integer such that there exist $\boldsymbol{X_M}$ and $\boldsymbol{Y_M}$, $n \times k$ rectangle matrices, such that:

$$\left\|\boldsymbol{M} - \boldsymbol{X_M}\boldsymbol{Y_M}^\top\right\|_2 \leq \epsilon_p$$

with $\|\cdot\|_2$ the matrix norm induced by the Euclidean norm. The matrix $\boldsymbol{M}$ is said to be low-rank for a given accuracy ϵ_p when storing $\boldsymbol{X_M}$ and $\boldsymbol{Y_M}$ requires less memory rather than $\boldsymbol{M}$. Thus, $\boldsymbol{M}$ is low-rank whenever the following inequality holds:

$$k\,(n+m) \leq mn.$$

If so, the low-rank approximation $\widetilde{\boldsymbol{M}} = \boldsymbol{X_M}\boldsymbol{Y_M}^\top$ of $\boldsymbol{M}$ is stored as $(\boldsymbol{X_M}, \boldsymbol{Y_M})$. In practice $\boldsymbol{X_M}$ and $\boldsymbol{Y_M}$ are obtained directly through Singular Values Decomposition or Rank Revealing QR (RRQR) factorization of $\boldsymbol{M}$ in which case $\boldsymbol{X}$ is orthogonal. As well as reducing memory requirements, the low-rank format reduces the complexity of algebraic operations. Indeed, operations on and between two low-rank matrices $\widetilde{\boldsymbol{M}}$ and $\widetilde{\boldsymbol{N}}$ can be achieved directly – exactly or approximately – on $\boldsymbol{X_M}$, $\boldsymbol{Y_M}$, $\boldsymbol{X_N}$ and $\boldsymbol{Y_N}$ with reduced complexities. These operations include multiplication, addition, row or column swap, etc. Further explanations and computations of complexities regarding these operations can be readily found in the introduction of Bebendorf[16] or in Théo Mary[15].

Obviously, global finite element matrices are not low-rank, but some extradiagonal subblocks representing long-range interactions may be. Thus, a block low-rank (BLR) factorization of $\boldsymbol{A}$ is obtained by putting some of these off-diagonal blocks into low-rank format or "compressing" them whenever the expected gains are greater than the overhead of doing so. In practice, a graph based analysis is done as to decide which blocks will be compressed or not, rather than doing unnecessary RRQR factorizations to obtain the numerical rank. It permits assessing with a purely algebraic criterion the geometric distance between the nodes considered (see admissibility condition in Théo Mary[15]) as one should expect the rank to decrease exponentially with reasonable hypothesis made and on elliptical partial differential equations[16].

Low-rank factorization consists in compressing the Frontal Matrices in block low-rank format to carry on the partial factorization with reduced complexity and memory footprint. For this work, we have chosen to use the MUMPS library. The user has control on several parameters:

- the precision ϵ_{BLR} which differs from ϵ_p previously mentioned only to a scaling;
- the ordering of some operations, leading to two variants: UCFS and UFSC.

Let us explain the last two denominations. During the factorization of a given Front $\boldsymbol{F}$, a loop is done on its blocks. Data are accessed as late as possible within the Front $\boldsymbol{F}$, and its blocks are Updated (i.e. the contribution from pivots within the front are applied, step (U)) just before we start to treat them. Step (F) corresponds to the Factorization of the diagonal block before carrying out division by the unitary lower triangular matrix obtained on blocks below - the Solve step (S). To make numerical pivoting possible, it is needed to merge the Factor and Solve steps together, which requires Compression (C) to take place either before or after the Factor+Solve (FS) step. In either case, UCFS and UFSC make pivoting possible. The variant UCFS should reduce even more the complexity but at the cost of degraded numerical pivoting because it is done in low-rank, whereas UFSC should be more precise but will not benefit from early compression. It is stated in Théo Mary [15] that the downside of UCFS is barely noticeable as a degraded solution could quickly be improved through cheap iterative refinement steps. We have used both variants in the current work to investigate if any differences could be highlighted either way. Regarding the iterative refinement process, our numerical tests have shown little interest in our case, it will not be used in the following.

Finally, one should note that there are other methods exploiting numerical ranks in the literature. One might be interested in the brief review of some of these methods in Théo Mary [15], which states for instance that BLR factorization should be preferred for solving systems repeatedly while Hierarchically Semi-Separable matrices (HSS) [16] should be used for aggressive preconditioning. Since our goal is to build a cheap preconditioner which approximates precisely enough the action of some generalized inverse repeatedly, this advocates for the use of BLR factorization.

3 | PRIMAL DOMAIN DECOMPOSITION METHODS

This section briefly recalls the Balancing domain decomposition method (BDD [6]), its coupling with Adaptive Multipreconditioning (AMP [17]), and the use of inexact solvers.

3.1 | Balancing Domain Decomposition method in a nutshell

We consider a linear(ized) elasticity problem set on a domain Ω and discretized with the finite element method. This results in a large sparse linear system of equations of the form $\boldsymbol{Ku} = \boldsymbol{f}$ where $\boldsymbol{u}$ is the vector of unknowns and $\boldsymbol{f}$ the right-hand side. The operator $\boldsymbol{K}$ (stiffness matrix) is assumed to be symmetric positive definite.

Let $(\Omega^s)_{1\leqslant s\leqslant N_d}$ be a non overlapping partition of Ω such that: $\bar{\Omega} = \bigcup_{s=1}^{N_d} \bar{\Omega}^s$ and $\Omega^s \bigcap \Omega^p = \emptyset, \forall s \neq p$. The interface between the subdomains Ω^s and Ω^p is denoted by $\Upsilon^{sp} = \bar{\Omega}^s \bigcap \bar{\Omega}^p$, the union of all the interfaces of the subdomain Ω^s is denoted by Υ^s, and the union of the interfaces of all subdomains is denoted by Υ. In the substructured formulations, only local quantities (e.g. restricted to one subdomain) are assembled such as the matrices $\boldsymbol{K}^s$ and the right-hand-side $\boldsymbol{f}^s$. The global system is equivalent to the substructured formulation:

$$\boldsymbol{K}^s\boldsymbol{u}^s = \boldsymbol{f}^s + {\boldsymbol{T}^s}^\top \boldsymbol{\lambda}_b^s \quad \forall\ 1 \leqslant s \leqslant N_d, \tag{1}$$

$$\sum_{s=1}^{N_d} \boldsymbol{B}^s \boldsymbol{T}^s \boldsymbol{u}^s = \boldsymbol{0}, \tag{2}$$

$$\sum_{s=1}^{N_d} \boldsymbol{A}^s \boldsymbol{\lambda}_b^s = \boldsymbol{0}, \tag{3}$$

where $\boldsymbol{T}^s : \Omega^s \to \Upsilon^s$ is the trace operator, $\boldsymbol{A}^s$ and $\boldsymbol{B}^s$ are primal and dual assembly operators respectively (see [18] for their definition). The Lagrange multiplier field $\boldsymbol{\lambda}_b^s$ enforces the continuity of the primal unknown across the subdomains interfaces. From a mechanical point of view, equations (1) are the equilibrium of all subdomains, (2) corresponds to the continuity of the displacement across interfaces and (3) expresses the equilibrium of the interface (action-reaction principle).

All unknowns can be separated between internal unknowns (denoted with subscript i) and boundary ones (denoted with subscript b). Internal degrees of freedom can be eliminated in order to express (1)-(3) only in terms of boundary

unknowns

$$\boldsymbol{S}^s \boldsymbol{u}_b^s = \hat{\boldsymbol{f}}_b^s + \boldsymbol{\lambda}_b^s \quad \forall\, 1 \leqslant s \leqslant N_d, \tag{4}$$

$$\sum_{s=1}^{N_d} \boldsymbol{B}^s \boldsymbol{u}_b^s = \boldsymbol{0}, \tag{5}$$

$$\sum_{s=1}^{N_d} \boldsymbol{A}^s \boldsymbol{\lambda}_b^s = \boldsymbol{0}, \tag{6}$$

where $\boldsymbol{S}^s$ and $\hat{\boldsymbol{f}}_b^s$ are primal Schur complements and condensed right-hand sides. The vector $\boldsymbol{u}_b^s$ is the trace of the displacement at the boundary.

$$\boldsymbol{S}^s = \boldsymbol{K}_{bb}^s - \boldsymbol{K}_{bi}^s \boldsymbol{K}_{ii}^{s^{-1}} \boldsymbol{K}_{ib}^s, \tag{7}$$

$$\hat{\boldsymbol{f}}_b^s = \boldsymbol{f}_b^s - \boldsymbol{K}_{bi}^s \boldsymbol{K}_{ii}^{s^{-1}} \boldsymbol{f}_i^s. \tag{8}$$

Finally, we would like to point out that assembly operators are orthogonal in the following sense:

$$\sum_{s=1}^{N_d} \boldsymbol{B}^s \boldsymbol{A}^{s^\top} = \boldsymbol{0}, \tag{9}$$

which means that any local interface vector $\boldsymbol{u}_b^s$ can be uniquely defined as a combination of a balanced vector and a continuous one $\boldsymbol{u}_b^s = \boldsymbol{B}^{s^\top} \boldsymbol{u}_d + \boldsymbol{A}^{s^\top} \boldsymbol{u}_p$.

The BDD method writes the interface problem in terms of one unique primal global unknown $\boldsymbol{u}_p$ such that local interface vectors are given by $\boldsymbol{u}_b^s = \boldsymbol{A}^{s^\top} \boldsymbol{u}_p$ and (5) is satisfied by construction thanks to the orthogonality property of assembly operators (9). Few algebraic manipulations lead to the primal formulation:

$$\underbrace{\sum_{s=1}^{N_d} \boldsymbol{A}^s \boldsymbol{S}^s \boldsymbol{A}^{s^\top}}_{\boldsymbol{S}} \boldsymbol{u}_p - \underbrace{\sum_{s=1}^{N_d} \boldsymbol{A}^s \hat{\boldsymbol{f}}_b^s}_{\boldsymbol{f}_p} = \sum_{s=1}^{N_d} \boldsymbol{A}^s \boldsymbol{\lambda}_b^s = \boldsymbol{0}. \tag{10}$$

The global primal Schur complement $\boldsymbol{S} = \sum_{s=1}^{N_d} \boldsymbol{A}^s \boldsymbol{S}^s \boldsymbol{A}^{s^\top}$ is never built explicitly. Since this system is solved using a Krylov iterative solver, only the result of a multiplication by $\boldsymbol{S}$ is needed. This computation is well suited to parallel computers since $\boldsymbol{S}$ is a sum of local contributions. Also, in the present implementation, no Schur complements are explicitly computed, the action of these Schur operators are evaluated implicitly.

