Two-level adaptation for Adaptive Multipreconditioned FETI

Christophe Bovet, Augustin Parret-Fréaud, Pierre Gosselet

Two-level adaptation for Adaptive Multipreconditioned FETI

Christophe Bovet[1], Augustin Parret-Fréaud[2] and Pierre Gosselet[3,4]

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

[2]Safran Tech, Rue des Jeunes Bois, Châteaufort, CS 80112 78772 Magny-Les-Hameaux, France

[3]LMT, ENS Paris-Saclay / CNRS, 61 avenue du Président Wilson, 94235 Cachan, France

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

preprint version of doi: 10.1016/j.advengsoft.2020.102952

Abstract

This article introduces two strategies to reduce the memory cost of the Adaptive Multipreconditioned FETI method (AMPFETI) while preserving its capability to solve ill conditioned systems efficiently. Their common principle is to gather search directions into aggregates which are frequently adapted in order to achieve the best compromise between the decrease of the solver error and the computational resources employed. The methods are assessed on two weak scalability studies on highly heterogeneous problems up to 10368 cores and half a billion of unknowns, and on two ill-conditioned industrial applications, related to the numerical homogenization of solid propellant and to the simulation of a multiperforated aircraft combustion chamber.

Keywords: Domain decomposition, FETI, Krylov subspace methods, multipreconditioning, adaptive multipreconditioning, AMPFETI, SFETI

See [6] for the updated citation.

1 Introduction

The continuous increase of high performance computing resources naturally appeals the engineers to take into account more complex physical phenomena and to simulate fine scale models. In the area of solid mechanics, for instance, this can be exemplified by the inclusion of small geometric details (microperforations, convoluted blade cooling channels...), or complex nonlinear heterogeneous materials (woven composite, crystal plasticity...). After the discretization of the involved partial differential equations, this results in large sparse linear(ized) systems which range from millions to billions of unknowns, and which are very often ill-conditioned.

Non overlapping domain decomposition methods, such as the Balancing Domain Decomposition (BDD) [20] or the Finite Element Tearing and Interconnecting (FETI) [13] and their constrained variants FETI-DP [10] and BDDC [8], provide a favorable framework to solve these linear systems in parallel. Recently developed domain decomposition methods, such as the ML-FETI-DP [31] or the HTFETI [22], are able to tackle really large-size problems while exhibiting a really good scalability. Indeed, these methods are well adapted to supercomputers since they rely on both iterative and direct solvers. Usually each subdomain is assigned to one computer node (or one socket) where a multithreaded direct solver is used. An interface problem is solved between subdomains with a Krylov solver (MPI parallelism) to obtain the global solution.

Thanks to this combination of direct and iterative solvers, domain decomposition methods are usually more robust than traditional iterative solvers as they ensure a logarithmic bound in H/h on the condition number, where h is the characteristic size of the finite element and H that of a subdomain [32]. Despite that, real engineering applications frequently exhibit pathological components that hinder the convergence of the underlying Krylov solver, such as jagged interfaces, bad aspect ratios, strong heterogeneities misplaced with respect to the interface, incompressibility etc.

Two kinds of strategy have been proposed to increase the robustness of domain decomposition solvers. The first strategy is to detect *a priori* the local contributions which are known to penalize the convergence, and to remove them by augmenting the Krylov solver. The problematic local information can be identified through local generalized eigenvalue problems [30, 28, 21] or reused from a previous solution [9, 12, 25, 14, 19]. The second strategy is to use a block Krylov solver [24, 15] or a multipreconditioned Krylov solver [7, 29, 4]. Multipreconditioning is well adapted to the FETI and BDD methods because of the additive nature of their classical preconditioners. Moreover, this technique has been successfully applied to solve non symmetric systems [16, 3] for which finding the suitable augmentation is still an open question.

The fast growing search space produced by a multipreconditioned solver is the key ingredient to ensure a fast convergence. In the original Simultaneous FETI method (SFETI, which corresponds to multipreconditioned conjugate gradient applied to FETI) [15], the algorithm generates at each iteration as many search directions as there are subdomains in the decomposition. However, these extra search directions need to be stored in memory which can become problematic in case of a large number of subdomains. To reduce the memory requirement of SFETI, a sparse storage format has recently been proposed in [23]. An alternative trail is followed in Adaptive Multipreconditioned FETI (AMPFETI) [4, 3, 5]: among the search directions generated by the subdomains, only the ones that are identified as useful for the convergence are kept individually, the others are reduced by summation as done in classical FETI [29]. Thanks to this test, the size of the generated search space is significantly reduced compared to SFETI. In [4], the AMPFETI's capability to solve ill-conditioned systems has been assessed up to 500 subdomains and 100 millions of unknowns. Note that for now, multipreconditioned solvers have been tested with FETI and BDD domain decomposition methods, but they could also be employed with the FETI-DP and BDDC methods.

The aim of this paper is to present two improvements of AMPFETI allowing addressing larger ill-conditioned problems. First, a better control of the memory requirement is achieved by aggregating search direction candidates before multipreconditioning. The paper provides guidelines for the construction of the aggregates. Second, in order to accelerate the convergence even more, aggregates are updated at each iteration in order to generate better search directions.

The article is organized as follows: Section 2 gives a short presentation of Adaptive Multipreconditioned FETI; then Section 3 details the methodology proposed to gather the search directions into adaptive aggregates. Scalability results on academic benchmarks are provided in Section 4 and Section 5 presents two realistic industrial simulations. Section 6 concludes the paper.

2 Adaptive multipreconditioned FETI: a short reminder

Adaptive multipreconditioned FETI is the combination of the FETI domain decomposition method [13, 11] and the adaptive multipreconditioned conjugate gradient solver [29, 7].

2.1 The FETI linear system

Let us consider the linear system of equation $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$ arising from the finite element discretization of a linear mechanical problem defined on domain Ω. Let $\boldsymbol{u}$ denote the vector of generalized displacement and $\boldsymbol{f}$ the vector of generalized forces. In the present study, we assume that the stiffness matrix $\boldsymbol{K}$ is symmetric positive definite. Let $(\Omega^s)_{1\leqslant s\leqslant N_d}$ be a non overlapping partition of Ω such that: $\forall s \neq p,\ \Omega^s \bigcap \Omega^p = \emptyset$ and $\bar{\Omega} = \bigcup_{s=1}^{N_d} \bar{\Omega}^s$ where $\bar{\Omega}^s$ is the closure of Ω^s.

The interface between two subdomains s and p is denoted by $\Upsilon^{sp} = \bar{\Omega}^s \bigcap \bar{\Omega}^p$ and the union of all interfaces of the subdomain s is denoted by Υ^s. The set gathering all interfaces is denoted by Υ.

In the FETI method, a Lagrange multiplier field $\boldsymbol{\lambda}$ that connects subdomains is introduced and the global linear system $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$ is substituted by an equivalent substructured formulation

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

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

where $\boldsymbol{T}^s : \Omega^s \to \Upsilon^s$ are trace operators and $\boldsymbol{B}^s$ are signed Boolean assembly operators on the global interface. From a mechanical point of view, equations (1) express the equilibrium of all subdomains and (2) corresponds to the continuity of the displacement across interfaces. Then, the local equilibrium is expressed in terms of interface unknown. Schur complements ($\boldsymbol{S}^s$) naturally arise and, after some algebraic manipulations, the classical FETI system is formed:

$$\begin{pmatrix} \boldsymbol{F} & \boldsymbol{G} \\ \boldsymbol{G}^\top & \boldsymbol{0} \end{pmatrix} \begin{pmatrix} \boldsymbol{\lambda} \\ \boldsymbol{\beta} \end{pmatrix} = \begin{pmatrix} \boldsymbol{d} \\ \boldsymbol{e} \end{pmatrix} \tag{3}$$

where

$$\boldsymbol{e} = -\left({\boldsymbol{f}^1}^\top \boldsymbol{R}^1 | \dots | {\boldsymbol{f}^{N_d}}^\top \boldsymbol{R}^{N_d} \right)^\top \quad ; \quad \boldsymbol{d} = -\sum_{s=1}^{N_d} \boldsymbol{B}^s\boldsymbol{T}^s{\boldsymbol{K}^s}^\dagger\boldsymbol{f}^s$$

$$\boldsymbol{F} = \sum_{s=1}^{N_d} \boldsymbol{B}^s{\boldsymbol{S}^s}^\dagger{\boldsymbol{B}^s}^\top \quad ; \quad \boldsymbol{S}^s = \boldsymbol{K}^s_{bb} - \boldsymbol{K}^s_{bi}{\boldsymbol{K}^s_{ii}}^{-1}\boldsymbol{K}^s_{ib}$$

$$\boldsymbol{G} = \left(\boldsymbol{B}^1\boldsymbol{T}^1\boldsymbol{R}^1 | \ldots | \boldsymbol{B}^{N_d}\boldsymbol{T}^{N_d}\boldsymbol{R}^{N_d}\right)$$

The Schur complement $\boldsymbol{S}^s$ is computed by condensing out the internal degrees of freedom (index i) on the boundary (index b). $\boldsymbol{S}^{s\dagger}$ denotes the generalized inverse of $\boldsymbol{S}^s$ in case the kernel $\boldsymbol{R}^s$ of $\boldsymbol{K}^s$ is not restricted to zero. The second row, $\boldsymbol{G}^\top\boldsymbol{\lambda} = \boldsymbol{e}$, is associated with the constraint that each subdomain must remain self-equilibrated. The vector $\boldsymbol{\beta}$ contains the rigid body mode coefficients of each subdomains. This vector is computed after $\boldsymbol{\lambda}$ has been found (see [13]).

Please note that the FETI operator $\boldsymbol{F}$ is never assembled since only the result of a multiplication by $\boldsymbol{F}$ is needed. Instead of solving a saddle point system, an initialization–projection strategy is applied and $\boldsymbol{\lambda}$ is sought as:

$$\boldsymbol{\lambda} = \boldsymbol{\lambda}_0 + \boldsymbol{\Pi}\tilde{\boldsymbol{\lambda}} \quad ; \quad \boldsymbol{G}^\top\boldsymbol{\lambda}_0 = \boldsymbol{e} \quad ; \quad \boldsymbol{G}^\top\boldsymbol{\Pi} = \boldsymbol{0}$$

The projector $\boldsymbol{\Pi}$ is of the form $\boldsymbol{\Pi} = \boldsymbol{I} - \boldsymbol{A}\boldsymbol{G}(\boldsymbol{G}^\top\boldsymbol{A}\boldsymbol{G})^{-1}\boldsymbol{G}^\top$, where $\boldsymbol{A}$ is a symmetric positive definite matrix most often chosen among the classical preconditioners $\tilde{\boldsymbol{S}}$ and the identity matrix.

Substituting this form into (3), and pre-multiplying by $\boldsymbol{\Pi}^\top$ leads to the final linear system:

$$\boldsymbol{\Pi}^\top\boldsymbol{F}\boldsymbol{\Pi}\tilde{\boldsymbol{\lambda}} = \boldsymbol{\Pi}^\top(\boldsymbol{d} - \boldsymbol{F}\boldsymbol{\lambda}_0) \tag{4}$$

In the classical FETI method, the system (4), is solved with a preconditioned conjugate gradient. The usual preconditioner is a scaled sum of (approximation of) Schur complements:

$$\tilde{\boldsymbol{S}} = \sum_{s=1}^{N_d} \tilde{\boldsymbol{B}}^s\check{\boldsymbol{S}}^s\tilde{\boldsymbol{B}}^{s\top} \tag{5}$$

where $(\tilde{\boldsymbol{B}}^s)$ are scaled assembly operators such that:

$$\left(\sum_{s=1}^{N_d} \boldsymbol{B}^s\tilde{\boldsymbol{B}}^{s\top}\right)\boldsymbol{B}^j = \boldsymbol{B}^j, \quad \forall\ 1 \leqslant j \leqslant N_d$$

Local approximations $\check{\boldsymbol{S}}^s$ of $\boldsymbol{S}^s$ can be chosen among $\boldsymbol{S}^s$ (Dirichlet preconditioner), $\boldsymbol{K}^s_{bb}$ (Lumped preconditioner) and $\mathrm{diag}(\boldsymbol{K}^s_{bb})$ (Superlumped preconditioner). In the following, we will single out the subdomains' contribution to the operator and to the preconditioner, and we will use the notation:

$$\boldsymbol{F}^s = \boldsymbol{B}^s\boldsymbol{S}^{s\dagger}\boldsymbol{B}^{s\top} \qquad ; \qquad \tilde{\boldsymbol{S}}^s = \tilde{\boldsymbol{B}}^s\check{\boldsymbol{S}}^s\tilde{\boldsymbol{B}}^{s\top}$$

2.2 Multipreconditioned FETI

There are many cases in which the preconditioned conjugate gradient used in the FETI algorithm converges very slowly or does not converge at all, even with state-of-the-art choices of the preconditioner. A typical pathological case is strong material heterogeneity misplaced with respect to the interface between subdomains [27]. Such a situation cannot be avoided for the simulation of solid propellant (see Section 5.1).

