LIÈGE université
Sciences Appliquées

GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models with application to the thermomechanical simulation of an ITER front mirror

A thesis submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy (PhD) in Engineering Science

by

Kim LIEGEOIS

Supervisor:	Maarten ARNST
Co-supervisor:	Romain BOMAN

Doctoral college in Aerospace and Mechanics

September 2020

Doctoral Thesis

GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models with application to the thermomechanical simulation of an ITER front mirror

Author:
Kim LIEGEOIS

Supervisors:
Prof. Maarten ARNST
Dr. Romain BOMAN

A thesis submitted in fulfillment of the requirements for the degree of Doctor in engineering sciences and technology

in the

Computational and Stochastic Modeling Research Group
Department of Aerospace and Mechanical Engineering

September 10, 2020

Thesis jury:

Prof. Maarten Arnst	(University of Liège)
Dr. Romain Boman	(University of Liège)
Prof. Christophe Geuzaine	(University of Liège)
Prof. Pierre Duysinx	(University of Liège)
Prof. Bedřich Sousedík	(University of Maryland)
Dr. Eric T. Phipps	(Sandia National Laboratories)
Dr. Philippe Mertens	(Forschungszentrum Jülich)
Dr. Thomas Toulorge	(CENAERO)

FREEDOM TO RESEARCH

Acknowledgments

First and foremost, I would like to express my sincere gratitude to my supervisors, Maarten Arnst and Romain Boman.

Prof. Maarten Arnst, thank you for this research project, thanks a lot for your help to search funding for the project, for all the time that you committed in our weekly meetings, your help, and for giving the opportunity to work on embedded ensemble propagation. Thanks a lot too for introducing me to your collaborators such as Eric T. Phipps, Christian Soize, Roger Ghanem, John D. Jakeman, and Paul Constantine.

Romain, I want to thank you for the considerable amount of time that you spent to help me. This thesis would have never been finished if you were not here to motivate me to work at the edges of my capabilities. Thanks a lot for *Waves*, your computational framework that helped me to dive quickly in the implementation work.

Besides my supervisors, I would like to thank the collaborators of this work, Eric T. Phipps, Tobias A. Wiesner, Philippe Mertens, and Yury Krasikov.

Eric T. Phipps, I want to gratefully acknowledge you for your confidence, for your precious help, for teaching me how to use and contribute to the Trilinos library, and for all those invitations to conferences. Finally, I would like to thank you for giving me those opportunities to visit Sandia National Laboratories in Albuquerque once a year for the past four years.

Tobias A. Wiesner, thanks a lot for introducing me to MueLu and for helping me to implement your algorithm inside MueLu. Thank you for your help concerning our common paper.

Philippe Mertens, I want to thank you for giving me the opportunity to work on the ITER project since the beginning of my master thesis, for all the time that you invested in our meetings at each of my visit at Forschungszentrum Jülich, for the interest that you manifested in the results, and for proposing me to contribute to one of your papers.

Yury Krasikov, thanks a lot for your availability, our discussion about results, and all the data that you provided me including CAD files and material properties.

I would like to thank the rest of my thesis committee: Prof. Christophe Geuzaine for helping me to better use gmsh and for giving me the opportunity to present my work at the CECI Scientific days 2019 and Prof. Pierre Duysinx for your time and support during thesis committee and for your invitations to several of your research group events.

I would like to thank the rest of the examination committee: Prof. Bedřich Sousedík and Dr. Thomas Toulorge for the time that you have spent reading and evaluating this thesis.

Moreover, I would like to thank all the people that helped me with Trilinos and Kokkos: Luc Berger, Eric C. Cyr, Christian Glusa, Glen Hansen, Mark Hoemmen, Kyungjoo Kim, Drew P. Kouri, Siva Rajamanickam, Denis Ridzal, and Christian Trott.

Thanks a lot to all of those who helped me to feel at home in New Mexico: Daniel D. Barnett, Marta D'Elia, Mauro Perego, and Julien Tranchida.

I would like to thank the rest of my colleagues for our discussions, your help, and support: Pablo Alarcón, Simon Bauduin, Dominik Boemer, Arnaud Budo, Kevin Bulthuis, Maxime Collet, Joffrey Coheur, Adrien Crovato, Edouardo Fernandez Sanchez, Luc Papeleux, Juliano Todesco, and Denis Trillet.

Finally, I would like to thank my friends, with a special thanks to Arthur Longrée for his proof-reading, my family, my in-laws, and my dear wife, Pauline Coulon, for their daily wholehearted support.

Abstract

Parametric computations, and in particular, uncertainty quantification, are key components of predictive simulation. Those computations in the context of multiphysics models typically require a substantial number of realizations of costly finite element models. In this work, we are particularly interested in problems with non-symmetric or indefinite matrices. As an illustration of this class of problems, we consider contact mechanics problems in saddle point formulation and thermomechanical simulations. Embedded ensemble propagation was proposed by Phipps et al. to improve the efficiency of nonintrusive uncertainty quantification methods of computational models on emerging computational architectures. It consists of simultaneously evaluating the model for a subset of samples together, instead of evaluating them individually. This method replaces scalars by vectors, vectors by matrices, and matrices by higher-order tensors. Having matrices instead of vectors raises questions such as the definition of an inner product. A first approach introduced to solve parametric linear systems with ensemble propagation is *ensemble reduction*. In Krylov methods for example, this reduction consists in coupling the samples together using an inner product that sums the sample contributions. Ensemble reduction has the advantages of being able to use optimized implementations of BLAS functions and having a stopping criterion which involves only one scalar. However, the reduction potentially decreases the rate of convergence of the iterative method due to the gathering of the spectra of the samples. In the work of Phipps et al., they have investigated the effect of ensemble propagation to solve symmetric and positive definite linear problems using the conjugate gradient method. In this work, we investigate ensemble propagation in the case of GMRES to be able to solve problems with non-symmetric or indefinite matrices. In particular, we investigate GMRES without ensemble reduction to solve each sample simultaneously but independently to improve the convergence compared to ensemble reduction. This raises two new issues which are solved in this thesis: the fact that optimized implementations of BLAS functions cannot be used anymore and that ensemble divergence, whereby individual samples within an ensemble must follow different code execution paths, can occur. We tackle those issues by implementing a high-performing ensemble dense matrix-vector product (GEMV) and by using masks. The proposed ensemble GEMV leads to a similar cost per GMRES iteration for both approaches, i.e. with and without reduction. For illustration, we study the performances of the new linear solver on four academic problems including one non-linear contact problem. These examples demonstrate improved ensemble propagation speed-up without reduction. Finally, the method is applied to accelerate the uncertainty quantification study of a model problem relevant for the design of an optomechanical system for ITER, the fusion reactor, for which the measured final speed-up of using embedded ensemble propagation is about 2.

Contents

Chapter 1

Introduction

"Computers can do two things: (1) move data from one place to another and (2) linear algebra."

Mark Hoemmen, Sandia National Laboratories

Whether a GPU is used to train a neural network, a mobile phone is used to record a video, or an HPC system is used to estimate the sea-level rise due to ice melting in Antarctica, computers are either requesting data movements or computing simple operations. Performance of finite element solvers is induced, among others, by how the data movements and the linear algebra are executed.

In this thesis, we investigate how to implement these data movements and linear algebra operations efficiently on emerging computational architectures with enhanced vector extensions, such as Intel Skylake, in the context of parametric computations of finite element models which lead to non-symmetric or indefinite matrices.

1.1 Context

The present work is motivated by three research axes: multiphysics models, parametric computations, and high performance scientific computing (HPC) as illustrated in Fig. 1.1. The overarching objective of this thesis is to accelerate parametric computations of multiphysics models on HPC architectures. In this section, we review those three axes and define the key concepts to introduce precisely our contributions.

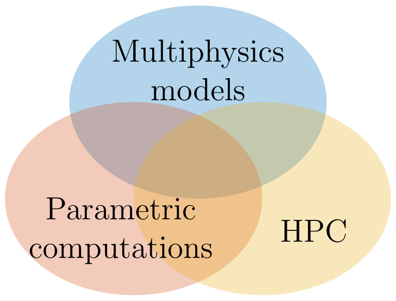


Figure 1.1: The three research axes that motivated the presented work.

The first axis is multiphysics modeling. The industrial context that drives this work is related to a mirror for one of the diagnostic systems of ITER, the largest tokamak fusion reactor currently under construction (section 1.1.1). This industrial context provides us with several multiphysics problems: due to the high temperatures it is exposed to, the mirror undergoes thermomechanical deformations that modify its optical properties. Moreover, the contact between the mirror and its mounts is altered, reducing the efficiency of the cooling system. These coupled mechanisms result in model problems with non-symmetric or indefinite matrices. These two properties of the matrices will impact the strategies considered to perform the numerical simulations (section 1.1.8).

The second axis is parametric computations, by which we mean computations that investigate the behavior of the predictions of physical engineering systems as a function of their parameters (section 1.1.2). In particular, we investigate the following numerical method to carry out parametric computation efficiently: embedded ensemble propagation (section 1.1.3). Embedded ensemble propagation relies on solving the computational model efficiently for several samples at the same time by improving the access to the memory and improving the CPU operations on the samples. This in turn will reduce the overall CPU cost at the expense of requiring modifications to the simulation code.

The last axis of this thesis is HPC. We want to contribute to the parametric computations on HPC systems and emerging architecture (section 1.1.7). To do so, we use the high-performance C++ software component library Trilinos (section 1.1.5) to investigate embedded ensemble propagation (section 1.1.6). The implementation of embedded ensemble propagation relies heavily on the C++ template mechanism (section 1.1.4).

Built on those three axes, the motivation is to accelerate the parametric computation of problems with non-symmetric or indefinite matrices, such as the model problems of the mirror of ITER, on HPC architectures.

1.1.1 Front mirror of the ITER CXRS diagnostic system

The leading numerical application of this work is related to the thermomechanical simulation of the front mirror for one of the diagnostic systems of the fusion reactor ITER.

In order for the deuterium-tritium fusion reaction to occur, deuterium and tritium particles must have enough energy to overcome the Coulomb energy barrier by tunneling effect. A possible way to reach that energy is to heat up the particles until they reach a plasma state. The reached temperature being very high (about 150 000 000 °C), the plasma must be confined to protect the plasma-facing surface of the reactor. In tokamak fusion reactors such as ITER, the confinement of the plasma is done magnetically. In order to operate both the heating and the confinement of the plasma, it is important to measure the plasma properties such as the temperature or impurity concentrations.

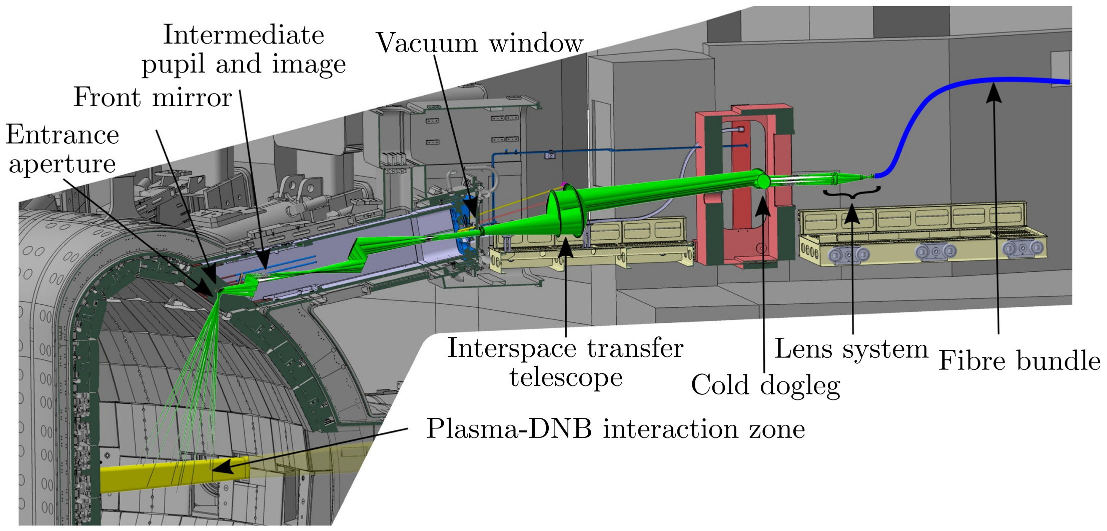

Intermediate pupil and image
Vacuum window
Front mirror
Entrance aperture
Interspace transfer telescope
Lens system
Fibre bundle
Cold dogleg
Plasma-DNB interaction zone

Figure 1.2: The CXRS core optical path and general setup. The front mirror is the closest reflecting surface to the plasma. Picture by courtesy of Krimmer et al. [2019].

In this context, the research group of Philippe Mertens from FZ Jülich, Germany is interested in the design of the front mirror of the Charge eXchange Recombination Spectroscopy (CXRS) diagnostic system of ITER illustrated in Fig. 1.2. This diagnostic system will include a sequence of mirrors that will reflect light emitted by the plasma toward a spectrometer. The first mirror is designed by Krasikov et al. [2015]. One of the key issues in the ongoing design process is that thermal fluxes and particle fluxes coming from the plasma can cause thermomechanical deformations of these mirrors, especially for the front mirror which is the closest to the plasma. The optical quality of this opto-mechanical system – by which we mean mirrors, mounts, and other components of the mirror sequence – can be adversely affected by those thermomechanical deformations. Moreover, neutron irradiation can alter material properties such as the thermal conductivity of mirrors or mounts. A preliminary study of this design has been the subject of my master thesis [Liegeois, 2015] and this issue will be investigated in more detail in this work in the context of parametric study.

These coupled thermomechanical models are at the core of this thesis in order to develop and analyze new parametric computation strategies.

1.1.2 Parametric computations and uncertainty quantification

Parametric computations investigate the behavior of the solution of a model as a function of its model parameters. Those computations can have several objectives such as optimizing a design, performing a sensitivity analysis of the response, or evaluating the robustness of the response with respect to uncertainties in the model input parameters [Ghanem et al., 2017; Soize, 2017].

We are interested in parametric computations of numerical models which can be seen as a function f that associates a unique solution $\boldsymbol{y}$ to an instance of the input parameters $\boldsymbol{p}$ as illustrated in Fig. 1.3.

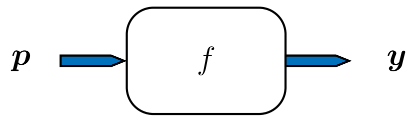


Figure 1.3: Illustration of a numerical model seen as an “input/output” system.

Usually, the function f is not known explicitly as it is the case when f corresponds to the solution of a numerical model, which is the focus of this thesis. In particular, we are interested in the parametric computation of finite element discretizations of a system of Partial Differential Equations (PDE).

Parametric computation strategies can be classified based on their impacts on the source code of the numerical model. On the one hand, there are the so-called *nonintrusive methods*, reviewed for instance by Arnst and Ponthot [2014], that can be implemented as wrappers around existing solvers without requiring modification of their source code. Nonintrusive methods have the advantage that they use the model f as a “black-box”. On the other hand, there are the so-called *intrusive methods* or *embedded methods* that require modification of the source code of the simulation software.

Nonintrusive methods for uncertainty quantification

The best-known example of a nonintrusive approach for uncertainty quantification is the Monte Carlo method as described in Robert and Casella [2013]. This approach consists in drawing N random samples $\boldsymbol{p}^{(1)}, \ldots, \boldsymbol{p}^{(N)}$ of the uncertain input parameters based on their probability density function and evaluating the computational model for each of those samples as illustrated in Fig. 1.4. Based on the output values $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(N)}$ for all of the N samples, it is then possible to have estimates of the statistical descriptors related to $\boldsymbol{y}$ such as the expected value or the standard deviation. The cost of this strategy depends both on N, the number of times that the model has to be evaluated, and the CPU cost of the evaluation of one instance of the model.

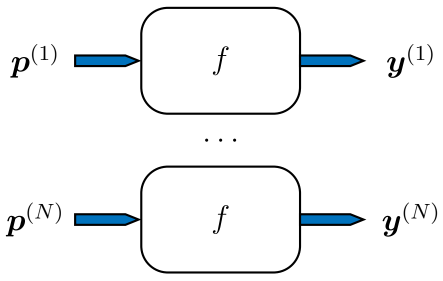


Figure 1.4: Illustration of the N evaluations of the computational model f.

When the evaluation of the model f is computationally expensive, a well-known approach to reduce the cost of an uncertainty quantification method such as the Monte Carlo method is to replace the so-called *high-fidelity* model f by a surrogate model $\hat{f}$. The surrogate model $\hat{f}$ is typically cheaper to evaluate.

A standard approach to approximate the function f is to use a linear combination of d polynomials $\phi^{(1)}, \dots, \phi^{(d)}$ which are orthonormal with respect to the probability density function of the input parameters:

$$f(\boldsymbol{p}) \approx \hat{f}(\boldsymbol{p}) = \sum_{i=1}^{d} \hat{f}^{(i)} \phi^{(i)}(\boldsymbol{p}), \tag{1.1}$$

where the weights $\hat{f}^{(1)}, \dots, \hat{f}^{(d)}$ have to be computed such that $f(\boldsymbol{p}) \approx \hat{f}(\boldsymbol{p})$. There are different strategies to compute those weights depending on the sense of the approximation.

A well-known strategy to compute the weights of (1.1) is the NonIntrusive Spectral Projection method (NISP) as described in Le Maître and Knio [2010]. The weights $f^{(1)}, \dots, f^{(d)}$ are computed using the orthogonality property by projecting f on the polynomials $\phi^{(1)}, \dots, \phi^{(d)}$, called *basis functions*, using a multidimensional quadrature rule $\left\{\left(w^{(k)}, \boldsymbol{p}^{(k)}\right); k = 1, \dots, N\right\}$:

$$\hat{f}^{(i)} \approx \sum_{k=1}^{N} w^{(k)} f\left(\boldsymbol{p}^{(k)}\right) \phi^{(i)}\left(\boldsymbol{p}^{(k)}\right), \quad i = 1, \dots, d. \tag{1.2}$$

A well-known limitation of this approach is the so-called *curse of dimensionality*: the number of samples N increases exponentially with the number of random variables as discussed in Le Maître and Knio [2010]. Improvements have been proposed to mitigate this curse of dimensionality by reducing the required number of samples with, for instance, sparse grid quadratures, proposed by Smolyak [1963] in the context of multidimensional quadrature and interpolation.

Surrogate-based approaches are used in other parametric computations than the uncertainty quantification too. For instance, the surrogate models can be used in so-called *surrogate-based optimization* or *meta-model based optimization* as done, for instance, in Dakota [Adams et al., 2019] developed at Sandia National Laboratories and Minamo [Baert et al., 2015] developed at Cenaero. Among others, Minamo provides surrogate models which support mixed variables such as both continuous and discrete input variables as discussed by Beauthier et al. [2014], several interpolators such as radial basis function networks and ordinary Kriging and automatic strategies to select and aggregate surrogate models as discussed in [Beaucaire et al., 2019b,a], and strategies to evaluate feasible regions of constrained optimization problems using surrogate models [Beauthier et al., 2017].

Intrusive methods for uncertainty quantification

A strategy closely related to the NISP, which uses the concept of orthonormal polynomials with respect to the probability density function, is the best-known intrusive method for uncertainty quantification: the stochastic Galerkin method as described by Ghanem and Spanos [1990, 1991]. The stochastic Galerkin method relies on the discretization of the weak form of the stochastic PDE. The method seeks an approximation of the solution $\boldsymbol{u}(\boldsymbol{x}, \boldsymbol{p})$ of the stochastic PDE where $\boldsymbol{x}$ denotes a position in the computational domain:

$$\hat{\boldsymbol{u}}(\boldsymbol{x}, \boldsymbol{p}) = \sum_{i=1}^{m} \sum_{j=1}^{d} u^{(i,j)} \psi^{(i)}(\boldsymbol{x}) \phi^{(j)}(\boldsymbol{p}), \tag{1.3}$$

where $\psi^{(1)}, \ldots, \psi^{(m)}$ and $\phi^{(1)}, \ldots, \phi^{(d)}$ are the basis functions of the computational domain and the stochastic space respectively, and $\{u^{(i,j)}; i = 1, \ldots, m, j = 1, \ldots, d\}$ are the $m \times d$ degrees of freedom of the discretized stochastic problem such that:

$$\mathrm{E}\left[\mathcal{M}_h(\hat{\boldsymbol{u}}, \boldsymbol{p})\, \phi^{(j)}\right] = 0, \quad j = 1, \ldots, d, \tag{1.4}$$

where $\mathrm{E}[\cdot]$ is the mathematical expectation and $\mathcal{M}_h(\hat{\boldsymbol{u}}, \boldsymbol{p})$ the residual of the discretized PDE. The output of the model $\boldsymbol{y}$ can then be evaluated using the approximated solution $\hat{\boldsymbol{u}}(\boldsymbol{x}, \boldsymbol{p})$. Discretizing both the computational domain and the stochastic space impacts the simulation software by modifying the system of equations that has to be solved. In particular, this approach replaces the deterministic system resulting from the discretized PDE with a so-called *stochastic polynomial chaos system* (1.4). This stochastic polynomial chaos system has a larger size $m \times d$ instead of m, two nested sparsity patterns, and raises challenges to be solved efficiently such as the choice of the preconditioner [Sousedík et al., 2014; Ullmann, 2010] or the choice of the ordering of the $m \times d$ degrees of freedom which impacts the memory access patterns [Phipps et al., 2014]. As illustrated in Le Maître and Knio [2010], the stochastic projection of non-differentiable problems will generally exhibit a slow convergence with the expansion order d. In consequence, as non-differentiable functions require larger expansion order, the size of their stochastic polynomial chaos system increases. This increases the CPU cost of the method for those types of problems.

There are other intrusive methods for uncertainty quantification such as the perturbation method which consists in computing derivatives of f and deducing how a small variability of the input parameters influences the output as discussed by Martinelli et al. [2010]. As opposed to previously mentioned methods, this approach is applicable only for small uncertainties, due to the local nature of the Taylor expansion approximation used. As in the case of the stochastic Galerkin method, this approach is not well suited for non-differentiable functions due to the Taylor expansion approximation. This approach is intrusive as it requires evaluation of derivatives with the automatic differentiation [Griewank, 1989] strategy for instance.

Numerical strategies to evaluate the computational model

For the remainder of this work, we restrict ourselves to the parametric computations methods which only require to evaluate the model f at N instances of the input variables $\boldsymbol{p}^{(1)}, \ldots, \boldsymbol{p}^{(N)}$ as illustrated in Fig. 1.4. These points can be random samples coming from a Monte Carlo strategy or integration points from a multidimensional quadrature rule in the case of the NISP strategy for instance. The largest CPU cost of these methods is the N evaluations of the computational model f. In this work, we investigate strategies to accelerate these computations without reducing the number of points N.

An interesting property of those parametric computation methods is the fact that the N evaluations of the model f are independent from each other implying that they can be evaluated in any order or simultaneously in parallel without modifying the results. The easiest way to implement a parallel approach to evaluate these N instances is to run each model evaluation independently from each other, using all the available nodes and cores of the computational system used. This approach is usually called the *embarrassingly parallel* approach and illustrated in Fig. 1.5. The drawbacks of this approach are that it requires a large amount of computational resources and the CPU cost can remain high.

A strategy has been proposed to improve machine utilization by performing more model evaluations concurrently using a single program-multiple data (SPMD) parallel

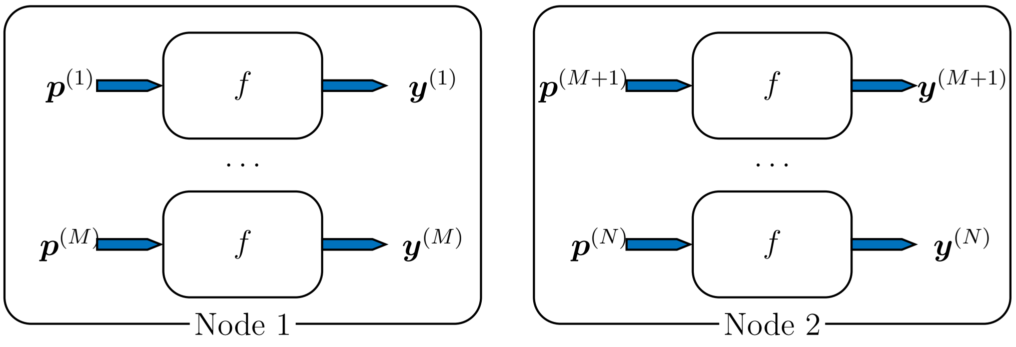


Figure 1.5: Illustration of an embarrassingly parallel approach for the N evaluations of the computational model f on two compute nodes of a supercomputer.

programming model. This has been studied in UQ Pipeline [Dahlgren et al., 2015] based on CRAM [Gyllenhaal et al., 2014]. CRAM is a tool to pack a large number of MPI-based simulations into a single job submitted to an HPC cluster. CRAM creates an MPI sub-communicator per simulation and associates it to the corresponding simulation. This approach reduces the memory pressure on the front-end node compared to submitting each simulation as independent jobs and, therefore, increases the total number of concurrent model evaluations. There are alternatives to UQ Pipeline which relies on the same principles such as CONDOR [Foster et al., 2017] and EMEWS [Ozik et al., 2016] which rely on the Swift parallel scripting language [Wilde et al., 2011]. Moreover, the Dakota library [Adams et al., 2019] provides this type of scheduling strategy, among others, for which only one instance of Dakota is loaded on all the processors used. Such a strategy has been studied in the context of machine learning to produce a massive physics-based dataset using MERLIN [Peterson et al., 2019].

Hadjidoukas et al. [2015] have used the concept of task-based parallelism to accelerate the evaluation of the model on massively parallel and hybrid computing architectures using the TORC library [Hadjidoukas et al., 2012], a task-parallel library for heterogeneous cluster computing platforms.

Recently, a strategy has been proposed to minimize the idle CPU time of the embarrassingly parallel approach using prediction of the wall-clock time of the model evaluations to optimize their scheduling [Künzner et al., 2019].

All the previously discussed strategies to evaluate the N instances are nonintrusive as they do not require modification of the source code and can be implemented as scripts which wrap the model evaluations. Phipps et al. [2017] proposed an alternative to evaluate the instances of the computational model: the embedded ensemble propagation method. Its main motivation is to reduce the total wall-clock time required to evaluate the N instances of the function f. Embedded ensemble propagation relies on the partition of the set of evaluation points $\boldsymbol{p}^{(1)}, \dots, \boldsymbol{p}^{(N)}$ into subsets, called ensembles, that share common characteristics and on the overloading of the solver source code to evaluate one ensemble at the time. This approach can be used with the other above-mentioned strategies or on its own. This embedded ensemble propagation method is introduced in the following section and impacts the solvers as discussed extensively in this thesis.

1.1.3 Embedded ensemble propagation

Phipps et al. [2017] proposed an embedded method for uncertainty quantification called *embedded ensemble propagation* to improve the so-called *throughput*, defined as the ratio of the number of floating point operations and the wall-clock time for evaluating the model at the sample values. Instead of propagating one sample at a time through the high-fidelity model, ensembles of s samples are propagated all at once as illustrated in Fig. 1.6.

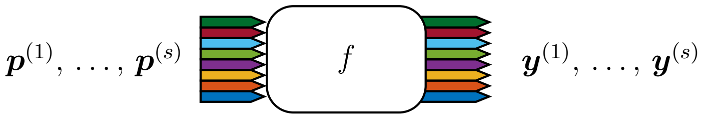

$\boldsymbol{p}^{(1)}, \ldots, \boldsymbol{p}^{(s)}$
f
$\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(s)}$

Figure 1.6: Illustration of embedded ensemble propagation with ensemble of size $s = 8$ where different colors represent different samples propagated altogether through the code.

Embedded ensemble propagation is an alternative to the embarrassingly parallel approach. Although being more complex to use as being embedded, this strategy has advantages compared to the embarrassingly parallel approach such as opportunities for vectorized instructions by looping over the samples, improved memory access patterns, reduced memory usage by computing and storing only once the data which do not vary from one sample to another (such as the mesh if all the samples share the same mesh), and fewer MPI communications. However, the strategy comes with some challenges such as *ensemble divergence*: different samples of a given ensemble may follow different branches of the code. Ensemble divergence can be of two types: control-flow divergence and function-call divergence. *Control-flow divergence* means that the samples within an ensemble can take different branches in an if-then-else condition (*if-then-else divergence*) or may require different numbers of iterations of a loop (*loop divergence*). *Function-call divergence* is observed when a function that needs to be evaluated for all the samples of an ensemble might not be implemented to support ensemble propagation, in this case, the function has to be evaluated sample-wise leading to reduced performance. Those advantages and challenges are discussed in more details in Chapter 2.

1.1.4 C++ template mechanism

The work presented in this thesis is heavily based on the C++ programming language and on its template mechanism. The C++ template mechanism is one of the abstraction mechanisms of C++ [Stroustrup, 2000]. Templates allow the programmer to define functions or classes with values or types as parameters. They allow one to define functions and classes once and allow their use with more than one data type.

As an example, let us assume that we want to define a vector type which can be used with single and double precision and which is able to compute inner products. Instead of defining two classes, one for each precision, we can implement a templated class as done in Listing 1.1 where the class `Vector` has two template parameters: `typename T`, the type of data that can be stored in the vector and `int n`, the number of entries that can be stored. Those parameters are set using `<>`; for example, a vector that can store 100 entries in singe precision is declared as `Vector<float, 100>`. We use this example as it is intrinsically linked to the solvers introduced later in this introduction.

```
template <typename T, int n>
class Vector
{
    T data[n];

public:
    T dot(const Vector<T, n> &v) const
    {
        T tmp = 0.;
        for (int i = 0; i < n; ++i)
            tmp += data[i] * v.data[i];
        return tmp;
    }
};
```

Listing 1.1: Templated vector class of type `T` and fixed size `n` (The code has been shortened for clarity reasons).

With the definition of Listing 1.1, it is possible to use the new class as illustrated in Listing 1.2 where `inner_1` is a `float` and `inner_2` a `double`.

```
Vector<float, 100> vec_1, vec_2; // Initialize vec_1 and vec_2 ...
float inner_1 = vec_1.dot(vec_2);

Vector<double, 100> vec_3, vec_4; // Initialize vec_3 and vec_4 ...
double inner_2 = vec_3.dot(vec_4);
```

Listing 1.2: Use of the templated vector class defined in Listing 1.1 with `float` and `double`.

The basic idea of embedded ensemble propagation which is studied in this thesis is the definition of a new C++ type called *ensemble type*. This type can then be used directly in classes such as our vector class example as the type of the stored data `T`. To be able to use this ensemble type in our vector class example, we have to define either an `operator+` and an `operator*` or a `dot` function for the ensemble type.

1.1.5 Trilinos

Trilinos [Heroux et al., 2005] is a C++ software component library developed by Sandia National Laboratories with the aim of breaking down PDE solvers into common building blocks, optimizing those blocks, and constructing optimized computational models based on those blocks. The library is made of several packages which are dedicated to specific tasks. We briefly review some of them here to clarify the original contributions of this thesis later in this introduction. Those packages have dependencies and are related to each other: for instance, the parametric computation packages rely on the solver packages which rely on the linear algebra packages.

The first key package for this work is the Tpetra package [Baker and Heroux, 2012] which implements templated linear algebra structures such as sparse matrices and vectors. Those data structures can be distributed over compute nodes which can communicate via MPI and are templated on the data type of the entries of the vectors and matrices. Tpetra provides member functions which allow one to compute linear algebra operations such as applying a sparse matrix to a vector. If the matrix and the vector are distributed over more than one compute node, Tpetra will take care of the distributed parallelism calling MPI to access the non-local data transparently. Although Tpetra provides distributed parallelism, Tpetra does not provide directly on-node parallelism, i.e. shared-memory parallelism. Tpetra relies on Kokkos and KokkosKernels for the on-node parallelism.

The Kokkos package [Edwards et al., 2012, 2014] provides templated array types, called *Kokkos views*, to store templated tensors of up to 8 dimensions, with the possibility to specify the memory layout of the entries of the tensor. Moreover, Kokkos provides an abstract programming model for threaded loops. This programming model can be seen as a wrapper around threading technologies such as CUDA on GPUs or OpenMP on CPUs with the aim of portability.

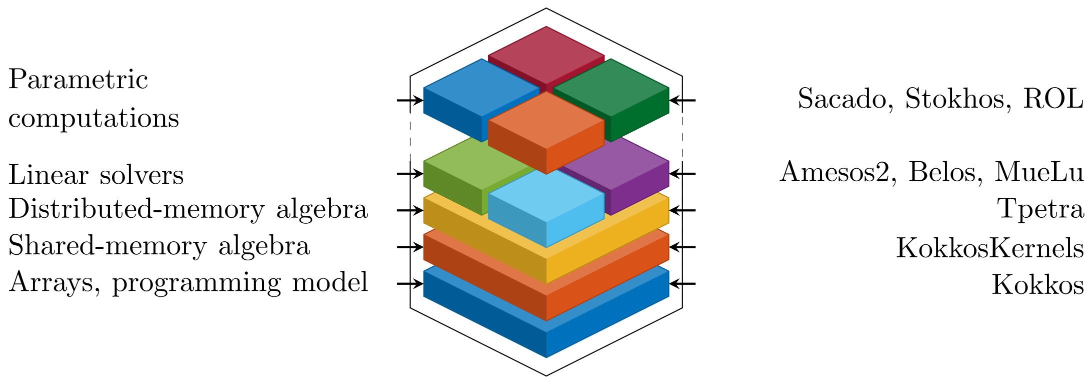


Figure 1.7: Illustration of some of the packages of the Tpetra solver stack.

On top of Kokkos, the KokkosKernels package [Deveci et al., 2016b] provides a templated interface to optimized implementations of so-called *kernel functions*: low level threaded functions for on-node parallelism linear algebra such as BLAS functions [Blackford et al., 2002]. For example, KokkosKernels provides interfaces to the Intel Math Kernel Libraries (MKL) for standard data types such as `double`. When no such optimized implementations are available for a given data type on a given computational architecture, KokkosKernels provides a default implementation of the kernel functions. This package is recent, not all the BLAS functions have a default implementation, and the package was not existing at the beginning of this thesis.

Trilinos provides a full solver stack based on the Tpetra package, called, in the remainder of this thesis, the Tpetra solver stack. This solver stack includes linear solvers, non-linear solvers, time integration schemes, mesh structures, and parametric computations among others and is partially illustrated in Fig. 1.7. In the presented work, four packages of the Tpetra solver stack are particularly used: Amesos2 [Bavier et al., 2012], Belos [Bavier et al., 2012], MueLu [Prokopenko et al., 2014], and Stokhos [Phipps, 2015]. Amesos2 is the package related to direct linear solvers, Belos is the package related to Krylov-based methods including Conjugate Gradient method (CG) and Generalized Minimal Residual method (GMRES), MueLu is the package related to multigrid solvers including multigrid preconditioners, and Stokhos is the package related to the embedded uncertainty quantification: the stochastic Galerkin and embedded ensemble propagation methods as discussed in section 1.1.6.

1.1.6 Embedded ensemble propagation in Trilinos

Stokhos is the embedded uncertainty quantification package of the Trilinos library in which Phipps [2015] provides both an implementation of the intrusive stochastic Galerkin method based on polynomial chaos discretizations [Ghanem et al., 2017; Phipps et al., 2017] and an implementation of embedded ensemble propagation discussed in section 1.1.3.

Those implementations are part of the larger research effort about embedded analysis capabilities as discussed by Pawlowski et al. [2012a,b]. This template-based generic programming approach has been used in Trilinos to embed augmented analysis such as automatic differentiation [Phipps and Pawlowski, 2012] in the Sacado package and the stochastic Galerkin and embedded ensemble propagation methods in the Stokhos package.

Embedded ensemble propagation was implemented by Phipps et al. [2017] in Stokhos using a template-based generic-programming approach and the definition of a C++ ensemble type, a new aggregate data type [Edwards et al., 2014] which stores the data for each sample as illustrated in Listing 1.3 and discussed in detail in section 2.4.

```cpp
template <int s>
class Ensemble
{
    // Private member data to store the values for the s samples
    double val[s];

public:
    // Member function to access the value of sample e
    double &operator[](int e) { return val[e]; }
    // Constructor which sets the value of every samples to v
    Ensemble(const double &v)
    {
        for (int e = 0; e < s; ++e)
            val[e] = v;
    }
    // Simple assignment operator which copies the value sample-wise
    Ensemble &operator=(const Ensemble &a)
    {
        for (int e = 0; e < s; ++e)
            val[e] = a.val[e];
        return *this;
    }
    // Addition assignment operator which adds the value sample-wise
    Ensemble &operator+=(const Ensemble &a)
    {
        for (int e = 0; e < s; ++e)
            val[e] += a.val[e];
        return *this;
    }
    // ...
};

// Multiplication operator which multiplies the value sample-wise
template <int s>
Ensemble<s>
operator*(const Ensemble<s> &a, const Ensemble<s> &b)
{
    Ensemble<s> c;
    for (int e = 0; e < s; ++e)
        c.val[e] = a.val[e] * b.val[e];
    return c;
}
```

Listing 1.3: Simplified ensemble type definition adapted from Phipps et al. [2017]. Only the portions relevant to the inner product routine are included, and complications such as expression templates are excluded.

The idea of the template-based generic-programming approach used to implement embedded ensemble propagation in Trilinos is to adapt existing codes using either a scalar type or an ensemble type as the data type. In other words, this new C++ ensemble type can be used as the template parameter of classes and functions such as our vector example defined in section 1.1.4. In Listing 1.4, we illustrate the use of ensemble propagation to evaluate the inner product of 8 vectors.

If we consider the previous example of the inner product, the inner product is computed sample-wise and no reduction is performed: the returned ensemble `inner_1` stores $s = 8$ values which correspond to the result of the inner product of the corresponding sample.

```
Vector<Ensemble<8>, 100> vec_1, vec_2;
Ensemble<8> inner_1 = vec_1.dot(vec_2); // Returns an ensemble!
```

Listing 1.4: Example of use of the ensemble type as a template argument of the vector class defined in Listing 1.1. Thanks to the definition of the multiplication of 2 ensembles (Listing 1.3 line 36), the templated dot product of Listing 1.1 which was used with float and double types is still valid and returns an ensemble (although most algorithms which use this dot product expect a scalar!).

Embedded ensemble propagation can be used in any parts of a solver including the matrix assembly processes or a non-linear iterative process. However, in this paragraph we will zoom on the iterative methods for linear problems in order to highlight some of the challenges. As discussed in [Phipps et al., 2017], Krylov-based methods [Saad, 2003], being projection methods, rely on the results of inner products and norm calculations. One of the challenges with such methods with embedded ensemble propagation is the notion of inner product of vectors of ensemble such as `vec_1` and `vec_2` of Listing 1.4. There are currently two approaches to use embedded ensemble propagation in Krylov-based methods. Phipps et al. [2017] introduced the so-called *ensemble reduction*: a strategy that defines an inner product that couples the samples together. This inner product computes the sum of all the sample-wise inner products of an ensemble and can be used to deduce norms and projections. Ensemble reduction has the advantages of being able to use optimized implementations of BLAS functions, for example, the inner product of two vectors of ensembles of size n and ensemble size s can be implemented as a classical inner product of 2 vectors of size $n \times s$. Moreover, ensemble reduction fully removes the occurrences of ensemble divergence inside Krylov-based methods as inner products and norms return a `double`: this removes the loop-divergence and if-then-else divergence when the followed branch depends on values of norms or inner products. D'Elia et al. [2020] introduced a second approach based on an inner product without ensemble reduction, as the one of Listing 1.4. The initial aim of this second approach is to easily access the norm of the residual of each sample individually to monitor their convergence. That information has been used in D'Elia et al. [2020] to group together samples that require similar number of iterations of the Krylov-based method. The use of this inner product without ensemble reduction raises challenges of ensemble divergence inside Krylov-based methods, which were solved in [D'Elia et al., 2020] for the particular case of CG [Saad, 2003].

The choice of using ensemble reduction is made at the compilation time as it impacts the return type of inner products and norms, impacting kernel functions and their optimization.

We observe in Listing 1.4 that the inner product returns an ensemble type as Listing 1.1 defined the return type of `dot` as a `T`. As a consequence, Listing 1.4 illustrates the use of embedded ensemble propagation without ensemble reduction. If we want to use ensemble reduction with the vector class of Listing 1.1, we have to implement a template specialization of the `dot` function to return the sum of all the entries as illustrated in Listing 1.5.

```
template <int s, int n>
class Vector<Ensemble<s>, n>
{
    Ensemble<s> data[n];

public:
    // Member function to compute the reduced inner product.
    // The function returns a double instead of an Ensemble<s>.
    double dot(const Vector<Ensemble<s>, n> &v2)
    {
        double tmp = 0.;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < s; ++j)
                tmp += data[i][j] * v2.data[i][j];
        return tmp;
    }
};
```

Listing 1.5: Simplified example of a partial template specialization of the vector class example of Listing 1.1 to use ensemble reduction. The main difference is the return type. Now the inner product returns the sum (the reduction) of the s inner products.

Phipps et al. [2017] have done a huge implementation work to define new specializations for the ensemble type such that the Tpetra solver stack, including the packages Amesos2, Belos, and MueLu, can be compiled with it and used efficiently. Embedded ensemble propagation being embedded impacted all the packages used including the low-level packages such as Kokkos and KokkosKernels. This implementation work includes partial template specialization of classes as illustrated in Listing 1.5, template specialization of functions, testing, and performance optimization of kernel functions for ensemble type with and without ensemble reduction.

At the beginning of this work, the ensemble type class was fully implemented and was compiled with and without ensemble reduction. Moreover all the fundamental work needed to use embedded ensemble propagation in the Tpetra solver stack was done and maintained using ensemble reduction. The nightly build tests of Trilinos were testing ensemble reduction implementation. The Tpetra solver stack was not automatically tested without ensemble reduction. This work addresses issues occurring in the Tpetra solver stack without ensemble reduction.

1.1.7 New computational architectures

Current computational architectures based on CPUs rely on three nested parallelism levels: the distributed-memory parallelism, the shared-memory parallelism, and the vector instructions. In this work we consider computational architectures which have more than one CPU per compute node.

The distributed-memory parallelism consists in distributing the computational work among different CPUs which do not necessarily share a common memory but can communicate together by sending and receiving messages, typically using MPI. The shared-memory parallelism consists in distributing the computational work among different threads of a same compute node which have access to a common main memory. This is typically done using a threading library such as OpenMP, Pthreads, or Intel Threading Building Blocks (TBB). The last level, the vector instructions, corresponds to the Single Instruction Multiple Data (SIMD) parallelism. One thread is able to apply one operation to a set of data altogether. The vector length is the number of operations that can be treated together by a vector instruction. For instance, on Intel architectures which support AVX-512, 512-bit Advanced Vector Extensions SIMD instructions for x86 instruction set architecture, one thread can apply a vector instruction to compute an operation with 8 `double` or 16 `float` altogether. Older Intel SIMD extensions such as AVX2 could have been considered too but we observed that they reach smaller performance than AVX-512 instructions on the examples considered. Moreover, AVX-512 includes more instructions than AVX2 such as masked instructions discussed and used in this thesis.

Emerging computational architectures, such as Intel Cascade Lake, come with more and more vector capabilities; the number of available Intel Intrinsics functions, the C++ functions that the compiler replaces with the proper assembly vector instructions, grows at each new AVX release. Therefore, the efficient usage of fine-grained SIMD parallelism is important to efficiently use the supercomputers of tomorrow. However, it is not obvious to implement all PDE solvers to efficiently use those SIMD instructions as, for instance, FEM solvers typically rely on sparse data structures that are unknown at the compilation time. There are two approaches to use the SIMD instructions in C++, either using the intrinsics such as the Intel Intrinsics or relying on the autovectorization done by the compiler: the compiler can vectorize loops specified by the user providing that those loops fulfill several criteria described later in this thesis.

Moreover, current computational architectures based on CPUs rely on memory hierarchy to accelerate memory access. The memory hierarchy separates the storage into different levels to reduce the response time accessing data. Typically, the storage is separated in high-speed but small capacity cache levels (such as L1, L2, and L3 cache) used to store frequently accessed data and in low-speed but high capacity main memory used to store all the required data.

Three other concepts of memory architecture are used in this thesis: the non-uniform memory access (NUMA) regions, the Translation Lookaside Buffer (TLB), and the unit-stride loads. Those concepts are described in more details in [Hennessy and Patterson, 2011].

In the case of multiprocessing, NUMA is a memory design where the response time of the main memory depends on the memory location relative to the processor: a processor accesses faster the memory associated to itself (its own NUMA region) than the memory associated to another processor. With that design in mind, it is important to allocate memory in the correct NUMA region and, if possible, avoid accessing this allocated memory with threads of another processor.

The TLB is a memory cache which stores the translation of the virtual memory address to corresponding physical memory address for the recent memory access. Taking into account the size of the TLB is important to reuse previously translated addresses while they are still in the TLB to avoid losing time in the translation process.

To hide latency of memory access, programs must access data that are stored contiguously in the memory, i.e. data which are stored with consecutive physical memory addresses. Such data loads are called unit-stride loads. The unit-stride load is an important concept which drove the design of the implementations presented in this work.

In this thesis, we discuss the influence of these details of the computer architecture of modern computers when using embedded ensemble propagation. We only briefly discuss the distributed-memory parallelism; we mainly address shared-memory parallelism, vectorization, and cache usage.

The performances presented in this thesis have been measured on the cluster called *Blake* of Sandia National Laboratories described in Appendix A.

1.1.8 Non-symmetric or indefinite matrices

The thermomechanical model problem of the mirror presented in section 1.1.1 has non-symmetric matrices due the coupling between the temperature field and the displacement fields. This corresponds to the first of the two classes of problems considered in this work. The second class of problems is contact problems in saddle-point formulation discretized with the Mortar finite element method which have indefinite matrices. Both of those two classes are described mathematically in Chapter 5.

In this work, we focused on the Generalized Minimal Residual method (GMRES) [Saad and Schultz, 1986; Saad, 2003] for the solution of linear problems with non-symmetric or indefinite matrices.

The typical implementation of GMRES [Saad and Schultz, 1986; Saad, 2003] relies on five steps:

1. the construction of an orthonormal basis, whose vectors are called Arnoldi vectors, of the Krylov subspace using a classical Gram-Schmidt approach and the construction of an upper Hessenberg matrix which stores the inner products of the vectors before orthogonalization,
2. if the last Arnoldi vector has a zero norm, GMRES encounters a lucky breakdown and the solution is exact,
3. the construction of a triangular system using the norm of the initial residual and applying Givens rotations on the upper Hessenberg matrix,
4. the solution of the triangular system to compute the coefficients of the approximation of the solution in the orthonormal basis,
5. the computation of the norm of the residual and the convergence test.

The application of ensemble propagation to the GMRES method introduces three types of ensemble divergence which will be addressed in the following chapter of this thesis:

1. an Arnoldi vector may require a normalization for certain samples but not for other samples for which GMRES encounters a lucky breakdown, example of if-then-else divergence,
2. different samples may require different numbers of iterations to converge and hence to exit the for loop in GMRES after different trip counts, example of loop divergence,
3. called BLAS functions, such as GEMV for the dense matrix-vector operations, may not support ensemble-typed inputs, example of function call divergence.

1.2 Contributions

The contributions of this thesis can be grouped into three axes: ensemble divergence, the effect of ensemble reduction, and ensemble propagation on non-academic problems.

- Ensemble divergence:
  - The main contribution of this thesis related to ensemble divergence consists in the implementation and the evaluation of the performance of a GEMV (dense matrix-vector product) with ensemble types, a key ingredient of the orthogonalization process inside ensemble GMRES without ensemble reduction. Not using ensemble reduction prevents us from using optimized implementations of BLAS. Moreover, the existing templated kernel available in Trilinos does not perform optimally not reaching the theoretical maximal performance. Since it is important to have optimal performance of this kernel to have results that can be compared with optimized kernels for standard data type, we implemented a new dense matrix-vector product using the Kokkos programming model with better performance than the default dense matrix-vector product of KokkosKernels. This new implementation uses a cache blocking strategy usually used in dense matrix-matrix products [Smith et al., 2014].

    The originalities of this contribution are the efficient implementation of the GEMV with ensemble types which reach theoretical maximal performance, its comparison with the default implementation of KokkosKernels, the study of the impact of the ensemble size on the implementation, and the comparison with the performance of the GEMV with ensemble reduction.
  - The other contributions are related to control-flow divergence. They aim at extending the work of D'Elia et al. [2020] to GMRES. Ensemble GMRES with ensemble reduction was already working before the work presented in this thesis; it was possible to compile the templated GMRES of Belos with the ensemble type using ensemble reduction and ensemble GMRES with ensemble reduction was tested. It was not the case for ensemble GMRES without ensemble reduction: none of the if-then-else divergences and the loop divergences were managed and the GEMV used was a default templated implementation of KokkosKernels which does not reach the theoretical maximal performance of the CPU. The convergence test implemented by D'Elia et al. [2020] for the CG cannot be directly used for ensemble GMRES without ensemble reduction as the templated GMRES of Belos relies on implicit norm computation deduced during the triangularization of the upper Hessenberg matrix.
    - A new strategy has been developed to deal with ensemble divergence using masks to ease readability and maintenance of code with some inspiration from the implementation introduced in Kretz and Lindenstruth [2012] for SIMD data type.
    - Status tests have been implemented to test the convergence of all the samples and stop the iterative process when they have all converged for ensemble GMRES without ensemble reduction based on the work of D'Elia et al. [2020] for the CG but using masks.
    - Divisions by zero have been prevented during the normalization of vectors in ensemble GMRES without ensemble reduction based on the work of D'Elia et al. [2020] for the CG but using masks.

    * A strategy to manage the lucky breakdown in ensemble GMRES without ensemble reduction has been developed and implemented using masks.
    * A strategy to use embedded ensemble propagation with contact problems where contact can have different status, locally active or inactive, for different samples of an ensemble has been developed and implemented using masks.

- Effect of ensemble reduction:
  - The impact of ensemble reduction on the convergence of ensemble GMRES has been studied to highlight that ensemble reduction leads to deteriorated convergence compared to ensemble GMRES without ensemble reduction. This influence on the convergence impacts the speed-up of ensemble GMRES.

    The originality of this contribution is the highlighting of the impacts of ensemble reduction on the convergence of ensemble GMRES both using convergence theory and numerical examples.
  - The CPU costs per iteration of ensemble GMRES have been measured to illustrate that they are similar independently of the use of ensemble reduction. This implies that the use of ensemble reduction impacts the total CPU cost mainly by impacting the number of iterations to converge.
  - The performance of ensemble GMRES has been evaluated on academic problems, including one contact problem, which have non-symmetric or indefinite matrices with and without ensemble reduction. These evaluations illustrate that the speed-up of ensemble GMRES without ensemble reduction is better than the speed-up of ensemble GMRES with ensemble reduction due to the negative effect of the reduction on the convergence.
- Ensemble propagation on non-academic problems:
  - Embedded ensemble propagation has been used for the first time on a non-academic problem to accelerate an uncertainty quantification study of a model problem relevant for the design of the ITER mirror.
  - An open-source multiphysics finite element simulation code which supports embedded ensemble propagation has been provided.

1.3 Publications

The presented work on ensemble GMRES without ensemble reduction and the comparison with ensemble GMRES with reduction has been published in [Liegeois et al., 2020]. The paper includes the Chapters 2 and 3 and parts of the Chapters 4 and 7: the sections 4.2.1, 4.2.4, 4.3, and 7.3.

Moreover, the ITER mirror model discussed in Chapter 8 has been used in [Mertens et al., 2019], a paper which I am one of the coauthors.

1.4 Outline of the thesis

The remainder of this thesis is organized as follows:

In Chapter 2, we introduce embedded ensemble propagation as a way to improve the throughput and the efficiency of nonintrusive parametric computation methods. Moreover, the advantages of the approach are listed and illustrated in the case of a sparse matrix-vector product. Possible causes of ensemble divergence are defined and classified. Parametric linear systems are introduced using a tensor notation. Finally, the use of a template-based generic-programming approach is discussed as a way to embed ensemble propagation into existing codes.

In Chapter 3, we discuss the use of ensemble propagation in the GMRES linear solver. To do so, we start by a review of GMRES with and without preconditioners in order to highlight the list of potential occurrences of ensemble divergence. Two approaches are then discussed. The first one uses ensemble reduction whereas the second one needs to take care of each divergence explicitly. Finally, a theoretical discussion on the impact of ensemble reduction on the convergence of ensemble GMRES is given at the end of the chapter.

In Chapter 4, we implement strategies to take care of each case of divergence of ensemble GMRES without ensemble reduction. In particular, we discuss the use of ensemble propagation in the dense matrix-vector product of the orthogonalization process of GMRES and how to implement it efficiently. We compare the performance of this proposed dense matrix-vector product with the performance of the corresponding dense matrix-vector product with ensemble reduction using optimized BLAS and show that they are similar. Finally, the mask strategy is discussed as a way to manage ensemble divergence in GMRES.

Chapter 5 is devoted to the description of thermomechanical and contact problems discretized with the mortar finite element method. Those classes of problems will be used in Chapter 7 as examples of problems with non-symmetric or indefinite matrices that are solved with GMRES.

In Chapter 6, we discuss the implemented code called *Katoptron*[1], mirror in Greek, used to analyze the performance of ensemble GMRES.

Chapter 7 illustrates the efficiency of ensemble GMRES and compares ensemble GMRES with and without ensemble reduction on four examples: a first example that illustrates the effect of the coupling on the convergence, a second example that illustrates speed-up of ensemble propagation as a function of the problem size, and two examples that illustrate speed-up on mesh-tying and contact problems.

In Chapter 8, we apply the previously discussed work as a way to improve the efficiency of an industrial model problem: an uncertainty quantification study to investigate the robustness assessment of the first mirror to uncertainty in the heat conduction coefficient of spacers.

Finally, the conclusion of Chapter 9 summarizes the contributions and results of this work and gives some comments on possible future work.

[1]`https://gitlab.uliege.be/am-dept/waves`

Chapter 2

Ensemble propagation

In this first chapter, we introduce embedded ensemble propagation as a way to improve the throughput and the efficiency of parametric computation methods.

In section 2.1, we start by defining the speed-up, the measure of the gained performance due to ensemble propagation and by listing the advantages and challenges of embedded ensemble propagation. In particular, we define and classify the occurrences of ensemble divergence, the main challenge addressed in this thesis.

After that, in section 2.2, we revisit parametric linear systems with embedded ensemble propagation as introduced in Phipps et al. [2017] but with an original formalism based on tensors to ease the discussion of the following chapters.

Section 2.3 relies on both of the previous sections and illustrates the usage of the tensor formalism to discuss the sparse matrix-vector product. Moreover, the example is used to illustrate the advantages of embedded ensemble propagation over a simpler approach such as embarrassingly parallel computations.

Section 2.4 is dedicated to the discussion of the implementation of ensemble propagation using C++ templates in Trilinos as a way to embed ensemble propagation into existing code.

2.1 Advantages and challenges

Embedded ensemble propagation consists in propagating subsets of samples, called ensembles, through a high-fidelity model instead of propagating one sample at a time. Ensemble propagation can reduce the total wall-clock time of evaluating samples of the high-fidelity model and therefore the wall-clock time of parametric computations.

In this work, it is important to say that we compare the sequential runs of parallelized sample runs with sequential runs of parallelized ensemble runs on the same architecture with the same amount of resources. From that, we can directly extrapolate these results to cases where several runs are launched at the same time using a distributed approach on different compute nodes as those runs are independent. It is, however, less straightforward to extrapolate to cases where different computations are launched at the same time on a same node with different threads as physical bounds such as the amount of memory and the memory bandwidth play a role. This is discussed in Appendix B. The reduction of the total wall-clock time is measured by the notion of speed-up defined in [Phipps et al., 2017] as:

$$S = \frac{\sum_{\ell=1}^{s} T^{(\ell)}}{T^{(e)}}, \tag{2.1}$$

where, for a given ensemble, s is the number of samples in the ensemble, also called the ensemble size, $T^{(\ell)}$ is the wall-clock time of evaluating the sample ℓ alone using all the available cores, and $T^{(e)}$ the wall-clock time of evaluating the ensemble using the same number of cores. Whenever the speed-up is greater than one, the use of ensemble propagation reduces the total wall-clock time of evaluating all the samples. For example, assuming that we have $s = 8$ and 1 CPU with 4 cores, $T^{(e)}$ is the wall-clock time of evaluating the 8 samples together using the 4 cores and $T^{(1)}$ is the wall-clock time of evaluating the first sample using the 4 cores. In this thesis, except in Appendix B, we do not consider the case where different cores can be associated to the computation of different samples.

As identified in [Phipps et al., 2017], ensemble propagation acts on the speed-up S in four ways:

(i) Samples of the same ensemble share common data which therefore need to be only computed, stored, and loaded once per ensemble. Typical examples include spatial meshes, graphs of sparse matrices, maps containing the distribution of rows of matrices on different MPI processes, or the Jacobian matrix of a given mesh element.

(ii) Arithmetic operations on ensembles facilitate autovectorization during the compilation.

(iii) Storing the elements of an ensemble with consecutive memory addresses reduces the total number of random memory accesses and therefore improves data prefetching and cache line usage.

(iv) Message passing costs are reduced by sending fewer but larger messages and therefore reducing the total MPI latency.

The impact of those improvements on the speed-up depends on the occurrence of *ensemble divergence* [Phipps et al., 2017]: for a given ensemble, different samples may follow different branches of the code. Ensemble divergence can be of two types:

(i) Control-flow divergence [Coutinho et al., 2011]: the samples within an ensemble can take different branches in an if-then-else condition (if-then-else divergence) or may require different numbers of iterations of a loop (loop divergence).

(ii) Function-call divergence: a function that needs to be evaluated for all the samples of a given ensemble might not be implemented to support ensemble propagation, in this case, the function has to be evaluated sample-wise potentially leading to reduced performance.

Ensemble divergence depends both on how the samples are grouped in ensembles and on the way the algorithm is implemented. Grouping methods have been proposed in [D'Elia et al., 2020] and [D'Elia et al., 2018] to reduce the loop divergence. Those strategies have been illustrated on the Conjugate Gradient (CG) method and improved the speed-up of ensemble propagation. Those grouping strategies are not the focus of this work.

A second challenge of ensemble propagation is the increased memory usage; as we need to store data for all the samples of the ensemble, more memory is needed compared to one sample alone. Depending on ensemble divergence, and especially the loop divergence, the amount of memory used with ensemble propagation can be smaller or larger than the amount of memory used by s samples alone. If there is no ensemble divergence at all, the amount of memory used with ensemble propagation is necessarily smaller as common data such as matrix graphs are stored only once per ensemble. If there is some loop divergence such as in an iterative solver and if each iteration of the loop requires some memory allocation, the memory saved by storing only once common data may not compensate the required extra memory.

2.2 Parametric linear systems

In this section we set up a parametric linear system, which serves in the next chapters as a concrete framework to describe our contributions to iterative linear solvers with embedded ensemble propagation. We will do so with the help of a new tensor-based formalism. By revisiting the description of the impact of ensemble propagation on the sparse matrix-vector product [Phipps et al., 2017], we will also recall key aspects of ensemble propagation and describe how they can be articulated in this new tensor-based formalism.

In this context, we are interested in the solution of a linear system, a ubiquitous computational operation in the solution of a computational mechanics model, for a subset of samples of the parameters together. That is, in the solution of a parametric linear system of the form

$$\boldsymbol{A}\big(\boldsymbol{\xi}^{(\ell)}\big)\, \boldsymbol{x}\big(\boldsymbol{\xi}^{(\ell)}\big) = \boldsymbol{b}\big(\boldsymbol{\xi}^{(\ell)}\big) \quad \text{for all } \ell = 1, \ldots, s; \tag{2.2}$$

here, we denote by $\boldsymbol{\xi}^{(1)}, \ldots, \boldsymbol{\xi}^{(s)}$ the subset of samples of the parameters, with s the ensemble size and by the n-dimensional square matrices $\boldsymbol{A}(\boldsymbol{\xi}^{(1)})$, …, $\boldsymbol{A}(\boldsymbol{\xi}^{(s)})$ and vectors $\boldsymbol{b}(\boldsymbol{\xi}^{(1)})$, …, $\boldsymbol{b}(\boldsymbol{\xi}^{(s)})$ and $\boldsymbol{x}(\boldsymbol{\xi}^{(1)}), \ldots, \boldsymbol{x}(\boldsymbol{\xi}^{(s)})$ the system matrices and right-hand sides and solution vectors for the samples $\boldsymbol{\xi}^{(1)}$, …, $\boldsymbol{\xi}^{(s)}$, respectively. Thus, n denotes the number of physical degrees of freedom per sample. The matrices $\boldsymbol{A}(\boldsymbol{\xi}^{(1)})$, …, $\boldsymbol{A}(\boldsymbol{\xi}^{(s)})$ are nonsingular sparse matrices resulting from the discretization of systems of partial differential equations with a discretization strategy such as the finite element method,

(a) Solution and right-hand-side vectors. (b) System matrices.

Figure 2.1: Ensemble propagation for parametric linear system: illustration of collecting vector and matrix samples in second-order and third-order tensors with 4 samples, respectively. Each color refers to a sample.

the finite volume method, the finite difference method, or the spectral element method. In this work, we restrict ourselves to the finite element method. We are particularly interested in the case in which these matrices are not necessarily symmetric positive definite.

Ensemble propagation was formalized in [Phipps et al., 2017] with a formalism based on the Kronecker product. This formalism involved gathering the n-dimensional square matrices $\boldsymbol{A}(\boldsymbol{\xi}^{(1)})$, …, $\boldsymbol{A}(\boldsymbol{\xi}^{(s)})$ along the diagonal of a block-diagonal ns-dimensional square matrix and concatenating the n-dimensional vectors $\boldsymbol{b}(\boldsymbol{\xi}^{(1)})$, …, $\boldsymbol{b}(\boldsymbol{\xi}^{(s)})$ and $\boldsymbol{x}(\boldsymbol{\xi}^{(1)}), \dots, \boldsymbol{x}(\boldsymbol{\xi}^{(s)})$ into two ns-dimensional vectors. This formalism relied on a notion of commuted Kronecker product to highlight key aspects of ensemble propagation; in particular, the resulting permutations of degrees of freedom were used to highlight the contiguous storage in memory of the sample values per physical degree of freedom, which we mentioned in performance benefit (iii) in the section 2.1. This formalism was well adapted to formalizing ensemble propagation as it was introduced in [Phipps et al., 2017] with reduced inner products that sum inner products across ensembles; however, it lends itself less well to formalizing ensemble propagation as it was more recently set up in [D'Elia et al., 2020] with ensemble inner products without this ensemble reduction; we will expand on the latter approach in this thesis.

To overcome this drawback, we introduce a new formalism based on three-dimensional tensors, by which we mean multidimensional arrays as in the review paper of Kolda and Bader [2009] (not to be confused with tensors from physics and engineering such as stress and strain tensors and tensor fields).

In the remainder of this thesis, we use the system of notation of [Kolda and Bader, 2009], which we will now recall concisely. The number of dimensions of a tensor is called order. Vectors (tensors of order one) are denoted by boldface lowercase letters such as $\boldsymbol{y}$, matrices (tensors of order two) by boldface uppercase letters such as $\boldsymbol{Y}$, and higher-order tensors by boldface Euler script letters such as $\mathscr{Y}$. The i-th entry of $\boldsymbol{y}$ is denoted by y_i, the (i, j)-th entry of $\boldsymbol{Y}$ by y_{ij}, and the (i, j, k)-th entry of third-order tensor $\mathscr{Y}$ by y_{ijk}. A colon is used to indicate all elements of a dimension. A fiber is obtained by fixing every index but one. The j-th column of $\boldsymbol{Y}$ is the fiber denoted by $\boldsymbol{y}_{:j}$, and the i-th row of $\boldsymbol{Y}$ is the fiber denoted by $\boldsymbol{y}_{i:}$. Third-order tensors have column, row, and tube fibers, denoted for $\mathscr{Y}$ by $\boldsymbol{y}_{:jk}$, $\boldsymbol{y}_{i:k}$, and $\boldsymbol{y}_{ij:}$, respectively. Slices are obtained by fixing all but two indices. Third-order tensors have horizontal, lateral, and frontal slices denoted for $\mathscr{Y}$ by $\boldsymbol{Y}_{i::}$, $\boldsymbol{Y}_{:j:}$, and $\boldsymbol{Y}_{::k}$, respectively.

Collecting $\boldsymbol{b}(\boldsymbol{\xi}^{(1)}), \dots, \boldsymbol{b}(\boldsymbol{\xi}^{(s)})$ and $\boldsymbol{x}(\boldsymbol{\xi}^{(1)}), \dots, \boldsymbol{x}(\boldsymbol{\xi}^{(s)})$ into column fibers of two second-

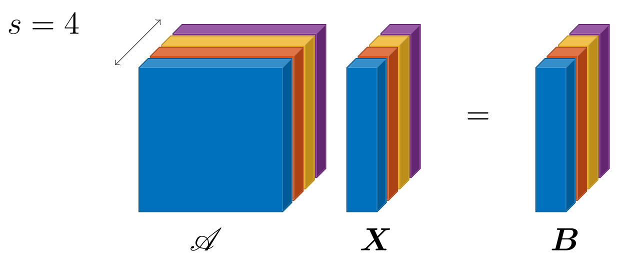


Figure 2.2: Ensemble propagation for parametric linear system: illustration of the parametric linear system in tensor-based formalism with 4 samples.

order tensors $\boldsymbol{B}$ and $\boldsymbol{X}$ (Fig. 2.1a),

$$\boldsymbol{x}_{:\ell} = \boldsymbol{x}\big(\boldsymbol{\xi}^{(\ell)}\big) \text{ for all } \ell = 1, \ldots, s, \qquad \boldsymbol{b}_{:\ell} = \boldsymbol{b}\big(\boldsymbol{\xi}^{(\ell)}\big) \text{ for all } \ell = 1, \ldots, s, \tag{2.3}$$

that is,

$$\boldsymbol{X} = \big[\boldsymbol{x}\big(\boldsymbol{\xi}^{(1)}\big) \quad \ldots \quad \boldsymbol{x}\big(\boldsymbol{\xi}^{(s)}\big)\big], \qquad \boldsymbol{B} = \big[\boldsymbol{b}\big(\boldsymbol{\xi}^{(1)}\big) \quad \ldots \quad \boldsymbol{b}\big(\boldsymbol{\xi}^{(s)}\big)\big], \tag{2.4}$$

and $\boldsymbol{A}(\boldsymbol{\xi}^{(1)})$, ..., $\boldsymbol{A}(\boldsymbol{\xi}^{(s)})$ as frontal slices of a third-order tensor $\mathscr{A}$ (Fig. 2.1b),

$$\boldsymbol{A}_{::\ell} = \boldsymbol{A}\big(\boldsymbol{\xi}^{(\ell)}\big) \quad \text{for all} \quad \ell = 1, \ldots, s, \tag{2.5}$$

we rewrite the parametric linear system in (2.2) as follows (Fig. 2.2):

$$\boldsymbol{A}_{::\ell}\, \boldsymbol{x}_{:\ell} = \boldsymbol{b}_{:\ell} \quad \text{for all} \quad \ell = 1, \ldots, s. \tag{2.6}$$

2.3 Ensemble sparse matrix-vector product

We will now recall key aspects of ensemble propagation and illustrate how they can be articulated within our tensor-based formalism. We will revisit, using the new formalism, the example of Phipps et al. [2017] of computing a sparse matrix-vector product for s samples together (Fig. 2.3):

$$\boldsymbol{z}_{:\ell} = \boldsymbol{A}_{::\ell}\, \boldsymbol{y}_{:\ell} \quad \text{for all} \quad \ell = 1, \ldots, s; \tag{2.7}$$

the matrix-vector product is a ubiquitous kernel in Krylov iterative methods to solve linear systems.

We assume that the matrix samples in the frontal slices $\boldsymbol{A}_{::1}$, ..., $\boldsymbol{A}_{::s}$ are represented using a Compressed Row Storage (CRS) format. Assuming that these frontal slices share a common sparsity pattern, such as a common sparsity pattern inherited from a common mesh, and hence row-offset and column-index arrays, we represent $\mathscr{A}$ with the following data structure:

- a column-index array $\boldsymbol{c}$, a first-order tensor, that collects the column indices of the entries present in the sparsity pattern listed row by row;
- a row offset array $\boldsymbol{r}$, a first-order tensor, that collects the positions of the beginning of each row in $\boldsymbol{c}$; the length of $\boldsymbol{r}$ is $n + 1$, and r_{n+1} is set equal to the number of entries present in the sparsity pattern plus one;

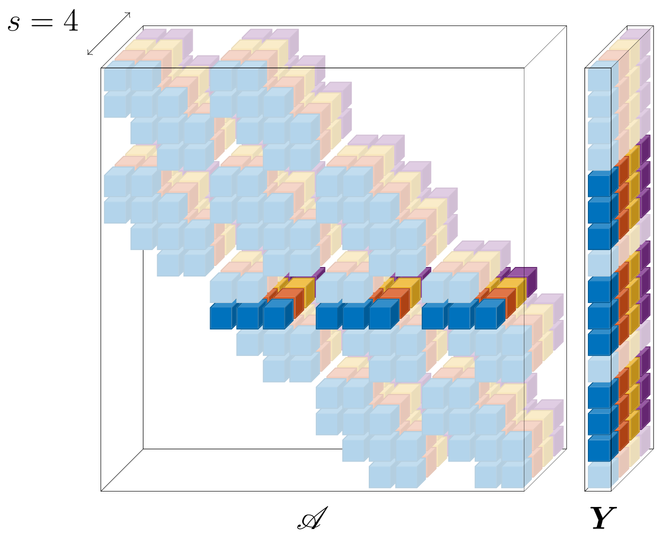


Figure 2.3: Illustration of the sparse matrix-vector product with 4 samples. Each color refers to a sample. Each non-zero element of the third-order tensor $\mathscr{A}$ and second-order tensor $\boldsymbol{Y}$ is represented as a small cube. The highlighted cubes represent the non-zero elements used during the application of a horizontal slice $\boldsymbol{A}_{i::}$ to $\boldsymbol{Y}$.

- a value array $\boldsymbol{D}$, here a second-order tensor, that here gathers in its column fibers $\boldsymbol{d}_{:1}$, ..., $\boldsymbol{d}_{:s}$ the values of the represented entries of the frontal slices $\boldsymbol{A}_{::1}$, ..., $\boldsymbol{A}_{::s}$ listed row by row.

Under the aforementioned assumptions, the column-index array and the offset array are sample-independent, so that they need to be stored in memory and streamed from it only once per ensemble, thus providing an illustration of performance benefit (i) that can be realized with ensemble propagation.

Using this CRS format, the sparse matrix-vector product for s samples together of (2.7) is obtained as follows:

$$z_{k\ell} = \sum_{i=r_k}^{r_{k+1}-1} d_{i\ell}\, y_{c_i\ell} \quad \text{for all} \quad k = 1, \ldots, n, \quad \text{for all} \quad \ell = 1, \ldots, s. \tag{2.8}$$

In an implementation based on (2.8), the entries $z_{k\ell}$ are obtained through a loop over the samples ($\ell = 1, \ldots, s$), a loop over the rows ($k = 1, \ldots, n$), and a loop over the entries present in the sparsity pattern for the given row ($i = r_k, \ldots, r_{k+1} - 1$) performing a multiply-add operation, with the loop over the entries present in the sparsity pattern for the given row ($i = r_k, \ldots, r_{k+1} - 1$) being the innermost loop.

Because there are no dependencies between the samples, the calculation in (2.8) is mathematically equivalent to the calculation

$$\boldsymbol{z}_{k:} = \sum_{i=r_k}^{r_{k+1}-1} \boldsymbol{d}_{i:} * \boldsymbol{y}_{c_i:} \quad \text{for all} \quad k = 1, \ldots, n, \tag{2.9}$$

in which $*$ is the Hadamard product of matrices, that is, the element-wise product of matrices. In an implementation based on (2.9), it is the loop over the samples ($\ell = 1, \ldots, s$) that is the innermost loop.

Although (2.8) and (2.9) are mathematically equivalent, the ordering of the loops used in the implementation has a performance impact. Whereas opportunities for use of vector parallel instructions in an implementation based on (2.8) depend on the sparsity pattern, an implementation based on (2.9) allows the loop over the samples to be parallelized with vector parallel multiply-add instructions, thus providing an illustration of how performance benefit (ii) can be realized with ensemble propagation through a reordering of loops.