3.1.1 | Preconditioner

The BDD preconditionner $\boldsymbol{M}_{BDD}^{-1}$ mimics the additive structure of $\boldsymbol{S}$, it is chosen as a scaled sum of generalized inverse of primal Schur complements defined by

$$\boldsymbol{M}_{BDD}^{-1} = \sum_{s=1}^{N_d} \tilde{\boldsymbol{A}}^s \boldsymbol{S}^{s\dagger} \tilde{\boldsymbol{A}}^{s^\top}, \tag{11}$$

where $\tilde{\boldsymbol{A}}^s$ are scaled primal assembly operators such that $\sum_s \boldsymbol{A}^s \tilde{\boldsymbol{A}}^{s^\top} = \boldsymbol{I}_\Upsilon$, and the superscript $\boldsymbol{S}^{s\dagger}$ denotes for a generalized inverse of $\boldsymbol{S}^s$. Classical scaling operators are multiplicity scaling and stiffness scaling (often called k-scaling) [19].

The action of $\boldsymbol{S}^{s\dagger}$ is obtained by solving a local problem with Neumann boundary conditions. Depending on the natural boundary conditions of the problem, $\boldsymbol{S}^s$ may be singular. Corresponding subdomains are commonly qualified as “floating subdomains”.

3.1.2 | Coarse problem

The BDD preconditioner is applied to the residual of the Krylov solver $\boldsymbol{z} = \boldsymbol{M}_{BDD}^{-1}\boldsymbol{r}$. For floating subdomains, local right-hand-sides must lie inside the image of $\boldsymbol{S}^s$ which leads to the solvability conditions:

$$\boldsymbol{R}_{\boldsymbol{b}}^{s\top}\tilde{\boldsymbol{A}}^{s\top}\boldsymbol{r} = \boldsymbol{0}, \quad \forall s, \tag{12}$$

where $\boldsymbol{R}_{\boldsymbol{b}}^s$ is the nullspace of $\boldsymbol{S}^s$. We rewrite this condition as $\boldsymbol{C}^\top\boldsymbol{r} = \boldsymbol{0}$ with

$$\boldsymbol{C} = \left(\tilde{\boldsymbol{A}}^1\boldsymbol{R}_{\boldsymbol{b}}^1 \mid \ldots \mid \tilde{\boldsymbol{A}}^{N_d}\boldsymbol{R}_{\boldsymbol{b}}^{N_d}\right). \tag{13}$$

These conditions provide an additional coarse problem which is enforced using an augmented Krylov solver. An augmentation projector $\boldsymbol{\Pi}_C$ such that $\boldsymbol{C}^\top\boldsymbol{S}\boldsymbol{\Pi}_C = \boldsymbol{0}$ is defined, and the solution is sought as:

$$\boldsymbol{u}_p = \boldsymbol{u}_0 + \boldsymbol{\Pi}_C\tilde{\boldsymbol{u}}, \tag{14}$$

$$\text{with} \qquad \boldsymbol{u}_0 = \boldsymbol{C}(\boldsymbol{C}^\top\boldsymbol{S}\boldsymbol{C})^{-1}\boldsymbol{C}^\top\boldsymbol{f}_p, \tag{15}$$

$$\boldsymbol{\Pi}_C = \boldsymbol{I} - \boldsymbol{C}(\boldsymbol{C}^\top\boldsymbol{S}\boldsymbol{C})^{-1}\boldsymbol{C}^\top\boldsymbol{S}. \tag{16}$$

The system to be solved by the Krylov solver is finally:

$$\boldsymbol{S}\boldsymbol{\Pi}_C\tilde{\boldsymbol{u}} = \left(\boldsymbol{f}_p - \boldsymbol{S}\boldsymbol{u}_0\right). \tag{17}$$

This coarse problem provides a mechanism for rapidly propagating the mechanical load information to all the subdomains which is essential to build a scalable method.

3.2 | Block low rank BDD Preconditioner

In order to build a cheap preconditioner with a small memory footprint for the BDD method, the basic idea is to replace the full-rank resolution of local problems with a compressed resolution. The BLR BDD Preconditioner $\boldsymbol{M}_{BLR}^{-1}$ can be written as:

$$\boldsymbol{M}_{BLR}^{-1} = \sum_{s=1}^{N_d}\tilde{\boldsymbol{A}}^s\boldsymbol{S}_{BLR}^{s\dagger}\tilde{\boldsymbol{A}}^{s\top}, \tag{18}$$

where $\boldsymbol{S}_{BLR}^{s\dagger}$ stands for a resolution with a BLR factorization. Available choices for the scaling remain unchanged. The compression threshold ϵ_{BLR} and the variants (`UCFS`, `UFSC`) are parameters of the preconditioner. The former offers real flexibility in terms of the cost and quality of the preconditioner. For simplicity, all subdomains use the same threshold and variant. Mumps proposes an iterative refinement process when using compressed factorization. This iterative refinement is not used in the following since it represents a significant additional cost and there is no guarantee that the solution will be improved.

The main disadvantage of the compressed preconditioner concerns the floating subdomains and the coarse problem. Indeed, depending on the compression level, the nullspace of local operator of floating subdomains may be lost or Mumps may be not able to compute it correctly, thus leading to a degraded scalability. To overcome this problem, other ways of constructing the coarse problem are examined in Section 4. Another possibility to recover a coarse grid mechanism is to rely on multipreconditioning and not on local operators nullspace.

3.3 | Adaptive Multipreconditioning

Multipreconditioning was proposed for iterative solvers[20] and adapted to domain decomposition methods[2]. It is a strategy which exploits the additive structure of the preconditioner in order to generate as many search directions as subdomains. In the preconditioning step of conjugate gradient, instead of computing the preconditioned residual as $\boldsymbol{z}_i = \sum_{s=1}^{N_d}\tilde{\boldsymbol{A}}^s\boldsymbol{S}^{s\dagger}\tilde{\boldsymbol{A}}^{s\top}\boldsymbol{r}_i$, the following block of vectors is generated: $\boldsymbol{Z}_i = \left(\ldots\ \tilde{\boldsymbol{A}}^s\boldsymbol{S}^{s\dagger}\tilde{\boldsymbol{A}}^{s\top}\boldsymbol{r}_i\ \ldots\right)$. Of course the classical

direction writes $z_i = Z_i \mathbf{1}$, where $\mathbf{1}$ is the vector filled with ones. The idea of multipreconditioning is to let the algorithm find the optimal combination of directions under the form $z_i = Z_i \alpha_i$, where α_i is the unknown vector of subdomains' magnitude of contribution.

Multipreconditioning must be used in conjunction with full reorthogonalization, and it is a numerically expensive option. Nevertheless, it proved to be an efficient cure to FETI and BDD's bad conditioning situations. Indeed, it was proved[21] that multipreconditioning is a technique to approximate on the fly the bad modes that would be detected and eliminated by GENEO coarse spaces[22].

In order to limit the numerical costs of multipreconditioning, a clever adaptation strategy was proposed by Spillane[17] where the effectiveness of each direction is predicted based on a costless criterion, making it possible to accumulate directions which contribute weakly to the decrease of the error and limit the memory and CPU footprint. A large-scale assessment of this approach has been carried out for the FETI method[3]. The method was further improved with more sophisticated aggregation of search directions depending on the subdomains' connectivity[4].

As mentioned in previous subsection, BLR-preconditioning may cause the disappearance of nullspace modes and BDD-coarse space may not be a numerical necessity to preserve the well-posedness of Neumann problems. Even though there is a mechanical urge to preserve rigid body motions coarse space in order to comply with Saint-Venant's principle, we wish to evaluate the ability of multipreconditioning to naturally bring out this information and ensure scalability.

4 | COARSE PROBLEM COMPUTATION: NULLSPACE AND GENERALIZED INVERSES

The coarse problem of the BDD method relies on the computation local preconditioner nullspace and generalized inverses. The exact computation of these kernels is not mandatory since it plays at the preconditioning level. However, it still impacts the rate of convergence. In this section we propose three different methods to evaluate the defect k^s, the nullspace of $\boldsymbol{K}^s$ and its generalized inverse. For readability, we drop the exponent s of the local operator $\boldsymbol{K}^s$.

4.1 | Mumps automatic nullspace detection (M)

The first method is simply to use Mumps' capabilities to evaluate the operator nullspace and null pivots. There are two user defined parameters for the detection of the kernel dimension in Mumps, CNTL(1) and CNTL(3). The control parameter CNTL(1) is a relative threshold for numerical pivoting. The default value CNTL(1) $= 10^{-2}$ is used in this work. The second control parameter CNTL(3) is a threshold to detect null pivots. According to the documentation, a pivot is considered to be null if the infinite norm of its row/column is smaller than a threshold *thres*. The default value of CNTL(3) $= 0$ provides an automatic process to determines this threshold, $thres = \varepsilon \times 10^{-5} \times \|A_{pre}\|$ where A_{pre} is the preprocessed matrix to be factorized and ε is machine precision. A positive value of CNTL(3) leads to the user defined threshold $thres = CNTL(3) \times \|A_{pre}\|$.

As shown in a previous work[23], the automatic kernel detection can be put on severe test when dealing with ill-conditioned systems. Often, the automatic threshold does not detect the right kernel size. It is however possible to recover the right kernel with a user defined threshold, but the admissible range for CNTL(3) becomes narrow. If Mumps allows both BLR compression and nullspace calculation to be enabled, we expect the estimation of the correct nullspace to be even more complex in those cases.