Recently, multipreconditioned strategies [26, 15, 5, 4] were proposed to solve these hard problems. In these methods, the contribution of each subdomain to (5) is considered as a separate preconditioner which, at each iteration, supplies one direction inside a search space of dimension N_d. All these strategies rely on multipreconditioned Krylov solvers [16, 3] and more specifically multipreconditioned conjugate gradient (MPCG) [7].

If $\boldsymbol{r}_{i+1}$ denotes the residual at iteration $i+1$, the classical preconditioned conjugate gradient generates a search space of dimension 1 materialized by the preconditioned residual $\boldsymbol{z}_{i+1}$, multipreconditioned conjugate gradient generates a larger search space, spanned by N_d vectors gathered in the matrix $\boldsymbol{Z}_{i+1}$:

$$\boldsymbol{z}_{i+1} = \sum_{s=1}^{N_d} \tilde{\boldsymbol{S}}^s\boldsymbol{r}_{i+1} \qquad ; \qquad \boldsymbol{Z}_{i+1} = \left[\tilde{\boldsymbol{S}}^1\boldsymbol{r}_{i+1}, \ldots, \tilde{\boldsymbol{S}}^{N_d}\boldsymbol{r}_{i+1}\right] \tag{6}$$

The optimality conditions of conjugate gradient enables to compute the best combinations of the columns of $\boldsymbol{Z}_{i+1}$ at each iteration, whereas classical preconditioned conjugate gradient corresponds to the fixed choice $\boldsymbol{z}_{i+1} = \boldsymbol{Z}_{i+1}\boldsymbol{1}$ (with $\boldsymbol{1} = \left(1, \ldots, 1\right)^\top$).

Algorithm 1 presents multipreconditioned FETI combined with the adaptive τ-test and an aggregation procedure described in Section 3. To recover the original multipreconditioned FETI, forget the τ-test of line 12, consider $C_k = \{\Omega^k\}\ \forall\ 1 \leqslant k \leqslant N_a = N_d$, and use the definition (6) of $\boldsymbol{Z}_{i+1}$ at line 13.

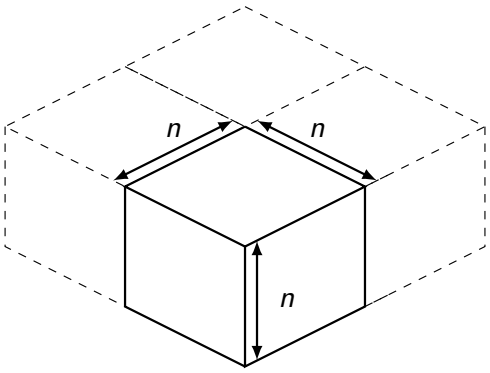


Figure 1: Cubic mesh with regular decomposition

In [19] it was shown that multipreconditioning permits to explore the subspace spanned by the "bad eigenvectors" detected by the Generalized Eigenvalues in the Overlaps (GenEO) algorithm, and thus it leads to a more stable convergence on ill-conditioned problems. But multipreconditioning comes with a high price. Indeed, the original multipreconditioned FETI generates at each iteration, two matrices $\boldsymbol{W}_i$ and $\boldsymbol{Q}_i$ (see Algorithm 1). Since, a full reorthogonalization is mandatory, all these matrices need to be stored.

To set the ideas, let us consider a regular cubic mesh with a cubic domain decomposition, and let n denote the number of nodes per subdomain edge (see Figure 1 and Figure 3). The memory footprint of all the elements of the sequences $(\boldsymbol{W}_i)_i$ and $(\boldsymbol{Q}_i)_i$ is $O(n^2 N_d n_{iter})$ and for $(\Delta_i)_i$ it is $O(N_d^2 n_{iter})$. This may become prohibitive when the number of subdomains N_d scales with the same rate as n does.

As mentioned in the introduction, one possibility to reduce the memory requirement of MPFETI is to express all operations in terms of $(\boldsymbol{Z}_i)_i$ instead of $(\boldsymbol{W}_i, \boldsymbol{Q}_i)_i$. Indeed, contrarily to $(\boldsymbol{W}_i, \boldsymbol{Q}_i)_i$, $(\boldsymbol{Z}_i)_i$ is sparse (the contribution of one subdomain only affects its neighbors). $(\boldsymbol{Z}_i)_i$ is thus much less expensive to store. This strategy, described in [23], comes with many algebraic manipulations.

Algorithm 1: AMPFETI with local τ-test and aggregation

Input: Convergence threshold $\epsilon > 0$, adaptation threshold $\tau > 0$, initial partition $\mathcal{P}_{N_a}$

$\boldsymbol{\Pi} = \boldsymbol{I} - \boldsymbol{AG}(\boldsymbol{G}^\top \boldsymbol{AG})^{-1}\boldsymbol{G}^\top$

$\tilde{\boldsymbol{\lambda}}_0 = \boldsymbol{AG}(\boldsymbol{G}^\top \boldsymbol{AG})^{-1}\boldsymbol{e}$

$\boldsymbol{r}_0 = \boldsymbol{\Pi}^\top(\boldsymbol{d} - \boldsymbol{F}\tilde{\boldsymbol{\lambda}}_0)$

$\boldsymbol{Z}_0 = \left(\sum_{\Omega^s \in C_1} \tilde{\boldsymbol{S}}^s \boldsymbol{r}_0 | \dots | \sum_{\Omega^s \in C_{N_a}} \tilde{\boldsymbol{S}}^s \boldsymbol{r}_0\right)$

$\boldsymbol{W}_0 = \boldsymbol{\Pi}\boldsymbol{Z}_0$

$\boldsymbol{\lambda}_0 = \boldsymbol{0}$, $i = 0$

while $\|\boldsymbol{r}_i\| > \epsilon$ **do**

  $\boldsymbol{Q}_i = \boldsymbol{F}\boldsymbol{W}_i$

  $\boldsymbol{\Delta}_i = \boldsymbol{Q}_i^\top \boldsymbol{W}_i \; ; \boldsymbol{\gamma}_i = \boldsymbol{Z}_i^\top \boldsymbol{r}_i \; ; \boldsymbol{\alpha}_i = \boldsymbol{\Delta}_i^\dagger \boldsymbol{\gamma}_i$

  $\boldsymbol{\lambda}_{i+1} = \boldsymbol{\lambda}_i + \boldsymbol{W}_i \boldsymbol{\alpha}_i$

  $\boldsymbol{r}_{i+1} = \boldsymbol{r}_i - \boldsymbol{\Pi}^\top \boldsymbol{Q}_i \boldsymbol{\alpha}_i$

  $t_i^C = \dfrac{(\boldsymbol{W}_i\boldsymbol{\alpha}_i)^\top \sum_{\Omega^s \in C} \boldsymbol{F}^s (\boldsymbol{W}_i \boldsymbol{\alpha}_i)}{\boldsymbol{r}_{i+1}^\top \sum_{\Omega^s \in C} \tilde{\boldsymbol{S}}^s \boldsymbol{r}_{i+1}}$

  $\boldsymbol{Z}_{i+1} = \text{concatenate}(\{\sum_{\Omega^s \in C} \tilde{\boldsymbol{S}}^s \boldsymbol{r}_{i+1} \, ; \, t_i^C < \tau\}, \sum_{t_i^C > \tau} \sum_{\Omega^s \in C} \tilde{\boldsymbol{S}}^s \boldsymbol{r}_{i+1})$

  $\boldsymbol{W}_{i+1} = \boldsymbol{\Pi}\boldsymbol{Z}_{i+1}$ **for** $0 \le j \le i$ **do** $\begin{cases} \boldsymbol{\Phi}_{i,j} = \boldsymbol{Q}_j^\top \boldsymbol{W}_{i+1} \\ \boldsymbol{W}_{i+1} \leftarrow \boldsymbol{W}_{i+1} - \boldsymbol{W}_j \boldsymbol{\Delta}_j^\dagger \boldsymbol{\Phi}_{i,j} \end{cases}$

return $\tilde{\boldsymbol{\lambda}}_0 + \boldsymbol{\Pi}\boldsymbol{\lambda}_i$

2.3 Adaptive multipreconditioned FETI

A major improvement of MPCG is the addition of an adaptive process [29] which allows singling out useful search directions. Before being inserted in the search subspace, a criterion enables to forecast the effectiveness of the search directions provided by the subdomains with regard to the convergence. Depending on a user parameter τ which can be linked with a target relative reduction of the $\boldsymbol{F}$-norm of the error, the adaptive criterion chooses between 1 and N_d search directions. In [29], two variants of the adaptive process were proposed:

- A global adaptive test (called Global τ-test) in which either all search directions are kept (leading to a full MPCG iteration), or they are all reduced to one direction (leading to a classical PCG).
- A local adaptive test (called Local τ-test) in which each search direction is considered individually. This variant allows a much finer selection of search direction candidates. But the local test is significantly more expensive to compute than the global test.

The adaptive algorithms [4] start from the full subspace generated by multipreconditioning (6), but now the columns are only considered as potential search directions (hence the superscript p):

$$\boldsymbol{Z}_{i+1}^{p} = \left[\tilde{\boldsymbol{S}}^{1}\boldsymbol{r}_{i+1}, \ldots, \tilde{\boldsymbol{S}}^{N_d}\boldsymbol{r}_{i+1}\right] \tag{7}$$

With the Global τ-test, the effect of multipreconditioning can be forecast by the quantity (with Algorithm 1 notations):

$$t_i^G = \frac{\boldsymbol{\gamma}_i^\top \boldsymbol{\alpha}_i}{\boldsymbol{r}_{i+1}^\top \boldsymbol{z}_{i+1}} \tag{8}$$

which is almost costless to compute. If t_i^G is less than a certain criterion τ then it is worth multipreconditioning using $\boldsymbol{Z}_{i+1}^p$ as search space, else the classical FETI approach with one dimensional search space $\boldsymbol{z}_{i+1}$ already performs satisfyingly.

With the Local τ-test, the contribution to the decrease of the error of each vector resulting from the local preconditioning $\tilde{\boldsymbol{S}}^s\boldsymbol{r}_{i+1}$ can be estimated individually. This estimate is given by the scalar t_i^s:

$$t_i^s = \frac{(\boldsymbol{W}_i\boldsymbol{\alpha}_i)^\top \boldsymbol{F}^s(\boldsymbol{W}_i\boldsymbol{\alpha}_i)}{\boldsymbol{r}_{i+1}^\top \tilde{\boldsymbol{S}}^s\boldsymbol{r}_{i+1}} \tag{9}$$

where all quantities defined at iteration i are known.

Note that computing the local test is significantly more expensive than the global one. If t_i^s is less than a certain criterion τ, then the vector is worth being isolated, else it can be harmlessly combined with other little-contributing vectors. Let I_i be the set of subdomains (Ω^s) such that $t_i^s < \tau$, the actual search subspace then can be written as:

$$\boldsymbol{Z}_{i+1} = \left[\left[\ldots, \tilde{\boldsymbol{S}}^s\boldsymbol{r}_{i+1}, \ldots\right]_{s\in I_i}, \sum_{s\notin I_i} \tilde{\boldsymbol{S}}^s\boldsymbol{r}_{i+1}\right] \tag{10}$$

We refer to [29] and [4] to get more details about the adaptive tests.