There are different ways of storing tensors into the memory. Second-order tensors can be stored using column-major layout, row-major layout, or even other layouts. A layout sets the ordering of the entries in memory by providing for each entry the memory address in terms of an offset from the memory address of the first entry. To store the $(n \times s)$-dimensional second-order tensor $\boldsymbol{Y}$ in the memory, a row-major layout involves storing the (k, ℓ)-th entry with an offset of $(k-1)\,s + \ell$, which we denote as follows:

$$y_{k\ell} \hookleftarrow y\left[(k-1)\,s+\ell\right]. \tag{2.10}$$

Using this layout results in storing consecutive entries of a same row contiguously in memory and consecutive entries of a same column with a stride of s. Thus, the sample values $y_{k1}, \ldots, y_{ks}$ for a given physical degree of freedom indexed by k, that is, $\boldsymbol{y}_{k:}$, are stored in contiguous memory; and the physical degrees of freedom $y_{1\ell}, \ldots, y_{n\ell}$ for a given sample ℓ, that is, $\boldsymbol{y}_{:\ell}$, are stored in interleaved memory (Fig. 2.4).

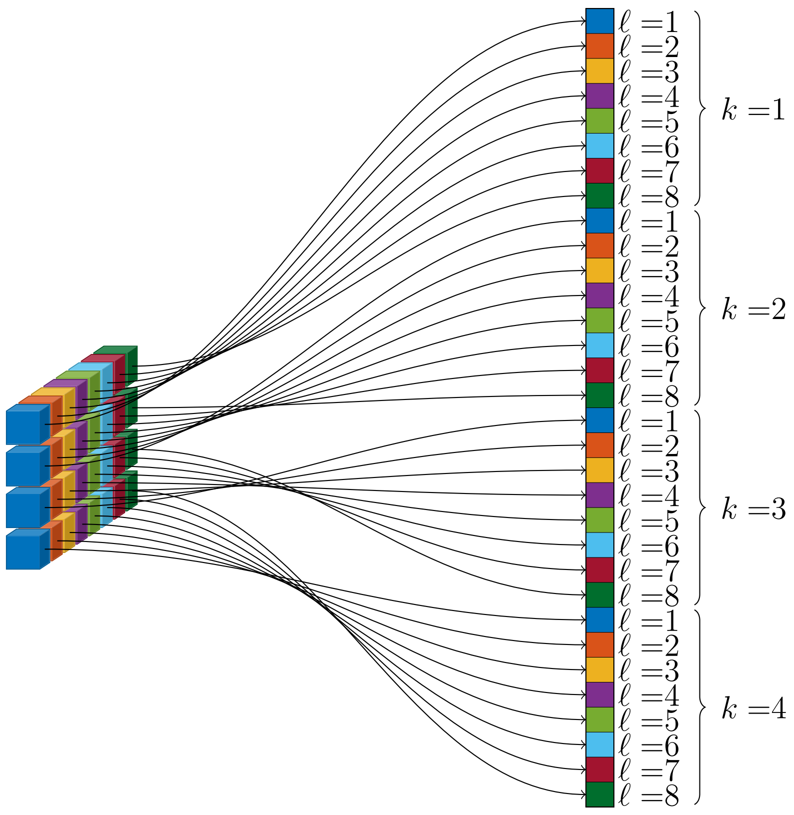


Figure 2.4: Illustration of the memory layout of a vector of 4 ensembles of size 8.

The choice of layout has a performance impact on (2.9) because if $\boldsymbol{Y}$, $\boldsymbol{D}$, and $\boldsymbol{Z}$ are stored using row-major layout, the entries $\boldsymbol{d}_{i:}$ and $\boldsymbol{y}_{c_i:}$ can be loaded using packed loads from contiguous memory from the offsets $(i-1)\,s$ and $(c_i-1)\,s$, respectively, and the entries $\boldsymbol{z}_{k:}$ can be stored using packed stores to contiguous memory from the offset $(k-1)\,s$. Packed loads and stores from and to contiguous memory result in more efficient transfers through the memory hierarchy, thus providing an illustration of how performance benefit (iii) can be realized with ensemble propagation by ensuring ensemble values are contiguous in memory.

In distributed-memory parallelism, the matrix-vector product calculation necessitates communicating entries of $\boldsymbol{Y}$ between compute nodes. Gathering these MPI communications required to access nonlocal entries of the right-hand side vector per ensemble reduces the total communication cost by amortizing the MPI latency. This provides an illustration of how performance benefit (iv) can be realized with ensemble propagation by sending fewer but larger messages.

2.4 Implementation based on templates

Building on the earlier work of [Pawlowski et al., 2012a] for automating embedded analysis capabilities, Phipps et al. [2017] proposed to use the template-based generic-programming approach introduced in Chapter 1 and expression templates [Veldhuizen, 1995] to incorporate ensemble propagation into codes and made available this approach in the Stokhos package [Phipps, 2015] of the Trilinos library [Heroux et al., 2005]. This embedded ensemble propagation consists in transforming code into one capable of propagating ensembles by using templating and operator-overloading capabilities of the C++ language. Concretely, as discussed in Chapter 1, it consists in implementing application code templated on the data type instead of using a floating-point type such as `double`. The templated implementation can then be instantiated with `double` to find back the algorithm with one sample at a time. The same implementation can be used with a new ensemble type that stores an ensemble of s values to use embedded ensemble propagation. Using expression templates, arithmetic and other operations are overloaded with ensemble loops that apply the operations sample-wise. The autovectorization capabilities of the compiler are relied upon to realize these ensemble loops with vector parallel instructions. The ensemble size s, which implies the number of iterations in the ensemble loops, is set at compilation time. Setting the trip count (which is the number of iterations in the loop, the ensemble size s in our case) at compilation time and the property that there are *no backward loop-carried dependencies* (the current iteration of the loop is not impacted by results computed at previous iterations of the loop) aid the compiler with the autovectorization as discussed in [Jeffers et al., 2016]. Best performance is reached when s is both a multiple of the vector instruction length and a multiple of the number of entries storable on a cache line. For example, for architectures that support AVX-512 and have 512-bit cache line, when using `double`, s should be 8, 16, 24, or 32.

2.5 Conclusions

In this chapter, we have introduced embedded ensemble propagation as a way to improve the throughput and the efficiency of parametric computations. The advantages of the approach have been listed and illustrated in the case of a sparse matrix-vector product. The ensemble divergence has been defined and its occurrences have been classified. Parametric linear systems have been introduced using a tensor notation. Finally, the use of a template-based generic-programming approach has been discussed as a way to embed ensemble propagation into existing code.

The originality of this chapter is that, as compared with the work of Phipps et al. [2017], we have classified the occurrences of ensemble divergence, have introduced embedded ensemble propagation using tensor notation, and have revisited the sparse matrix-vector product using the tensor notations.

The speed-up defined in this chapter is used in all the results sections to highlight the interest of ensemble propagation. The types of ensemble divergence are used in Chapter 3, 4, and 5 to discuss strategies to handle them in algorithms. The tensor formalism of this chapter is used in Chapter 3 to formally introduce ensemble GMRES to solve parametric linear system.

Chapter 3

Ensemble GMRES

In this chapter, we formally introduce ensemble GMRES, a solver for parametric linear systems which is able to deal with problems with non-symmetric or indefinite matrices.

In section 3.1, we first start by recalling GMRES applied to solve one linear system at a time. In particular, section 3.1.1 introduces a right preconditioner to accelerate the convergence of GMRES to solve one linear system. Finally, section 3.1.2 lists the types of ensemble divergence that can occur when solving parametric linear systems with ensemble GMRES.

Two methodological approaches are then explained to tackle those types of ensemble divergence.

Section 3.2 is dedicated to ensemble GMRES with ensemble reduction following the same approach as the one used in [Phipps et al., 2017] for CG with ensemble reduction. This approach solves one linear system which couples the s linear systems together and minimizes the root mean square of the norm of the residual of the samples. A zero norm of the residual of the coupled system is achieved only if all the residuals of each linear systems have a zero norm. This approach with reduction, due to the template-based generic programming approach chosen in section 2.4 and due to the implementation work of Phipps et al. [2017], was working before this thesis.

Section 3.3 is dedicated to ensemble GMRES without ensemble reduction. In this second approach, the systems are not coupled together to maintain the same convergence as if they were propagated alone in GMRES. In this approach it is necessary to deal with all occurrences of ensemble divergence explicitly. This second approach required implementation work discussed in Chapter 4.

In section 3.4, we discuss the impact of ensemble reduction on the convergence of ensemble GMRES. To do so, we use theoretical results on the convergence of GMRES for systems with normal matrices to discuss how the coupling of the samples influences the convergence. This is illustrated on an analytical example of a bar problem.

3.1 GMRES for one sample

Let us concisely recall the Generalized Minimal Residual (GMRES) method [Saad and Schultz, 1986; Saad, 2003] for solving the n-dimensional linear system

$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}. \tag{3.1}$$

The GMRES method is a projection method that seeks an approximate solution $\boldsymbol{x}^{(m)}$ from $\boldsymbol{x}^{(0)}+\mathcal{K}_m(\boldsymbol{A},\boldsymbol{r}^{(0)})$, where $\boldsymbol{x}^{(0)}$ is an initial guess, $\boldsymbol{r}^{(0)}=\boldsymbol{b}-\boldsymbol{A}\,\boldsymbol{x}^{(0)}$ is the corresponding residual, and $\mathcal{K}_m(\boldsymbol{A},\boldsymbol{r}^{(0)})$ is the Krylov subspace

$$\mathcal{K}_m(\boldsymbol{A},\boldsymbol{r}^{(0)}) := \operatorname{span}\left\{\boldsymbol{r}^{(0)},\boldsymbol{A}\,\boldsymbol{r}^{(0)},\dots,\boldsymbol{A}^{m-1}\,\boldsymbol{r}^{(0)}\right\}, \tag{3.2}$$

by imposing that

$$\boldsymbol{b}-\boldsymbol{A}\,\boldsymbol{x}^{(m)}\perp \boldsymbol{A}(\mathcal{K}_m(\boldsymbol{A},\boldsymbol{r}^{(0)})), \tag{3.3}$$

that is

$$\boldsymbol{z}^{\mathrm{T}}\,(\boldsymbol{b}-\boldsymbol{A}\,\boldsymbol{x}^{(m)})=0 \quad \text{for all} \quad \boldsymbol{z}\in \boldsymbol{A}(\mathcal{K}_m(\boldsymbol{A},\boldsymbol{r}^{(0)})). \tag{3.4}$$

The approximate solution $\boldsymbol{x}^{(m)}$ minimizes the norm of the residual:

$$\boldsymbol{x}^{(m)} = \underset{\boldsymbol{x}\in\boldsymbol{x}^{(0)}+\mathcal{K}_m(\boldsymbol{A},\boldsymbol{r}^{(0)})}{\arg\min}\ \|\boldsymbol{b}-\boldsymbol{A}\,\boldsymbol{x}\|. \tag{3.5}$$

There exist several implementation strategies to generate an orthonormal basis of the Krylov subspace such as strategies based on classical Gram-Schmidt orthonormalization, modified Gram-Schmidt orthonormalization, Householder orthonormalization, DGKS [Daniel et al., 1976], or TSQR [Demmel et al., 2008]. In this work, we restricted ourselves to the strategies relying on the level-2 BLAS function GEMV which allows the possibility to have more cache reuse than the level-1 BLAS functions. Other strategies such as those based on the level-1 BLAS functions can be considered too. The expected speed-up of both the strategies based on the level-1 BLAS functions and the strategies based on the level-2 BLAS function GEMV, being limited by the memory bandwidth, is expected to be equal to 1. All the results of this thesis are generated using the DGKS strategy: the classical Gram-Schmidt orthonormalization followed by a second orthonormalization if required to improve the numerical stability. Therefore, the considered implementation of the GMRES method is based on the Arnoldi method that builds an orthonormal basis $\boldsymbol{v}_1, \dots, \boldsymbol{v}_{m+1}$, whose vectors are named Arnoldi vectors, by Gram-Schmidt orthonormalization of $\boldsymbol{r}^{(0)}$, $\boldsymbol{A}\boldsymbol{r}^{(0)}$, …, $\boldsymbol{A}^m\boldsymbol{r}^{(0)}$. Specifically, the Arnoldi method successively multiplies the previous vector $\boldsymbol{v}_j$ with $\boldsymbol{A}$ and orthonormalizes the resulting $\boldsymbol{w}$ against all previous $\boldsymbol{v}_i$'s. Let the resulting $\boldsymbol{v}_1, \dots, \boldsymbol{v}_{m+1}$ be collected as column vectors in the $n\times(m+1)$-dimensional matrix $\boldsymbol{V}$ and the coefficients of the Gram-Schmidt orthonormalization in the $(m+1)\times m$-dimensional upper Hessenberg matrix $\boldsymbol{H}$.

As a consequence of the orthonormality property, the solution to the reduced-dimensional problem in (3.4) is given by

$$\boldsymbol{x}^{(m)} = \boldsymbol{x}^{(0)} + \boldsymbol{V}_{:(1:m)}\,\boldsymbol{y}, \tag{3.6}$$

with $\boldsymbol{y}$ in $\mathbb{R}^m$ such that

$$\|\beta\,\boldsymbol{e}_1-\boldsymbol{H}\,\boldsymbol{y}\|^2 = \min_{\boldsymbol{z}\in\mathbb{R}^m}\|\beta\,\boldsymbol{e}_1-\boldsymbol{H}\,\boldsymbol{z}\|^2, \tag{3.7}$$

with $\beta=\|\boldsymbol{r}^{(0)}\|$ and $\boldsymbol{e}_1$ the first column of the appropriately sized identity matrix.

This least-squares problem is solved by factoring $\boldsymbol{H}$ into the product of an orthogonal matrix $\boldsymbol{Q}$ and an upper triangular matrix $\boldsymbol{U}$,

$$\boldsymbol{H} = \boldsymbol{Q}\begin{bmatrix}\boldsymbol{U}\\ \boldsymbol{0}\end{bmatrix}. \tag{3.8}$$

This factorization is performed using a progressive manner, as discussed in the section 6.5.3 in [Saad, 2003], using Givens rotations at each iteration of the GMRES method.

The factorization (3.8) is such that

$$\left\| \begin{bmatrix} \boldsymbol{g} \\ e \end{bmatrix} - \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{0} \end{bmatrix} \boldsymbol{y} \right\|^2 = \min_{\boldsymbol{z} \in \mathbb{R}^m} \left\| \begin{bmatrix} \boldsymbol{g} \\ e \end{bmatrix} - \begin{bmatrix} \boldsymbol{U} \\ \boldsymbol{0} \end{bmatrix} \boldsymbol{z} \right\|^2, \text{ with } \begin{bmatrix} \boldsymbol{g} \\ e \end{bmatrix} = \boldsymbol{Q}^{\mathrm{T}} \beta\, \boldsymbol{e}_1. \tag{3.9}$$

As a result, the solution to the reduced-dimensional problem in (3.4) can be computed as

$$\boldsymbol{x}^{(m)} = \boldsymbol{x}^{(0)} + \boldsymbol{V}_{:(1:m)}\, \boldsymbol{U}^{-1}\, \boldsymbol{g}, \tag{3.10}$$

and the norm of the residual is given, assuming infinite precision, by

$$\|\boldsymbol{b} - \boldsymbol{A}\, \boldsymbol{x}^{(m)}\| = e. \tag{3.11}$$

In this thesis, we consider two types of stopping criterion: the stopping criterion based on the *implicit norm* e of (3.9) and the stopping criterion based on the *explicit norm* $\|\boldsymbol{b} - \boldsymbol{A}\, \boldsymbol{x}^{(m)}\|$. The advantage of using the implicit norm is that we have a way to evaluate an approximation of the norm of the residual without extra cost as we do not need to compute $\boldsymbol{x}^{(m)}$. The advantage of using the explicit norm is that this approach is more precise. However it requires to compute $\boldsymbol{x}^{(m)}$ and to evaluate the norm of the residual.

The algorithm is shown in Algo. 1. For brevity's sake, we do not cover the case where the maximal number of iterations is not sufficiently large to enforce the convergence of the solution in the sense of the considered convergence test and we do not explicitly describe the progressive factorization of $\boldsymbol{H}$.

Algorithm 1: GMRES for a single sample individually.

```
1  $r^{(0)} = b - A\, x^{(0)}$
2  $\beta = \|r^{(0)}\|$
3  $v_{:1} = r^{(0)}/\beta$
4  for $j = 1, \dots, m$ do
5      $w = A\, v_{:j}$
6      $h_{(1:j)j} = V^{\mathrm{T}}_{:(1:j)}\, w$                      inner products
7      $v_{:(j+1)} = w - V_{:(1:j)}\, h_{(1:j)j}$                     update
8      $h_{(j+1)j} = \|v_{:(j+1)}\|$                                  normalization
9      if $h_{(j+1)j} \neq 0$ then
10         $v_{:(j+1)} = v_{:(j+1)}/h_{(j+1)j}$
11     else                                                           lucky breakdown
12         $m = j$
13         break
14     if $q^{\mathrm{T}}_{:(j+1)} e_1 \leq \varepsilon$ then                 convergence test
15         $m = j$
16         break
17 $y = \arg\min_z \|\beta\, e_1 - H_{(1:m+1)(1:m)}\, y\|$
18 $x^{(m)} = x^{(0)} + V_{:(1:m)}\, z$
```

3.1.1 Right-preconditioned GMRES

Preconditioning is known as a way to improve robustness and efficiency of iterative techniques as described in [Saad, 2003]. In this thesis, we consider right-preconditioned GMRES as described in this subsection.

Given a right preconditioner $\boldsymbol{M}^{-1}$, the right-preconditioned GMRES solves (3.1) by solving

$$\boldsymbol{A}\,\boldsymbol{M}^{-1}\,\boldsymbol{u} = \boldsymbol{b}, \tag{3.12}$$

$$\boldsymbol{M}\,\boldsymbol{x} = \boldsymbol{u}. \tag{3.13}$$

The right-preconditioned GMRES method is a projection method that seeks an approximate solution $\boldsymbol{x}^{(m)}$ from $\boldsymbol{x}^{(0)}+\boldsymbol{M}^{-1}(\mathcal{K}_m(\boldsymbol{AM}^{-1},\boldsymbol{r}^{(0)}))$, where $\boldsymbol{x}^{(0)}$ is an initial guess, $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}^{(0)}$ is the corresponding residual, and $\mathcal{K}_m(\boldsymbol{AM}^{-1},\boldsymbol{r}^{(0)})$ is the Krylov subspace

$$\mathcal{K}_m(\boldsymbol{AM}^{-1},\boldsymbol{r}^{(0)}) := \operatorname{span}\left\{\boldsymbol{r}^{(0)}, \boldsymbol{AM}^{-1}\,\boldsymbol{r}^{(0)}, \dots, \left(\boldsymbol{AM}^{-1}\right)^{m-1}\,\boldsymbol{r}^{(0)}\right\}, \tag{3.14}$$

by imposing that

$$\boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}^{(m)} \perp \boldsymbol{AM}^{-1}(\mathcal{K}_m(\boldsymbol{AM}^{-1},\boldsymbol{r}^{(0)})). \tag{3.15}$$

The algorithm is shown in Algo. 2.

For the remainder of this thesis the term GMRES is used to denote the right-preconditioned GMRES.

Algorithm 2: Right-preconditioned GMRES for a single sample individually.

```
1  $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}^{(0)}$
2  $\beta = \|\boldsymbol{r}^{(0)}\|$
3  $\boldsymbol{v}_{:1} = \boldsymbol{r}^{(0)}/\beta$
4  for $j = 1, \dots, m$ do
5      $\boldsymbol{w} = \boldsymbol{AM}^{-1}\,\boldsymbol{v}_{:j}$
6      $\boldsymbol{h}_{(1:j)j} = \boldsymbol{V}^{\mathrm{T}}_{:(1:j)}\,\boldsymbol{w}$                          inner products
7      $\boldsymbol{v}_{:(j+1)} = \boldsymbol{w} - \boldsymbol{V}_{:(1:j)}\,\boldsymbol{h}_{(1:j)j}$                update
8      $h_{(j+1)\,j} = \|\boldsymbol{v}_{:\,(j+1)}\|$                            normalization
9      if $h_{(j+1)\,j} \neq 0$ then
10         $\boldsymbol{v}_{:\,(j+1)} = \boldsymbol{v}_{:\,(j+1)}/h_{(j+1)\,j}$
11     else                                                      lucky breakdown
12         $m = j$
13         break
14     if $\boldsymbol{q}^{\mathrm{T}}_{:(j+1)}\boldsymbol{e}_1 \leq \varepsilon$ then                         convergence test
15         $m = j$
16         break
17 $\boldsymbol{y} = \arg\min_{\boldsymbol{z}}\,\|\beta\,\boldsymbol{e}_1 - \boldsymbol{H}_{(1:m+1)(1:m)}\,\boldsymbol{y}\|$
18 $\boldsymbol{x}^{(m)} = \boldsymbol{x}^{(0)} + \boldsymbol{M}^{-1}\,\boldsymbol{V}_{:(1:m)}\,\boldsymbol{y}$
```

3.1.2 Challenges in applying ensemble propagation to GMRES

Let us now consider the application of ensemble propagation to the GMRES method for solving the n-dimensional parametric linear system (2.6). The application of ensemble propagation to the GMRES method introduces the three types of ensemble divergence introduced in Chapter 1:

1. an Arnoldi vector may require a normalization for certain samples but not for other samples, so-called if-then-else divergence,

2. different samples may require different numbers of iterations to converge and hence exit the for loop in Algo. 1 after different trip counts, so-called loop divergence,

3. called BLAS functions, such as GEMV for the dense matrix-vector operations, may not support ensemble-typed inputs, so-called function call divergence.

There are two ways to overcome the stated challenges. The first one involves a mathematical reformulation of the GMRES, using ensemble reduction, to suppress every ensemble divergence. The second one consists in explicitly managing every occurrence of ensemble divergence. We will call this second approach ensemble GMRES without ensemble reduction.

3.2 Ensemble GMRES with ensemble reduction

Ensemble reduction was introduced by Phipps et al. [2017] as a way of overcoming ensemble-divergence challenges arising in the application of ensemble propagation to Krylov-based linear solvers.

Formally, ensemble reduction is mathematically equivalent to the gathering of the system matrices along the diagonal of a block-diagonal matrix and the concatenation of the left-hand-side and right-hand-side vectors

$$\begin{bmatrix} \boldsymbol{A}_{::1} & & \\ & \ddots & \\ & & \boldsymbol{A}_{::s} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{:1} \\ \vdots \\ \boldsymbol{x}_{:s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_{:1} \\ \vdots \\ \boldsymbol{b}_{:s} \end{bmatrix}, \tag{3.16}$$

followed by the application of the GMRES method to this ns-dimensional system with the projection defined with the inner product of the ns-dimensional Euclidean vector space, that is, the so-called reduced inner product that involves a summation over the ensemble, as illustrated in Fig. 3.1 and in Listing 1.5.

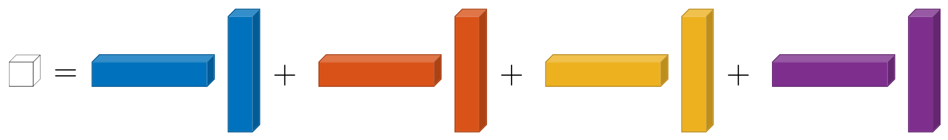


Figure 3.1: Illustration of the reduced inner product of two vectors of ensemble values. Each color refers to a particular sample. The result is drawn in white to represent the fact that it is not related to a particular sample.

It is important to stress that this block-diagonal matrix and vectors are never computed in practice. This mathematical equivalence is discussed in order to ease the interpretation of the impact of ensemble reduction on the convergence.

In order to formalize ensemble reduction within our tensor-based formalism, it is convenient to associate a linear operator $\mathcal{A}$ to the third-order tensor $\mathscr{A}$

$$\begin{aligned} \mathcal{A} &: \mathbb{R}^{n\times s} \to \mathbb{R}^{n\times s} && (3.17)\\ &: \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_{:1} & \dots & \boldsymbol{x}_{:s} \end{bmatrix} \mapsto \mathcal{A}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{A}_{::1}\boldsymbol{x}_{:1} & \dots & \boldsymbol{A}_{::s}\boldsymbol{x}_{:s} \end{bmatrix}; && (3.18) \end{aligned}$$

then, the parametric linear system reads as

$$\mathcal{A}(\boldsymbol{X}) = \boldsymbol{B}. \tag{3.19}$$

Moreover, associating a linear operator $\mathcal{M}$ to the right preconditioner:

$$\begin{aligned} \mathcal{M} &: \mathbb{R}^{n\times s} \to \mathbb{R}^{n\times s} && (3.20)\\ &: \boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_{:1} & \dots & \boldsymbol{x}_{:s} \end{bmatrix} \mapsto \mathcal{M}(\boldsymbol{X}) = \begin{bmatrix} \boldsymbol{M}_{::1}^{-1}\boldsymbol{x}_{:1} & \dots & \boldsymbol{M}_{::s}^{-1}\boldsymbol{x}_{:s} \end{bmatrix}, && (3.21) \end{aligned}$$

allows us to write the preconditioned parametric linear system as

$$\begin{aligned} \mathcal{A} \circ \mathcal{M}(\boldsymbol{U}) &= \boldsymbol{B}, && (3.22)\\ \boldsymbol{X} &= \mathcal{M}(\boldsymbol{U}). && (3.23) \end{aligned}$$

The GMRES method with ensemble propagation and ensemble reduction then involves seeking an approximate solution $\boldsymbol{X}^{(m)}$ from $\boldsymbol{X}^{(0)} + \mathcal{M}(\mathcal{K}_m(\mathscr{A}\mathscr{M}, \boldsymbol{R}^{(0)}))$, where $\boldsymbol{X}^{(0)}$ is an initial guess, $\boldsymbol{R}^{(0)} = \boldsymbol{B} - \mathcal{A}(\boldsymbol{X}^{(0)})$ is the corresponding residual, and $\mathcal{K}_m(\mathscr{A}\mathscr{M}, \boldsymbol{R}^{(0)})$ is the Krylov subspace

$$\mathcal{K}_m(\mathscr{A}\mathscr{M}, \boldsymbol{R}^{(0)}) \equiv \operatorname{span}\left\{\boldsymbol{R}^{(0)}, \mathcal{A}\circ\mathcal{M}(\boldsymbol{R}^{(0)}), \dots, (\mathcal{A}\circ\mathcal{M})^{m-1}(\boldsymbol{R}^{(0)})\right\}, \tag{3.24}$$

by imposing that

$$\boldsymbol{B} - \mathcal{A}(\boldsymbol{X}^{(m)}) \perp \mathcal{A}\circ\mathcal{M}(\mathcal{K}_m(\mathscr{A}\mathscr{M}, \boldsymbol{R}^{(0)})), \tag{3.25}$$

that is

$$\sum_{\ell=1}^{s} \boldsymbol{z}_{:\ell}^{\mathrm{T}} (\boldsymbol{b}_{:\ell} - \boldsymbol{A}_{::\ell}\boldsymbol{x}_{:\ell}^{(m)}) = 0 \quad \text{for all} \quad \boldsymbol{Z} \in \mathcal{A}\circ\mathcal{M}(\mathcal{K}_m(\mathscr{A}\mathscr{M}, \boldsymbol{R}^{(0)})). \tag{3.26}$$

The Arnoldi method now involves building an orthonormal basis $\boldsymbol{V}_{:1:}, \dots, \boldsymbol{V}_{:(m+1):}$ by Gram-Schmidt orthonormalization of $\boldsymbol{R}^{(0)}$, $\mathcal{A}\circ\mathcal{M}(\boldsymbol{R}^{(0)})$, …, $(\mathcal{A}\circ\mathcal{M})^{m}(\boldsymbol{R}^{(0)})$ with respect to the reduced inner product. As this approach uses the reduced inner product and the Krylov subspace of (3.24) the coefficients are still scalars and collected in the $(m+1)\times m$-dimensional upper Hessenberg matrix $\boldsymbol{H}$.

The solution to the reduced-dimensional problem in (3.26) is then given by

$$\boldsymbol{x}_{:\ell}^{(m)} = \boldsymbol{x}_{:\ell}^{(0)} + \boldsymbol{M}_{::\ell}^{-1}\boldsymbol{V}_{:(1:m)\ell}\,\boldsymbol{y} \quad \text{for all} \quad \ell = 1, \dots, s, \tag{3.27}$$

with $\boldsymbol{y}$ in $\mathbb{R}^m$ such that

$$\|\beta\,\boldsymbol{e}_1 - \boldsymbol{H}\,\boldsymbol{y}\|^2 = \min_{\boldsymbol{z}\in\mathbb{R}^m} \|\beta\,\boldsymbol{e}_1 - \boldsymbol{H}\,\boldsymbol{z}\|^2, \tag{3.28}$$

with $\beta = \sqrt{\sum_{\ell=1}^{s} \|\boldsymbol{r}_{:\ell}^{(0)}\|^2}$.

We can see that ensemble reduction couples the samples together as they share the same reduced degrees of freedom $\boldsymbol{y}$.

The use of ensemble reduction is shown in the Algo. 3.

Algorithm 3: GMRES with ensemble reduction.

```
1  $\boldsymbol{r}^{(0)}_{:\ell} = \boldsymbol{b}_{:\ell} - \boldsymbol{A}_{::\ell}\,\boldsymbol{x}^{(0)}_{:\ell}$   for all   $\ell = 1, \dots, s$
2  $\beta = \sqrt{\sum_{\ell=1}^{s} \|\boldsymbol{r}^{(0)}_{:\ell}\|^2}$
3  $\boldsymbol{v}_{:1\ell} = \boldsymbol{r}^{(0)}_{:\ell}/\beta$   for all   $\ell = 1, \dots, s$
4  for $j = 1, \dots, m$ do
5      $\boldsymbol{w}_{:\ell} = \boldsymbol{A}_{::\ell}\boldsymbol{M}^{-1}_{::\ell}\,\boldsymbol{v}_{:j\ell}$   for all   $\ell = 1, \dots, s$
6      $\boldsymbol{h}_{(1:j)j} = \sum_{\ell=1}^{s} \boldsymbol{V}^{\mathrm{T}}_{:(1:j)\ell}\,\boldsymbol{w}_{:\ell}$                                inner products
7      $\boldsymbol{v}_{:(j+1)\ell} = \boldsymbol{w}_{:\ell} - \boldsymbol{V}_{:(1:j)\ell}\,\boldsymbol{h}_{(1:j)j}$   for all   $\ell = 1, \dots, s$        update
8      $h_{(j+1)j} = \sqrt{\sum_{\ell=1}^{s} \|\boldsymbol{v}_{:(j+1)\ell}\|^2}$                                normalization
9      if $h_{(j+1)j} \neq 0$ then
10         $\boldsymbol{v}_{:(j+1)\ell} = \boldsymbol{v}_{:(j+1)\ell}/h_{(j+1)j}$   for all   $\ell = 1, \dots, s$
11     else                                                                 lucky breakdown
12         $m = j$
13         break
14     if $\boldsymbol{q}^{\mathrm{T}}_{:(j+1)}\boldsymbol{e}_1 \leq \varepsilon$ then                                      convergence test
15         $m = j$
16         break
17 $\boldsymbol{y} = \arg\min_{\boldsymbol{z}} \|\beta\,\boldsymbol{e}_1 - \boldsymbol{H}_{(1:m+1)(1:m)}\,\boldsymbol{z}\|$
18 $\boldsymbol{x}^{(m)}_{:\ell} = \boldsymbol{x}^{(0)}_{:\ell} + \boldsymbol{M}^{-1}_{::\ell}\boldsymbol{V}_{:(1:m)\ell}\,\boldsymbol{y}$   for all   $\ell = 1, \dots, s$
```

This ensemble reduction suppresses every ensemble divergence as, conceptually, this approach is equivalent to applying the GMRES method to an ns-dimensional system. This implies that samples will always follow the same branch in the normalization process at line 9 in Algo. 3 and will always have the same trip count in GMRES at line 14 in Algo. 3 as the norms are computed reducing the norm of each sample. Moreover, $\boldsymbol{H}$ is of order two and, therefore, BLAS functions for scalar-typed inputs are sufficient, this suppresses the function call divergence. The drawback of ensemble reduction is that samples are coupled to each other which impacts both convergence and accuracy:

- the mathematical equivalence with the block-diagonal matrix (3.16) implies the union of the spectrum of the system matrices, as discussed in [Phipps et al., 2017], and the union of their singular values. The union of the spectrum implies a potentially slower convergence of GMRES compared to GMRES applied to each sample individually. The union of the singular values implies a potentially increased condition number.

- using the reduced norm impacts the convergence criterion. In particular, using an absolute criterion based on $\sqrt{\sum_{\ell=1}^{s} \|\boldsymbol{r}^{(j)}_{:\ell}\|^2}$, the reduced norm of the residual at the iteration j, leads to an increased number of iterations to converge compared to using a GMRES on each sample individually with a same tolerance value. To overcome this issue, we use a relative convergence criterion, by which we mean a convergence criterion based on the ratio $\sqrt{\sum_{\ell=1}^{s} \|\boldsymbol{r}^{(j)}_{:\ell}\|^2}/\sqrt{\sum_{\ell=1}^{s} \|\boldsymbol{r}^{(0)}_{:\ell}\|^2}$. This convergence criterion implies the convergence of the root mean square of the norm of the residual of each sample which is a weaker convergence than the relative convergence of all the samples which will be used as the convergence criterion for GMRES without ensemble reduction in the next section. Mathematically, the latter is a sufficient condition for the former:

$$\frac{\|\boldsymbol{r}^{(j)}_{:\ell}\|}{\|\boldsymbol{r}^{(0)}_{:\ell}\|} \leq \varepsilon \quad \text{for all} \quad \ell = 1, \dots, s \;\Rightarrow\; \frac{\sqrt{\sum_{\ell=1}^{s} \|\boldsymbol{r}^{(j)}_{:\ell}\|^2}}{\sqrt{\sum_{\ell=1}^{s} \|\boldsymbol{r}^{(0)}_{:\ell}\|^2}} \leq \varepsilon. \tag{3.29}$$

3.3 Ensemble GMRES without ensemble reduction

Using an ensemble inner product without ensemble reduction in Krylov-based solvers has already been proposed in [D'Elia et al., 2020]. It means that an inner product of 2 vectors of `Ensemble<s>` returns an `Ensemble<s>`, as discussed in Chapter 1, instead of a scalar as in the previous case with reduction. In [D'Elia et al., 2020], the motivation for introducing this inner product without ensemble reduction was to enable a fine-grained monitoring of the convergence of each sample individually so as to facilitate the grouping of samples. The use of this inner product without ensemble reduction raises challenges of ensemble divergence, which were solved in [D'Elia et al., 2020] for the particular case of the conjugate gradient method using the `EnsembleTrait` proposed by Phipps et al. [2017] and discussed in section 4.1. Here, we show that the use of the inner product without ensemble reduction can have a performance impact by accelerating convergence as compared with the use of the inner product with ensemble reduction. And we highlight and solve the new challenges of ensemble divergence in the case of GMRES.

This approach requires explicit modifications of the algorithm, as said in Chapter 1, to manage the if-then-else and the loop divergences. In this work, those modifications rely on the so-called *masked assignments* [Kretz and Lindenstruth, 2012] and the so-called *logical reduction functions* [Kretz, 2015]. The masked assignments are conditional assignments which assign different values depending on the value of a conditional. The logical reduction functions are logical operations applied on a set of booleans such as *for all* or *for at least one* as discussed in detail in the next chapter.

Conceptually, this approach solves the s linear systems at the same time by continuing the Arnoldi method for every sample of the ensemble, independently of whether they have individually converged, up to the convergence of all of them.

The GMRES method with ensemble propagation without ensemble reduction involves seeking an approximate solution $\boldsymbol{x}^{(m)}_{:\ell}$ from $\boldsymbol{x}^{(0)}_{:\ell} + \boldsymbol{M}^{-1}_{::\ell}\mathcal{K}_m(\boldsymbol{A}_{::\ell}\boldsymbol{M}^{-1}_{::\ell}, \boldsymbol{r}^{(0)}_{:\ell})$ for all $\ell = 1$, $\dots$, s, where $\boldsymbol{x}^{(0)}_{:\ell}$ is an initial guess and $\boldsymbol{r}^{(0)}_{:\ell} = \boldsymbol{b}_{:\ell} - \boldsymbol{A}_{::\ell}\,\boldsymbol{x}^{(0)}_{:\ell}$ is the corresponding residual, by imposing that

$$\boldsymbol{b}_{:\ell} - \boldsymbol{A}_{::\ell}\,\boldsymbol{x}^{(m)}_{:\ell} \perp \boldsymbol{A}_{::\ell}\boldsymbol{M}^{-1}_{::\ell}(\mathcal{K}_m(\boldsymbol{A}_{::\ell}\boldsymbol{M}^{-1}_{::\ell}, \boldsymbol{r}^{(0)}_{:\ell})) \quad \text{for all} \quad \ell = 1, \dots, s, \tag{3.30}$$

that is

$$\begin{aligned} \boldsymbol{z}^{\mathrm{T}}_{:\ell}\,(\boldsymbol{b}_{:\ell} - \boldsymbol{A}_{::\ell}\,\boldsymbol{x}^{(m)}_{:\ell}) = 0 \quad \text{for all} \quad \boldsymbol{z}_{:\ell} \in \boldsymbol{A}_{::\ell}\boldsymbol{M}^{-1}_{::\ell}(\mathcal{K}_m(\boldsymbol{A}_{::\ell}\boldsymbol{M}^{-1}_{::\ell}, \boldsymbol{r}^{(0)}_{:\ell}))& \\ \text{for all} \quad \ell = 1, \dots, s.& \end{aligned} \tag{3.31}$$

The Arnoldi method now involves building an orthonormal basis $\boldsymbol{v}_{:1\ell}$, $\dots$, $\boldsymbol{v}_{:(m+1)\ell}$ by Gram-Schmidt orthonormalization of $\boldsymbol{r}^{(0)}_{:\ell}$, $\boldsymbol{A}_{::\ell}\boldsymbol{M}^{-1}_{::\ell}\,\boldsymbol{r}^{(0)}_{:\ell}$, $\dots$, $(\boldsymbol{A}_{::\ell}\boldsymbol{M}^{-1}_{::\ell})^m\,\boldsymbol{r}^{(0)}_{:\ell}$ for all

$\ell = 1, \ldots, s$. Because there is no reduction, the coefficients are sample-dependent and collected in the $(m+1)\times m\times s$-dimensional tensor $\mathscr{H}$.

The solution to the reduced-dimensional problem in (3.31) is then given by

$$\boldsymbol{x}_{:\ell}^{(m)} = \boldsymbol{x}_{:\ell}^{(0)} + \boldsymbol{M}_{::\ell}^{-1} \boldsymbol{V}_{:(1:m)\ell}\, \boldsymbol{y}_{:\ell} \quad \text{for all} \quad \ell = 1, \ldots, s, \tag{3.32}$$

with $\boldsymbol{Y}$ in $\mathbb{R}^{m\times s}$ such that

$$\|\beta_\ell\, \boldsymbol{e}_1 - \boldsymbol{H}_{::\ell}\, \boldsymbol{y}_{:\ell}\|^2 = \min_{\boldsymbol{z}_{:\ell}\in\mathbb{R}^m} \|\beta_\ell\, \boldsymbol{e}_1 - \boldsymbol{H}_{::\ell}\, \boldsymbol{z}_{:\ell}\|^2 \quad \text{for all} \quad \ell = 1, \ldots, s, \tag{3.33}$$

with $\beta_\ell = \|\boldsymbol{r}_\ell^{(0)}\|$ for all $\ell = 1, \ldots, s$.

We can see that the samples are not coupled together and that the degrees of freedom are sample-dependent.

The normalization process of the Arnoldi vectors is managed with a loop over the samples around an if-then-else statement as shown at line 9 in Algo. 4. This loop will be implemented using a masked assignment as discussed in the next chapter.

Algorithm 4: GMRES without ensemble reduction.

```
1  $\boldsymbol{r}_{:\ell}^{(0)} = \boldsymbol{b}_{:\ell} - \boldsymbol{A}_{::\ell}\, \boldsymbol{x}_{:\ell}^{(0)}$   for all  $\ell = 1, \ldots, s$
2  $\beta_\ell = \|\boldsymbol{r}_{:\ell}^{(0)}\|$   for all  $\ell = 1, \ldots, s$
3  $\boldsymbol{v}_{:1\ell} = \boldsymbol{r}_{:\ell}^{(0)}/\beta_\ell$   for all  $\ell = 1, \ldots, s$
4  for $j = 1, \ldots, m$ do
5      $\boldsymbol{w}_{:\ell} = \boldsymbol{A}_{::\ell}\boldsymbol{M}_{::\ell}^{-1}\, \boldsymbol{v}_{:j\ell}$   for all  $\ell = 1, \ldots, s$
6      $\boldsymbol{h}_{(1:j)j\ell} = \boldsymbol{V}_{:(1:j)\ell}^{\mathrm{T}}\, \boldsymbol{w}_{:\ell}$   for all  $\ell = 1, \ldots, s$        inner products
7      $\boldsymbol{v}_{:(j+1)\ell} = \boldsymbol{w}_{:\ell} - \boldsymbol{V}_{:(1:j)\ell}\, \boldsymbol{h}_{(1:j)j\ell}$   for all  $\ell = 1, \ldots, s$        update
8      $h_{(j+1)j\ell} = \|\boldsymbol{v}_{:(j+1)\ell}\|$   for all  $\ell = 1, \ldots, s$        normalization
9      for $\ell = 1, \ldots, s$ do
10         if $h_{(j+1)j\ell} \neq 0$ then
11             $\boldsymbol{v}_{:(j+1)\ell} = \boldsymbol{v}_{:(j+1)\ell}/h_{(j+1)j\ell}$   for all  $\ell = 1, \ldots, s$
12     if $h_{(j+1)j\ell} = 0$   for all  $\ell = 1, \ldots, s$ then        lucky breakdown
13         $m = j$
14         break
15     for $\ell = 1, \ldots, s$ do
16         if $h_{j(j-1)\ell} = 0$ then
17             $h_{jj\ell} = 1$
18             $h_{(j+1)j\ell} = 0$
19     if $\boldsymbol{q}_{:(j+1)\ell}^{\mathrm{T}}\boldsymbol{e}_1 \leq \varepsilon$   for all  $\ell = 1, \ldots, s$ then        convergence test
20         $m = j$
21         break
22 $\boldsymbol{y}_{:\ell} = \arg\min_{\boldsymbol{z}_{:\ell}} \|\beta_\ell\, \boldsymbol{e}_1 - \boldsymbol{H}_{(1:m+1)(1:m)\ell}\, \boldsymbol{z}_{:\ell}\|$   for all  $\ell = 1, \ldots, s$
23 $\boldsymbol{x}_{:\ell}^{(m)} = \boldsymbol{x}_{:\ell}^{(0)} + \boldsymbol{M}_{::\ell}^{-1}\boldsymbol{V}_{:(1:m)\ell}\, \boldsymbol{y}_{:\ell}$   for all  $\ell = 1, \ldots, s$
```

3.3.1 Loop divergence

As already highlighted, different samples may require different numbers of iterations to converge. In order to manage this loop divergence, we require the GMRES method to continue the Arnoldi method until the reduced basis is sufficiently rich for every sample to have converged. Doing so enforces that the accuracy of the approximations is at least as good as that of the approximations computed without ensemble propagation. Thus, the sample-dependent convergence criteria are replaced with an overall convergence criterion that involves a logical ensemble reduction of the type AND to stop only if every sample has converged. As already stated in (3.29), this convergence criterion is stronger than the relative convergence criterion with ensemble reduction with the same tolerance value.

3.3.2 If-then-else divergence

Continuing to update the approximation of all samples to reach convergence for all samples raises a new issue: if a sample ℓ encounters a lucky breakdown at an iteration j_ℓ or has a vanishing norm $h_{(j_\ell+1)\,j_\ell\,\ell}$ due to underflow, the least-squares problem to be solved for this sample is no longer well posed. The most straightforward way to address this issue would be to solve individually the least-squares problems with a sample-dependent size j_ℓ, however, this would prevent the use of ensemble propagation inside the least-squares solve. An alternative approach is therefore preferred and involves maintaining least-squares problems of the same size m and modifying them such that $\boldsymbol{y}_{(j_\ell+1:m)\ell} = \mathbf{0}$.

A way to enforce $\boldsymbol{y}_{(j_\ell+1:m)\ell} = \mathbf{0}$ is to enforce that $\boldsymbol{H}_{(1:m+1)(1:m)\ell}$ is full rank. It can be proven that $\boldsymbol{H}_{(1:j_\ell+1)(1:j_\ell)\ell}$ is full rank, therefore, the corrections

$$h_{ii\ell} = 1 \quad \text{for all} \quad i = j_\ell + 1, \dots, m, \tag{3.34}$$

$$h_{(i+1)i\ell} = 0 \quad \text{for all} \quad i = j_\ell + 1, \dots, m, \tag{3.35}$$

are sufficient to enforce the full rank of $\boldsymbol{H}_{(1:m+1)(1:m)\ell}$.

The algorithm shown in Algo. 4 treats the control-flow divergence using two masked assignments, i.e. two conditional assignments, one for the normalization at lines 10 to 11 of Algo. 4 and one for the lucky breakdown at lines 16 to 18 of Algo. 4, and one logical reduction function per stopping criterion. Those notions will be extensively discussed in Chapter 4.

3.3.3 Function call divergence

As opposed to GMRES with ensemble reduction, this approach does not allow the efficient use of implementations of BLAS functions for scalar types and the function call divergence must be overcome implementing BLAS functions that support ensemble-typed inputs as discussed in Chapter 4.

3.4 Discussion on the impact of ensemble reduction on the convergence of ensemble GMRES

In this section, we discuss the consequences of ensemble reduction on the convergence of ensemble GMRES. To do so, we first discuss the convergence of GMRES for one sample, we then discuss how the reduction impacts the convergence coupling the samples together, and finally we illustrate this influence on an example.

The theory for the convergence of GMRES in the general case is complex as discussed in [Meurant and Tebbens, 2015]. Therefore, in this section, we restrict ourselves to the case of GMRES applied to normal matrices. In this particular case, the convergence of GMRES is fully determined by two quantities [Meurant and Tebbens, 2015]: eigenvalues and components of the initial residual in the eigenvector basis.

3.4.1 Convergence of GMRES

Let $\sum_{I_k}$ denote the summation over all possible sets I_k of k indices $i_1, \dots, i_k$ such that $1 \leq i_1 < \dots < i_k \leq n$ where n is the number of degrees of freedom. The residual at iteration k of GMRES with a zero initial guess applied to the system

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \tag{3.36}$$

where $\boldsymbol{A}$ is a normal matrix with distinct eigenvalues and spectral factorization $\boldsymbol{A} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^\star$ and $\boldsymbol{b}$ is a right-hand side of unit norm is [Meurant and Tebbens, 2015]:

$$\left\|\boldsymbol{r}^{(1)}\right\|^2 = \frac{\sum_{I_2} \omega_{i_1}\omega_{i_2} \prod_{i_\ell < i_j \in I_2} \left|\lambda_{i_j} - \lambda_{i\ell}\right|^2}{\sum_{i=1}^{n} \omega_i \left|\lambda_i\right|^2} \quad \text{for} \quad k = 1, \tag{3.37}$$

$$\left\|\boldsymbol{r}^{(k)}\right\|^2 = \frac{\sum_{I_{k+1}} \left[\prod_{j=1}^{k+1} \omega_{i_j}\right] \prod_{i_\ell < i_j \in I_{k+1}} \left|\lambda_{i_j} - \lambda_{i_\ell}\right|^2}{\sum_{I_k} \left[\prod_{j=1}^{k} \omega_{i_j} \left|\lambda_{i_j}\right|^2\right] \prod_{i_\ell < i_j \in I_k} \left|\lambda_{i_j} - \lambda_{i_\ell}\right|^2} \quad \text{for all} \quad k = 2, \dots, n-1, \tag{3.38}$$

where $\omega_{i_j} = |\boldsymbol{e}_{i_j}^T \boldsymbol{c}|^2$ with $\boldsymbol{c} = \boldsymbol{\Phi}^\star \boldsymbol{b}$ and $\boldsymbol{e}_{i_j}$ is the i_j–th column of the appropriately sized identity matrix.

This theorem tells us that the more the eigenvalues are close to each other (the more they are clustered) the faster the convergence of GMRES. Moreover, the more the projection of the initial residual on the eigenvectors is spread the slower the convergence of GMRES. The best convergences occur when the initial residual can be precisely represented by a limited number of eigenvectors whose eigenvalues are similar. And the worst cases occur when the initial residual has a non-zero projection on all the eigenvectors which have disparate eigenvalues.

When we have the λ_i and ω_i, although the theorem above holds, it is infeasible to compute the convergence of a large problem using (3.37) and (3.38) due to the large number of computational operations. The cardinality of the set $\{I_k\}$ is

$$\#\left\{I_k\right\} = \sum_{i_1=1}^{n-k} \sum_{i_2=i_1+1}^{n-k+1} \dots \sum_{i_k=i_{k-1}+1}^{n} 1 = C_n^k, \tag{3.39}$$

which is intractable for large n with large k.

3.4.2 Insight into impact of ensemble reduction

We now consider the equivalent block diagonal system arising in ensemble GMRES with ensemble reduction as discussed in section 3.2:

$$\begin{bmatrix} \boldsymbol{A}_{::1} & & \\ & \ddots & \\ & & \boldsymbol{A}_{::s} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{:1} \\ \vdots \\ \boldsymbol{x}_{:s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_{:1} \\ \vdots \\ \boldsymbol{b}_{:s} \end{bmatrix}, \tag{3.40}$$

[0]A normal matrix is a matrix which commutes with its conjugate transpose.

where we assume now that $\boldsymbol{A}_{::1}$, …, $\boldsymbol{A}_{::s}$ are normal matrices.

Based on the formulas (3.37) and (3.38), we can deduce the impact of ensemble reduction on the convergence of ensemble GMRES. Due to the reduction, the spectrum of the block-diagonal matrix is the union of the spectra of the different samples of the ensemble and the projections of the normalized block right-hand side is the union of the projections of the right-hand sides of the different samples. If the sample variability is small enough, in other words, if the blocks $\boldsymbol{A}_{::1}$, …, $\boldsymbol{A}_{::s}$ are similar, the spectra of the different samples should be similar. Therefore, the convergence should not be impacted too much as long as the projections of the initial residual on the eigenvectors are similar too.

3.4.3 Illustration on a Dirichlet problem for the Laplacian

In order to illustrate the influence of ensemble reduction on the convergence of ensemble GMRES, we look into a Dirichlet problem for the Laplacian:

$$\begin{cases} -\kappa \dfrac{d^2u}{dx^2} & = b \quad \text{in} \quad]0,1[\\ u & = 0 \quad \text{at} \quad x = 0 \text{ and } x = 1 \end{cases}. \tag{3.41}$$

We discretize the domain $]0,1[$ using $n+1$ linear elements which leads to the linear system:

$$\boldsymbol{A}\,\boldsymbol{x} = \boldsymbol{b}, \tag{3.42}$$

where:

$$\boldsymbol{A} = \kappa \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{bmatrix}, \quad \boldsymbol{b} = h^2 \begin{bmatrix} b\,(x_1) \\ b\,(x_2) \\ \vdots \\ b\,(x_{n-1}) \\ b\,(x_n) \end{bmatrix}, \tag{3.43}$$

where $h = 1/(n+1)$.

We can deduce analytically the n eigenvectors $\boldsymbol{\phi}_1$, …, $\boldsymbol{\phi}_n$ which we store as columns in the matrix $\boldsymbol{\Phi}$ and the n associated eigenvalues λ_1, …, λ_n of the system (3.42):

$$\boldsymbol{\phi}_k = \begin{bmatrix} \sin(kh\pi) \\ \sin(2\,kh\pi) \\ \vdots \\ \sin((n-1)\,kh\pi) \\ \sin(n\,kh\pi) \end{bmatrix} \quad \text{for all} \quad k = 1, \dots, n, \tag{3.44}$$

$$\lambda_k = 2\,\kappa\,(1 - \cos(kh\pi)) \quad \text{for all} \quad k = 1, \dots, n. \tag{3.45}$$

Given a right-hand side with unit norm and a zero initial guess, we can now compute the convergence of GMRES applied to the system (3.42) with the equations (3.37) and (3.38) as we have λ_1, …, λ_n, $\boldsymbol{X}$, and $\boldsymbol{b}$.

We provide numerical results for $n = 6$ and four test cases:

- Test case (1): $\kappa = 1$ and $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$,

- Test case (2): $\kappa = 1$ and $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$,
- Test case (3): $\kappa = 1.5$ and $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$,
- Test case (4): $\kappa = 1.5$ and $\boldsymbol{b}^{\mathrm{T}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$.

Tests (1) and (2) share the same spectrum as they share the same value of κ. However their projections of $\boldsymbol{b}$ on the eigenvectors are different as $\boldsymbol{b}$ is changed. The tests (1) and (3), although they do not have the same spectrum as κ has changed, share the same projection of $\boldsymbol{b}$ on the eigenvectors as $\boldsymbol{b}$ is the same and as $\boldsymbol{\Phi}$ is not influenced by κ.

For those four test cases, the eigenvalues $\lambda_1, \ldots, \lambda_n$ and the weights $\omega_1, \ldots, \omega_n$ can be computed and stored into vectors:

- Test case (1):

$$\boldsymbol{\lambda}^{\mathrm{T}} = \begin{bmatrix} 0.198 & 0.753 & 1.555 & 2.445 & 3.247 & 3.802 \end{bmatrix}, \tag{3.46}$$
$$\boldsymbol{\omega}^{\mathrm{T}} = \begin{bmatrix} 0.272 & 0.054 & 0.175 & 0.175 & 0.054 & 0.272 \end{bmatrix}, \tag{3.47}$$

- Test case (2):

$$\boldsymbol{\lambda}^{\mathrm{T}} = \begin{bmatrix} 0.198 & 0.753 & 1.555 & 2.445 & 3.247 & 3.802 \end{bmatrix}, \tag{3.48}$$
$$\boldsymbol{\omega}^{\mathrm{T}} = \begin{bmatrix} 0.054 & 0.175 & 0.272 & 0.272 & 0.175 & 0.054 \end{bmatrix}, \tag{3.49}$$

- Test case (3):

$$\boldsymbol{\lambda}^{\mathrm{T}} = \begin{bmatrix} 0.297 & 1.130 & 2.332 & 3.668 & 4.870 & 5.703 \end{bmatrix}, \tag{3.50}$$
$$\boldsymbol{\omega}^{\mathrm{T}} = \begin{bmatrix} 0.272 & 0.054 & 0.175 & 0.175 & 0.054 & 0.272 \end{bmatrix}, \tag{3.51}$$

- Test case (4):

$$\boldsymbol{\lambda}^{\mathrm{T}} = \begin{bmatrix} 0.297 & 1.130 & 2.332 & 3.668 & 4.870 & 5.703 \end{bmatrix}, \tag{3.52}$$
$$\boldsymbol{\omega}^{\mathrm{T}} = \begin{bmatrix} 0.054 & 0.175 & 0.272 & 0.272 & 0.175 & 0.054 \end{bmatrix}. \tag{3.53}$$

First the convergence of each test alone are computed using (3.37) and (3.38) and are illustrated in Fig. 3.2. We observe that the tests (1) and (3) have the same convergence despite the fact the spectra are different. The same behavior can be observed for the tests (2) and (4). This is explained by the equations (3.37) and (3.38), although the spectra are different they are the same up to a constant multiplier. If all the eigenvalues are multiplied by α, we have obviously that $\left\|\boldsymbol{r}^{(1)}\right\|^2$ is not impacted due to (3.37). We have the same property for $\left\|\boldsymbol{r}^{(k)}\right\|^2$ too and it can be deduced from (3.38) that the numerator has been multiplied by $(\alpha^2)^{C_{k+1}^2}$ and the denominator by $(\alpha^2)^k\,(\alpha^2)^{C_k^2}$ which is the same due to the fact that:

$$C_{k+1}^2 = \frac{(k+1)!}{2!(k-1)!} = \frac{k\,(k+1)}{2} = \frac{k\,(k-1)}{2} + k = \frac{k!}{2!(k-2)!} + k = C_k^2 + k. \tag{3.54}$$

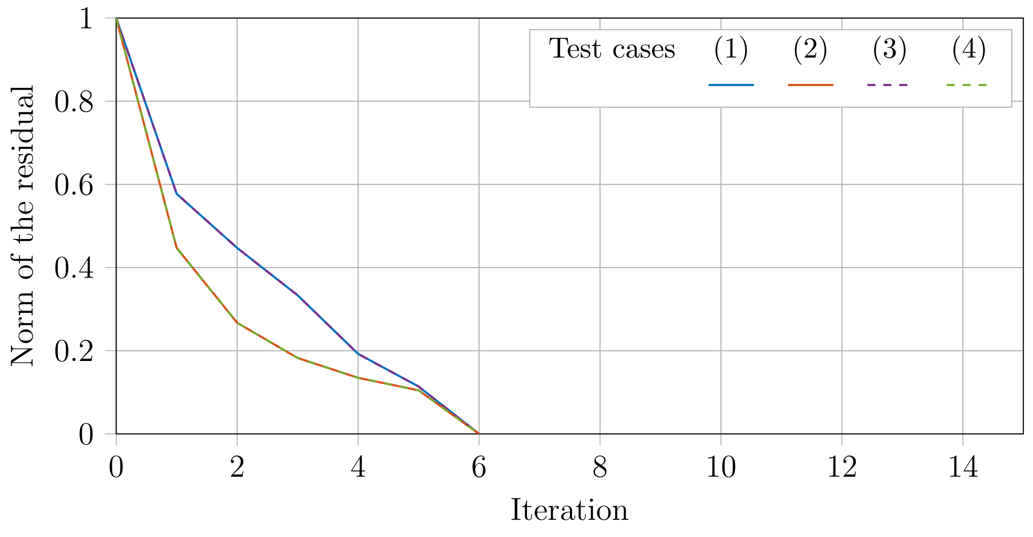


Figure 3.2: Convergence of GMRES for each test alone.

A second simpler interpretation of this result can be given based on the Krylov subspace. Multiplying the matrix by a non-zero scaling factor α impacts the Krylov subspsace as follows:

$$\mathcal{K}_m(\alpha \boldsymbol{A}, \boldsymbol{r}^{(0)}) = \text{span}\left\{\boldsymbol{r}^{(0)}, \alpha\, \boldsymbol{A}\, \boldsymbol{r}^{(0)}, \dots, \alpha^{m-1}\, \boldsymbol{A}^{m-1}\, \boldsymbol{r}^{(0)}\right\} \equiv \mathcal{K}_m(\boldsymbol{A}, \boldsymbol{r}^{(0)}), \tag{3.55}$$

in other words, the Krylov subspace of the scaled matrix is the same as the Krylov subspace of the initial matrix. This results in the fact that the norm of the residual at an iteration k is independent of the scaling factor α due to the equation (3.5).

From the convergence curves of each test alone, we observe that $\boldsymbol{b}$ influences the convergence due to its projection on the eigenvectors as test cases (1) and (2) do not share the same convergence. The test case (2) converges faster because its weights ω_1, …, ω_n are more clustered than test (1), i.e. the eigenvalues with the larger weight are closer in the case of the test (2).

We can now look on how ensemble reduction influences GMRES. To do so, we couple two test cases into an ensemble of two samples, we gather their n eigenvalues in a vector of $2\,n$ eigenvalues, and compute the $2\,n$ projections of the normalized gathered right-hand sides.

The convergence curves are then computed (Fig. 3.3). The first observation is that gathering (1) and (2) into one ensemble (1,2) (or equivalently (3) and (4)), we gather the same spectrum and, as we cannot find more than n different values of λ_k, GMRES converges in n iterations. The convergence of the ensemble reduced norm is included between convergence curve of (1) and (2) as we now look into the convergence of the root mean square of the residual of the samples. This illustrates the impact of ensemble reduction on the convergence of GMRES through the gathering of $\boldsymbol{\omega}$ alone.

The second observation is that coupling samples (1) and (3) (or equivalently (2) and (4)) increases the number of iterations to reach a zero residual and increases the norm of the residual at any fixed iteration as, now, there are $2\,n$ different eigenvalues. This illustrates the impact of ensemble reduction on the convergence of GMRES through the coupling of the spectra alone.

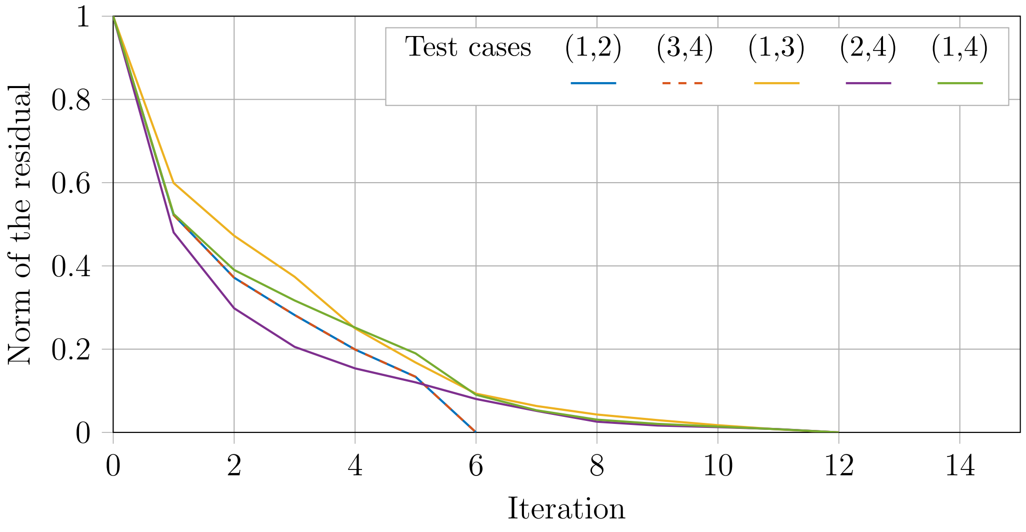


Figure 3.3: Convergence of ensemble GMRES with reduction.

Coupling samples (1) and (4) together combines the two effects: we have $2\,n$ different eigenvalues and different $\boldsymbol{\omega}$. However the predominant effect here is linked to the aggregation of the spectra. The second effect can be isolated comparing the curves (2,4) and (1,4); they both have the same aggregated spectrum but different $\boldsymbol{\omega}$; they both converge to 0 slower but (2,4) converges faster as (1) converges slower than (4) and (2).

3.5 Conclusions

In this chapter, we have introduced ensemble GMRES both with and without ensemble reduction to solve the parametric linear system of Chapter 2 with ensemble propagation. We have discussed the potential occurrences of ensemble divergence in GMRES and how both approaches treat them.

The originality of this chapter is that, we have formally reviewed the existing ensemble GMRES with ensemble reduction with tensor notations, formally introduced ensemble GMRES without ensemble reduction with tensor notations, and highlighted ensemble divergence in ensemble GMRES. Those discussed occurrences of ensemble divergence are more complex than the occurrences of ensemble divergence in CG due to the need of BLAS functions for ensemble type such as the GEMV and the impact of the lucky breakdown on GMRES as discussed in section 3.3. Moreover, we have discussed, using a numerical example and theoretical convergence results, how ensemble reduction influences the convergence of ensemble GMRES.

In Chapter 4, the implementation work required for ensemble GMRES without ensemble reduction is discussed. In Chapter 7, both approaches, ensemble GMRES with and without reduction, are compared both in terms of speed-up and impact on the number of iterations to converge.

Chapter 4

Efficient ensemble GMRES without ensemble reduction

In this chapter, we will discuss how the code divergence highlighted in the previous chapter is treated. In particular, we will discuss masked assignment, logical reduction, and function-call divergence in the context of the template-based generic programming approach as discussed in Chapter 2.

Section 4.1 describes more precisely the implementation context of ensemble GMRES without ensemble reduction in Trilinos.

Section 4.2 is dedicated to the General Matrix Vector product (GEMV) used in lines 6 and 7 of Algo. 4. An efficient GEMV implementation for ensemble type is necessary for the efficiency of lines 6 and 7 of Algo. 4. In section 4.2.1, we discuss the update process of the orthogonalization, i.e. line 7 of Algo. 4. In section 4.2.2, we discuss the inner products of the orthogonalization, i.e. line 6 of Algo. 4.

In section 4.3, the control-flow divergence has been tackled using masks, a data type that stores an array of booleans which are the results of comparison operators sample-wise and that can be used to do masked assignment, in other words, conditional assignment, or logical reduction.

In this chapter, to alleviate the text, we use *GMRES* instead of *ensemble GMRES* and *reduction* instead of *ensemble reduction*.

4.1 Implementation context

The presented GMRES without reduction has been implemented in this thesis in the Trilinos library introduced in section 1.1.5. This implementation work relies on the existing templated GMRES of Belos.

During the orthogonalization process, Belos calls the Tpetra package to compute the inner products and the update of lines 6 and 7 of Algo. 4 respectively. Tpetra relies on the GEMM of KokkosKernels for the on-node parallelism of those operations. When using the ensemble propagation without reduction, the GEMM of KokkosKernels calls the GEMV of KokkosKernels if the left-hand side has only one column as in our case.

In this context, we have observed that the performance of the default GEMV of KokkosKernels, the templated package which provides templated interface to optimized implementations of kernel functions as discussed in section 1.1.5, was not optimal for the ensemble types on the tested architecture as illustrated later in this chapter. Therefore, and in order to be able to compare GMRES without reduction with GMRES with reduction, which uses optimized implementations of the GEMV such as the one provided by the MKL, we study, propose, and implement an efficient GEMV for ensemble types in this work.

The implementations related to GEMV discussed in this chapter are made as a C++ template specialization of the GEMV function of the KokkosKernels package from Trilinos. This way, the Tpetra function calls made by Belos to compute lines 6 and 7 of Algo. 4 are unchanged. Those implementations are based on the Kokkos programming model [Edwards et al., 2012, 2014] discussed in the introduction.

GMRES without reduction has been implemented in a template specialization of classes of Belos to introduce the required modifications as discussed in section 3.3. In order to tackle the control-flow divergence, we investigated an alternative to the `EnsembleTrait` proposed by Phipps et al. [2017] and described in section 4.3.1. This alternative is to define and use masks for ensemble types as discussed in section 4.3.3 with some inspiration from the work of Kretz and Lindenstruth [2012]. Those masks improve the readibility of the code, make the maintenance easier, and are better suited to the autovectorization, or can be mapped to vector instructions.

The ensemble sparse matrix-vector product is not discussed in this chapter as the ensemble sparse matrix-vector product used in ensemble GMRES is the one used in the ensemble conjugate gradient method which was already implemented.

All the implementations discussed in this chapter, except the implementation of the mask based on the Intel Intrinsics, have been made available in Trilinos. The implementation of the mask based on the Intel Intrinsics will be included in Trilinos later. There is some discussion about using a common SIMD data type for different Trilinos packages. This data type should provide a mask implementation and an interface to the Intel Intrinsics. We are currently waiting the output of this discussion before including the implementation of the mask based on the Intel Intrinsics.

4.2 Efficient dense matrix-vector product for ensemble propagation

In this section we are interested in the efficient computation of the generalization of the scalar-typed General Matrix Vector product (GEMV) to ensemble-typed GEMV which can be written as a tensor contraction:

$$\mathbf{w}_{:\ell} = \alpha_\ell \, \mathbf{w}_{:\ell} + \beta_\ell \, \mathbf{C}_{::\ell} \, \mathbf{z}_{:\ell} \quad \text{for all} \quad \ell = 1, \ldots, s, \tag{4.1}$$

where tensors are stored using the interleaved memory layout as illustrated in Fig. 2.4 for $\mathbf{W}$ and $\mathbf{Z}$. Such a computation is limited by the memory bandwidth as its arithmetic intensity, which is of 2 flops per loaded $c_{ik\ell}$, is significantly smaller than the machine flops/byte ratio of most commonly available systems. It is worth emphasizing the fact that the third-order tensor $\mathscr{C}$ of (4.1) is dense. In the case of GMRES without reduction $\mathscr{C}$ of (4.1) is the tensor used to store the Arnoldi vectors. Such a BLAS operation with interleaved memory layout is known as Compact BLAS [Kim et al., 2017] and has been initially developed to reduce the wall-clock time of computing the BLAS/LAPACK routines on large groups of very small matrices.

As the scalar-typed GEMV is memory bound and as its implementations such as in the MKL or in the BLIS [Van Zee and Van De Geijn, 2015] are highly optimized, we cannot expect to reach ensemble propagation speed-up greater than 1 for the GEMV. However, having an optimized implementation of the tensor contraction for ensembles remains essential not to deteriorate ensemble propagation speed-up of GMRES without reduction in comparison to the speed-up of GMRES with reduction; the latest uses optimized BLAS implementations which reach theoretical throughput. To be comparable with GMRES with reduction, GMRES without reduction requires optimal implementations of the kernel functions.

As the GEMV is limited by the memory bandwidth, we present here measured memory bandwidth using the STREAM Triad Memory Bandwidth benchmark [McCalpin, 1995], which consists in computing $\boldsymbol{a} = \boldsymbol{b} + q\,\boldsymbol{c}$ where $\boldsymbol{a}$, $\boldsymbol{b}$, and $\boldsymbol{c}$ are vectors and q is a scalar, on the considered architecture: one compute node with 2 Intel(R) Xeon(R) Platinum 8160 CPUs which have 24 cores per CPU. This test has been performed with a fixed problem size of 2.4 GB, different numbers of threads with compact affinity and using hyper-threading, and different vector instructions and is illustrated in Fig. 4.1. The best measured memory bandwidth are 101.2108 GB/s and 199.7486 GB/s while using 1 and 2 CPU respectively. Because we are using the default first touch policy to control the memory placement and because the threaded loops in the STREAM benchmarks are simple enough, we place ourselves in a case where the threads access only local memory inside the threaded loops. This implies that all the threads of the CPU 1 access a first NUMA region whereas all the threads of the CPU 2 access a second NUMA region. The results of Fig. 4.1 confirm the importance of using enough cores and vectorization.

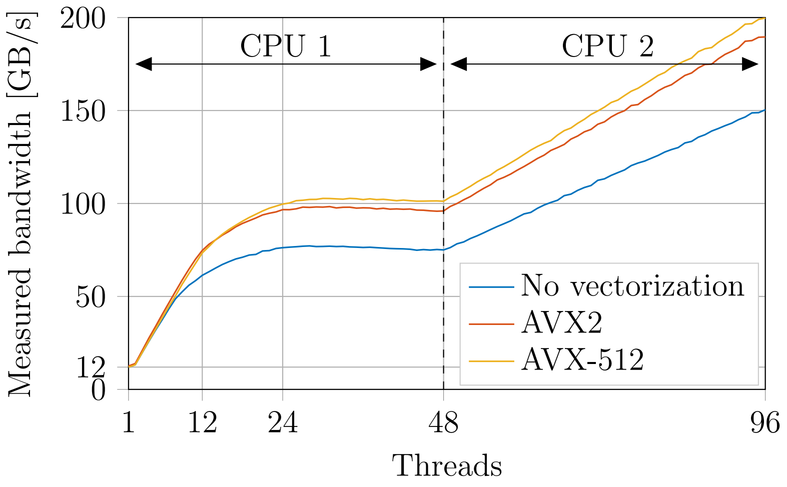


Figure 4.1: Benchmarked STREAM Triad Memory Bandwidth on a Skylake node with 2 Intel(R) Xeon(R) Platinum 8160 CPU with a problem size of 2.4 GB with a maximum of $24 \times 2 \times 2$ threads.

For the remainder of this work, we will restrict ourselves to cases where only one NUMA region is used per MPI process. This way, we remove the difficulty of defining in which NUMA region the entries of the tensors are allocated. The 2 NUMA regions can still be used using 2 MPI process and binding them to a particular region.

With that restriction in mind, from the maximal measured bandwidth, the fact that a double type requires 8 bytes to be stored, and that 2 operations (one addition and one multiplication) are computed per loaded $c_{ik\ell}$, we deduce that the maximal throughput that we can expect is about $101 \times 2/8 \approx 25$ GFLOPS.

In the literature about dense matrix-matrix multiplications in BLAS implementations

[Goto and van de Geijn, 2008; Smith et al., 2014; Van Zee and Van De Geijn, 2015], among other discussions, strategies are discussed to improve the memory usage of dense matrix-matrix multiplications.

By analyzing the implementations discussed in those papers for dense matrix-matrix products, we deduced that there are three key aspects to reach the maximal theoretical throughput in the case of dense matrix-vector products. First, the code has to exploit as much as possible the parallelism of the architecture as the memory bandwidth grows with the number of threads used. Second, transfers through the memory hierarchy should be done as efficiently as possible using unit-stride loads and reusing data from the cache. Third, vector instructions have to be used correctly, for instance the gather-scatter memory addressing should be avoided.