4.2 | Incomplete factorization and fixing-nodes framework

The other two methods reuse the graph based approach proposed in a previous work[23]. This framework is briefly reminded here, we refer to the original paper and the references herein for more details. The overall methodology relies on the partial factorization of the operator and on the analysis of a well-chosen Schur complement. Let c be a

nonempty subset of $\{1, \dots n\}$, called fixing variables, the incomplete $\boldsymbol{L}\boldsymbol{L}^\top$ factorization is:

$$\boldsymbol{K} = \begin{bmatrix} \boldsymbol{K}_{\overline{cc}} & \boldsymbol{K}_{\overline{c}c} \\ \boldsymbol{K}_{c\overline{c}} & \boldsymbol{K}_{cc} \end{bmatrix} = \begin{bmatrix} \boldsymbol{L}_{\overline{cc}} & \boldsymbol{0} \\ \boldsymbol{L}_{c\overline{c}} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_{\overline{cc}}^\top & \boldsymbol{L}_{c\overline{c}}^\top \\ \boldsymbol{0} & \boldsymbol{S}_{cc} \end{bmatrix}. \tag{19}$$

The construction of c in the paper[23], based on graph centrality measures, not only ensures that $\boldsymbol{K}_{\overline{cc}}$ remain full-rank, but it minimizes its condition number, which is an important feature when dealing with large ill conditioned systems. A generalized inverse $\boldsymbol{K}^+$ of $\boldsymbol{K}$ is given by

$$\boldsymbol{K}^+ = \begin{bmatrix} \boldsymbol{L}_{\overline{cc}}^{-\top} & -\boldsymbol{L}_{\overline{cc}}^{-\top} \boldsymbol{L}_{c\overline{c}}^\top \boldsymbol{S}_{cc}^\dagger \\ \boldsymbol{0} & \boldsymbol{S}_{cc}^\dagger \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_{\overline{cc}}^{-1} & \boldsymbol{0} \\ -\boldsymbol{L}_{c\overline{c}} \boldsymbol{L}_{\overline{cc}}^{-1} & \boldsymbol{I} \end{bmatrix}. \tag{20}$$

Since the Schur complement $\boldsymbol{S}_{cc}$ is a small dense matrix, the use of the Moore-Penrose generalized inverse, obtained by SVD, is reliable and affordable here. From a practical point of view, once the fixing variables have been selected, the partial factorization is performed with the Mumps library. The fact that Mumps can activate both BLR compression and partial factorization is a very interesting opportunity here.

What remains to be done is to choose a criterion to determine Moore-Penrose generalized inverse of $\boldsymbol{S}_{cc}$, and a way to compute a basis of the nullspace.

4.2.1 | Low energy modes (E)

With this method, a relative criterion $\sigma_j \leq \epsilon\ \sigma_{max}$ is used to estimate the "null" singular values. The singular value decomposition of $\boldsymbol{S}_{cc}$ also provides the nullspace of the Schur complement $\boldsymbol{R}_c$ and the nullspace of the full matrix is deduced from $\boldsymbol{R}_c$

$$\boldsymbol{R} = \begin{bmatrix} -\boldsymbol{K}_{\overline{cc}}^{-1} \boldsymbol{K}_{\overline{c}c} \boldsymbol{R}_c \\ \boldsymbol{R}_c \end{bmatrix} \tag{21}$$

where $\boldsymbol{K}_{\overline{cc}}^{-1}$ makes use of the BLR compression. Here both the coarse space $\boldsymbol{C}$ and the coarse projector $\boldsymbol{\Pi}_C$ take into account the BLR compression.

4.2.2 | Hybrid geometric–algebraic detection (G)

In our experiments, it appeared that BLR compression may have a strong impact on the estimation of the defect (size of the nullspace) and on the basis input in the coarse problem, while Saint-Venant's principle urges us to preserve actual rigid body motions for the coarse problem. Thus, we propose another strategy inspired by the hybrid geometric–algebraic approach of Farhat and Géradin[24].

The method requires knowing the nullspace in the case of a totally floating subdomain. Let $\boldsymbol{R}_u$ be a basis of this totally unrestrained nullspace. In 3D elastostatics on connected domains, $\boldsymbol{R}_u$ is made of the six rigid body modes (3 translations and 3 rotations). The method of Farhat and Géradin[24] permits to calculate the combinations of rigid body motions which are not precluded by the Dirichlet conditions. These combination form the actual nullspace $\boldsymbol{R}$ of the subdomain.

Once the dimension k of the nullspace is known, the generalized inverse is computed using Equation (20) where the Moore-Penrose generalized inverse $\boldsymbol{S}_{cc}^\dagger$ considers that the k-smallest singular values are zero. This treatment differs from the original paper of Farhat and Géradin[24] where exactly k fixing nodes were deduced from the knowledge of the nullspace.

The hybrid geometric–algebraic method leads to coarse space $\boldsymbol{C}$ and projector $\boldsymbol{\Pi}_C$ being the same as those constructed without compression. Only the generalized inverse is impacted by the BLR compression.

5 | NUMERICAL EXPERIMENTS

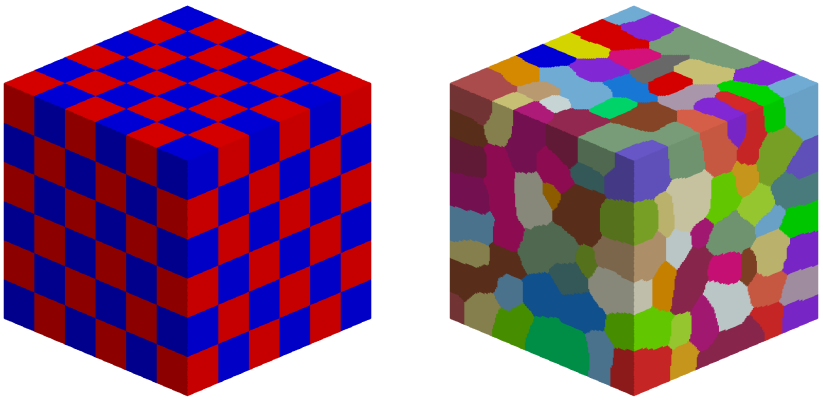
5.1 | Remarks on the implementation and dependencies

The proposed methods have been implemented in the finite element suite Z-Set 9.1[†]. In all configurations, the local direct solves are performed with the MUMPS library (version 5.5.1) [25]. MUMPS is linked with the BLAS library provided by Intel MKL. The coarse problem is solved with the Pardiso direct solver. The Eigen library[‡] is used for dense linear algebra. Communication are handled by the MPI protocol. The MPI library depends on the supercomputer used.

5.2 | Description of the weak scaling test case

For $n_c \in \{4, .., 16\}$, we consider a set of three-dimensional heterogeneous cubes made of n_c^3 identical sub-cubes (see Figure 2). Each sub-cube is discretized with the same ruled mesh made of $110,592$ eight-node brick elements (c3d8), leading to a total number of approximately $n_c^3 \times 206,763$ degrees of freedom. With this setup, the H/h ratio equals 40 where h is the diameter of the finite elements and H that of the subdomains.

The cube is clamped on one face and subjected to a prescribed unitary displacement in the three space directions on the opposite face, all other faces being traction-free. The material behavior is isotropic linear elastic, with a Poisson's coefficient of 0.3 and two values of Young's modulus assigned following a checkerboard pattern in order to obtain a coefficient jump E_r/E_b between two adjacent sub-cubes. Three ratios of Young's modulus are used: 10^0, 10^2 and 10^4. Finally, an unstructured decomposition in $N_d = n_c^3$ subdomains is obtained with a graph partitioning software which leads to interfaces not aligned with the heterogeneity. For a given number of subdomains, the partitioning is computed once and reused for all solvers configurations and for both coefficient jumps. The choice $N_d = n_c^3$, combined with the use of an automatic graph partitioning software leads to a lot of traversing heterogeneities that are known to strongly deteriorate the convergence of domain decomposition methods. Such a configuration is represented in Figure 2 for $n_c = 6$.



(a) Checkerboard cube. Red and blue areas correspond to the two different materials.

(b) Automatic decomposition. Each color represents a different subdomain.

FIGURE 2 Heterogeneous cube (configuration with $N_d = 216$, $n_c = 6$).

[†] http://www.zset-software.com/
[‡] http://eigen.tuxfamily.org/

n_c	N_d	#DOFs total	#cores
4	64	12.52M	384
6	216	41.99M	1,296
8	512	99.22M	3,072
10	1000	193.44M	6,000
16	4096	790.12M	24,576

TABLE 1 Checkerboard cube, weak parallel scalability: configurations.

All preconditioners make use of the stiffness scaling, they differ by the local operator $\boldsymbol{S}^{s\dagger}$ (with or without BLR compression) and the way to construct the coarse space $\boldsymbol{C}$. To make it easier to identify the method used to build the coarse problem, each method is assigned a letter (see the column Kernel in Table 2 for instance):

- M refers to the Mumps automatic nullspace detection (Section 4.1),
- G stands for the geometric–algebraic detection (Section 4.2.2),
- E corresponds to the low energy modes (Section 4.2.1).

The convergence is triggered when $\|\boldsymbol{r}_i\|/\|\boldsymbol{r}_0\| \leq \epsilon = 10^{-6}$. When AMPCG is used, the number of aggregates is 32 and the τ-test threshold is set to 10^{-2}.

Six cores are allocated to each subdomain, a shared memory parallelism is used at several steps including (but not limited to) local operators and coarse problem factorization. The study starts from 64 subdomains and goes up to 4096 subdomains which corresponds to a total number of 24,576 cores and 790.12 millions unknowns. Table 1 summarizes the different configurations.

5.3 | Weak scalability study on the Sator supercomputer

5.3.1 | Presentation of the hardware

Sator is Onera's in-house supercomputer. It is a parallel scalar cluster with 43,600 cores supplied by NEC. Thanks to three groups of computing nodes (Broadwell, Skylake and Cascade Lake), the Linpack performance of Sator is 1.8 PFlop/s. In this work, only the Cascade Lake partition has been used. It is made of 400 compute nodes with Intel Xeon "Cascade Lake 6240R" bi-processors (19,200 cores). Each node has 2×24 cores at 2.4 GHz and 192 GB of RAM (4GB RAM per core). The interconnection network is based on an Intel Omnipath 100Gbps fabric, in a Fat-tree topology. Communications are handled with Intel MPI 22.2.0. Since the largest queue in Sator is limited, the weak scalability only goes up to 3,072 cores in this section.