Algorithm 1 presents Local AMPFETI combined with an aggregation procedure which will be described in Section 3. To recover the original Local AMPFETI, just consider $C_k = \{\Omega^k\}\ \forall\ 1 \leqslant k \leqslant N_a = N_d$.

It is worth noting that unfortunately, AMPFETI cannot be combined with the sparse storage strategy of [23]. Indeed, for a given iteration j, all the columns (before selection) of the previous preconditioned residual $(\ldots, \tilde{\boldsymbol{S}}^s\boldsymbol{r}_i, \ldots)_{s;i<j}$ need to be stored in order to rebuild the selected search directions, even if they are scarce.

3 Two-level adaptive strategy

3.1 Gathering search direction into aggregates

Even with a very stringent test, the memory requirement of the original AMPFETI may become high when the system to be solved is ill-conditioned and the number of subdomains is large.

As a complement to the selection threshold τ, we propose to choose *a priori* the granularity of the multipreconditioning by gathering search directions into aggregates. Indeed, the MPCG framework is not limited to starting from individual contributions from subdomains as in (7).

Let $\mathcal{P}_{N_a} = (C_k)_{1\leq k\leq N_a}$ be a partition of the subdomains' connectivity graph into $N_a \leqslant N_d$ aggregates. We can then propose N_a potential search directions under the form:

$$\boldsymbol{Z}_{i+1}^{p} = \left(\sum_{\Omega^s\in C_1} \tilde{\boldsymbol{S}}^s\boldsymbol{r}_{i+1}, \ldots, \sum_{\Omega^s\in C_{N_a}} \tilde{\boldsymbol{S}}^s\boldsymbol{r}_{i+1}\right) \tag{11}$$

Then, the adaptive τ-test can be applied on each aggregate's contribution, as described in Algorithm 1. In the end, at most N_a vectors are retained and used by the solver, then the memory footprint for the storage of $(\boldsymbol{W}_i)_i$ and $(\boldsymbol{Q}_i)_i$ can not exceed $O(n^2 N_a n_{iter})$ and $O(N_a^2 n_{iter})$ for $(\Delta_i)_i$.

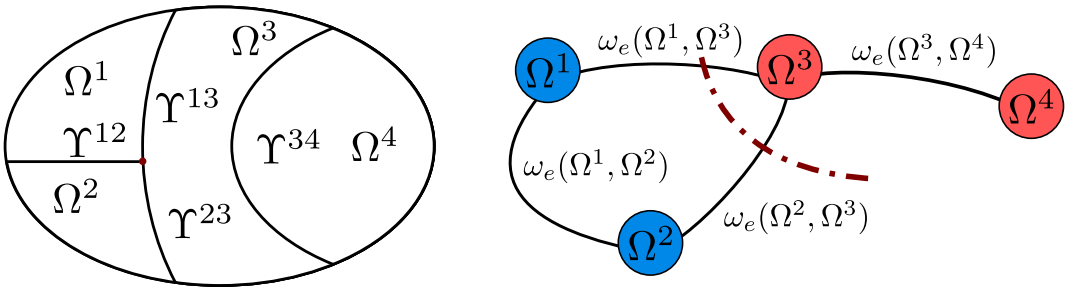


Figure 2: Illustration of a subdomain's connectivity graph. The graph is decomposed in two colors (blue and red). The dashed line represents the separation. The cut of the partition is $\omega_e(\Omega^1,\Omega^3)+\omega_e(\Omega^2,\Omega^3)$.

An interesting point is that the one-aggregate case ($N_a = 1$) corresponds to the genuine FETI method equipped with a classical conjugate gradient.

In practice, the partition $(C_k)_{1\leq k\leq N_a}$ is provided by a graph partitioning library such as Metis [18]. More precisely, let $G = \left(\left\{\Omega^1,\ldots,\Omega^{N_d}\right\},\Upsilon\right)$ be the subdomain's connectivity graph. Subdomains (Ω^s) are the graph vertexes. There is an edge between two subdomains (Ω^s,Ω^p) if they share one interface Υ^{sp}.

The choice of optimal weights is hard to now *a priori*. In this Section, uniform weights are applied to vertexes and edges. In Section 3.3, weights coming from the value of the local τ-test will be employed. Graph partitioning algorithms aim to provide a partition that minimizes both the unbalance and the cut of the partition. In our case, the unbalance is not critical, whereas minimizing the cut is essential. The cut of the partition is given by:

$$\text{cut}\left(\mathcal{P}_{N_a}\right) = \sum_{k<p}\text{cut}\left(C_k,C_p\right) = \sum_{k<p}\sum_{\substack{\Omega^r\in C_k\\ \Omega^s\in C_p}}\omega_e\left(\Omega^r,\Omega^s\right) \tag{12}$$

where $\omega_e(\Omega^r,\Omega^s)$ is the weight of the edge between Ω^r and Ω^s.

The cut is directly linked to the spread of the preconditioned residual $\boldsymbol{Z}_i$, which controls the performance of the algorithm (see Section 2.3 in [4]) since it drives the cost of the computation of $\boldsymbol{FW}_i$ which is operated as:

$$\boldsymbol{FW}_i = \boldsymbol{FZ}_i - (\boldsymbol{FAG})(\boldsymbol{G}^\top\boldsymbol{AG})^{-1}\boldsymbol{G}^\top\boldsymbol{Z}_i - \sum_{j=0}^{i-1}\boldsymbol{Q}_j\Delta_j^\dagger\Phi_{i,j}$$

The matrix $(\boldsymbol{FAG})$ is computed during the initialization and the last critical point is the computation of:

$$\boldsymbol{FZ}_i = \sum_{s=1}^{N_d}\boldsymbol{F}^s\boldsymbol{Z}_i$$

Computing all $\boldsymbol{F}^s\boldsymbol{Z}_i$ means parallel resolution of local problems (with direct solvers) with multiple right-hand sides. The matrix $\boldsymbol{Z}_i$ is sparse and the cut of the partition directly defines the number of right-hand sides. Indeed, due to the aggregation, $\boldsymbol{F}^s$ needs only to be applied on the columns of $\boldsymbol{Z}_i$ which correspond to aggregates containing either Ω^s or one of its neighbors Ω^s. That is why, the cut of the partition has to be minimized.

Aggregation is thus a simple tool to control the cost of the multipreconditioning. The remaining question is, for a given number of aggregates N_a, how to choose the partition that leads to the fastest decrease of the error while preserving a low cut. Giving a general answer to this question seems to be out of reach, anyhow it is possible to derive efficient heuristics.

In the simple case of a soft matrix with localized stiff inclusions, it has been observed that the subdomains in the vicinity of the heterogeneity contain more numerical information [4]. This information was properly caught by the local adaptive test since the search direction proposed by these subdomains were more often selected.

Starting from this observation, two different strategies to adapt the aggregates over the iterations are presented in the following. Both are based on the value of the local adaptive test. In Section 3.2, Section 3.3 and in all numerical tests, the number of aggregates N_a is fixed over the iterations. This is not a limitation of the method, but a technical choice since a more efficient memory reuse is obtained in the code by keeping N_a constant.

3.2 Discrete aggregate adaptation

At each iteration, the N_a aggregates are separated into three disjoint groups depending on the value of the adaptive test.

$$\mathcal{P}_{N_a} = \mathcal{S}_c \bigsqcup \mathcal{J}_c \bigsqcup \mathcal{U}$$

The set $\mathcal{S}_c := \left\{C \in \mathcal{P}_{N_a} \mid t_i^C \ll \tau\right\}$ contains aggregates that are candidates to be split since, according to the value of the local τ-test, they contain valuable information to achieve convergence. Conversely, the set $\mathcal{J}_c := \left\{C \in \mathcal{P}_{N_a} \mid t_i^C \gg \tau\right\}$ gathers the aggregates that are candidate to be joined. Aggregates that are not in the two previous sets remain unchanged $\mathcal{U} := \mathcal{P}_{N_a} \setminus (\mathcal{J}_c \cup \mathcal{S}_c)$. The "much greater" and "much less" relations are defined by two heuristics linked with the cardinal of the aggregate. The proposition $t_i^C \ll \tau$ is considered to be true if $t_i^C \operatorname{card} C \leq \tau$. Analogously, the proposition $t_i^C \gg \tau$ is considered to be true if $t_i^C / \operatorname{card} C \geq \tau$. Once the aggregates have been classified, various strategies can be used to actually perform the merger and the split.

Join strategies Two strategies are described in Algorithm 2 and 3. The former simply proposes to join aggregates two-by-two if they are connected, i.e. if they contain, at least, two neighboring subdomains. This strategy makes aggregates smoothly evolve and it preserves their connectivity. The latter joins all the connected components of the subgraph associated with the aggregates to be joined. It induces a more massive adaptation while still ensuring the connectivity.

For both algorithms, aggregates are sorted with respect to the value of $(t_i^C / \operatorname{card}(C))_C$ to favor the largest values of the τ-test : aggregates associated with large $(t_i^C / \operatorname{card}(C))$ are joined first.

After the joining step, the number of aggregates has decreased and some free space is available for the split step.

Algorithm 2: Join strategy based on a two-by-two merging.

Input : Set of aggregates selected for join $\mathcal{J}_c^i$, criteria $(t_i^C / \operatorname{card}(C))_C$
Output: Set of actually joined aggregates $\mathcal{AJ}_c^i$

Sort $\mathcal{J}_c^i$ in descending order with respect to $(t_i^C / \operatorname{card}(C))$
Build the subgraph $\mathcal{G}_J = (\mathcal{J}_c^i, \Upsilon_C)$
$\mathcal{AJ}_c^i = \{\}$
for $C_p \in \mathcal{J}_c^i$ **do**
    for $C_q \in \mathcal{J}_c^i \setminus \{C_p\}$ **do**
        if *C_p and C_q are connected* **then**
            Insert $C_p \cup C_q$ in $\mathcal{AJ}_c^i$
            Remove C_p and C_q from $\mathcal{J}_c^i$
            break
Return $\mathcal{AJ}_c^i$

Algorithm 3: Join strategy based on the connected components of the subgraph.

Input : Set of aggregates selected for join $\mathcal{J}_c^i$, criteria $(t_i^C / \operatorname{card}(C))_C$
Output: Set of actually joined aggregates $\mathcal{AJ}_c^i$

Sort $\mathcal{J}_c^i$ in descending order with respect to $(t_i^C / \operatorname{card}(C))$
Build the subgraph $\mathcal{G}_J = (\mathcal{J}_c^i, \Upsilon_C)$
Compute the connected components of $\mathcal{G}_J$ (via Depth First Traversal)
$\mathcal{AJ}_c^{i+1} = \{\}$
foreach *connected component P of $\mathcal{G}_J$* **do**
    Insert $(\cup_{C \in P} C)$ in $\mathcal{AJ}_c^i$
    Remove $(C)_{C \in P}$ from $\mathcal{J}_c^i$
Return $\mathcal{AJ}_c^i$

Split strategies In the same way, two split strategies are described in Algorithm 4 and 5. Algorithm 4 proposes to bisect aggregates as long as there is free space available (keep in mind that total number of aggregates is fixed). Algorithm 5 enables to split selected aggregates into more than two parts.

Algorithm 4: Split strategy based on aggregates bisection.

Input : Set of aggregates selected for split $\mathcal{S}_c^i$, criteria $(t_i^C \operatorname{card}(C))_C$, free space
Output: Set of actually split aggregates $\mathcal{AS}_c^i$

Sort $\mathcal{S}_c^i$ in ascending order with respect to $(t_i^C \operatorname{card}(C))$
$\mathcal{AS}_c^{i+1} = \{\}$
for $C_p \in \mathcal{S}_c^i$ **do**
    if *freespace is available* **then**
        Append bisect$(C_p, 2)$ to $\mathcal{AS}_c^i$
        Update the freespace
Return $\mathcal{AS}_c^i$

Algorithm 5: Split strategy based on a uniform partitioning of aggregates.