Although computing sequentially s GEMVs with scalar-typed inputs is feasible and will give the correct outputs, the performance of this approach will be bad. Due to the interleaved memory layout of $\mathscr{C}$, we will not be able to load $\mathbf{C}_{::\ell}$ for a given ℓ with a unit stride. Therefore, we must write an efficient implementation of the tensor contraction compatible with the interleaved memory layout which is able to use unit stride reading, as many threads as possible, and as efficiently as possible the vector instructions.

Two different GEMVs occur in GMRES in lines 6 and 7 of Algo. 4, the inner products and the update for which the corresponding matrices $\mathbf{C}_{::\ell}$ are short fat and tall skinny matrices respectively. As those two cases are different, we have chosen to implement two algorithms, one for each case.

The strategies described and used in this section can be used for the implementation of GEMV for other data types providing that the tile size, a parameter introduced in this section, is changed accordingly based on the size of the data type.

4.2.1 Case of the update

In this subsection, we will restrict ourselves to the case of the update of line 7 of Algo. 4. Therefore, we are interested in the tensor contraction of the form (4.1) where the third order tensor $\mathscr{C}$ has the dimension $n \times j \times s$ and is stored with the following layout:

$$c_{ik\ell} \hookleftarrow c\left[(i-1)\,s + (k-1)\,n\,s + \ell\right], \tag{4.2}$$

meaning that two ensembles $\boldsymbol{c}_{ik:}$ and $\boldsymbol{c}_{(i+1)k:}$ are stored contiguously in the memory. Moreover, the first dimension, n, in our case, is the number of degrees of freedom per sample, which is expected to be high and is fixed, j is the dimension of the current Krylov subspace, which is increased by one at every iteration of GMRES and is expected to be significantly smaller than n, and s is the ensemble size which is fixed by the user. This so-called left layout has been chosen in the Belos package to store the Arnoldi vectors as it has the advantages that adding a new Arnoldi vector does not modify the memory addresses of the previously allocated vectors as j does not appear in (4.2).

First, we review the default implementation of the GEMV of the KokkosKernels package. This implementation is illustrated in Algo 5 where **parfor** stands for a threaded loop. The algorithm consists in a parallel loop over the rows, a loop over the columns, and a loop over the samples. The memory access pattern of this implementation is illustrated in Fig. 4.4a. Due to the memory layout of the third order tensor $\mathscr{C}$, the memory access of $\mathscr{C}$ in line 4 of Algo 5 are not ideal for CPU as $\boldsymbol{c}_{ik:}$ and $\boldsymbol{c}_{i(k+1):}$ are not stored contiguously in the memory. For small ensemble size, this results in jumps in the memory which are not amortized by a sufficiently large amount of streamed memory to load $\boldsymbol{c}_{i(k+1):}$. However,

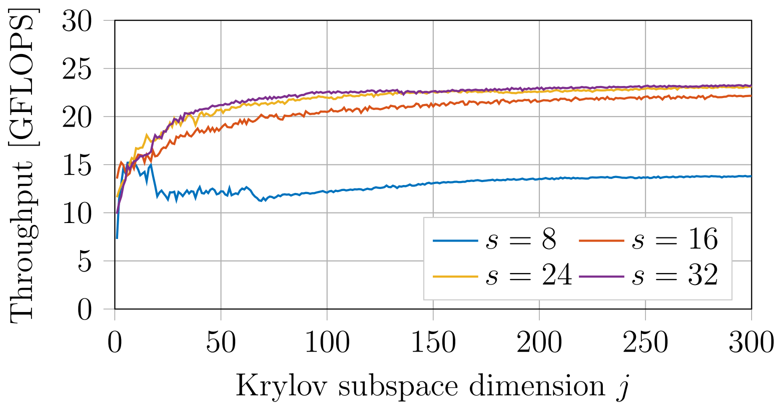


Figure 4.2: Throughput of the update in GMRES when using the default implementation of the GEMV of the KokkosKernels package.

if s is large, this effect is reduced: the ratio between the number of memory jumps and the size of loaded memory decreases.

Algorithm 5: Update: default implementation of KokkosKernels

```
1 parfor $i = 1$ to $n$ do
2     $w_{i\ell} = \alpha_\ell \, w_{i\ell}$   for all   $\ell = 1, \dots, s$
3     for $k = 1, \dots, j$ do
4         $w_{i\ell} = w_{i\ell} + \beta_\ell \, c_{ik\ell} \, z_{k\ell}$   for all   $\ell = 1, \dots, s$
```

Fig. 4.2 illustrates the throughput of the update of GMRES on a problem of size $n = 32000$ with the default implementation of the KokkosKernels. We observe that the performance of the default implementation nearly reaches the theoretical limit for ensemble of size of at least 16. However, for ensembles of size 8, the performance of the default implementation reaches 60% of the theoretical limit. This is due to the fact that the jump in the memory address moving from $\boldsymbol{c}_{ik:}$ to $\boldsymbol{c}_{i(k+1):}$ is not amortized over a sufficiently large load as discussed earlier.

In this work, we propose an alternative implementation of the GEMV which improves the memory access for the considered memory layout. We have used a cache blocking strategy to reuse entries of the left-hand side matrix $\mathbf{W}$ from cache using a tiling strategy. The algorithm is listed in Algo. 6 and is illustrated in Fig. 4.3. This algorithm loops over the tiles, over the columns, over the rows, and over the samples. The memory access pattern of this implementation is illustrated in Fig. 4.4b.

Algorithm 6: Update: threaded tiled approach

```
1 parfor $t = 1$ to $n - n_c + 1$ by $n_c$ do                    Over the tiles
2     for $i = t, \dots, t + n_c - 1$ do
3         $w_{i\ell} = \alpha_\ell \, w_{i\ell}$   for all   $\ell = 1, \dots, s$
4     for $k = 1, \dots, j$ do                                  Over the columns
5         $\gamma_\ell = \beta_\ell \, z_{k\ell}$   for all   $\ell = 1, \dots, s$
6         for $i = t, \dots, t + n_c - 1$ do                    Over the rows
7             $w_{i\ell} = w_{i\ell} + \gamma_\ell \, c_{ik\ell}$   for all   $\ell = 1, \dots, s$
```

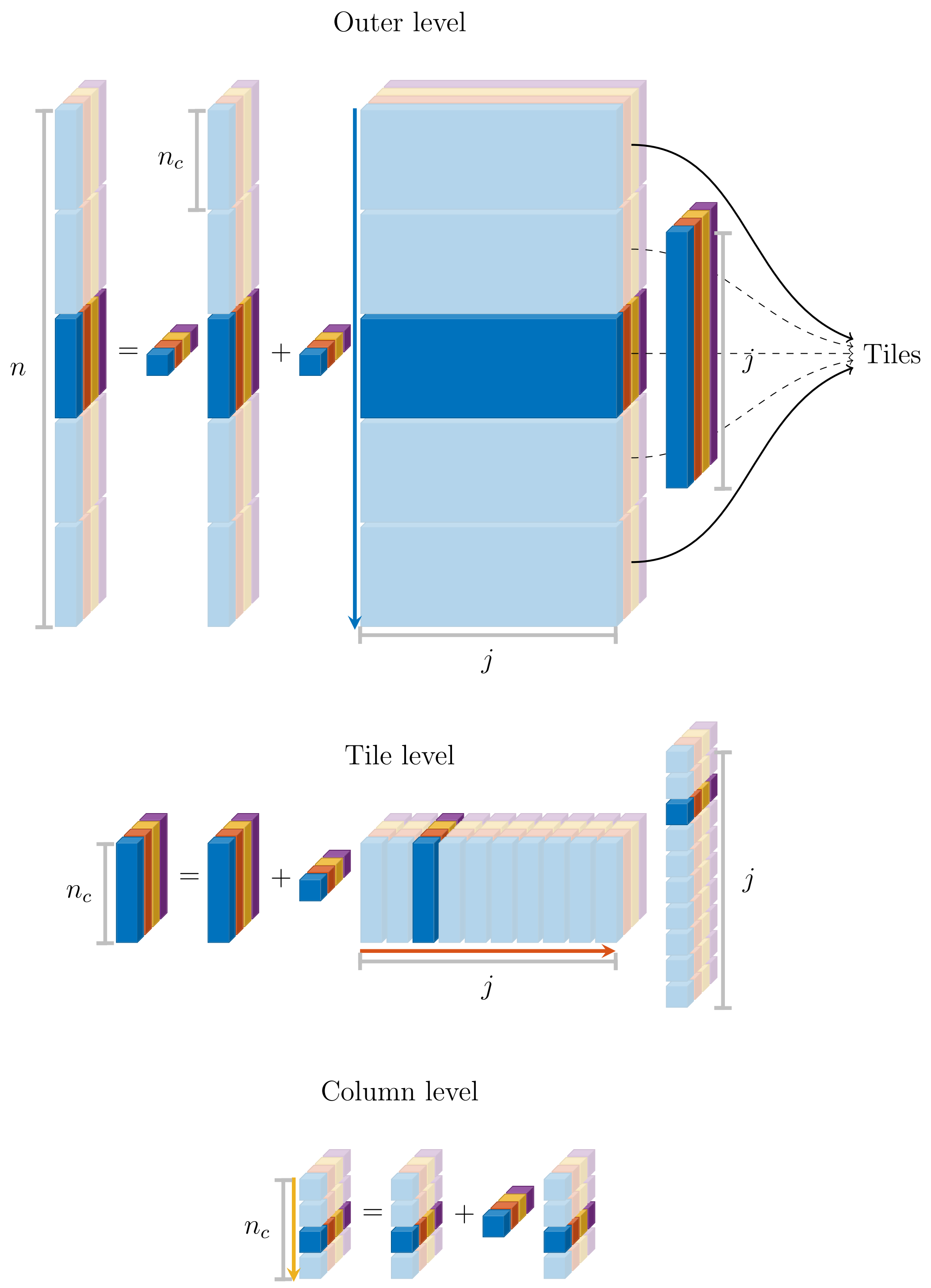
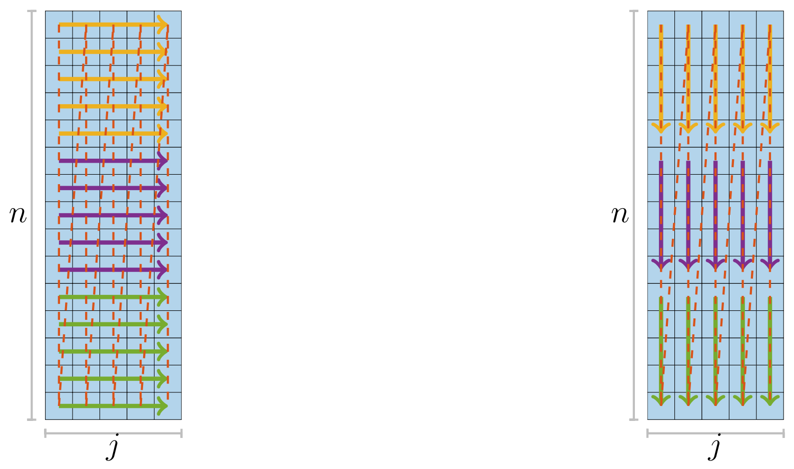


Figure 4.3: Illustration of the threaded tiled approach of the GEMV for the case of the update of line 7 of Algo. 4 with ensembles of size 4.



(a) Default GEMV. (b) Tiled approach.

Figure 4.4: Illustration of the memory access patterns to read $\mathbf{C}_{::\ell}$ from the main memory with the default GEMV and the tiled approach with 3 threads. The arrows represent the memory access pattern of a thread (one color per thread). The dashed red line represents the left layout. The default GEMV suffers from jumps in the memory address moving from one column to the next one while the tiled approach follows the layout.

This algorithm splits the leading dimension of size n into tiles of size at most n_c. Such a strategy is called *strip mining* and allows cache blocking of $\mathbf{W}_{(t:t+n_c-1)(1:s)}$. We have chosen to parallelize this leading dimension because it was the one that, being the largest one, has the most opportunities for threading and because this dimension does not vary during the simulation. We have chosen to parallelize it with tiles instead of simply the rows as for any entry $c_{ik\ell}$ there is no entry into the tube fiber $\mathbf{c}_{ik:}$ such that they are contiguous in the memory; in other words, we cannot have a load with a unit stride longer than s while looping on the entries of a given row for a given thread. This is not the case when we are using tiles, we have unit-stride loads of $n_c s$ entries for a given thread. Increasing n_c allows larger unit-stride loads but increases the cache pressure; the choice of n_c is a trade-off between those considerations.

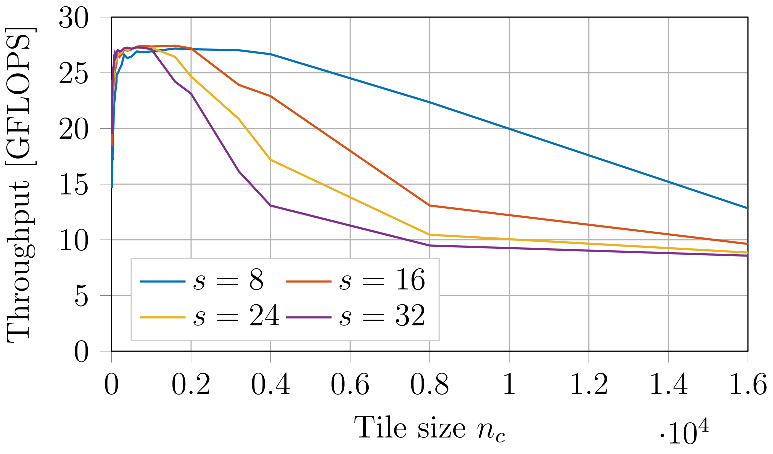
Whereas usual dense-matrix functions require vectorization to be introduced explicitly for the GEMV of standard data type, we inherit opportunities for the vectorization from ensemble propagation and the fact that the loop over the samples is the innermost loop in Algo. 6. On the tested modern architecture with modern compilers, we observe that the autovectorized assembly code of the ensemble GEMV is performing optimally.

Ensemble propagation impacts this algorithm in three ways: the ensemble size s affects the memory use per tile and, therefore, the tile size n_c has to be inversely proportional to s to keep $\mathbf{W}_{(t:t+n_c-1)(1:s)}$ in cache, ensemble propagation adds an extra loop over the samples ℓ, and the vectorization of the ensemble loop takes place at the elementary operation levels.

First, we have to choose the values of the tile size n_c for different ensemble sizes for the considered architecture. To deduce n_c we have to consider both the memory considerations and the requirement of having all the threads computing as often as possible. We will first deduce values of n_c from examples where all the threads are used and, then, explain how to correct n_c to enforce the second consideration.

In order to do so, we have evaluated the tensor contraction for a fixed size $n \times j \times s$ of the third order tensor $\mathscr{C}$ where $n = 2 \times 24 \times 16000 = 768000$, $j = 300$, and $s = 8, 16, 24$, and 32, and have evaluated the throughput as functions of the tile size n_c and the ensemble size s. In order to remove the impact of unevenly distributed work among the threads, we have only evaluated n_c such that n is a multiple of Nn_c where $N = 48$ is the number of threads. The results are illustrated in Fig. 4.5.

We perform those tests on the above-mentioned Skylake CPU which has a 1 MB L2 cache per core and 2 threads per core, therefore, using 2 threads per core, each thread can at most store $65536 = 1024 \times 1024/8/2$ `double` in the L2 cache.



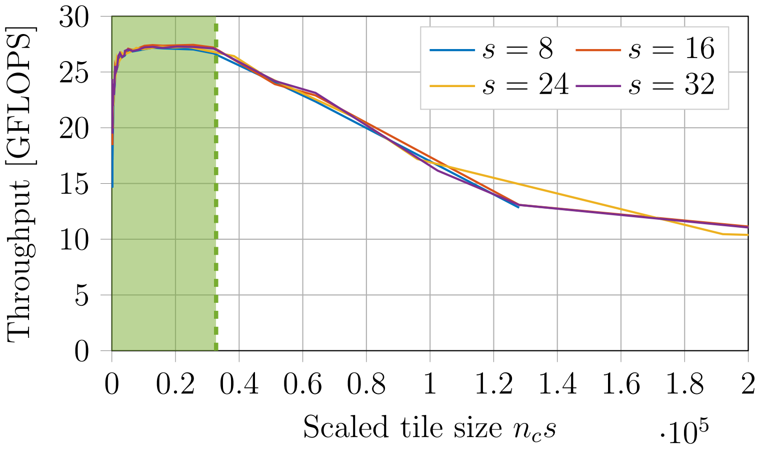


Figure 4.5: Influence of n_c on the throughput of the proposed implementation of the GEMV, the vertical dashed line represents the value of $n_c^{\max} s = 32768$, which is half of the maximum number of `double` that can be stored in L2 cache, and the shading represents the n_c such that $n_c \leq n_c^{\max}$. The scaled tile size corresponds to the multiplication of the tile size n_c by the ensemble size s.

We choose the scaled $n_c s$ in such a way that the reused data $\mathbf{W}_{(t:t+n_c-1):}$ uses at most about half of the smallest value among the memory addressable by the TLB and the size of the L2 cache as described in the practical rule of Goto and van de Geijn [2008]. Therefore, for a given ensemble size s, we have that the maximal tile size $n_c^{\max}$ is given by:

$$n_c^{\max} s \leq \frac{65536}{2}, \tag{4.3}$$

where the value 2 comes from the practical rule of Goto and van de Geijn [2008]. The corresponding $n_c^{\max}$ are listed in Table 4.1.

s	8	16	24	32
$n_c^{\max}$	4096	2048	1365	1024
$n_c^{\max}s$	32768	32768	32760	32768

Table 4.1: Maximal tile size $n_c^{\max}$ for the different ensemble sizes. In the case of $s = 24$, 32768 cannot be divided by s leading to a scaled $n_c^{\max}s$ which is not 32768.

From the results of Fig. 4.5, we see that increasing the size of the unit-stride load $n_c s$ improves the throughput up to a certain $n_c^{\max}s = 32768$ where the cache pressure starts to reduce the throughput. In particular, if we increase $n_c s$ beyond that threshold, the throughput tends to half of the maximal throughput as both the entries of $\mathscr{C}$ and $\mathbf{W}$ have to be streamed from the main memory.

The results of Fig. 4.5 show that the ensemble size s influences the optimal values of n_c; the larger the ensemble size s the larger the memory to store the tile.

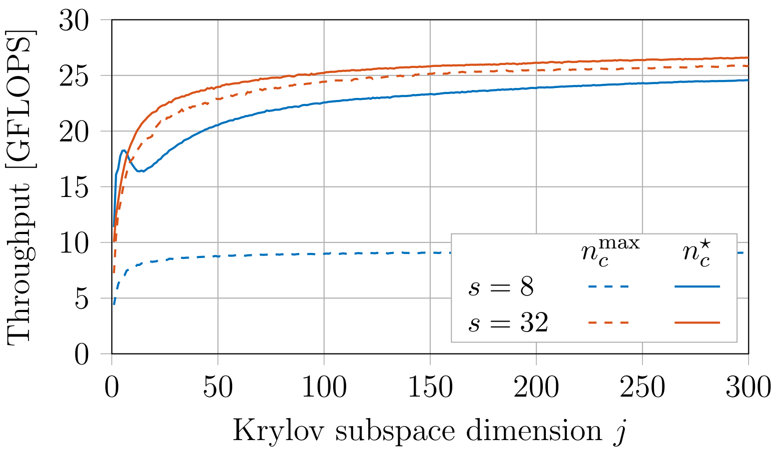


Figure 4.6: Impact of the correction (4.4) on the throughput of the update in GMRES.

Due to the fact that both the products of the values of $n_c^{\max}$ and the number of available threads are relatively large, especially for small ensemble size, choosing $n_c = n_c^{\max}$ leads to unevenly distributed work among the threads for small n. As a consequence, n_c cannot be independent of n if we want to use evenly all the threads. Given a maximal admissible tile size $n_c^{\max}$ which is limited by the cache size as discussed above, we compute a smaller $n_c^{\star}$ which distributes evenly the n rows among the N threads as follows:

$$n_c^{\star} := \left\lceil \frac{n}{N \left\lceil \frac{n}{N\, n_c^{\max}} \right\rceil} \right\rceil, \tag{4.4}$$

where $\lceil x \rceil$ is the least integer greater than or equal to x.

This way, we have that each thread will have the same number of tiles which is $\left\lceil \frac{n}{N\, n_c^{\max}} \right\rceil$. Although $n_c^{\star}$ is unknown, $n_c^{\star}$ is necessarily smaller than the maximal admissible tile size $n_c^{\max}$ due to (4.4) which enforces the cache blocking and is necessarily in the shaded region of Fig. 4.5.

We show the impact of that correction on an example where we apply the update of GMRES on a problem of relatively small size $n = 30000$ for different Krylov subspace dimensions, for ensemble size $s = 8$ and $s = 32$ with and without the correction (4.4).

The throughput of that example is shown in Fig. 4.6. We see that the correction improves the throughput for the two ensemble sizes but the effect is more important for smaller ensemble sizes as expected.

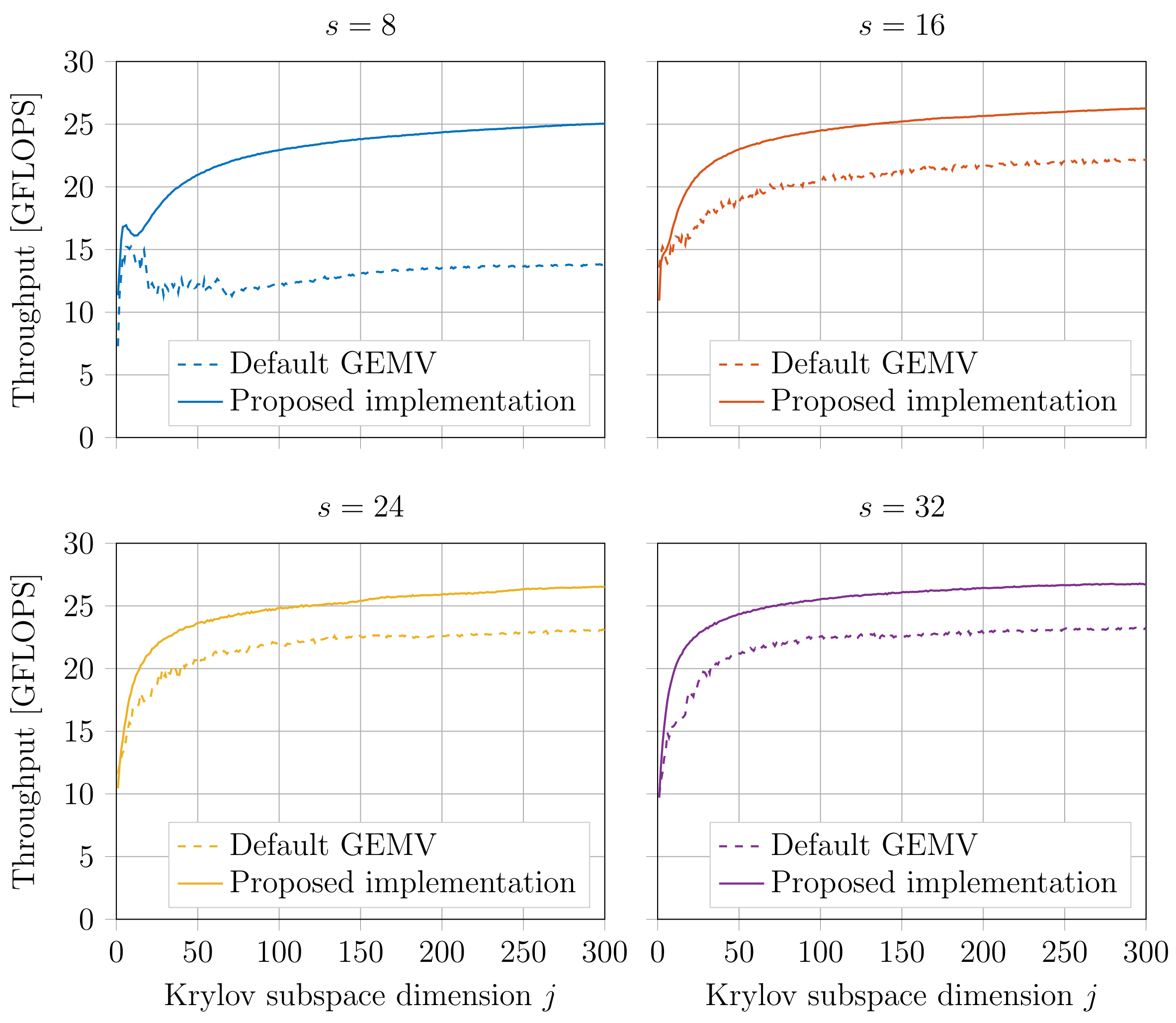


Figure 4.7: Comparison of throughput of the update in GMRES when using the default Trilinos implementation ($n_c = 1$) and our tiled approach ($n_c = n_c^\star$).

Finally, we compare in Fig. 4.7 the throughput of the update of GMRES on a problem of size $n = 32000$ with the default implementation and our proposed implementation with $n_c^{\max}$ chosen as in Table 4.1. The KokkosKernels default GEMV implementation is similar to the one presented in this thesis with $n_c = 1$. On the tested cases, the throughput of our implementation is larger than the throughput of the default KokkosKernels GEMV due to the improved memory access and is less sensitive to the ensemble size.

In order to prove that the choice of $n_c^{\max} s$ is related to L2 cache size, we have measured the cache misses of all cache levels during the numerical experiment shown in Fig. 4.5. Those misses have been measured with PAPI [Terpstra et al., 2010] which provides a portable tool for use of the performance counter hardware such as cache misses and hits. The measured data are shown in Fig. 4.8. We can observe that, for large values of the $n_c s$, the cache misses of all levels tend to $1.2\,10^6$ per sample which corresponds to the case where the entries of $\mathbf{W}_{(t:t+n_c-1):}$ are not reused from cache and have to be reloaded from main memory. However, for smaller values of $n_c s$, we observe that both L2 and L3

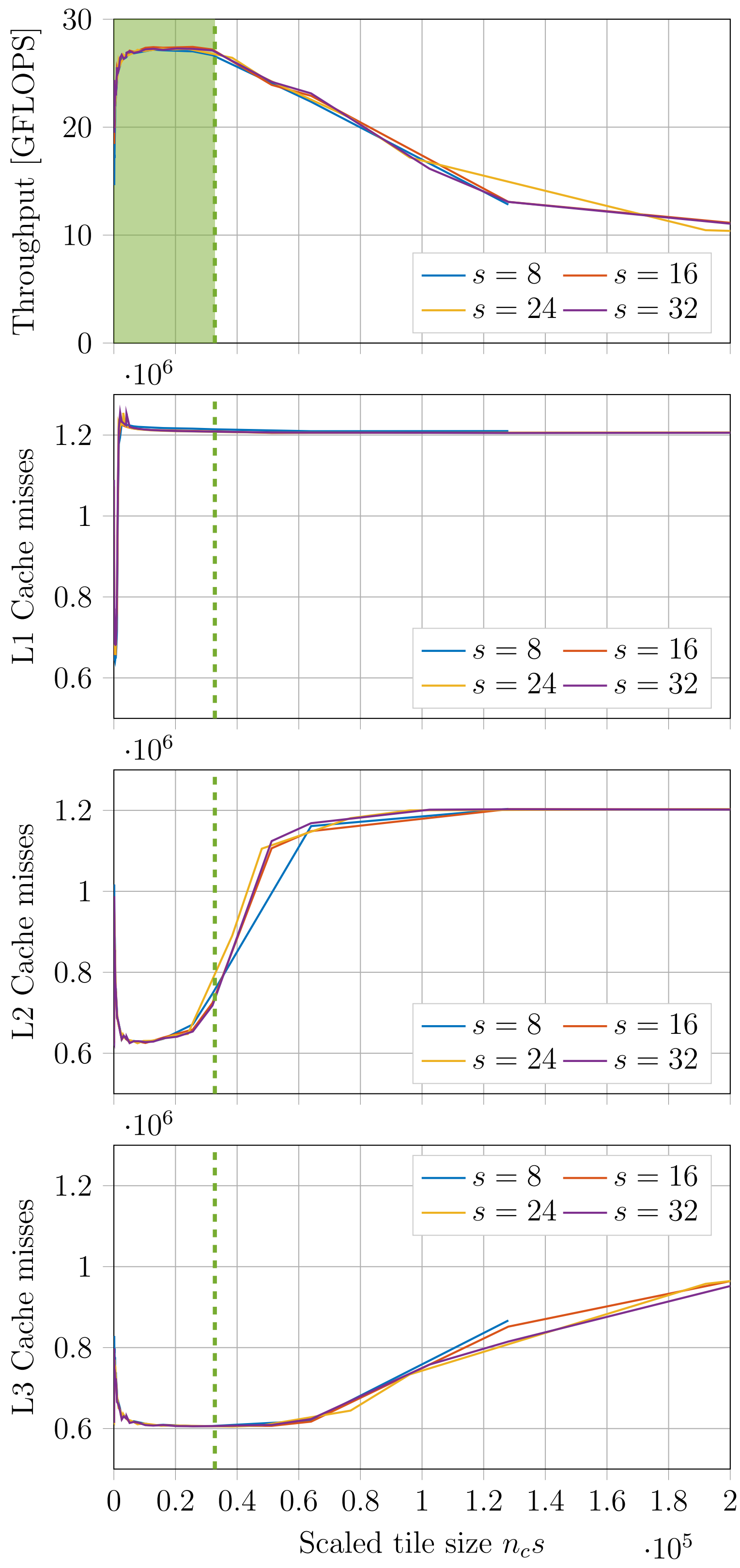


Figure 4.8: Influence of the scaled $n_c s$ on the L1, L2, and L3 cache misses per sample. For large values of the scaled $n_c s$, the cache misses of all level tend to the same value because the entries of $\mathbf{W}_{(t:t+n_c-1):}$ are not reused from cache. For smaller values, we see a correlation between L2 cache misses and the throughput of the tiled algorithm.

cache misses are about $0.6e6$ per sample which is consistent with the fact that entries of $\mathbf{W}_{(t:t+n_c-1):}$ are reused from cache. Finally, we observe that the highlighted $n_c^{\max}s$ consistent with Goto and van de Geijn [2008] is necessarily related to L2 and not L1 nor L3.

4.2.2 Case of the inner products

In this section, we will restrict ourselves to the case of the inner products of line 6 of Algo. 4. Therefore, we are interested in the tensor contraction of the form (4.1) where the third order tensor $\mathscr{C}$ the dimensions $j \times n \times s$ where n is the number of degrees of freedom per sample and j is the dimension of the current Krylov subspace as in the case of the update. Moreover the third order tensor $\mathscr{C}$ is now stored with the following layout:

$$c_{ik\ell} \hookleftarrow c\left[(i-1)\,n\,s + (k-1)\,s + \ell\right], \tag{4.5}$$

meaning that two ensembles $\boldsymbol{c}_{ik:}$ and $\boldsymbol{c}_{i(k+1):}$ are stored contiguously in the memory. The layout is different from the case of the update because we use the same data for both the inner products and the update and one of the operations requires a transpose operation. This tensor contraction is strongly related to the update of the previous subsection.

First, once again, we review the default implementation of the KokkosKernels package in the case of the inner products. This implementation is illustrated in Algo 7 where **parred** stands for a *parallel reduce*: a parallel loop which, here, computes the sum of the contribution of each thread. The algorithm consists in a parallel reduce over the columns, followed by a loop over the rows, and finally a loop over the samples. Once again, this default implementation has not an ideal memory access pattern for CPU especially for small ensemble sizes. The memory access pattern of this implementation is illustrated in Fig. 4.10a.

Algorithm 7: Inner products: default implementation of KokkosKernels

```
1 parfor $k = 1,\dots,j$ do
2     $w_{k\ell} = \alpha_\ell\, w_{k\ell}$   for all   $\ell = 1,\dots,s$
3 parred $i = 1$ to $n$ do
4     for $k = 1,\dots,j$ do
5         $w_{k\ell} = w_{k\ell} + \beta_\ell\, c_{ki\ell}\, z_{i\ell}$   for all   $\ell = 1,\dots,s$
```

In this work, we propose a tiled strategy which relies on the transpose of the one of the update with some modifications. Just applying the transpose of the strategy of the update directly leads to the Algo. 8 which does not perform correctly due to data race.

Although this algorithm streams data with the same unit stride as the update algorithm and reuses $\mathbf{Z}_{(t:t+n_c-1):}$ from L2 providing that n_c is sufficiently small, this algorithm is incorrect as it is not thread safe. Data races occur when the threads try to update $w_{k\ell}$.

In order to solve that issue, we use two strategies:

- Firstly, the threads are grouped in teams of W threads which are run on *nearby hardware threads* such as threads which share a common cache. This is controlled by the indexing of the threads and how they are bonded to hardware threads. On CPUs which support hyper-threading, we can create teams of 2 threads which are bonded to a same physical core. Those two threads will, therefore, share all their cache levels. Those teams are created using Kokkos.

Algorithm 8: First threaded tiled approach

```
1  parfor $k = 1, \dots, j$ do
2      $w_{k\ell} = \alpha_\ell\, w_{k\ell}$   for all   $\ell = 1, \dots, s$
3  parfor $t = 1$ to $n - n_c + 1$ by $n_c$ do
4      for $k = 1, \dots, j$ do
5          $\gamma_\ell = 0$   for all   $\ell = 1, \dots, s$
6          for $i = t, \dots, t + n_c - 1$ do
7              $\gamma_\ell$ += $c_{ik\ell}\, z_{i\ell}$   for all   $\ell = 1, \dots, s$
8          $w_{k\ell}$ += $\beta_\ell\, \gamma_\ell$   for all   $\ell = 1, \dots, s$
```

A parallel reduce is then used to compute the γ_ℓ of the full team which is then used to update the value of $w_{k\ell}$ by only one thread of the team. This reduces the number of threads which try to access and update $w_{k\ell}$ simultaneously in the main memory.

- Secondly, instead of looping on the rows starting at the first index and ending at the index j, the loop is split into two loops and the first index is evenly distributed among the j rows.

Those two strategies lead to the Algo. 9 which is a richer implementation than the default implementation in KokkosKernels.The memory access pattern of this implementation is illustrated in Fig. 4.10c.

Algorithm 9: Second threaded tiled approach

```
1  parfor $k = 1, \dots, j$ do
2      $w_{k\ell} = \alpha_\ell\, w_{k\ell}$   for all   $\ell = 1, \dots, s$
3  parfor $t = 1$ to $n - W\, n_c + 1$ by $W\, n_c$ do
4      Get team id $T$
5      $f = \mathrm{mod}(T, j)$
6      for $k = f, \dots, j$ do
7          $\gamma_\ell = 0$   for all   $\ell = 1, \dots, s$
8          parred $i = t, \dots, t + W\, n_c - 1$ do
9              $\gamma_\ell$ += $c_{ik\ell}\, z_{i\ell}$   for all   $\ell = 1, \dots, s$
10         $w_{k\ell}$ += $\beta_\ell\, \gamma_\ell$   for all   $\ell = 1, \dots, s$
11     for $k = 1, \dots, f - 1$ do
12         $\gamma_\ell = 0$   for all   $\ell = 1, \dots, s$
13         parred $i = t, \dots, t + W\, n_c - 1$ do
14             $\gamma_\ell$ += $c_{ik\ell}\, z_{i\ell}$   for all   $\ell = 1, \dots, s$
15         $w_{k\ell}$ += $\beta_\ell\, \gamma_\ell$   for all   $\ell = 1, \dots, s$
```

Finally, we show in Fig. 4.9 the throughput of the inner products of GMRES on a problem of size $n = 32000$ with the default GEMV implementation of the KokkosKernels to compute the inner product without reduction and with our proposed implementation with $W = 4$. The values of $n_c^{\max}$ of Table 4.1 are still valid as the L2 cache size is unchanged and as the number of reusable entries $n_c s$ is the same for both algorithms.

Once again, we observe the impact of the memory access pattern of the default implementation on the throughput especially for small ensemble sizes. As opposed to the case of the update, we observe that the throughput of the inner products for larger ensemble sizes reaches 80% of the theoretical limit. This is due to the overhead cost of the parallel reduce. The performance of the proposed implementation is better for all ensemble sizes due to both the memory access pattern and the strategy used to reduce the synchronization cost of the shared memory.

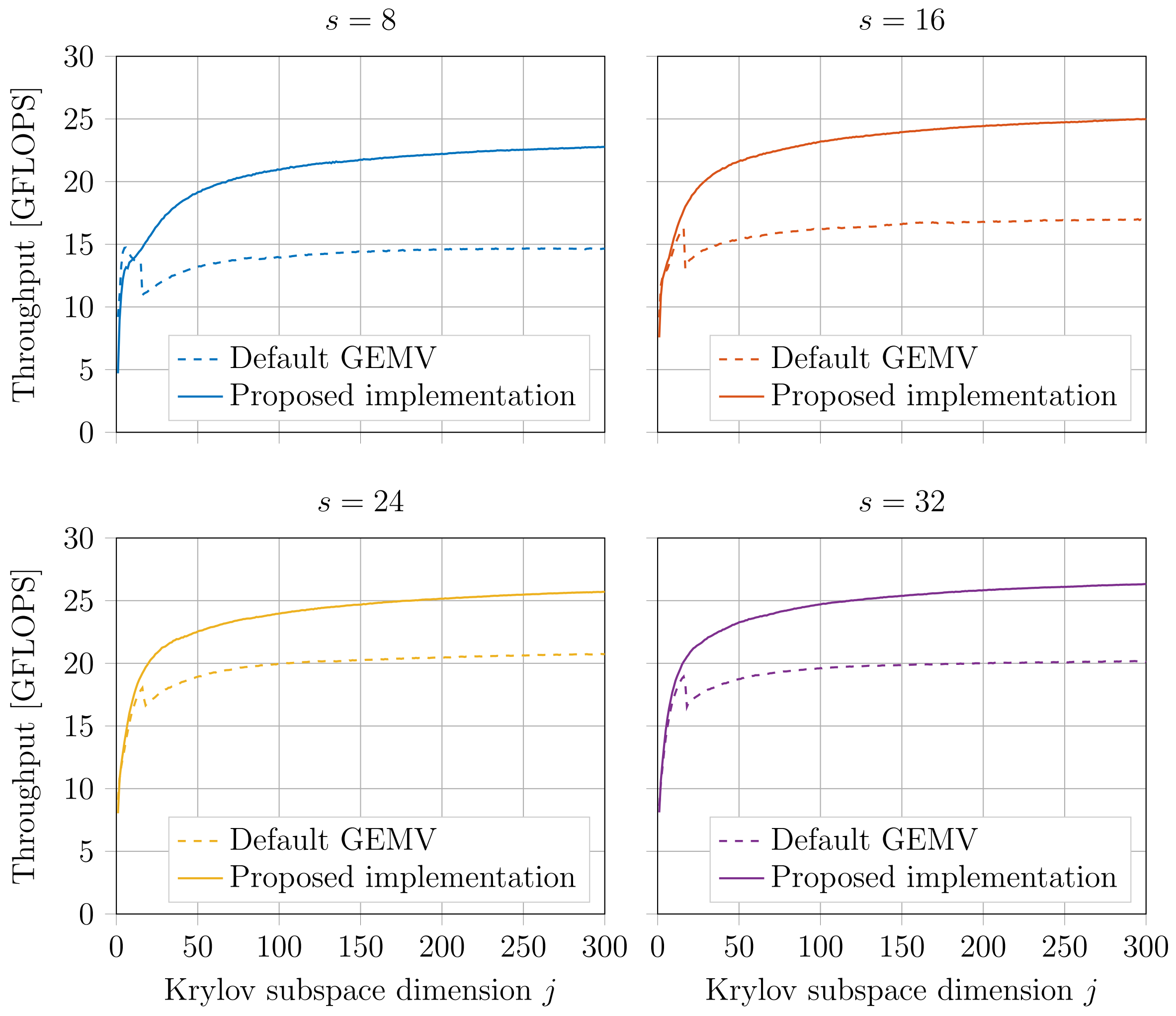


Figure 4.9: Comparison of throughput of the inner products in GMRES when using the default Trilinos implementation and our tiled approach.

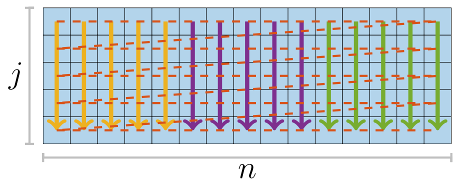
In parallel to this work, KokkosKernels has recently[1] included an implementation of the GEMV for the inner products based on two nested parallel loops using a team-based approach. This approach relies on a parallel for which loops over the rows which associates a full row to a team. Each team performs a parallel reduce over the entries of their associated row. The memory access pattern of this approach is illustrated in Fig. 4.10b. This approach has a better memory access pattern than the default GEMV tested in this thesis but has two drawbacks:

- the number of threads that can be used depends on the number of threads per team

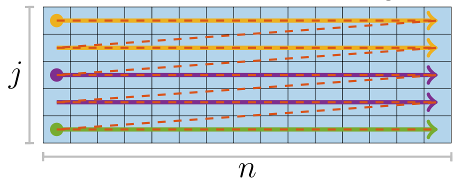
[1]`https://github.com/trilinos/Trilinos/commit/b4316dfd25c8ba29798d406ccf86657c196f2ff2`

and the number of rows j. If j is too small, the number of threads used will be too small resulting in a reduced memory bandwidth.

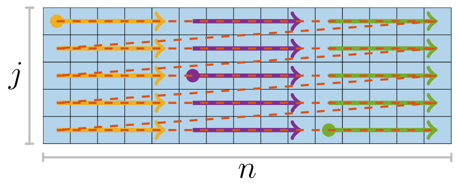
- the probability of reusing entries of $\mathbf{Z}$ from the L2 cache depends on the size of n. If n is small, $\mathbf{Z}$ will be kept in cache, otherwise, if n is large and if j is sufficiently large such that a team of threads is required to compute at least a second row, entries of $\mathbf{Z}$ will have to be streamed from the main memory again.



(a) Default GEMV tested for the inner products: `SingleLevelTransposeGEMV`.



(b) New KokkosKernels GEMV for the inner products: `TwoLevelTransposeGEMV` .



(c) Tiled approach.

Figure 4.10: Illustration of the memory access patterns to read $\mathbf{C}_{::\ell}$ from the main memory with the default GEMV and the tiled approach. The arrows represent the memory access pattern of a thread (one color per thread) in the case of the tested default GEMV, `SingleLevelTransposeGEMV`, and of a team of thread in the case of the new KokkosKernels GEMV, `TwoLevelTransposeGEMV`, and the tiled approach. The dashed red line represents the left layout. The colored dots represent the starting point of the memory access pattern of a team. The first GEMV implementation suffers from jumps in the memory address moving from one row to the next one. The two others follow the layout. However, `TwoLevelTransposeGEMV` has less opportunity to use a larger number of threads for small j and has a lower probability to reuse entries of $\mathbf{Z}$ from the L2 cache for larger n.

4.2.3 Influence of hyper-threading

For the results discussed in this chapter, the hyper-threading of Intel CPUs has been used, i.e. each physical core has been used with 2 threads. Although the performance of our ensemble GEMV reaches full memory bandwidth, performances of other parts of the full-simulation code can be reduced due to the use of hyper-threading. Therefore, in this subsection, we revisit the results discussed previously without hyper-threading to highlight its effect on our ensemble GEMV.

Before discussing the throughput of ensemble GEMV, we first show the measured memory bandwidth without hyper-threading on one NUMA region in Fig. 4.11 with a best memory bandwidth of 102.8772 GB/s. We observe the same memory bandwidth as previously with hyper-threading (see Fig. 4.1).

With this measurement in mind, we can now measure the throughput and cache misses as illustrated in Fig. 4.8 but without hyper-threading; those are illustrated in Fig. 4.12. We see from those graphs that, as there is now only one thread per CPU core, each thread can use scaled $n_c s$ two times larger before starting to face reduced performance.

Another comment is that the maximal scaled $n_c^{\max} s$ deduced with hyper-threading continues to be efficient without hyper-threading whereas the opposite is not true and may lead to a large drop in the performance. This is due to the correction (4.4) as this correction results in the following cases:

- if $n \geq N\, n_c^{\max}$, then, due to (4.4), we have that $\frac{n_c^{\max}}{2} \leq n_c^\star \leq n_c^{\max}$ which is a region of good performance as illustrated by the blue area in Fig. 4.12,
- if $n < N\, n_c^{\max}$, then, increasing $n_c^{\max}$ does not influence $n_c^\star$.

In other words, if we use the same $n_c^{\max}$ as in the hyper-threading case or if we double it $2\, n_c^{\max}$, for small number of degrees of freedom ($n < N\, n_c^{\max}$) it does not influence $n_c^\star$ and for larger number of degrees of freedom, the performance is good with $\frac{n_c^{\max}}{2} \leq n_c^\star \leq n_c^{\max}$.

To conclude these remarks; the implemented ensemble GEMV has good performance regardless of the use of hyper-threading with the maximal scaled tile size deduced with two threads per core. While not using the hyper-threading, this maximal scaled tile size can be at most double.

As discussed in this section, the performance of ensemble GEMV is not influenced by the use of the hyper-threading. However, we observed that another part of the code, the sparse matrix-vector product with `double`, can have deteriorated performances using the hyper-threading. To measure meaningful speed-ups, we decided to disable the hyper-threading in Chapters 7 and 8.

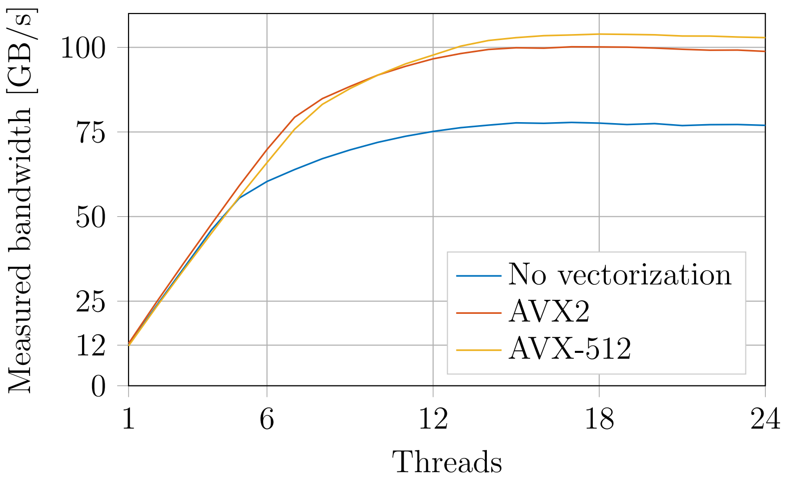


Figure 4.11: Benchmarked STREAM Triad Memory Bandwidth on a Skylake node with 1 Intel(R) Xeon(R) Platinum 8160 CPU with a problem size of 2.4 GB and no hyper-threading.

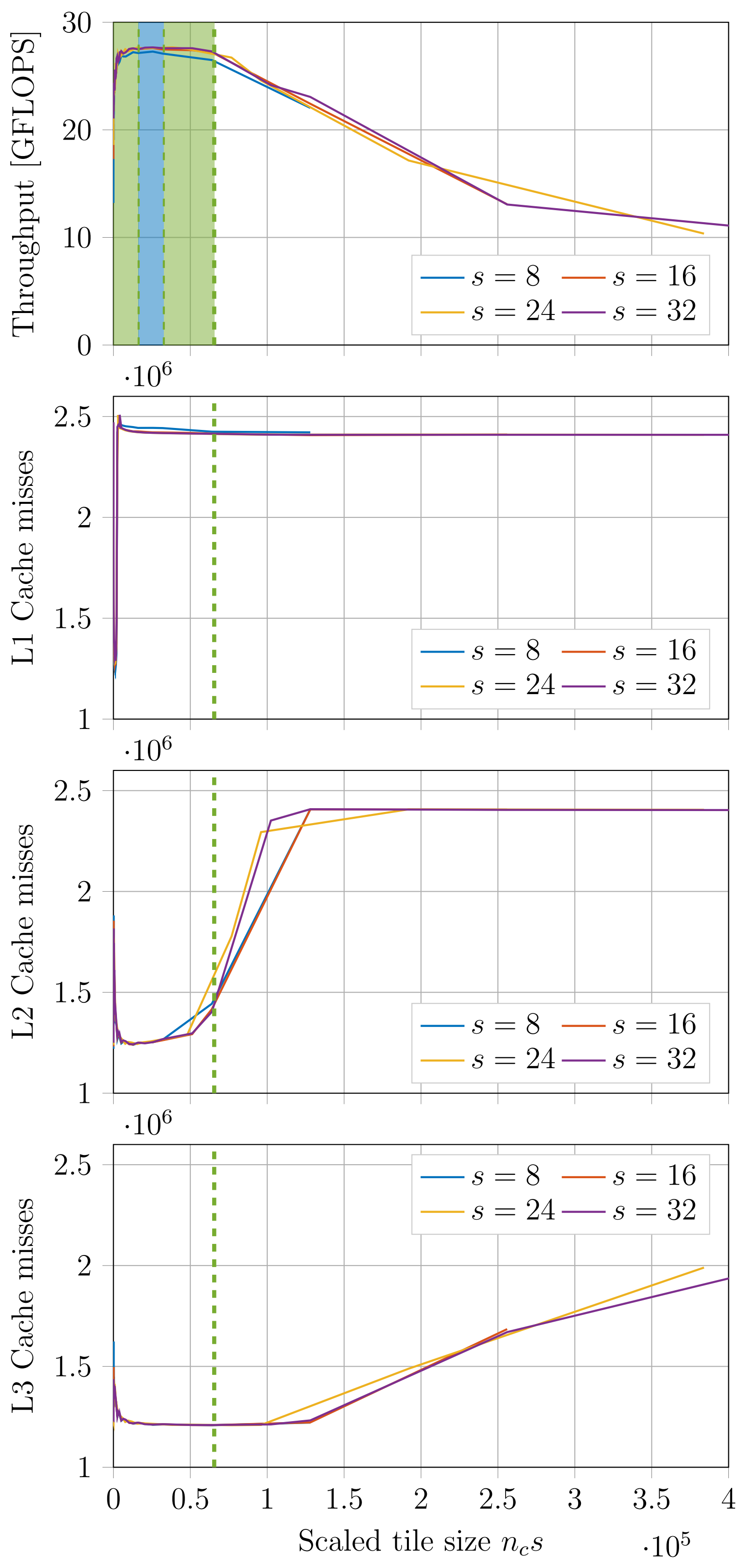


Figure 4.12: Influence of the scaled $n_c s$ on the L1, L2, and L3 cache misses per sample without hyper-threading. The blue region corresponds to the scaled $n_c s$ such that $\frac{n_c^{\max}}{2} \leq n_c^\star \leq n_c^{\max}$ using $n_c^{\max}$ computed with hyper-threading. For large values of the scaled $n_c s$, the cache misses of all level tend to the same value because the entries of $\mathbf{W}_{(t:t+n_c-1):}$ are not reused from cache. For smaller values, we see a correlation between L2 cache misses and the throughput of the tiled algorithm. We observe that the blue region has good performance and, therefore, the values of $n_c^{\max}$ can be chosen independently of the use of the hyper-threading.

4.2.4 Comparison with reduction

In this section, we compare the performance of GEMV without reduction with the performance of GEMV with reduction to illustrate that their throughput and CPU cost are similar. The goal of this comparison is to deduce that the difference in the performance of ensemble GMRES with and without reduction is only influenced by the number of iterations to converge. This is predictable as both GEMV with and without reduction have the same time complexity, i.e. the number of elementary operations performed by the algorithm, $O(njs)$, i.e. the same number of floating-point operations, and the same arithmetic intensity 2 flops per loaded $c_{ik\ell}$.

First, we show in Fig. 4.13 and Fig. 4.14 the wall-clock time and throughput respectively of the update of GMRES with and without reduction on a problem of size $n = 32000$ with $n_c^{\max}$ chosen as in Table 4.1. The update with reduction has been computed using the MKL GEMV. We see that both approaches have similar throughput and wall-clock time (in this particular example, our implementation is even slightly faster than MKL). This is consistent with the fact that they both have the same time complexity and the same arithmetic intensity. As a consequence the wall-clock time of one iteration of GMRES is similar with and without reduction.

Finally, we show in Fig. 4.15 and Fig. 4.16 the wall-clock time and throughput respectively of the inner products of GMRES with and without reduction on a problem of size $n = 32000$ with $n_c^{\max}$ chosen as in Table 4.1. The inner products with reduction have been computed using the Intel MKL GEMV. As observed for the update case, we see that both approaches have similar throughput and wall-clock time. In Fig. 4.16, we observe two different behaviors for $j \leq 100$ and $j > 100$, we think that this two behaviors are potentially explained by the fact that the MKL GEMV has at least two implementations one for smaller matrix $j \leq 100$ and one for larger matrix $j > 100$. In the later, we observe a periodicity of 48 in the throughput, we think that this periodicity is explained by an uneven distribution of the work among the threads in the second implementation; We think that the rows are distributed among the threads and that if there is a number of rows which can be divided by 48 each thread has the same amount of work to compute. However, if there is one extra row, all the threads except one will have to wait for the completion of the work of the last thread. This belief is consistent with the plateau of Fig. 4.15. Those beliefs cannot be easily verified as we do not have access to the source code of the MKL GEMV.

The fact that both inner products and update in GMRES have CPU costs which are independent of the use of reduction leads to the same wall-clock time of one iteration of GMRES. As a consequence, we know that the wall-clock time of GMRES without reduction will be smaller than the wall-clock time of GMRES with reduction due to faster convergence of GMRES without reduction compared to GMRES with reduction as discussed in Chapter 3.

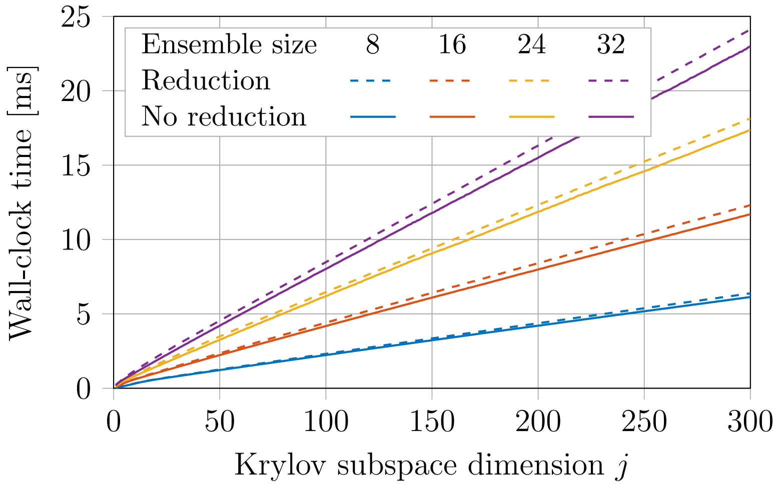


Figure 4.13: Wall-clock time of the update in GMRES with and without reduction.

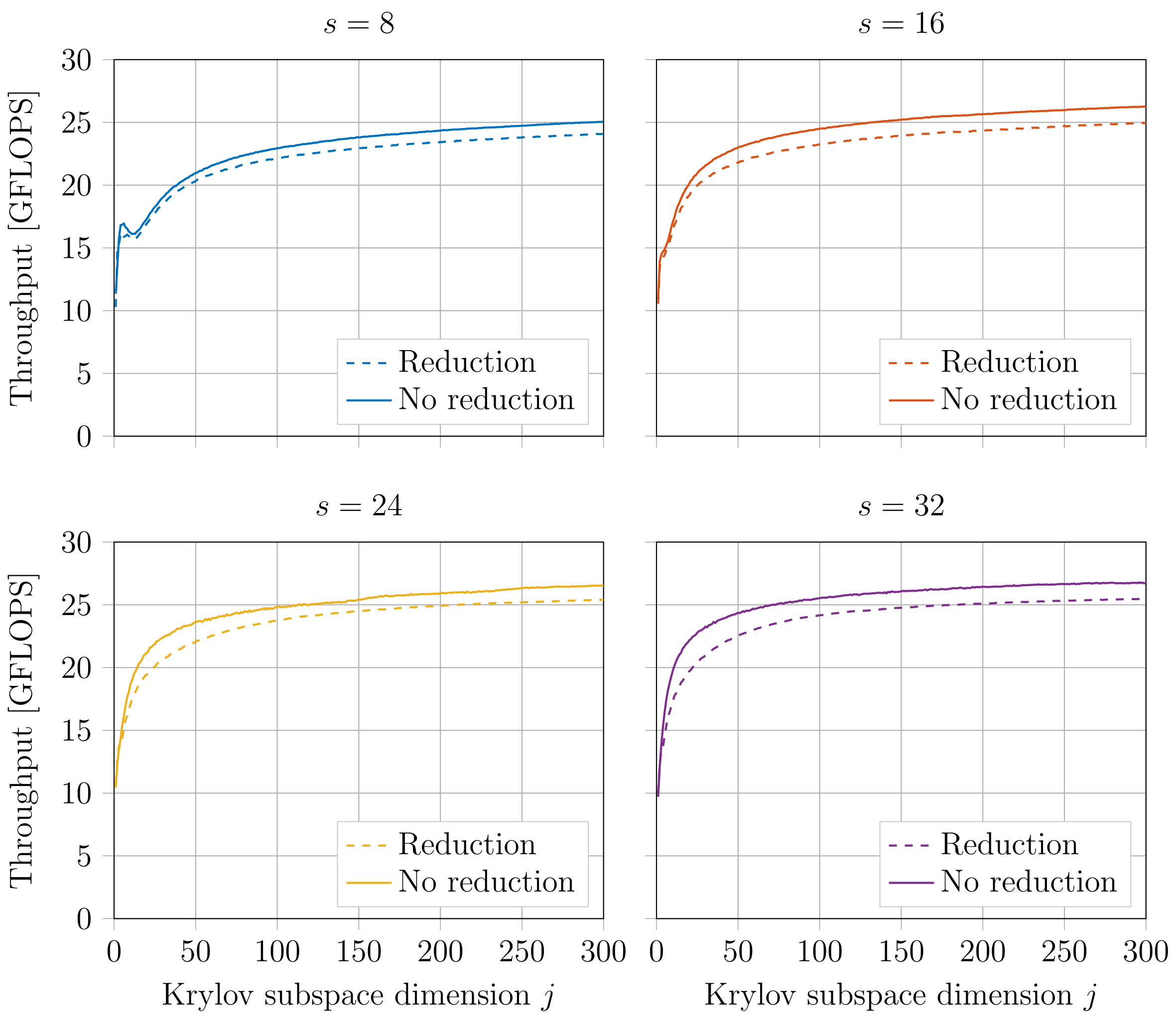


Figure 4.14: Throughput of the update in GMRES with and without reduction. We observe that both approaches have the similar throughput.

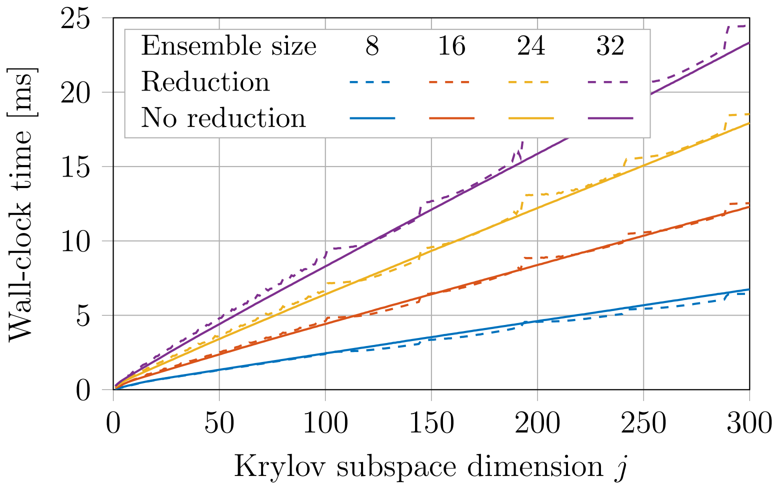


Figure 4.15: Wall-clock time of the inner products in GMRES with and without reduction.

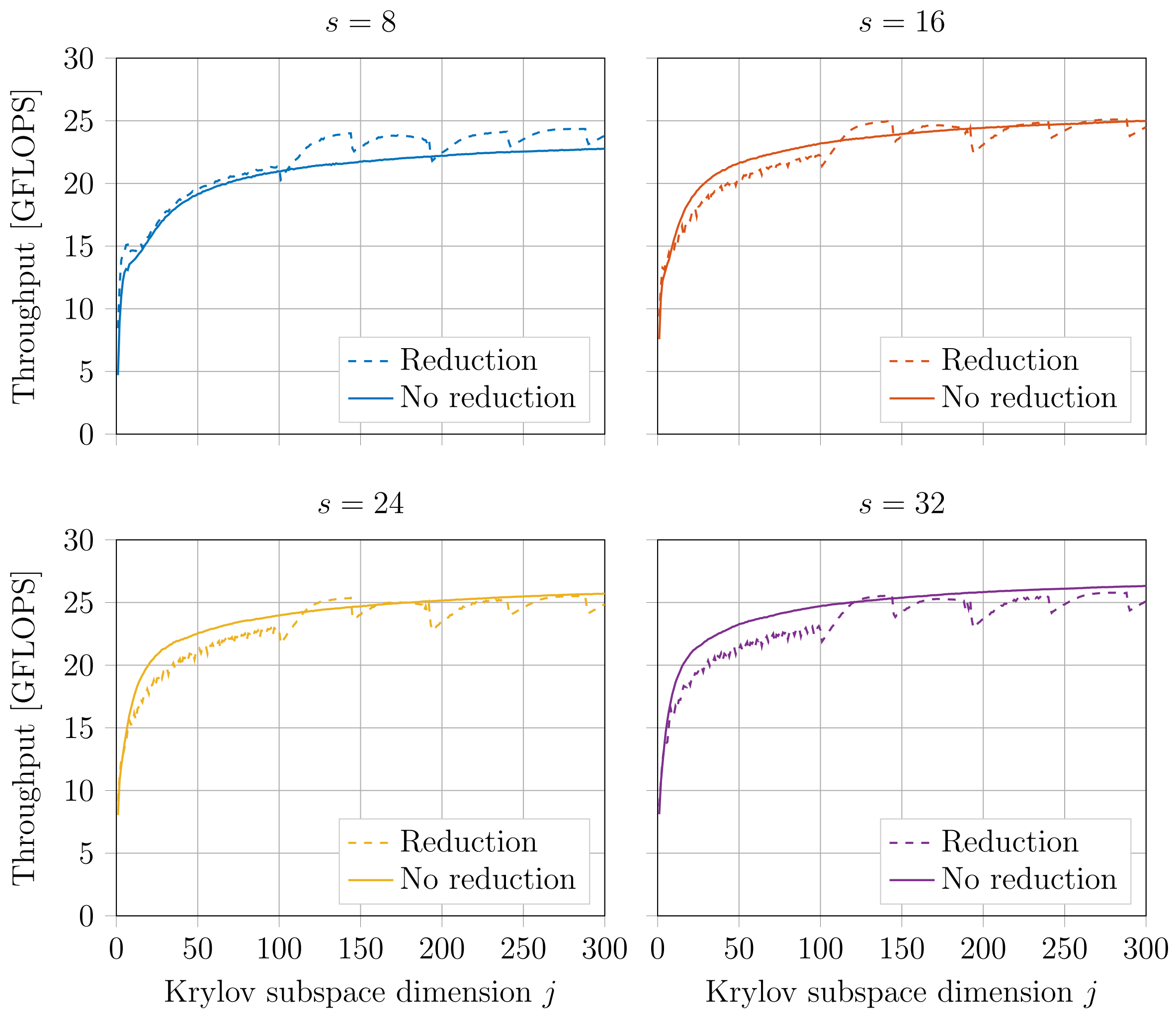


Figure 4.16: Throughput of the inner products in GMRES with and without reduction. We observe that both approaches have roughly the same throughput.

4.3 Controlling control-flow divergence with masks

As introduced in section 1.1.3 and section 2.1, control-flow divergence [Coutinho et al., 2011] occurs when the samples of an ensemble want to follow different code branches. Such cases occur when the result of a comparison operation does not return the same value for each sample.

In this section we discuss the use of masks as an alternative to the `EnsembleTrait`, a strategy introduced by Phipps et al. [2017] to manage occurrences of control-flow divergence. In section 4.3.1, we review the `EnsembleTrait`. In section 4.3.2, we propose a C++ interface for the mask class, in other words, the features of the mask and how they can be used. Section 4.3.3 is more technical and is dedicated to a proposed implementation of the class and its member functions. Finally, in section 4.3.4, we use the mask class to manage occurrences of control-flow divergence in GMRES without reduction.

4.3.1 `EnsembleTrait` strategy

Before introducing the masks and how to use them, we first recall how the control-flow divergence was solved in [Phipps et al., 2017] with a so-called *type trait*, one of the C++ utilities to support template metaprogramming as discussed in [Stroustrup, 2000]. The type trait is a generic programming strategy useful when the data types used as template parameters are too different from each other. In the considered case of embedded ensemble propagation, we use ensemble data types which have s entries instead of standard data types such as `double`. The ensemble types come with member functions to access the entry of a given sample or to get the ensemble size s. Those member functions are not defined for the standard types inside the standard library, therefore, it is not generic to call those member functions explicitly in the code as it would prevent the compilation of the code with standard data type. The workaround used in [Phipps et al., 2017] is to define a type trait called `EnsembleTrait` which encapsulates those member functions for ensemble types and which provides default behavior for standard data type; for example, the ensemble size returned by the `EnsembleTrait` for `double` is $s = 1$ and accessing the first sample of the `double` returns its value. This `EnsembleTrait` allows one to manage the control-flow divergence by looping explicitly over the samples in a generic way which remains valid for standard data type. The `EnsembleTrait` is illustrated in Listing 4.1.

```
// Base template definition of EnsembleTrait (used for "double")
template <typename T>
struct EnsembleTrait
{
    typedef T value_type;
    static const int ensemble_size = 1;
    static const T &coeff(const T &x, const int i) { return x; }
    static T &coeff(T &x, const int i) { return x; }
};
// Specialization of EnsembleTrait for Ensemble
template <typename T, int s>
struct EnsembleTrait<Ensemble<T, s>>
{
    typedef T value_type;
    static const int ensemble_size = s;
    static const T &coeff(const Ensemble<T, s> &x, const int i) { return x[i]; }
    static T &coeff(Ensemble<T, s> &x, const int i) { return x[i]; }
};
```

Listing 4.1: `EnsembleTrait as defined in [Phipps et al., 2017].`

As an example, we recall the example used in [Phipps et al., 2017] to highlight ensemble divergence. Assuming that we want to compute $\mathbf{y}$ such that:

$$y_\ell = \begin{cases} x_\ell + x_\ell^2, & \text{if } x_\ell > 0 \\ x_\ell, & \text{if } x_\ell \leq 0 \end{cases}, \quad \text{for all} \quad \ell = 1, \dots, s, \tag{4.6}$$

the implementation using `EnsembleTrait` proposed in [Phipps et al., 2017] is the one shown in Listing 4.2 where the implementation explicitly loops over the samples, does the sample-wise comparison $x_\ell > 0$, and performs the correct operation depending on the result of the comparison. In Listing 4.2 and in the remainder of this section, `T` is the stored data type that can be `double` or `Ensemble`.

```
typedef EnsembleTrait<T> ET;
typedef typename ET::value_type ScalarValue;
const int s = ET::ensemble_size;
T x = ...
T y;
for (int l = 0; l < s; ++l)
{
    const ScalarValue &xl = ET::coeff(x, l);
    ScalarValue &yl = ET::coeff(y, l);
    if (xl > 0)
        yl = xl + xl * xl;
    else
        yl = xl;
}
```

Listing 4.2: Ensemble divergence with `EnsembleTrait.`

We have noticed that implementations based on the `EnsembleTrait` where we loop over the samples explicitly are not always autovectorized correctly by the compiler. In order for the loop over the samples to be autovectorized, “The loop must contain straight-line code (a single basic block). There should be no jumps or branches, but masked assignments are allowed.” [Jeffers et al., 2016]. While using the `EnsembleTrait` it is possible to implement loops for which the body includes jumps or branches, the compiler can vectorize such loops if it can automatically generate the masked assignment which is not always the case. For instance, on an Intel architecture supporting AVX-512 (Intel Xeon Phi KNL) with the Intel compiler, we have observed that one assembly code generated by the autovectorization of a loop including branches in its body implemented with `EnsembleTrait` was using AVX2 instructions instead of AVX-512 instructions. This results in performance issues as AVX2 instructions have smaller throughput.

A second strategy introduced in [Phipps et al., 2017] to deal with the control-flow divergence was the definition of the ensemble comparisons. Those ensemble comparisons return a boolean value which corresponds to the result of the comparison evaluated for the first sample of the ensemble only. If the results of the comparison of the other samples are different, this leads to a potentially incorrect code path.

4.3.2 Mask interface

In this section, we will show how masking can be used in the context of ensemble propagation as an alternative to the `EnsembleTrait` in order to improve performance and to control the followed code path. Masking removes the requirement of looping explicitly over the samples in the code with `EnsembleTrait` by relying on masked assignments. The use of masking results in more concise implementations that can be vectorized more easily by the compiler as the jumps and the branches discussed in the previous section are removed using the masks.

Masking, in the vectorization context, is a technique which enables the execution of conditional branching instructions inside a vectorized loop [Lorie and Strong Jr, 1984]. The CPU, depending on boolean values loaded in a mask register, evaluates only the correct branch for each entry of the vector type using only one machine instruction. The compiler can generate masking instructions while autovectorizing the code as long as the computational work inside the branches is sufficiently simple as discussed in [Jeffers et al., 2016]. On Intel architectures, the masking instructions can be called explicitly from C++ codes using Intel Intrinsics [Lomont, 2011], a set of C++ functions that the compiler replaces with the proper assembly vector instructions. The main drawbacks of using Intel Intrinsics explicitly everywhere in the code are the portability, the amount of work that has to be done to update the code for larger vector sizes in the future as larger vector sizes come with new Intel intrinsics requiring to replace all the Intel intrinsics calls, and the fact that the code cannot be templated anymore. To overcome these issues, several implementations of intrinsics wrappers have been introduced such as, among others, Vc [Kretz and Lindenstruth, 2012], Boost.SIMD [Estérie et al., 2014], Cyme [Ewart et al., 2014], UME::SIMD [Karpiński and McDonald, 2017], and MIPP [Cassagne et al., 2015]. In particular, Vc has led to the submission of a proposal to the C++ standards committee to include SIMD data types.

As an alternative, with some inspiration from Kretz and Lindenstruth [2012], we have overloaded the comparison operators of the ensemble types to return a `Mask` object. This new `Mask` object contains, as member data, the results of the sample-wise comparison.

This new object can be used to do masked assignments and logical reductions; the masked assignment is a conditional assignment which assigns different values depending on the value of a conditional and the logical reduction is a logical operation applied on a set of booleans.

Masked assignment

For example, the masked assignment can be used to rewrite the code of Listing 4.2 in Listing 4.3 where the comparison `x>0` creates a `Mask` object with the result of the sample-wise comparison. This `Mask` is then used to assign values to `y`; if, for a given sample, the corresponding value of `Mask` is `true`, the corresponding value of `y` is equal to the corresponding value of `x+pow(x,2)`, otherwise, it is equal to the corresponding value of `x` by default. This implementation is more concise, can improve the readability, and eases the vectorization by the compilers.

```
T x = ...
T y;
MaskAssign(x > 0, y) = {x + pow(x, 2), x};
```

Listing 4.3: Ensemble divergence with masked assignment.

The implemented masked assignments have been motivated by some of the available Intel intrinsic instructions. For example, we have implemented the masked assignment `MaskAssign(m, a)/= {b, c, d}` which computes:

$$a_\ell = \begin{cases} \frac{b_\ell}{c_\ell}, & \text{if } m_\ell \\ d_\ell, & \text{otherwise} \end{cases}, \quad \text{for all} \quad \ell = 1, \dots, s, \tag{4.7}$$

based on the Intel intrinsic instruction `_mm512_mask_div_pd`.

We think that such interfaces are good candidates as they provide the possibility to specify all the arguments in one line of code; for example, b_ℓ, c_ℓ, and d_ℓ of (4.7) can be specified in one line.

Logical reduction

The logical reduction allows more complex ensemble comparisons than the one introduced in section 4.3.1. For instance, it is possible to write with logical reduction the examples of Listing 4.4 where `all_positive` is `true` if $x_\ell > 0$ for all $\ell = 1, \dots, s$ and `false` otherwise and `all_negative` is `true` if $x_\ell <= 0$ for all $\ell = 1, \dots, s$ and `false` otherwise. It is important to note that if `all_positive` is `false`, the value of `all_negative` is not necessarily `true`.

```
T x = ...
bool all_positive = AND(x > 0);
bool all_negative = AND(x <= 0);
```

Listing 4.4: Examples of logical reduction.