5.3.2 | Focus on a small test case ($N_d = 64$)

In order to reduce the number of calculations and select only the most promising configurations, the focus is made on the smallest test case with 64 subdomains and 384 cores. Several counters and timers are provided to compare the results:

- The size of the coarse problem is shown in column ***#C***.
- The column t(s) represents the total time of the simulation, including the construction and the factorization of the local operators, the computation of the coarse problem and the time spent in the iterations.
- The column $\boldsymbol{S}^{s+}$(Go) shows the memory footprint of the local preconditioner.
- For AMPCG, the number of search directions is given in column *#s.dir.* (for CG it equals the number of iterations since we use full reorthogonalization).

Heterogeneity $E_r/E_b = 10^0$				Solver CG			Solver AMPCG			
BLR	ϵ_{BLR}	Kernel	$\#\boldsymbol{C}$	#iter	t(s)	$\boldsymbol{S}^{s+}$(Go)	#iter	t(s)	$\boldsymbol{S}^{s+}$(Go)	#s.dir.
		M	192	65	72	2.5	59	71	2.5	90
UCFS	10^{-1}	M	0	169	109	1.4	168	112	1.4	199
UCFS	10^{-3}	M	0	120	90	1.8	115	93	1.7	146
UCFS	10^{-5}	M	0	257	165	2.1	58	77	2.1	358
UFSC	10^{-1}	M	0	168	108	1.4	168	112	1.4	199
UFSC	10^{-3}	M	0	115	89	1.8	120	94	1.7	151
UFSC	10^{-5}	M	0	257	167	2.1	62	79	2.1	347
		G	192	65	79	2.5	59	77	2.5	90
UCFS	10^{-1}	G	192	98	89	1.4	97	90	1.4	128
UCFS	10^{-3}	G	192	61	72	1.8	58	72	1.8	89
UCFS	10^{-5}	G	192	65	77	2.1	59	76	2.1	90
UFSC	10^{-1}	G	192	98	88	1.4	97	91	1.4	128
UFSC	10^{-3}	G	192	61	72	1.8	59	73	1.7	90
UFSC	10^{-5}	G	192	65	77	2.2	59	75	2.1	90
		E	192	65	81	2.4	59	78	2.4	90
UCFS	10^{-1}	E	192	165	125	1.3	151	122	1.3	182
UCFS	10^{-3}	E	192	66	75	1.6	63	76	1.6	94
UCFS	10^{-5}	E	192	65	77	2.0	59	76	2.0	90
UFSC	10^{-1}	E	192	151	119	1.3	151	121	1.3	182
UFSC	10^{-3}	E	192	66	75	1.7	63	76	1.6	94
UFSC	10^{-5}	E	192	65	78	2.0	59	76	2.0	90

TABLE 2 Small checkerboard cube, summary of the results with $E_r/E_b = 10^0$ (homogeneous case)

5.3.2.1 | Homogeneous probem

The results of the homogeneous test case are summarized in Table 2. This test case being well conditioned, all variants converge in less than 500 iterations. As expected, the convergence is strongly degraded when Mumps looses the nullspace due to the BLR compression. AMPCG is able to compensate for this loss for $\epsilon_{BLR} = 10^{-5}$, at the cost of a much larger search space. However, the multipreconditioning does not improve the convergence for moderate and large compression.

The geometric–algebraic (G) and the low energy mode (E) provide similar and much better convergence rates. They differ only for the highest level of compression where the geometric–algebraic method performs better. The low energy modes probably drift away from the original operator's nullspace for the highest compression. A degraded convergence is expected in this situation as shown by Dohrmann [12].

The BLR compression significantly reduces the memory footprint of the local preconditioner. The gain is about 40% for $\epsilon_{BLR} = 10^{-1}$, 27% for a moderate compression ($\epsilon_{BLR} = 10^{-3}$) and 20% for a small one ($\epsilon_{BLR} = 10^{-5}$). Interestingly, moderate and low compression improve both resolution time and memory footprint here (for both CG and AMPCG). Also, the geometric–algebraic method with high compression leads to the same total time than the uncompressed results while reducing the memory footprint of the preconditioner of 40%. Finally, the two BLR variants UCFS and UFSC lead to very similar results.

5.3.2.2 | Moderate heterogeneity

The results obtained with $E_r/E_b = 10^2$ are summarized in Table 3. Again, the convergence is strongly degraded when Mumps does not detect the correct nullspace. The (G) method performs better, especially for moderate and high compression. It is the only one that reaches convergence with CG for a high BLR compression. The multipreconditioning clearly improves the convergence and time to solution. However, both (M) and (E) do not reach convergence with a high compression. Regarding the difference between the two BLR variants, no clear trend can be identified. Finally, the memory gain provided by the compression seems not affected by the material heterogeneity.

5.3.2.3 | High heterogeneity

The results obtained with $E_r/E_b = 10^4$ are summarized in Table 4. The system is ill-conditioned due to the high heterogeneity. Mumps does not compute the correct coarse space even without BLR compression and very few configurations with the CG converge in less than 500 iterations.

Heterogeneity $E_r/E_b = 10^2$				Solver CG			Solver AMPCG			
BLR	ϵ_{BLR}	Kernel	$\#\boldsymbol{C}$	#iter	t(s)	$\boldsymbol{S}^{s+}$(Go)	#iter	t(s)	$\boldsymbol{S}^{s+}$(Go)	#s.dir.
		M	192	142	115	2.5	102	102	2.5	288
UCFS	10^{-1}	M	0	>500		1.4	>500		1.4	
UCFS	10^{-3}	M	0	345	204	1.8	232	168	1.8	515
UCFS	10^{-5}	M	0	335	209	2.2	124	120	2.1	572
UFSC	10^{-1}	M	0	>500		1.4	>500		1.4	
UFSC	10^{-3}	M	0	353	206	1.7	240	173	1.8	537
UFSC	10^{-5}	M	0	335	210	2.1	158	137	2.2	548
		G	192	142	125	2.5	102	110	2.5	288
UCFS	10^{-1}	G	192	320	209	1.4	308	211	1.4	370
UCFS	10^{-3}	G	192	208	153	1.7	138	125	1.8	324
UCFS	10^{-5}	G	192	142	120	2.1	106	108	2.1	261
UFSC	10^{-1}	G	192	323	211	1.4	319	215	1.4	350
UFSC	10^{-3}	G	192	221	160	1.7	147	129	1.8	333
UFSC	10^{-5}	G	192	142	120	2.1	107	110	2.1	262
		E	192	142	126	2.4	102	112	2.4	288
UCFS	10^{-1}	E	192	>500		1.3	>500		1.3	
UCFS	10^{-3}	E	192	297	206	1.6	215	174	1.6	399
UCFS	10^{-5}	E	192	142	122	2.0	106	109	2.0	261
UFSC	10^{-1}	E	192	>500		1.3	>500		1.3	
UFSC	10^{-3}	E	192	259	186	1.7	216	175	1.7	433
UFSC	10^{-5}	E	192	142	123	2.0	106	110	2.0	261

TABLE 3 Small checkerboard cube, summary of the results with $E_r/E_b = 10^2$ (moderate heterogeneity)

Heterogeneity $E_r/E_b = 10^4$				Solver CG			Solver AMPCG			
BLR	ϵ_{BLR}	Kernel	$\#\boldsymbol{C}$	#iter	t(s)	$\boldsymbol{S}^{s+}$(Go)	#iter	t(s)	$\boldsymbol{S}^{s+}$(Go)	#s.dir.
		M	149	>500		2.5	>500		2.5	
UCFS	10^{-1}	M	0	>500		1.4	>500		1.4	
UCFS	10^{-3}	M	0	>500		1.7	313	267	1.7	1189
UCFS	10^{-5}	M	0	>500		2.1	183	186	2.1	1122
UFSC	10^{-1}	M	0	>500		1.4	>500		1.4	
UFSC	10^{-3}	M	0	>500		1.7	286	254	1.8	1198
UFSC	10^{-5}	M	0	>500		2.1	>500		2.1	
		G	192	393	291	2.5	108	141	2.5	851
UCFS	10^{-1}	G	192	>500		1.4	393	308	1.4	877
UCFS	10^{-3}	G	192	>500		1.8	275	256	1.8	1046
UCFS	10^{-5}	G	192	393	276	2.1	108	140	2.1	895
UFSC	10^{-1}	G	192	>500		1.4	335	276	1.4	904
UFSC	10^{-3}	G	192	>500		1.8	229	222	1.7	1034
UFSC	10^{-5}	G	192	393	273	2.1	108	140	2.1	922
		E	192	393	288	2.4	108	143	2.4	851
UCFS	10^{-1}	E	192	>500		1.3	>500		1.3	
UCFS	10^{-3}	E	192	>500		1.6	327	301	1.6	1097
UCFS	10^{-5}	E	192	>500		2.0	223	232	2.0	1036
UFSC	10^{-1}	E	192	>500		1.3	>500		1.3	
UFSC	10^{-3}	E	192	>500		1.6	377	351	1.6	1213
UFSC	10^{-5}	E	192	>500		2.0	208	221	2.0	1056

TABLE 4 Small checkerboard cube, summary of the results with $E_r/E_b = 10^4$ (high heterogeneity)

Multipreconditioning is essential for convergence but only the (G) method provides satisfactory results with compression. The best results are obtained with a small compression where both the memory footprint and the total time are improved. For a larger level of compression, the number of iterations and the total time significantly increase, despite a larger search space.

5.3.3 | Weak scalability results

After analysing the previous results and in order to reduce the number of data, only the best configurations are shown in the following. Multipreconditioning is only considered for the moderate and high heterogeneity. The level of BLR compression is adapted to the heterogeneity of the material, the higher the heterogeneity, the lower the level of compression. Also, since the two variants UCFS and UFCS lead to very similar results, only UCFS is used in the following.