Input : Set of aggregates selected for split $\mathcal{S}_c^i$, criteria $(t_i^C \operatorname{card}(C))_C$, F (free space)
Output: Set of actually split aggregates $\mathcal{AS}_c^i$

Sort $\mathcal{S}_c^i$ in ascending order with respect to $(t_i^C \operatorname{card}(C))$
$\mathcal{AS}_c^i = \{\}$
$P = \text{ones}(\text{card}(\mathcal{S}_c^i))$
while $F > 0$ *(free space is available)* **do**
    for $C_p \in \mathcal{S}_c^i$ **do**
        if $P(p) < \text{card}(C_p)$ **then**
            $P(p) \leftarrow P(p) + 1$
            $F \leftarrow F - 1$ (update the free space)
for $C_p \in \mathcal{S}_c^i$ **do**
    Split C_p into $P(p)$ parts
    Append the parts to $\mathcal{AS}_c^i$
Return $\mathcal{AS}_c^i$

Algorithm 6 summarizes the adaptation process. For computational efficiency there is a one-iteration delay between the classification and the aggregates update. Indeed, with this delay the local contribution to the preconditioned residual $\boldsymbol{Z}_i^p$ does not need to be recomputed.

The discrete adaptation has the advantage to present a negligible computational cost. However, the evolution of the aggregates may be really slow, even stopped, when the number of aggregates is much smaller than the number of subdomains (see Section 4). This phenomenon has motivated the development of a more continuous process presented hereafter.

Algorithm 6: AMPFETI with discrete aggregate adaptation based on the local τ-test.

Input: Initial aggregate partition split and join strategies.

AMPFETI initialization ($\ell 1$ – $\ell 6$)
while $\|\boldsymbol{r}_i\| > \epsilon$ **do**
    AMPFETI beginning of iteration ($\ell 8$ – ℓ 11)
    Adapt aggregates according to $(\mathcal{S}_c^i, \mathcal{J}_c^i, \mathcal{U}^i)$
    Local test with the newly formed aggregates ($\ell 12$)
    Classify aggregates in $(\mathcal{S}_c^{i+1}, \mathcal{J}_c^{i+1}, \mathcal{U}^{i+1})$
    AMPFETI end of iteration ($\ell 13$ – ℓ 14)

3.3 Aggregate adaptation via weighted graph partitioning

At each iteration, a new partition $(C_k)_{1 \le k \le N_a}$ is computed by a graph partitioning library. The value of the τ-test is used to define the vertexes' weight in the graph, leading to a balanced partition.

More precisely, the weights are proportional to $1/t_i^C$ in order to correctly adapt aggregates. With this choice, the vertex weight is really high when the local τ-test has detected some "meaningful information". It thus produces aggregates with few subdomains in these areas. In the case of heterogeneous plates with localized inclusions (see Section 4.2), it leads to small aggregates near the heterogeneity for instance. All

the subdomains of a given aggregate C are assigned the same vertex weight:

$$\forall \Omega^s \in C,\ \omega_v(\Omega^s) = \frac{1}{1 + \frac{t_i^C}{\sum_{k=1}^{N_a} t_i^k}} \tag{13}$$

As can be seen in (13), the value of the τ-test t_i^C is normalized with respect to the values coming from all aggregates. When the number of aggregates is small, this strategy is expected to give better results than the discrete one since the adaptation is much less constrained. The computational cost remains small since the number of edges of the subdomain connectivity graph remains low because this graph only concerns short range interactions.

Algorithm 7: Aggregate adaptation via weighted graph partitioning based on the local τ-test.

Input: Initial aggregate partition
AMPFETI initialization ($\ell 1$ – $\ell 6$)
while $\|\boldsymbol{r}_i\| > \epsilon$ **do**
  AMPFETI start of iteration ($\ell 8$ – ℓ 11)
  Call the graph partitioner and adapt the aggregates
  Local test with the newly formed aggregates ($\ell 12$)
  Compute the vertex weights $(\omega(\Omega^s))_{1 \leqslant s \leqslant N_d}$
  AMPFETI end of iteration ($\ell 13$ – ℓ 14)

4 Numerical results

In this section, the performance of the AMPFETI variants are compared. Section 4.1 compares the results obtained by AMPFETIG and AMPFETIL with various numbers of fixed aggregates. A weak scalability study on a checkerboard heterogeneous cube emphasizes that grouping search directions into aggregates provides a better control of the memory footprint without significantly degrading the time to solution. In Section 4.2, the benefit of the two-level adaptive strategy of AMPFETIL is demonstrated with a weak scalability test with localized inclusions. Let us note that FETI and MPFETI algorithms are not taken into account here. A detailed comparison of FETI, MPFETI, AMPFETI showing the good performance of AMPFETI can be found in [4].

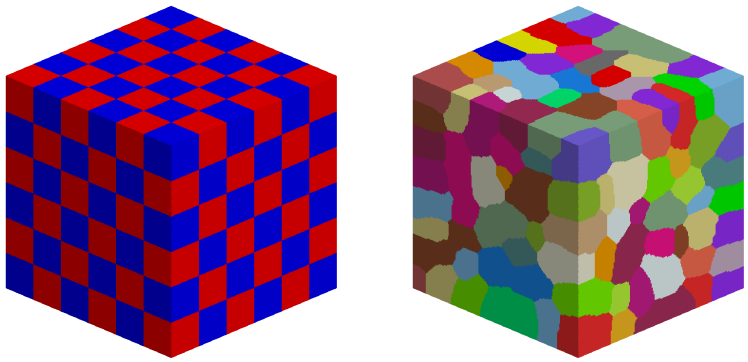
Remarks on the implementation and hardware computing resources The solvers have been implemented in the finite element suite Z-Set 9.0[1]. All the computations presented in this section have been performed using the Part 1 of the *Occigen* supercomputer. This supercomputer is made of Haswell bi-processors (E5-2690V3@2.6 GHz) which are interconnected with an Infiniband FDR 56 Gb/s network. In all configurations, MUMPS solver (version 5.1.2) [2] is used in association with the BLAS library provided by Intel 17.2 MKL for local solves. The Pardiso direct solver is used to solve the coarse problem. Eigen library[2] is used for dense linear algebra. Communication are handled by the OpenMPI library.

4.1 Weak scalability study on a highly heterogeneous checkerboard cube

This subsection is dedicated to the study of AMPFETI's scalability. The test-case is a cube with checkerboard heterogeneity pattern. Automatic graph partitioning is employed, which results in numerical difficulties spread all over the structure, making it uninteresting to try adapting aggregates. The performance of AMPFETIG is also mentioned since this variant of the solver seems well suited to that configuration.

For $n_c \in \{5, 6, 7, 8, 9, 10, 11\}$, we consider a set of three-dimensional heterogeneous cubes made of n_c^3 identical sub-cubes (see Figure 3). Each sub-cube is discretized with the same ruled mesh made of 110592 eight-node brick elements (c3d8), leading to a total number of approximately $n_c^3 \times 352,947$ degrees of freedom. With this setup, the H/h ratio equals to 48 where h is the diameter of the finite elements and H that of a typical subdomain. 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_1/E_2 between two adjacent sub-cubes. Two ratios of Young's modulus are used: 10^3 and 10^6. 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

[1] `http://www.zset-software.com/`
[2] `http://eigen.tuxfamily.org/`



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

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

Figure 3: Heterogeneous cube (configuration with $N_d = 216$, $n_c = 6$).

n_c	N_d	#DOFs total	#DOFs on interface	#cores
5	125	41,992,563	2,645,043	750
6	216	72,412,707	4,785,987	1296
7	343	114,818,259	7,752,734	2058
8	512	171,199,875	11,641,947	3072
9	729	243,548,211	17,084,832	4374
10	1000	333,853,923	23,613,033	6000
11	1331	444,107,667	32,187,410	7986
12	1728	576,300,099	42,008,133	10368

Table 1: Checkerboard cube, weak parallel scalability: configurations.

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 the FETI method. Such a configuration is represented in Figure 3 for $n_c = 6$. 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 125 subdomains and goes up to 1728 subdomains which corresponds to a total number of 10368 cores and 576 millions unknowns.

All algorithms are equipped with the best state of the art preconditioner and projector, they use a combination of Primal Schur complement and stiffness scaling. The convergence is triggered when $\|\boldsymbol{r}_i\|/\|\boldsymbol{r}_0\| \leq \epsilon$. The convergence criterion ϵ is set to 10^{-6} and the τ-test threshold is set to 10^{-2}. The influence of the choice of the preconditioner, projector and threshold is discussed at length in [4]. Several numbers of aggregates have been tested $N_a \in \{32, 64, 125\}$. Table 1 summarizes the different configurations.

The results are shown in Figures 4-5. The letter G (resp. L) stands for AMPFETI with the global τ-test (resp. local τ-test), the appended number indicates the number of aggregates. The total wall time includes the setup, the factorization of local operators and the convergence loop of the algorithm. For both AMPFETIG and AMPFETIL, and whatever the Young's modulus ratio, the same behavior is observed.

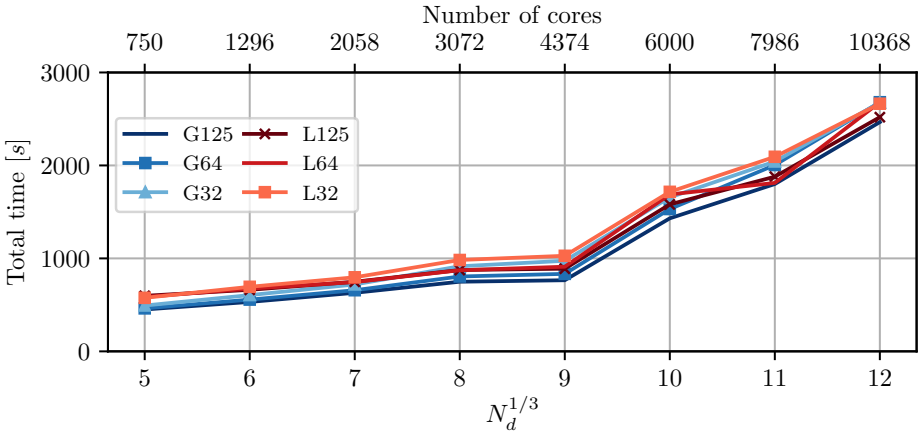
- The number of iterations needed to achieve convergence slightly increases with the number of subdomains. This behavior is not uncommon for such high level of heterogeneity and automatic decomposition. In GENEO's approaches, it is often observed that the number of modes to be eliminated per subdomains increases with the number of subdomains. In AMPFETI approaches, the bad modes must be captured through extra iterations, at least one iteration per bad mode, before superconvergence can be observed.

- Regarding the time to solution, a change of slope is clearly visible after 729 subdomains ($n_c = 9$). This change of slope is induced by two phenomena. First, the time spent in building and solving the coarse problem strongly increases when the number of subdomains becomes large in our implementation. For now, $\boldsymbol{G}^\top \boldsymbol{A}\boldsymbol{G}$ is communicated to all "subdomains" and all of them perform the factorization with a multithreaded solver (redondant work). A global communication with a

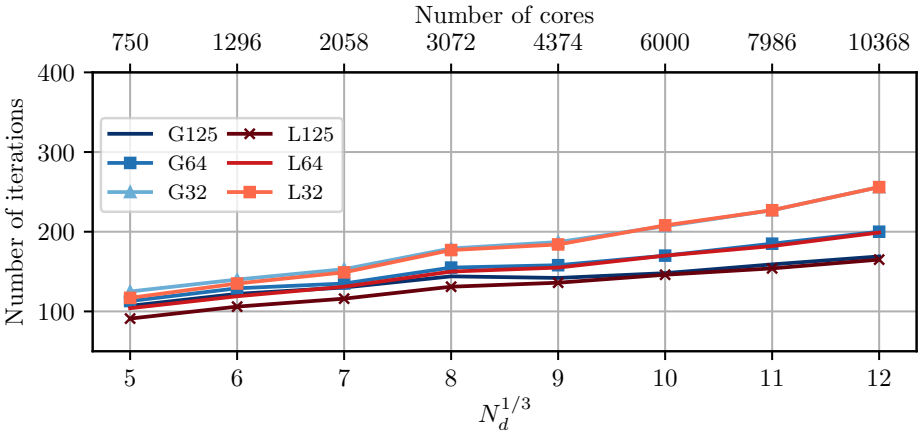
significant amount of data happens here. The implementation of a better strategy, inspired from [17], will be considered in the future. Also, a superlinear increase of the cost of all MPI_Allreduce operations involved by the full reorthogonalization is observed (even if the increase of the search space size is limited).