4.3.3 Implementation of masks

The new `Mask` object is implemented in Listing 4.5. Its member data is an array of `unsigned char` which is equivalent to an array of `__mask8`, the mask of size 8 defined in AVX-512. This way, if we build with AVX-512 support, the masked Intel Intrinsics instruction can be directly called as long as we use ensemble types with `double` as the value type.

We have used an approach with some inspiration from [Kretz and Lindenstruth, 2012] for AVX2 to implement the mask and the masked assignment. Kretz and Lindenstruth [2012] proposed to define a mask type which stores the results of a comparison of SIMD data types element-wise and, then, use this mask type inside a `Vc::Common::WriteMaskedVector` class to do the masked assignment depending on the booleans stored in the mask. In our work we define two classes: one for the mask in Listing 4.5 and one for the masked assignment in Listing 4.6. The work of Kretz and Lindenstruth [2012] covers broader discussion on SIMD data types and is not limited to the mask and the masked assignment. Some of the differences of the presented work with their mask and their masked assignment are that the interface and the implementation of the classes are different and, here, we use ensemble sizes that can be larger than the width of the vector instruction[2], we use AVX-512 instruction sets, and, as AVX-512 defines a mask type,

[2]Kretz [2018] has discussed that aspect. They have defined two types: `fundamental`, a SIMD data type which has a length equal to the width of the vector instruction, and `arbitrary`, a SIMD data type which has an arbitrary length chosen by the user.

we want our mask type to be as close as possible to the AVX-512 mask type but keep the possibility to use larger ensemble sizes such as 32 and not restrict ourselves to the ensemble size equal to the width of the vector instruction.

There are two implementations of the mask and masked assignment: one which relies on Intel Intrinsics AVX-512 instructions, and a default one which accesses the data sample-wise. If the AVX-512 instructions are used, two utility functions are used `data= M512D_EP_LOAD(a,i)` and `M512D_EP_STORE(a,i,data)`. The first one loads a vector of type `__mm512d` from the entries $a_{8\,i}$, ..., $a_{8\,i+7}$ of the ensemble **a** and the second one stores a vector of type `__mm512d` in the entries $a_{8\,i}$, ..., $a_{8\,i+7}$ of the ensemble **a**.

In the presented listing `KOKKOS_INLINE_FUNCTION` is a Kokkos macro to define inline functions that can be compiled both on the CPU and the GPU. If built on the CPU only, `KOKKOS_INLINE_FUNCTION` is equivalent to `inline` whereas `KOKKOS_INLINE_FUNCTION` is equivalent to `inline __device__ __host__` if built with CUDA support. The macro `STOKHOS_HAVE_AVX_512` is defined if the current build supports AVX-512 and is, therefore, not defined if built for GPU.

The macros `STOKHOS_HAVE_PRAGMA_IVDEP`, `STOKHOS_HAVE_PRAGMA_VECTOR_ALIGNED`, and `STOKHOS_HAVE_PRAGMA_UNROLL` are defined if Stokhos is built using the pragmas `ivdep` , `vector aligned`, and `unroll` respectively. The pragmas `ivdep`, `vector aligned`, and `unroll` instruct the compiler to ignore assumed vector dependencies, to use aligned data movement instructions for all array references when vectorizing, and to unroll a counted loop [Intel, 2019].

```
// Mask type
template <typename T>
class Mask
{
    static const int s = EnsembleTrait<T>::size;

public:
    static const int size_uc = s / sizeof(unsigned char);
    unsigned char data[size_uc];

    KOKKOS_INLINE_FUNCTION
    bool get(int i)
    {
        int j1 = i / sizeof(unsigned char);
        int j2 = i % sizeof(unsigned char);
        // Return the j2th bit of the j1th unsigned char:
        return (this->data[j1] >> j2) & 1;
    }

    KOKKOS_INLINE_FUNCTION
    void set(int i, bool b)
    {
        int j1 = i / sizeof(unsigned char);
        int j2 = i % sizeof(unsigned char);
        // Change the j2th bit of the j1th unsigned char:
        if (b)
            this->data[j1] |= 1 << j2;
        else
            this->data[j1] &= ~(1 << j2);
    }

    Mask(bool a)
    {
        for (int i = 0; i < s; ++i)
            this->set(i, a);
    }
};

//Overload of the ensemble comparisons
template <typename T, int s>
KOKKOS_INLINE_FUNCTION
    Mask<Ensemble<T, s>>
    operator<=(const Ensemble<T, s> &a1, const Ensemble<T, s> &a2)
{
    Mask<Ensemble<T, s>> mask;
#ifdef STOKHOS_HAVE_AVX_512
#ifdef STOKHOS_HAVE_PRAGMA_UNROLL
#pragma unroll
#endif
    for (int i = 0; i < mask.size_uc; ++i)
        mask.data[i] =
            _mm512_cmp_pd_mask(M512D_EP_LOAD(a1, i), M512D_EP_LOAD(a2, i), 2);
#else
#ifdef STOKHOS_HAVE_PRAGMA_IVDEP
#pragma ivdep
#endif
#ifdef STOKHOS_HAVE_PRAGMA_VECTOR_ALIGNED
#pragma vector aligned
#endif
#ifdef STOKHOS_HAVE_PRAGMA_UNROLL
#pragma unroll
#endif
    for (int i = 0; i < s; ++i)
        mask.set(i, a1.dat[i] <= a2.dat[i]);
#endif
    return mask;
}
// ...
```

Listing 4.5: Mask type and overloads of the ensemble comparisons.

As said in section 4.3.2, the implemented masked assignments have been motivated by some of the available Intel intrinsic instructions. For example, the masked assignment inspired by `_mm512_mask_div_pd` is implemented in Listing 4.6. In Listing 4.6, if the AVX-512 instructions are not used, the utility function `get(i)` of the `Mask` object is used to access the value of the comparison for the sample `i` as implemented in Listing 4.5.

```
template <typename T>
class MaskAssign
{
private:
    T &data;
    Mask<T> m;

public:
    MaskAssign(Mask<T> m_, T &data_) : m(m_), data(data_){};

    KOKKOS_INLINE_FUNCTION
    MaskAssign<T> &operator/=(const std::initializer_list<T> &st)
    {
        auto st_array = st.begin();
#ifdef STOKHOS_HAVE_AVX_512
#ifdef STOKHOS_HAVE_PRAGMA_UNROLL
#pragma unroll
#endif
        for (int i = 0; i < m.size_uc; ++i)
            M512D_EP_STORE(data, i,
                           _mm512_mask_div_pd(
                               M512D_EP_LOAD(st_array[2], i),
                               m.data[i],
                               M512D_EP_LOAD(st_array[0], i),
                               M512D_EP_LOAD(st_array[1], i)));
#else
        typedef EnsembleTrait<T> ET;
        const int size = ET::ensemble_size;
#ifdef STOKHOS_HAVE_PRAGMA_IVDEP
#pragma ivdep
#endif
#ifdef STOKHOS_HAVE_PRAGMA_VECTOR_ALIGNED
#pragma vector aligned
#endif
#ifdef STOKHOS_HAVE_PRAGMA_UNROLL
#pragma unroll
#endif
        for (int i = 0; i < size; ++i)
            if (m.get(i))
                ET::coeff(data, i) =
                    ET::coeff(st_array[0], i) / ET::coeff(st_array[1], i);
            else
                ET::coeff(data, i) = ET::coeff(st_array[2], i);
#endif
        return *this;
    }
    //...
};
```

Listing 4.6: Mask assignments.

An example of logical reduction is the `AND` function which takes a mask as input and returns a boolean which is `true` only if the mask stores `true` for every sample. This function is implemented in Listing 4.7. The logical reductions are used to control the code path followed by the ensemble and not to decide the followed path based on the first sample as discussed in [Phipps et al., 2017].

```
template <typename T>
KOKKOS_INLINE_FUNCTION bool AND(Mask<T> m)
{
    const unsigned char all_true = 255;
    for (int i = 0; i < m.size_uc; ++i)
        if (m.data[i] != all_true)
            return false;
    return true;
}
//...
```

Listing 4.7: Logical reduction.

4.3.4 Masks in ensemble GMRES

As discussed in the previous chapter, the control-flow divergences occur in GMRES during the lucky breakdown, the normalization process, and testing the stopping criterion. There are other occurrences of control-flow divergences occuring in some functions called by GMRES such as the ROTG and the TRSM functions which compute the Givens rotations and the solution of a lower triangular system respectively as those functions have if-then-else statements. In this section we do not discuss all the possible occurrences of ensemble divergence, we restrict ourselves to illustrating the use of the masks in the normalization process lines 8–11 of Algo. 4 and in the stopping criterion lines 19–21 of Algo. 4. Other occurrences have been treated with similar strategies.

Normalization

For the normalization of GMRES, we will first illustrate the use of `EnsembleTrait` and then the masked assignment. In both cases, we use a temporary variable `norm_inv` which is the inverse of the norm of the vector if the norm is not 0 for a given sample and 0 otherwise.

We illustrate in Listing 4.8 how `EnsembleTrait` can be used to normalize the vector if its norm is not 0. In this case, each sample can follow a different code path.

In the second approach, we use the masked assignment `MaskAssign(m,a)/={b,c,d}` as discussed in section 4.3.2 and 4.3.3 and implemented in Listing 4.6 to compute the temporary variable `norm_inv` as follows: `MaskAssign(norm>0,norm_inv)/={1.,norm,0.}` . The code is shown in Listing 4.9 for the normalization process.

```cpp
typedef EnsembleTrait<T> ET;
const int s = ET::ensemble_size;

T norm_inv;
for (int l = 0; l < s; ++l)
    if (ET::coeff(norm, l) > 0)
        ET::coeff(norm_inv, l) = 1. / ET::coeff(norm, l);
    else
        ET::coeff(norm_inv, l) = 0.;

for (int i = 0; i < n; ++i)
    v[i] *= norm_inv;
```

Listing 4.8: Normalisation with EnsembleTrait.

```cpp
T norm_inv;
MaskAssign(norm > 0, norm_inv) /= {1., norm, 0.};

for (int i = 0; i < n; ++i)
    v[i] *= norm_inv;
```

Listing 4.9: Normalisation with masked assignment.

Both of those implementations have exactly the same output, however, the second implementation is more concise and relies directly on Intel Intrinsics.

Stopping criterion

For the stopping criterion, we do the same too; we illustrate the use of `EnsembleTrait` and mask to test if all samples have converged or not. The first approach consists in looping over all the samples explicitly and check their norm as illustrated in Listing 4.10. The second approach consists in using a mask and a logical reduction `AND` as shown in Listing 4.11.

```
typedef EnsembleTrait<T> ET;
const int s = ET::ensemble_size;
// GMRES iterations
for (int j = 0; j < j_max; ++j)
{
    // ...
    bool has_converged = true;
    for (int l = 0; l < s; ++l)
        // Test if at least one sample has a residual norm
        // strictly larger than the tolerance
        if (ET::coeff(norm, l) > tol)
        {
            has_converged = false;
            break;
        }
    if (has_converged)
        break;
}
```

Listing 4.10: Stopping criterion with EnsembleTrait.

```
// GMRES iterations
for (int j = 0; j < j_max; ++j)
    // ...
    // Test if all the samples have a residual norm
    // smaller than the tolerance
    if (AND(norm <= tol))
        break;
```

Listing 4.11: Stopping criterion with mask and logical reduction.

Once again, both implementations have exactly the same output, however, the second implementation is more concise and potentially more readable.

4.4 Conclusions

In section 4.2, we have proposed a new implementation for the ensemble GEMV which takes place in GMRES without reduction and have discussed its implementation using a standard tiling strategy. This work leads to richer implementations than those that were available as default implementations in KokkosKernels. We have shown that this tiling strategy is able to reach very good performance similar to the Intel MKL implementation and have shown how the ensemble size impacts the choice of the tile size and the performance of the algorithm. In particular, we have discussed the fact that the expected speed-up of the orthogonalization process due to ensemble propagation is 1 for large problems and that it should be independent of the use of reduction. This implies that the wall-clock time of GMRES without reduction will be smaller than the wall-clock time of GMRES with reduction due to faster convergence of GMRES without reduction compared to GMRES with reduction as discussed in Chapter 3.

After that, in section 4.3, we have proposed to use masks with some inspiration from Kretz and Lindenstruth [2012] and the Intel intrinsic instructions for AVX-512 in ensemble propagation as an alternative to the `EnsembleTrait`. This alternative avoids looping explicitly on the samples and relying on the autovectorization, and it improves the readability of the code. Finally, we have illustrated how to use it to tackle the control-flow divergence in GMRES without reduction.

Although that we think that this alternative is promising as it eases the work of the compiler and may lead to more optimized code, this alternative requires more investigations. Those investigations include,

- a study of the combined use of expression templates and Intel intrinsic instructions; the mask implementation based on the `EnsembleTrait` supports the expression templates,
- the addition of SIMD intrinsics of other compiler vendors,
- the study of the best implementation with CUDA,
- a deep performance analysis on different architectures and compilers.

There is another important point to investigate. A current effort of Kretz [2018] is to define SIMD data types in the C++ standard, this effort includes the definition of mask in the C++ standard. This raises questions such as should the ensemble types rely or not on the future C++ standard and should the ensemble types use the standard mask?

The work of this chapter is tested on finite element models in Chapter 7.

Chapter 5

Mathematical formulation of the problems

In this thesis, we are interested in problems with non-symmetric or indefinite matrices arising from the discretization of PDEs. This chapter describes mathematically the two classes of problems considered in this work: contact problems and thermomechanical problems.

Section 5.1 describes contact problems and mesh-tying problems which have indefinite matrices. Being non-linear, contact problems raise new occurrences of ensemble divergence which have to be tackled. Those are discussed in the section 5.1.7.

Section 5.2 describes thermomechanical problems which have non-symmetric matrices due to the one-way coupling between the temperature and displacement fields.

Moreover, this chapter describes the preconditioners used in the two classes of problems.

The majority of the content of this chapter, including the theory, is not novel; this chapter is mainly a summary of existing references. However, the work discussed in section 5.1.7 is novel and relies on other sections of this chapter. The reading of the content of this chapter is not required to understand the results of the following chapters and the reader might want to move forward.

5.1 Mechanical problems including contact

The first class of problems considered in this work is the contact problems and the mesh-tying problems.

5.1.1 Strong form

The content of this section is based on [Wohlmuth, 2011, chap. 2], [Wriggers, 2006, chap. 3-4], and [Wiesner, 2015, chap. 5].

In this section, we assume an elastic behavior with small displacement assumptions and isotropic materials.

Two-bodies contact problem

We first name one of the bodies the *master* which constrains the *slave*, the other body. We denote by Ω^{s} and Ω^{m} the d-dimensional open bounded domain of $\mathbb{R}^d$ occupied by the deformable slave and master body in the undeformed configuration respectively. Moreover, we denote by Ω the union of Ω^{s} and Ω^{m}.

The boundaries of Ω^{s} and Ω^{m} are denoted by Γ^{s} and Γ^{m} respectively and are decomposed into six subsets $\Gamma^{\mathrm{s}}_{\mathbf{u}}, \Gamma^{\mathrm{s}}_{\boldsymbol{\sigma}}, \Gamma^{\mathrm{s}}_{\mathrm{c}}, \Gamma^{\mathrm{m}}_{\mathbf{u}}, \Gamma^{\mathrm{m}}_{\boldsymbol{\sigma}}$, and $\Gamma^{\mathrm{m}}_{\mathrm{c}}$ such that:

$$\Gamma^{\mathrm{s}} = \bar{\Gamma}^{\mathrm{s}}_{\mathbf{u}} \cup \bar{\Gamma}^{\mathrm{s}}_{\boldsymbol{\sigma}} \cup \bar{\Gamma}^{\mathrm{s}}_{\mathrm{c}}, \tag{5.1}$$

$$\Gamma^{\mathrm{m}} = \bar{\Gamma}^{\mathrm{m}}_{\mathbf{u}} \cup \bar{\Gamma}^{\mathrm{m}}_{\boldsymbol{\sigma}} \cup \bar{\Gamma}^{\mathrm{m}}_{\mathrm{c}}, \tag{5.2}$$

$$\Gamma_{\mathbf{u}} = \bar{\Gamma}^{\mathrm{s}}_{\mathbf{u}} \cup \bar{\Gamma}^{\mathrm{m}}_{\mathbf{u}}, \tag{5.3}$$

$$\Gamma_{\boldsymbol{\sigma}} = \bar{\Gamma}^{\mathrm{s}}_{\boldsymbol{\sigma}} \cup \bar{\Gamma}^{\mathrm{m}}_{\boldsymbol{\sigma}}, \tag{5.4}$$

$$\Gamma_{\mathrm{c}} = \bar{\Gamma}^{\mathrm{s}}_{\mathrm{c}} \cup \bar{\Gamma}^{\mathrm{m}}_{\mathrm{c}}, \tag{5.5}$$

where $\Gamma_{\mathbf{u}}$, $\Gamma_{\boldsymbol{\sigma}}$, and Γ_{c} stand respectively for the part of the boundary where Dirichlet boundary conditions are applied, the part of the boundary where Neumann boundary conditions are applied, and the potential contact surface. We denote by $\mathbf{n}$ the external outward unit normal.

To each point $\mathbf{x}$ of the potential contact interface $\Gamma^{\mathrm{s}}_{\mathrm{c}}$, we can associate $\boldsymbol{\gamma}(\mathbf{x})$ a point of $\Gamma^{\mathrm{m}}_{\mathrm{c}}$:

$$\boldsymbol{\gamma}(\mathbf{x}) = \underset{\mathbf{y}\in\Gamma^{\mathrm{m}}_{\mathrm{c}}}{\arg\min} \|\mathbf{x}-\mathbf{y}\|_2, \quad \text{on } \Gamma^{\mathrm{s}}_{\mathrm{c}}, \tag{5.6}$$

where $\|.\|_2$ stands for the Euclidean norm. If $\Gamma^{\mathrm{s}}_{\mathrm{c}}$ and $\Gamma^{\mathrm{m}}_{\mathrm{c}}$ describe two locally convex regions, we can prove that this association is unique [Wriggers, 2006, p. 59] for each point $\mathbf{x}$ of the potential contact interface $\Gamma^{\mathrm{s}}_{\mathrm{c}}$.

This association $(\mathbf{x}, \boldsymbol{\gamma}(\mathbf{x}))$ is useful to define the contact non-penetration condition which will be introduced later in this section. If the assumption of small deformations and small displacements does not hold, we have to update $\boldsymbol{\gamma}(\mathbf{x})$ based on the deformed configuration. However, if the assumption holds, we can compute this association on the undeformed configuration as written above and $\boldsymbol{\gamma}(\mathbf{x})$ can be computed once for all and will not depend on the displacements.

This association is not the only way to define the contact condition introduced later in this section. For example, in the literature, others define a gap function directly as follows [Pantuso et al., 2000]:

$$g(\mathbf{x}) = \inf_{\mathbf{y}\in\Gamma_c^m} \|\mathbf{x}-\mathbf{y}\|_2, \quad \text{on } \Gamma_c^s. \tag{5.7}$$

Another association found in the literature is defined using the intersection of a straight line through the point $\mathbf{x}$ parallel to the vector $\mathbf{n}$ and the surface Γ_c^m. It can be shown that this intersection is equivalent to $\boldsymbol{\gamma}(\mathbf{x})$ for sufficiently simple geometry.

The different notations are represented in Fig. 5.1.

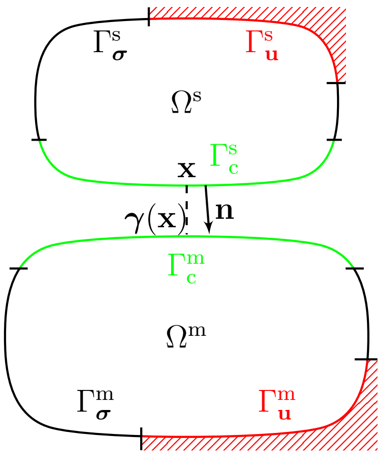


Figure 5.1: Geometrical configuration of the two bodies problem.

The mechanical stresses have to fulfill the local balance of momentum:

$$\mathbf{div_x}\,\boldsymbol{\sigma} + \mathbf{f} = \mathbf{0} \quad \text{in } \Omega, \tag{5.8}$$

where $\boldsymbol{\sigma}$ is the Cauchy stress tensor and $\mathbf{f}$ represents the external load per unit volume.

We assume small deformations, small displacements, and linear elastic isotropic behavior and we can write:

$$\boldsymbol{\sigma} = \mathbf{C}(\boldsymbol{\varepsilon}_\mathbf{x}\,\mathbf{u}) \quad \text{in } \Omega, \tag{5.9}$$

$$\boldsymbol{\varepsilon}_\mathbf{x}\,\mathbf{u} = \frac{1}{2}\left(\mathbf{D_x}\,\mathbf{u} + (\mathbf{D_x}\,\mathbf{u})^\mathrm{T}\right) \quad \text{in } \Omega, \tag{5.10}$$

where $\mathbf{C}$ is the fourth-order elasticity tensor, $\boldsymbol{\varepsilon}_\mathbf{x}\,\mathbf{u}$ the strain tensor associated to the displacement $\mathbf{u}$, and $\mathbf{D_x}$ is the gradient operator.

Neumann and homogenous Dirichlet boundary conditions are respectively represented as follows:

$$\boldsymbol{\sigma}(\mathbf{n}) = \mathbf{t} \quad \text{on } \Gamma_{\boldsymbol{\sigma}}, \tag{5.11}$$

$$\mathbf{u} = \mathbf{0} \quad \text{on } \Gamma_{\mathbf{u}}, \tag{5.12}$$

where $\mathbf{t}$ represents the surface load per unit surface.

In order to introduce the contact conditions, we introduce the gap between the two points $(\mathbf{x}, \boldsymbol{\gamma}(\mathbf{x}))$ after displacement:

$$g(\mathbf{x}) = (\mathbf{x} + \mathbf{u}(\mathbf{x}) - \boldsymbol{\gamma}(\mathbf{x}) - \mathbf{u}(\boldsymbol{\gamma}(\mathbf{x}))) \cdot \mathbf{n}(\mathbf{x}) = [\mathbf{u}(\mathbf{x})] \cdot \mathbf{n}(\mathbf{x}) - g_0 \quad \text{on } \Gamma_c^s, \tag{5.13}$$

where $[\mathbf{u}(\mathbf{x})] = \mathbf{u}(\mathbf{x}) - \mathbf{u}(\boldsymbol{\gamma}(\mathbf{x}))$ and $g_0 = (\mathbf{x} - \boldsymbol{\gamma}(\mathbf{x})) \cdot \mathbf{n}(\mathbf{x})$ is the initial gap, i.e., the gap when the body is not deformed and $\mathbf{x} + \mathbf{u}(\mathbf{x})$ is the position of the point $\mathbf{x}$ after the deformation.

We decompose the contact traction $\boldsymbol{l}$ as follows:

$$-\boldsymbol{\sigma}(\mathbf{n}) = \boldsymbol{l} = l\,\mathbf{n} + \boldsymbol{l}_T \quad \text{on } \Gamma_{\mathrm{c}}, \tag{5.14}$$

where $l\,\mathbf{n}$ and $\boldsymbol{l}_T$ stand respectively for the normal and tangential parts of the contact surface force. The presence of the minus sign implies that the value of l represents the contact pressure.

Contact can either have or not have friction. If a frictional contact is assumed, where the contact is locally closed, tangential forces are transferred between the master body and the slave one. In this case, in transient responses, the two bodies at a closed surface can either be sticking or sliding in the tangential direction. Such behavior introduces a second conditional statement compared to the frictionless contact conditions where no tangential forces are transferred. This new occurrence of ensemble divergence would have to be tackled in future work.

In this thesis, we have implemented two sets of contact conditions: the frictionless contact conditions in steady state and the sticking contact conditions in steady state. To ease the theoretical discussion, we only present the theory for the frictionless contact conditions in steady state:

- The slave body cannot penetrate into the master body:

$$g \leq 0 \quad \text{on } \Gamma_{\mathrm{c}}^{\mathrm{s}}. \tag{5.15}$$

- The stresses in normal direction developed on $\Gamma_{\mathrm{c}}^{\mathrm{s}}$ have to be compressive stresses or have to vanish:

$$l \geq 0 \quad \text{on } \Gamma_{\mathrm{c}}^{\mathrm{s}}. \tag{5.16}$$

- The stresses in normal direction developed on $\Gamma_{\mathrm{c}}^{\mathrm{s}}$ have to vanish when the gap is locally open, i.e. $g < 0$, and the gap has to be closed, i.e. $g = 0$, when the stresses in normal direction are compressive stresses:

$$l\,g = 0 \quad \text{on } \Gamma_{\mathrm{c}}^{\mathrm{s}}. \tag{5.17}$$

- The stresses in tangential directions developed on $\Gamma_{\mathrm{c}}^{\mathrm{s}}$ have to vanish if we assume frictionless contact, and the deformable body can move freely according to the tangential directions:

$$\boldsymbol{l}_T = \mathbf{0} \quad \text{on } \Gamma_{\mathrm{c}}^{\mathrm{s}}. \tag{5.18}$$

The combination of the first three conditions enforces $(g(\mathbf{x}), l(\mathbf{x}))$ to be such as represented in Fig. 5.2 for all $\mathbf{x} \in \Gamma_{\mathrm{c}}^{\mathrm{s}}$.

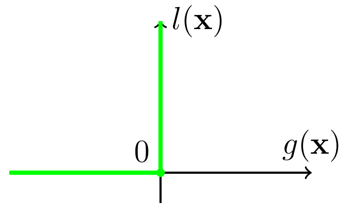


Figure 5.2: Admissible $(g(\mathbf{x}), l(\mathbf{x}))$ in green represented in $\mathbb{R}^2$.

The case of a rigid body is a particular case of the two-body contact problems where $\mathbf{u}(\boldsymbol{\gamma}(\mathbf{x})) = \mathbf{0}$. The slave body is the deformable body and the master body the rigid body. This notation can be interpreted as when we move the master surface in the direction of the slave surface, this slave surface has to be deformed to verify the non-penetration condition.

Mesh-tying problem

Mesh-tying problems can be seen as two body contact problems where the two bodies are glued together. In other words, the gap in all directions:

$$\mathbf{g}(\mathbf{x}) = \mathbf{x} + \mathbf{u}(\mathbf{x}) - \boldsymbol{\gamma}(\mathbf{x}) - \mathbf{u}(\boldsymbol{\gamma}(\mathbf{x})) = [\mathbf{u}(\mathbf{x})] \quad \text{on } \Gamma_\mathrm{c}^\mathrm{s}, \tag{5.19}$$

must be zero.

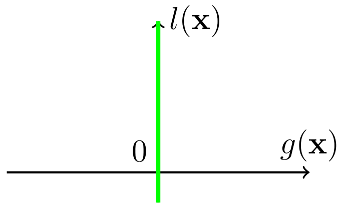


Figure 5.3: Admissible $(g(\mathbf{x}), l(\mathbf{x}))$ in green represented in $\mathbb{R}^2$.

5.1.2 Weak form

From the strong form we will derive variational inequalities such that the solution of the strong form is always a solution of the variational inequalities but the converse is not necessarily true.

First, we denote by $\boldsymbol{\mathcal{V}}$ the space of sufficiently regular displacement fields on Ω that satisfy the homogeneous Dirichlet boundary condition on $\Gamma_\mathbf{u}$.

And we define a subset $\boldsymbol{\mathcal{K}}$ of $\boldsymbol{\mathcal{V}}$ of sufficiently regular displacement fields on Ω that satisfy the homogeneous Dirichlet boundary condition on $\Gamma_\mathbf{u}$ and also the contact condition $g_0 - [\mathbf{v}] \cdot \mathbf{n} \geq 0$ on $\Gamma_\mathrm{c}^\mathrm{s}$. The subset $\boldsymbol{\mathcal{K}}$ is convex as for all $\mathbf{v}, \mathbf{w} \in \boldsymbol{\mathcal{K}}$, we have that $\alpha\,\mathbf{v} + (1-\alpha)\,\mathbf{w}$ verify the homogeneous Dirichlet boundary condition for any $\alpha \in [0,1]$ and we have that $g_0 - [\mathbf{v}] \cdot \mathbf{n} \geq 0$ and $g_0 - [\mathbf{w}] \cdot \mathbf{n} \geq 0$ and therefore $g_0 - [\alpha\,\mathbf{v} + (1-\alpha)\,\mathbf{w}] \cdot \mathbf{n} \geq 0$ is true for any $\alpha \in [0,1]$.

Starting with the balance law in strong form (5.8), we multiply it with $(\mathbf{v} - \mathbf{u})$, where $\mathbf{u}$ is the solution to the strong form and $\mathbf{v}$ is in $\boldsymbol{\mathcal{K}}$ and then we use the divergence theorem:

$$\int_\Omega \mathbf{C}(\boldsymbol{\varepsilon}_\mathbf{x}\,\mathbf{u}) : \boldsymbol{\varepsilon}_\mathbf{x}(\mathbf{v} - \mathbf{u}) dV = \int_\Omega \mathbf{f} \cdot (\mathbf{v} - \mathbf{u}) dV + \int_{\Gamma_{\boldsymbol{\sigma}}} \mathbf{t} \cdot (\mathbf{v} - \mathbf{u}) dS + \int_{\Gamma_\mathrm{c}} -l\,\mathbf{n} \cdot (\mathbf{v} - \mathbf{u}) dS. \tag{5.20}$$

The last term of (5.20) can be rewritten as follows:

$$\int_{\Gamma_\mathrm{c}} -l\,\mathbf{n} \cdot (\mathbf{v} - \mathbf{u}) dS = \int_{\Gamma_\mathrm{c}^\mathrm{s}} -l\,\mathbf{n} \cdot (\mathbf{v} - \mathbf{u}) dS + \int_{\Gamma_\mathrm{c}^\mathrm{m}} -l\,\mathbf{n} \cdot (\mathbf{v} - \mathbf{u}) dS \tag{5.21}$$

$$= \int_{\Gamma_\mathrm{c}^\mathrm{s}} -l\,\mathbf{n} \cdot ([\mathbf{v}] - [\mathbf{u}]) dS \tag{5.22}$$

Now, we can use the fact that $\mathbf{u}$ is the solution of the strong form, $\mathbf{v}$ is kinematically admissible, and the contact conditions to deduce that:

$$\begin{aligned} -l\,\mathbf{n}\cdot([\mathbf{v}]-[\mathbf{u}]) &= -l\,([\mathbf{v}]\cdot\mathbf{n}-g_0+g_0-[\mathbf{u}]\cdot\mathbf{n}) && (5.23)\\ &= -l\,([\mathbf{v}]\cdot\mathbf{n}-g_0)-l\,(g_0-[\mathbf{u}]\cdot\mathbf{n}) && (5.24)\\ &= l\,(g_0-[\mathbf{v}]\cdot\mathbf{n}) && (5.25)\\ &\geq 0, && (5.26)\end{aligned}$$

Therefore, we can bound the last term of (5.20) and deduce the following inequality:

$$\int_\Omega \mathbf{C}(\boldsymbol{\varepsilon}_\mathbf{x}\,\mathbf{u}) : \boldsymbol{\varepsilon}_\mathbf{x}(\mathbf{v}-\mathbf{u})dV \geq \int_\Omega \mathbf{f}\cdot(\mathbf{v}-\mathbf{u})dV + \int_{\Gamma_{\boldsymbol{\sigma}}} \mathbf{t}\cdot(\mathbf{v}-\mathbf{u})dS. \tag{5.27}$$

Therefore, we can write the problem as follows:

Find $\mathbf{u}\in\boldsymbol{\mathcal{K}}$ such that

$$\int_\Omega \mathbf{C}(\boldsymbol{\varepsilon}_\mathbf{x}\,\mathbf{u}) : \boldsymbol{\varepsilon}_\mathbf{x}(\mathbf{v}-\mathbf{u})dV \geq \int_\Omega \mathbf{f}\cdot(\mathbf{v}-\mathbf{u})dV + \int_{\Gamma_{\boldsymbol{\sigma}}} \mathbf{t}\cdot(\mathbf{v}-\mathbf{u})dS, \quad \forall\mathbf{v}\in\boldsymbol{\mathcal{K}}. \tag{5.28}$$

We can write the previous variational inequality more abstractly as follows:

Find $\mathbf{u}\in\boldsymbol{\mathcal{K}}$ such that

$$a(\mathbf{u},\mathbf{v}-\mathbf{u}) \geq f(\mathbf{v}-\mathbf{u}), \quad \forall\mathbf{v}\in\boldsymbol{\mathcal{K}}. \tag{5.29}$$

where a is the functional that associates to any displacement field $\mathbf{v}$ in $\boldsymbol{\mathcal{V}}$ and virtual displacement field $\mathbf{w}$ in $\boldsymbol{\mathcal{V}}$ the internal virtual work:

$$a(\mathbf{v},\mathbf{w}) = \int_\Omega \mathbf{C}(\boldsymbol{\varepsilon}_\mathbf{x}\,\mathbf{v}) : \boldsymbol{\varepsilon}_\mathbf{x}\mathbf{w}dV, \tag{5.30}$$

and f is the functional that associates to any virtual displacement field $\mathbf{w}$ in $\boldsymbol{\mathcal{V}}$ the external virtual work:

$$f(\mathbf{v}) = \int_\Omega \mathbf{f}\cdot\mathbf{v}dV + \int_{\Gamma_{\boldsymbol{\sigma}}} \mathbf{t}\cdot\mathbf{v}dS. \tag{5.31}$$

A variational inequality of this type, which involves seeking a solution in a convex set such that an inequality is satisfied for all test functions in that convex set, is called a variational inequality of the first kind in the literature [Glowinski, 1984, P_1]. The Lions and Stampacchia theorem can be used to show the existence and uniqueness of the solution of this variational inequality [Glowinski, 1984, Theorem 3.1].

We can, now, introduce the indicator functional $\chi_{\boldsymbol{\mathcal{K}}}$ of $\boldsymbol{\mathcal{K}}$ defined as follows:

$$\chi_{\boldsymbol{\mathcal{K}}}(\mathbf{v}) = \begin{cases} +\infty & \text{if } \mathbf{v}\notin\boldsymbol{\mathcal{K}} \\ 0 & \text{if } \mathbf{v}\in\boldsymbol{\mathcal{K}} \end{cases}. \tag{5.32}$$

We can use the indicator functional to write the variational inequality of the first kind as a variational inequality of the second kind:

Find $\mathbf{u} \in \boldsymbol{\mathcal{V}}$ such that

$$a(\mathbf{u}, \mathbf{v} - \mathbf{u}) + \chi_{\boldsymbol{\mathcal{K}}}(\mathbf{v}) - \chi_{\boldsymbol{\mathcal{K}}}(\mathbf{u}) \geq f(\mathbf{v} - \mathbf{u}), \quad \forall \mathbf{v} \in \boldsymbol{\mathcal{V}}. \tag{5.33}$$

Any variational inequality of the first kind can be written as a variational inequality of the second kind. However, the set of variational inequalities of the second kind is larger than the set of variational inequalities of the first kind because it includes variational inequalities for which the functional $\chi_{\boldsymbol{\mathcal{K}}}$ is replaced by a more complex functional.

Once again, this variational inequality has a unique solution following [Glowinski, 1984, Theorem 4.1].

Minimization problem

We can define the potential energy functional $J(\mathbf{v})$ as follows:

$$J(\mathbf{v}) = \frac{1}{2} a(\mathbf{v}, \mathbf{v}) - f(\mathbf{v}). \tag{5.34}$$

We can show now that the potential energy functional at all $\mathbf{v} \in \boldsymbol{\mathcal{K}}$ is bounded by the potential energy functional at the solution of the variational inequality of the first kind:

$$\begin{aligned} J(\mathbf{v}) &= \frac{1}{2} a(\mathbf{u} + \mathbf{v} - \mathbf{u}, \mathbf{u} + \mathbf{v} - \mathbf{u}) - f(\mathbf{u} + \mathbf{v} - \mathbf{u}) && (5.35) \\ &= J(\mathbf{u}) + a(\mathbf{u}, \mathbf{v} - \mathbf{u}) - f(\mathbf{v} - \mathbf{u}) + \frac{1}{2} a(\mathbf{v} - \mathbf{u}, \mathbf{v} - \mathbf{u}) && (5.36) \\ &\geq J(\mathbf{u}) + a(\mathbf{u}, \mathbf{v} - \mathbf{u}) - f(\mathbf{v} - \mathbf{u}) \qquad \geq J(\mathbf{u}). && (5.37) \end{aligned}$$

Therefore, we can write the variational inequality of the first kind as an equivalent minimization problem:

Find $\mathbf{u} \in \boldsymbol{\mathcal{K}}$ such that

$$J(\mathbf{u}) \leq J(\mathbf{v}), \quad \forall \mathbf{v} \in \boldsymbol{\mathcal{K}}. \tag{5.38}$$

The previous energy-minimization problem can be written as follows:

Find $\mathbf{u} \in \boldsymbol{\mathcal{V}}$ such that

$$\begin{aligned} \mathbf{u} = \arg \inf_{\mathbf{v} \in \boldsymbol{\mathcal{V}}} \quad & \frac{1}{2} \int_{\Omega} \mathbf{C}(\boldsymbol{\varepsilon}_{\mathbf{x}} \mathbf{v}) : \boldsymbol{\varepsilon}_{\mathbf{x}} \mathbf{v} dV - \int_{\Omega} \mathbf{f} \cdot \mathbf{v} dV - \int_{\Gamma_{\boldsymbol{\sigma}}} \mathbf{t} \cdot \mathbf{v} dS && (5.39) \\ \text{such that} \quad & [\mathbf{v}] \cdot \mathbf{n} - g_0 \leq 0 \text{ on } \Gamma_{\mathrm{c}}^{\mathrm{s}} && (5.40) \end{aligned}$$

Saddle-point problem

We introduce a Lagrange multiplier $z \geq 0$ on $\Gamma_{\mathrm{c}}^{\mathrm{s}}$. If $\mathbf{v} \in \boldsymbol{\mathcal{K}}$, we have that, for any Lagrange multiplier $z \geq 0$ on $\Gamma_{\mathrm{c}}^{\mathrm{s}}$, we have:

$$\int_{\Gamma_{\mathrm{c}}^{\mathrm{s}}} ([\mathbf{v}] \cdot \mathbf{n} - g_0) \, z \, dS \leq 0. \tag{5.41}$$

Moreover, we know that there exist at least one $z \geq 0$ on Γ_c such that:

$$\int_{\Gamma_\mathrm{c}^\mathrm{s}} ([\mathbf{v}] \cdot \mathbf{n} - g_0)\, z\, dS \ = 0, \tag{5.42}$$

as it would be true for $z = 0$ on $\Gamma_\mathrm{c}^\mathrm{s}$ which obviously verifies $z \geq 0$ on $\Gamma_\mathrm{c}^\mathrm{s}$.

Therefore, we can write that:

$$\sup_{z \geq 0 \text{ on } \Gamma_\mathrm{c}^\mathrm{s}} \int_{\Gamma_\mathrm{c}^\mathrm{s}} ([\mathbf{v}] \cdot \mathbf{n} - g_0)\, z\, dS \ = 0, \ \forall \mathbf{v} \in \boldsymbol{\mathcal{K}}. \tag{5.43}$$

Moreover, if we consider $\mathbf{v} \in \boldsymbol{\mathcal{V}} \setminus \boldsymbol{\mathcal{K}}$, we have that $\mathbf{v} \cdot \mathbf{n} - g_0 > 0$ at some points on $\Gamma_{\mathrm{c}^\mathrm{s}}$ and we can deduce that:

$$\sup_{z \geq 0 \text{ on } \Gamma_\mathrm{c}^\mathrm{s}} \int_{\Gamma_\mathrm{c}^\mathrm{s}} ([\mathbf{v}] \cdot \mathbf{n} - g_0)\, z\, dS \ = +\infty, \ \forall \mathbf{v} \in \boldsymbol{\mathcal{V}} \setminus \boldsymbol{\mathcal{K}}. \tag{5.44}$$

These observations allow us to write the previous problem as follows:

Find $\mathbf{u} \in \boldsymbol{\mathcal{V}}$ such that

$$\begin{aligned} \mathbf{u} = \arg \inf_{\mathbf{v} \in \boldsymbol{\mathcal{V}}} \sup_{z \geq 0 \text{ on } \Gamma_\mathrm{c}^\mathrm{s}} \frac{1}{2} \int_\Omega \mathbf{C}(\boldsymbol{\varepsilon}_\mathbf{x}\, \mathbf{v}) : \boldsymbol{\varepsilon}_\mathbf{x} \mathbf{v} dV - \int_\Omega \mathbf{f} \cdot \mathbf{v} dV - \int_{\Gamma_{\boldsymbol{\sigma}}} \mathbf{t} \cdot \mathbf{v} dS \\ + \int_{\Gamma_\mathrm{c}^\mathrm{s}} ([\mathbf{v}] \cdot \mathbf{n} - g_0)\, z dS \end{aligned} \tag{5.45}$$

which leads us to the saddle-point problem:

Find $\mathbf{u} \in \boldsymbol{\mathcal{V}}$ and $l \geq 0$ on $\Gamma_\mathrm{c}^\mathrm{s}$ such that

$$\int_\Omega \mathbf{C}(\boldsymbol{\varepsilon}_\mathbf{x}\, \mathbf{u}) : \boldsymbol{\varepsilon}_\mathbf{x} \mathbf{v} dV + \int_{\Gamma_\mathrm{c}^\mathrm{s}} [\mathbf{v}] \cdot \mathbf{n}\, l dS = \int_\Omega \mathbf{f} \cdot \mathbf{v} dV + \int_{\Gamma_{\boldsymbol{\sigma}}} \mathbf{t} \cdot \mathbf{v} dS, \ \forall \mathbf{v} \in \boldsymbol{\mathcal{V}} \tag{5.46}$$

$$\int_{\Gamma_\mathrm{c}^\mathrm{s}} ([\mathbf{u}] \cdot \mathbf{n} - g_0)\, (z - l) dS \leq 0, \ \forall z \geq 0 \text{ on } \Gamma_\mathrm{c}^\mathrm{s} \tag{5.47}$$

We can write the previous saddle-point problem more abstractly as follows:

Find $\mathbf{u} \in \boldsymbol{\mathcal{V}}$ and $l \geq 0$ on $\Gamma_\mathrm{c}^\mathrm{s}$ such that

$$a(\mathbf{u}, \mathbf{v}) \ + \ b(\mathbf{v}, \boldsymbol{l}) = f(\mathbf{v}) \qquad \forall \mathbf{v} \in \boldsymbol{\mathcal{V}}, \tag{5.48}$$

$$b(\mathbf{u}, \boldsymbol{z} - \boldsymbol{l}) \qquad \leq g(\boldsymbol{z} - \boldsymbol{l}) \quad \forall z \geq 0 \text{ on } \Gamma_\mathrm{c}^\mathrm{s} \tag{5.49}$$

where a and f are defined in (5.30) and (5.31) respectively, b is the functional that associates to any displacement field $\mathbf{v}$ and contact force $\boldsymbol{z}$ their virtual work,

$$b(\mathbf{v}, \boldsymbol{z}) = \int_{\Gamma_\mathrm{c}^\mathrm{s}} [\mathbf{v}] \cdot \boldsymbol{z} dS, \tag{5.50}$$

and g is as follows:

$$g(\boldsymbol{z}) = \int_{\Gamma_\mathrm{c}^\mathrm{s}} g_0\, \boldsymbol{z} \cdot \mathbf{n} dS. \tag{5.51}$$

Mesh-tying problem

The main difference between the mesh-tying problem and the contact problem is that the set $\boldsymbol{\mathcal{K}}$ of sufficiently regular displacement fields on Ω that satisfy the homogeneous Dirichlet boundary condition on $\Gamma_{\mathbf{u}}$ and also the mesh-tying condition is a linear space and not only a convex cone as in the contact problem case.

Therefore, following the same approach as in the previous subsection, we can deduce the mesh-tying problem in saddle-point formulation:

Find $\mathbf{u} \in \boldsymbol{\mathcal{V}}$ and $\boldsymbol{l}$ such that

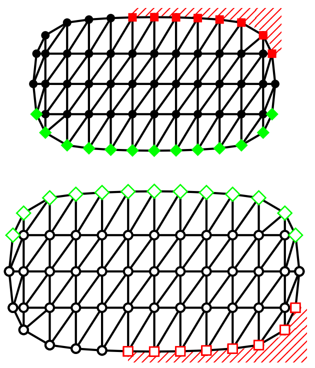
$$a(\mathbf{u}, \mathbf{v}) + b(\mathbf{v}, \boldsymbol{l}) = f(\mathbf{v}) \quad \forall \mathbf{v} \in \boldsymbol{\mathcal{V}}, \tag{5.52}$$

$$b(\mathbf{u}, \boldsymbol{z}) = 0 \quad \forall \boldsymbol{z} \tag{5.53}$$

where a, b, and f are defined in (5.30), (5.50), and (5.31) respectively.

5.1.3 Discretization

We note $T^{h,\mathrm{s}}$ and $T^{h,\mathrm{m}}$ a mesh of Ω^{s} and Ω^{m} respectively as shown in Fig. 5.4 with a mesh characteristic size denotes by h, $P^{h,\mathrm{s}}$ and $P^{h,\mathrm{m}}$ the set of all vertices of $T^{h,\mathrm{s}}$ and $T^{h,\mathrm{m}}$ not being on $\bar{\Gamma}^{\mathrm{s}}_{\mathbf{u}}$ and $\bar{\Gamma}^{\mathrm{m}}_{\mathbf{u}}$ respectively, $P^{h,\mathrm{s}}_{\mathbf{u}}$ and $P^{h,\mathrm{m}}_{\mathbf{u}}$ the set of all vertices of $T^{h,\mathrm{s}}$ and $T^{h,\mathrm{s}}$ on $\bar{\Gamma}^{\mathrm{s}}_{\mathbf{u}}$ and $\bar{\Gamma}^{\mathrm{m}}_{\mathbf{u}}$ respectively, and $P^{h,\mathrm{s}}_{\mathrm{c}}$ the set of all vertices of $P^{h,\mathrm{s}}$ on $\bar{\Gamma}^{\mathrm{s}}_{\mathrm{c}}$ as illustrated in Fig. 5.4.



◆ slave nodes, $P^{h,\mathrm{s}}_{\mathrm{c}}$,

• slave inner nodes,

■ Dirichlet slave nodes, $P^{h,\mathrm{s}}_{\mathbf{u}}$,

◇ master nodes, $P^{h,\mathrm{m}}_{\mathrm{c}}$,

○ master inner nodes,

□ Dirichlet master nodes, $P^{h,\mathrm{m}}_{\mathbf{u}}$.

Figure 5.4: Mesh of the two-body contact problem.

The Mortar Finite Element Method is a mixed (or hybrid) FEM where both the displacement $\mathbf{u}$ (primal variable) and the surface forces on the potential contact surface $\boldsymbol{l}$ (dual variable) are discretized using shape functions. For the primal variable, we use linear finite elements and we denote by $S^1(\Omega, T^h)$ the finite element space for linear finite elements associated with T^h

For the primal variable, we use linear finite elements and we denote by $S^1(\Omega, T^{h,\mathrm{s}}) \times S^1(\Omega, T^{h,\mathrm{m}})$ the finite element space for linear finite elements associated with $T^{h,\mathrm{s}}$ and $T^{h,\mathrm{m}}$.

The variables are discretized as follows:

$$\mathbf{u}^h = \sum_{p \in P^{h,\mathrm{s}}} \mathbf{u}_p\, \phi_p + \sum_{p \in P^{h,\mathrm{m}}} \mathbf{u}_p\, \phi_p, \qquad \mathbf{v}^h = \sum_{p \in P^{h,\mathrm{s}}} \mathbf{v}_p\, \phi_p + \sum_{p \in P^{h,\mathrm{m}}} \mathbf{v}_p\, \phi_p, \tag{5.54}$$

$$\boldsymbol{l}^h = \sum_{p \in P^{h,\mathrm{s}}_{\mathrm{c}}} \boldsymbol{l}_p\, \psi_p, \qquad \boldsymbol{z}^h = \sum_{p \in P^{h,\mathrm{s}}_{\mathrm{c}}} \boldsymbol{z}_p\, \psi_p, \tag{5.55}$$

where ϕ_p is the standard scalar first-order nodal basis function associated to the vertex p, i.e. the usual d-dimension hat function, and ψ_p will be defined in the next section.

It can be seen that the primal variables are discretized on both volumes and the Lagrange multipliers are only discretized on the slave side. This can be explained by the fact that the constraint $q \leq 0$ is only defined on $\Gamma_\mathrm{c}^\mathrm{s}$.

The discrete saddle point-problem can now be written as follows:

Find $\mathbf{u}^h$ and $\boldsymbol{l}^h$ with $\boldsymbol{l}_{T,p} = \mathbf{0}$ and $\boldsymbol{l}_p \cdot \boldsymbol{n} \geq 0$, $\forall p \in P_\mathrm{c}^{h,\mathrm{s}}$ such that

$$\begin{aligned} a(\mathbf{u}^h, \mathbf{v}^h) + b(\mathbf{v}^h, \boldsymbol{l}^h) &= f(\mathbf{v}^h) && \forall \mathbf{v}^h, && (5.56)\\ b(\mathbf{u}^h, \boldsymbol{z}^h - \boldsymbol{l}^h) &\leq g(\boldsymbol{z}^h - \boldsymbol{l}^h) && \forall \boldsymbol{z}^h : \boldsymbol{z}_{T,p} = \mathbf{0},\ \boldsymbol{z}_p \cdot \boldsymbol{n} \geq 0,\ \forall p \in P_\mathrm{c}^{h,\mathrm{s}}. && (5.57) \end{aligned}$$

Writing $\boldsymbol{u}$ and $\boldsymbol{l}$ the column vector corresponding to the concatenation of the $\mathbf{u}_p$ and $\boldsymbol{l}_p$, we can write the previous problem as follows:

Find $\mathbf{u}$ and $\boldsymbol{l}$ such that

$$\begin{cases} \boldsymbol{K}\boldsymbol{u} + \boldsymbol{G}\boldsymbol{l} = \boldsymbol{f} \\ g_q \leq 0, g_q\, l_q = 0, l_q \geq 0, \boldsymbol{l}_{T,q} = \mathbf{0}, \forall q \in P_\mathrm{c}^{h,\mathrm{s}} \end{cases}, \tag{5.58}$$

where

$$g_q := \int_{\Gamma_\mathrm{c}^\mathrm{s}} ([\mathbf{u}^h] \cdot \mathbf{n} - g_0^h)\, \psi_q dS, \tag{5.59}$$

and $\boldsymbol{K}$, $\boldsymbol{G}$, $\boldsymbol{f}$ are defined such that

$$\begin{aligned} \boldsymbol{w}^\mathrm{T}\, \boldsymbol{K}\, \boldsymbol{u} &= a(\mathbf{w}^h, \mathbf{u}^h), && (5.60)\\ \boldsymbol{w}^\mathrm{T}\, \boldsymbol{G}\, \boldsymbol{l} &= b(\mathbf{w}^h, \boldsymbol{l}^h), && (5.61)\\ \boldsymbol{w}^\mathrm{T}\, \boldsymbol{f} &= f(\mathbf{w}^h). && (5.62) \end{aligned}$$

Mesh-tying problem

In the case of the mesh-tying problem, the previous problem is written as follows:

Find $\mathbf{u}$ and $\boldsymbol{l}$ such that

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{G} \\ \boldsymbol{G}^\mathrm{T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{l} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \mathbf{0} \end{bmatrix}, \tag{5.63}$$

where $\boldsymbol{K}$, $\boldsymbol{G}$, $\boldsymbol{f}$ are defined such that

$$\begin{aligned} \boldsymbol{w}^\mathrm{T}\, \boldsymbol{K}\, \boldsymbol{u} &= a(\mathbf{w}^h, \mathbf{u}^h), && (5.64)\\ \boldsymbol{w}^\mathrm{T}\, \boldsymbol{G}\, \boldsymbol{l} &= b(\mathbf{w}^h, \boldsymbol{l}^h), && (5.65)\\ \boldsymbol{w}^\mathrm{T}\, \boldsymbol{f} &= f(\mathbf{w}^h). && (5.66) \end{aligned}$$

5.1.4 Lagrange multipliers shape functions

We will now discuss possible choices for ψ_p.

In this section, even if we have always assumed that $\bar{\Gamma}_\mathrm{c} \cap \bar{\Gamma}_\mathbf{u} = \emptyset$, we will discuss the modification of the shape functions if this assumption is not verified to link with previous literature on the Mortar method.

Standard shape function

The first shape function that has been used in the Mortar method, [Bernardi et al., 1993], is the so-called *standard shape function* ψ_p.

The standard shape function ψ_p is defined as the trace on $\Gamma_\mathrm{c}^\mathrm{s}$ of the d-dimensional scalar standard nodal basis function ϕ_p used in the body and associated to the node p:

$$\psi_p := \phi_p \,|_{\Gamma_\mathrm{c}^\mathrm{s}}, \; \forall p \in P_\mathrm{c}^{h,\mathrm{s}}. \tag{5.67}$$

They are shown in Fig. 5.5. These shape functions have been used in [Puso, 2004],[Puso and Laursen, 2004a], and [Puso and Laursen, 2004b].

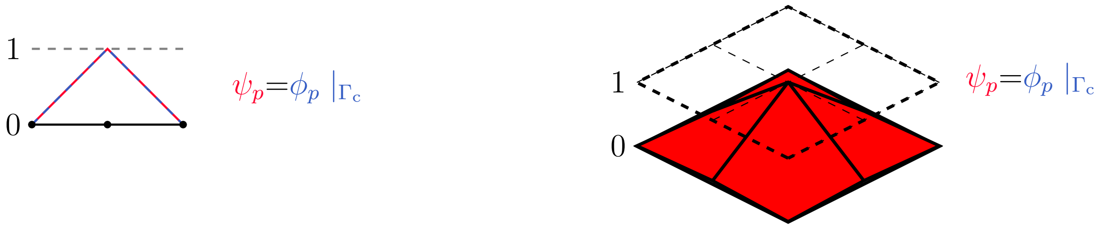

1
0
ψ_p=ϕ_p $|_{\Gamma_\mathrm{c}}$
1
0
ψ_p=ϕ_p $|_{\Gamma_\mathrm{c}}$

Figure 5.5: Standard shape functions in 2D and 3D: in both cases the shape function of the Lagrange multiplier ψ_p is equal to the trace on $\Gamma_\mathrm{c}^\mathrm{s}$ of the d-dimensional scalar standard nodal basis function ϕ_p used in the body and associated to the node p.

As we have restricted ourselves on the case of scalar standard first-order nodal basis function, one advantage of these shape functions is that they are positive or null on their support.

In order to not have an overconstrained problem, we cannot associate Lagrange shape functions to the nodes constrained with Dirichlet boundary conditions. However, we want our discretization to be able to reproduce the constant force between the body and the rigid body, i.e. we need a partition of unity, to guarantee optimal convergence [Flemisch and Wohlmuth, 2007]. Therefore, we need to modify the shape functions in the neighborhood of the Dirichlet nodes. A possible way is to replace the part of the shape function on the element including a Dirichlet node by a constant unitary function divided by the number of non-Dirichlet constrained nodes of the mesh on this element. Therefore we will have:

$$\sum_{p \in P_\mathrm{c}^{h,\mathrm{s}}} \psi_p(\mathbf{x}) = 1, \forall \mathbf{x} \in \Gamma_\mathrm{c}^\mathrm{s}. \tag{5.68}$$

Such modifications are shown in Fig. 5.6.

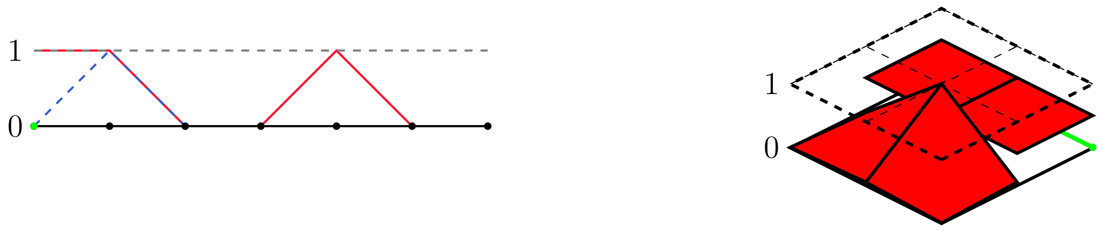


Figure 5.6: Modified standard shape functions in 2D and 3D: Dirichlet conditions are applied at the green point in 2D and on the green line in 3D. The Lagrange multiplier shape functions are represented in red. In both cases, the modifications of the Lagrange multiplier shape functions are such that the sum of all the shape functions equals 1 everywhere. In the 3D example, the plateau has to be equal to 0.5 as 2 shape functions contribute to each half of this plateau.

It can be shown that such shape functions verify the discrete inf-sup condition [Belgacem, 1999] and that they lead to a convergence $O(h)$ in the energy norm $a(\mathbf{u}-\mathbf{u}^h, \mathbf{u}-\mathbf{u}^h)$ for frictionless problems where h is the mesh characteristic size.

Dual discontinuous shape function

After that, a second type of shape functions has been introduced in [Wohlmuth, 2000]. These new functions ψ_p have been built such that they have the following property:

- They verify a biorthogonality property with the primal shape function ϕ_p:

$$\int_{\Gamma_{\mathrm{c}}^{\mathrm{s}}} \phi_p \, \psi_q \, dS = \delta_{pq} \int_{\Gamma_{\mathrm{c}}^{\mathrm{s}}} \phi_p \, dS, \; \forall p \in P_{\mathrm{c}}^{h,\mathrm{s}}, \forall q \in P_{\mathrm{c}}^{h,\mathrm{s}} \tag{5.69}$$

  where δ_{pq} is the Kronecker delta. This property enforces the partition of unity.

- They have the same support as the standard shape function:

$$\operatorname{supp}(\phi_p) \cap \Gamma_{\mathrm{c}}^{\mathrm{s}} \equiv \operatorname{supp}(\psi_p), \forall p \in P_{\mathrm{c}}^{h,\mathrm{s}}. \tag{5.70}$$

- They are not necessarily continuous at the vertices. Continuity enforcement, as in the standard shape function case, is not necessary to have a stable method [Wohlmuth, 2000, p. 992].

There exist different choices of basis functions which satisfy the previous constraints. In this thesis we will focus on the most popular choice, the so-called *dual discontinuous shape function* introduced in [Wohlmuth, 2000]. These functions are obtained *by an element-wise biorthogonalization process of the local nodal finite elements followed by a node-wise glueing step* [Wohlmuth, 2011].

We can do the element-wise biorthogonalization process writing for an element e of the interface:

$$\psi_q \mid_e := \sum_{p \in P_e} c_{pq} \, \phi_p \mid_e, \; \forall q \in P_e, \tag{5.71}$$

where P_e stands for the set of all mesh nodes included in the element e, $f \mid_e$ for the restriction of f on the element e, and where the coefficients c_{pq} should verify, i.e. be the solution of the element mass matrix system:

$$\sum_{r \in P_e} \left(\int_e \phi_r \, \phi_p \, dS \right) c_{rq} = \delta_{pq} \int_e \phi_p \, dS. \tag{5.72}$$

Then, we can do the node-wise glueing step:

$$\psi_q := \sum_e \psi_q \mid_e . \tag{5.73}$$

This gives us the dual discontinuous shape functions shown in Fig. 5.7.

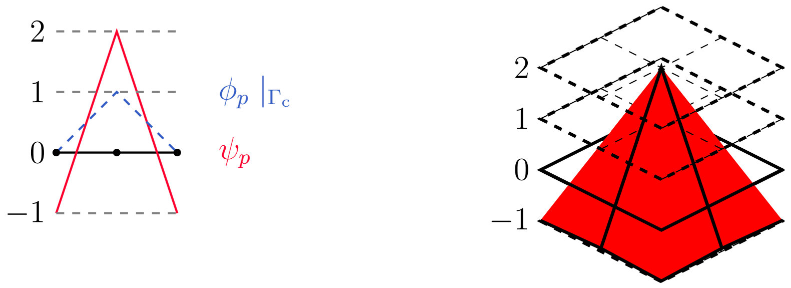


Figure 5.7: Dual discontinuous shape functions in 2D and 3D.

As in the previous case, we have to change the shape functions in the neighborhood of the Dirichlet nodes as shown in Fig. 5.8. Such a modification allows the partition of unity without violating the biorthogonality property. We have to stress the fact that the biorhogonality property has to be verified for each couple $(p, q) \in P_c^{h,s} \times P_c^{h,s}$ and $P_c^{h,s}$ has been defined as a subspace of $P^{h,s}$ where none Dirichlet nodes are included. Therefore, a modification on the neighborhood of the Dirichlet nodes does not violate the biorthogonality property.

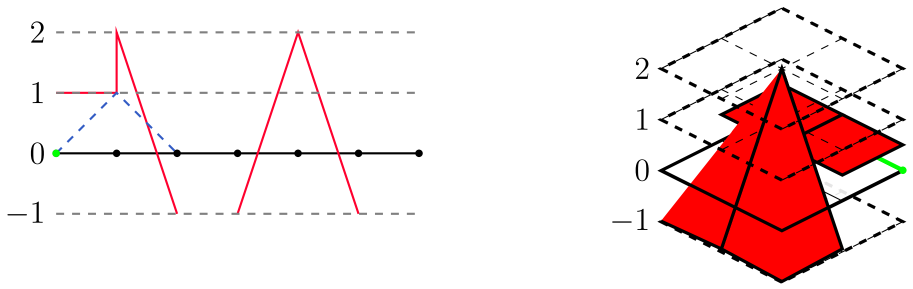


Figure 5.8: Modified dual discontinuous shape functions in 2D and 3D.

The main advantage of this approach is that the biorthogonality is useful to simplify the system to solve and the active set strategy discussed later in section 5.1.5. The biorthogonality property enforces the diagonality of the coupling matrix between primal and dual variables, this makes the problem sparser and even allows the analytical computation of the inverse of the coupling matrix.

The inf-sup condition is theoretically proved for the discontinuous dual shape functions, proof in [Wohlmuth, 2000] and the method continues to converge in $O(h)$ in the energy norm for frictionless problems [Wohlmuth, 2000].

There exist two difficulties concerning the use of dual shape functions, which are not shared with standard shape functions known in the literature [Popp et al., 2013b]. The main issue is when the mesh is not sufficiently fine on a curved interface, this issue has been observed in [Flemisch et al., 2005] and [Flemisch and Wohlmuth, 2007] too. A possible solution for this issue is to use a Petrov-Galerkin formulation and use standard shape functions for the weight Lagrange multiplier $\boldsymbol{z}$ [Popp et al., 2013b]. The problem is no longer a standard saddle-point problem, but we unify the advantages of both shape functions [Popp et al., 2013b]. It can be at least partially proved theoretically that the good properties of the problem are preserved and therefore, we keep the validation of the inf-sup condition and good convergence property. However, using Petrov-Galerkin, we lose the symmetry of the system.

5.1.5 Active set strategy

In this section we will derive the algorithm to solve the discrete problem:

$$\begin{cases} \boldsymbol{K}\boldsymbol{u} + \boldsymbol{G}\boldsymbol{l} = \boldsymbol{f} \\ g_q \leq 0, g_q\, l_q = 0,\ l_q \geq 0,\ \boldsymbol{l}_{T,q} = \boldsymbol{0},\ \forall q \in P_{\mathrm{c}}^{h,\mathrm{s}} \end{cases}. \tag{5.74}$$

As (5.74) has inequalities, the system is nonlinear and cannot be solved directly using a linear solver.

We will derive an iterative strategy based on a semi-smooth nonlinear complementary function and a semi-smooth Newton method.

We introduce a positive constant c and we rewrite (5.74) as follows:

$$\begin{cases} \boldsymbol{K}\boldsymbol{u} + \boldsymbol{G}\boldsymbol{l} = \boldsymbol{f} \\ C(l_q, \mathbf{u}^h) = 0,\ \boldsymbol{l}_{T,q} = \boldsymbol{0},\ \forall q \in P_{\mathrm{c}}^{h,\mathrm{s}} \end{cases}, \tag{5.75}$$

where $C(l_q, \mathbf{u}^h)$ is a semi-smooth complementary function defined as follows:

$$C(l_q, \mathbf{u}^h) := l_q - \max(0, l_q + c\, g_q). \tag{5.76}$$

We have that $x \mapsto \max(0, x)$ is Lipschitz continuous and nondifferentiable at the origin, however, as it is semi-smooth we can define its derivative as follows:

$$\frac{d}{dx}\max(0, x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}. \tag{5.77}$$

Based on the definition (5.76) and the derivative (5.77), we can define two different sets of node indices which regroup the nodes where the derivatives of $\max(0, l_q + c\, g_q)$ will be zero and non-zero; they are respectively named the inactive set $\mathscr{I}$ and the active set $\mathscr{A}$:

$$\mathscr{I} = \left\{q \in P_{\mathrm{c}}^{h,\mathrm{s}} : l_q + c\, g_q \leq 0\right\}, \tag{5.78}$$

$$\mathscr{A} = \left\{q \in P_{\mathrm{c}}^{h,\mathrm{s}} : l_q + c\, g_q > 0\right\}. \tag{5.79}$$

For a given inactive set and active set, we can deduce a block partition of $\boldsymbol{G}$ as follows:

$$\boldsymbol{G} = [\boldsymbol{G}_{\mathscr{I}} \mid \boldsymbol{G}_{\mathscr{A}}], \tag{5.80}$$

where $\boldsymbol{G}_{\mathscr{I}}$ and $\boldsymbol{G}_{\mathscr{A}}$ stand for the columns of $\boldsymbol{G}$ associated to the degrees of freedom of all nodes of $\mathscr{I}$ and $\mathscr{A}$ respectively.

We can now derive the semi-smooth Newton method to solve (5.75). For that we start from an initial guess $\begin{bmatrix}\mathbf{u}^{0\mathrm{T}} & \boldsymbol{l}_{\mathscr{I}}^{0\;\mathrm{T}} & \boldsymbol{l}_{\mathscr{A}}^{0\;\mathrm{T}}\end{bmatrix}^{\mathrm{T}}$ which has to be iteratively corrected:

$$\begin{bmatrix}\mathbf{u}^{k+1}\\ \boldsymbol{l}_{\mathscr{I}}^{k+1}\\ \boldsymbol{l}_{\mathscr{A}}^{k+1}\end{bmatrix} = \begin{bmatrix}\Delta\mathbf{u}^{k}\\ \Delta\boldsymbol{l}_{\mathscr{I}}^{k}\\ \Delta\boldsymbol{l}_{\mathscr{A}}^{k}\end{bmatrix} + \begin{bmatrix}\mathbf{u}^{k}\\ \boldsymbol{l}_{\mathscr{I}}^{k}\\ \boldsymbol{l}_{\mathscr{A}}^{k}\end{bmatrix}, \tag{5.81}$$

to impose that:

$$C(l_q^k, \mathbf{u}^{hk}) + \Delta C(l_q^k, \mathbf{u}^{hk}) = 0. \tag{5.82}$$

Based on the status of a node q, either active or inactive, i.e. either closed or open gap, the expression (5.82) imposes different corrections. Therefore, we will first derive the two corrections and then regroup the results:

- If the gap is closed at node q, node q is active and $q \in \mathscr{A}$:

$$C(l_q^k, \mathbf{u}^{hk}) + \Delta C(l_q^k, \mathbf{u}^{hk}) = 0 \tag{5.83}$$
$$\Rightarrow$$
$$g_q^{k+1} = 0 \tag{5.84}$$
$$\Rightarrow$$
$$\boldsymbol{N}_{\mathscr{A}}\,\Delta\mathbf{u}^k = -\,(\boldsymbol{N}_{\mathscr{A}}\,\mathbf{u}^k - \mathbf{g}_{0,\mathscr{A}}), \tag{5.85}$$

where g_q^{k+1} is g_q at the Newton iteration $k+1$.

Using the constraint on the tangential part of the multiplier, we have:

$$\boldsymbol{T}_{\mathscr{A}}\,\Delta\boldsymbol{l}_{\mathscr{A}}^k = -\boldsymbol{T}_{\mathscr{A}}\,\boldsymbol{l}_{\mathscr{A}}^k \tag{5.86}$$

where $\boldsymbol{N}$, $\boldsymbol{T}$ are defined as follows:

$$\mathbf{v}_s^{\mathrm{T}}\,\boldsymbol{N}\,\boldsymbol{l} = \int_{\Gamma_{\mathrm{c}}^{\mathrm{s}}} \mathbf{v}^h \cdot \mathbf{n}\,l^h\,dS, \tag{5.87}$$

$$\boldsymbol{z}\,\boldsymbol{T}\,\boldsymbol{l} = \int_{\Gamma_{\mathrm{c}}^{\mathrm{s}}} \boldsymbol{z}_T^h \cdot \boldsymbol{l}_T^h\,dS, \tag{5.88}$$

and $\mathbf{g}_0$ is a vector corresponding to the concatenation of $\int_{\Gamma_{\mathrm{c}}^{\mathrm{s}}} g_0^h\,\psi_q dS$ for all $q \in P_{\mathrm{c}}^{h,\mathrm{s}}$.

- If the gap at node q is open, node q is inactive and $q \in \mathscr{I}$:

$$C(l_q^k, \mathbf{u}^{hk}) + \Delta C(l_q^k, \mathbf{u}^{hk}) = 0 \tag{5.89}$$
$$\Rightarrow$$
$$l_q^{k+1} = 0 \tag{5.90}$$
$$\Rightarrow$$
$$\Delta l_{\mathscr{I}}^k = -\,l_{\mathscr{I}}^k. \tag{5.91}$$

Using the constraint on the tangential part of the multiplier, we have:

$$\Delta\boldsymbol{l}_{\mathscr{I}}^k = -\boldsymbol{l}_{\mathscr{I}}^k \tag{5.92}$$

To summarize, we have:

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{G}_{\mathscr{I}} & \boldsymbol{G}_{\mathscr{A}} \\ \mathbf{0} & \boldsymbol{I}_{\mathscr{I}} & \mathbf{0} \\ \boldsymbol{N}_{\mathscr{A}}^{\mathrm{T}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{\mathscr{A}} \end{bmatrix} \begin{bmatrix} \Delta\boldsymbol{u}^k \\ \Delta\boldsymbol{l}_{\mathscr{I}}^k \\ \Delta\boldsymbol{l}_{\mathscr{A}}^k \end{bmatrix} = - \begin{bmatrix} \boldsymbol{K} & \boldsymbol{G}_{\mathscr{I}} & \boldsymbol{G}_{\mathscr{A}} \\ \mathbf{0} & \boldsymbol{I}_{\mathscr{I}} & \mathbf{0} \\ \boldsymbol{N}_{\mathscr{A}}^{\mathrm{T}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{\mathscr{A}} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^k \\ \boldsymbol{l}_{\mathscr{I}}^k \\ \boldsymbol{l}_{\mathscr{A}}^k \end{bmatrix} + \begin{bmatrix} \boldsymbol{f} \\ \mathbf{0} \\ \boldsymbol{g}_{0,\mathscr{A}} \\ \mathbf{0} \end{bmatrix}. \tag{5.93}$$

Using (5.81) in (5.93), we deduce that:

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{G}_{\mathscr{I}} & \boldsymbol{G}_{\mathscr{A}} \\ \mathbf{0} & \boldsymbol{I}_{\mathscr{I}} & \mathbf{0} \\ \boldsymbol{N}_{\mathscr{A}}^{\mathrm{T}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{T}_{\mathscr{A}} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^{k+1} \\ \boldsymbol{l}_{\mathscr{I}}^{k+1} \\ \boldsymbol{l}_{\mathscr{A}}^{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \mathbf{0} \\ \boldsymbol{g}_{0,\mathscr{A}} \\ \mathbf{0} \end{bmatrix}. \tag{5.94}$$

Finally, using the fact that the tangential part of the active Lagrange multipliers is zero, we can write the previous system as follows:

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{N}_{\mathscr{I}} & \boldsymbol{N}_{\mathscr{A}} \\ \mathbf{0} & \boldsymbol{I}_{\mathscr{I}} & \mathbf{0} \\ \boldsymbol{N}_{\mathscr{A}}^{\mathrm{T}} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^{k+1} \\ \boldsymbol{l}_{\mathscr{I}}^{k+1} \\ \boldsymbol{l}_{\mathscr{A}}^{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \mathbf{0} \\ \boldsymbol{g}_{0,\mathscr{A}} \end{bmatrix}. \tag{5.95}$$

Therefore, we can derive the following Algorithm 10 by choosing an initial guess for the active set $\mathscr{A}_0$, construct the system (5.95), solve it, update the active set and continue until convergence occurs. This algorithm is known in the literature as the *active set strategy*. In the algorithm, we used the fact that:

$$\begin{aligned} g_q^{k+1} &= b(\mathbf{u}^{h(k+1)}, \varphi_q) - g(\varphi_q) \qquad (5.96)\\ &= \mathbf{e}_q^{\mathrm{T}} \left(\boldsymbol{N}^{\mathrm{T}} \boldsymbol{u}^{k+1} - \boldsymbol{g}_0 \right), \qquad (5.97) \end{aligned}$$

to write:

$$l_q^{k+1} + c\, g_q^{k+1} = l_q^{k+1} + c\, \mathbf{e}_q^{\mathrm{T}} \left(\boldsymbol{N}^{\mathrm{T}} \boldsymbol{u}^{k+1} - \boldsymbol{g}_0 \right). \tag{5.98}$$

Algorithm 10: Active set strategy

```
1 $k \leftarrow 0$
2 Choose an initial guess for the active set $\mathscr{A}_k$
3 do
4     Given $\mathscr{A}_k$, compute the solution of
          $$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{N}_{\mathscr{I}} & \boldsymbol{N}_{\mathscr{A}} \\ \mathbf{0} & \boldsymbol{I}_{\mathscr{I}} & \mathbf{0} \\ \boldsymbol{N}_{\mathscr{A}}^{\mathrm{T}} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u}^{k+1} \\ \boldsymbol{l}_{\mathscr{I}}^{k+1} \\ \boldsymbol{l}_{\mathscr{A}}^{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \mathbf{0} \\ \boldsymbol{g}_{0,\mathscr{A}} \end{bmatrix}. \tag{5.99}$$
5     $\mathscr{A}_{k+1} \leftarrow \left\{ q \in P_{\mathrm{c}}^h : l_q^{k+1} + c\, \mathbf{e}_q^{\mathrm{T}} \left( \boldsymbol{N}^{\mathrm{T}} \boldsymbol{u}^{k+1} - \boldsymbol{g}_0 \right) > 0 \right\}$
6     $k \leftarrow k + 1$
7 while $\mathscr{A}_k \neq \mathscr{A}_{k-1}$
```

5.1.6 Numerical solution strategy

In this section, we present the numerical solution strategy used to solve the linear system (5.95).