5.3.3.1 | Homogeneous probem

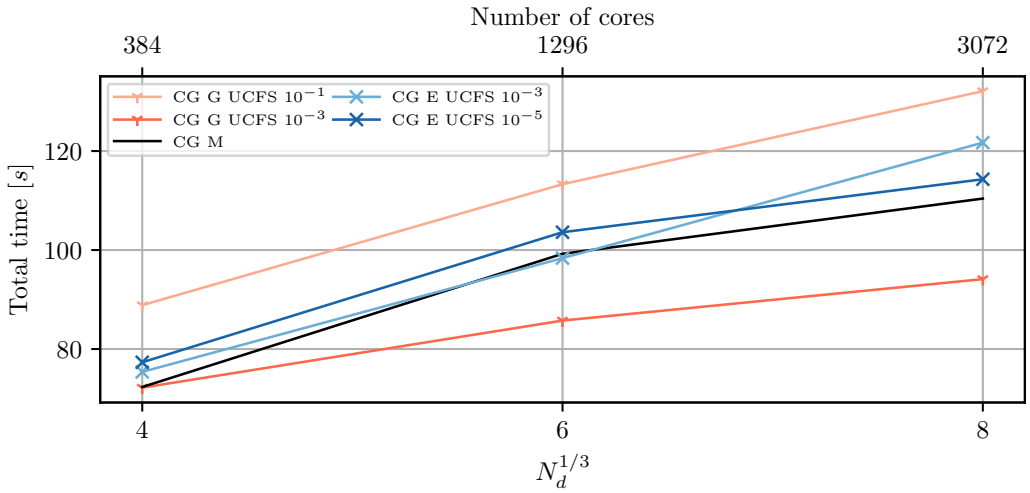
The parallel performance of the homogeneous test case are shown in Figure 3. The trends observed in Section 5.3.2 are confirmed. Moderate and low BLR compression do not significantly penalize the rate of convergence. As in section 5.3.2, (G) with a moderate compression ratio ($\epsilon_{BLR} = 10^{-3}$) converges faster, both in terms of total time and number of iterations. It sounds surprizing, but somehow the compressed preconditioner works better than the classic one. With a high compression ratio $\epsilon_{BLR} = 10^{-1}$, the convergence rate of (G) is slowed down significantly. The purpose of a such a configuration is mainly to reduce the memory footprint of the preconditioner. Variants (E) slightly increase the solution time due to a higher number of iterations and/or due to the overhead caused by the partial factorization (as observed in a previous work [23]).

5.3.3.2 | Moderate heterogeneity

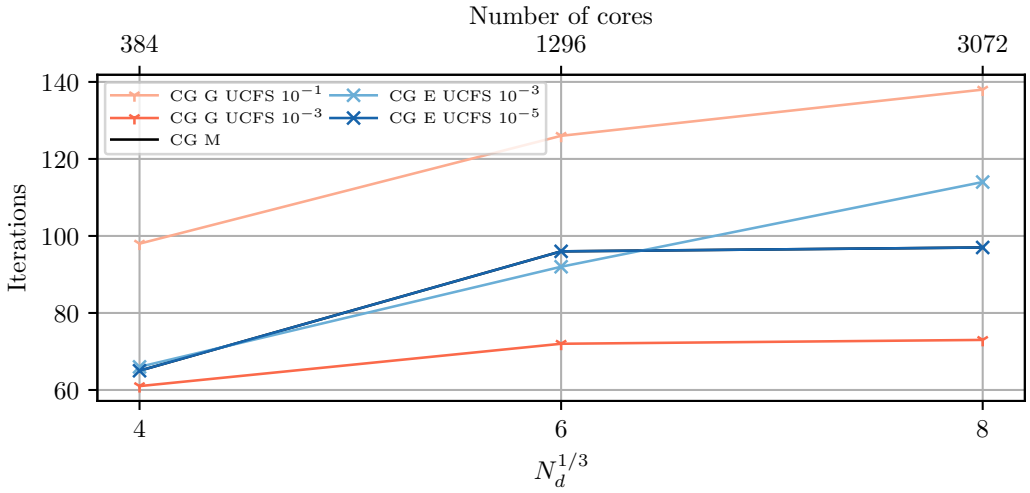
For the moderate heterogeneity test case, fives curves are considered: the CG solver without and with a low compression ratio, and the MPCG solver with a low compression ratio. Both (G) and (E) are considered when using compression. The parallel performance with $E_r/E_b = 10^2$ are shown in Figure 4. As before, a constant number of iterations is not expected due to the automatic domain decomposition. Also, the larger the problem, the larger is the condition number due to material heterogeneity. As expected, multipreconditioned solvers tend to be faster to converge thanks to an enlarged search space. The convergence rate of the CG solver is quite satisfactory and remains competitive in terms of time to solution. Both (G) and (E) give similar results in terms of number of iterations and search space size. However, the time to solution is much shorter for the (G) method, a closer look at the internal timers suggests that the time spent in backward and forward substitutions is faster with this method.

5.3.3.3 | High heterogeneity

Only two curves are shown for the highly heterogeneous test case: MPCG solver with geometric–algebraic nullspace detection, without and with low compression ratio. The MPCG solver with (E) without compression leads to the same convergence as with (G). The MPCG with (E) and BLR compression does not converge in less than 500 iterations for the test case with 3,072 cores. The parallel performance with $E_r/E_b = 10^4$ are shown in Figure 5. For this ill-conditioned test case, BLR compression slightly degrades the convergence rate but the time to solution and the search space size remain similar.

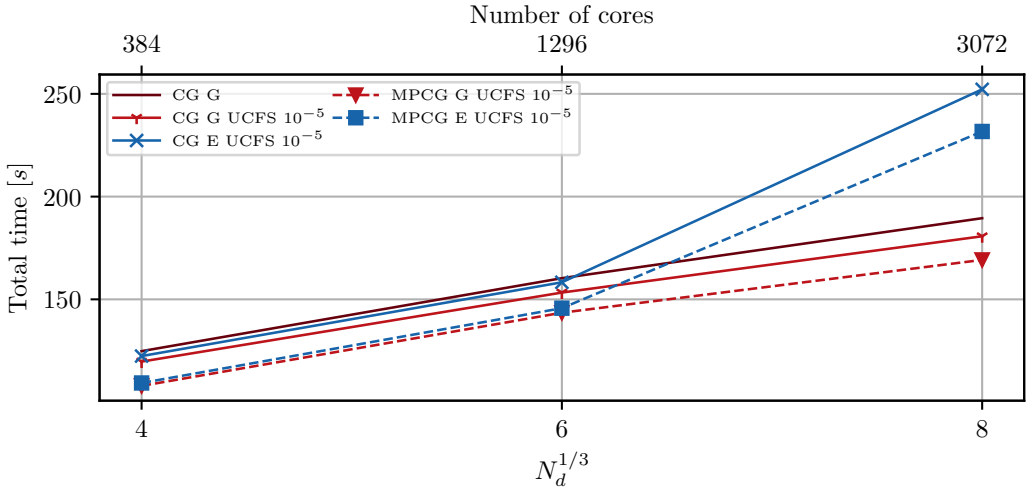


(a) Total wall time.

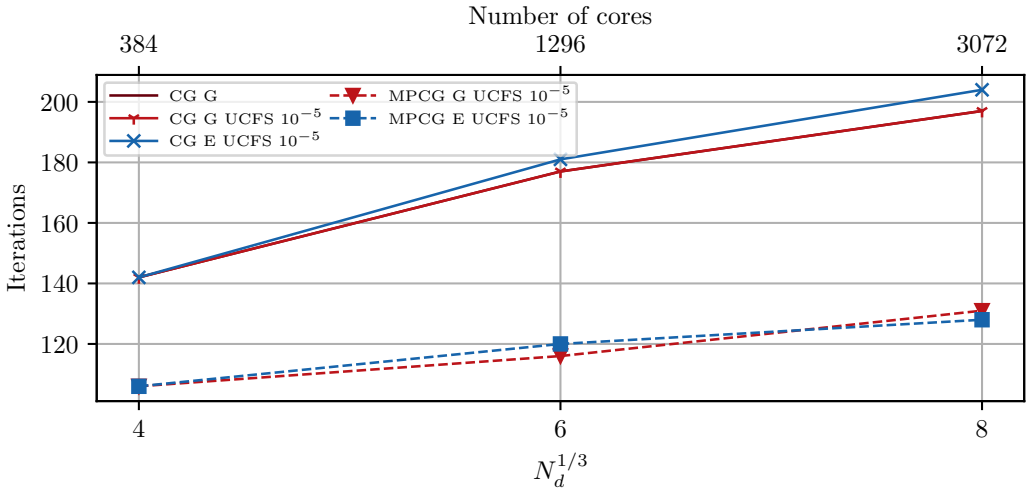


(b) Number of iterations.

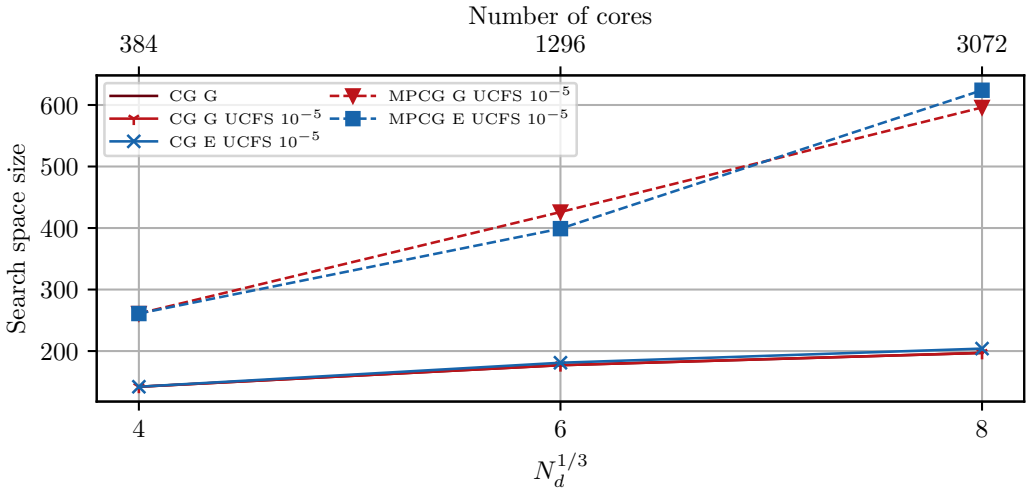
FIGURE 3 Checkerboard cube, weak parallel scalability (homogeneous case $E_r/E_b = 10^0$): total time and number of iterations (the minimization space size is equal to the number of iterations for CG). Sator supercomputer.