- As expected, the size of the search space decreases with the use of smaller number of aggregates. For the largest test case ($N_d = 1728$) and whatever the heterogeneity, the search space generated with 32 aggregates is approximately 15% smaller than the one produced with 125 aggregates. The increase of the time to solution is limited since the additional time induced by the higher number of iterations is somehow compensated by time saved in the orthogonalization.

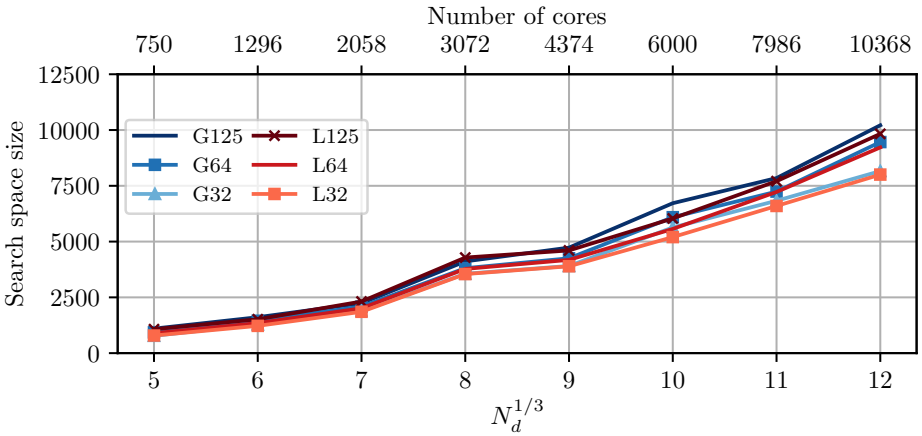
- The increase of the number of iterations with respect to N_d is slightly faster when the Young's modulus ratio is 10^6. It was expected because n_c also increases leading to more and more material coefficent jumps in the global problem and thus deteriorating the condition number.



(a) Total wall time.

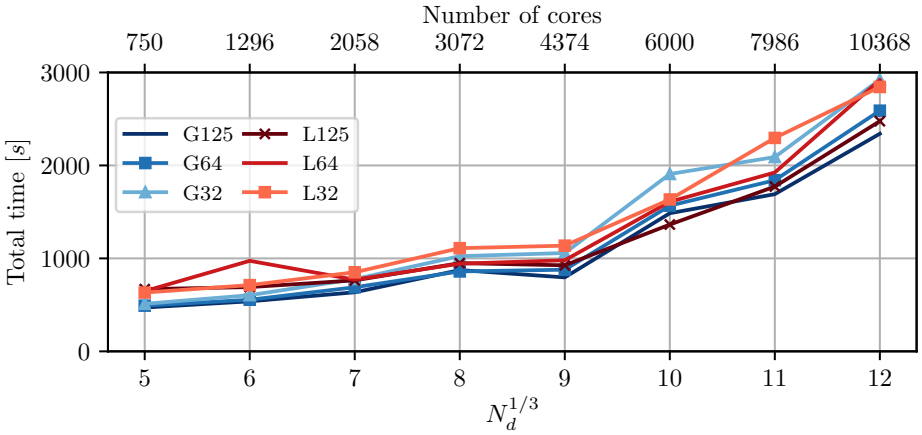


(b) Number of iterations.

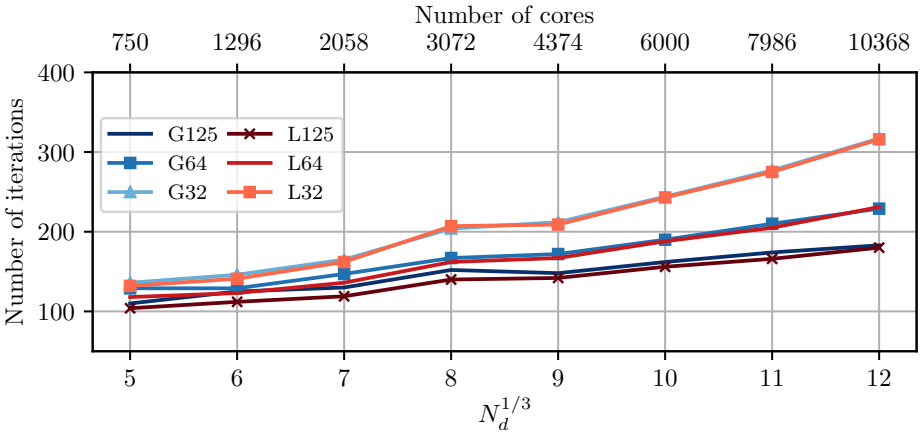


(c) Minimization space size.

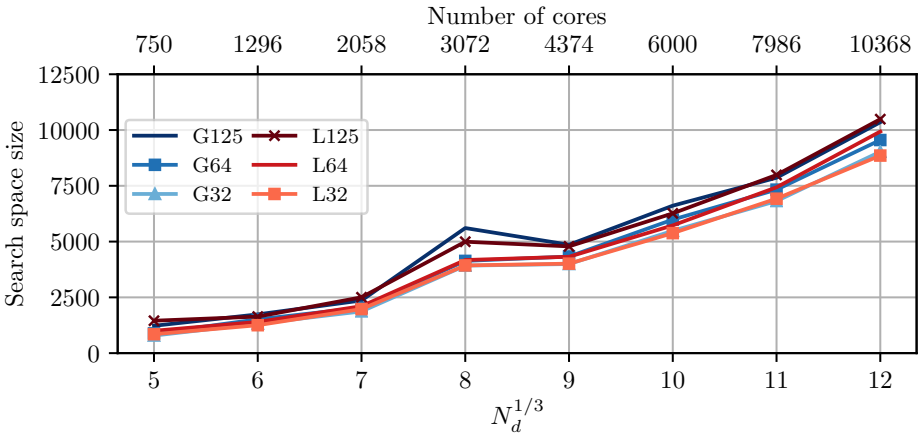
Figure 4: Checkerboard cube, weak parallel scalability ($E_1/E_2 = 10^3$): wall time, number of iterations and minimization space size.



(a) Total wall time.



(b) Number of iterations.

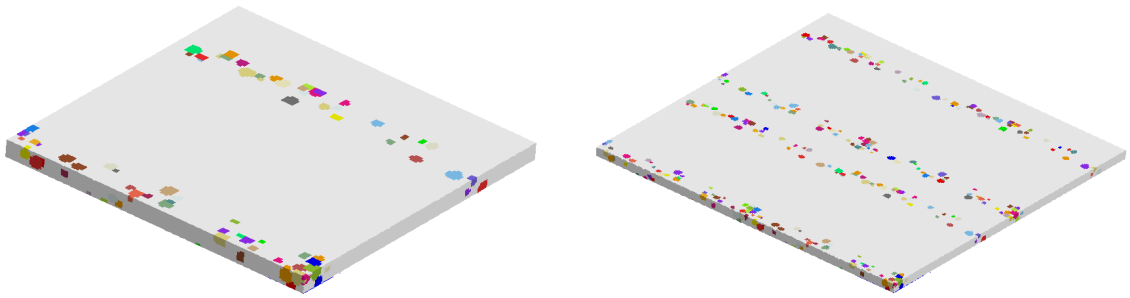


(c) Minimization space size.

Figure 5: Checkerboard cube, weak parallel scalability ($E_1/E_2 = 10^6$): wall time, number of iterations and minimization space size.

4.2 Weak scalability study on a three dimensional elastic plate with stiff inclusions

While the previous section has shown the benefit of gathering search directions into aggregates, this section highlights the advantage of the two-level adaptation. Only AMPFETI with the local τ-test is shown here since it is the only algorithm that is concerned with this adaptation.



(a) Heterogeneous 3D plate pattern.

(b) Test case with 2×2 repetition.

Figure 6: Weak scalability with localized inclusions.

The pattern is defined by a three-dimensional heterogeneous plate made of a soft matrix and stiff inclusions. In this pattern, two hundred inclusions are inserted in two strips $0 \leq x \leq 0.2\ x_{max}$ and $0.6\ x_{max} \leq x \leq 0.8\ x_{max}$ (see Figure 6). The matrix behavior is isotropic elastic linear with a Poisson's coefficient of 0.45. All spherical inclusions have a Poisson's coefficient of 0.3. The Young's modulus of the inclusions are randomly selected using uniform distribution, such that the maximum Young modulus ratio $E_{inclusion}/E_{matrix}$ is 10^6. The plate is clamped on one side and subjected to a prescribed unitary displacement in the three space directions on the opposite side, all other faces remain traction-free. The pattern is discretized with a ruled mesh made of $125 \times 125 \times 10$ eight-node brick elements (c3d8), leading to a number of $174,636$ nodes. To perform the weak scalability study, this pattern is repeated in the directions x and y, and an unstructured decomposition is obtained with a partitioning software. The number of subdomains per pattern is 120. Thus, the study starts from 2×2 patterns (480 subdomains) and goes up to 7×7 patterns (5880 subdomains). Since subdomains are really small in this test case (about $5,000$ unknowns per subdomain), only one core is attributed to each one. Table 2 summarizes the different configurations. All algorithms are equipped with stiffness scaling and Dirichlet preconditioner for $\tilde{\boldsymbol{S}}$ and $\boldsymbol{A}$. The convergence is triggered when $\|\boldsymbol{r}_i\|/\|\boldsymbol{r}_0\| \leq \epsilon$. The convergence criterion ϵ is set to 10^{-6} and the τ-test threshold is set to 10^{-2}. Several numbers of aggregates have been used $N_a \in \{32, 64, 125, 216\}$. For each number of aggregates, the adaptation based on weighted graph partitioning and two combinations of discrete split and join strategies have been tested. Table 3 gives the correspondence between the legend and the adaptation strategy presented in Section 3. The letter L stands for the non-adaptive AMPFETI with local test.

The results are summarized in Tables 4 - 7. Figure 8 compares the scalability of the L^2 variant whereas Figure 7 presents a focus on the 5880-subdomain case.

We observe that for all cases the convergence of the two-level adaptive variants is faster than the fixed aggregates variant (L). As shown in Tables 4 and Tables 5, when the number of aggregates is much smaller than the number of subdomains, all discrete adaptation variants remain inactive and do not provide any improvement. This is not the case for the continuous variant. As seen in Figure 8, the L^2 variant always leads to fewer iterations than the fixed aggregate configuration. For the case with 5880 subdomains and 64 aggregates, the gain is about 22%. More, the search space size for both configurations is approximately equal. Because of the small size of the subdomains (about 5000 unkowns), the gain in

N_d	#DOFs total	#cores
480	2,079,033	480
1080	4,665,408	1080
1920	8,283,033	1920
3000	12,931,908	3000
4320	18,612,033	4320
5880	25,323,408	5880

Table 2: Three dimensional plate with stiff inclusions, weak parallel scalability: studied configurations.

Discrete evolution of aggregates			Evolution of aggregates via graph partitioning	
Variant	Split strategy	Join strategy	Variant	Cost function
A	Bisection	Two-by-Two	L^2	Linear (13)
B	Uniform	Only connected		

Table 3: Three dimensional plate with stiff inclusions, weak parallel scalability: tested combinations.

terms of number of iterations has little effect on the time to solution. Indeed, the time spent in local solves is much less than the time spent in the coarse problem. So even if a gain is obtained in terms of iteration, it is not so much visible in the computational time. This setup was chosen to be able to reach a large number of subdomains without consuming too much computational resources.

The search space size reduction is about 10% between $N_a = 32$ and $N_a = 216$. The aggregate process leads to moderate gain here because, contrary to the checkerboard test case, the local τ-test of AMPFETIL does not saturate here but remains "frugal".

As presented in Figure 7, with the two-level adaptation method, the algorithm usually keeps selecting search directions a little longer, which means more relevant search directions are generated.