The linear system is solved using the right-preconditioned GMRES method discussed in section 3.1.1. As already discussed, using a preconditioner can reduce the number of iterations required to converge.

In this thesis, we consider multigrid preconditioners as discussed in more details in Appendix C. Multigrid methods are motivated by the observation that relaxation-based methods typically converge slowly for error modes with low-frequency. Using a relaxation-based method directly as the preconditioner would damp high-frequency error modes but not the low-frequency ones. The idea of the multigrid methods is to use a so-called *smoother*, a method that damps high-frequency modes, such as relaxation-based methods, on systems of different sizes. The multigrid method relies on two phases: the multigrid setup where a hierarchy of increasingly smaller linear systems called levels is computed and the multigrid apply phase where the method visits the levels and applies the smoothers on the linear system on each level.

This way, when applying the preconditioner, even low-frequency error modes can be damped as a low-frequency mode on a higher level can be seen as a higher-frequency mode on a lower level.

In this thesis we restrict ourselves to the use of the algebraic multigrid methods (AMG) which construct the multigrid hierarchy from the graph of the operator of the finest level. We restricted ourselves to those preconditioners as they were already implemented in the MueLu package which supports embedded ensemble propagation. When evaluating the preconditioner for an ensemble, the results are mathematically equivalent to applying the preconditioner to each sample individually.

Preconditioner

In this work, we use the full aggregation-based algebraic multigrid strategy for structural contact problems in saddle-point formulation using a Mortar approach proposed by Wiesner [2015] as a right preconditioner which can be seen as a Monolithic AMG preconditioner, in particular as a AMG(SIMPLE), as discussed in [Verdugo and Wall, 2016].

This is a multigrid strategy as discussed in Appendix C that maintains the block structure of the block operator by aggregating both displacement and Lagrange multipliers as illustrated in Fig. 5.9.

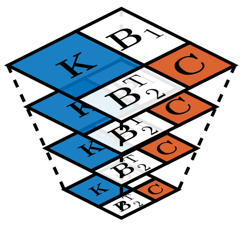


Figure 5.9: Illustration of the multigrid hierarchy of the Monolithic AMG preconditioner for contact problem. The finest level is represented on top of the coarsest levels.

On intermediate multigrid levels, a SIMPLE relaxation [Benzi et al., 2005; Li and Vuik, 2004] is used as blocked smoother.

The SIMPLE smoother for contact problem

$$\boldsymbol{A} := \begin{bmatrix} \boldsymbol{K} & \boldsymbol{B}_1 \\ \boldsymbol{B}_2^{\mathrm{T}} & \boldsymbol{C} \end{bmatrix}, \tag{5.100}$$

where, $\boldsymbol{B}_1 = \begin{bmatrix} \boldsymbol{N}_{\mathscr{I}} & \boldsymbol{N}_{\mathscr{A}} \end{bmatrix}$, $\boldsymbol{B}_2 = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{N}_{\mathscr{A}} \end{bmatrix}$, and $\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I}_{\mathscr{I}} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}$, has block preconditioning matrix $\boldsymbol{M}$:

$$\boldsymbol{M} := \begin{bmatrix} \boldsymbol{K} & \boldsymbol{0} \\ \boldsymbol{B}_2^{\mathrm{T}} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{D}^{-1}\boldsymbol{B}_1 \\ \boldsymbol{0} & \frac{1}{\beta}\boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}\boldsymbol{D}^{-1}\boldsymbol{B}_1 \\ \boldsymbol{B}_2^{\mathrm{T}} & \frac{1}{\alpha\beta}\boldsymbol{C} + (1 - \frac{1}{\alpha\beta})\boldsymbol{B}_2^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{B}_1 \end{bmatrix}, \tag{5.101}$$

where $\boldsymbol{D}$ is an approximation of $\boldsymbol{K}$, $\boldsymbol{R} := \frac{1}{\alpha}\boldsymbol{C} - \frac{1}{\alpha}\boldsymbol{B}_2^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{B}_1$ is an approximation of the Schur complement, and α and β are damping factors chosen in the interval $]0,1]$ discussed in [Elman et al., 2008]. In this example, we use the matrix diagonal of $\boldsymbol{K}$ as the approximation $\boldsymbol{D}$.

In the particular case of the mesh-tying problem, we have:

$$\boldsymbol{A} := \begin{bmatrix} \boldsymbol{K} & \boldsymbol{G} \\ \boldsymbol{G}^{\mathrm{T}} & \boldsymbol{0} \end{bmatrix}, \tag{5.102}$$

has block preconditioning matrix $\boldsymbol{M}$:

$$\boldsymbol{M} := \begin{bmatrix} \boldsymbol{K} & \boldsymbol{0} \\ \boldsymbol{G}^{\mathrm{T}} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} \boldsymbol{I} & \boldsymbol{D}^{-1}\boldsymbol{G} \\ \boldsymbol{0} & \frac{1}{\beta}\boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}\boldsymbol{D}^{-1}\boldsymbol{G} \\ \boldsymbol{G}^{\mathrm{T}} & (1 - \frac{1}{\alpha\beta})\boldsymbol{G}^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{G} \end{bmatrix}, \tag{5.103}$$

For both cases, contact problems and mesh-tying problems, there exist cases where the saddle-point problem is well posed but the matrix $\boldsymbol{K}$ is singular. This happens for instance in the example discussed in section 7.3. This may have an impact on the results; the results may be different if a preconditioner had been chosen that is better able to deal with the singularity of the main matrix block.

The parameters used for these multigrid preconditioners are discussed in the results section.

5.1.7 Ensemble divergence

When applying ensemble propagation on the active set strategy of Algorithm 10, two new potential occurrences of ensemble divergences are introduced:

- A new occurrence of loop divergence: different samples of an ensemble may require a different number of active set iterations to converge,
- A new if-then-else divergence: the set of active Lagrange multipliers $\mathscr{A}_{k+1}$ is now sample dependent, this impacts the sparsity pattern of the matrix of the system.

Whereas the new loop divergence can be solved using a previously used strategy: to wait for the last sample to converge; two subproblems arise from that strategy as, we now have two nested iterative processes the active set strategy and the GMRES method:

- Let's assume that the active set strategy has converged for a first sample ℓ_1 but has not converged for a second sample ℓ_2. As we are waiting for the last sample to converge and as ℓ_2 has not converged, the active set strategy requires at least one more iteration. Therefore, we have to update the activity and solve the updated linear system for all the samples. However, the active set strategy has already converged for ℓ_1 and, therefore, we already have a converged solution of the updated linear system for ℓ_1 as this linear system is the same as the one of the previous active set iteration. As a consequence, the initial guess for GMRES used to solve the updated linear system for ℓ_1 has already a small absolute norm. However, as we are using relative residual norms, even for ℓ_1, the initial relative norm is 1. During this current GMRES solver, the relative residual norm associated to ℓ_1 will continue to decrease. We must avoid the case where the sample ℓ_1 is the limiting sample for the new GMRES solver, i.e. we need to enforce that ℓ_1 is not the slowest sample to converge in the new GMRES solver in the sense of the relative norm,

- If we make the same assumption, i.e. if we assume that sample ℓ_1 has converged at the previous active set iteration but not ℓ_2, as discussed in the previous subproblem, the sample ℓ_1 continues to be updated in a following GMRES solver. The norm of the residual of sample ℓ_1 continues to decrease. However, this norm can be reduced up to a point where the implicit and explicit residual are no more coherent leading to loss of accuracy described just below.

  As already said, the implicit norm is equal to the explicit norm in infinite precision. However, in practice the condition number of the matrix of the linear system influences the difference between the two norms. What can happen is that GMRES may converge in the sense of the implicit norm but not in the sense of the explicit norm (this depends both on the tolerance and the condition number), this issue is named as *loss of accuracy*.

The first problem is solved updating the tolerance of GMRES for the converged samples and setting it to 1. This enforces the fact the initial guess of each converged sample has already converged, in the sense of the relative norm of the residual of the new GMRES solver, and the associated sample is not the limiting sample. The second problem cannot be avoided but does not impact the precision of the results nor the number of iterations to converge.

The new if-then-else divergence, the sample dependence of the activity, is more challenging than the above-mentioned loop divergence as this if-then-else divergence can potentially impact the sparsity pattern and the data layout. This if-then-else divergence can be treated using a strategy used in the context of contact mechanics with the assumptions of small deformations and small displacements without ensemble propagation. We first explain this standard strategy without ensemble propagation and, then, explain how it can be used to resolve the if-then-else divergence.

Although the activity of Lagrange multipliers cannot be known a priori, the small deformations and small displacements assumptions allow us to compute once for all the association $(\mathbf{x}, \boldsymbol{\gamma}(\mathbf{x}))$ defined in (5.6). The independence of this association with respect to the activity of the Lagrange multipliers allows us to precompute a sparsity pattern which includes the sparsity patterns of all possible contact configurations. The sparsity pattern of the blocked operator of (5.95) is always included in the sparsity pattern of the block matrix:

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{N} \\ \boldsymbol{N}^{\mathrm{T}} & \boldsymbol{I} \end{bmatrix}. \tag{5.104}$$

In other words, it is possible to compute, based on the mesh and the definition of $\Gamma_{\mathrm{c}}^{\mathrm{s}}$ and $\Gamma_{\mathrm{c}}^{\mathrm{m}}$, a sparsity pattern which will include all sparsity patterns of any active set. In order not to have too large Mortar matrices, $\Gamma_{\mathrm{c}}^{\mathrm{s}}$ and $\Gamma_{\mathrm{c}}^{\mathrm{m}}$ should be close to the actual contact interface.

In the case of contact problems with ensemble propagation, assuming the sample independence of the mesh, we have that the association $(\mathbf{x}, \boldsymbol{\gamma}(\mathbf{x}))$ is sample independent too. Therefore, we can, once again, compute the smallest union of all feasible sparsity patterns that might be encountered during the active set strategy with ensemble propagation as illustrated in Fig. 5.10.

This approach suppresses ensemble divergence but it has the drawback of increasing the amount of memory used as zero values are stored for inactive contact.

In practice, we use masked assignments as introduced in section 4.3 to implement this strategy in section 6.4.3.

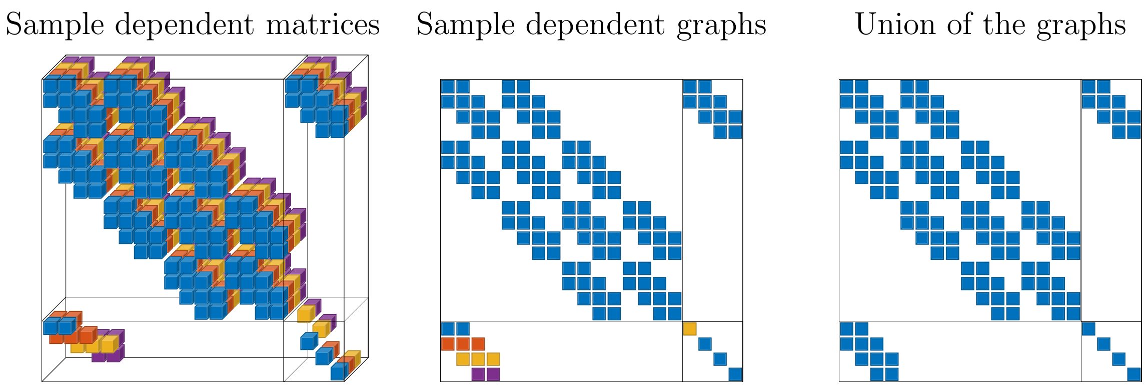


Figure 5.10: Local contact status changes the graph of the Mortar matrices of the contact formulation. Samples of the same ensemble can have different graphs. This ensemble divergence is managed by using the union of all feasible graphs.

5.2 Thermomechanical problems

The second class of problems considered in this work is the thermomechanical problems.

5.2.1 Strong form

First, as done in the previous section, we denote by Ω the d-dimensional open bounded domain of $\mathbb{R}^d$ occupied by the deformable body in the undeformed configuration.

In this study, we assume small deformations, small displacements, and a linear thermoelastic behavior. We can write the stress tensor as follows:

$$\boldsymbol{\sigma} = \mathbf{C}\boldsymbol{\varepsilon}_{\mathbf{x}}\,\mathbf{u} - \mathbf{D}\left(T - T_{ref}\right) \quad \text{in } \Omega, \tag{5.105}$$

where $\mathbf{D}$ is a second-order tensor called stress-temperature modulus, T is the temperature, and T_{ref} is the reference temperature for thermal expansion.

We can write the following balance laws:

$$\begin{aligned} &\nabla \cdot (\mathbf{C}\boldsymbol{\varepsilon}_{\mathbf{x}}\,\mathbf{u} - \mathbf{D}\,(T - T_{ref})) + \mathbf{f} = \mathbf{0} \quad \text{in } \Omega, && (5.106)\\ &k\,\Delta T + r = 0 \quad \text{in } \Omega, && (5.107) \end{aligned}$$

where k is the isotropic thermal conductivity and r is the internal heat generation per unit volume.

The boundary of Ω is denoted by Γ and is decomposed into four subsets $\Gamma_{\mathbf{u}}$, $\Gamma_{\boldsymbol{\sigma}}$, $\Gamma_{\mathbf{q}}$, and Γ_T such that:

$$\begin{aligned} \Gamma &= \bar{\Gamma}_{\mathbf{u}} \cup \bar{\Gamma}_{\boldsymbol{\sigma}}, && (5.108)\\ \Gamma &= \bar{\Gamma}_{\mathbf{q}} \cup \bar{\Gamma}_T, && (5.109) \end{aligned}$$

where $\Gamma_{\mathbf{u}}$, $\Gamma_{\boldsymbol{\sigma}}$, $\Gamma_{\mathbf{q}}$, and Γ_T stand respectively for the part of the boundary where Dirichlet boundary conditions are applied on the displacement fields, the part of the boundary where surface tractions are applied, the part of the boundary where Dirichlet boundary conditions are applied on the temperature field, and the part of the boundary where heat fluxes are applied.

Neumann and Dirichlet boundary conditions are represented as follows:

$$\begin{aligned} &\boldsymbol{\sigma}(\mathbf{n}) = \mathbf{t} \quad \text{on } \Gamma_{\boldsymbol{\sigma}}, && \mathbf{u} = \mathbf{0} \quad \text{on } \Gamma_{\mathbf{u}}, && (5.110)\\ &\mathbf{q}\cdot\mathbf{n} = h \quad \text{on } \Gamma_{\mathbf{q}}, && T = 0 \quad \text{on } \Gamma_T, && (5.111) \end{aligned}$$

where $\mathbf{t}$ represents the surface load per unit surface and h the heat flux imposed.

5.2.2 Discretization

After discretization using the finite element method we deduce the non-symmetric system:

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{S} \\ \mathbf{0} & \boldsymbol{L} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{t} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{l} \end{bmatrix}, \qquad (5.112)$$

where $\boldsymbol{K}$ is the stiffness matrix, $\boldsymbol{L}$ the matrix linked to the discretization of the heat problem, $\boldsymbol{S}$ the coupling matrix linked to the thermal expansion, $\boldsymbol{f}$ the vector of external forces, $\boldsymbol{l}$ the vector of external heat generation, $\boldsymbol{u}$ the mechanical degree of freedom, and $\boldsymbol{t}$ the value of the temperature minus the reference temperature T_{ref}. The shape functions used to discretize the temperature field are the standard shape functions used for the discretization of the displacement fields discussed earlier in this chapter.

The matrices $\boldsymbol{K}$, $\boldsymbol{L}$, and $\boldsymbol{S}$ and the vectors $\boldsymbol{f}$ and $\boldsymbol{l}$ are defined such that:

$$\begin{aligned} \boldsymbol{w}^{\mathrm{T}}\,\boldsymbol{K}\,\boldsymbol{u} &= \int_\Omega \mathbf{C}(\boldsymbol{\varepsilon}_{\mathbf{x}}\,\mathbf{w}^h) : \boldsymbol{\varepsilon}_{\mathbf{x}}\mathbf{u}^h\,dV, && (5.113)\\ \boldsymbol{w}^{\mathrm{T}}\,\boldsymbol{S}\,\boldsymbol{t} &= -\int_\Omega \mathbf{D}T^h : \boldsymbol{\varepsilon}_{\mathbf{x}}\,\mathbf{w}^h\,dV, && (5.114)\\ \boldsymbol{v}^{\mathrm{T}}\,\boldsymbol{L}\,\boldsymbol{t} &= \int_\Omega k\,\nabla\,v^h \cdot \nabla\,T^h\,dV, && (5.115)\\ \boldsymbol{w}^{\mathrm{T}}\,\boldsymbol{f} &= -\int_\Omega \mathbf{D}T_{ref} : \boldsymbol{\varepsilon}_{\mathbf{x}}\,\mathbf{w}^h\,dV + \int_\Omega \mathbf{f}\cdot\mathbf{w}^h\,dV + \int_{\Gamma_{\boldsymbol{\sigma}}} \mathbf{t}\cdot\mathbf{w}^h\,dS, && (5.116)\\ \boldsymbol{v}^{\mathrm{T}}\,\boldsymbol{l} &= \int_\Omega r\,\mathbf{v}^h\,dV - \int_{\Gamma_{\mathbf{q}}} h\,\mathbf{v}^h\,dS. && (5.117) \end{aligned}$$

5.2.3 Numerical solution strategy

As in the case of the contact problems, we use the right-preconditioned GMRES method discussed in section 3.1.1.

Once again, we use a multigrid precondtioner as discussed in section 5.1.6, in Appendix C, and below.

In the case of thermomechanical problems, we consider, in addition to the GMRES stopping criterion based on the norm of the residual, stopping criteria based on the norm of the residual of each block to enforce the convergence of each physics individually.

Preconditioner

For thermomechanical problems, we use a Monolithic AMG preconditioner which accounts for the coupling between temperature and displacement field at all multigrid levels as discussed in [Verdugo and Wall, 2016] and illustrated in Fig. 5.11. All levels of the multigrid hierarchy have both the temperature and displacement fields. As proposed and defined in [Verdugo and Wall, 2016], we use backward block Gauss-Seidel iterations as a block level smoother.

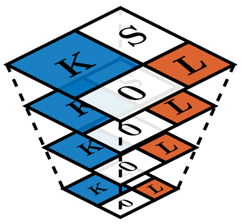


Figure 5.11: Illustration of the multigrid hierarchy of the Monolithic AMG preconditioner for thermomechanical problem. The finest level is represented on top of the coarsest levels.

5.3 Conclusions

In this chapter, we have formally described the two classes of problems used as examples for ensemble GMRES in the remainder of this thesis.

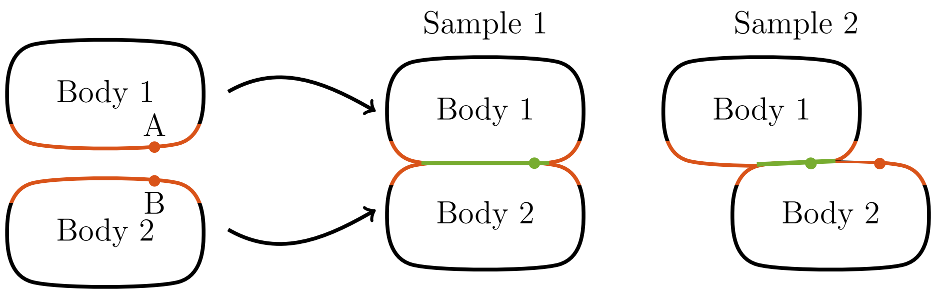


Figure 5.12: Linear elasticity contact problems with large displacements: the bodies are modelled as elastic bodies with small strain theory and their contact can be either locally **closed** or **open**. Due to the large sliding, the point A of Body 1 can be in contact with point B (Sample 1) or with any other point of the surface of Body 2 (Sample 2) .

Moreover, we have discussed how to solve ensemble divergence arising in the contact problems using the union of the graph of potential contact configuration.

As future work, it would be interesting to consider contact problems with large sliding as it raises new type of ensemble divergence as the potential contact interfaces are a priori unknown and can change from one sample to another as illustrated in Fig. 5.12.

Chapter 6

Implementation details

In this chapter, we describe *Katoptron*[1]: the software developed and used during this thesis. Katoptron is a FE solver implemented in the Waves framework using the Trilinos library with the aim of computing thermomechanical deformations of mirrors.

[1]`https://gitlab.uliege.be/am-dept/waves`

6.1 Waves

Waves is an open-source code which has been developed by Romain Boman since 2014 with the purpose of being a toolbox for PhD students to develop PDEs solvers in the department of Aerospace and Mechanical Engineering at University of Liège.

The architecture of Waves is based on two sets of libraries:

- a set of common necessary libraries which give common features such as gmsh mesh [Geuzaine and Remacle, 2009] input supports, VTK [Schroeder et al., 2004] output writers, time monitor, Gauss-point definitions;
- a set of finite element solvers such as the "waves" solver used to solve the wave equation on imported meshes. This solver was the first one developed and gave its name to the software.

Waves is mainly written in C++ and wrapped in Python with SWIG [Beazley, 2003]. Waves can be compiled using CMake [Martin and Hoffman, 2010] and supports CTest.

The models, such as finite element discretization of the wave problem, are defined in Python where the mesh and an arbitrary number of boundary conditions and material properties are set. Afterwards, the C++ compiled functions take those data as inputs, construct the matrices and vectors of the linear systems to solve, solve them, and extract results with different output types such as VTK, gmsh, or text file. The post processing is then pursued in Python where the user can extract values of the solution along lines, cuts, at points, or even use in situ visualization.

6.2 Trilinos

Trilinos [Heroux et al., 2005] is a software component library developed by Sandia National Laboratories with the aim of breaking down PDEs solvers into common building blocks, optimizing those blocks, and constructing optimized solvers based on those blocks.

6.3 Katoptron

In order to test ensemble propagation, we have implemented a finite element code based on the Tpetra solver stack from Trilinos in the Waves framework. The Tpetra solver stack is discussed in more details in the remainder of this chapter.

The simulation can be decomposed into five main steps:

1. preprocessing
2. assembling the linear system to solve,
3. solution of the linear system,
4. active set update and potentially the requirement to solve a new linear system,
5. postprocessing.

Steps 1 and 5 follow the standard of Waves and do not require more explanation. The step 3 relies on the Belos GMRES solver [Bavier et al., 2012], with the modifications discussed in Chapter 4, and MueLu multigrid preconditioners [Prokopenko et al., 2014] as discussed in Appendix C and, therefore, is fully based on Trilinos. Both the steps 2 and 4 require use of Trilinos data types, in particular Tpetra sparse matrices and vectors, to be able to use the above-mentioned solvers. These two steps have required more implementation work and are explained in detail in the next section.

The code is fully templated on the data type to use embedded ensemble propagation and relies on Tpetra and Kokkos for the MPI+X parallelization where X is OpenMP or CUDA.

The implementation work done during this thesis has been included into two codes: Waves and Trilinos. The part of the code included in Trilinos has been tested with CUDA. The part of the work included in Waves, i.e. the implementation of Katoptron, has not been developed nor tested with CUDA supports. This is due to the fact that some Trilinos packages used in Katoptron, such as Moertel and PyTrilinos, do not support Kokkos nor CUDA yet.

Katoptron can solve problems with the following properties:

1. mechanical or thermomechanical problems,
2. linear elasticity and no dependence of the material parameters on the temperature,
3. with or without mesh-tying and contact interface,
4. meshed with 3D elements with either hexahedrons or tetrahedrons,
5. with Dirichlet and Neumann boundary conditions.

6.4 Hybrid parallelism

The hybrid parallelization of GMRES, the multigrid preconditioner, and the level smoothers are inherited from the Tpetra solver stack which relies on the Petra Object Model (POM) [Boman et al., 2004; Heroux et al., 2005; Baker and Heroux, 2012] for the distributed parallelism and the KokkosKernels package for the node-level parallelism [Deveci et al., 2016b].

The hybrid parallelism that had to be implemented in Katoptron were located into steps 2 and 4 discussed in the previous section.

In order to describe precisely the approach used, it is necessary to introduce the POM.

6.4.1 Petra Object Model

Distributed sparse linear algebra in Trilinos is based on the Petra programming model. The word Petra stands for *foundation* in Greek.

There are currently two maintained implementations of Petra; Epetra and Tpetra which stand for Essential Petra and Templated Petra respectively.

Both implementations rely on the Petra Object Model, an abstract model which describes matrices and vectors distributed over MPI processes.

The POM defines the following abstractions:

- `Map` defines how the global elements of distributed objects, such as entries of a distributed vector or rows of a distributed matrix, are distributed across a system and the mapping between local elements of a given compute node to global elements,
- `Import` and `Export` define the communication patterns between two maps, they can be precomputed once and be reused several times; given two maps, we can compute once the `Import` from one to another one and use this communication pattern to transform several vectors defined using one map to vectors defined using the second map,
- `Vector` and `MultiVector` define dense vectors and dense blocked vectors, i.e. a collection of vectors stored together which is typically used in blocked Krylov methods, distributed across a system following a given map,
- `Operator` is an abstraction for linear operators from one `Vector`/`MultiVector` to another.

6.4.2 Matrix assembly

We will restrict ourselves to the discussion of the matrix assembly of the primal variables as the computation of the Mortar matrix $\boldsymbol{G}$ has been done using the Moertel package [Heroux et al., 2005] which supplies capabilities for nonconforming mesh-tying and contact formulations using Mortar methods. In the current version of the code, the computation of the Mortar matrix does not leverage Kokkos directly and thus is single threaded.

Mesh partitioning

First we want to partition the mesh and associate a partition to each MPI process. Two properties are required for the partitions:

- The amount of computational work on each partition has to be comparable otherwise some process will wait for others and the HPC machine will not be optimally used. This means that, if the mesh has elements of the same type, the number of elements of each partition has to be roughly the same. However, if the mesh has tetrahedra and hexaedra, for example, the partition will not have the same number of elements as tetrahedra require less wall-clock time than hexahedrons.
- The number of connections between partitions has to be as small as possible such that the cost of sending and receiving information between MPI processes is as small as possible.

The mesh partition has been made using METIS [Karypis, 2011] which uses multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes [Karypis and Kumar, 1998a; Schloegel et al., 1998; Karypis and Kumar, 1998b], methods developed in the Department of Computer Science & Engineering at the University of Minnesota. METIS can be directly used in gmsh. Another possibility would have been to use Chaco, a tool developed at Sandia Labs [2] or Zoltan a package of Trilinos. In this work we only used the multilevel k-way directly through gmsh.

The multilevel k-way is based on three phases:

[2] `http://www.cs.sandia.gov/CRF/chac_p2.html`

- Coarsening phase: The size of the graph is iteratively decreased.
- Initial partition phase: The smallest graph is partitioned in k partitions.
- Uncoarsening phase: The partitioning of the coarsest mesh is projected on the next finer mesh and the partitions are refined, the projected partitioning may not be optimal.

The idea is to coarsen the graph iteratively and to partition the graph on the coarsest level into k parts such that each one has a comparable amount of work and such as the interface work is minimized. After that, the graph is uncoarsed iteratively and the partition is prolongated from a coarse level to a finer level [Karypis and Kumar, 1998b], after the prolongation the solution is reoptimized to minimized the interface work as illustrated moving from the coarsest level to one finer level in Fig. 6.1.

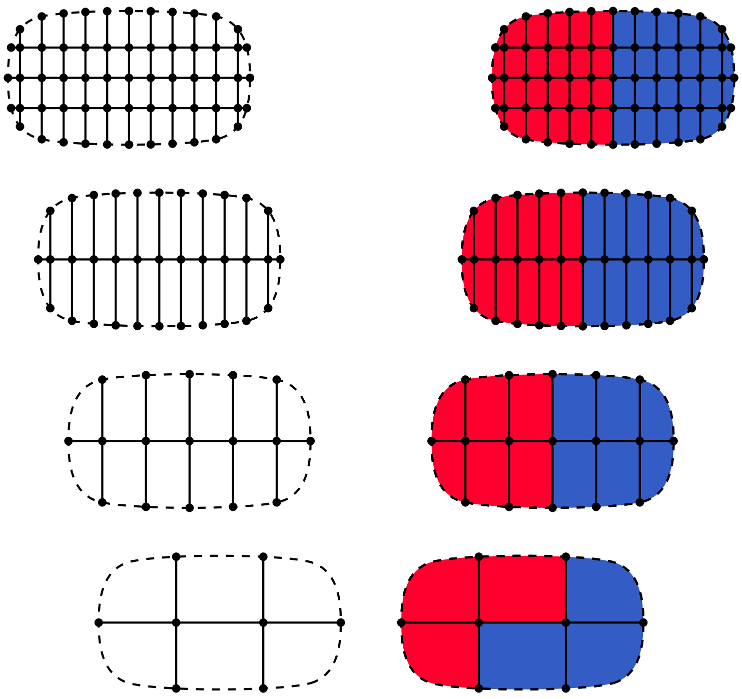


Figure 6.1: Representation of the multilevel 2-way partitioning algorithm

Distributed-memory parallelism

There are two approaches to compute the sparse matrix based on the partitioned mesh illustrated in Fig. 6.2: using MPI communication to sum the contributions of different MPI processes at their interface or using ghost elements which are, for a given partition, all the elements included in another partition such that they share at least one node with the given partition.

We first describe the approach without ghost elements. We associate one partition to each MPI process and associate one MPI process to each element as illustrated in Fig. 6.3; every element is included in only one mesh partition and, therefore, associated to only one MPI process. Due to the fact that the interface between the partition is shared by at least two MPI processes, we cannot directly associate one MPI process to each node and, therefore, to each degree of freedom. Therefore, we construct a `Map` of nodes with overlaps due to the nodes on the interfaces of the partition which are associated to at least two processors as illustrated in Fig. 6.4. A second `Map` of nodes is built based on the previous `Map` of nodes with overlaps such that the new one is 1-to-1 using the function createOneToOne. The new `Map` without overlaps is illustrated in Fig. 6.5.

Based on those two Maps of nodes and the number of degrees of freedom per node, we can build two last `Map` associated to the degrees of freedom with and without overlaps.

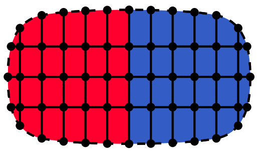


Figure 6.2: Representation of the partitioned mesh. The colors represent the mesh partitions.

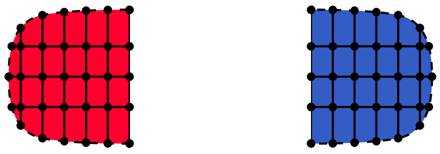


Figure 6.3: Representation of the distribution of the elements on the MPI processes. The elements of a mesh partition are associated to one MPI process.

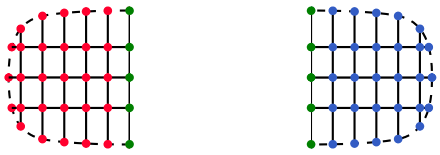


Figure 6.4: Representation of the distribution of the nodes on the MPI processes with overlaps. As the nodes on the interface of the mesh partitions are shared between at least two MPI processes, we create a `Map` of nodes which includes the overlaps. The nodes in green are the nodes of the overlap. Those nodes are stored in the memory associated to each MPI process.

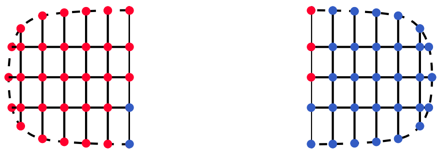


Figure 6.5: Representation of the distribution of the nodes on the MPI processes without overlaps. The nodes of the overlaps of Fig. 6.4 are distributed among the MPI processes in order to have a `Map` of nodes which is 1-to-1.

Each MPI process computes its assembly matrix associated to its mesh partition and stores it in a distributed sparse matrix with the `Map` of degrees of freedom with overlaps as the Row Map. The sum of the contributions of different MPI processes at the interface is then done using an `Export` to the `Map` of degrees of freedom without overlap and summing the entries with the same global IDs as illustrated in Fig. 6.6.

This first approach has the advantage that the element matrix of each element is only computed once. However, it requires MPI communication to sum the contribution of each MPI process to the rows associated to the nodes of the overlaps and it requires more memory to store the contribution of the MPI processes to the interface nodes which can be freed at the end of the assembly after the summation.

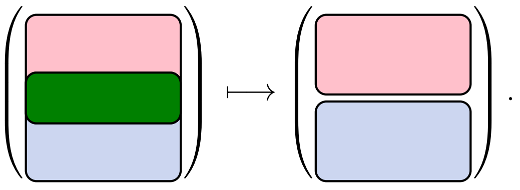


Figure 6.6: Representation of the summation of the contribution of the MPI processes to the row of the matrix associated to the degrees of freedom of the nodes included in the overlaps. The arrow $\longmapsto$ represents a POM export operation.

A second approach relies on the use of ghost elements as illustrated in Fig. 6.7. Knowing the information of the ghost elements such as their node positions and their constitutive parameters, it is possible for the MPI process associated to the given partition to evaluate the elemental matrix associated to the ghost elements and add their contributions to the rows associated with the degrees of freedom of the current MPI process. The two main advantages are that no MPI communication is required during the matrix assembly and that less memory is required but the wall-clock time of evaluating the element matrices increases due to the fact that the same computation is repeated by at least 2 MPI processes for the ghost elements.

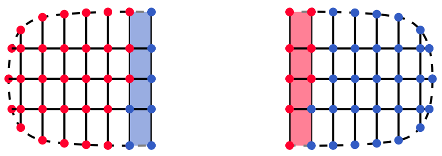


Figure 6.7: Representation of the ghost elements. The elements in blue are the ghost elements associated to the mesh partition in blue of Fig. 6.2 which are taken into account by the MPI process associated to the mesh partition in red to compute the entries of the matrix associated to its associated nodes illustrated in Fig. 6.5.

We have implemented and tested both approaches in the code. We did not observe a large impact of the chosen approach on the wall-clock time of the solver and decided to use the second approach to reduce the memory requirement.

Shared-memory parallelism

For a given MPI process, Kokkos is used to parallelize the assembly process of a given mesh partition using shared-memory parallelism.

A Kokkos parallel-for loop is used to loop over the finite elements of the partition, each thread computes the elemental matrix associated to the element, and sums its contribution into the local assembly matrix while avoiding data race between threads. This is done by forcing the use of atomic add when summing the contribution of the current element matrix to the assembly matrix associated to the mesh partition.

A fully standalone example that combines both parallelism and only relies on Trilinos is illustrated in `https://github.com/kliegeois/FEAssembly`.

6.4.3 Active set update

The primal variables and dual variables are distributed on different MPI processes. In order to update the activity of the constraints and to check the convergence, it is necessary to know whether at least one of the dual variables has to be updated on at least one of the MPI processes or not. The following strategy is applied:

- each MPI process updates locally its dual variables if needed and computes the number of updated dual variables as illustrated in Listing 6.1,
- the MPI processes sum the number of updated dual variables accross all processes using an MPI Allreduce,
- if the sum of the number of updated dual variables is zero, the MPI processes leave the non-linear system solution and write results on disk; otherwise, they start a new linear solve.

```cpp
// Number of updated Lagrange multipliers
int n_local_Lag = 0;

// Compute lambda+c (N^T u - g) and stores it in the array lambda_plus_c
scalar *lambda_plus_c;
//...

// loop over the local Lagrange multipliers
for (int i = 0; i < i_max; ++i)
{
    // Test the new activity of the Lagrange multipliers and store it into a mask
    auto new_activity = (lambda_plus_c[i] > 0.);
    // Test if the activity has changed for at least one sample
    if (!AND(new_activity == old_activity[i]))
    {
        ++n_local_Lag;
        // Extract the matrix entries of B on the row i
        Teuchos::ArrayView<const local_ordinal_type> local_indices;
        Teuchos::ArrayView<const scalar> values;
        scalar tmp;
        B->getLocalRowView(i, local_indices, values);

        // Loop over the entries of B_2^T of (5.100) to update the activity
        for (int j = 0; j < local_indices.size(); ++j)
        {
            MaskAssign(new_activity, tmp) = {values[j], 0.};
            B_2->replaceLocalValues(i, tuple(local_indices[j]), tuple(tmp));
        }
        // Update the activity of C (the block 11)
        MaskAssign(new_activity, tmp) = {0., 1.};
        C->replaceLocalValues(i, tuple(0), tuple(tmp));
        // Update the activity of g_2 (the right-hand side)
        MaskAssign(new_activity, tmp) = {g[i], 0.};
        g_2->replaceLocalValues(i, tmp);
        // Update the old activity
        old_activity[i] = new_activity;
    }
}
```

Listing 6.1: Active set update on one MPI process with mask and logical reduction.

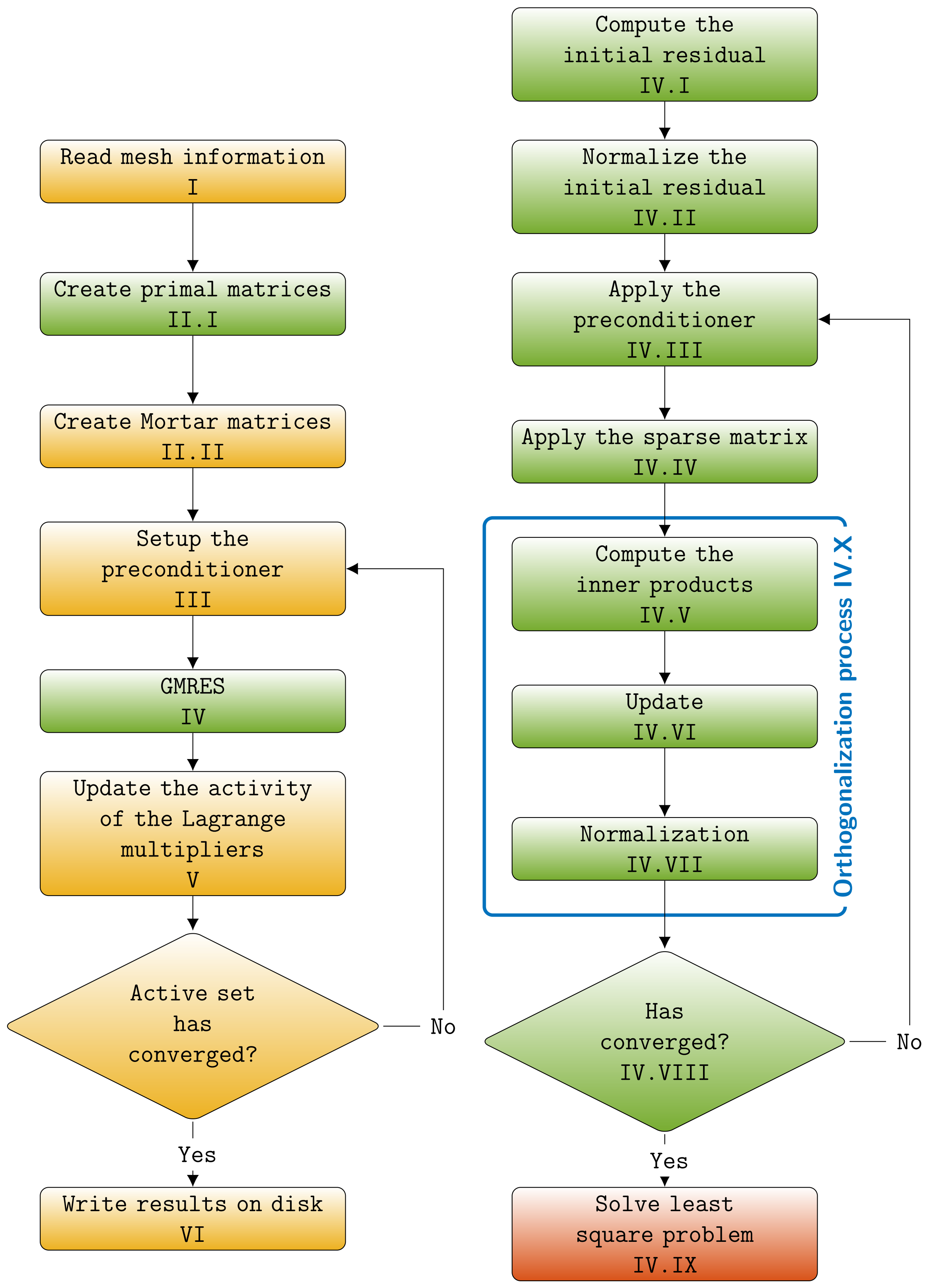
6.5 Overview of the algorithm and its implementation

In this section, we split the algorithm used into components, discuss how they are implemented, if they are parallelized, and give them tags to be referenced in results sections.

We first look at the full simulation algorithm illustrated using a flow chart in Fig. 6.8a. Each computational task with hybrid parallelism is represented in green, all tasks which are only parallelized using distributed memory are illustrated in orange, and all tasks which are not parallelized are illustrated in red. First, each MPI process reads its corresponding mesh partition during the task (I). After that, the required matrices are computed in (II): the primal matrices ($\boldsymbol{K}$, $\boldsymbol{S}$, and $\boldsymbol{L}$) are computed in (II.I), this task is described in detail in a previous section of this chapter, and the computation of the Mortar matrices in (II.II) which relies on Moertel which is not parallelized using shared memory. The computation of the matrices is followed by the setup of the multigrid preconditioner (III). During this step, the hierarchy of levels is built using an AMG strategy as discussed in Appendix C. This task is done using Muelu. The preconditioned system is then solved using GMRES in task (IV). We discuss it in more details later in this section. When the linear solver has converged, the activity of the constraints is updated (V) as discussed in the previous section and if the problem has at least one Lagrange multiplier for which the activity has changed, we have to solve a new preconditioned linear system. At the end, results are written on disk (VI) using VTK.

Zooming in on GMRES (IV), we have the flow chart of Fig. 6.8b which illustrates Algo. 2. The first step is the computation of the initial residual (IV.I) which corresponds to line 1 of Algo. 2. This residual is then normalized (IV.II) and is the first Arnoldi vector of the Krylov subspace as written in lines 2 and 3 of Algo. 2. After that, the iterative process starts by applying the preconditioner (IV.III) to the previous Arnoldi vector as done in line 5 of Algo. 2. The next step is to apply the sparse matrix to the resulting vector (IV.IV) as done in line 5 of Algo. 2. The orthogonalization process (IV.X) tries to orthonormalize the resulting vector with the previous Arnoldi vectors computing the inner products of the vector with the previous Arnoldi vectors (IV.V) and removing their contribution (IV.VI) before normalizing the vector (IV.VII) if its norm is not 0. We use a classical Gram-Schmidt's approach with a second orthogonalization process (DGKS) if needed to improve the stability as introduced in [Daniel et al., 1976]. The status tests are then evaluated to check if the iterative process should continue or not. When the iterative process has converged, the least square problem is solved. This step is not parallelized but is typically very small. This process has been implemented in this work as a partial template specialization of Belos classes [Bavier et al., 2012].

Finally, a third flow chart is provided in Fig. 6.9 to highlight the main CPU cost contribution of applying the multigrid preconditioner used in this work and discussed in Appendix C: the coarse smoother (IV.III.I), the pre-smoother (IV.III.II), the residual computation (IV.III.III), the restriction (IV.III.IV), the prolongation (IV.III.V), and the post-smoother (IV.III.VI). The multrigrid preconditioners used are implemented with MueLu [Prokopenko et al., 2014] and the smoothers are implemented in Ifpack2 [Prokopenko et al., 2016].



(a) Flow chart of the full simulation.   (b) Flow chart of GMRES.

Figure 6.8: Flow charts of the non-linear and linear solvers.

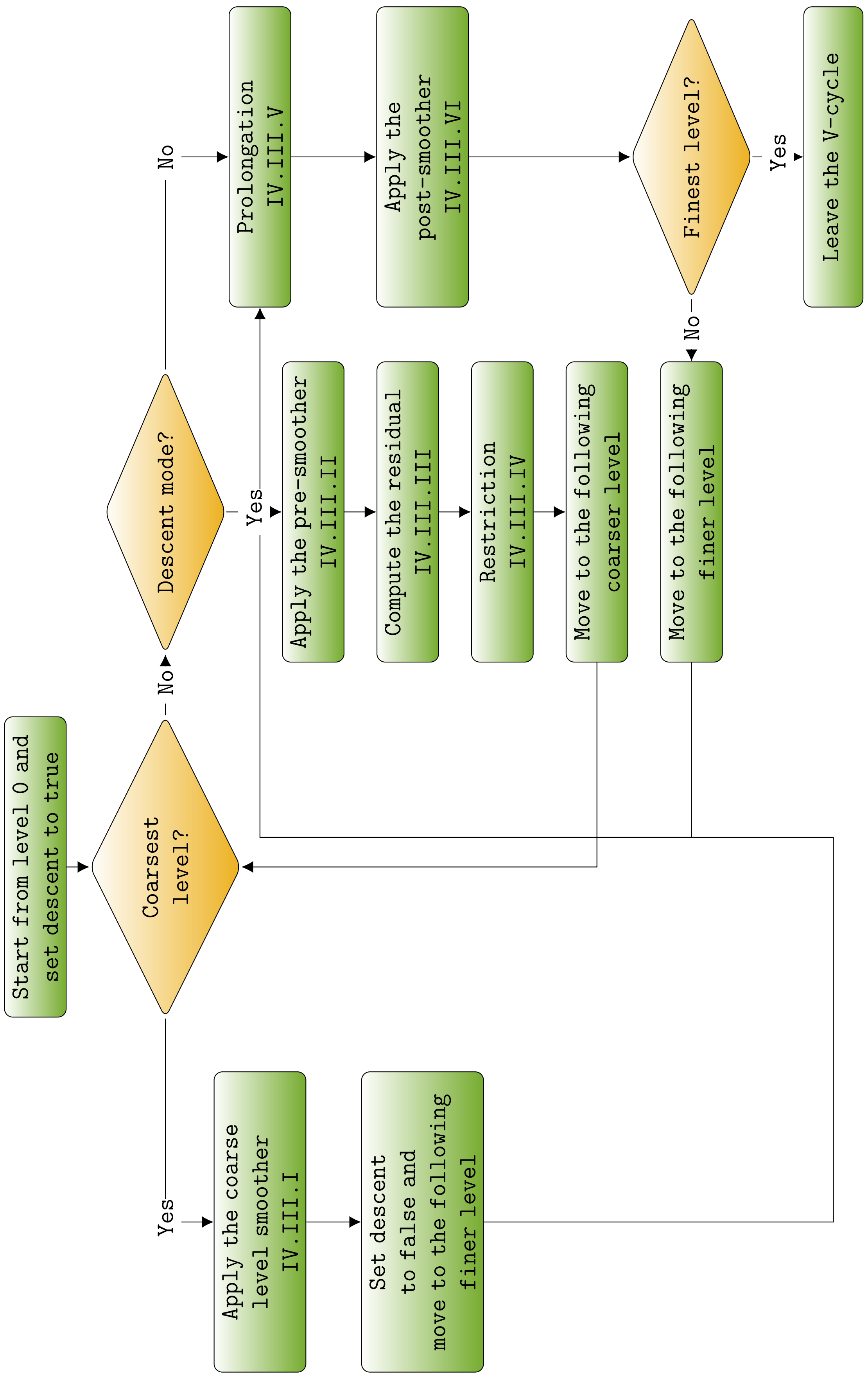

Start from level 0 and set descent to true
Coarsest level?
Yes
Apply the coarse level smoother IV.III.I
Set descent to false and move to the following finer level
No
Descent mode?
Yes
Apply the pre-smoother IV.III.II
Compute the residual IV.III.III
Restriction IV.III.IV
Move to the following coarser level
No
Prolongation IV.III.V
Apply the post-smoother IV.III.VI
Finest level?
No
Move to the following finer level
Yes
Leave the V-cycle

Figure 6.9: Flow chart of the preconditioner used, a mutigrid V-cycle.

6.6 Conclusions

In this chapter, Katoptron, a finite element solver which supports ensemble propagation, based on the Tpetra solver stack, and implemented in the Waves framework, has been described. This solver is used in the following chapters to generate the numerical results and the performance analysis.

Chapter 7

Numerical results on academic problems

In this chapter, we evaluate the performance of ensemble GMRES with and without ensemble reduction on four academic problems on the Blake system described in Appendix A.

In the first section, section 7.1, we illustrate with a tensile test on a cube how ensemble reduction can influence the convergence of ensemble GMRES as previously discussed in section 3.4.2. In particular, we discuss how the use of a preconditioner and the choice of the initial guess can influence ensemble divergence inside ensemble GMRES with reduction. This discussion is independent of the HPC system used.

The second section, section 7.2, is dedicated to the speed-up of the different components of GMRES. This section does not include ensemble divergence analysis and answers the question of which speed-up can be achieved on 1 CPU of Blake and how the problem size influences this speed-up.

In the third section, section 7.3, we evaluate the performance of ensemble GMRES on a mesh-tying problem. This time, both ensemble divergence and the speed-up play a role in the analysis.

The last section, section 7.4, is dedicated to the evaluation of ensemble GMRES on a contact problem, this is the first time in the whole literature that ensemble propagation is used to solve a non-linear parametric contact problem. The section includes some discussion on the grouping strategy which can be used to improve the speed-up.

Finally, the chapter ends with a conclusion which recalls the key aspects of this chapter.

7.1 Tensile test on a cube

In order to illustrate the impact of ensemble reduction on the convergence of ensemble GMRES as done in section 3.4.2, we look into a problem with a symmetric positive definite matrix: a linear elastic cube with the small displacement and small deformation approximation.

This problem leads to a disctretized linear system:

$$\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}. \tag{7.1}$$

where $\boldsymbol{K}$ is the stiffness matrix and $\boldsymbol{f}$ the external loads.

In this section, we consider a deformable cube with edge length $c = 10\,\text{mm}$ pulled along a fixed direction at a given side and clamped on the other side as illustrated in Fig. 7.1. The cube is disctretized with 4 elements in each direction. The Poisson ratio is 0.2 for all the tests of this section.

We consider 3 different cases:

- Varying Young's modulus and fixed surface load (test case 1),
- Fixed Young's modulus and varying surface load (test case 2),
- Varying Young's modulus and varying surface load (test case 3).

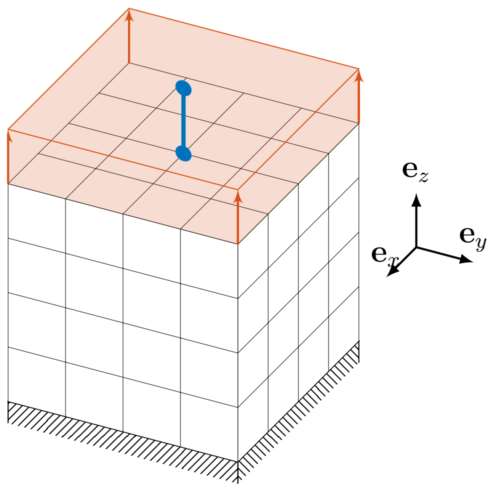


Figure 7.1: Mesh and geometry of the cube. The cube is pulled with a surface force in red. The quantity of interest is the displacement in blue.

These cases are deliberately simple for educational purposes. They are sufficiently simple to understand the influence of the reduced inner product even without the spectral theory used in section 3.4.2. This theory is then used to support the predicted behavior.

7.1.1 Varying Young's modulus

We consider 3 ensembles of 8 samples each. The first ensemble includes 8 samples for which the Young's modulus is linearly spaced from 0.75 MPa to 1.25 MPa. The second ensemble includes 8 samples for which the Young's modulus is linearly spaced from 0.95 MPa to 1.05 MPa. The third ensemble includes 8 samples for which the Young's modulus is linearly spaced from 1.2 MPa to 1.3 MPa. For the 3 ensembles we fix the surface force to 0.1 MPa.

When we consider the GMRES method to solve a linear system, we have that the convergence in the sense of the relative norm of the residual is not impacted by the multiplication of the left-hand side matrix by a scaling factor α. This is due to the fact that the Krylov subspace is independent of α. Therefore, in this example, every sample propagated alone shares the same convergence. We will observe that this is true for ensemble propagation without reduction too but not for ensemble reduction as the block diagonal matrix of (3.40) is now multiplied by different scaling factors for the different blocks.

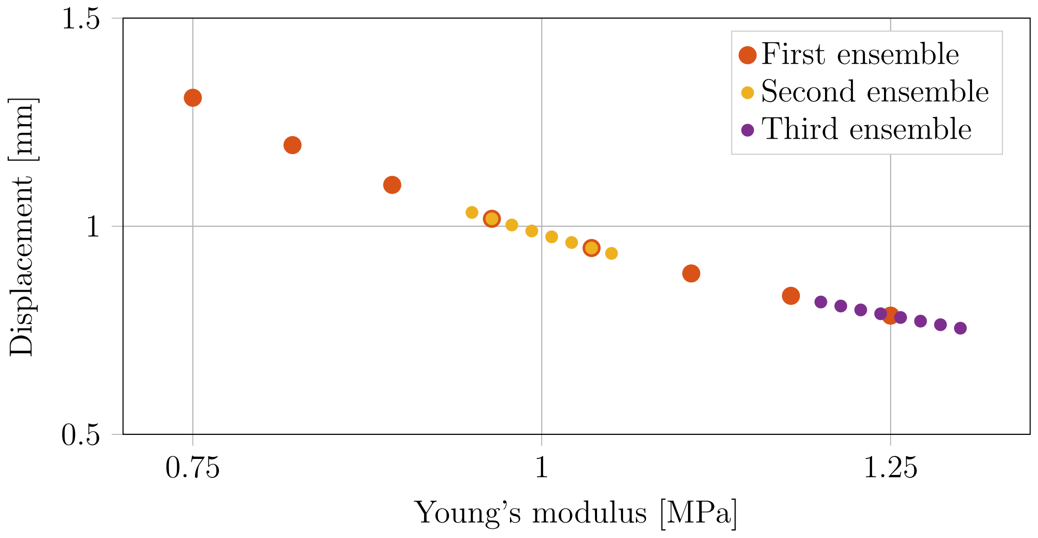


Figure 7.2: Quantity of interest of the 3 considered ensembles of test 1. The first ensemble is centered on $E = 1$ MPa and has a range width of 0.5 MPa. The second ensemble is centered on $E = 1$ MPa too but has a range width of 0.1 MPa. The last ensemble is centered on $E = 1.25$ MPa and has a range width of 0.1 MPa.

We can reach the same theoretical result using the spectral theory of section 3.4.2. We know that the eigenvalues of two different samples are the same up to a constant multiplier α. Therefore, every sample propagated alone share the same convergence as being independent of the constant multiplier.

First the quantity of interest, the vertical displacement of the point at the center of the pulled surface, is shown as a function of the Young's modulus in Fig. 7.2. We observe the predictable behavior: the larger the Young's modulus the smaller the displacement.

The eigenvalues of the stiffness matrix are represented in Fig. 7.3 for the smallest, mean, and largest value of the Young's modulus of the first considered ensemble. As already said, we observe that the spectra are scaled by the factor α changing the Young's modulus.

Finally, the convergence of ensemble GMRES is represented in Fig. 7.4. We observe, as discussed, that the gathering of the spectrum decreases the convergence of ensemble GMRES with ensemble reduction as the number of iterations required for ensemble 1, 2, and 3 to converge increases while using ensemble reduction. However, this increase is different for the 3 ensembles and two phenomena can be highlighted. For a fixed centered value, the larger the variability in the Young's modulus, the larger the required number of iterations to converge. If the variability decreases, for instance comparing ensemble 1

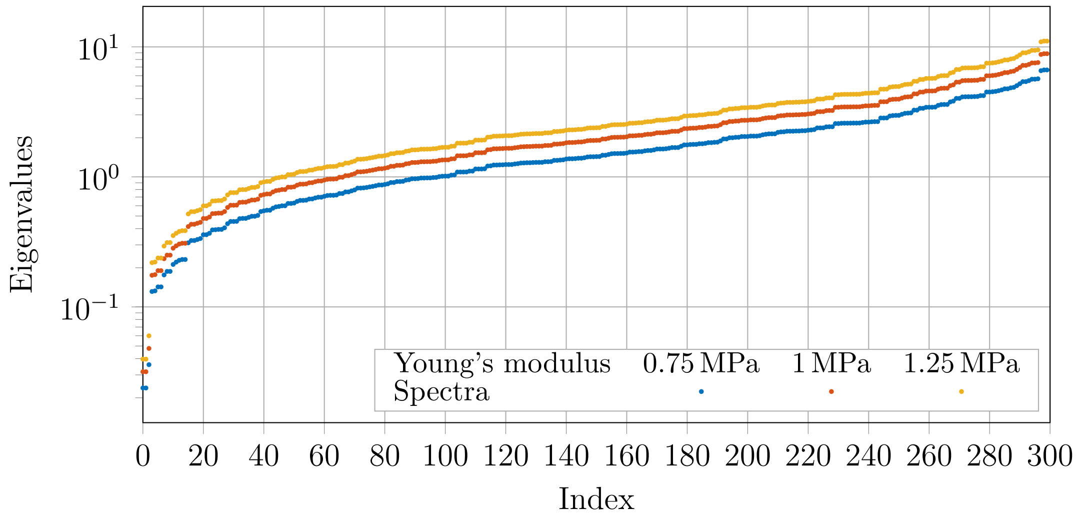


Figure 7.3: Spectra of the stiffness matrix for Young's modulus equal to 0.75 MPa, 1. MPa, and 1.25 MPa. The spectrum is the same for the 3 values up to a constant multiplier.

and 2, the spectra of each sample inside a same ensemble become closer which accelerates the convergence of ensemble GMRES with reduction. The second phenomenon is that increasing the centered value improves the convergence. This is due to the fact that this increased centered value decreases the relative variability: moving from the ensemble 2 to 3, the relative variability decreases from 0.1/1 to 0.1/1.2. The convergence of ensemble 2 and 3 would have been the same if the range width of ensemble 3 was 0.12 MPa instead of 0.1 MPa for the same reason as in the discussion of section 3.4.2.

When not using ensemble reduction, each sample individually converges as fast as without ensemble propagation.

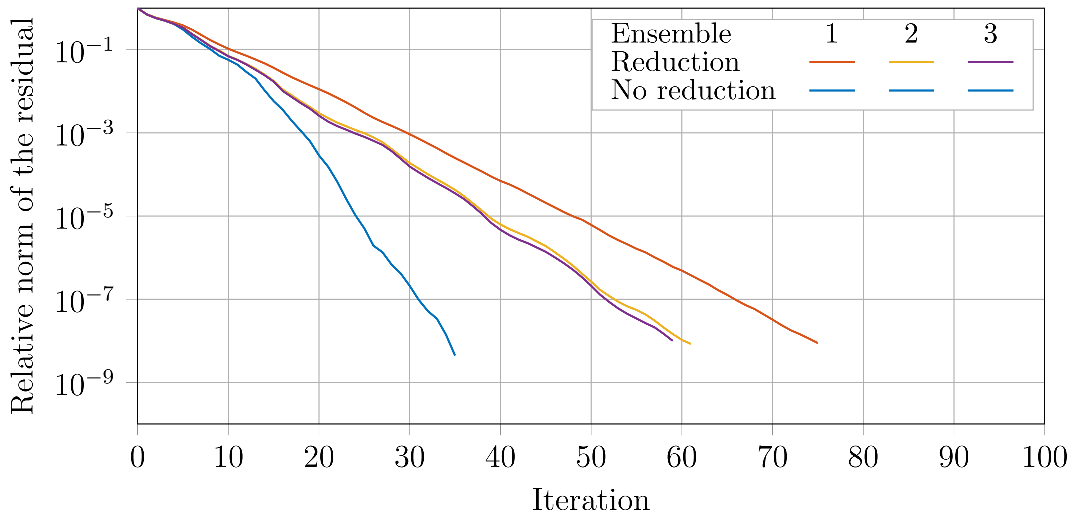


Figure 7.4: Convergence of the 3 ensembles of the test 1 with and without ensemble reduction. The convergence of each sample propagated alone is identical to the convergence without reduction and represented in blue. The range width and the centered value of the Young's modulus of the ensemble impact the convergence using ensemble reduction: the smaller the relative range, the smaller the number of iterations required to converge.

7.1.2 Varying surface load

In this case, we consider one ensemble of size 8 for which we fix the Young's modulus to 1 MPa and use a surface load which varies linearly between 0.075 MPa and 0.125 MPa. The quantity of interest is shown in Fig. 7.5.

For this example, starting with a zero initial guess and multiplying the right-hand side by a scaling factor α, we have, once again, that the Krylov subspace is not modified. In this case, we even have the property that the coefficients of the projection of the solution on the Arnoldi vectors are independent of the scaling factor α. As a consequence, we can predict that the convergence of each sample of this example is independent of the use of ensemble reduction and is the same as propagating the sample alone. Once again, we can deduce the same behavior using the convergence theory.

We observe, on Fig. 7.6 the expected behavior; the convergence with ensemble reduction is as fast as each sample individually.

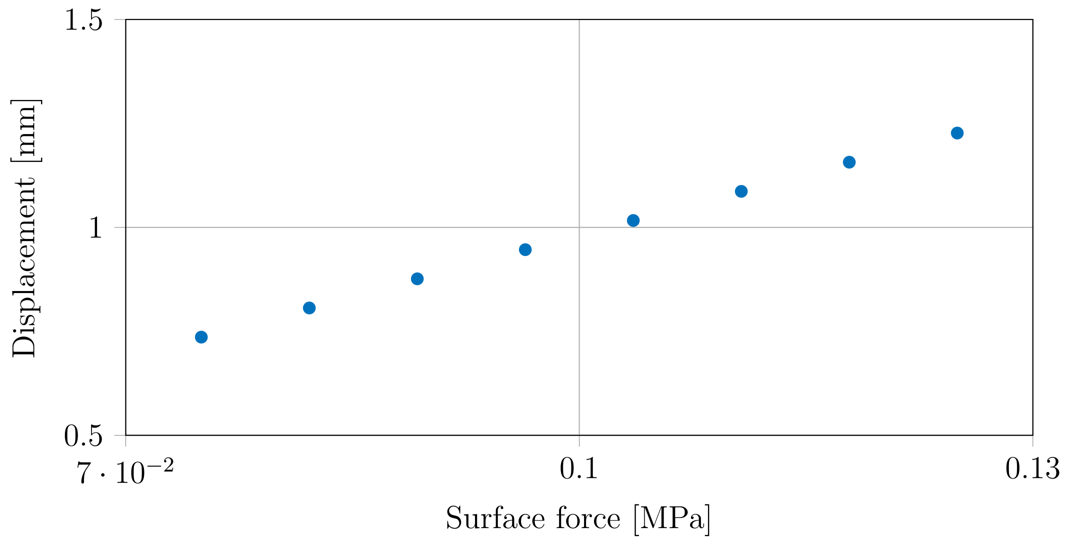


Figure 7.5: Quantity of interest of test 2 with 1 ensemble of size 8 for which we use a surface load which varies linearly between 0.075 MPa and 0.125 MPa.

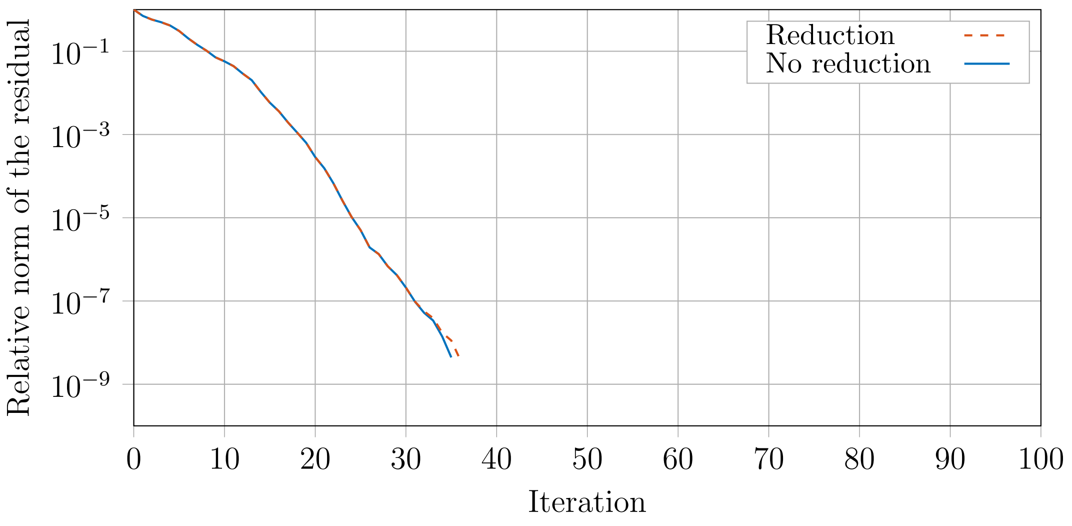


Figure 7.6: Convergence of test 2: using ensemble reduction does not impact the convergence as the sample share the same spectrum and have the same projection on the eigenvectors.

7.1.3 Varying Young's modulus and surface load

Finally, we scale both the surface force and the Young's modulus proportionally, leading to a sample-independent solution as illustrated by the quantity of interest in Fig. 7.7. Although the solutions are sample independent, the spectra of the sample matrix are sample dependent as discussed in section 7.1.1 and illustrated in Fig. 7.3.

Therefore, while using ensemble reduction, we have a poorer convergence as seen in Fig. 7.8.

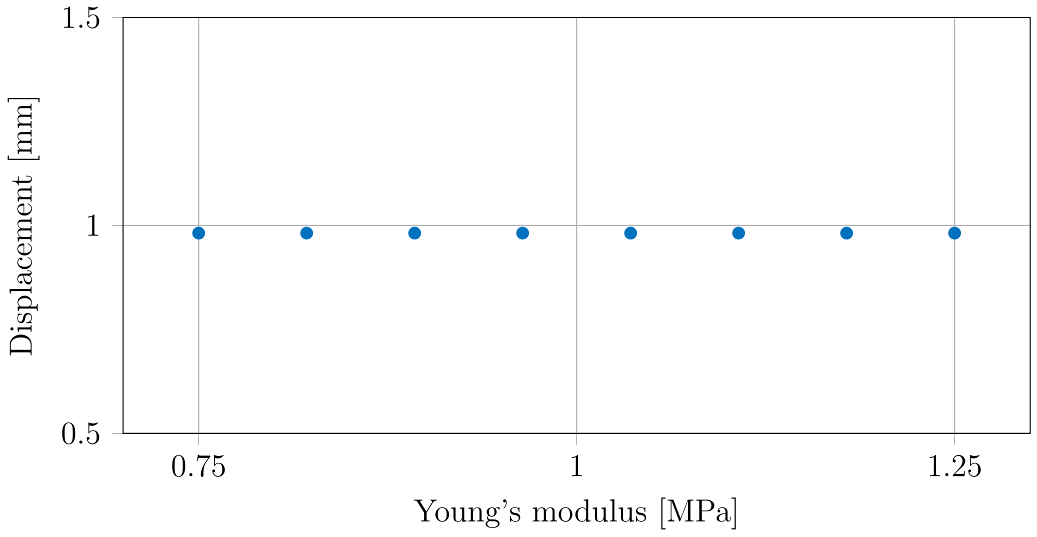


Figure 7.7: Quantity of interest of test 3: we use one ensemble of size 8 with a surface load which varies linearly between 0.075 MPa and 0.125 MPa and the Young's modulus which varies linearly between 0.75 MPa and 1.25 MPa. The quantity of interest is sample independent.

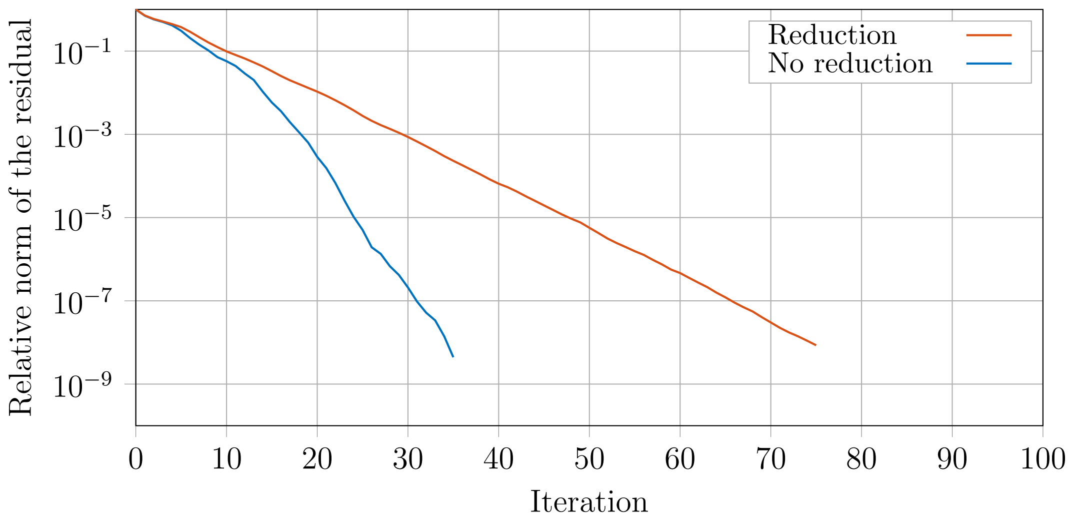


Figure 7.8: Convergence of test 3 for one ensemble of size 8 with a surface load which varies linearly between 0.075 MPa and 0.125 MPa and the Young's modulus which varies linearly between 0.75 MPa and 1.25 MPa. Even if the quantity of interest is sample independent, the spectra of the sample matrix are sample dependent leading to a slower convergence of ensemble GMRES with ensemble reduction.

7.1.4 Comments on the use of a preconditioner

Preconditioners are used in Krylov methods to improve their convergence as discussed in section 3.1.1. Using a Jacobi preconditioner on this example would fully remove the sample dependencies by rescaling the eigenvalues of the matrices. With such a preconditioner, ensemble divergence due to a scaling of the eigenvalues is suppressed and, therefore, convergence of ensemble GMRES with reduction is improved. To illustrate and confirm that, the convergence of the first ensemble of size 8 defined in section 7.1.1 using one Jacobi iteration as preconditioner is illustrated in Fig. 7.9. We observe that ensemble divergence is fully removed by the preconditioner in this simple case. Although finding a preconditioner that fully removes the sample variability is easy for this example, this is typically not the case for more complex problems. Moreover, such a property is not necessarily inherited from preconditioners studied in the context of deterministic problems.