(a) Total wall time.

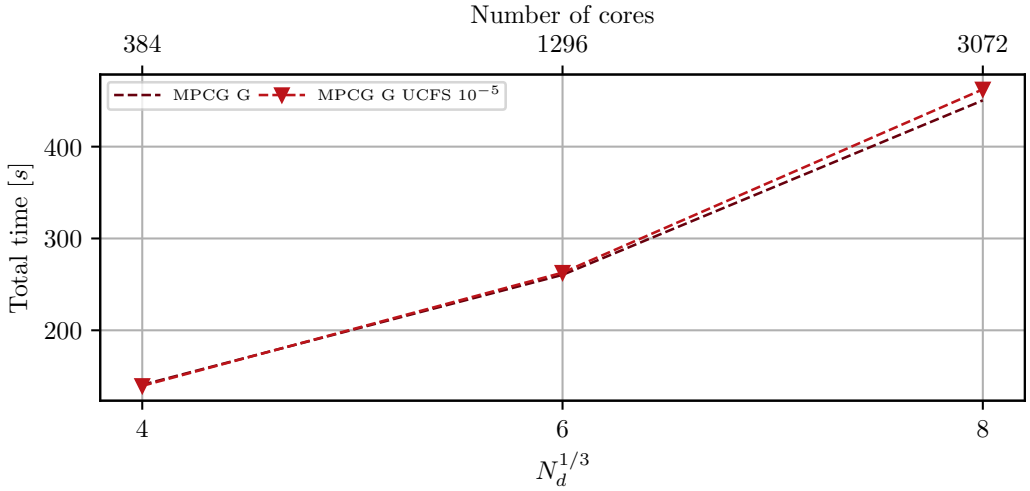


(b) Number of iterations.

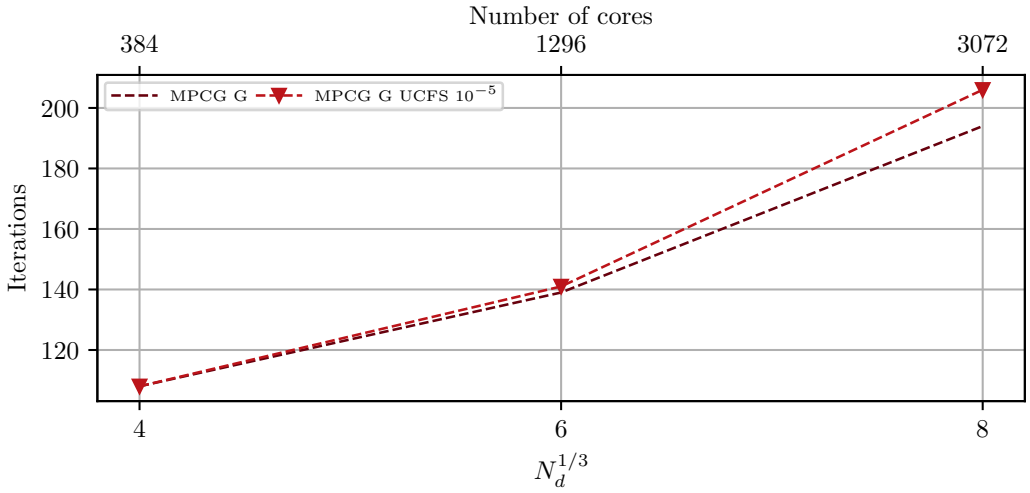


(c) Minimization space size.

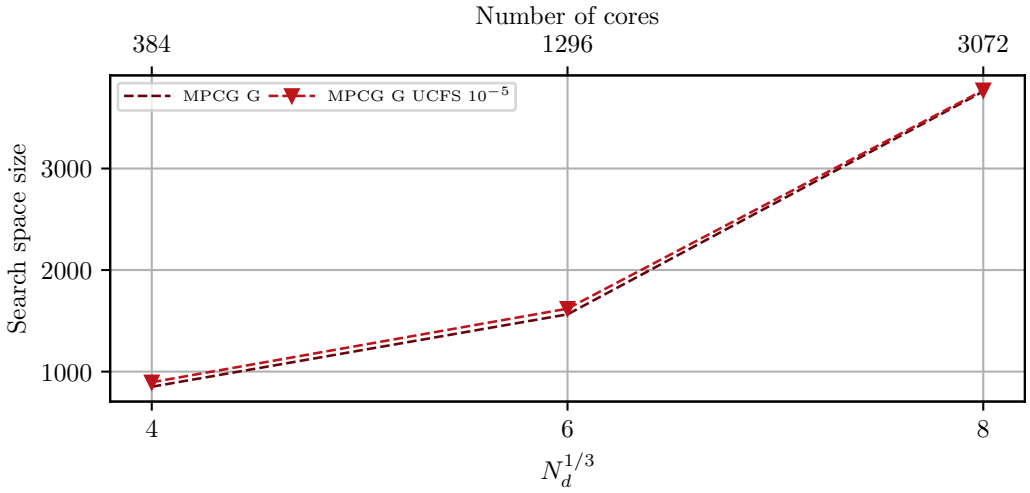
FIGURE 4 Checkerboard cube, weak parallel scalability (moderate heterogeneity $E_r/E_b = 10^2$): total time, number of iterations and minimization space size. Sator supercomputer.



(a) Total wall time.



(b) Number of iterations.



(c) Minimization space size.

FIGURE 5 Checkerboard cube, weak parallel scalability (high heterogeneity $E_r/E_b = 10^4$): total time, number of iterations and minimization space size. Sator supercomputer.

5.4 | Weak scalability study on the Topaze supercomputer

This section presents the scalability study carried out on the Topaze supercomputer. The main interest here is that the available memory per core is only 2 GB, which initially motivated the use of BLR compression. Also, the compute nodes use AMD processors and it is the first time that our implementation is benchmarked on such an architecture.

5.4.1 | Presentation of the Topaze supercomputer

The Topaze supercomputer is managed by the french Computing Center for Research and Technology (CCRT, http://www-ccrt.cea.fr). It is made of 864 nodes, 2.45GHz AMD Milan bi-socket with 64 cores per socket. With 864 compute nodes (111,592 cores) and a theoretical Peak performance of 4.34 PFlop/s, Topaze is ranked 238 in the TOP500 (list from Nov. 2023). One specificity of Topaze is that the RAM per core is only 2GB which motivates the use of compression techniques. Compute nodes are connected through a EDR InfiniBand network in a pruned Fat-tree topology. The communication are handled with OpenMPI 4.1.4.

5.4.2 | Weak scaling results

5.4.2.1 | Homogeneous probem

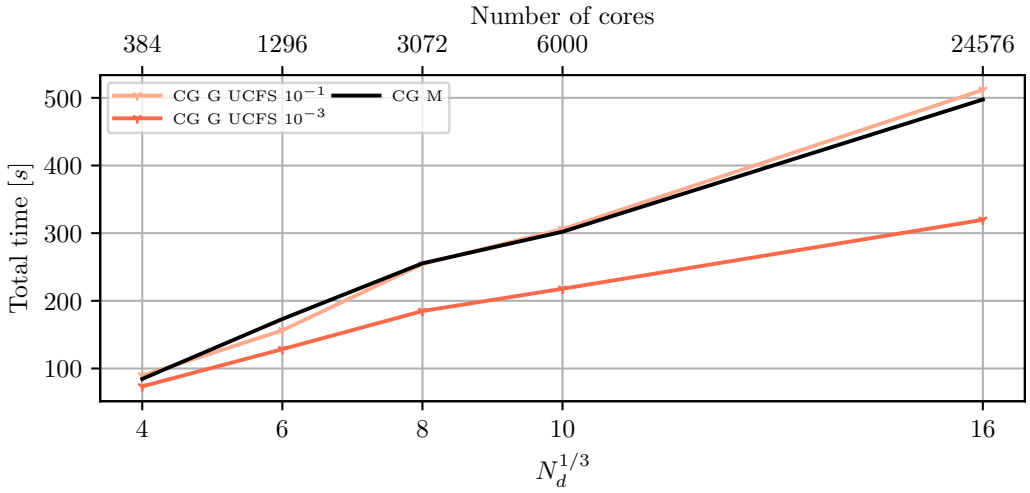
For the homogeneous test case and in order to reduce the number of simulations, only the CG solver is used with or without BLR compression. The weak scaling results are shown in Figure 6. Whatever the solver is, the number of iterations slightly increases with the size of the problem due to the automatic subdomain decomposition. Again, the configuration with a moderate compression provides the best performance, both in terms of iterations and time to solution. For the largest test case with 24,576 cores and 790.12M dofs, the time to solution is about 300s which represents a gain of about 40%. Also, the configuration with high compression provides the same time to solution than the uncompressed one, despite a greater number of iterations.