N_d	480	1080	1920	3000	4320	5880
	Number of iterations					
L	83	118	184	267	336	389
A	83	118	184	267	336	389
B	83	118	184	267	336	389
L^2	80	96	155	217	274	342
	Search space dimension					
L	1014	2076	3571	5603	7689	10381
A	1014	2076	3571	5603	7689	10381
B	1014	2076	3571	5603	7689	10381
L^2	1081	1995	3520	5607	7712	10548
	Time to solution (s)					
L	8.865	33.9	191.0	249	455	911
A	8.777	30.0	132.1	207	445	945
B	8.79	29.5	200.5	208	463	922
L^2	9.013	28.7	81.9	201	440	843

Table 4: Weak scalability with heterogeneous inclusions: summary of results for the case with 32 aggregates. The user parameter τ is set to 0.01.

N_d	480	1080	1920	3000	4320	5880
	Number of iterations					
L	68	79	124	161	190	259
A	65	79	124	161	190	259
B	65	79	124	161	190	259
L^2	64	69	106	140	168	202
	Search space dimension					
L	1047	2130	3631	5732	7821	10723
A	1062	2130	3631	5732	7821	10723
B	1092	2130	3631	5732	7821	10723
L^2	1206	2204	3645	5682	7774	10722
	Time to solution (s)					
L	10.08	30.26	98.5	208	444	791
A	9.075	28.97	85.1	213	390	790
B	9.072	33.56	85.6	212	386	756
L^2	10.09	28.98	85.0	211	444	750

Table 5: Weak scalability with heterogeneous inclusions: summary of results for the case with 64 aggregates. The user parameter τ is set to 0.01.

A focus on aggregates evolution In order to "feel" how the algorithm behaves, we propose to visualize the aggregates evolution given by the variant B on the test case with 480 subdomains with 120

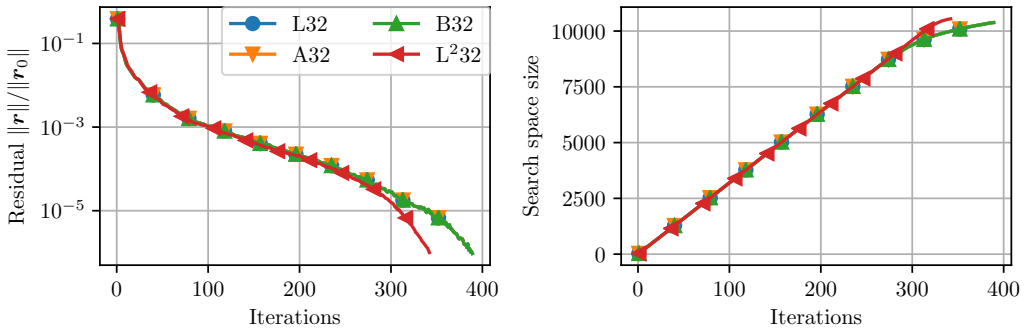
N_d	480	1080	1920	3000	4320	5880
	Number of iterations					
L	56	59	87	108	126	153
A	52	55	87	108	126	153
B	51	56	87	108	126	153
L^2	56	53	81	102	114	130
	Search space dimension					
L	1229	2339	3732	6188	8003	11065
A	1400	2222	3732	6188	8003	11065
B	1361	2341	3732	6188	8003	11065
L^2	1271	2443	4038	5927	7939	10781
	Time to solution (s)					
L	10.33	30.61	105.1	201	413	741
A	10.85	27.02	79.5	192	361	750
B	10.49	30.3	80.9	234	432	746
L^2	10.44	30.12	97.7	191	418	757

Table 6: Weak scalability with heterogeneous inclusions: summary of results for the case with 125 aggregates. The user parameter τ is set to 0.01.

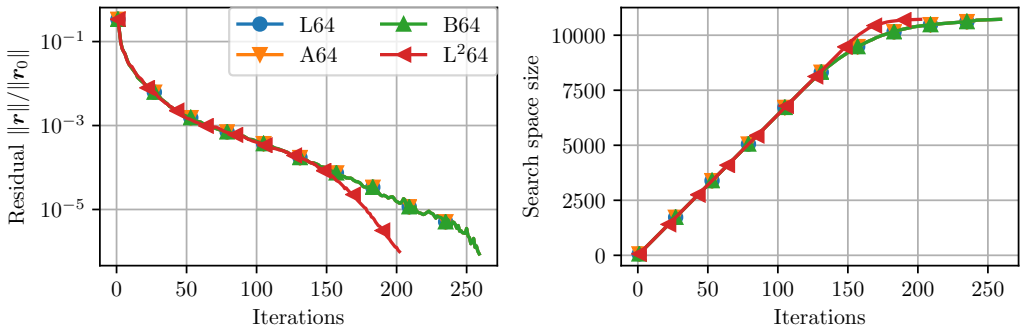
N_d	480	1080	1920	3000	4320	5880
	Number of iterations					
L	51	47	72	86	109	117
A	48	42	66	88	102	117
B	47	41	71	86	102	117
L^2	49	46	65	83	89	96
	Search space dimension					
L	1502	2697	4229	6621	8809	11252
A	1449	2599	4181	6367	8761	11252
B	1522	2671	4021	6412	8761	11252
L^2	1529	2800	4313	6374	8413	11118
	Time to solution (s)					
L	11.66	33.86	85.4	199	427	756
A	11.04	29.52	83.5	191	476	768
B	11.51	31.4	78.5	198	418	776
L^2	11.51	31.1	83.9	288	419	738

Table 7: Weak scalability with heterogeneous inclusions: summary of results for the case with 216 aggregates. The user parameter τ is set to 0.01.

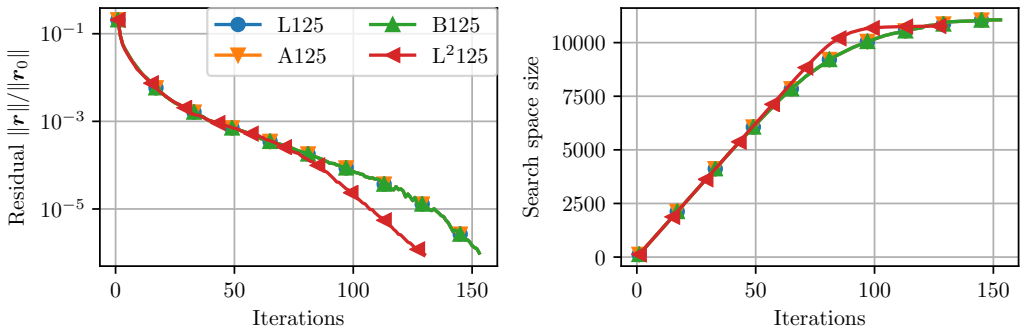
aggregates. In Figure 9b-9f, each dot represents a subdomain. These dots are situated at the gravity center of subdomains and they are linked if they belong to the same aggregate. Figure 9a is an above view of the geometry allowing to compare the aggregates evolution with the position of the stiff inclusions. Starting from a uniform distribution of aggregates, as soon as the second iteration, the algorithm generates "small" aggregates close to the stiff inclusions. After iteration 11, the distribution obtained is the final one. In Figure 9f, one can observe that close to the inclusions, allmost all aggregates only contain one subdomain. Conversely, aggregates localized in the homogeneous part of the problem gather several, and sometimes many, subdomains.



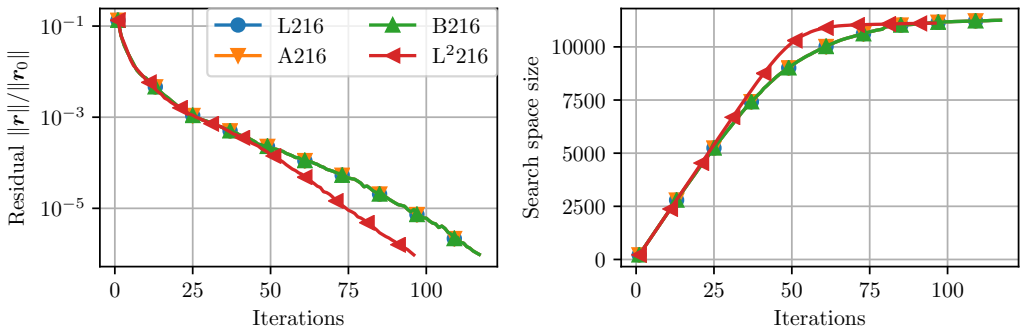
(a) Convergence results with 32 aggregates.



(b) Convergence results with 64 aggregates.

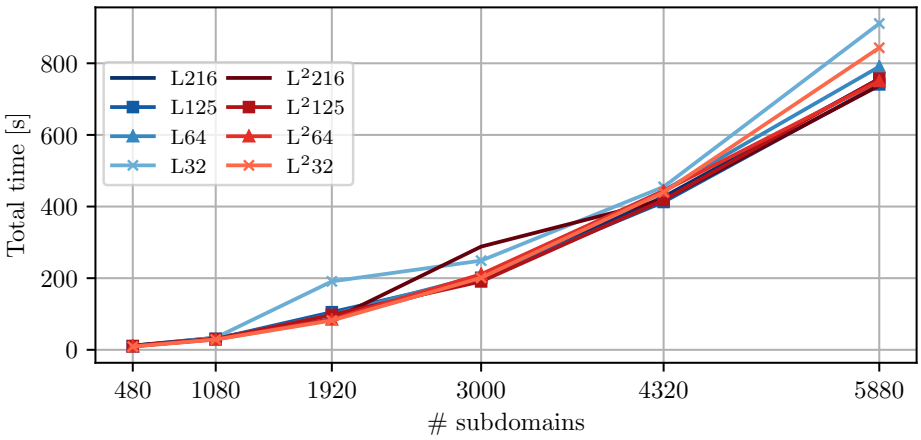


(c) Convergence results with 125 aggregates.

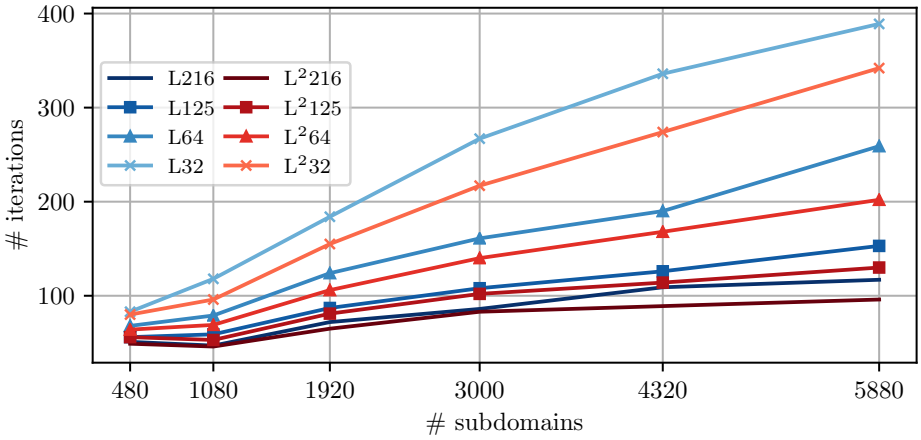


(d) Convergence results with 216 aggregates.

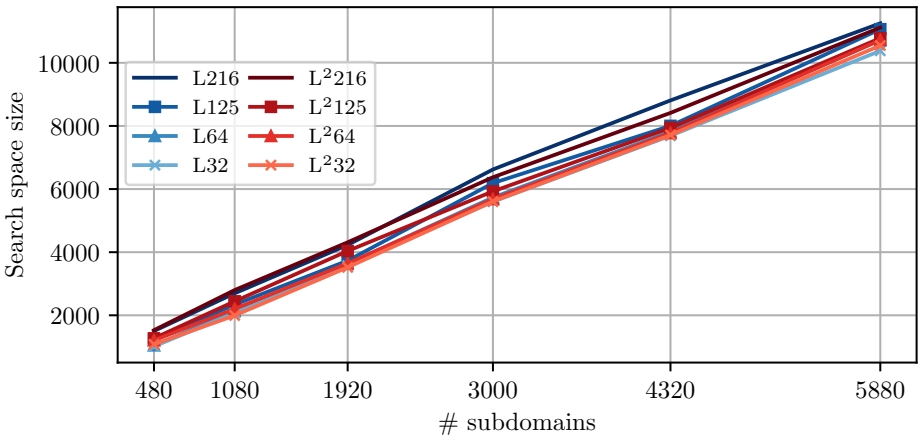
Figure 7: Weak scalability test with localized inclusions, focus on the results for the case with 5880 subdomains.