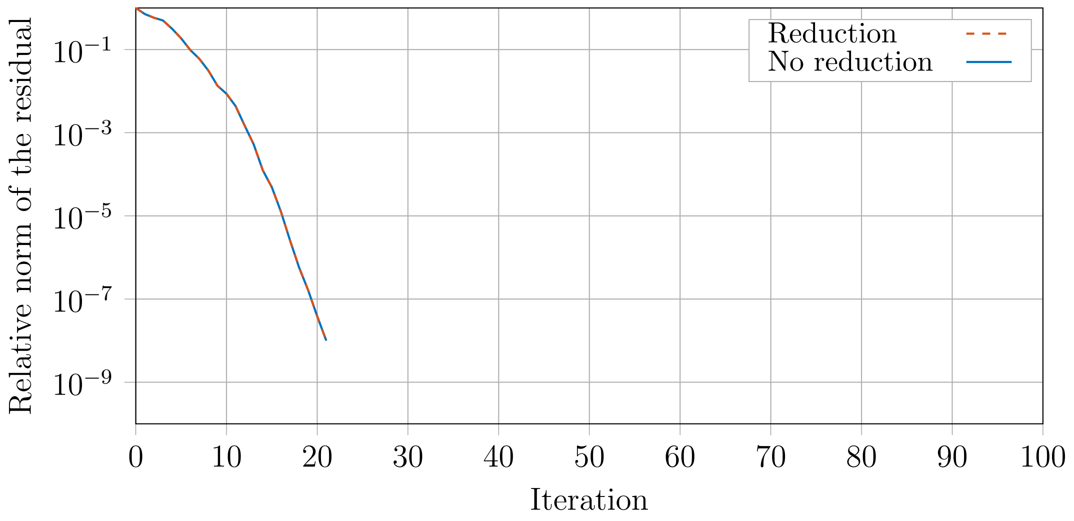


Figure 7.9: Convergence of the first ensemble of test case 1 using a Jacobi preconditioner. Using this preconditioner allows ensemble GMRES with reduction to converge as fast as GMRES applied on each sample alone.

7.1.5 Comments on the choice of the initial guess

In the previous tests, GMRES has always started with a zero initial guess. If the samples of a same ensemble are close to each other and if we start from a zero initial guess, GMRES with ensemble reduction will converge fast. First, it will converge towards an average value which can be close to the actual solutions for the different samples. If now, instead of starting from a zero initial guess, we start from the solution of the mean case, the initial residual for the different samples becomes more different from one to each other as the mean solution has been removed; we illustrate in this section the impact on the convergence of ensemble GMRES with reduction.

As opposed to the previous examples, this one cannot be explained directly by simpler intuitions as it is not straightforward to understand how the solution for each sample is different from the mean solution. Therefore, we restrict ourselves to the analysis with the convergence theory. In order to interpret the following results, we have to highlight that the theorem of section 3.4.1 assumes that the initial guess is zero. However, it is still possible to use this theorem by replacing the system (3.36) by the system:

$$\boldsymbol{A}\,\boldsymbol{y} = \boldsymbol{b} - \boldsymbol{A}\,\boldsymbol{x}^{(0)}, \tag{7.2}$$

where $\boldsymbol{x}^{(0)}$ is the non-zero initial guess and $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{x}^{(0)}$ is the difference between the solution $\boldsymbol{x}$ of the initial system (3.36) and the initial guess $\boldsymbol{x}^{(0)}$. With such a modification, the formulas (3.37) and (3.38) can still be used with the modification that, now:

$$\omega_{i_j} = |\boldsymbol{e}_{i_j}^T \boldsymbol{c}|^2 \text{ with } \boldsymbol{c} = \boldsymbol{\Phi}^\star \boldsymbol{r}^{(0)} = \boldsymbol{\Phi}^\star \boldsymbol{b} - \boldsymbol{\Phi}^\star \boldsymbol{A}\, \boldsymbol{x}^{(0)} = \boldsymbol{\Phi}^\star \boldsymbol{b} - \boldsymbol{\Lambda}\, \boldsymbol{\Phi}^\star\, \boldsymbol{x}^{(0)}. \tag{7.3}$$

Therefore, even if $\boldsymbol{\Phi}$ is sample independent in this case and therefore the weights ω_{i_j} are sample independent if starting from a zero initial guess, the weights ω_{i_j} are now sample dependent if GMRES starts from a non-zero initial guess as $\boldsymbol{\Lambda}$ is sample dependent. This results in the fact that the convergence of each sample propagated alone has not necessarily the same convergence curve as the other samples.

The fact that both the spectra of each sample and the projection of their initial residual on the eigenvectors are sample dependent reduces the convergence rate of ensemble GMRES with ensemble reduction compared to GMRES applied to each sample one at a time or to ensemble GMRES without ensemble reduction. This is illustrated in Fig. 7.10 for the first ensemble of the test case 1 where we start from the solution for $E = 1\,\text{MPa}$. We observe that, indeed, the convergence curve of each sample alone is sample dependent. The norms of residual shown in this figure are relative to the norm of the initial residual.

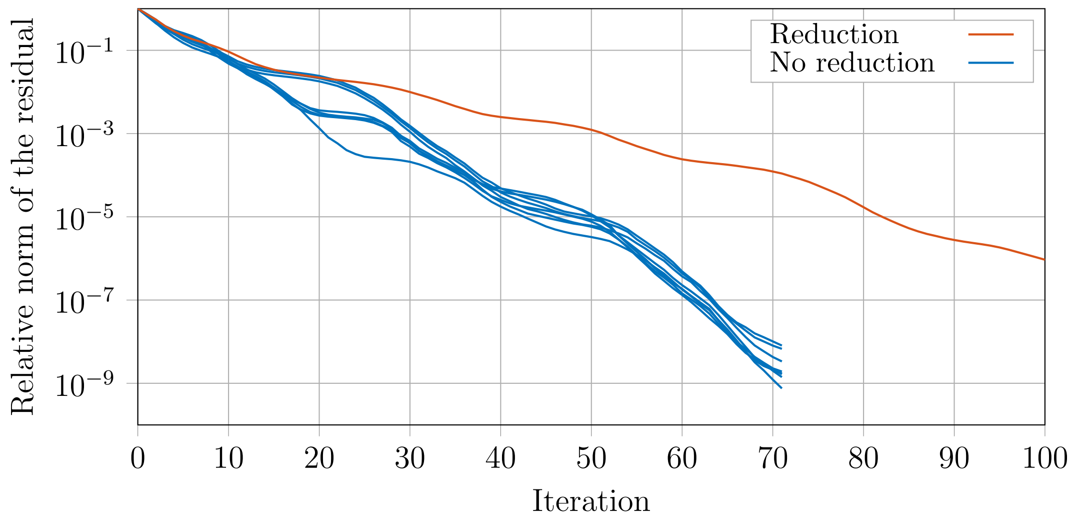


Figure 7.10: Convergence of the first ensemble of test case 1 starting from the solution for $E = 1\,\text{MPa}$. The non-zero initial guess leads to a sample dependent projection of the initial residual on the eigenvectors which leads to potential different convergence for each sample propagated alone.

7.1.6 Conclusion

These very simple tests illustrate how ensemble reduction can impact the convergence of ensemble GMRES by coupling samples together. The results presented show that ensemble GMRES without ensemble reduction is always more efficient, or at least not slower, in terms of the number of iterations to converge in the sense of the relative norm of the residual, than ensemble GMRES with ensemble reduction. Those simple cases and predicted behavior are consistent with the theory discussed in section 3.4.2.

The effect of ensemble reduction on the convergence depends on how different the projections of the initial residual of each sample on the eigenvectors of their associated matrices are and on the difference of the sample spectra. Preconditioners can help reduce ensemble divergence or even to suppress it in simple cases. The initial guess can influence the performance of ensemble GMRES with reduction by increasing the number of iterations required to converge.

7.2 Speed-up test

In this second example, we illustrate speed-up of ensemble GMRES with identical samples, i.e. with no sample variance to remove the effect of convergence on the speed-up.

The goal of this example is to highlight the speed-up of the different parts of ensemble GMRES as a function of the number of degrees of freedom per sample and the ensemble size.

To do so, we consider the same problem as the previous example where, now, the mesh has $9 \times 9 \times n_h$ hexahedra and n_h the number of hexahedra along the z direction is changed to cover a large range of numbers of degrees of freedom $n = 300\,(n_h + 1)$.

As an illustration, Fig. 7.11 includes two different meshes with $n_h = 9$ and $n_h = 36$.

Such a mesh refinement influences the aspect ratio of the elements. This is, however, not an issue for this example as we do not take into account the convergence of the GMRES method; we only measure speed-up of the different parts for a number of iterations fixed.

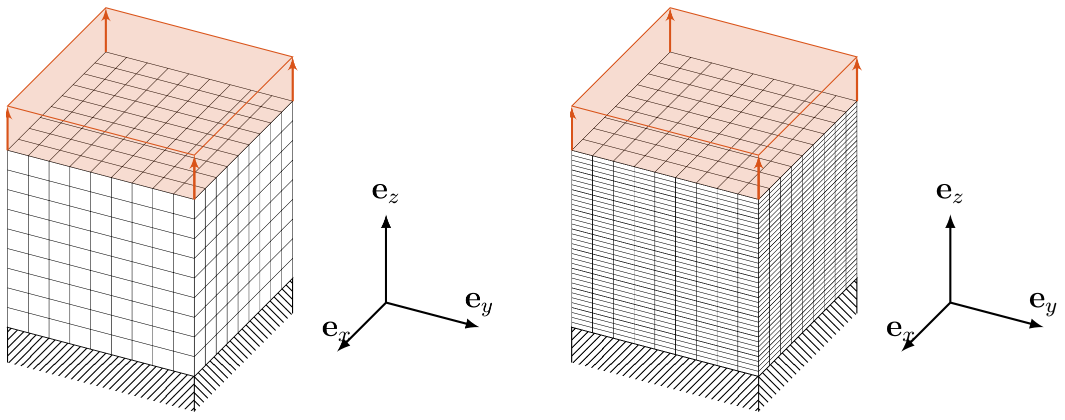


Figure 7.11: Examples of two meshes with $n_h = 9$ (left) and $n_h = 36$ (right).

In the following subsections, we investigate the speed-up of the sparse matrix-vector product (IV.IV in Fig. 6.8b), applying Gauss-Seidel iterations, a relaxation-based method as a preconditioner (IV.III in Fig. 6.8b), and the orthogonalization process (IV.X in Fig. 6.8b) running the example on one NUMA region of the Blake system described in Appendix A without hyper-threading. Each test has been run once with 24 threads.

7.2.1 Sparse matrix-vector product

As explained in section 2.2, Phipps et al. [2017] studied intensively the sparse matrix-vector product with ensemble propagation with sparse matrices in compressed row storage format. They derived optimistic and pessimistic throughput bounds for the sparse matrix-vector product that we recall here.

Assuming that the data type of each entry of the matrix is stored with b_{data} bytes, that the vectors are using the same data type and that the integers used to store the graph of the matrix are stored with b_{int} bytes, we have the following throughput bounds for the throughput of the sparse matrix-vector product without ensemble propagation [Phipps et al., 2017]:

- Pessimistic bound: for a given entry of the sparse matrix, the column index associated to this entry, the entry itself, and the value of the vector associated to the column index are not in the cache hierarchy and must be streamed from main memory. The total number of bytes that must be streamed is:

$$2\,b_{\text{data}} + b_{\text{int}}. \tag{7.4}$$

- Optimistic bound: for a given entry of the sparse matrix, the value of the vector associated to its column index can be reused from cache because it has been used not too long ago. The total number of bytes that must be streamed is:

$$b_{\text{data}} + b_{\text{int}}. \tag{7.5}$$

The throughput of the algorithm is therefore in the range:

$$\text{Bandwidth} \times \left[\frac{2\,\text{FLOPS}}{b_{\text{data}} + b_{\text{int}}}, \frac{2\,\text{FLOPS}}{2\,b_{\text{data}} + b_{\text{int}}} \right]. \tag{7.6}$$

In the case of the ensemble sparse matrix-vector product, because the column index is shared by all the samples, per stream, it is used s times and we have the throughput:

$$\text{Bandwidth} \times \left[\frac{2\,s\,\text{FLOPS}}{b_{\text{data}}\,s + b_{\text{int}}}, \frac{2\,s\,\text{FLOPS}}{2\,b_{\text{data}}\,s + b_{\text{int}}} \right]. \tag{7.7}$$

Those bounds rely on the assumption that the largest cache is not sufficiently large to fully store all the data needed for the computation of the sparse matrix-vector product. As a consequence of the increased memory usage due to ensemble propagation, there are matrix sizes for which the assumption is not true without ensemble propagation and true with ensemble propagation. For such cases, speed-up of ensemble propagation can be expected to be smaller than 1 as all the data needed for the computation of the sparse matrix-vector product can be stored into the largest cache not using ensemble propagation. However, for sufficiently large matrices the derived bounds predict speed-up larger than 1.

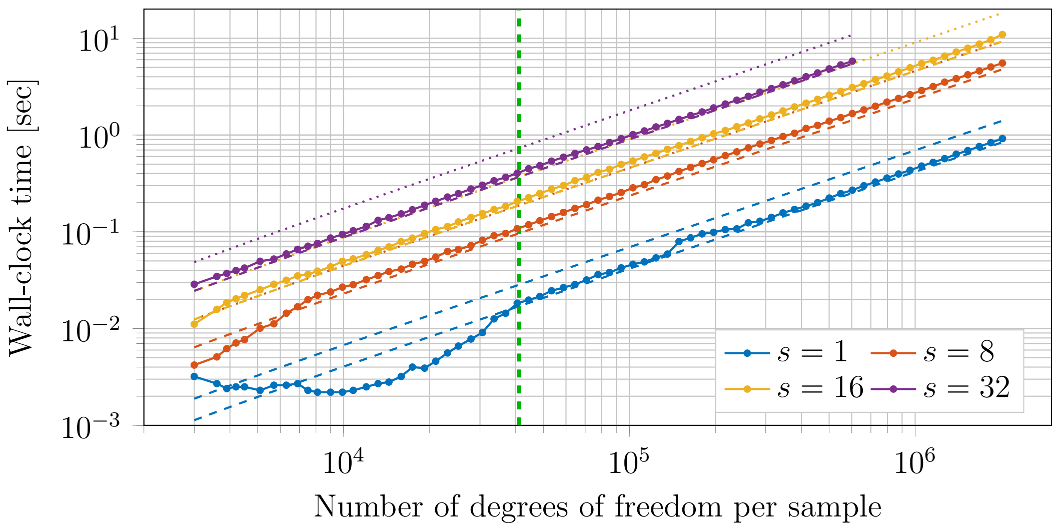


Figure 7.12: Wall-clock time of the matrix-vector product.

We now illustrate the accumulated wall-clock time of the sparse matrix-vector product of ensemble GMRES with 50 iterations for different ensemble sizes and numbers of degrees of freedom per sample in Fig. 7.12 where the dashed lines represent the wall-clock time bounds based on the throughput and the time complexity of the sparse matrix-vector product[1]. The related speed-up are illustrated in Fig. 7.13.

As stated before, we observe that for cases with less than 41097 degrees of freedom per sample, represented as the dashed green line in Fig. 7.12, the speed-up of ensemble propagation can be smaller than 1 due to the fact that data fit in cache for $s = 1$ but not necessarily for larger s.

For larger matrices, we observe that both the wall-clock time with and without ensemble propagation mainly follow the wall-clock time curve deduced by the optimistic bounds which lead to a speed-up of about 1.5 as $b_{\text{data}} = 8$ and $b_{\text{int}} = 4$ in our case.

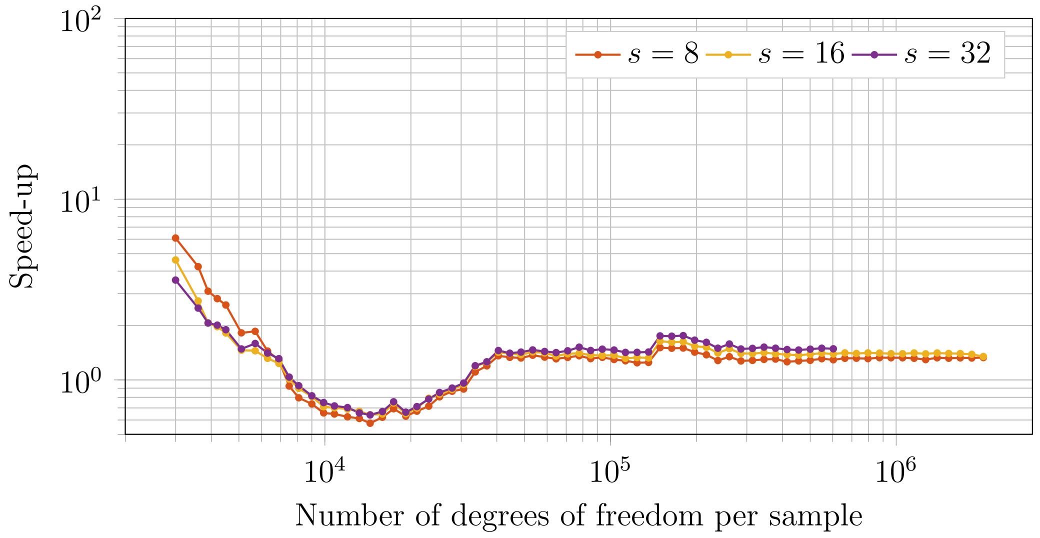


Figure 7.13: Speed-up of the sparse matrix-vector product.

7.2.2 Gauss-Seidel iterations

Concerning the preconditioner, here we restrict ourselves to using the threaded Gauss-Seidel of Ifpack2 [Prokopenko et al., 2016] which calls the threaded Gauss-Seidel of the KokkosKernels package based on graph coloring [Deveci et al., 2016a]. This brings us important information on the speed-up of ensemble propagation of multigrid preconditioners which we will use in the next examples.

We observe on Fig. 7.14 that the wall-clock time of the threaded Gauss-Seidel decreases with larger problem sizes for $s = 1$ with about 10^4 degrees of freedom. We suspect this behavior to be due to the fact that, for the smaller problems with $s = 1$, the number of threads is too large leading to potential overhead costs such as false sharing. Extra information about false sharing and the impact of ensemble propagation on it can be found in Appendix D. The threaded Gauss-Seidel used in this work is based on a graph coloring strategy to parallelize the loop over the rows. First, the algorithm colors the

[1]The time complexity, i.e. the number of elementary operations performed by the algorithm, of the sparse matrix-vector product used here is $O(\text{nnz})$ where nnz is the number of non zero entries of the sparse matrix.

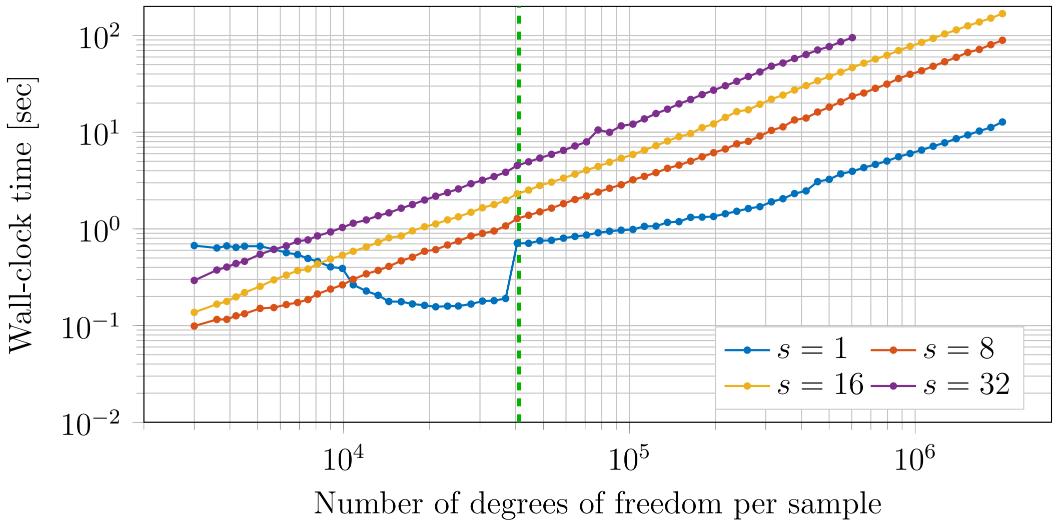


Figure 7.14: Wall-clock time of the threaded Gauss-Seidel iteration.

rows of the matrix $\boldsymbol{A}$ such that each row i does not have the same color as any related row j, i.e. if $a_{ij} \neq 0$, i and j do not share the same color. This first step aims to have as few colors as possible. The second step replaces the loop over the rows of the Gauss-Seidel by two nested loops: a serial loop over the colors and a threaded loop over the rows of a given color. The potential false sharing is occurring during the parallel computation of the values of a same graph color and disappears for $s = 8$, $s = 16$, and $s = 32$ as the scalar type fills the cache line and prevents false sharing. In order to investigate this statement, we have evaluated the performance of the threaded Gauss-Seidel applied on a diagonal matrix and the performance of a threaded Jacobi preconditioner and have observed the same behavior: for small problems with $s = 1$, the wall-clock time decreases when increasing the problem size. This, hand in hand with Appendix D, highly suggests that the decrease of the wall-clock time of the threaded Gauss-Seidel with larger problem sizes is due to false sharing as the coloring of a diagonal matrix only requires one color.

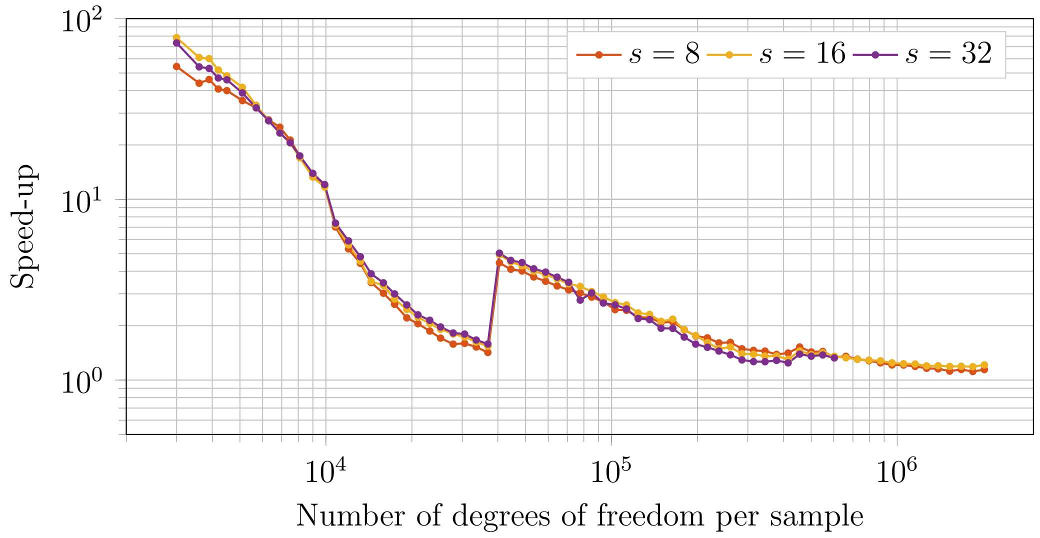


Figure 7.15: Speed-up of the Gauss-Seidel iteration.

The jump in wall-clock time for $s = 1$ at about 41097 degrees of freedom is explained by the fact that the matrix cannot be fully stored anymore in cache.

The speed-up illustrated in Fig. 7.15 is larger for smaller problem sizes. This observation is interesting for multigrid preconditioners as the problem size decreases when moving to coarser levels.

7.2.3 Orthogonalization process

Finally, the wall-clock time and speed-up of the orthogonalization process illustrated in Fig. 7.16 and 7.17 are as expected; the speed-up is 1 provided that the problem size is sufficiently large. Otherwise, a larger speed-up can be observed due to the fact that the GEMV of the MKL does not reach the full memory bandwidth for those problem sizes.

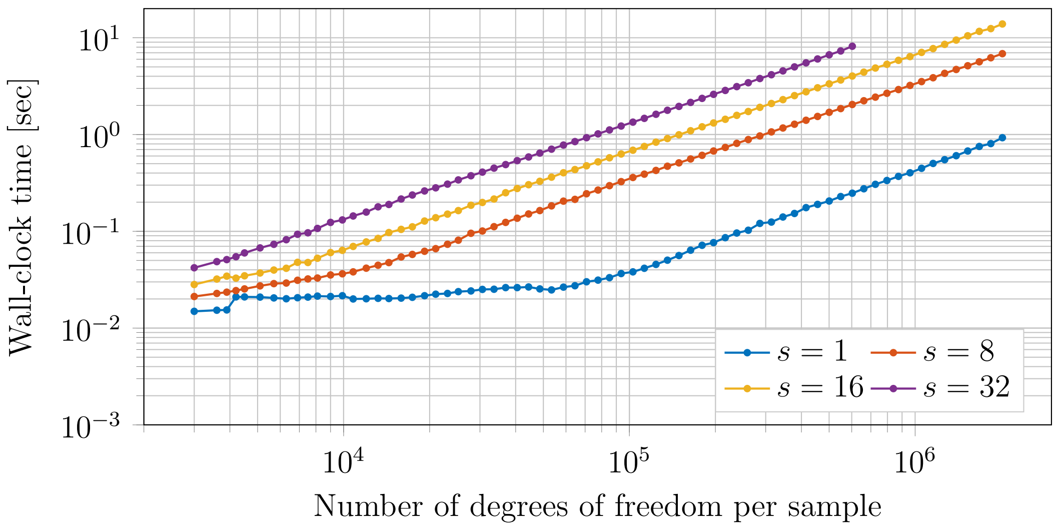


Figure 7.16: Wall-clock time of the orthogonalization process.

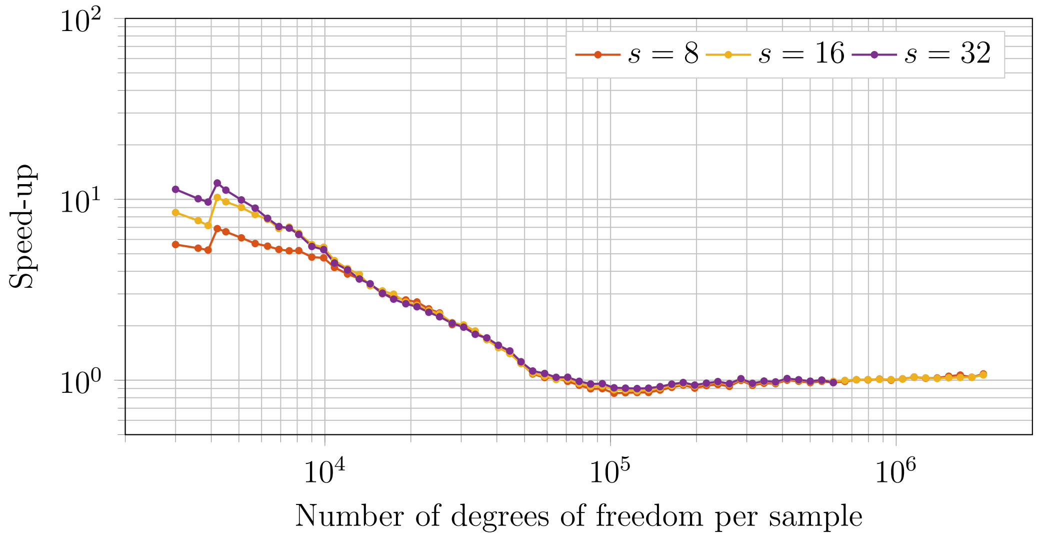


Figure 7.17: Speed-up of the orthogonalization process.

7.2.4 Relative CPU costs

The speed-up of a fixed number of iterations of ensemble GMRES depends on the speed-ups of each computational parts alone, which have been illustrated in the above subsections, and on their relative size in terms of CPU costs; the longer its duration, the larger the influence of its speed-up.

In this subsection we illustrate the average wall-clock time contributions of the different computational parts of ensemble GMRES for the studied example with 50, 100, and 200 iterations of ensemble GMRES in Fig. 7.18.

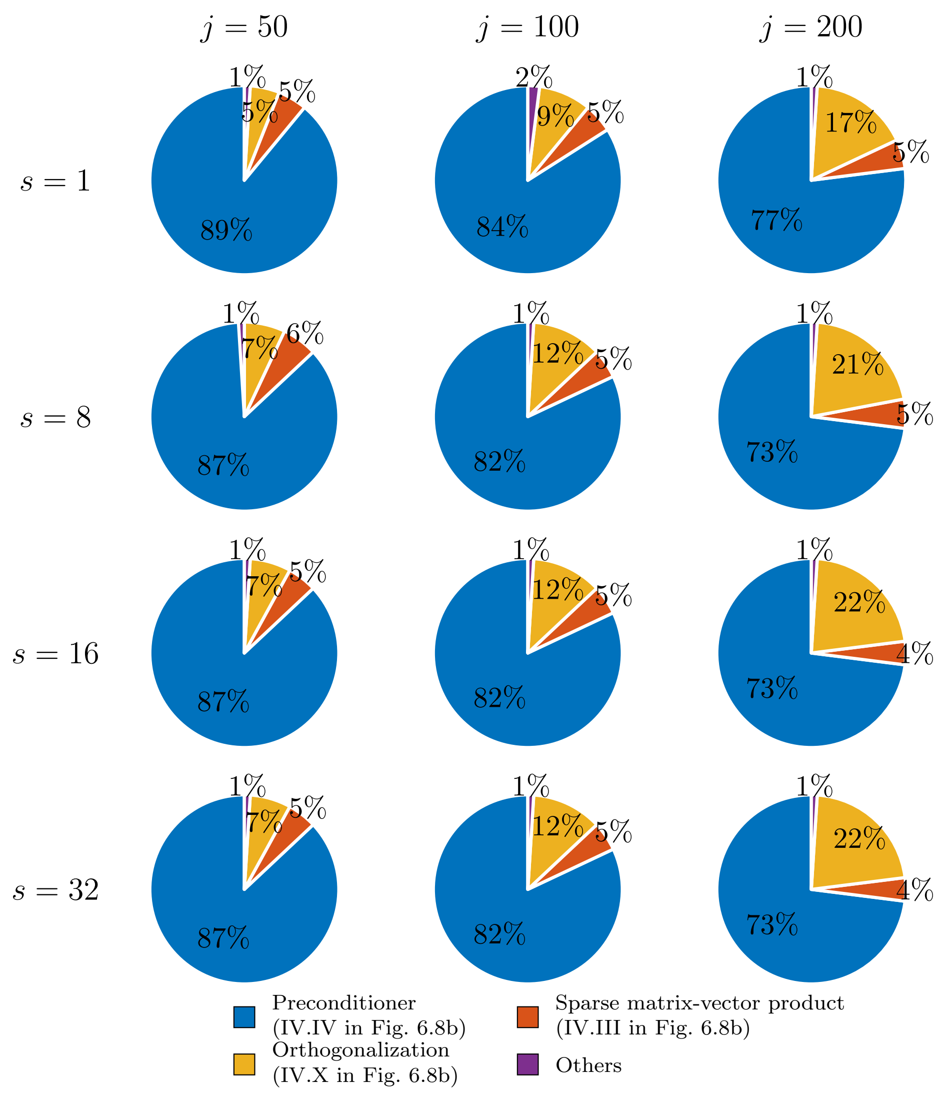


Figure 7.18: CPU contributions of the computational parts of GMRES with 6.027 10^5 degrees of freedom per sample.

As expected, we observe that using ensemble propagation increases the relative contribution of the orthogonalization process as its speed-up is the smallest. As expected too, increasing the number of iterations of GMRES increases the relative contribution of the orthogonalization process as its cost per iteration grows with the number of iterations.

7.2.5 Conclusions

Finally, this example illustrated speed-up of ensemble propagation on the different parts of an ensemble GMRES with a given preconditioner. In order to deduce the speed-up of the full ensemble GMRES the convergence has to be taken into account as it impacts the speed-up of ensemble GMRES in two ways:

- The larger the number of iterations required to converge, the larger the relative cost of the orthogonalization CPU cost as its cost per iteration grows with the number of iterations. This results in a larger, in terms of wall-clock time, relative part of ensemble GMRES with less good speed-up and reduces the expected speed-up of the full ensemble GMRES. This issue can be mitigated using a GMRES restart strategy.
- The increase of the number of iterations either due to ensemble reduction or due to different trip count values for different samples of an ensemble increases the number of iterations leading to a reduced speed-up too.

In terms of wall-clock time, ensemble GMRES differs from the ensemble CG studied in [Phipps et al., 2017] mainly by the inclusion of an orthogonalization process. Moreover, as the orthogonalization process has a speed-up which tends to 1, we can deduce that, for a fixed number of Krylov iterations, the speed-up of the ensemble CG is better than the speed-up of ensemble GMRES for problems that can be solved both with CG and GMRES. Although both the first two examples presented in this work could have been solved with ensemble CG, the following ones do not have symmetric positive definite matrices and ensemble CG cannot be applied anymore.

7.3 Mesh-tying problem

In this section, we illustrate on an indefinite problem the equivalent wall-clock time of one iteration of ensemble GMRES with and without reduction thanks to the implementation of the previous sections. We illustrate the accelerated convergence of ensemble GMRES without reduction and the improved speed-up of ensemble propagation.

The example is a mesh-tying problem in saddle-point formulation which is symmetric but not positive definite. This example is a first step towards ensemble propagation applied to non-linear contact problems with Mortar formulation.

The example has been run on 1 NUMA region of the Blake system described in Appendix A with hyper-threading, i.e. with 48 threads.

7.3.1 Equations of the mesh-tying problem seen as a saddle-point problem

In this section we are interested in a mesh-tying problem seen as a saddle-point problem as described in section 5.1.3 and in [Wohlmuth, 2001]:

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{G} \\ \boldsymbol{G}^{\mathrm{T}} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{l} \end{bmatrix} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{0} \end{bmatrix}, \tag{7.8}$$

where $\boldsymbol{K}$ is the stiffness matrix, $\boldsymbol{G}$ is the Mortar matrix arising from the discretization of the coupling between the displacement $\boldsymbol{u}$ and the Lagrange multipliers $\boldsymbol{l}$ which enforce continuity constraints at the mesh-tying interface, and $\boldsymbol{f}$ is the vector including the external forces.

7.3.2 Mechanical Lamé parameters represented using a random field

Uncertainties are introduced in the model by using a log-normal random field for the shear modulus μ:

$$\mu(\mathbf{x}) = \exp\left[\log\left(\frac{g}{\sqrt{1+\delta^2}}\right) + \sqrt{\log\left(1+\delta^2\right)}\,\chi^{\mu}(\mathbf{x})\right], \tag{7.9}$$

where g is the mean value of the shear modulus, δ the coefficient of variation, and $\chi^{\mu}(\mathbf{x})$ is a Gaussian random field realized by using a spectral approach with a Gaussian covariance function based on equation (39) in [Poirion and Soize, 1995] with a correlation length L.

The first Lamé parameter λ is then computed as follows:

$$\lambda(\mathbf{x}) = \frac{2\mu(\mathbf{x})\nu}{1-2\nu}, \tag{7.10}$$

where ν is the Poisson's coefficient which is constant and sample independent.

7.3.3 Preconditioner for the saddle-point problem

In this work, we use the preconditioner of section 5.1.6 with threaded Gauss-Seidel iterations as sub-smoothers for the block $\boldsymbol{K}$ with 3 sweeps and a damping factor of 0.8,

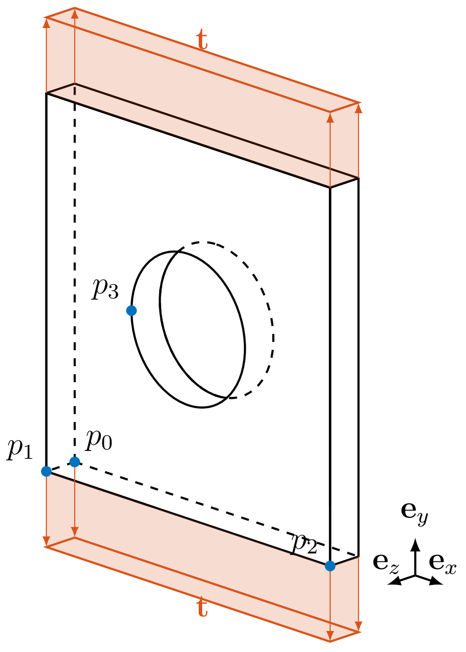


Figure 7.19: Illustration of the plate with a hole example where the plate is pulled along the $\mathbf{e}_y$ direction with a surface force $\mathbf{t}$ on both opposite surfaces.

and Basker [Booth et al., 2017], a sparse-direct solver that is built into Trilinos, as the smoother of the block $\boldsymbol{R}$ and as the coarsest level solver.

The multigrid preconditioner used is implemented in the MueLu package [Prokopenko et al., 2014]. We used two multigrid levels and a SIMPLE smoother with $\alpha = 0.8$ and $\beta = 1$ on the fine level.

7.3.4 Plate with a hole example

The example considered is a plate with a hole as shown in Fig. 7.19. The plate is a 10 cm × 10 cm × 1 cm with a hole of radius 2 cm at the center. The plate is pulled along the $\mathbf{e}_y$ direction with a surface force $\mathbf{t}$ on both opposite surfaces with a magnitude of 700 MPa on each side. The average Young's modulus is 70000 MPa and the Poisson's coefficient is 0.35. The plate is clamped as illustrated in Fig. 7.19: the point p_0 is clamped in all directions, p_1 in the directions $\mathbf{e}_x$ and $\mathbf{e}_y$, and p_2 in the direction $\mathbf{e}_y$.

We are not taking into account the symmetry of the geometry to mesh it as the random field, which represents the Lamé parameters, does not share the same symmetry property.

A 3D hexahedral mesh has been used with finer hexahedra in a domain close to the hole as shown on the cut in Fig. 7.20. This problem has 32088 degrees of freedom per sample including 2088 Lagrange multipliers.

The displacement along $\mathbf{e}_y$ and the equivalent von Mises stress of the cut for the mean case are shown in Fig. 7.21 and 7.22 respectively.

As discussed before, we introduce variability between samples using a 2D log-normal random field which is constant along $\mathbf{e}_z$. We consider the six cases described in Table 7.1. In Fig 7.23 and 7.24, we illustrate six realizations of the shear modulus μ for the test cases 2 and 6 respectively.

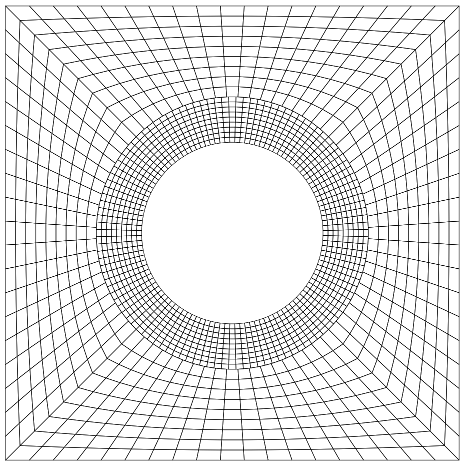


Figure 7.20: Cut of the 3D hexahedron mesh with finer elements in a region of width 1 cm close to the hole.

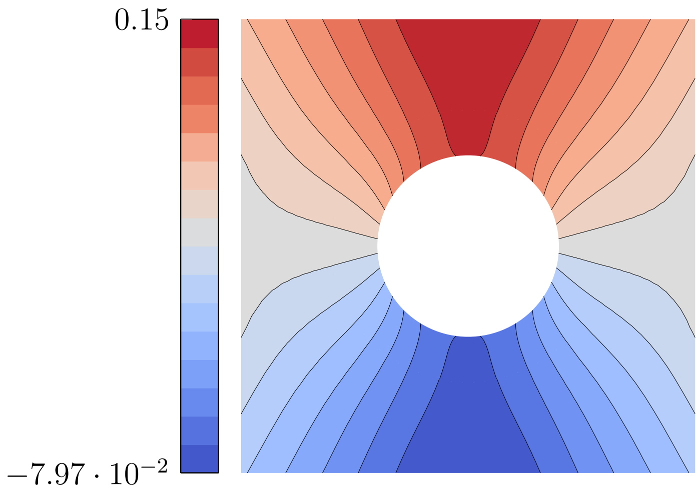


Figure 7.21: Vertical displacement of the mean sample in cm.

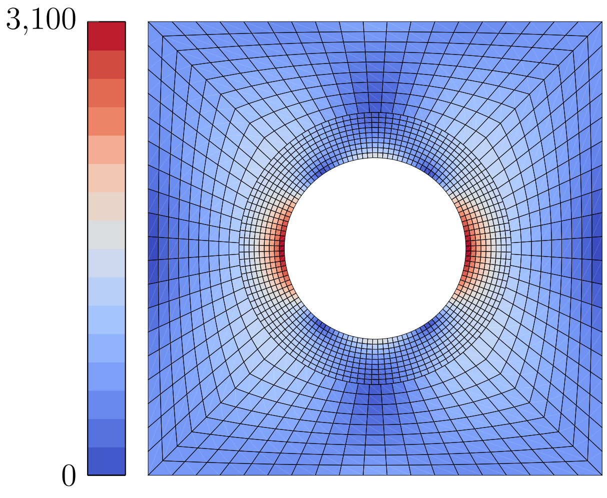


Figure 7.22: Equivalent von Mises stress in average per element of the mean sample in MPa.

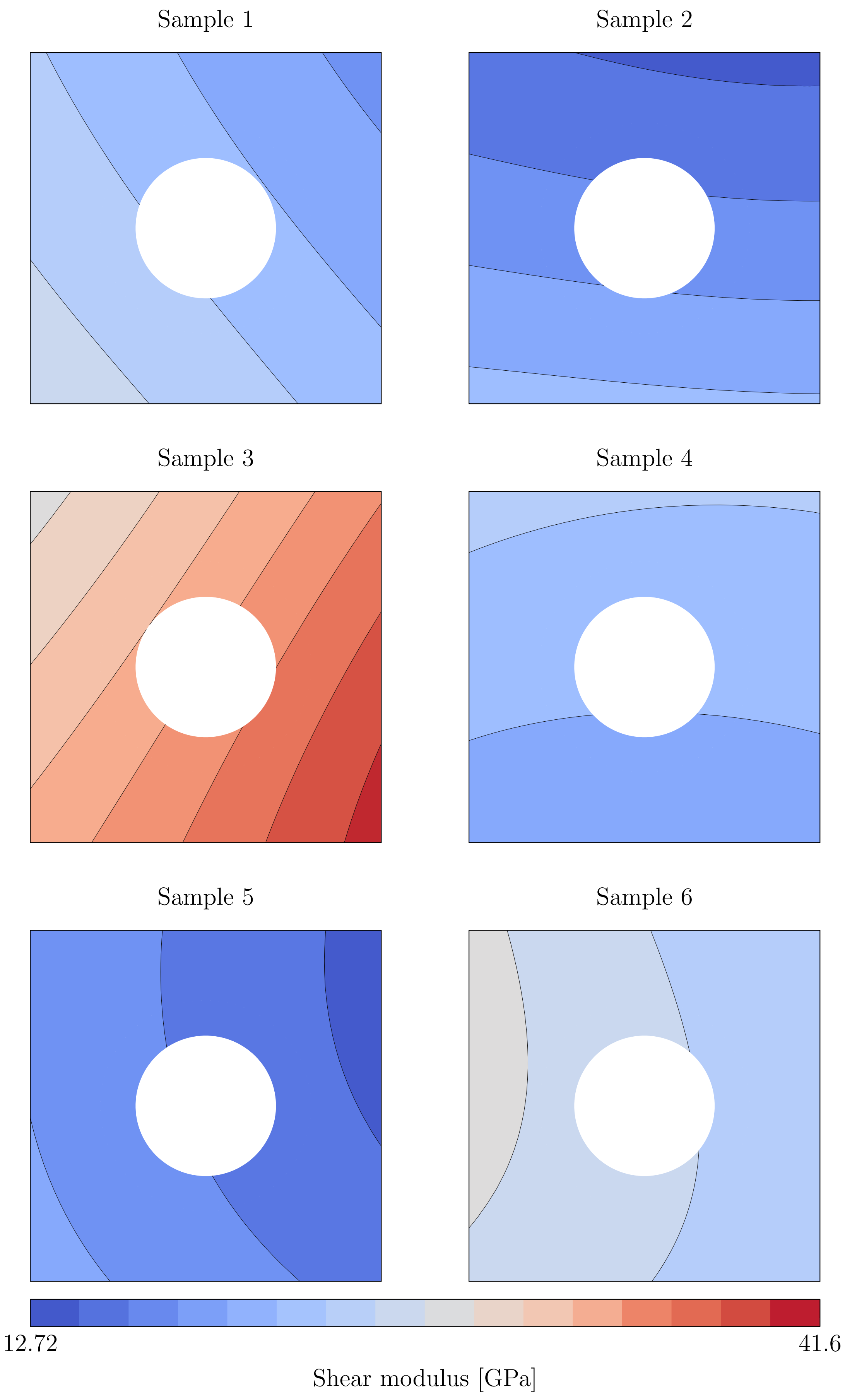


Figure 7.23: Six samples of the test case 2 with a correlation length of $L = 20\,\text{cm}$ and $\delta = 0.3$.

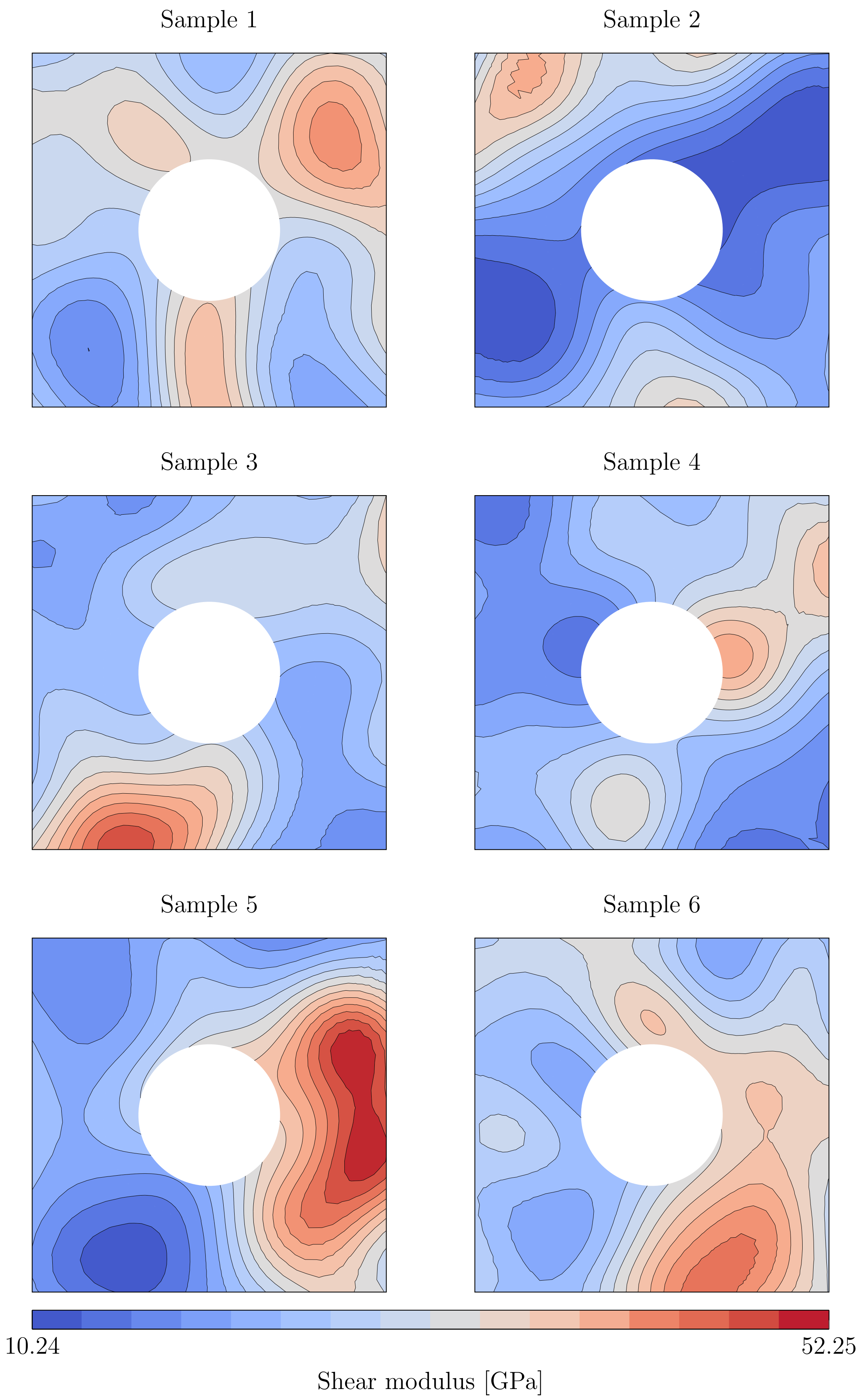


Figure 7.24: Six samples of the test case 6 with a correlation length of $L = 2\,\mathrm{cm}$ and $\delta = 0.3$.

	$\delta = 0.1$	$\delta = 0.3$
$L = 20\,\mathrm{cm}$	Test case 1	Test case 2
$L = 5\,\mathrm{cm}$	Test case 3	Test case 4
$L = 2\,\mathrm{cm}$	Test case 5	Test case 6

Table 7.1: Different test cases with different correlation lengths L (in both $\mathbf{e}_x$ and $\mathbf{e}_y$ directions) and different coefficients of variation.

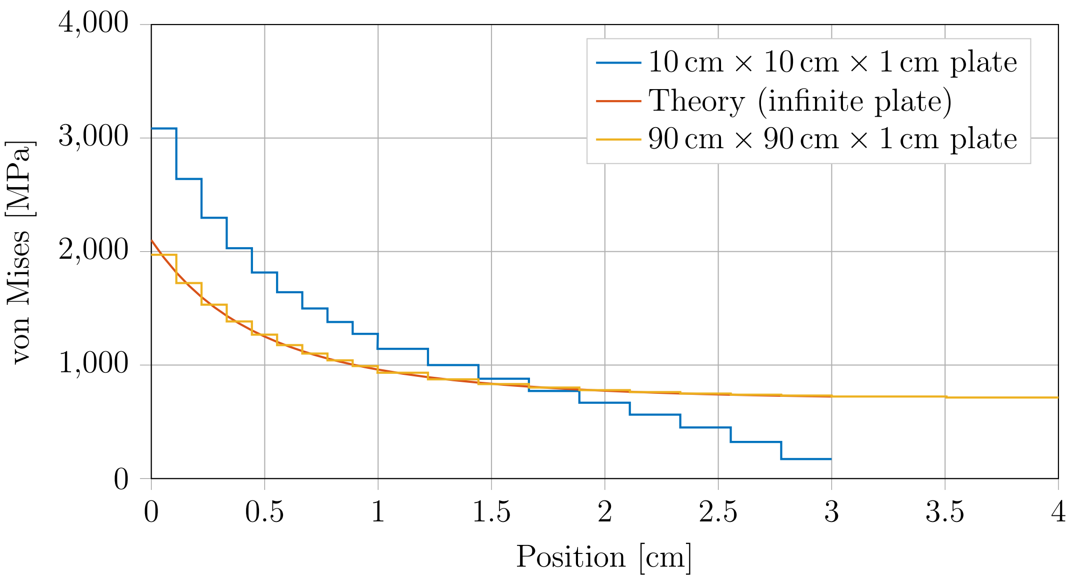


Figure 7.25: Numerical von Mises stresses in average per element for $10\,\mathrm{cm} \times 10\,\mathrm{cm} \times 1\,\mathrm{cm}$ plate and $90\,\mathrm{cm} \times 90\,\mathrm{cm} \times 1\,\mathrm{cm}$ plate, and theoretical von Mises stresses computed using Kirsch equations.

7.3.5 Convergence of the solution for the mean case

In order to validate the mesh and to show that it has converged, we have extracted the von Mises stress on the line $\mathbf{e}_x$ passing by the point p_3 and illustrate them in Fig. 7.25. In Fig. 7.25 we have added the theoretical von Mises stress computed using the Kirsch equations [Kirsch, 1898] which assume an infinite plate in one directional tension. The numerical results in blue have larger stresses close to the hole and smaller stresses close to the boundary on the line $y = 5\,\mathrm{cm}$ we found that such differences are due to the fact that the hole in the plate has a significant radius compared to the dimension of the plate. This has been highlighted using the mesh shown in Fig. 7.20 surrounded by a uniform mesh such that the plate is now $90\,\mathrm{cm} \times 90\,\mathrm{cm} \times 1\,\mathrm{cm}$ and we have noticed that the results converged to the one of the Kirsch equations as illustrated by the yellow curve in Fig. 7.25. This demonstrates that the mesh is sufficiently refined to represent correctly the von Mises stresses.

7.3.6 Results

The most interesting speed-up of using ensemble propagation is the speed-up of the full simulation to answer the question of how much time we can save using ensemble propagation or how many more samples can we evaluate in the same wall-clock time compared to not using ensemble propagation.

In these results, we use 64 Monte Carlo samples for each case just to study the performance.

Speed-up of each computational part

In order to remove the influence of the rate of convergence of GMRES, we discuss the speed-up for a fixed number of 200 iterations without taking into account a stopping criterion. Moreover, by removing the influence of the convergence of GMRES, the different cases described in Table 7.1 do not influence the speed-up here as exactly the same floating-point operations have to be computed independently of the values of the random field.

We split the total wall-clock time in four categories: the matrix assembly process where the saddle-point matrix is computed (II in Fig. 6.8a), the sparse matrix-vector product that is evaluated while applying the matrix to the previous Arnoldi vector in GMRES (IV.IV in Fig. 6.8b), the application of the preconditioner (IV.III in Fig. 6.8b), and the orthogonalization process of GMRES (IV.X in Fig. 6.8b).

Looking first at the speed-up of the orthogonalization process listed in Fig. 7.26, we observe that the speed-up is almost independent of whether ensemble reduction is used or not. This is consistent with the results of Fig. 4.14 and Fig. 4.13 from the Chapter 4 and confirms that the wall-clock time of one iteration of ensemble GMRES with and without reduction is equivalent thanks to the performance of the ensemble GEMV discussed in the previous section.

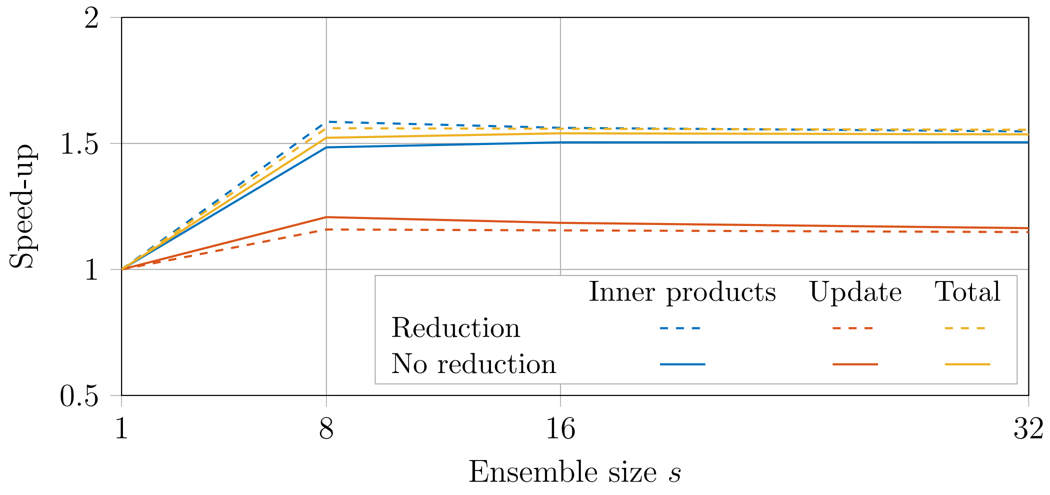


Figure 7.26: Speed-up of the inner products (IV.V in Fig. 6.8b), the update (IV.VI in Fig. 6.8b), and the full orthogonalization process (IV.X in Fig. 6.8b) with and without reduction.

In section 4.2, we have explained that GEMV is memory bound and, therefore, the speed-up of applying ensemble propagation on the update has to be close to 1.0 providing that both the update without ensemble propagation and the one with ensemble propagation have maximal theoretical performance. However, we see from the results in Fig. 7.26 that the speed-up of both the update and the inner product are larger than 1.0. This is explained by the fact that, while not applying ensemble propagation, the measured throughput of the GEMV of the MKL with a matrix stored using a left layout and a size of 32000×300 is 19 GFLOPS. The throughput is, however, increased with larger matrix size; for example, if ensemble reduction is used and the ensemble size is 8, the throughput of the update applied on a $(8 \times 32000) \times 300$ matrix is now 24 GFLOPS. Therefore, the speed-up which are larger than 1.0 are explained by the fact that the tested problem has too few degrees of freedom for one sample to reach maximal throughput of the MKL GEMV on the CPU considered.

The speed-up of the remaining parts, the assembly process, the sparse matrix-vector product, and the preconditioner are independent of whether ensemble reduction is used or not. The results of those categories are listed in Fig. 7.27. The speed-up of the matrix assembly process is large because the computation of the Mortar matrix is sample independent and, therefore, is only computed once per ensemble. This Mortar matrix is sample independent because the samples share the same mesh. The wall-clock time of the full simulation listed in Table 7.2 lists the wall-clock time to compute the Mortar matrix with and without ensemble propagation; we see the impact of only computing the matrix once per ensemble and its influence on the speed-up of the matrix assembly process.

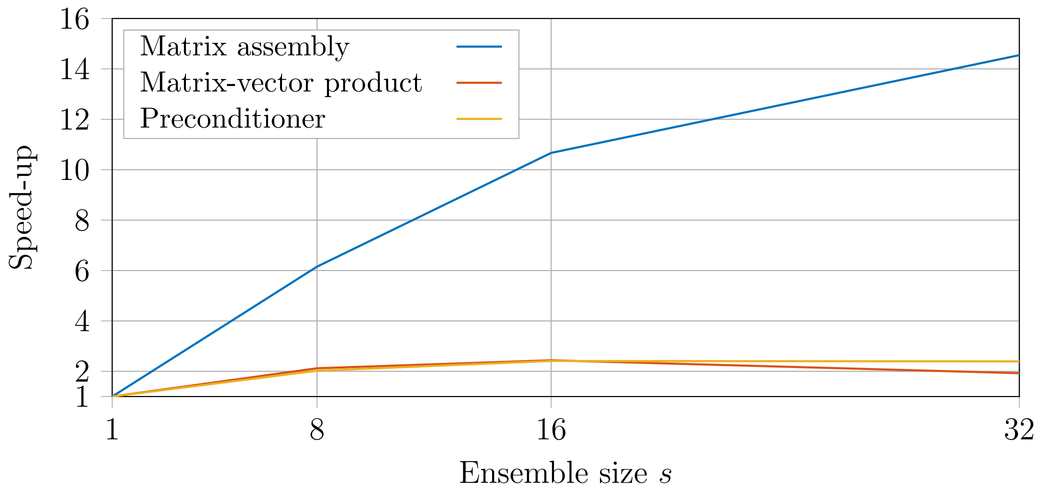


Figure 7.27: Speed-up of the matrix assembly (II in Fig. 6.8a), sparse matrix-vector product (IV.IV in Fig. 6.8b), and the preconditioner (IV.III in Fig. 6.8b).

The speed-up of the sparse matrix-vector product and the preconditioner are similar to measured speed-up in [Phipps et al., 2017] on Haswell CPUs for the conjugated gradient for problems with symmetric positive definite matrices.

GMRES iteration count

For the impact of ensemble propagation on the convergence of GMRES, we list the number of iterations to reach the convergence with a relative tolerance of 10^{-7} for the six test cases with and without reduction for the different ensemble sizes in Fig. 7.28 and Fig. 7.29 respectively. The samples are grouped using a natural ordering based on the order in which they are generated: the first s generated samples are grouped in the first ensemble. In other words, there is no particular effort to group the samples to reduce ensemble divergence in this case; this favors ensemble GMRES without ensemble reduction compared to ensemble GMRES with ensemble reduction.

We see from those results that when ensemble reduction is used, the samples are coupled and, therefore, the number of iterations required depends on the variability of the samples inside the given ensemble. Increasing the coefficient of variation increases the number of iterations to converge. This is due to the fact that an increased coefficient of variation leads to an increased variation of the spectra which are coupled by ensemble reduction. In particular, for the test case 6, the number of iterations to converge for $s = 32$ is multiplied by 2.3 compared to the largest number of iterations of one sample to converge.

However, when ensemble reduction is not used, ensemble GMRES waits for the slowest sample of a given ensemble to converge before stopping the update of this ensemble as expected. The variability of the samples inside a given ensemble has no other influence on the convergence without reduction.

Those results illustrate the accelerated rate of convergence of ensemble GMRES without reduction compared to ensemble GMRES with reduction. The acceleration is particularly significant for the test case 6 where the first ensemble of size 32 converges in 258 iterations with reduction and 131 without reduction.

The relative increase in computational work R, defined in [Phipps et al., 2017] as

$$R = \frac{s \sum_{e=0}^{N_s} J^{(e)}}{\sum_{\ell=0}^{N} j^{(\ell)}}, \tag{7.11}$$

where $J^{(e)}$ is the number of iterations to reach convergence of GMRES for the ensemble e, N_s is the number of ensembles of size s, and $j^{(\ell)}$ the number of iterations to reach convergence of GMRES for the sample ℓ alone, is another way to highlight the impact of ensemble propagation on the convergence of GMRES.

The relative increase in computational work over the ensembles of fixed size is shown in Fig. 7.30 for the six cases with and without ensemble reduction. From those results, we observe, as expected, larger relative increases in computational work for ensemble with reduction as more iterations are needed to converge. As ensemble GMRES without reduction waits for the slowest sample to converge, R is close to 1 which will lead to improved speed-up. Moreover, we can observe that increasing the coefficient of variation increases R when using ensemble reduction but has a lower influence on R without reduction. In other words, the variability of the samples has a lower influence on the convergence of ensemble GMRES without reduction.

Total speed-up

The total speed-up is determined by both the speed-up of each computational part and by the relative increase in computational work. The results of the total speed-up are listed in Fig. 7.31.

Ensemble GMRES with ensemble reduction avoids ensemble divergence and allows to use optimized BLAS implementations such as MKL but increases the number of iterations to converge which leads to the speed-up shown in Fig. 7.31. On the other hand, ensemble GMRES without reduction avoids the coupling of the spectra leading to an improved speed-up providing that occurrences of ensemble divergence are tackled and high-performing ensemble GEMV is implemented as discussed in this work.

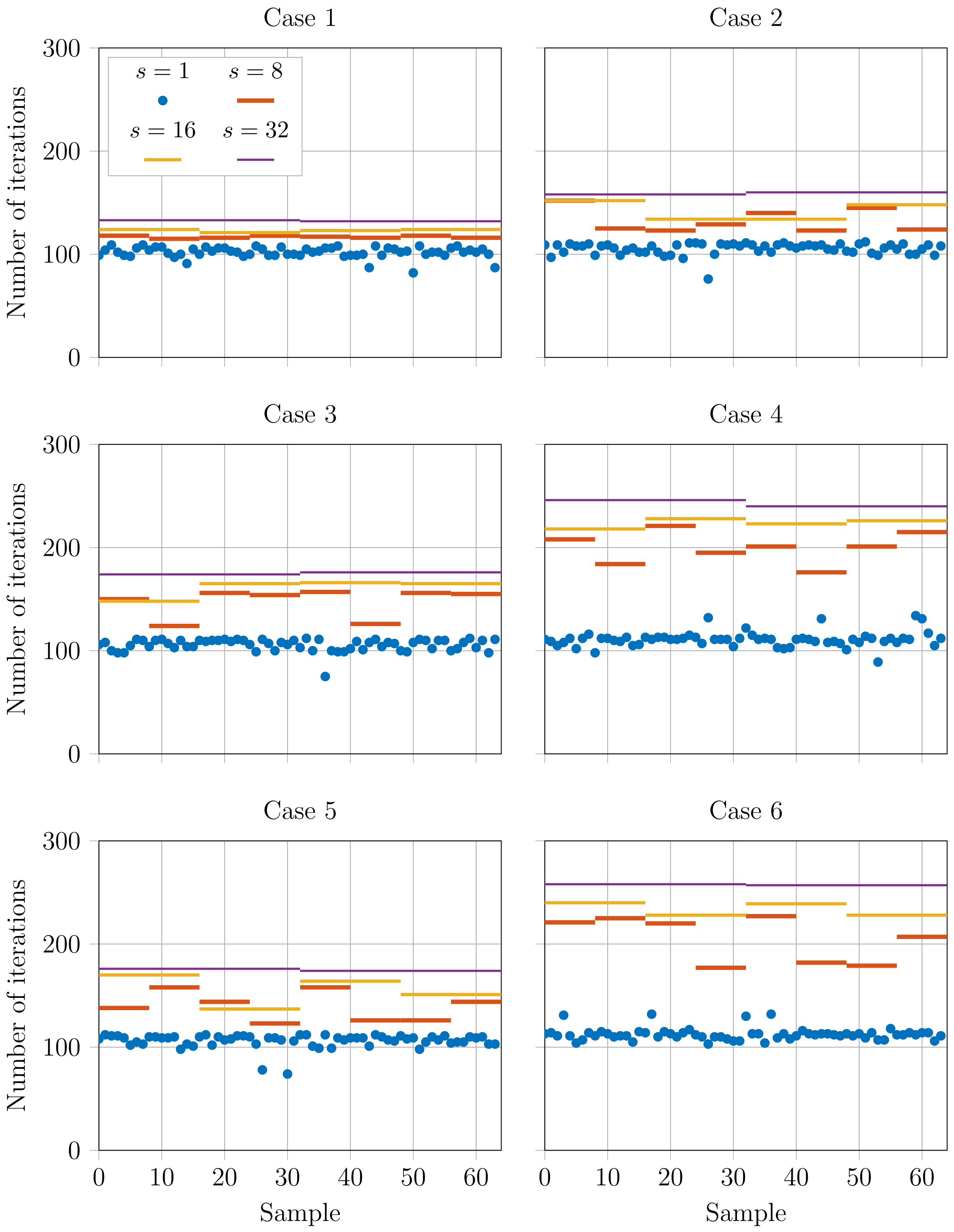


Figure 7.28: Number of iterations to converge with ensemble reduction for the 6 cases. The blue dots represent the required number of iterations of GMRES to converge for each sample propagated alone. The orange lines, yellow lines, and purple lines represent the required number of iterations of ensemble GMRES with ensemble reduction to converge with ensemble of size 8, 16, and 32 respectively. We observe that using ensemble GMRES with ensemble reduction requires more iterations than propagating the samples alone. This effect is particularly visible for the cases 4 and 6.

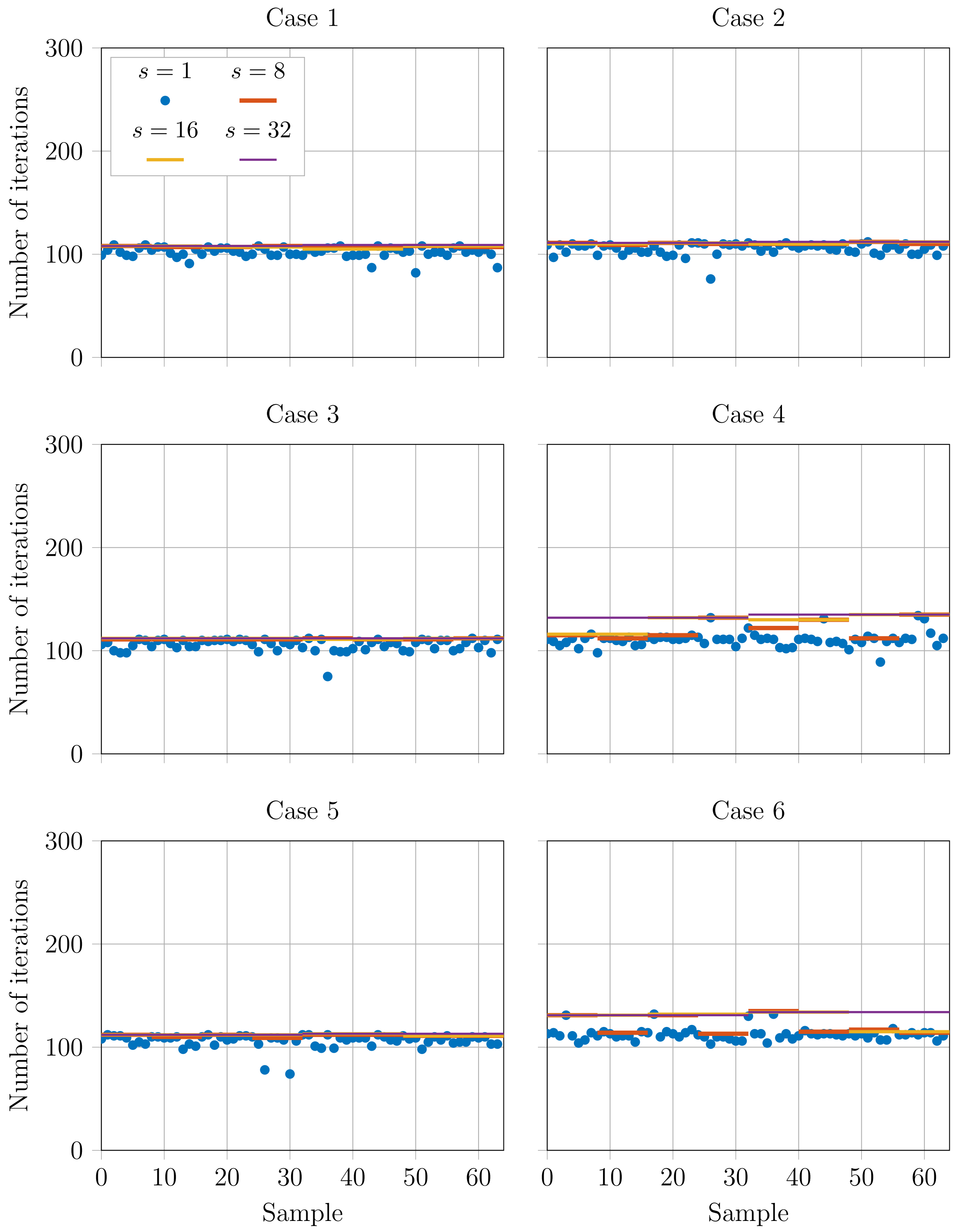


Figure 7.29: Number of iterations to converge without ensemble reduction for the 6 cases. The blue dots represent the required number of iterations of GMRES to converge for each sample propagated alone. The orange lines, yellow lines, and purple lines represent the required number of iterations of ensemble GMRES without ensemble reduction to converge with ensemble of size 8, 16, and 32 respectively. We observe that using ensemble GMRES without ensemble reduction requires to wait for the slowest sample to converge.

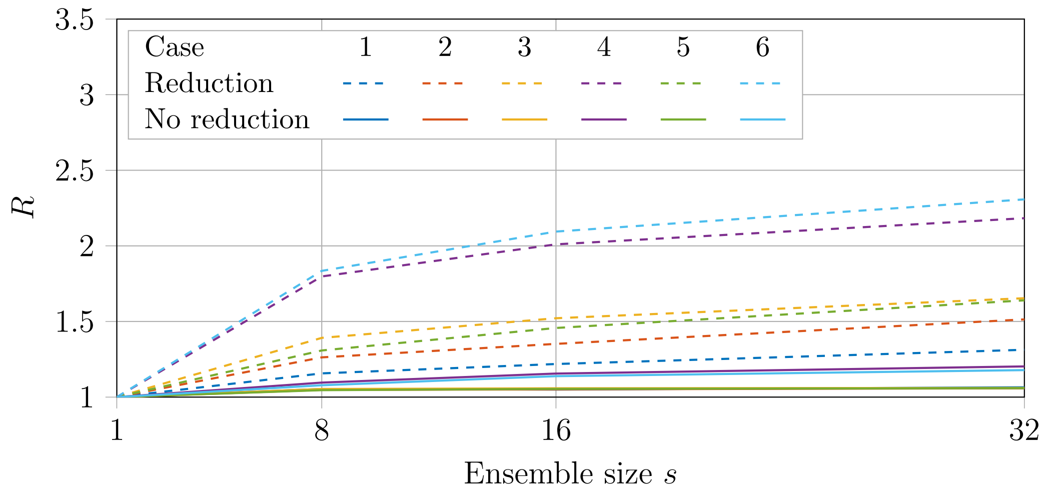


Figure 7.30: Relative increase in computational work for the six cases with and without reduction.