5.4.2.2 | Moderate heterogeneity

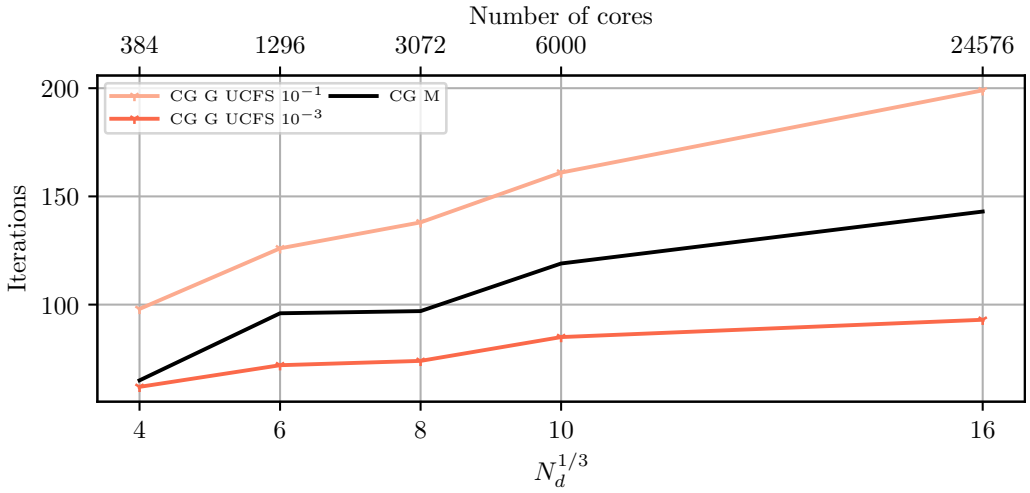
For the moderate heterogeneity test case, only three curves are considered: the CG solver with (G) nullspace without and with a low compression ratio, and the MPCG solver with a low compression ratio. The results are shown in Figure 7. Once again, the CG solver performs well with low BLR compression, the convergence rate is the same as without compression and the time to solution is reduced. Due to the larger search space, the MPCG solver with low BLR compression gives the best convergence rate. However, the cost of orthogonalising this search space tends to dominate the computation time for large problems ($\geq 6,000$ cores).

5.4.2.3 | High heterogeneity

The weak scaling results obtained with $E_r/E_b = 10^4$ are shown in Figure 8. Here only MPCG without compression is able to converge in less than 500 iterations for large problems. The test case with 24,756 cores ran out of memory. Multipreconditioning provides robustness at the cost of a large search space: the number of iterations is only doubled between 384 and 6,000 cores. For this type of problem, a restart of the MPCG solver should be implemented, in the same spirit as, for example, the GMREs-DR algorithm [26]. This is however out of the scope of the present study.

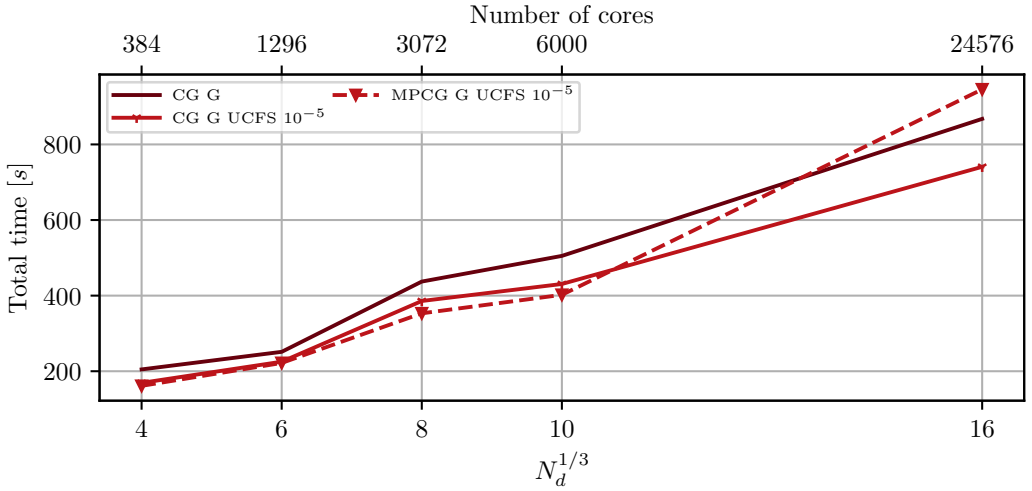


(a) Total wall time.

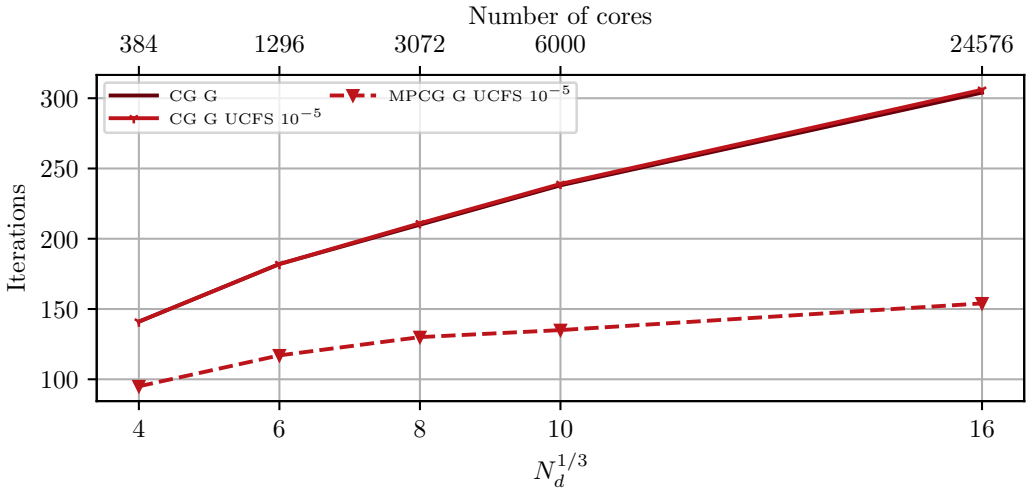


(b) Number of iterations.

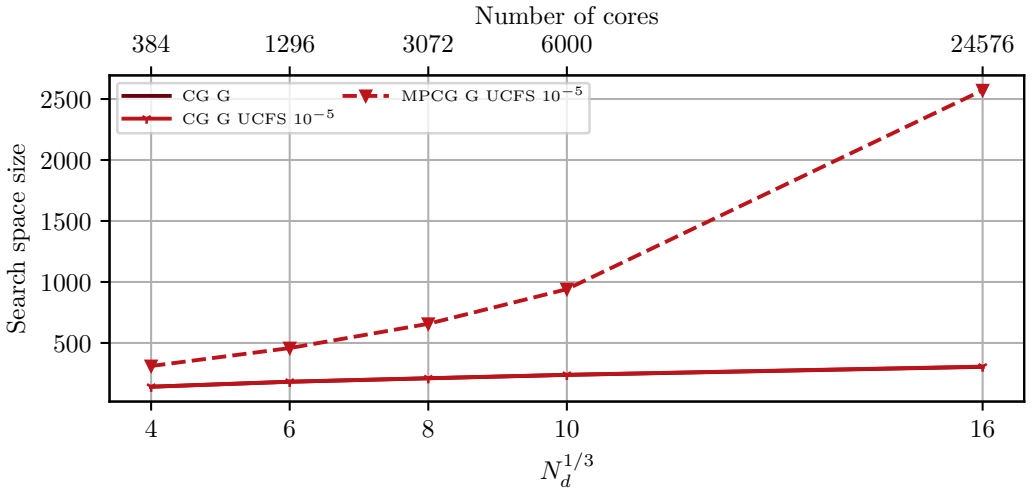
FIGURE 6 Checkerboard cube, weak parallel scalability (homogeneous case $E_r/E_b = 10^0$): wall time, number of iterations and minimization space size. Topaze supercomputer.



(a) Total wall time.

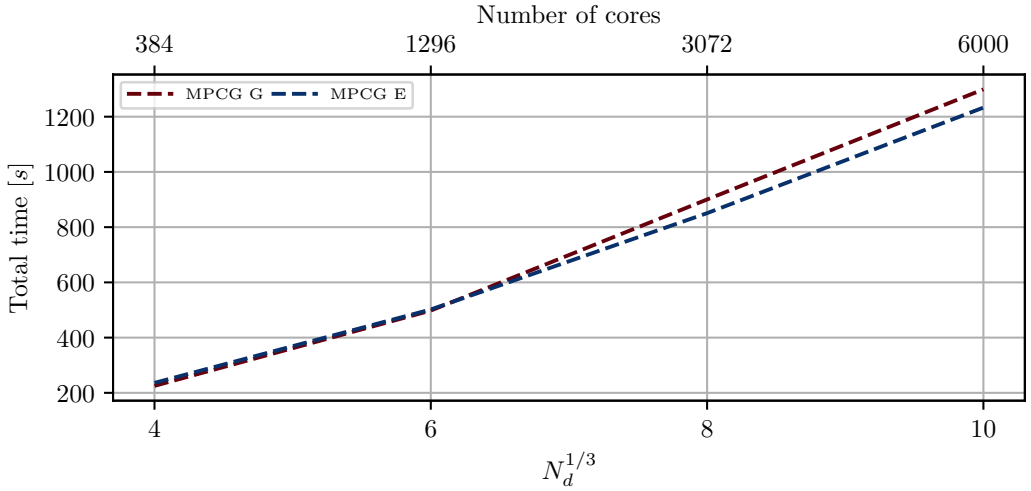


(b) Number of iterations.

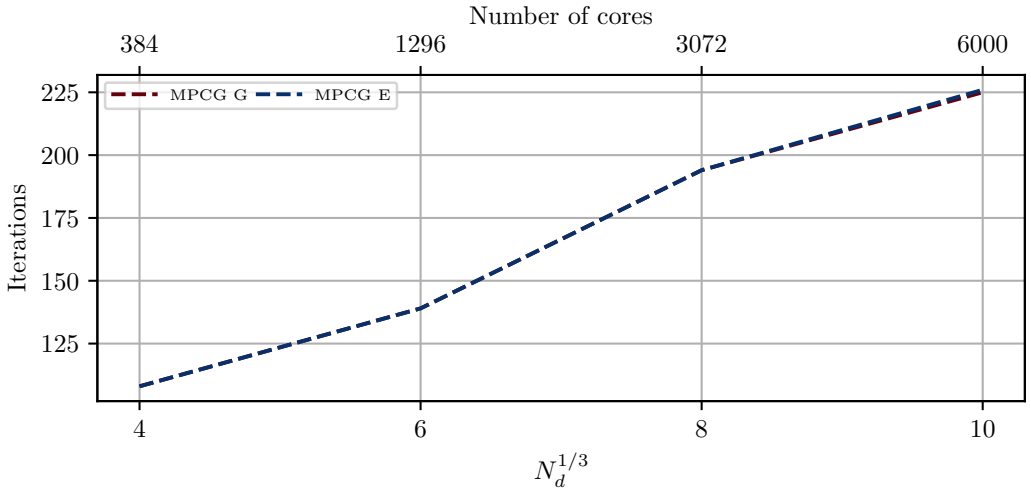


(c) Minimization space size.