(a) Total wall time.

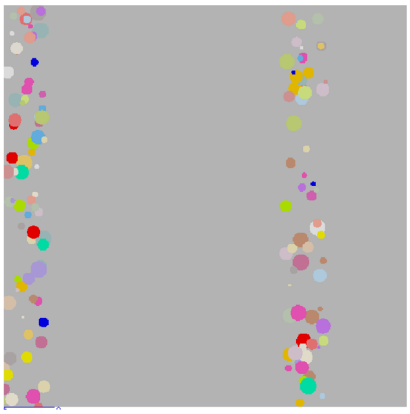


(b) Number of iterations.

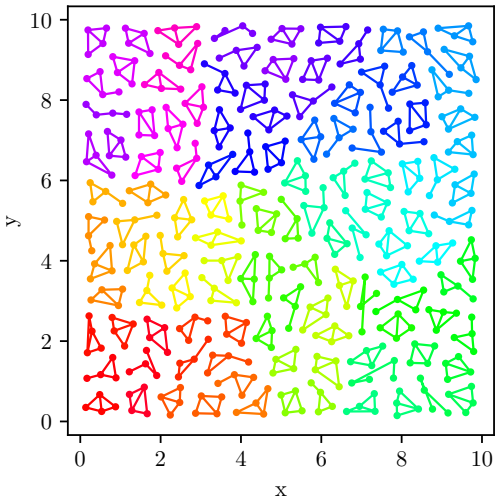


(c) Minimization space size.

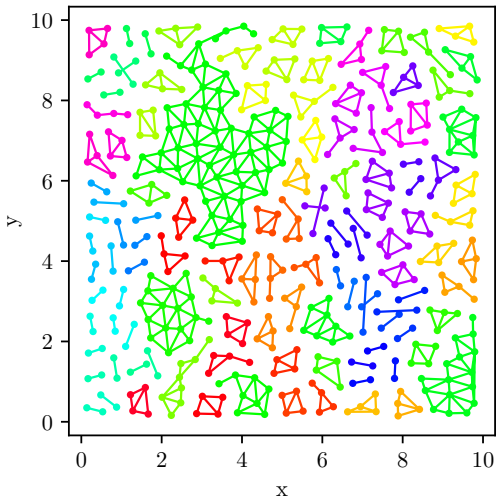
Figure 8: Weak scalability test with localized inclusions, focus on the results of the graph based update of aggregates: wall time, number of iterations and minimization space size.



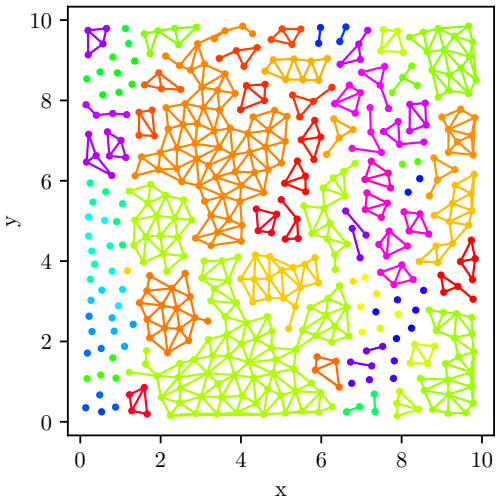
(a) Above view of the problem.



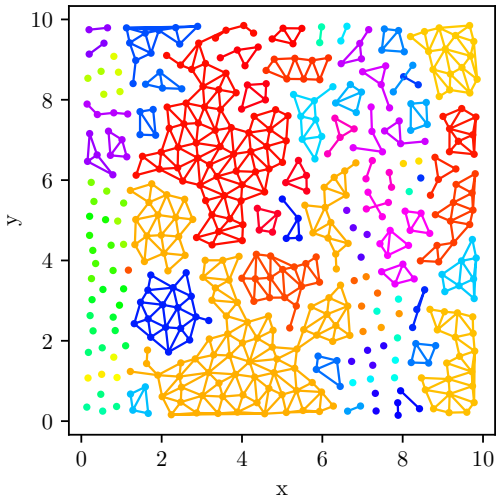
(b) Initial aggregates.



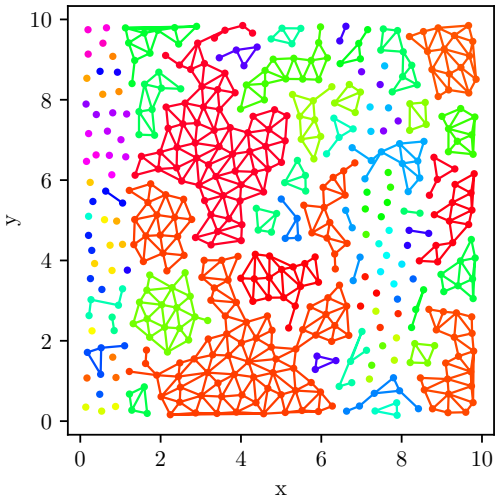
(c) Aggregates at iteration 2.



(d) Aggregates at iteration 4.



(e) Aggregates at iteration 6.



(f) Aggregates at iteration 11 (final).

Figure 9: Aggregates evolution for variant B with 480 subdomains and 120 aggregates. Each dot represents a subdomain. Dots are linked if they belong to the same aggregate.

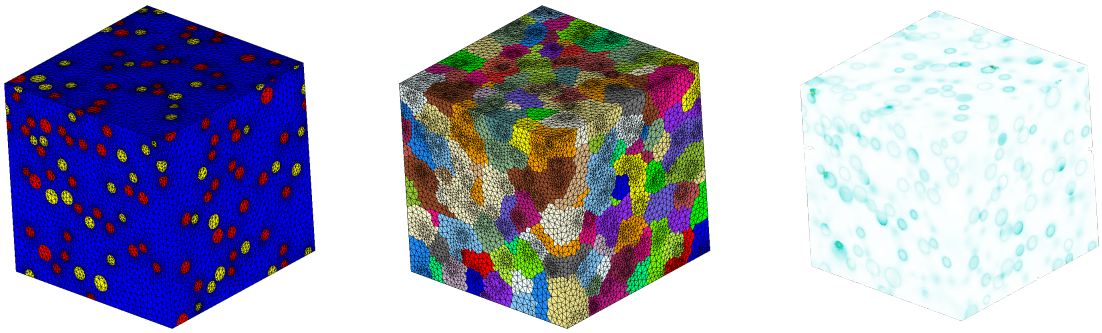
5 Applications to real engineering problems

This section presents some typical real engineering problems where the use of the AMPFETI solver is essential to achieve convergence.

5.1 Numerical homogenization of the mechanical behavior of solid propellants

Solid propellants are energetic materials composed of an organic matrix and numerous metallic inclusions. Due to the large dispersion of the sizes of the particles, this type of material brings several space scales into play. The simulation of a Representative Volume Element (RVE) with the finite element method leads to many unknowns, making the use of iterative solvers mandatory. However, the numerical homogenization of this type of material with domain decomposition methods is a challenging problem since it gathers almost all classical hard points that penalize the convergence of iterative solvers.

First, the material is highly heterogeneous, the jump in Young modulus between the matrix and the metallic inclusions is approximately 10^5. Because of the high density of inclusions in the RVE, heterogeneity is very frequently misplaced with respect to the domain decomposition interface. Also, elements of the mesh of the matrix that are located between two close inclusions often exhibit poor quality factors which degrade even more the condition number of the linear system to be solved. More, the organic matrix is almost incompressible, and mixed pressure–displacement–volume variation finite element needs to be used [1] (if a linear elastic behavior is assumed for the matrix, the Poisson coefficient is 0.499).



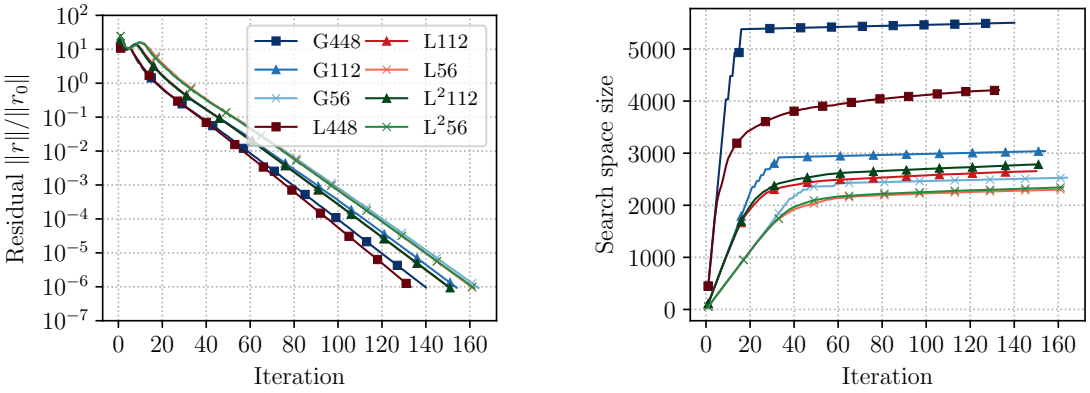
(a) Finite element mesh of the RVE, the red and yellow colors correspond to two different types of inclusions.

(b) Domain decomposition in 448 subdomains.

(c) Von Mises stress field, the white color characterises low stresses area.

Figure 10: Numerical homogenization of the mechanical behavior of solid propellants.

This is a typical case where the AMPFETI solver is needed to achieve convergence. As an example, we present some results on the linearized problem. Figure 10a shows the finite element mesh and the domain decomposition for one distribution of particles. The finite element mesh contains 5,494,528 quadratic tetrahedrons and the global system has 22 millions of degrees of freedom; 448 subdomains are used for the partitioning. The computation is run on 448 cores (MPI parallelism only). Figure 11 shows the convergence of the solver and the generated search space for various configurations. The curves G448 and L448 correspond to the original AMPFETI (with one subdomain per aggregate, G means global τ-test, L means local τ-test). Although the inclusions are spread out over all the RVE, all local variants generate a smaller search space than their global counterpart. It is especially visible when comparing G448 with L448. The use of aggregates allow to reduce the search space size without degrading too much the convergence. Here, the second level of adaptation (L^2112 and L^256) leads to the same results as the one level (L112 and L56). It may be explained by the fact that inclusions are spread out over all the RVE and not localized.



(a) Convergence history.

(b) Evolution of the total search space size.

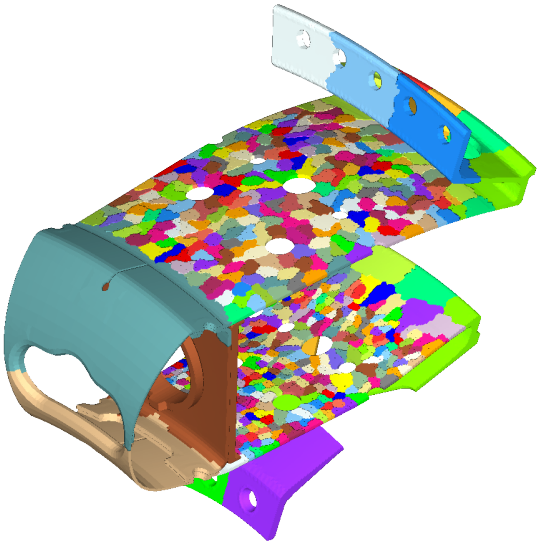
Figure 11: Solid propellant: convergence and search space size. The local preconditioner is the Dirichlet one with a *k-scaling*. The user parameter τ is 0.01.

5.2 High fidelity simulation of a multiperforated aircraft combustion chamber

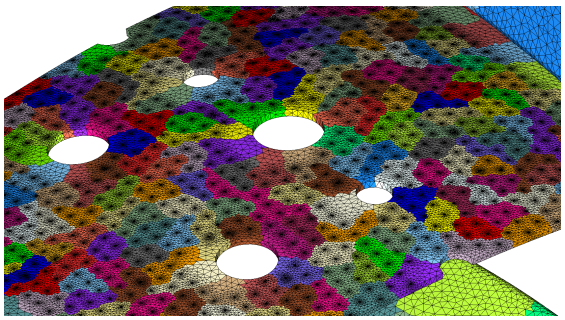
Aircraft combustion chambers are complex systems subjected to strong thermomechanical loading. To estimate the fatigue lifespan, several full load cycles need to be simulated. Thanks to AMPFETI, the simulation can be performed in a reasonable CPU time, while taking into account the anisothermal elastoviscoplastic highly nonlinear material and the fine description of the geometry. The material used is modeled with a complex elasto-viscoplastic law. This law has two potentials, the first potential is elasto-viscoplastic, it includes a Norton flow, two hardening mechanisms (one kinematic and one isotropic). The second potential is viscoplastic with a Norton flow. All material parameters are temperature dependant. Also, a complete set of 1200 micro-perforations is modeled and their influence on the stress and temperature field can be analyzed.