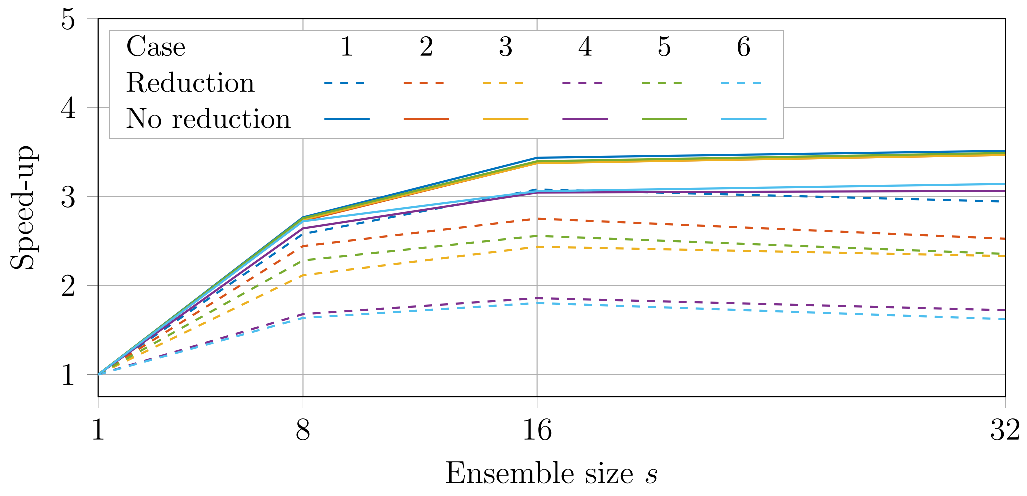


Figure 7.31: Speed-up of the full simulation for the six cases with and without reduction.

Finally, the total speed-up without reduction is equal to 3.5 for the test case 1, 2, 3, and 5 and equal to 3 for the test case 4 and 6 for which the relative increase R is larger. This speed-up is similar to the results of D'Elia et al. [2020] with the CG with ensemble propagation keeping in mind that no grouping strategy is used in this work.

Although grouping strategies are beyond the scope of this thesis, the effect of such strategies can be deduced from the results presented. The goal of a grouping strategy is to group similar samples in the same ensemble. Depending on the grouping strategy used, the meaning of similar samples is different. In the presented example, we have illustrated the effect of the coefficient of variation δ. This coefficient controls the deviation of the samples from the mean case. Therefore, reducing this coefficient reduces the difference between samples of the same ensemble and simulates the effect of a grouping strategy. As a consequence, the comparison between results with $\delta = 0.3$ and with $\delta = 0.1$ shows that the use of a grouping strategy reduces the relative increase in computational work R for both GMRES with and without ensemble reduction. This improves the speed-up for both ensemble GMRES but has a larger impact on the speed-up of ensemble GMRES with ensemble reduction.

	Tag		With reduction			Without reduction		
Ensemble size		1	8	16	32	8	16	32
Matrix assembly								
$\boldsymbol{K}$ assembly	II.I	0.153	0.360	0.496	1.152	0.448	0.544	1.216
$\boldsymbol{G}$ assembly	II.II	0.900	0.904	0.896	0.992	0.984	1.008	0.992
Total	II	1.053	1.264	1.392	2.144	1.432	1.552	2.208
Preconditioner setup								
Total	III	0.232	0.264	0.288	0.352	0.264	0.304	0.352
GMRES								
Orthogonalization	IV.X	0.110	1.760	4.448	10.848	0.600	1.328	2.848
Matrix-vector product	IV.IV	0.115	0.784	1.536	4.320	0.448	0.848	2.240
Preconditioner								
Block $\boldsymbol{K}$ smoother		0.303	2.176	4.384	9.376	1.288	2.432	4.832
Block $\boldsymbol{R}$ smoother		0.385	2.744	4.256	9.280	1.872	2.816	5.760
Restriction	IV.III.IV	0.033	0.072	0.096	0.128	0.040	0.048	0.064
Prolongation	IV.III.V	0.042	0.152	0.272	0.512	0.088	0.144	0.256
Residual computation	IV.III.III	0.132	0.792	1.568	3.360	0.456	0.864	1.728
Coarse solver	IV.III.I	0.246	2.656	5.776	12.448	1.704	3.456	7.072
Total	IV.III	1.558	10.471	19.707	43.528	6.599	11.708	24.120
Total	IV	1.828	13.264	26.048	59.168	7.784	14.064	29.440
Total		3.162	14.888	27.968	61.888	9.504	16.096	32.192

Table 7.2: Average wall-clock time in seconds per ensemble for the test case 6 where the tags refer to Fig. 6.8a, Fig. 6.8b, and Fig. 6.9.

	Tag		With reduction			Without reduction		
Ensemble size		1	8	16	32	8	16	32
Matrix assembly								
$\boldsymbol{K}$ assembly	II.I	1.	3.400	4.935	4.250	2.732	4.500	4.026
$\boldsymbol{G}$ assembly	II.II	1.	7.965	16.071	29.032	7.317	14.286	29.032
Total	II	1.	6.665	12.103	15.716	5.883	10.856	15.261
Preconditioner setup								
Total	III	1.	7.030	12.889	21.091	7.030	12.211	21.091
GMRES								
Orthogonalization	IV.X	1.	0.500	0.396	0.324	1.467	1.325	1.236
Matrix-vector product	IV.IV	1.	1.173	1.198	0.852	2.056	2.170	1.643
Preconditioner								
Block $\boldsymbol{K}$ smoother		1.	1.114	1.106	1.034	1.882	1.993	2.007
Block $\boldsymbol{R}$ smoother		1.	1.122	1.447	1.328	1.645	2.188	2.139
Restriction	IV.III.IV	1.	3.667	5.500	8.250	6.600	11.000	16.500
Prolongation	IV.III.V	1.	2.211	2.471	2.625	3.818	4.667	5.250
Residual computation	IV.III.III	1.	1.333	1.347	1.257	2.318	2.444	2.444
Coarse solver	IV.III.I	1.	0.741	0.681	0.632	1.155	1.139	1.113
Total	IV.III	1.	1.175	1.249	1.131	1.888	2.129	2.066
Total	IV	1.	1.103	1.123	0.989	1.879	2.080	1.987
Total		1.	1.699	1.809	1.635	2.661	3.143	3.143

Table 7.3: Speed-up for the test case 6 where the tags refer to Fig. 6.8a, Fig. 6.8b, and Fig. 6.9.

The wall-clock times of the test case 6 are listed in Table 7.2 and the corresponding speed-up in Table 7.3 to improve the understanding of the speed-up. It can be deduced from those measures that, as already said, the matrix assembly process and the preconditioner setup are independent of using ensemble reduction. Moreover, we observe a large speed-up for the matrix assembly process due to the large reuse of the computation of the Mortar matrix. The speed-up of the preconditioner setup is interesting too and is due to the amortized computation of the aggregates and the prolongation and restriction operators. In ensemble GMRES with reduction, the relative wall-clock time spent in the orthogonalization increases as more iterations are needed to converge and as the time complexity of the GEMV depends more than linearly on the Krylov subspace dimension. Concerning the preconditioner application, although the restriction and prolongation operations and the residual computation have good speed-up, their impact on the speed-up of applying the preconditioner is reduced by their relatively small wall-clock time compared to the smoothers.

Uncertainty quantification study

Finally, we tested ensemble GMRES without reduction on 6400 samples to compute the probability density function of the equivalent von Mises stress at the point p_3 of Fig. 7.19, the point on the hole boundary where the stress is theoretically maximal, for the test case 6. To highlight the fact that the results of the uncertainty quantification are not modified by the use of ensemble propagation, we have computed a probability density function of the quantity of interest with data propagated with and without ensemble propagation. The computed probability density functions are shown in Fig. 7.32 and are identical with and without ensemble propagation.

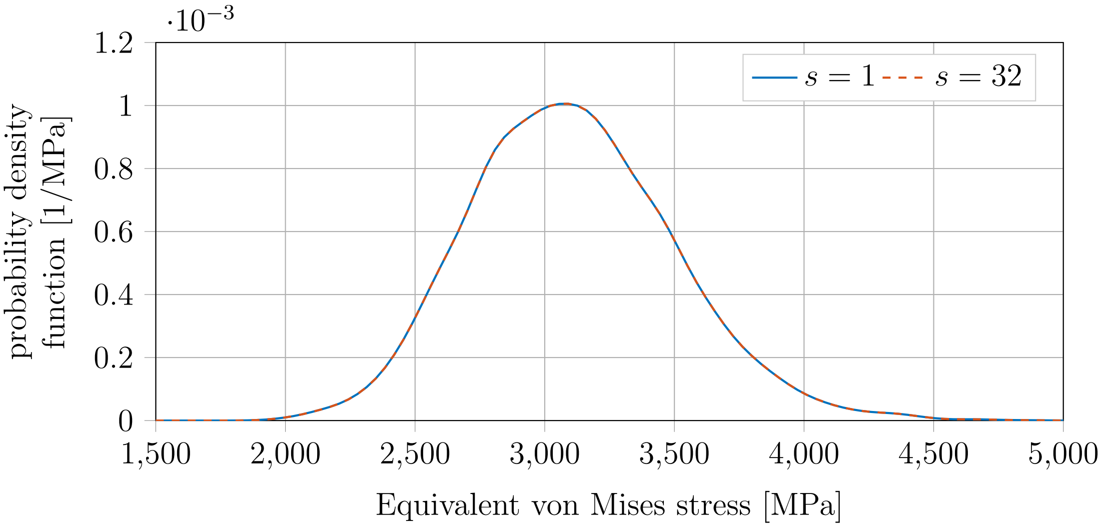


Figure 7.32: Probability density function of the quantity of interest: the Equivalent von Mises stress at the point p_3 of Fig. 7.19, computed without ensemble propagation ($s = 1$) and with ensemble propagation with ensembles of size 32 for the test case 6.

Fig. 7.33 illustrates the cumulative speed-up for the 200 ensembles of size 32 and the convergence of the speed-up to 3.1352 on 1 NUMA region of the Blake system described in Appendix A with hyper-threading, i.e. with 48 threads. Without using ensemble propagation, the evaluation of the 6400 samples took 5.66 hours of wall-clock time; this wall-clock time has been reduced to 1.8 hours using ensemble propagation with ensembles of size 32. The measured speed-up over the 6400 samples is 3.1352 which is consistent with results of Fig. 7.31.

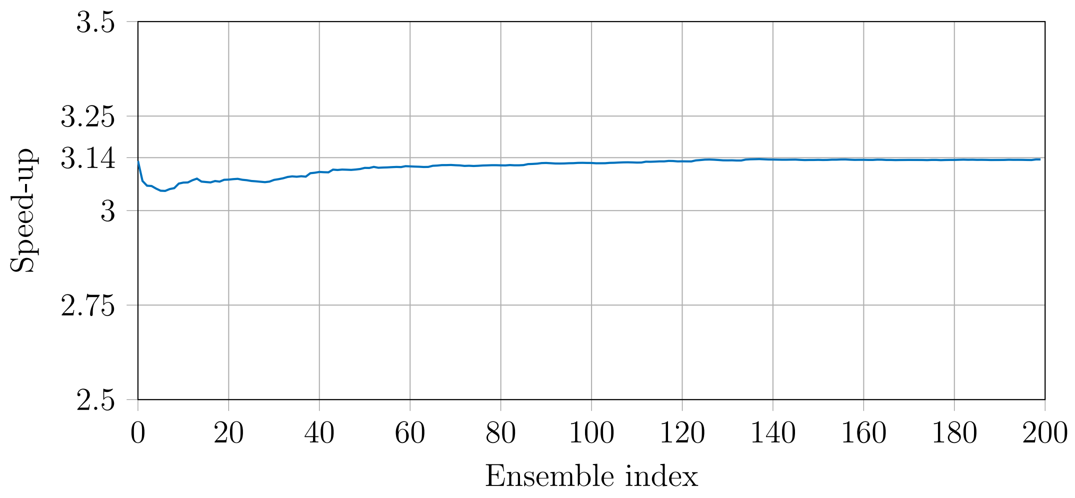


Figure 7.33: Cumulated speed-up of the full simulation for the test case 6 without ensemble reduction and ensemble of size 32.

7.4 Beam contact problem

The example that we propose to study with contact and ensemble propagation is a uniformly loaded beam fixed at both ends above a rigid surface as illustrated in Fig. 7.34. The beam has a length $2\,L$, a height H, and a width W. The material of the beam is linear elastic and has a Young's modulus E and a Poisson ratio ν. A pressure p is applied on the top of the beam and the rigid surface is located at a distance d below the bottom surface of the beam. Due to the symmetry of the problem, we can restrict ourselves to the computation of one half of the beam as shown in Fig. 7.35 where the green surface represents the surface where symmetry constraints are imposed. We do not take into account the symmetry along the $\mathbf{e}_y$ direction in order to simplify the discretization of the Lagrange multipliers as discussed in section 7.4.3. This problem is partially treated in [Kikuchi and Oden, 1988, p. 146] and named as the beam contact problem.

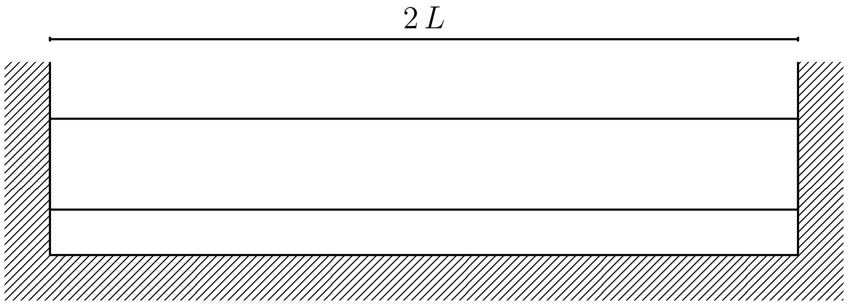


Figure 7.34: Beam fixed at both ends above a rigid surface.

The example has been run on 1 NUMA region of the Blake system described in Appendix A with hyper-threading.

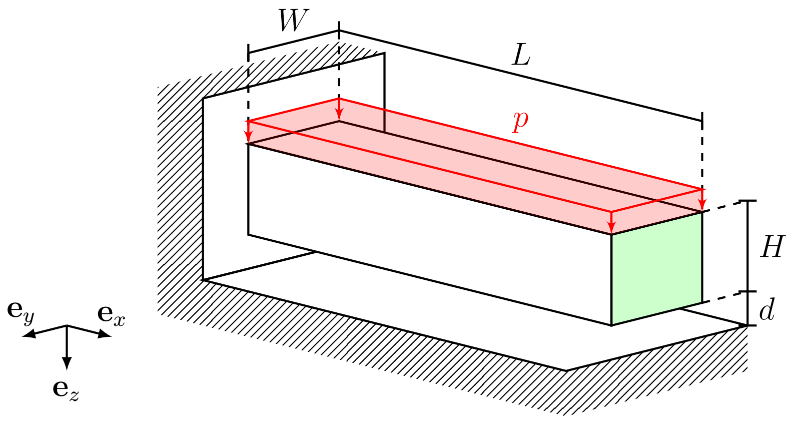


Figure 7.35: Geometrical configuration of one half of the uniformly loaded beam.

This example can have different states depending on whether the pressure is sufficient to make a partial contact between the beam and the ground, or not. In the next subsection, we will derive the analytical solutions using the beam theory. After that, in the second subsection, we will discuss the model and parameter values.

7.4.1 Theoretical results

The theory of this section is partially treated in [Kikuchi and Oden, 1988, p. 146] where they considered a cantilever beam fixed at one end and where the other end was free. The symmetry constrains considered in this example impact the boundary conditions and the solutions.

We start with the Euler-Bernoulli equation to compute the deflection w of the beam:

$$\frac{d^2}{dx^2}\left(E\,I\,\frac{d^2}{dx^2}w\right) = q(x), \tag{7.12}$$

where E is the Young's modulus, I is the area moment of inertia of the cross section, and $q(x)$ the external load per unit length applied at position x where the position x is shown in Fig. 7.36 where green color is used to show where the symmetry constrains are imposed.

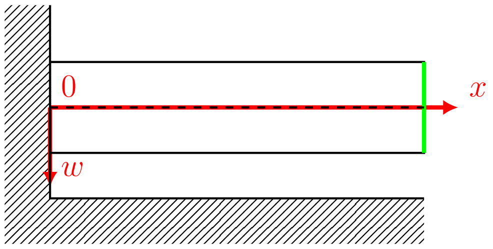


Figure 7.36: Coordinate system of the beam deflection problem.

As we have a rectangular cross section, we know that:

$$I = \frac{W\,H^3}{12}. \tag{7.13}$$

No contact state

In this case, we have the external load per unit length:

$$q(x) = p\,W, \tag{7.14}$$

and the following boundary conditions:

$$w(0) = 0, \tag{7.15}$$

$$\frac{d}{dx}w(0) = 0, \tag{7.16}$$

$$\frac{d}{dx}w(L) = 0, \tag{7.17}$$

$$\frac{d^3}{dx^3}w(L) = 0. \tag{7.18}$$

Therefore, we can solve (7.12) using (7.14) and the conditions (7.15), (7.16), (7.17), and (7.18):

$$w(x) = \frac{p\,W}{24\,E\,I}\left(x^4 - 4\,L\,x^3 + 4\,L^2\,x^2\right). \tag{7.19}$$

Using (7.19), we deduce that this solution is valid as long as:

$$\max_{x\in[0,L]} w(x) \leq d, \tag{7.20}$$

which is true as long as:

$$w(L) \leq d, \tag{7.21}$$

and therefore as long as $p \leq p_1$ where:

$$p_1 = \frac{24\,E\,I\,d}{W\,L^4}. \tag{7.22}$$

An example of deflection without contact is shown in Fig. 7.37.

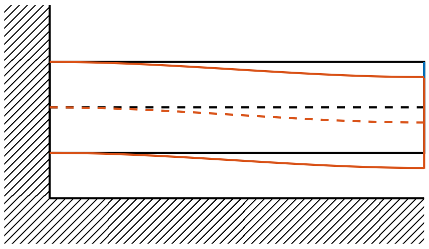


Figure 7.37: Deflection w of the uniformly loaded beam with no contact.

Partial contact state

First, we assume that there is an $a \in]0, L[$ such that the contact is closed for all $x \in [a, L]$ and open for any $x \in [0, a[$ as shown in Fig. 7.38.

This leads us to the following external load per unit length:

$$q(x) = \begin{cases} p\,W, & \text{if } x < a, \\ 0, & \text{if } x \geq a, \end{cases} \tag{7.23}$$

and the following boundary conditions:

$$w(0) = 0, \quad (7.24) \qquad \frac{d}{dx}w(0) = 0, \quad (7.25)$$

$$w(a) = d, \quad (7.26) \qquad \frac{d}{dx}w(a) = 0, \quad (7.27) \qquad \frac{d^2}{dx^2}w(a) = 0. \quad (7.28)$$

The fact that $q(x) = 0$ if $x \geq a$ is due to the reaction force of the rigid surface on the beam which cancels the applied pressure.

Therefore, we can solve (7.12) using (7.23) and the conditions (7.24), (7.25), (7.27), and (7.28):

$$w(x) = \begin{cases} \frac{p\,W}{24\,E\,I}\left(x^4 - \frac{8}{3}\,a\,x^3 + 2\,a^2\,x^2\right), & \text{if } x < a \\ d, & \text{if } x \geq a \end{cases}. \tag{7.29}$$

Using (7.29) and (7.26), we find that a has to be:

$$a = \sqrt[4]{72 \frac{E\,I\,d}{p\,W}}. \tag{7.30}$$

These assumptions are valid as long as $a \leq L$, and therefore, using (7.30), we deduce that $a \leq L$ is verified provided that $p \geq p_2$ where:

$$p_2 = \frac{72\,E\,I\,d}{W\,L^4}. \tag{7.31}$$

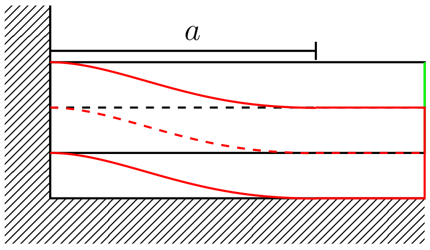


Figure 7.38: Deflection w of the uniformly loaded beam and definition of a.

7.4.2 Model and parameters

We will use a 3D finite element model with hexahedra as shown in Fig. 7.39 with $n_L \times n_W \times n_H$ hexahedra where n_L, n_W, and n_H stand respectively for the number of elements along the length, the width, and the height of the beam. The previous section on the beam theory will not be used to group samples in ensembles or to derive meshes, etc. but will be used to analyze the results.

Now, we will introduce numerical values based on [Kikuchi and Oden, 1988, p. 146] and define the random parameters and quantities of interest:

- $L = 50$ cm, $W = 1$ cm, $H = 5$ cm, $d = 1$ cm, and $\nu = 0.29$,
- $n_L = 60$, $n_W = 6$, and $n_H = 6$,
- p is random and varies uniformly in the range $[0.5\text{ kN/cm}^2, 2.5\text{ kN/cm}^2]$,
- E is random and varies uniformly in the range $[2.05\ 10^4\text{ kN/cm}^2, 2.15\ 10^4\text{ kN/cm}^2]$,
- The deflection of the point $(L, 0, H/2)$ is the first quantity of interest[2],
- The deflection of the point $(L/2, 0, H/2)$ is the second quantity of interest.

Based on the beam theory subsection, we deduce that such uncertainty ranges lead to each of the three states with a non-zero probability.

[2]The origin of the axes is the middle point of the clamped side as illustrated in Fig. 7.36.

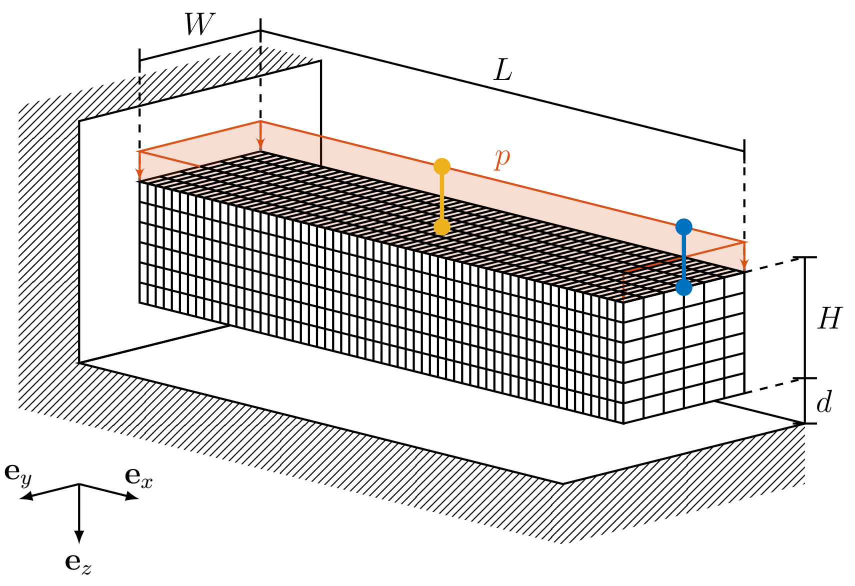


Figure 7.39: Mesh of the beam.

7.4.3 Discretization of the Lagrange multipliers

In this case, we can either use modified or unmodified shape functions as the contact will never be locally closed in the neighborhood of the Dirichlet boundary condition. Therefore, the Lagrange multipliers will be zero and the modification has no influence.

A possible mesh of the potential contact interface is shown in Fig. 7.40. In the case of unmodified shape functions, all the potential contact nodes have standard and dual shape functions as shown in Fig. 5.5 and Fig. 5.7 respectively. In the case of modified shape functions, all the black nodes of Fig. 7.40 have standard and dual shape functions as shown in Fig. 5.5 and Fig. 5.7 respectively, all the Dirichlet constrained nodes, the red nodes shown in Fig. 5.5 have no Lagrange multipliers, and the green nodes have modified shape functions as shown in Fig. 5.6 and Fig. 5.8 respectively.

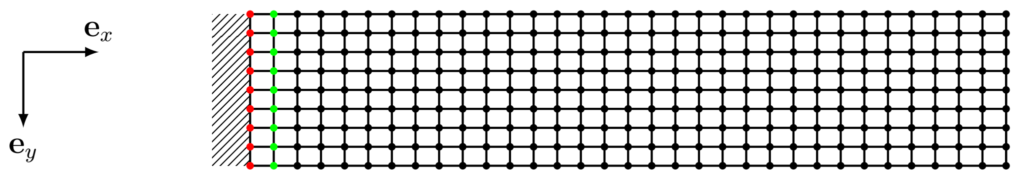


Figure 7.40: Mesh of the potential contact surface. The black dots represent the nodes for which the shape functions of the Lagrange multipliers do not need to be modified. The red dots represent the nodes of the potential contact surface where Dirichlet boundary conditions are applied. The green dots represent the nodes for which a modification of the shape function of the Lagrange multipliers can be applied.

7.4.4 Sampling strategy

In this example, we are using a Quasi-Monte Carlo strategy based on a Halton sequence of 640 samples. Those samples are illustrated in Fig. 7.41 where the dashed lines represent the boundary between the three states identified in section 7.4.1.

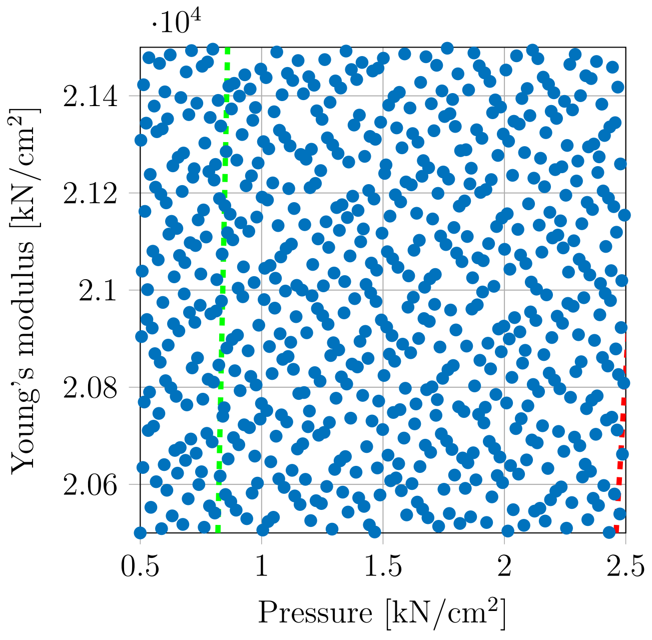


Figure 7.41: 640 samples of the beam contact problem, the dashed lines represent the theoretical pressure p_1 and p_2 as described in section 7.4.1.

7.4.5 Preconditioner and stopping criterion

We use, for this problem, the multigrid preconditioner as introduced for the problem of section 7.3 and a stopping criterion with relative tolerance of 10^{-8}.

7.4.6 Results

For this example, we consider both the use of ensemble propagation with and without grouping [D'Elia et al., 2018, 2020]. We first start with the results without grouping before discussing results with grouping.

Without grouping

First, we illustrate the quantities of interest of the evaluated samples in Fig. 7.42 and 7.43. We observe that those values are consistent with the beam theory which predicts p_1 in (7.22).

Afterwards, the samples have been evaluated using ensemble GMRES with and without ensemble reduction and ensemble sizes of 8, 16, and 32. As done for the previous numerical example, we investigate the effect of the ensemble sizes and ensemble reduction on the convergence by looking at the number of Krylov iterations to converge. Here, as the problem is now non-linear, we look at the sum of the number of iterations of all the linear solves performed during the active set strategy and illustrate them in Fig.7.44. Once again, we observe that ensemble reduction increases the number of iterations to converge compared to not using ensemble reduction. Concerning ensemble GMRES without reduction, we observe that the total number of iterations can be slightly larger than the maximal value of the total number of iterations among the samples of a given ensemble as illustrated in the zoomed part of Fig.7.44. This is explained by the fact that for each of the linear systems we wait for the slowest sample to converge; however, for differ-

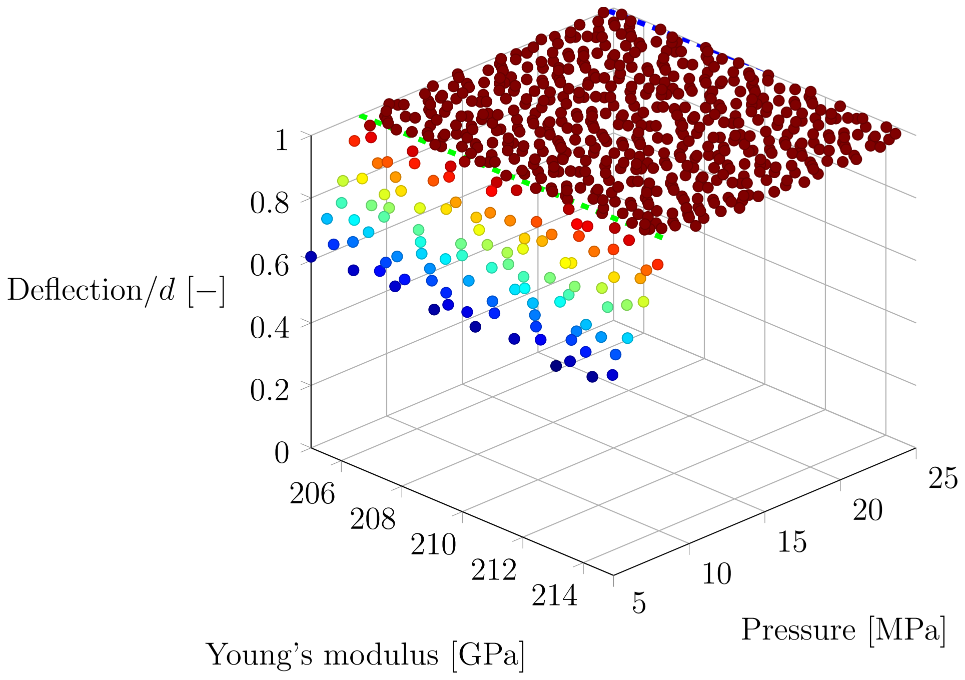


Figure 7.42: Deflection at point $(L, 0, H/2)$, the dashed lines represent the theoretical pressure p_1 and p_2 as described in section 7.4.1.

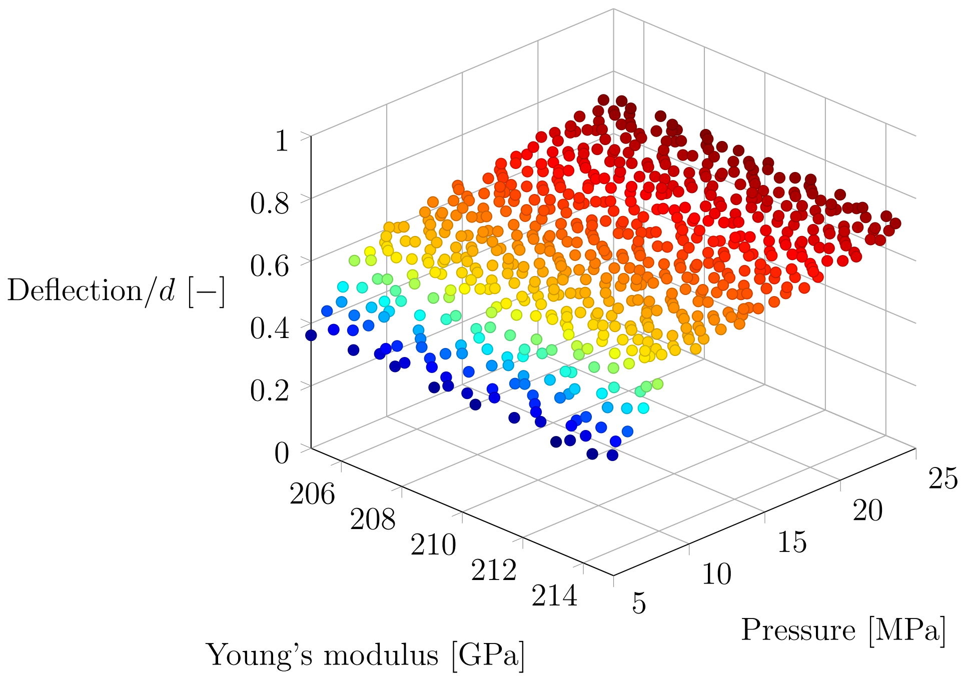


Figure 7.43: Deflection at point $(L/2, 0, H/2)$.

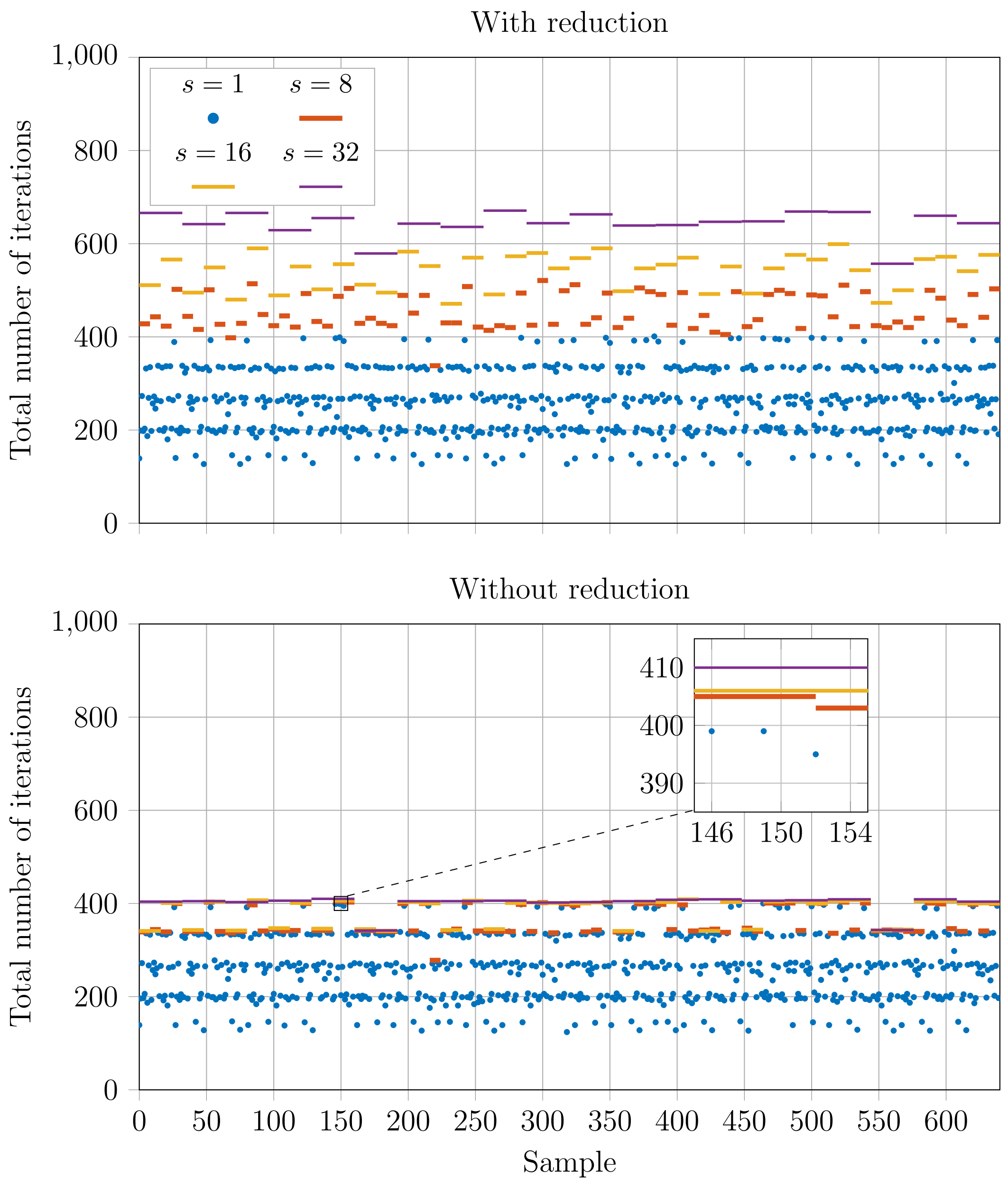


Figure 7.44: Total number of iterations to converge with and without ensemble reduction. The blue dots represent the required total number of iterations of GMRES to converge for each sample propagated alone. The orange lines, yellow lines, and purple lines represent the required total number of iterations of ensemble GMRES to converge with ensembles of size 8, 16, and 32 respectively. We observe that ensemble GMRES without ensemble reduction converges faster than ensemble GMRES with ensemble reduction for all ensembles.

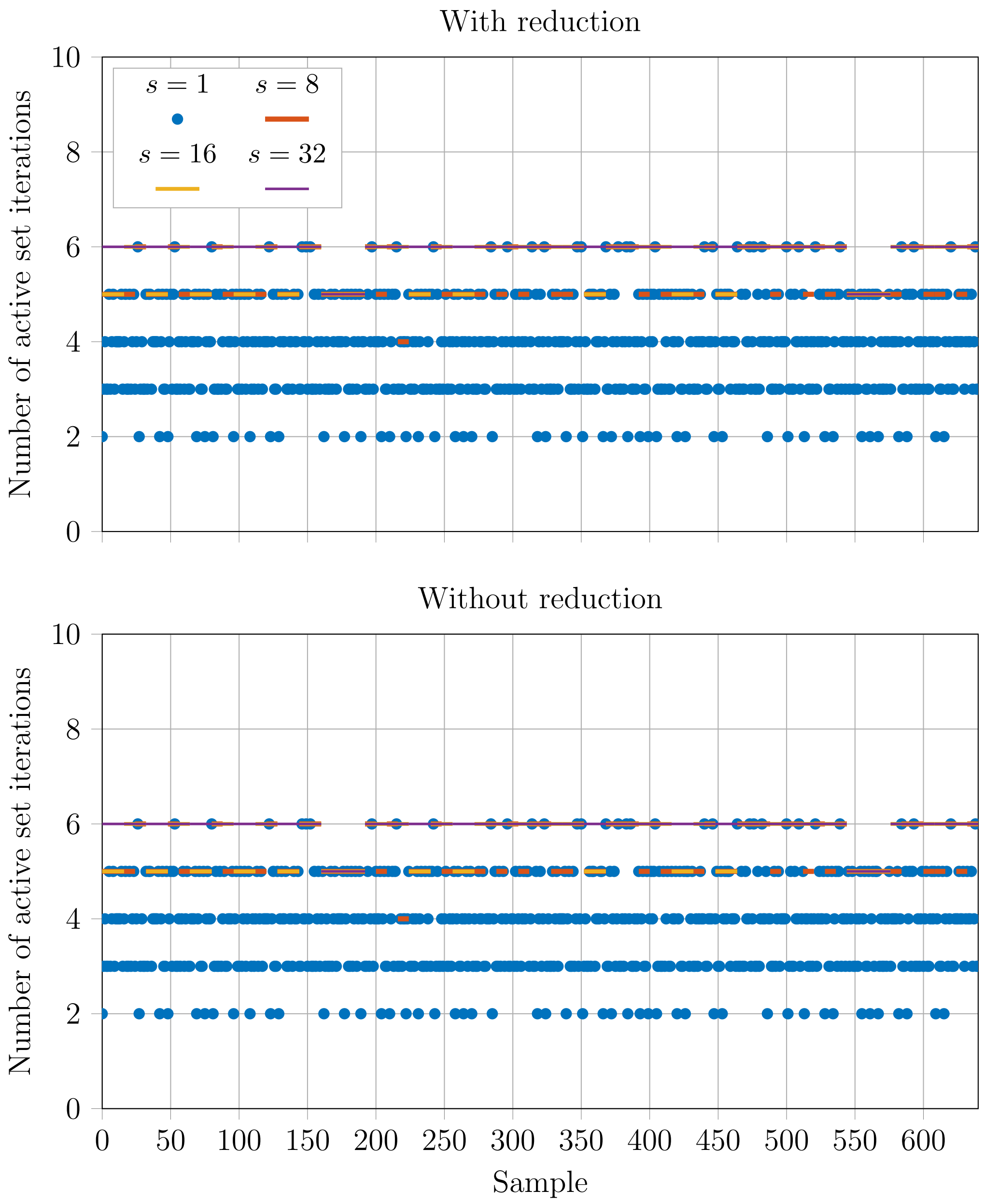


Figure 7.45: Number of active set iterations with and without ensemble reduction. The blue dots represent the required number of active set iterations to converge for each sample propagated alone. The orange lines, yellow lines, and purple lines represent the required number of active set iterations to converge with ensembles of size 8, 16, and 32 respectively. We observe that the required number of active set iterations is independent of the use of ensemble reduction for all ensembles.

ent consecutive linear systems of the active set iteration, the slowest sample is different impacting the total number of iterations.

Once again, the impact of ensemble reduction on the convergence influences the speed-up of the full simulation as illustrated in Fig. 7.46. If we now compare the reached speed-up with the ones of the mesh-tying example, we observe that the speed-ups of the beam contact problem are larger although the number of iterations is increased. This is explained by the fact that, although the total number of iterations has increased, the number of iterations per linear solve has decreased and by the fact that the number of degrees of freedom is smaller in this example. The last explanation relies on the results of section 7.2 which illustrated that the speed-up of the different parts of ensemble GMRES are usually larger for smaller problems.

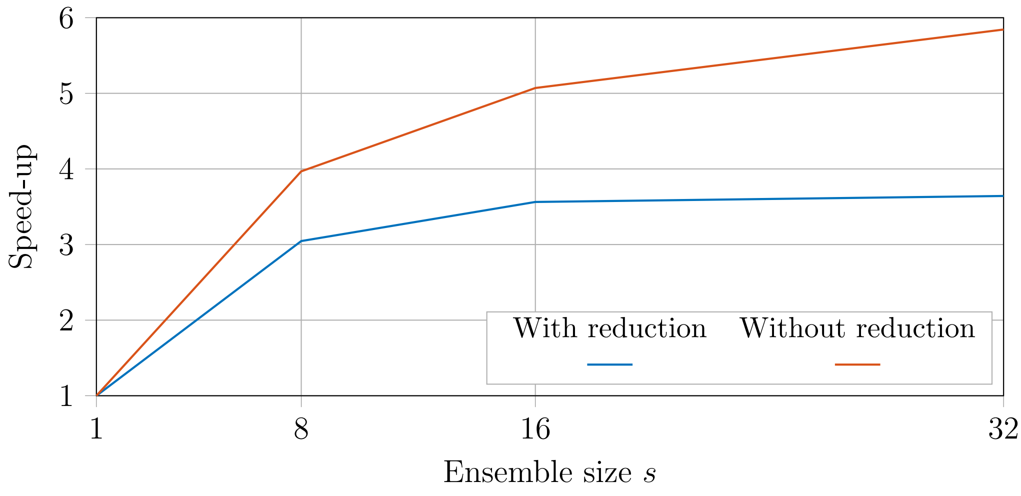


Figure 7.46: Speed-up of the full simulation with and without reduction.

The wall-clock time of the different parts of the simulation are listed in table 7.4 and their corresponding speed-ups are listed in table 7.5.

With grouping

What we can observe from the convergence of the samples in Fig. 7.44 is that, typically, five different numbers of total iterations are reached 140, 200, 280, 330, and 400. From such an observation, we can wonder what would be the impact of propagating samples that converge in 140 iterations together, etc. In other words, what is the impact of grouping the samples together based on their total number of iterations? Grouping strategies and their influences are outside the scope of this thesis, however, we think that it is interesting to illustrate its impact here in order to open the door to future research.

Therefore, we have indexed the 640 quasi-Monte Carlo samples in such a way that they are ordered such that their total number of iterations if propagated alone grows with the sample index. After that, we have propagated those samples again both with and without ensemble reduction. The convergence results are illustrated in Fig. 7.47.

From Fig. 7.47, we observe that both ensemble GMRES with and without reduction converge faster due to the grouping strategy. In particular, ensemble GMRES without reduction has nearly exactly the same behavior as samples propagated alone. This faster convergence for both approaches improves their speed-up as illustrated in Fig. 7.49 by

	Tag		With reduction			Without reduction		
Ensemble size		1	8	16	32	8	16	32
Matrix assembly								
$\boldsymbol{K}$ assembly	II.I	0.005	0.074	0.100	0.125	0.074	0.100	0.135
$\boldsymbol{G}$ assembly	II.II	0.197	0.200	0.200	0.200	0.200	0.200	0.200
Total	II	0.202	0.274	0.3	0.325	0.274	0.300	0.335
Preconditioner setup								
Total	III	0.496	0.747	0.823	0.917	0.509	0.556	0.633
GMRES								
Orthogonalization	IV.X	0.175	0.486	1.192	3.396	0.335	0.642	1.347
Matrix-vector product	IV.IV	0.173	0.718	1.254	2.472	0.556	0.867	1.565
Preconditioner	IV.III							
Block $\boldsymbol{K}$ smoother		0.925	1.716	3.266	7.654	1.234	2.264	4.732
Block $\boldsymbol{R}$ smoother		0.010	0.061	0.107	0.206	0.048	0.074	0.145
Restriction	IV.III.IV	0.069	0.154	0.191	0.249	0.118	0.130	0.161
Prolongation	IV.III.V	0.077	0.207	0.330	0.544	0.167	0.227	0.317
Residual computation	IV.III.III	0.199	0.768	1.294	2.501	0.597	0.899	1.591
Coarse solver	IV.III.I	0.112	0.797	1.597	3.398	0.658	1.198	2.311
Total	IV	2.097	5.972	10.198	20.297	4.637	7.247	13.110
Total		3.255	8.620	14.427	28.535	6.639	10.137	17.915

Table 7.4: Average wall-clock time in second per ensemble.

	Tag		With reduction			Without reduction		
Ensemble size		1	8	16	32	8	16	32
Matrix assembly								
$\boldsymbol{K}$ assembly	II.I	1.	0.541	0.800	1.280	0.541	0.800	1.185
$\boldsymbol{G}$ assembly	II.II	1.	7.880	15.760	31.520	7.880	15.760	31.520
Total	II	1.	5.898	10.773	19.889	5.898	10.773	19.296
Preconditioner setup								
Total	III	1.	5.312	9.643	17.309	7.796	14.273	25.074
GMRES								
Orthogonalization	IV.X	1.	2.880	2.349	1.649	4.179	4.361	4.157
Matrix-vector product	IV.IV	1.	1.928	2.207	2.239	2.489	3.193	3.537
Preconditioner	IV.III							
Block $\boldsymbol{K}$ smoother		1.	4.312	4.531	3.867	5.997	6.537	6.255
Block $\boldsymbol{R}$ smoother		1.	1.311	1.495	1.553	1.667	2.162	2.207
Restriction	IV.III.IV	1.	3.584	5.780	8.867	4.678	8.492	13.714
Prolongation	IV.III.V	1.	2.976	3.733	4.529	3.689	5.427	7.773
Residual computation	IV.III.III	1.	2.073	2.461	2.546	2.667	3.542	4.003
Coarse solver	IV.III.I	1.	1.124	1.122	1.055	1.362	1.496	1.551
Total	IV	1.	2.809	3.290	3.306	3.618	4.630	5.119
Total		1.	3.021	3.610	3.650	3.922	5.138	5.814

Table 7.5: Speed-up per ensemble.

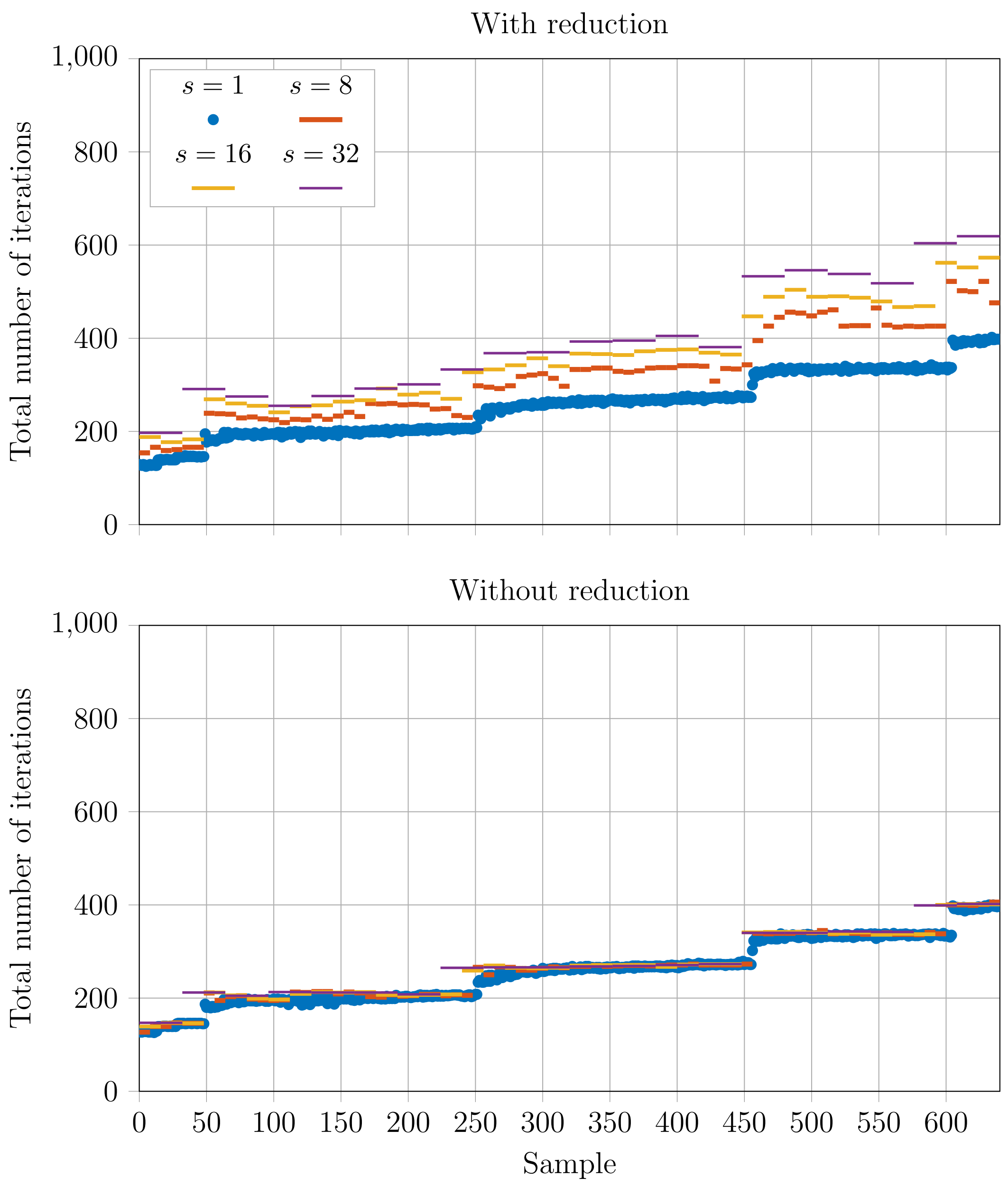


Figure 7.47: Total number of iterations to converge with and without ensemble reduction when grouping the samples. The blue dots represent the required total number of iterations of GMRES to converge for each sample propagated alone. The orange lines, yellow lines, and purple lines represent the required total number of iterations of ensemble GMRES to converge with ensembles of size 8, 16, and 32 respectively. As for the case without grouping, we observe that ensemble GMRES without ensemble reduction converges faster than ensemble GMRES with ensemble reduction for all ensembles.

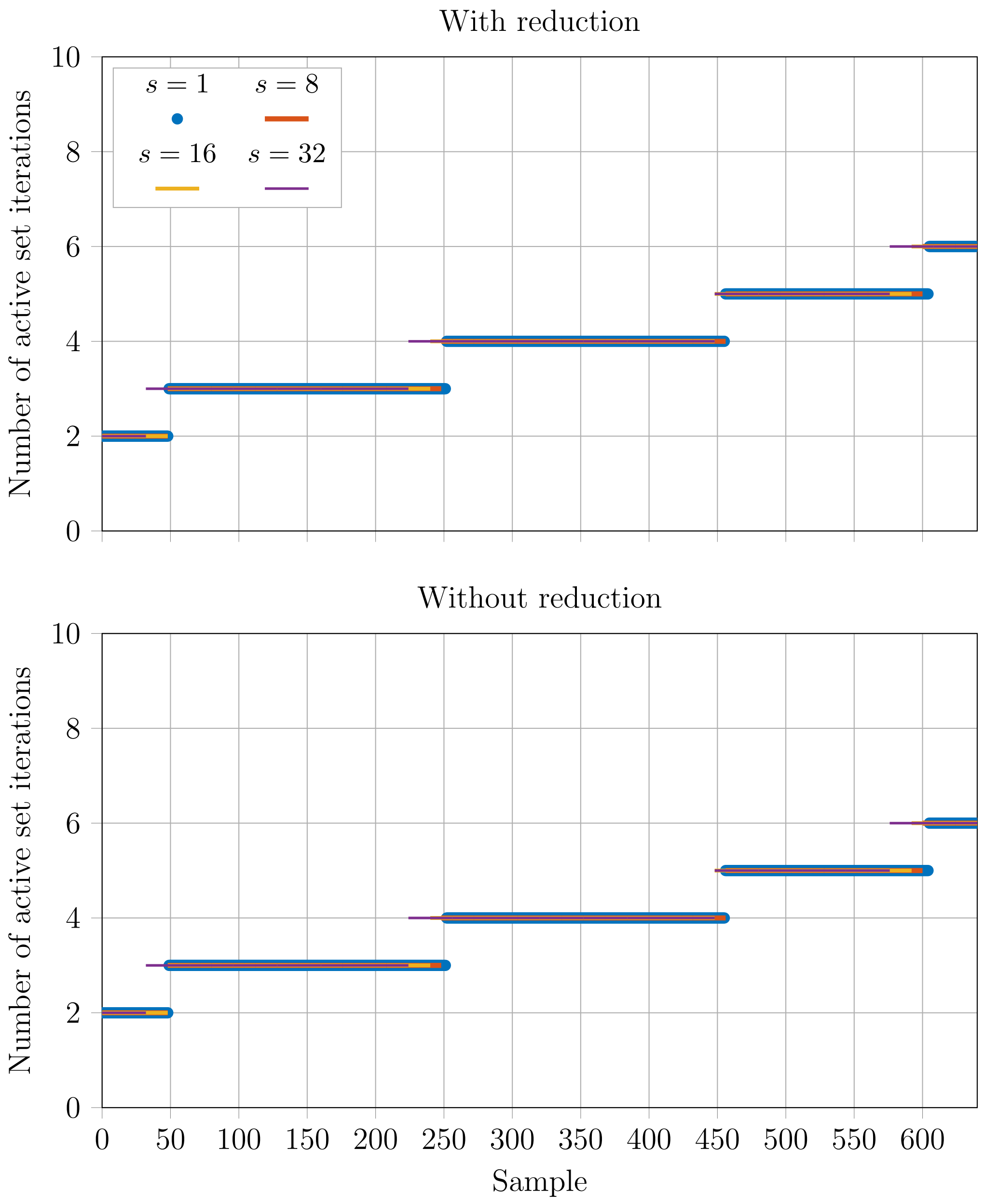


Figure 7.48: Number of active set iterations with and without ensemble reduction. The blue dots represent the required number of active set iterations to converge for each sample propagated alone. The orange lines, yellow lines, and purple lines represent the required number of active set iterations to converge with ensembles of size 8, 16, and 32 respectively. We observe once again that for any ensemble the required number of active set iterations is independent of the use of ensemble reduction. Moreover, we observe that the grouping strategy used has grouped samples with similar required number of active set iterations together.

reducing the wall-clock time of their computational parts as listed in table 7.6. The corresponding speed-ups are listed in table 7.7.

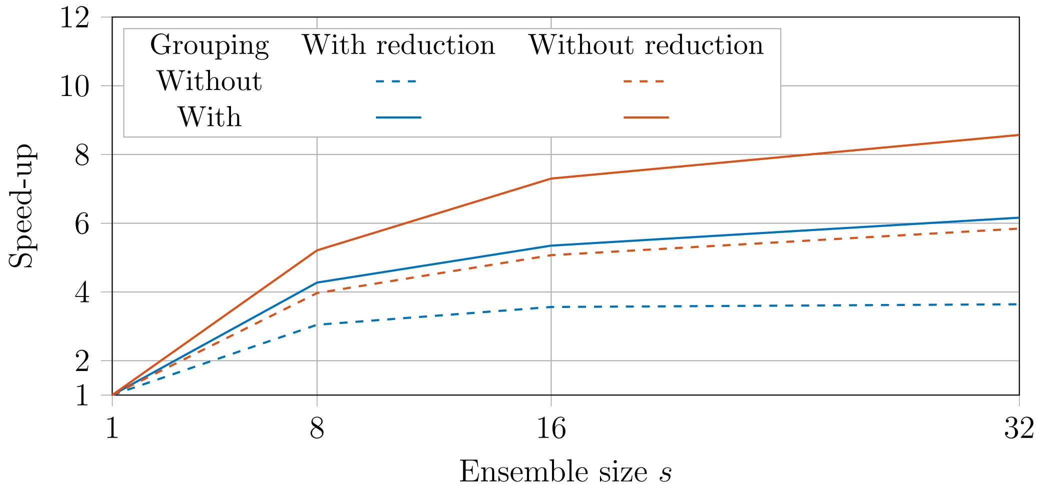


Figure 7.49: Speed-up of the full simulation with grouping with and without reduction.

As conclusion of this grouping subsection, we have observed that using a perfect grouping strategy, i.e. in the sense that using a grouping strategy which relies on information which are not available a priori, improves the speed-up of both approaches but that ensemble GMRES without reduction remains the fastest approach. D'Elia et al. [2020] proposed to build and use a surrogate model to predict the number of iterations required for a sample to converge. This surrogate allows one to group the samples with similar predicted number of iterations into an ensemble. The surrogate is built and trained with the number of iterations of the previously run samples and can be updated after each ensemble evaluation with the new information.

	Tag		With reduction			Without reduction		
Ensemble size		1	8	16	32	8	16	32
Matrix assembly								
$\boldsymbol{K}$ assembly	II.I	0.005	0.074	0.100	0.125	0.074	0.100	0.135
$\boldsymbol{G}$ assembly	II.II	0.197	0.200	0.200	0.200	0.200	0.200	0.200
Total	II	0.202	0.274	0.300	0.325	0.274	0.300	0.335
Preconditioner setup								
Total	III	0.496	0.747	0.823	0.917	0.509	0.556	0.633
GMRES								
Orthogonalization	IV.X	0.175	0.345	0.804	1.901	0.245	0.451	0.926
Matrix-vector product	IV.IV	0.173	0.499	0.829	1.498	0.405	0.605	1.064
Preconditioner								
Block $\boldsymbol{K}$ smoother		0.925	1.220	2.376	4.79	0.904	1.576	3.211
Block $\boldsymbol{R}$ smoother		0.010	0.042	0.072	0.128	0.036	0.053	0.099
Restriction	IV.III.IV	0.069	0.107	0.125	0.151	0.086	0.091	0.110
Prolongation	IV.III.V	0.077	0.144	0.217	0.332	0.122	0.159	0.215
Residual computation	IV.III.III	0.199	0.534	0.854	1.511	0.435	0.630	1.083
Coarse solver	IV.III.I	0.112	0.560	1.064	2.097	0.478	0.834	1.571
Total	IV	2.097	4.158	6.942	12.452	3.395	5.067	8.927
Total		3.255	6.128	9.962	17.57	5.061	7.286	12.435

Table 7.6: Average wall-clock time in second per ensemble after grouping.

	Tag		With reduction			Without reduction		
Ensemble size		1	8	16	32	8	16	32
Matrix assembly								
$\boldsymbol{K}$ assembly	II.I	1.	0.541	0.800	1.280	0.541	0.800	1.185
$\boldsymbol{G}$ assembly	II.II	1.	7.880	15.760	31.520	7.880	15.760	31.520
Total	II	1.	5.898	10.773	19.889	5.898	10.773	19.296
Preconditioner setup								
Total	III	1.	5.312	9.643	17.309	7.796	14.273	25.074
GMRES								
Orthogonalization	IV.X	1.	4.058	3.483	2.946	5.714	6.208	6.048
Matrix-vector product	IV.IV	1.	2.774	3.339	3.696	3.417	4.575	5.203
Preconditioner								
Block $\boldsymbol{K}$ smoother		1.	6.066	6.229	6.180	8.186	9.391	9.218
Block $\boldsymbol{R}$ smoother		1.	1.905	2.222	2.500	2.222	3.019	3.232
Restriction	IV.III.IV	1.	5.159	8.832	14.623	6.419	12.132	20.073
Prolongation	IV.III.V	1.	4.278	5.677	7.422	5.049	7.748	11.46
Residual computation	IV.III.III	1.	2.981	3.728	4.214	3.660	5.054	5.880
Coarse solver	IV.III.I	1.	1.600	1.684	1.709	1.874	2.149	2.281
Total	IV	1.	4.035	4.833	5.389	4.941	6.622	7.517
Total		1.	4.249	5.228	5.928	5.061	7.148	8.376

Table 7.7: Speed-up per ensemble after grouping.

7.5 Conclusions

To summarize, in this chapter, we have applied ensemble GMRES on four academic problems. The results illustrated that the number of iterations required for the convergence of ensemble GMRES with ensemble reduction is larger than the number of iterations required for the convergence of ensemble GMRES without ensemble reduction. We have brought insight into the speed-up of the different computational parts regardless of the use of ensemble reduction and we have illustrated that the orthogonalization process, for a fixed number of iterations, has a similar CPU cost with and without ensemble reduction. The consequence of those two observations is that the speed-up of ensemble GMRES without ensemble reduction is better than the speed-up of ensemble GMRES with ensemble reduction.

Finally, we have illustrated the speed-up of ensemble propagation on two uncertainty quantification studies with problems with indefinite matrices: a mesh-tying problem and a contact problem. We have illustrated, once again, that ensemble reduction deteriorates the speed-up of the parametric computation by increasing the number of iterations required for the convergence of ensemble GMRES and we have illustrated a first example of the impact of grouping strategy on ensemble GMRES.

Chapter 8

Application to a diagnostic mirror for ITER

The work presented in this chapter has been done in collaboration with the research group of Philippe Mertens from FZ Jülich, Germany. They are interested in the design of the front mirror of the Charge eXchange Recombination Spectroscopy (CXRS) diagnostic system of ITER as discussed in section 1.1.1.

In this chapter, we investigate the performance of embedded ensemble propagation to simulate the behavior of the CXRS front mirror [Mertens, 2018; Krasikov et al., 2015].

In particular, we evaluate the performance in the context of a model problem with uncertainty related to modifications of the material properties due to neutron irradiation.

The considered tests of this chapter are thermomechanical problems without contact. They are used to evaluate the performance of the code with a large number of degrees of freedom (about 10^7) and are the only tests of this thesis which use distributed-memory parallelism.

The developed code and one of the examples of this chapter have been used to generate a result included in [Mertens et al., 2019].

Finally, the model problem is used to perform an uncertainty quantification analysis to evaluate the temperature on the mirror surface, the deformation of the reflective surface, and the preload loss in the studs due to the thermal expansion using embedded ensemble propagation.

8.1 Description of the problem

8.1.1 Context

In the complex system which ITER is, as much information as possible on the state of the plasma is needed in order to tune correctly the different parameters of the magnetic confinement and of the heating of the plasma. Several diagnostics will be used to measure different physical quantities, such as the density and the temperature of the plasma.

This study addresses the core charge-exchange recombination spectroscopy (cCXRS) diagnostic which is planned to be located in one of the upper port plugs as illustrated in Fig. 8.1. This system uses the principle of active spectroscopy activated by hydrogen beams. The diagnostic neutral beam reduces by one the charge of ions in the hot plasma. These particles are excited: they have a state of higher energy than the ground state and decay by emitting visible radiation. If this radiation is measured, the density of specific impurities in the plasma or the ion temperature can be deduced. From the spectral intensities and widths, physical parameters like concentration and velocity of specific ions can be deduced.

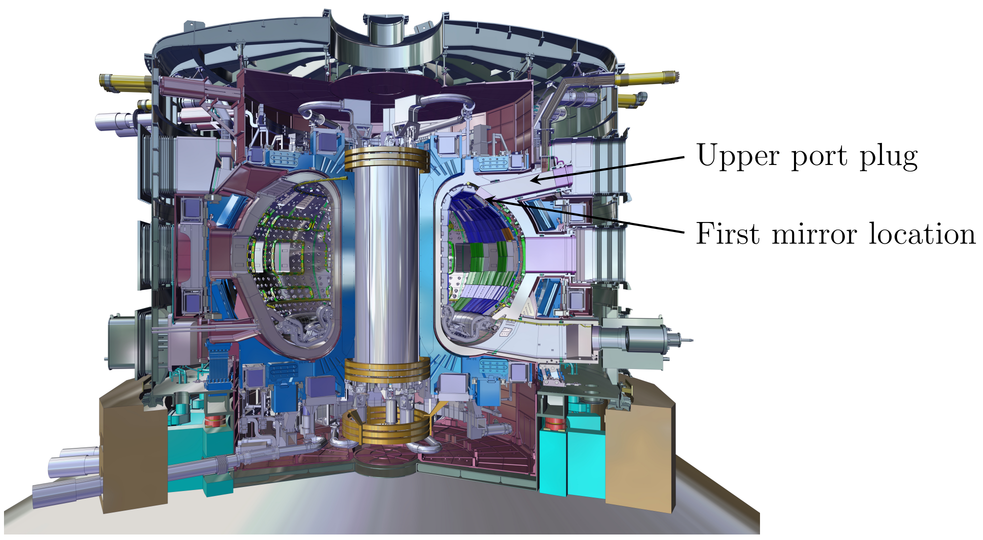


Figure 8.1: ITER, upper port plug, and first mirror location. Picture by courtesy of ITER Organization.

The system will be made of a chain of mirrors, the purpose of which is to transport the optical signal from the vessel to spectrometers outside the reactor. Then the spectrometers will decompose the optical signal according to the wavelength and a computer can compare the decomposed signal to the spectrum of known particles.

The mirror studied in this chapter is the first mirror of the optical chain and is the one located closest to the plasma. Being the closest to the plasma, this mirror is exposed to high radiation and fluxes of particles which escape the plasma and is the most vulnerable in-vessel optical component being subject to both erosion and deposition of impurities.

Due to this high radiation and these fluxes, the temperature of the first mirror increases which deforms the reflecting surface of the mirror and may lead to blurred images at the end of the optical chain. Moreover, due to the irradiation by the neutrons, material properties of some parts of the mirror assembly are uncertain.

In such a context, we set up a model problem representative of parametric computations useful to evaluate the robustness of the mirror with uncertainty related to irradiated material. This model problem is used to evaluate the performance of ensemble GMRES on a non-academic problem.

8.1.2 Geometry

The studied geometry has been supplied by FZ Jülich and is illustrated in Fig. 8.2 and Fig 8.3. In this work, we restrict ourselves to a sub-assembly illustrated in Fig. 8.4 of the full assembly of Fig. 8.2 except in section 8.5 where the full assembly is considered. The main components of the mirror assembly are:

- The mirror: The mirror is a thin plate with a flat optical surface the role of which is to reflect the incoming optical signal. It is not actively cooled; there are no cooling channels going directly through it. The mirror surface is 167 mm long and 84 mm wide.
- The mirror substrate: The mirror plate is bound to a second component: the mirror substrate which is machined with threads to be bolted to the holder.
- The holder: The holder plays the role of a heat sink for the mirror system. It has a water cooling channel passing through it and is cooled by forced convection.
- The studs and nuts: The mirror is bolted to the holder with three bolts, two M16 bolts and one M24 bolt. These bolts play a key structural role since they should ensure a good thermal contact between the heated mirror and the actively cooled holder. The stiffness and strength of these bolts should allow relatively high preloading in order to keep good thermal contact and prevent dynamic detachment of the bolted parts. The nuts have a second role, they must electrically isolate the mirror from the holder to prevent large eddy current loop across the parts and to enable the use of the mirror as cathode during mirror cleaning discharges.
- The spacers: spacers with lowest possible thermal resistivity are added between the holder and the mirror substrate to improve the heat transfer from the mirror to the cooling channel. Those spacers have a sufficiently high electrical resistance for the same reason as the nuts.
- The washers: finally, the washers are introduced to help the preloading of the studs during the assembly process.

The components of the assembly are illustrated in two cuts shown in Fig. 8.5 and Fig. 8.6. The cut illustrated of Fig. 8.5 which is illustrated again in Fig. 8.7 will be used in the results section.

8.1.3 Mesh

Although this assembly includes contact between parts, we have restricted ourselves to simulate the full assembly without taking into account the fact that contact can be decreased or lost between components. Due to this restriction, we have fused the meshes of the different components at their interface to have a conforming discretization at the interfaces. As opposed to the previously shown examples, this mesh is a full tetrahedron

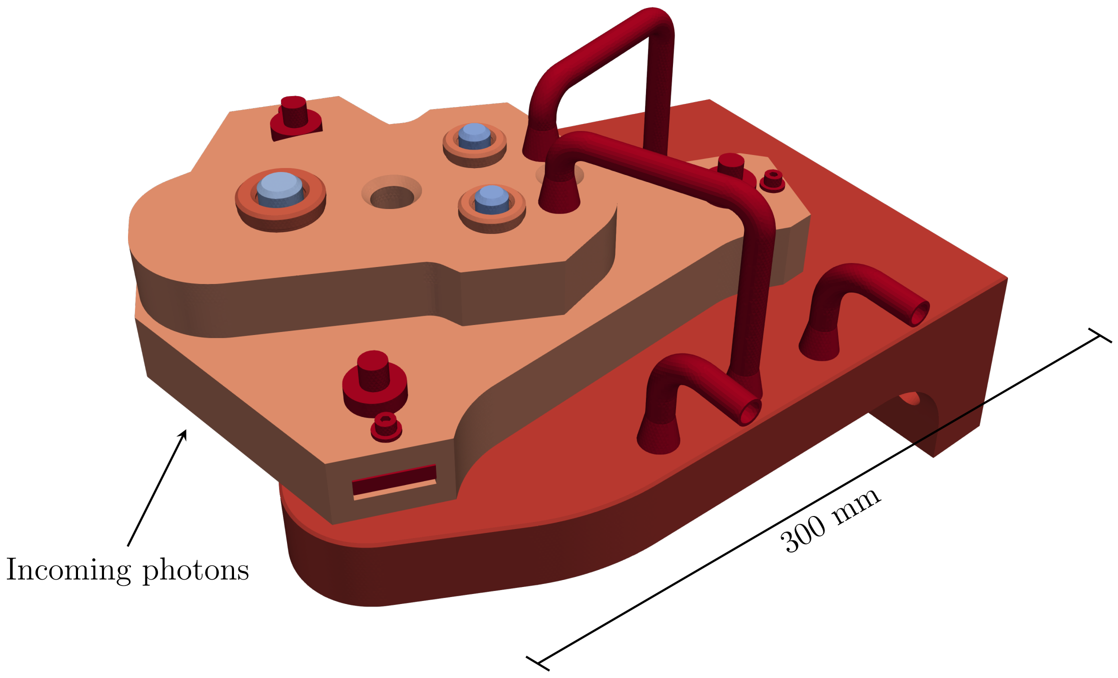


Figure 8.2: Front mirror assembly.

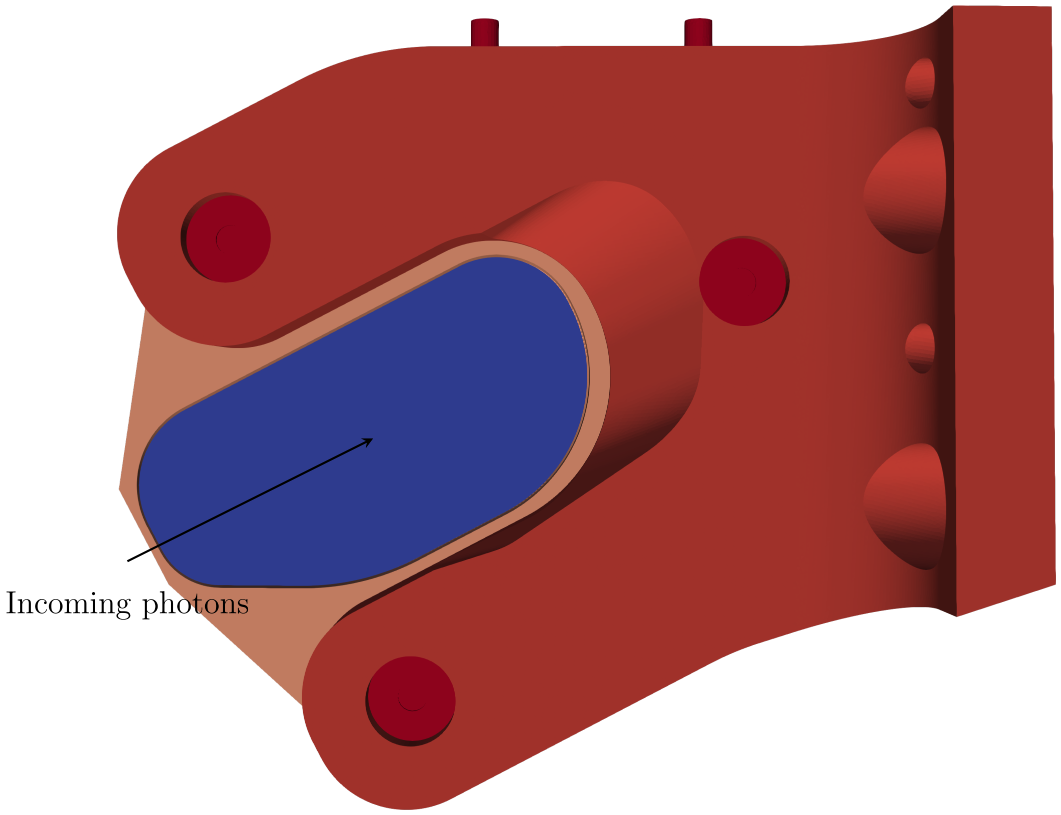


Figure 8.3: Front mirror assembly seen from the bottom. The reflective surface is represented in blue.