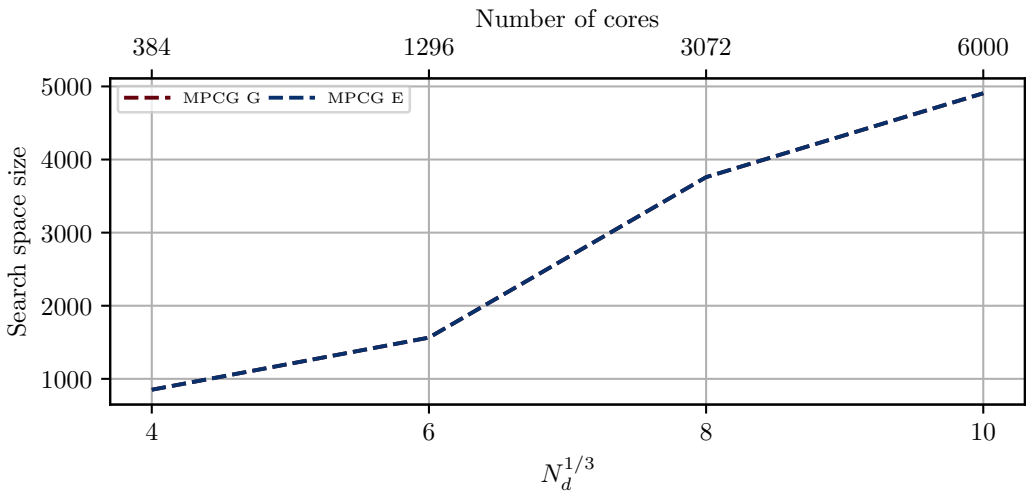
FIGURE 7 Checkerboard cube, weak parallel scalability (moderate heterogeneity $E_r/E_b = 10^2$): wall time, number of iterations and minimization space size. Topaze supercomputer.



(a) Total wall time.



(b) Number of iterations.



(c) Minimization space size.

FIGURE 8 Checkerboard cube, weak parallel scalability (high heterogeneity $E_r/E_b = 10^4$): wall time, number of iterations and minimization space size. Topaze supercomputer.

6 | CONCLUSION AND PERSPECTIVES

In order to adapt to modern supercomputer designs where the available memory per core is constantly decreasing, this paper proposes to use block low-rank factorization methods to equip primal domain decomposition methods with low memory footprint preconditioner. The BLR compression makes it difficult for the Mumps solver to detect the correct kernel to use. The nullspace is often not detected and the BDD method falls back to the Neumann-Neumann method: scalability is lost. Two alternative strategies have been tested: the hybrid geometric–algebraic approach and the low energy modes. The former makes the coarse problem independent of BLR compression, but requires the knowledge of the nullspace in the case of a completely floating subdomain. The latter is fully algebraic and takes compression into account, but numerical results suggest that the hybrid geometric–algebraic approach is preferable whenever available. Indeed, low energy modes seem to drift away from the original operator's nullspace for a high level of compression, which significantly degrades the convergence rate [12]. The BLR preconditioner has also been combined with adaptive multipreconditioning in order to increase the robustness of the solver with respect to material heterogeneity.

Weak scalability studies were presented using two supercomputers (Sator and Topaze) and three heterogeneity ratios. Numerical results show that BLR compression can improve both memory and solution time. It is especially interesting for reasonably well conditioned problems. For the largest homogeneous test case with 24,576 cores and 790.12M dofs, the time to solution is about 300s, which represents a 40% gain over the uncompressed preconditioner, while the memory footprint of the preconditioner is reduced by 20%.

The results also show that AMPBDD is robust with respect to material heterogeneity but generates a large search space. Unfortunately, multipreconditioning is unable to compensate for the loss of the correct coarse space in most situations. The largest ill-conditioned test case has approximately 200 million of unknowns and runs on 6,000 cores. Block low rank factorization is not sufficient here, and a GMRES-DR-style restart procedure will need to be investigated in the near future. However, this is the first large-scale evaluation of this solver. AMPBDD is particularly useful for simulating crack propagation problems because the nullspace computation only plays at the preconditioner level. One prospect of this work is the extension of AMPBDD phase field fracture [27] to larger scale problems solved on low memory supercomputers.

DATA AVAILABILITY STATEMENT

Data will be made available on request for the benchmarks presented in Section 5. The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. Farhat C, Roux FX. The dual Schur complement method with well-posed local Neumann problems. Contemporary Mathematics. 1994;157:193. doi: 10.1137/0914047

2. Gosselet P, Rixen D, Roux FX, Spillane N. Simultaneous FETI and block FETI: Robust domain decomposition with multiple search directions. International Journal for Numerical Methods in Engineering. 2015;104(10):905–927. nme.4946doi: 10.1002/nme.4946

3. Bovet C, Parret-Fréaud A, Spillane N, Gosselet P. Adaptive multipreconditioned FETI: Scalability results and robustness assessment. Computers & Structures. 2017:1–20. doi: 10.1016/j.compstruc.2017.07.010

4. Bovet C, Parret-Fréaud A, Gosselet P. Two-level adaptation for Adaptive Multipreconditioned FETI. Advances in Engineering Software. 2021;152:102952. doi: 10.1016/j.advengsoft.2020.102952

5. Farhat C, Lesoinne M, LeTallec P, Pierson K, Rixen D. FETI-DP: a Dual-Primal Unified FETI Method - Part I: a Faster Alternative to the Two-Level FETI Method. International Journal for Numerical Methods in Engineering. 2001;50(7):1523–1544. doi: 10.1002/nme.76

6. Mandel J. Balancing domain decomposition. Communications in Numerical Methods in Engineering. 1993;9(3):233. doi: 10.1002/cnm.1640090307

7. Dohrmann CR. A preconditionner for substructuring based on constrained energy minimization. SIAM Journal for Scientific Computing. 2003;25:246. doi: 10.1137/s1064827502412887

8. Bramble JH, Pasciak JE, Vassilev AT. Analysis of non-overlapping domain decomposition algorithms with inexact solves. Math. Comput.. 1998;67(221):119. doi: 10.1090/S0025-5718-98-00879-5

9. Börgers C. The Neumann-Dirichlet domain decomposition method with inexact solvers on the subdomains. Numerische Mathematik. 1989;55(2):123–136. doi: 10.1007/BF01406510

10. Haase G, Langer U, Meyer A. The approximate Dirichlet Domain Decomposition method. Part I: An algebraic approach. Computing. 1991;47(2):137–151. doi: 10.1007/BF02253431

11. Haase G, Langer U, Meyer A. The approximate Dirichlet Domain Decomposition method. Part II: Applications to 2nd-order Elliptic B.V.P.s. Computing. 1991;47(2):153–167. doi: 10.1007/BF02253432

12. Dohrmann CR. An approximate BDDC preconditioner. Numerical Linear Algebra with Applications. 2007;14(2):149–168. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nla.514doi: 10.1002/nla.514

13. Li J, Widlund OB. On the use of inexact subdomain solvers for BDDC algorithms. Computer Methods in Applied Mechanics and Engineering. 2007;196(8):1415–1428. doi: 10.1016/j.cma.2006.03.011

14. Liu JW. The multifrontal method for sparse matrix solution: Theory and practice. SIAM review. 1992;34(1):82–109.

15. Mary T. Block Low-Rank multifrontal solvers: complexity, performance, and scalability. PhD thesis. Université Paul Sabatier-Toulouse III, ; 2017.

16. Bebendorf M. Hierarchical matrices. Springer, 2008.

17. Spillane N. An Adaptive Multipreconditioned Conjugate Gradient Algorithm. SIAM J. Sci. Comput.. 2016;38(3):A1896–A1918. doi: 10.1137/15M1028534

18. Gosselet P, Rey C. Non-overlapping domain decomposition methods in structural mechanics. Archives of Computational Methods in Engineering. 2006;13(4):515–572.

19. Rixen DJ, Farhat C. A simple and efficient extension of a class of substructure based preconditioners to heterogeneous structural mechanics problems. International Journal for Numerical Methods in Engineering. 1999;44(4):489–516. doi: 10.1002/(SICI)1097-0207(19990210)44:4<489::AID-NME514>3.0.CO;2-Z

20. Bridson R, Greif C. A multipreconditioned conjugate gradient algorithm. SIAM J. Matrix Anal. Appl.. 2006;27(4):1056–1068 (electronic). doi: 10.1137/040620047

21. Leistner MC, Gosselet P, Rixen DJ. Recycling of Solution Spaces in Multi-Preconditioned FETI Methods Applied to Structural Dynamics. International Journal for Numerical Methods in Engineering. 2018. doi: 10.1002/nme.5918

22. Spillane N, Rixen DJ. Automatic spectral coarse spaces for robust FETI and BDD algorithms. International Journal for Numerical Methods in Engineering. 2013;95(11):953–990. doi: 10.1002/nme.4534

23. Bovet C. On the use of graph centralities to compute generalized inverse of singular finite element operators: Applications to the analysis of floating substructures. International Journal for Numerical Methods in Engineering. 2022;124(9):1933–1964. doi: 10.1002/nme.7193

24. Farhat C, Géradin M. On the general solution by a direct method of a large scale singular system of linear equations: application to the analysis of floating structures. International Journal for Numerical Methods in Engineering. 1998;41(4):675–696. doi: 10.1002/(SICI)1097-0207(19980228)41:4<675::AID-NME305>3.0.CO;2-8

25. Amestoy PR, Duff IS, Koster J, L'Excellent JY. A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling. SIAM Journal on Matrix Analysis and Applications. 2001;23(1):15–41. doi: 10.1137/S0895479899358194

26. Morgan RB. GMRES with Deflated Restarting. SIAM Journal on Scientific Computing. 2002;24(1):20–37. doi: 10/cdk9g4

27. Rannou J, Bovet C. Domain decomposition methods and acceleration techniques for the phase field fracture staggered solver. International Journal for Numerical Methods in Engineering. 2024:e7544. doi: 10.1002/nme.7544