Figure 12a shows the domain decomposition. A zoom on the finite element mesh close to the perforations is visible in Figure 12b. The finite element mesh contains quadratic tetrahedrons and hexahedrons, the global system has approximately 25 million degrees of freedom. There are 576 subdomains, the computation is run on 3456 cores (hybrid MPI–OpenMP parallelism). Because of the material nonlinearity, a Newton-Raphson scheme is needed and AMPFETI is used as a linear solver for the successive tangent linear systems. Each Newton iteration lasts approximately one minute, this time includes the resolution of the tangent system and the integration of the nonlinear material law. As an example, the Von mises stress field at some instant of the cycle is shown in12c.

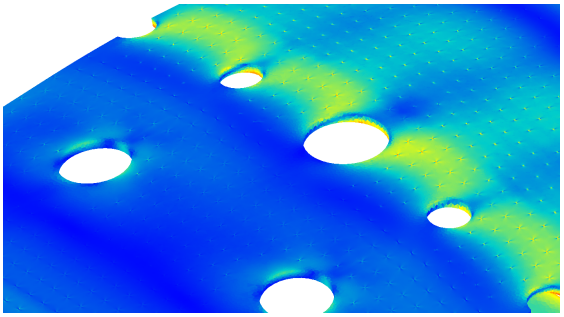
Figure 13 shows the convergence of the solver and the generated search space for various configurations for the first tangent system. The curve L576 corresponds to the original AMPFETI (with one subdomain per aggregate). Again, the use of aggregates allows a significant reduction of the search space size without degrading too much the convergence. Also, the second level of adaptivity (L^{2}288 and L^{2}144) leads approximately to the same results as their one level counterparts.



(a) 576-subdomain decomposition.

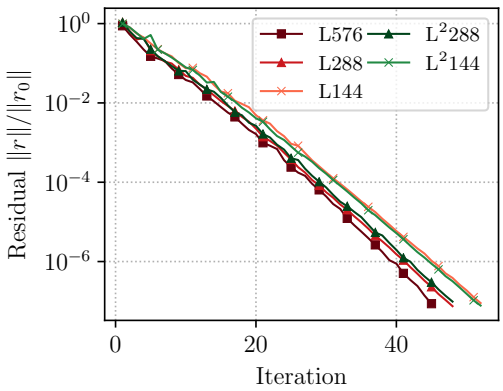


(b) Finite element mesh.

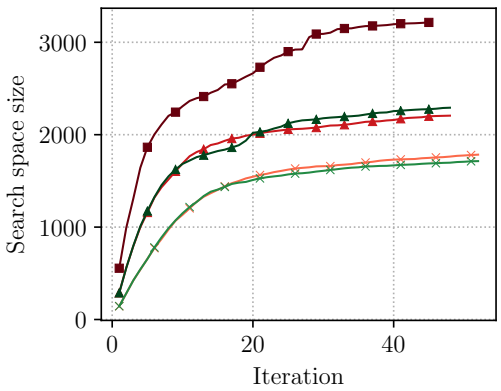


(c) Von mises stress fields

Figure 12: Multiperforated aircraft combustion chamber simulation.



(a) Convergence history.



(b) Evolution of the total search space size.

Figure 13: Multiperforated aircraf combustion chamber: convergence and search space size for the first tangent system. The local preconditioner is the Dirichlet one with a *k-scaling*. The user parameter τ is 0.1.

6 Conclusion and perspectives

This paper presents strategies to reduce the memory cost of the Adaptive Multipreconditioned FETI method.

First, the gathering of search directions into aggregates allows a better control of the memory requirements of the solver since it limits the search space growth per iteration. It is especially useful for large scale ill conditioned systems.

Then, a second level of adaptation has been introduced to improve the convergence by updating aggregates at each iteration. It is especially efficient when the computation complexity is localized in few subdomains and when the number of multipreconditioning aggregates is small in comparison to the number of subdomains. The update of the multipreconditioning aggregates makes use of the local τ-test. Two evolution strategies have been derived. The first one is based on aggregates classification into three sets (split candidates, join candidates, unchanged); the update of aggregates looks like a discrete system with a limited number of states. This strategy has the advantage to be costless, but it may be inadequate when the number of aggregates is too small. The second one, based on weighted graph partitioning, is more flexible but has a higher computational cost.

The benefits of these modifications have been assessed with two weak scalability studies going up to 10368 cores and 576 millions of unknowns. Finally, two engineering simulations leading to ill-conditioned systems have been presented. In these simulations, the use of AMPFETI was essential to achieve convergence.

One perspective of this work is to replace the sum in the gathering process by a weighted sum and to look for optimal coefficients. Other perspectives are to associate multipreconditioning with classical deflation and restarting techniques and to propose adaptive strategy for the non symmetric case.

Acknowledgements

This work benefited from the support of the project SEMAFOR ANR-14-CE07-0037 of the French National Research Agency (ANR). This work was granted access to the HPC resources of CINES under the allocation 2017-A0040910442 made by GENCI. The author would also like to thanks V. Chiaruttini for providing the engineering test cases presented in Section 5.

References

[1] Dina Al Akhrass, Julien Bruchon, Sylvain Drapier, and Sébastien Fayolle. Integrating a logarithmic-strain based hyperelastic formulation into a three-field mixed finite element formulation to deal with incompressibility in finite-strain elastoplasticity. *Finite Elements in Analysis and Design*, pages 61–70, 2014.

[2] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[3] Christophe Bovet, Pierre Gosselet, and Nicole Spillane. Multipreconditioning for nonsymmetric problems: The case of orthomin and biCG. *Comptes Rendus Mathematique*, 355(3):354–358, 2017.

[4] Christophe Bovet, Augustin Parret-Fréaud, Nicole Spillane, and Pierre Gosselet. Adaptive multipreconditioned FETI: Scalability results and robustness assessment. *Computers & Structures*, pages 1–20, 2017.

[5] Christophe Bovet, Augustin Parret-Fréaud, Nicole Spillane, and Pierre Gosselet. Méthode de décomposition de domaine multipréconditionnée et adaptative pour les problèmes mal conditionnés. In *13e colloque national en calcul des structures*, Giens, Var, France, May 2017. Université Paris-Saclay.

[6] Christophe Bovet, Augustin Parret-Fréaud, and Pierre Gosselet. Two-level adaptation for adaptive multipreconditioned FETI. *Advances in Engineering Software*, 152, 2021.

[7] Robert Bridson and Chen Greif. A multipreconditioned conjugate gradient algorithm. *SIAM J. Matrix Anal. Appl.*, 27(4):1056–1068 (electronic), 2006.

[8] C. R. Dohrmann. A preconditionner for substructuring based on constrained energy minimization. *SIAM Journal for Scientific Computing*, 25:246, 2003.

[9] Z. Dostal. Conjugate Gradient Method with Preconditioning by Projector. *Computer Mathematics*, 23(3-4):315–323, 1988.

[10] C. Farhat, M. Lesoinne, P. LeTallec, K. Pierson, and D. Rixen. 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*, 50(7):1523–1544, 2001.

[11] C. Farhat and F. X. Roux. The dual Schur complement method with well-posed local Neumann problems. *Contemporary Mathematics*, 157:193, 1994.

[12] Charbel Farhat, Luis Crivelli, and Francois-Xavier Roux. Extending substructure based iterative solvers to multiple load and repeated analyses. *Computer Methods in Applied Mechanics and Engineering*, 117(1):195 – 209, 1994.

[13] Charbel Farhat and Francois-Xavier Roux. A method of finite element tearing and interconnecting and its parallel solution algorithm. *International Journal for Numerical Methods in Engineering*, 32(6):1205, 1991.

[14] P. Gosselet, C. Rey, and J. Pebrel. Total and selective reuse of Krylov subspaces for the solution to a sequence of nonlinear structural problems. *International Journal for Numerical Methods in Engineering*, 94(1):60–83, 2013.

[15] Pierre Gosselet, Daniel Rixen, François-Xavier Roux, and Nicole Spillane. Simultaneous FETI and block FETI: Robust domain decomposition with multiple search directions. *International Journal for Numerical Methods in Engineering*, 104(10):905–927, 2015. nme.4946.

[16] Chen Greif, Tyrone Rees, and Daniel B. Szyld. GMRES with multiple preconditioners. *SeMA Journal*, pages 1–19, 2016.

[17] Vaclav Hapla and David Horak. TFETI Coarse Space Projectors Parallelization Strategies. In Roman Wyrzykowski, Jack Dongarra, Konrad Karczewski, and Jerzy Waśniewski, editors, *Parallel Processing and Applied Mathematics*, pages 152–162, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[18] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[19] Michael C. Leistner, Pierre Gosselet, and Daniel J. Rixen. Recycling of Solution Spaces in Multi-Preconditioned FETI Methods Applied to Structural Dynamics. *International Journal for Numerical Methods in Engineering*, 2018.

[20] Jan Mandel. Balancing domain decomposition. *Communications in Numerical Methods in Engineering*, 9(3):233, 1993.

[21] Jan Mandel and Bedřich Sousedík. Adaptive selection of face coarse degrees of freedom in the BDDC and the FETI-DP iterative substructuring methods. *Comput. Methods Appl. Mech. Engrg.*, 196(8):1389–1399, 2007.

[22] Michal Merta, Lubomir Riha, Ondrej Meca, Alexandros Markopoulos, Tomas Brzobohaty, Tomas Kozubek, and Vit Vondrak. Intel xeon phi acceleration of hybrid total feti solver. *Advances in Engineering Software*, 112:124 – 135, 2017.

[23] Roberto Molina and François-Xavier Roux. New implementations for the Simultaneous-FETI method. *International Journal for Numerical Methods in Engineering*, 118(9):519–535, 2019.

[24] Dianne P. O'Leary. The block conjugate gradient algorithm and related methods. *Linear Algebra and its Applications*, 29:293–322, 1980. Special Volume Dedicated to Alson S. Householder.

[25] F. Risler and C. Rey. Iterative accelerating algorithms with Krylov subspaces for the solution to large-scale non-linear problems. *Numerical algorithms*, 23:1, 2000.

[26] D. Rixen. *Substructuring and Dual Methods in Structural Analysis*. PhD thesis, Université de Liège, Belgium, Collection des Publications de la Faculté des Sciences appliquées, n.175, 1997.

[27] Daniel J. Rixen and Charbel Farhat. A simple and efficient extension of a class of substructure based preconditioners to heterogeneous structural mechanics problems. *International Journal for Numerical Methods in Engineering*, 44(4):489–516, 1999.

[28] N. Spillane. *Robust domain decomposition methods for symmetric positive definite problems*. PhD thesis, Thèse de l'Ecole doctorale de Mathématiques de Paris centre, Laboratoire Jacques Louis Lions, Université Pierre et Marie Curie, Paris, 2014.

[29] Nicole Spillane. An Adaptive Multipreconditioned Conjugate Gradient Algorithm. *SIAM J. Sci. Comput.*, 38(3):A1896–A1918, 2016.

[30] Nicole Spillane and Daniel J. Rixen. Automatic spectral coarse spaces for robust FETI and BDD algorithms. *International Journal for Numerical Methods in Engineering*, 95(11):953–990, 2013.

[31] Jari Toivanen, Philip Avery, and Charbel Farhat. A multilevel FETI-DP method and its performance for problems with billions of degrees of freedom. *International Journal for Numerical Methods in Engineering*, 116(10-11):661–682, 2018.

[32] Andrea Toselli and Olof Widlund. *Domain decomposition methods—algorithms and theory*, volume 34 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 2005.