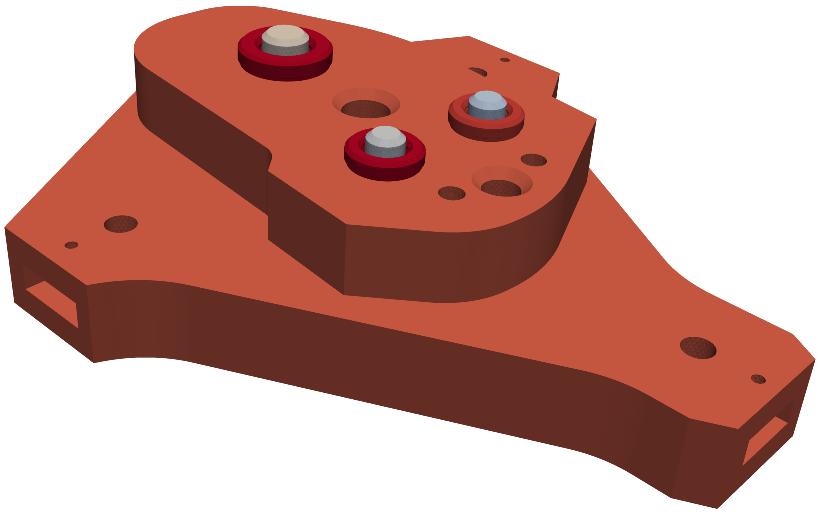


Figure 8.4: Sub-assembly of the front mirror which will be used for the numerical study.

mesh and is shown in Fig. 8.8. The characteristic length of the tetrahedra is 1.5 mm. We have meshed the different volumes in such a way that their common interfaces share the same nodes.

The main reason why we have not tested this model with contact constraints is that not all the components of the assembly are clamped, i.e. Dirichlet boundary conditions are not applied on all the components. This implies that the stiffness matrix is singular even if the full system is well posed. This singularity impacts the numeral strategy such as the preconditioner which merits deeper investigation.

8.2 Materials properties

The material properties of this section can be found in the ITER material database [Barabash, 2013] (Collection of material properties for material usually used in ITER design) or were given by FZ Jülich. Those values are assumed to not be temperature dependent in this work.

As explained in [Mertens et al., 2019], the mirror is made of rhodium, its parameters are listed in Table 8.1. The mirror substrate (the holder welded to the mirror) and the metallic spacers are in a tungsten-copper alloy (Table 8.2). Previously, an actively cooled holder in stainless steel (Table 8.3) was considered but the temperature field of the simulation was too high, therefore, in discussion with FZ Jülich, we have chosen to use CuCrZr (Table 8.5) for the actively cooled holder. The heat conducting spacers are in AlN ceramic (Table 8.4). The studs and the washers are in Inconel 718 (Table 8.6). In order to electrically isolate the mirror from the holder, both the nuts and the WCu spacers are isolated using a ceramic surface coating.

Young's Modulus	379 000 MPa
Poisson's coefficient	0.26
Thermal conductivity	0.150 kW/(m K)
Linear coefficient of thermal expansion	0.781 10^{-05} 1/K

Table 8.1: Rhodium (Rh).

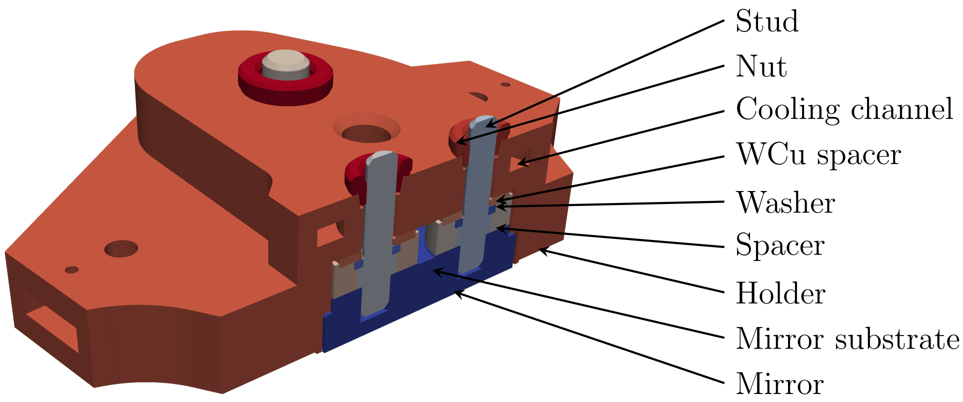


Figure 8.5: Cut 1 of the assembly with components.

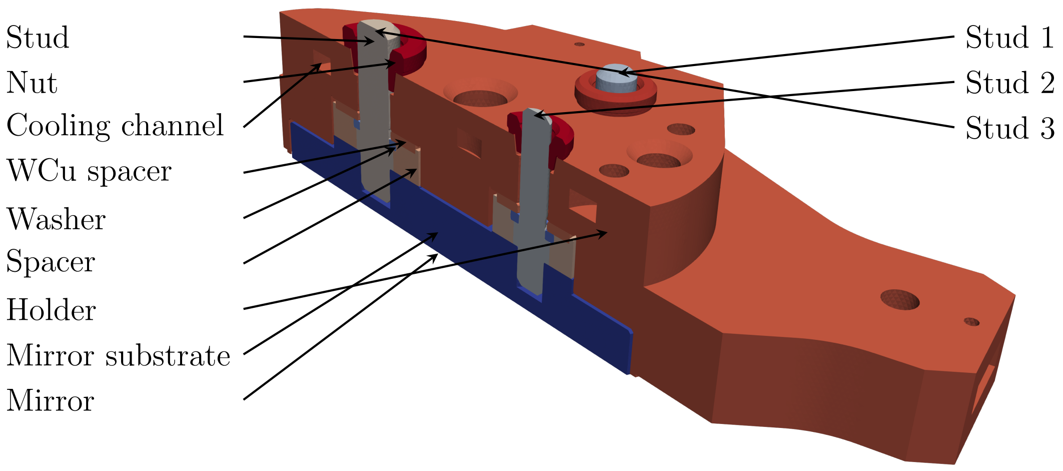


Figure 8.6: Cut 2 of the assembly with components and labeling of the studs.

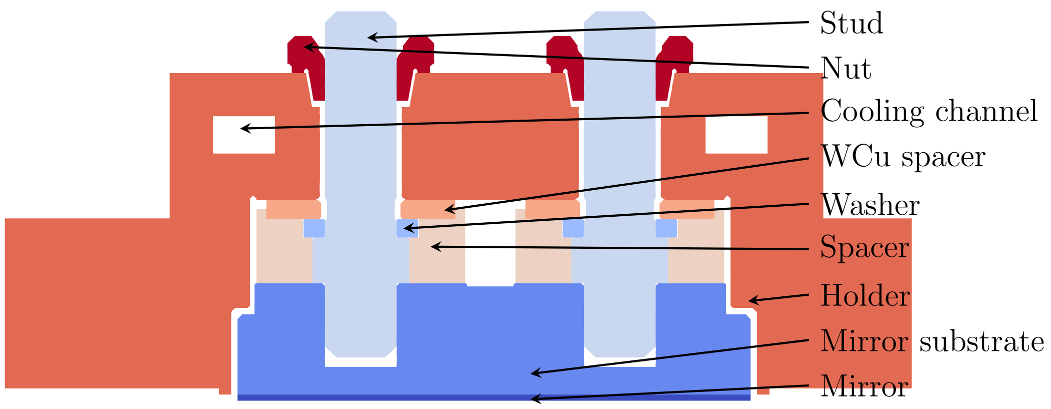


Figure 8.7: Cut 1 of the assembly with components.

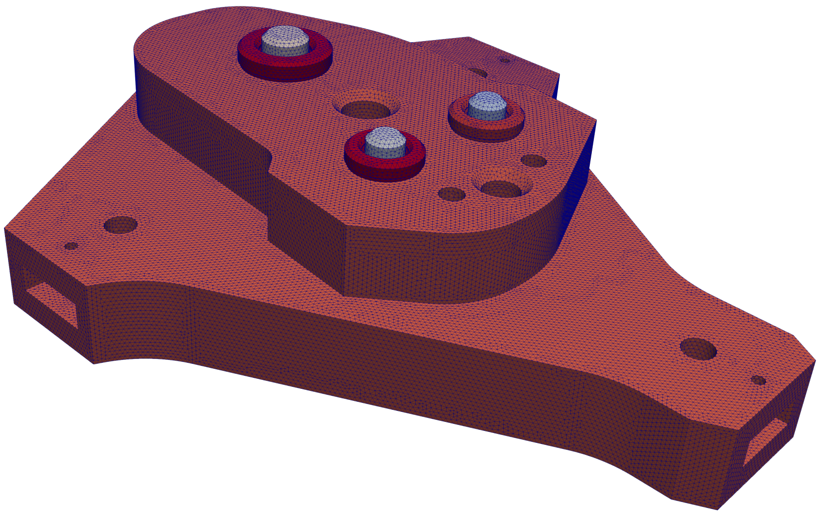


Figure 8.8: Mesh of the assembly with 329057 nodes and 1.7 million tetrahedra.

Young's Modulus	280 000 MPa
Poisson's coefficient	0.298
Thermal conductivity	0.180 kW/(m K)
Linear coefficient of thermal expansion	0.88 10^{-05} 1/K

Table 8.2: Tungsten-copper alloy W(80%)Cu(20%) (WCu).

Young's Modulus	200 000 MPa
Poisson's coefficient	0.3
Thermal conductivity	0.0153 kW/(m K)
Linear coefficient of thermal expansion	1.57 10^{-05} 1/K

Table 8.3: Stainless steel 55316 used in section 8.5.

Young's Modulus	320 000 MPa
Poisson's coefficient	0.24
Thermal conductivity	0.180 kW/(m K)
Linear coefficient of thermal expansion	0.48 10^{-05} 1/K

Table 8.4: AlN ceramic.

Young's Modulus	118 000 MPa
Poisson's coefficient	0.33
Thermal conductivity	0.345 kW/(m K)
Linear coefficient of thermal expansion	1.8 10^{-05} 1/K

Table 8.5: Copper-chromium-zirconium alloy (CuCrZr).

Young's Modulus	183 000 MPa
Poisson's coefficient	0.31
Thermal conductivity	0.0158 kW/(m K)
Linear coefficient of thermal expansion	1.38 10^{-05} 1/K

Table 8.6: Inconel 718.

8.2.1 Thermal loads

The thermal loads are induced by the particle fluxes that reach the assembly. Particles are emitted by the plasma. These particles have high energy and can transfer their energy to components of the reactor by elastic or inelastic collisions, which are respectively energy transfer of the incoming particles to nuclei or electrons ([Naujoks, 2006], p. 68-71). These particles have a property called the penetration depth, also named mean depth ([Naujoks, 2006], p. 68), which corresponds to their normal mean penetration into the matter. In the current study, three different types of particles play an important role: the photons, the fast charge exchange neutrals, and the neutrons.

The neutrons have a mean depth bigger than the dimensions of the assembly of the first mirror and the holder. This means that to model this loading, it can be described as a uniform volumetric heat generation depending only on the probability of collision. This probability of collision between neutrons and the nuclei or electrons depends on the material, values are listed in Table 8.7. Those values are peak values and correspond to a conservative approach; if the mirror is able to deal with the steady state at peak value, it will be able to not blur the image with smaller values of the load.

Rh	20.00 MW/m^3
WCu	2.45 MW/m^3
Stainless steel	0.70 MW/m^3
AlN	0.30 MW/m^3
CuCrZr	0.60 MW/m^3
Inconel 718	0.80 MW/m^3

Table 8.7: Assumed neutron volumetric heating for each material at peak value.

The mean depth of photons and fast charge exchange neutrals is small in comparison to the dimensions of the assembly. This allows their influence to be seen as a heat flux: the particles transfer energy and heat the material, but they do so only in a spatial region near the surface of the first mirror. These two types of particle fluxes can be seen as surface heating. We estimate the integrated photons and fast charge exchange neutral heating on the mirror surface to 20 W (around 1.65 kW/m^2).

Water channels are modeled imposing 70°C on their surfaces.

Radiation between the components of the assembly is neglected. Taking that radiation into account would reduce the temperature of the mirror leading to smaller thermal deformation; not taking it into account is a conservative approach as fulfilling the optical criteria without radiation leads to fulfilling the optical criteria with radiation.

8.2.2 Boundary conditions

Even if we have more than one body, only 6 rigid body modes have to be prevented as we have fused the meshes together. The choice of a clamping strategy is not conceptually easy as the holder will be bolted to a second holder which will be able to thermally expand. We consider two clamping strategies: a first one which may be too constraining and a second one which may be too loose. The reality should be between those two strategies. Using those two strategies, we will illustrate the fact that the deformation of the reflective surface is similar in both cases. However, we will illustrate that the position of this surface will be impacted by the clamping option. Therefore, to evaluate more precisely this position, it will be necessary to model the second holder as discussed in section 8.5.

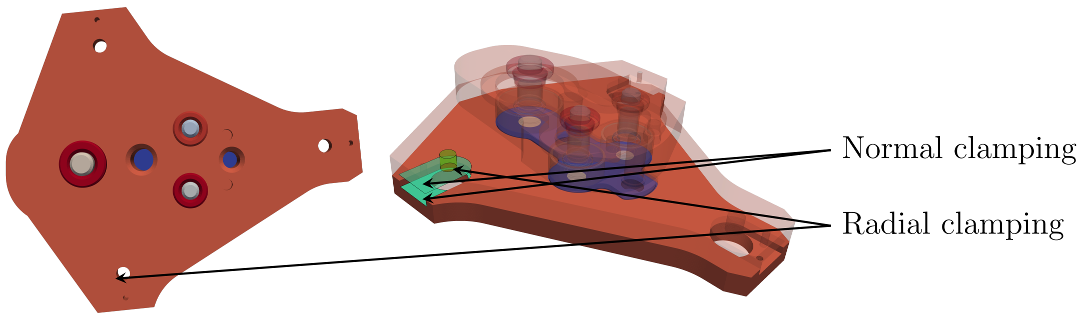


Figure 8.9: Option A: Surfaces on which normal and radial clamping are applied for one bolt (same conditions for the other two bolts).

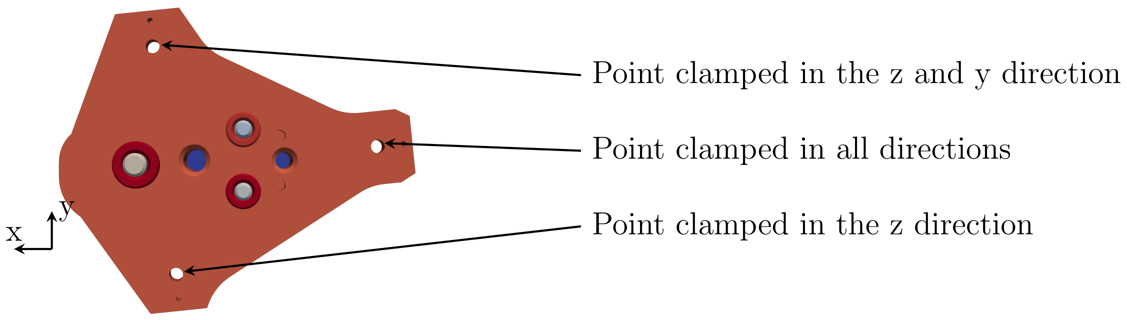

Point clamped in the z and y direction
Point clamped in all directions
Point clamped in the z direction
x
y

Figure 8.10: Option B: Second clamping strategy, one point clamped in all directions, one clamped in the z (out of plane) direction, and a last point clamped in two directions.

In the clamping strategy A, the 6 rigid body modes are suppressed by enforcing zero displacements that model the bolting of the first actively cooled holder to the second one. We enforce zero displacements along all directions except the axial direction of the three cylindrical surfaces one per bolt (where the bolts are supposed to be) and enforcing a zero normal displacement on six surfaces, two per bolt, as shown in Fig. 8.9. This strategy is, however, too constraining as it prevents the thermal expansion of the assembly creating internal compressive stresses. This explains why we consider a second clamping option too.

In the clamping strategy B, the 6 rigid body modes are suppressed by enforcing zero displacement along all directions at one point and zero displacement along two directions at another point, and zero displacement along one direction at a last point as illustrated in Fig. 8.10.

The clamping strategy does not influence the thermal distribution but influences both displacement and stresses.

8.2.3 Influence of the deformation on the optical performance

When the thermal distortion of the mirror surface is computed, we have to evaluate its impact on the optical performance and deduce whether the measurement is blurred or not. Although it is possible to import directly geometry with node-wise coordinates into ray tracing softwares, it has two main disadvantages:

- The ray tracing software uses the outward normal of reflecting surfaces to compute the reflected ray. Importing the nodal position of deformed points computed with finite elements does not enforce the smoothness of the outward normal at the edges between elements.

- The amount of data per sample that has to be transferred to the optical software is relatively large and does not allow the user to directly identify how the mirror has been deformed. Moreover, importing surfaces node-wise prevents some sensitivity analysis of ray tracing software.

Therefore, instead of importing the surface node-wise, the researchers at FZ Jülich use another strategy: the construction of an approximation of the deformation of the mirror surface with Zernike polynomials. The computed coefficients can then be introduced in the ray tracing software to deform the mirror surface and compute the impact of the deformation on the optical quality. Although those polynomials provide a suitable way to easily communicate between software, those coefficients should be interpreted with caution. The Zernike polynomials are orthogonal on the unit disk but not on the considered mirror surface. This lack of orthogonality impacts both the numerical strategy to compute the coefficients and the interpretation that can be drawn from them. Typically, when considering mirrors with disk surface, the coefficients can be computed projecting the deformation on the polynomials and the coefficients can be interpreted as piston mode, rotation, curvature change, and higher order deformation of the mirror surface. Due to the lack of orthogonality, we use a weighted least-squares fitting [Draper and Smith, 1998] where the weight of each nodal value is the area of the surface associated to this node to compute the first four Zernike coefficients. The computed coefficients can be seen as the approximation of the piston mode, the approximations of the rotations, and the approximation of the curvature change. The remaining deformation is left as irregularity as illustrated in Fig. 8.11. Those values can, afterwards, be used to deduce whether the image of the optical system is blurred or not.

Based on optical sensitivity analysis admissible values for the deformation [Krasikov et al., 2017], change of position, and orientation of the front mirror have been defined. Those values are listed in Table 8.8. These optical tolerances are based on the decomposition of the mirror deformation shown in Fig. 8.11 where the values to be compared with tolerance values for curvature and irregularity correspond respectively to distances d_1 and d_2. It is important to emphasize that the spatial frequency of the irregularity change should be checked too: if d_2 is smaller than the tolerance and if the irregularity is sufficiently smooth without any high frequency space distribution, then the irregularity will not blur the image; if one of the previous conditions is not fulfilled, the image will be blurred.

Translation along z	Rotation along x	Rotation along y	Curvature	Irregularity
± 2 mm	± 1.75 mrads	± 1.75 mrads	± 100 fringes[1]	± 100 fringes

Table 8.8: Optical tolerances.

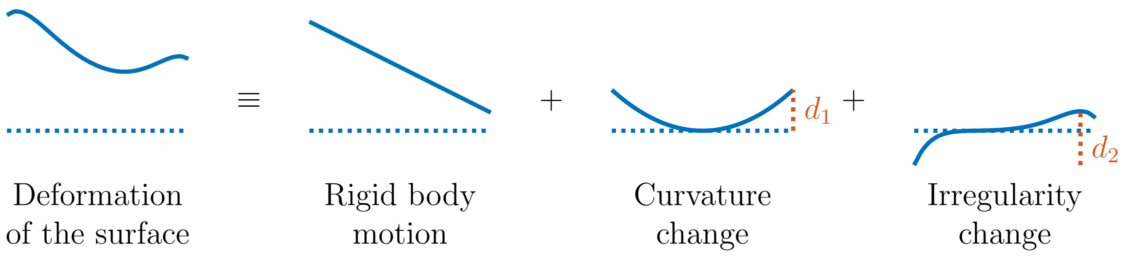


Figure 8.11: Decomposition of the deformation of the mirror surface in rigid body motion, change in curvature, and change in irregularity. The blue dotted line corresponds to the initial configuration of the mirror surface. The quantities d_1 and d_2 correspond to the values which have to be compared with the curvature and irregularity tolerance respectively.

8.2.4 Uncertain parameters

In this subsection, we discuss the choice of the uncertain parameters and the chosen uncertainty quantification strategy.

Due to neutron irradiation, material properties of the parts used of the assembly evolve during their lifetime. In particular, the heat conductivity of the AlN spacers decreases when those spacers are irradiated. The heat conductivity of the AlN ceramic before irradiation is known to be 180 W/mK and, at the end of the lifetime of ITER, is estimated to be at 30 W/mK. Such a reduction of the heat conductivity results in increased temperature on the mirror and larger deformation of the reflective surface.

The mirror assembly has three AlN spacers, two thin spacers and a thicker spacer located closer to the plasma. In order to ease the illustration of the results, we restrict ourselves to 2 uncertain parameters by assuming that the two thin spacers associated with studs 1 and 2 of Fig. 8.6 have the same heat conductivity k_1 and that the thickest spacer associated with stud 3 of Fig. 8.6 has potentially another heat conductivity k_2. Moreover, the thickest spacer, the one associated with stud 3 of Fig. 8.6, is located closer to the plasma, therefore, we expect that it will be more irradiated than the thinnest spacers and and that $k_2 \leq k_1$. From all those physical considerations, we have chosen to model the uncertain parameters (k_1, k_2) as two random variables with uniform probability density function in the triangle with vertices (30 W/mK,30 W/mK), (180 W/mK,180 W/mK), and (180 W/mK,30 W/mK).

[1]A fringe corresponds to the distance between two points of maximal luminous intensity in the Michelson interferometer in the case of He laser. In other words, two fringes equal 632.8 nm. Then, 100 fringes correspond to 0.0316 mm.

Concerning the uncertainty quantification strategy, we build a polynomial chaos expansion surrogate model [Arnst and Ponthot, 2014; Le Maître and Knio, 2010]:

$$qoi \approx \sum_{m=0}^{d} \sum_{n=0}^{d-m} c_{m,n} \phi_{m,n}(k_1, k_2), \tag{8.1}$$

where d is the maximal degree of the polynomials used.

The samples are chosen following the strategy of [Xiao and Gimbutas, 2010] which computes weights and locations of samples to have precise numerical integrations of polynomials up to a certain degree; as an example, the 453 samples related to the integration of polynomials of degree 50 are illustrated in Fig. 8.12.

In this work, we have chosen to use orthogonal functions $\phi_{m,n}$ such that:

$$\int_{30}^{180} \int_{30}^{k_1} \phi_{m,n}(k_1, k_2) \phi_{o,p}(k_1, k_2) dk_2 dk_1, \tag{8.2}$$

is zero if $m \neq o$ or $n \neq p$. To do so, we use the orthogonal basis for all polynomials of degrees at most d:

$$K_{m,n}(u, v) = P_m \left(\frac{2u + v + 1}{1 - v} \right) \left(\frac{1 - v}{2} \right)^m P_n^{2m+1,0}(v), \tag{8.3}$$

with m and n two positive or null integers such that $m + n \leq d$, $P_n^{\alpha,\beta}$ the nth degree Jacobi polynomial corresponding to parameters α and β, and P_m the mth degree Legendre polynomial. The functions $K_{m,n}(u, v)$ are defined on the triangle with vertices (-1,-1),(-1,1), and (1,-1) and are orthogonal with respect to the uniform weight function [Koornwinder, 1975; Xiao and Gimbutas, 2010].

In order to transform from the triangle with vertices (30 W/mK,30 W/mK), (180 W/mK,180 W/mK), and (180 W/mK,30 W/mK) to the triangle with vertices (-1,-1), (-1,1), and (1,-1), we use this transformation:

$$u = 1 - \frac{k_1 - 30\,\mathrm{W/mK}}{75\,\mathrm{W/mK}}, \tag{8.4}$$

$$v = -1 + \frac{k_1 - 30\,\mathrm{W/mK}}{75\,\mathrm{W/mK}}. \tag{8.5}$$

8.2.5 Quantities of interest

The chosen quantities of interest are the maximal, mean, and minimal temperature reached on the mirror surface, the displacement of the mirror surface: the value of the piston mode, both rotations, the curvature change, and the irregularity, and the preload loss in the studs.

The preload loss is evaluated computing the integral of σ_{zz} through the cut of the stud by cutting every tetrahedron of the stud with a cutting plane, triangularizing the 2D shapes, looping on the newly defined triangles, and computing the sum over those triangles of the product of their area and their vertical stress.

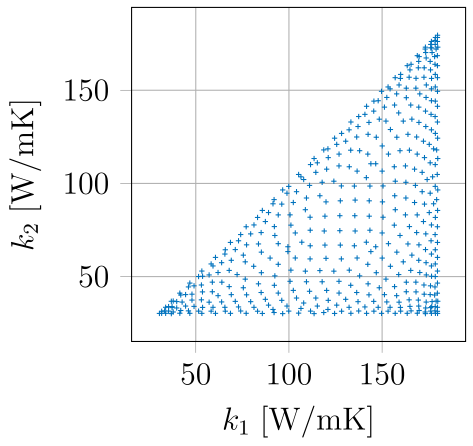


Figure 8.12: 453 samples of the mirror problem based on [Xiao and Gimbutas, 2010] to integrate polynomials of degree 50.

8.2.6 Preconditioner

For this problem, we use the multigrid preconditioner described in section 5.2.3 with 1 iteration of the backward block Gauss-Seidel with a damping factor equal to 1 as level smoother, the threaded Gauss-Seidel as smoother of the two blocks, and 3 levels with sizes listed in Table 8.9. The smoother of the first block is 20 iterations of the threaded Gauss-Seidel with a damping factor equal to 0.97 and the smoother of the second block is 20 iterations of the threaded Gauss-Seidel with a damping factor equal to 1.04. The damping factors of the threaded Gauss-Seidel have been computed minimizing the spectral radius of the Successive Over Relaxation iteration matrix. The spectral radii have been computed with a power method [Golub and Van Loan, 2012].

level	rows	non-zero entries	non-zero entries per row	coarsening ratio	level smoother
0	1 316 228	60 551 257	46		BGS
1	82 048	5 985 434	43.74	16.04	BGS
2	6 560	301 834	38.32	12.51	Klu

Table 8.9: Multigrid information.

8.2.7 GMRES stopping criteria

For this problem, we use the combination of four stopping criteria to enforce the convergence of GMRES for the temperature and displacement fields. The strategy used is to require the enforcement of the stopping criteria sequentially; the first stopping criterion has to be fulfilled before evaluating the second one, etc. The four stopping criteria are the following:

- An implicit stopping criterion:

$$\boldsymbol{q}_{:(j+1)}^{\mathrm{T}}\boldsymbol{e}_1 \leq 10^{-5}, \tag{8.6}$$

  this criterion is based on (3.9). This is an implicit stopping criterion, as discussed in section 3.1, as it does not require the computation of the residual nor the solution vector.

- An explicit stopping criterion based on the relative norm of the residual linked to the temperature field:

$$\frac{\|\boldsymbol{L}\boldsymbol{t}^{(j)} - \boldsymbol{l}\|}{\|\boldsymbol{L}\boldsymbol{t}^{(0)} - \boldsymbol{l}\|} \leq 10^{-7}, \tag{8.7}$$

- An explicit stopping criterion based on the relative norm of the residual linked to the displacement fields:

$$\frac{\|\boldsymbol{K}\boldsymbol{u}^{(j)} + \boldsymbol{S}\boldsymbol{t}^{(j)} - \boldsymbol{f}\|}{\|\boldsymbol{K}\boldsymbol{u}^{(0)} + \boldsymbol{S}\boldsymbol{t}^{(0)} - \boldsymbol{f}\|} \leq 10^{-6}, \tag{8.8}$$

- An explicit stopping criterion based on the absolute norm of the residual linked to the vertical displacement field of the mirror:

$$\|\boldsymbol{W}\left(\boldsymbol{K}\boldsymbol{u}^{(j)} + \boldsymbol{S}\boldsymbol{t}^{(j)} - \boldsymbol{f}\right)\| \leq 10^{-3}, \tag{8.9}$$

  where $\boldsymbol{W}$ is a diagonal weight matrix with 0 on the diagonal and 1 on the diagonal for the degrees of freedom associated to the vertical displacement of the mirror. The units of the weight matrix are chosen to nondimensionalize the vector $\boldsymbol{f}$.

8.3 Numerical results

Before looking at the results of the uncertainty quantification, we will investigate the two extreme cases where the irradiation has not already reduced the heat conductivity of AlN ceramic and where it has reduced it to 30 W/(m K).

8.3.1 Material before irradiation

The temperature field computed for the case where all the AlN ceramic spacers have a heat conductivity of 180 W/(m K) is illustrated in Fig. 8.13 and in Fig. 8.14 and Fig. 8.15 for the cut 1 and 2 respectively.

We observe that the actively cooled holder is the coolest part and has a temperature which is close to the water temperature of 70 °C. This is due to the good heat conductivity of the CuCrZr. The hottest part, as illustrated in the two cuts, is the mirror itself and the hottest point of the reflective surface is located in the closest region to the plasma; on the edge of the surface on the side of the thickest spacer as illustrated in Fig. 8.20. The AlN spacers need to conduct all the heat generated by the emitted particles which reach the mirror and the mirror substrate to the holder as we observe that the studs themselves do not conduct the main part heat as expected. This explains the high temperature gradients inside the AlN spacers.

We can illustrate the associated displacement field for the clamping option A in the cut in Fig. 8.16 and Fig. 8.17 and for the clamping option B in the cut in Fig. 8.18 and Fig. 8.19. We observe that the mirror surface Fig. 8.21 has faced a positive displacement along the z axis away from the plasma and negative displacement closer to the plasma for the clamping option A whereas, for the clamping option B, we observe a positive displacement at each point of the mirror surface Fig. 8.22.

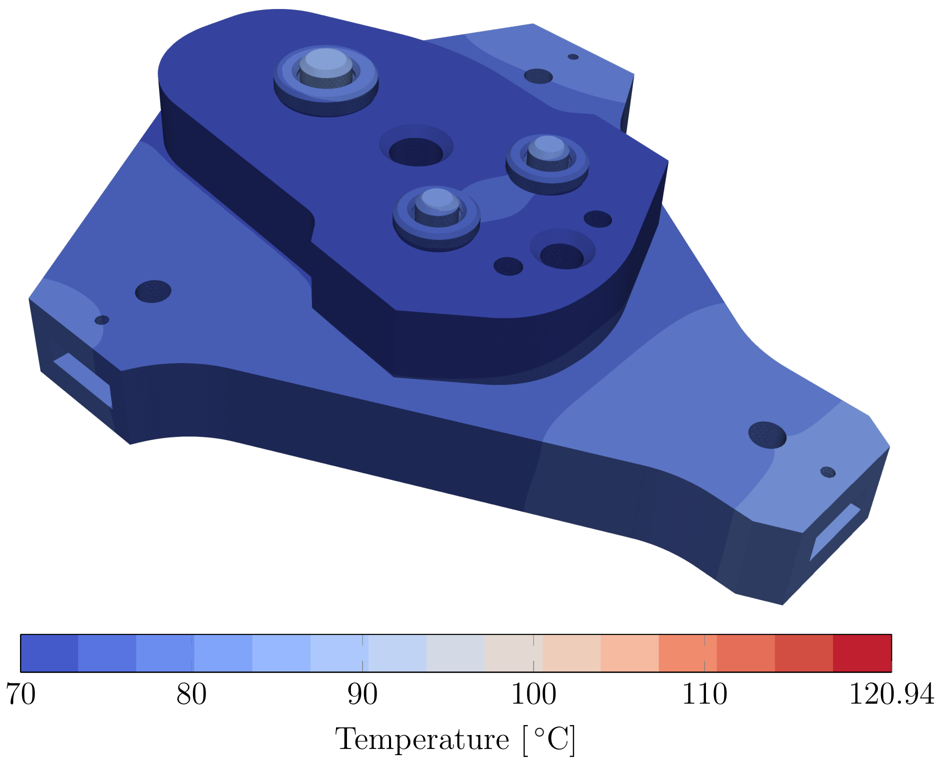


Figure 8.13: Temperature of the assembly before irradiation.

8.3.2 Material after irradiation

When the heat conduction of AlN is decreased due to neutron irradiation, the maximal temperature is increased as seen in Fig. 8.23, 8.24, and 8.29 because the spacers have to extract the same heat from the mirror and mirror substrate but with a heat conduction divided by 6 leading to a temperature difference multiplied by 6 inside the spacers.

This increased temperature impacts the displacement field in both clamping strategies as illustrated in Fig. 8.25, 8.26, Fig. 8.30, Fig. 8.27, 8.28, and 8.31.

When comparing Fig. 8.30 and 8.31 with Fig. 8.21 and Fig. 8.22 respectively, we observe that the increased temperature gradient due to the increased temperature of the mirror surface leads to an increased curvature change.

In table 8.10, we list the coefficients extracted from the mirror displacement for the two cases before and after irradiation with the two clamping options. We observe that, regardless of the clamping option, the mirror with non-irradiated AlN ceramic fulfils the optical criteria whereas the case with fully irradiated AlN ceramic has too large a curvature change leading to blurred measurement of the optical device. In the next section, based on the uncertainty quantification study, we will investigate intermediate states.

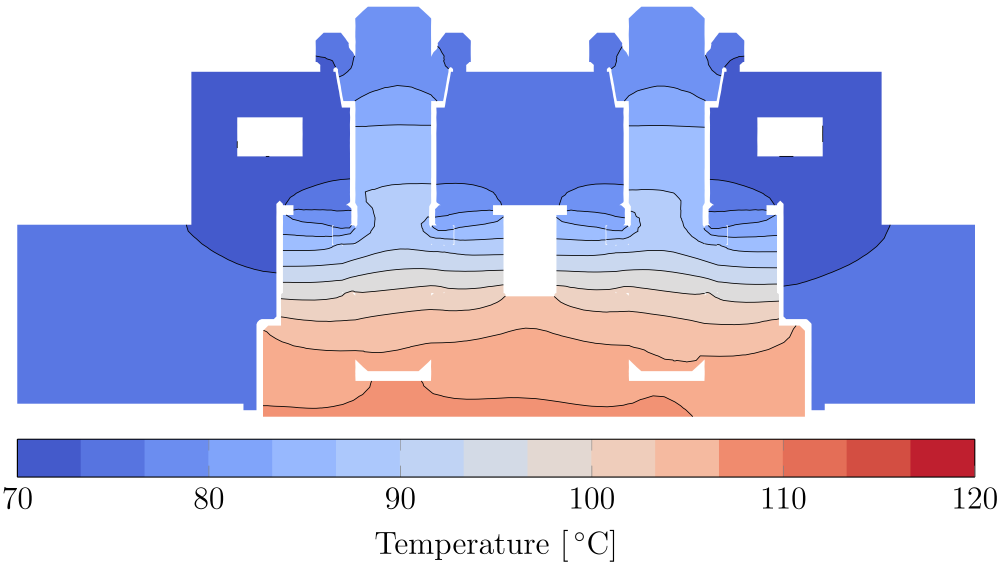


Figure 8.14: Temperature of the first cut before irradiation.

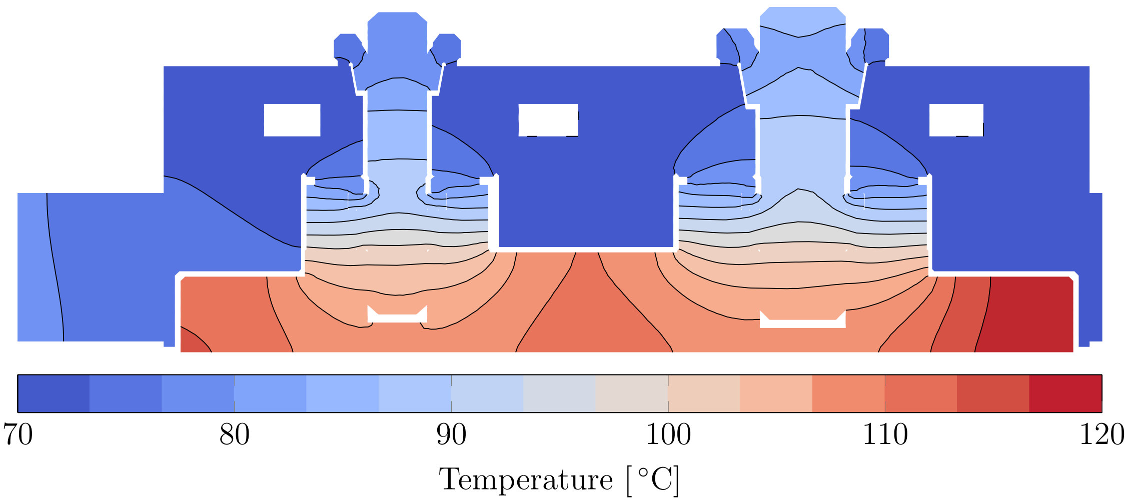


Figure 8.15: Temperature of the second cut before irradiation.

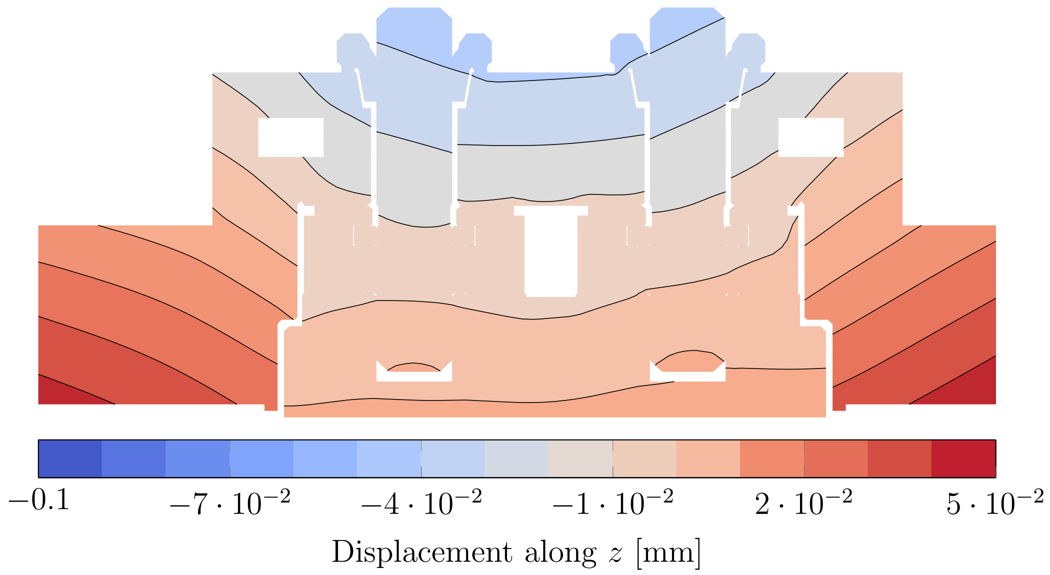


Figure 8.16: Displacement along z of the first cut before irradiation for the clamping option A.

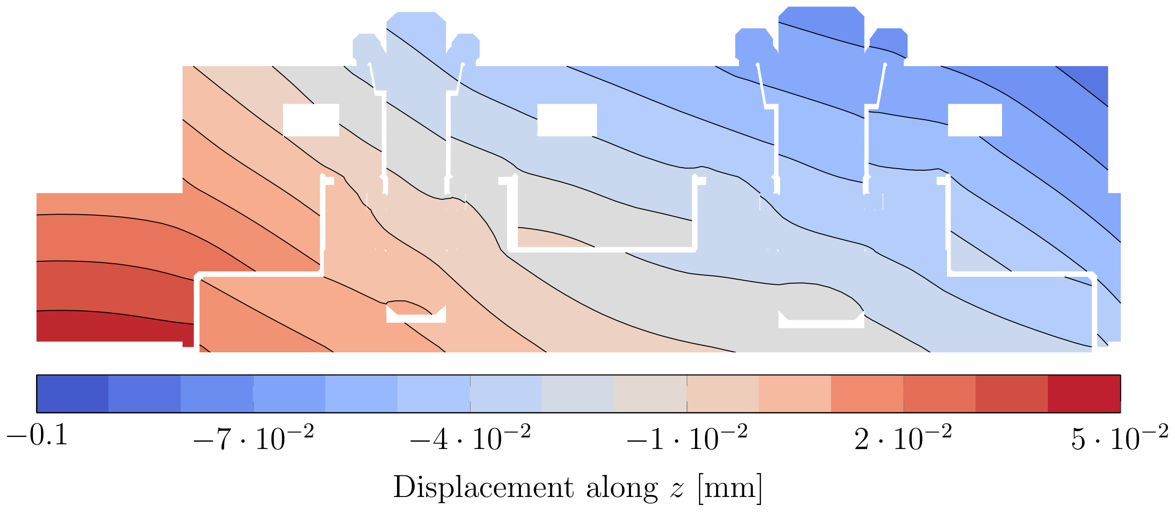


Figure 8.17: Displacement along z of the second cut before irradiation for the clamping option A.

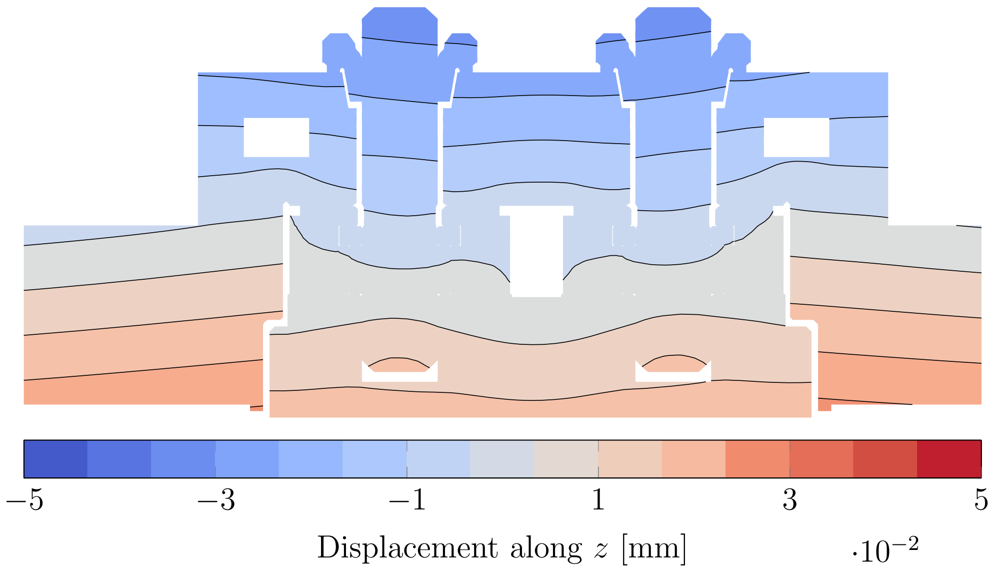


Figure 8.18: Displacement along z of the first cut before irradiation for the clamping option B.

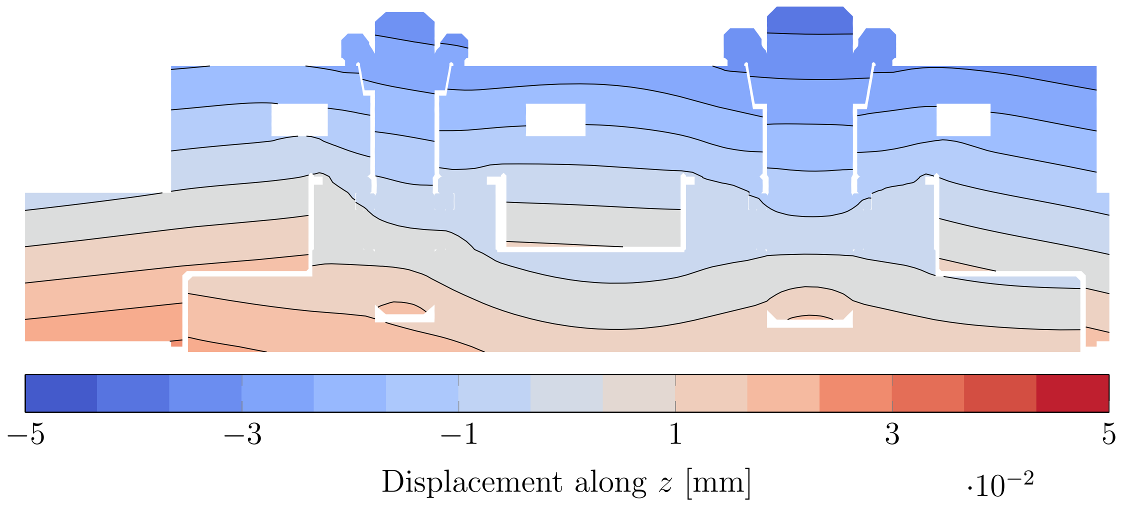


Figure 8.19: Displacement along z of the second cut before irradiation for the clamping option B.

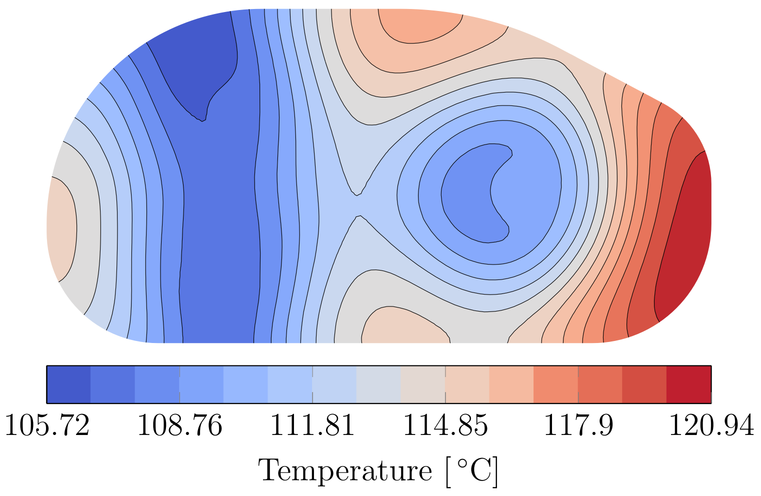


Figure 8.20: Temperature of the mirror surface before irradiation.

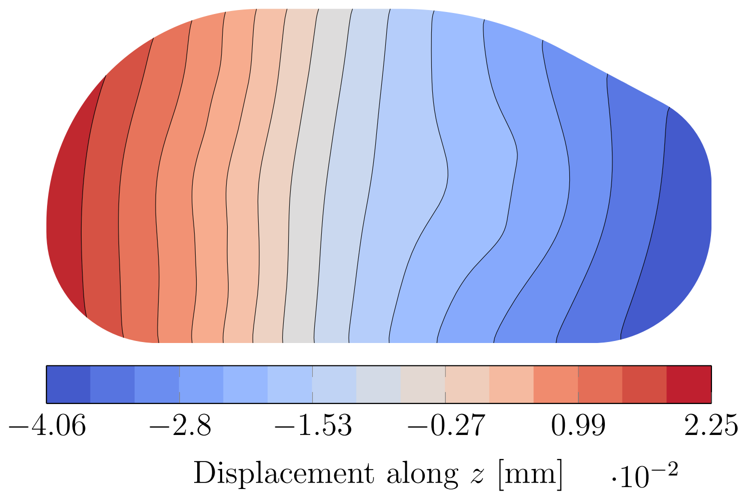


Figure 8.21: Normal displacement (normal initially aligned along the z axis) of the mirror surface for clamping option A before irradiation.

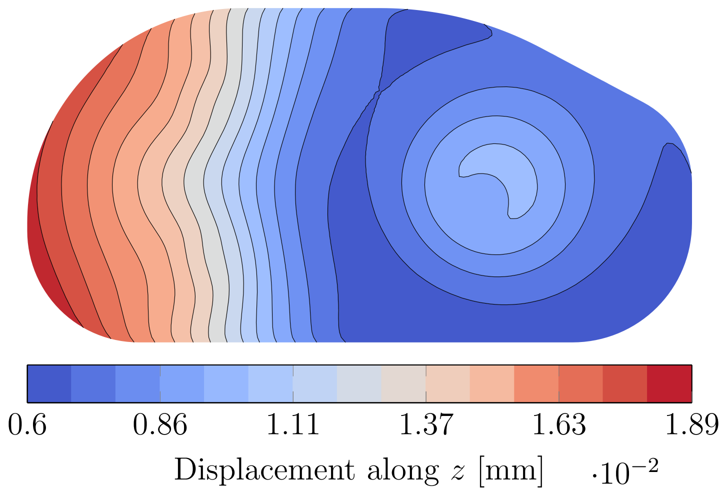


Figure 8.22: Normal displacement (normal initially aligned along the z axis) of the mirror surface for clamping option B before irradiation.

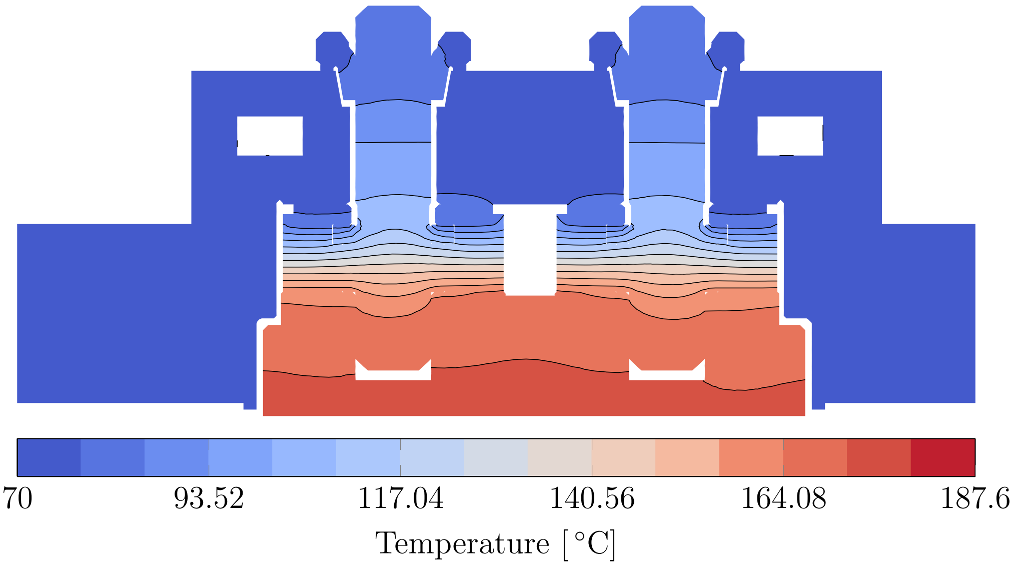


Figure 8.23: Temperature of the first cut after irradiation.

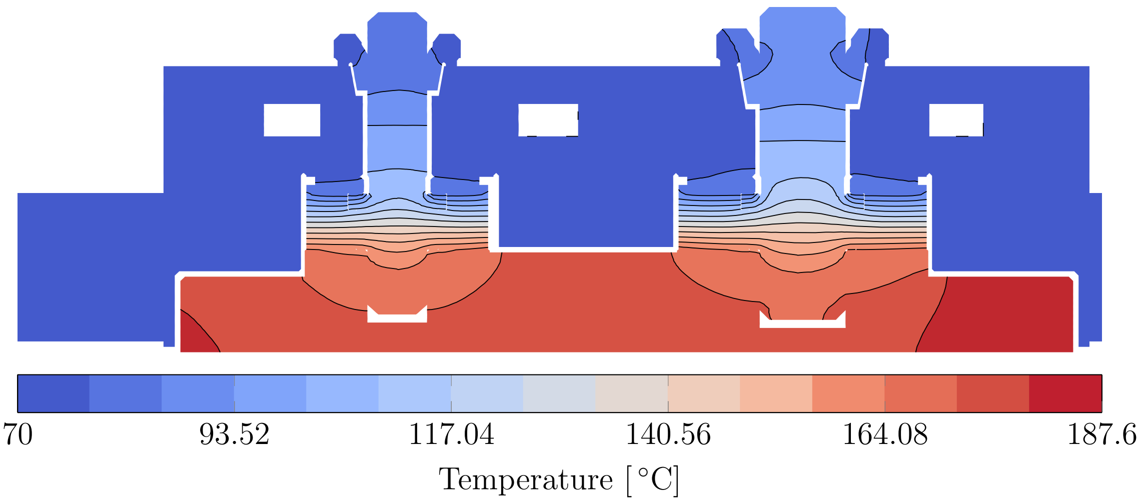


Figure 8.24: Temperature of the second cut after irradiation.

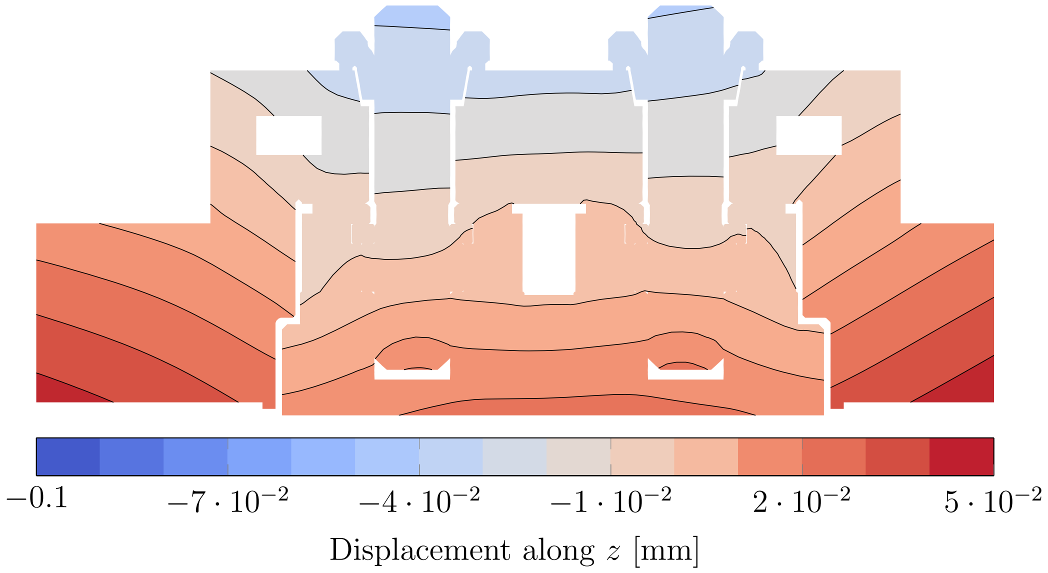


Figure 8.25: Displacement along z of the first cut after irradiation for the clamping option A.

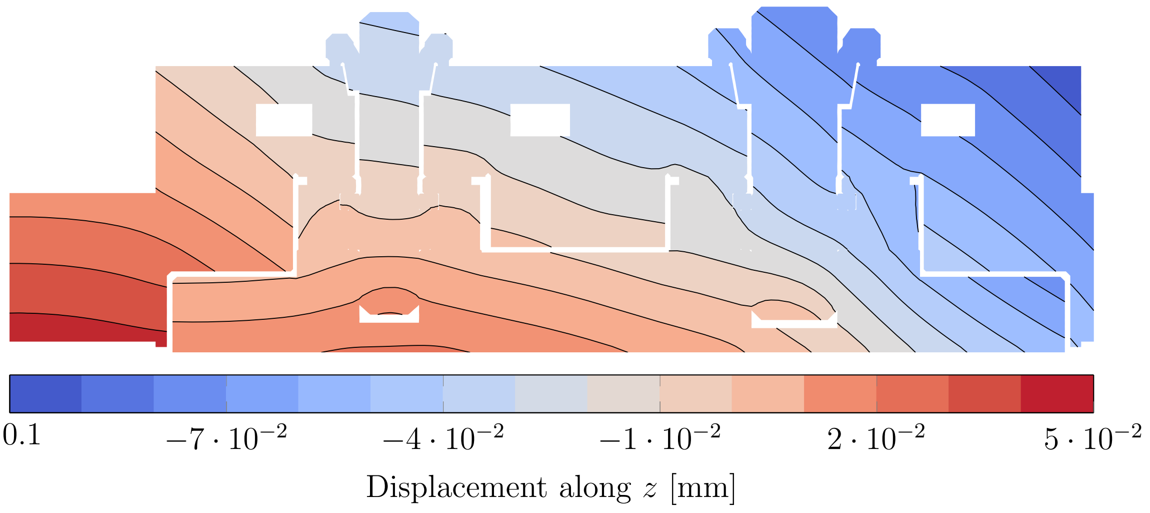


Figure 8.26: Displacement along z of the second cut after irradiation for the clamping option A.

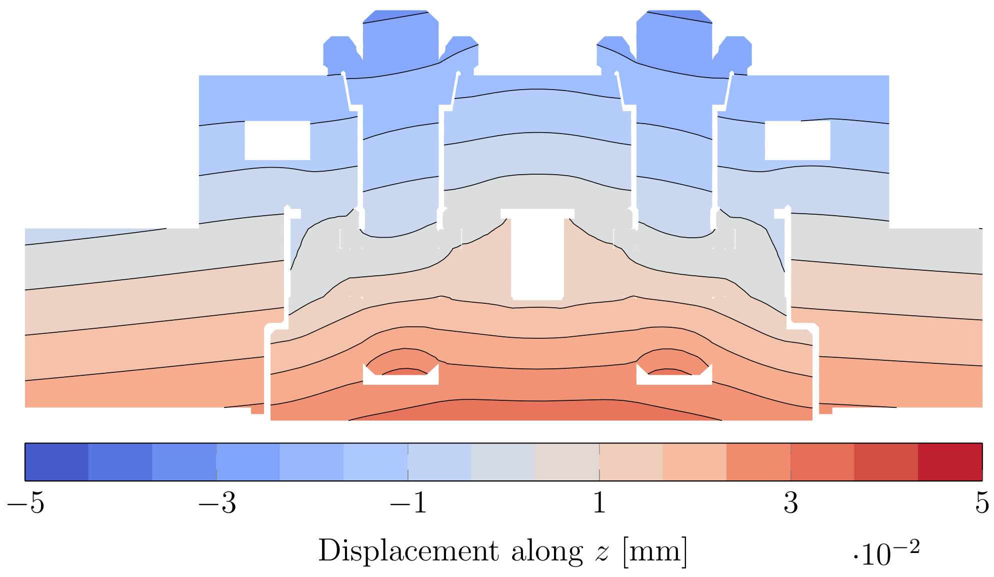


Figure 8.27: Displacement along z of the first cut after irradiation for the clamping option B.

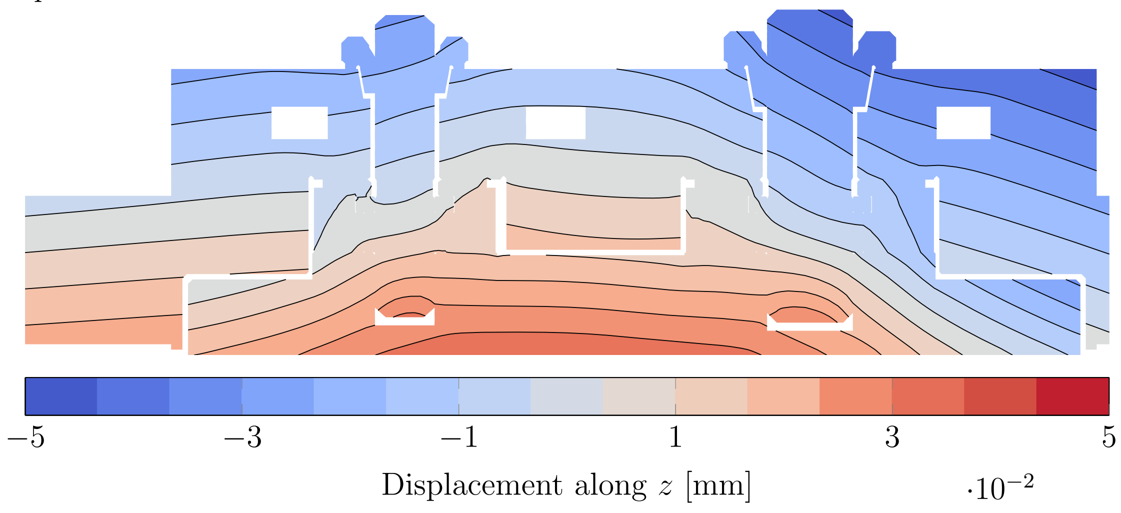


Figure 8.28: Displacement along z of the second cut after irradiation for the clamping option B.

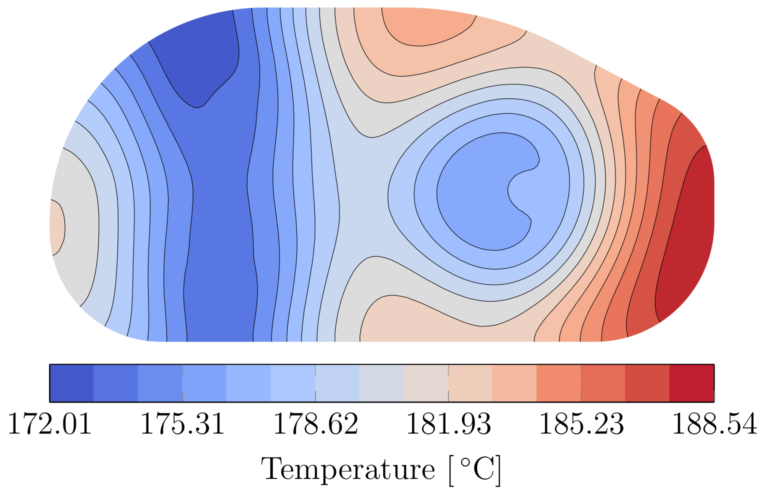


Figure 8.29: Temperature of the mirror surface after irradiation.

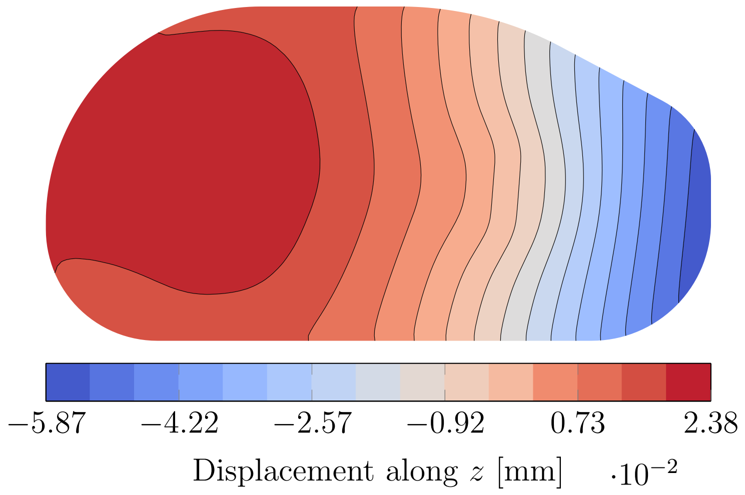


Figure 8.30: Normal displacement (normal initially aligned along the z axis) of the mirror surface for clamping option A after irradiation.

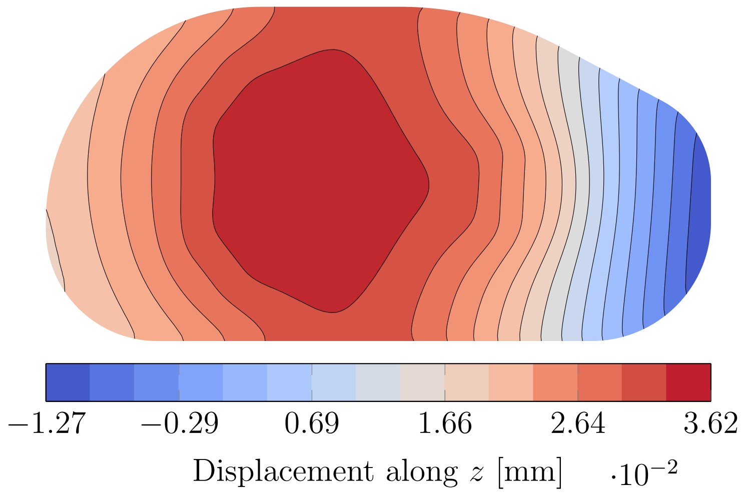


Figure 8.31: Normal displacement (normal initially aligned along the z axis) of the mirror surface for clamping option B after irradiation.

	Tolerance	Before irradiation		After irradiation	
		option A	option B	option A	option B
Translation along z [mm]	± 2	-0.01501	0.01110	-0.01824	0.00752
Rotation around y [mrads]	± 1.75	0.35026	0.04512	0.60480	0.31118
Rotation around x [mrads]	± 1.75	-0.04077	-0.00387	-0.08064	-0.04135
Curvature change [mm]	± 0.0316	0.01090	0.00832	-0.04723	-0.04990
Irregularity change [mm]	± 0.0316	0.00688	0.00553	0.00742	0.00891

Table 8.10: Optical tolerances of the two extreme cases with the two clamping options.

8.3.3 Uncertainty quantification

Now that the two extreme cases have been discussed in detail, we have evaluated the convergence of the surrogate model with respect to the degree d. To do so, we have evaluated the expectation:

$$E\left\{\|q^{(d+1)} - q^{(d)}\|^2\right\}, \tag{8.10}$$

where $q^{(d)}$ is the surrogate model built with polynomials of degree smaller or equal to d of the quantity of interest:

$$q^{(d)}(k_1, k_2) = \sum_{m=0}^{d} \sum_{n=0}^{d-m} c_{m,n}\phi_{m,n}(k_1, k_2). \tag{8.11}$$

Fig. 8.32 illustrates the expectation $E\left\{\|q^{(d+1)} - q^{(d)}\|^2\right\}$ as a function of the degree d for the curvature change for the clamping option A. Those results illustrate the fact that the surrogate model converges. Due to the already small value of $E\left\{\|q^{(d+1)} - q^{(d)}\|^2\right\}$ for $d = 10$ and because the number of points that have to be evaluated grows more than linearly with d, we choose $d = 10$.

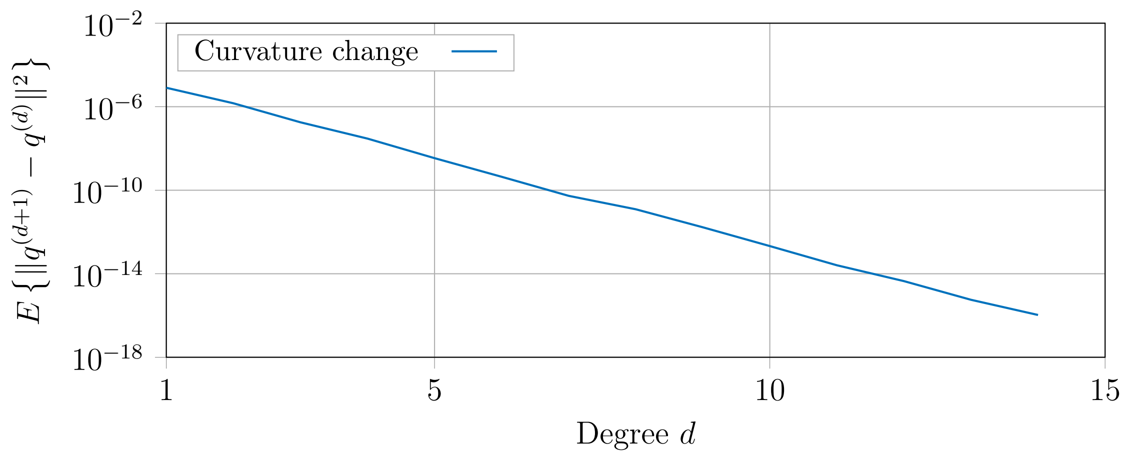


Figure 8.32: Convergence of the surrogate model.

We have evaluated the finite element model for the 79 samples for $d = 10$ and list their quantities of interest for both clamping strategies in Fig. 8.33, Fig. 8.34, Fig. 8.35, Fig. 8.36, and Fig. 8.37. We construct the surrogate models with $d = 10$ using the quantities of interest of the computed samples and illustrate them in Fig. 8.38, 8.39, 8.40, 8.41, and 8.42.

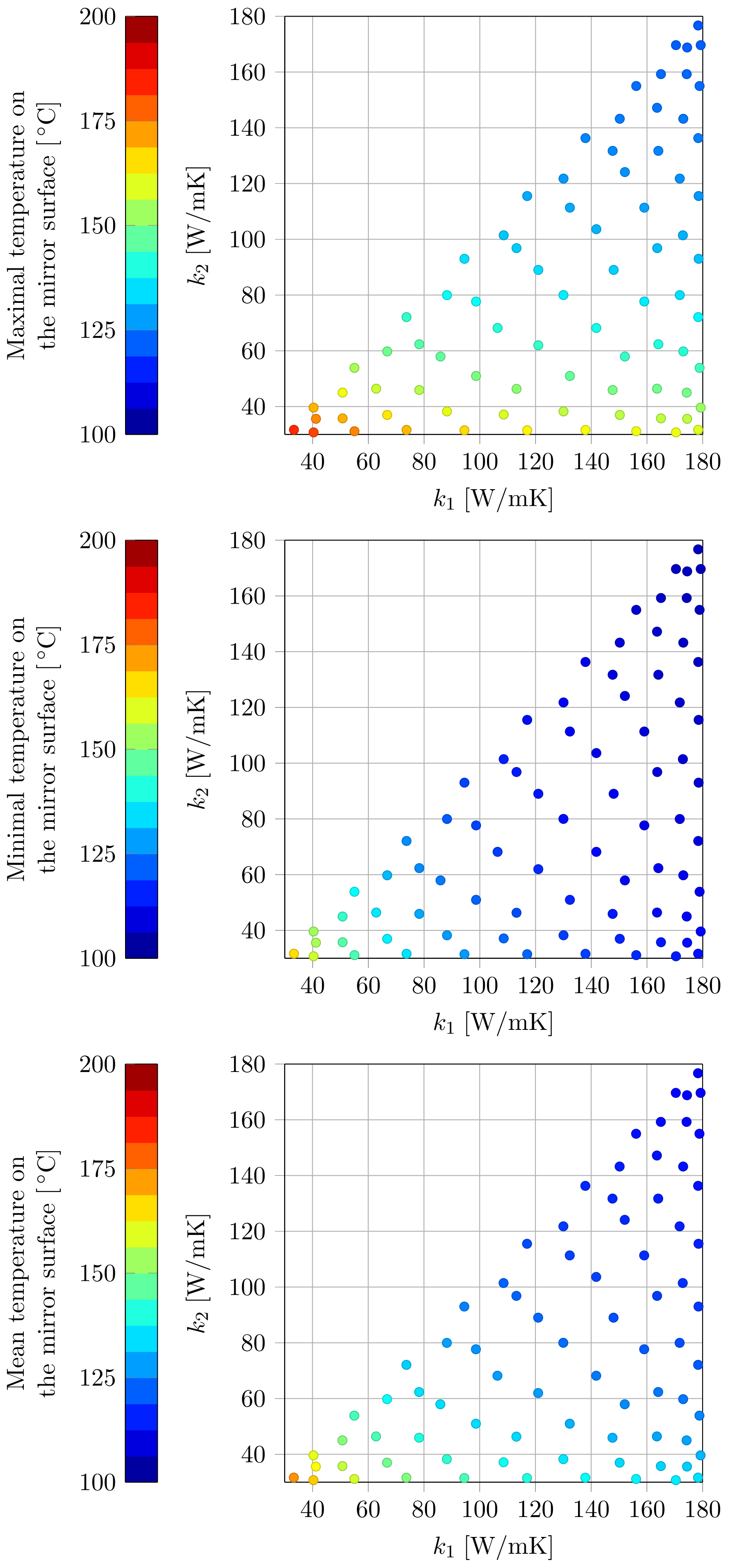


Figure 8.33: Quantities of interest.

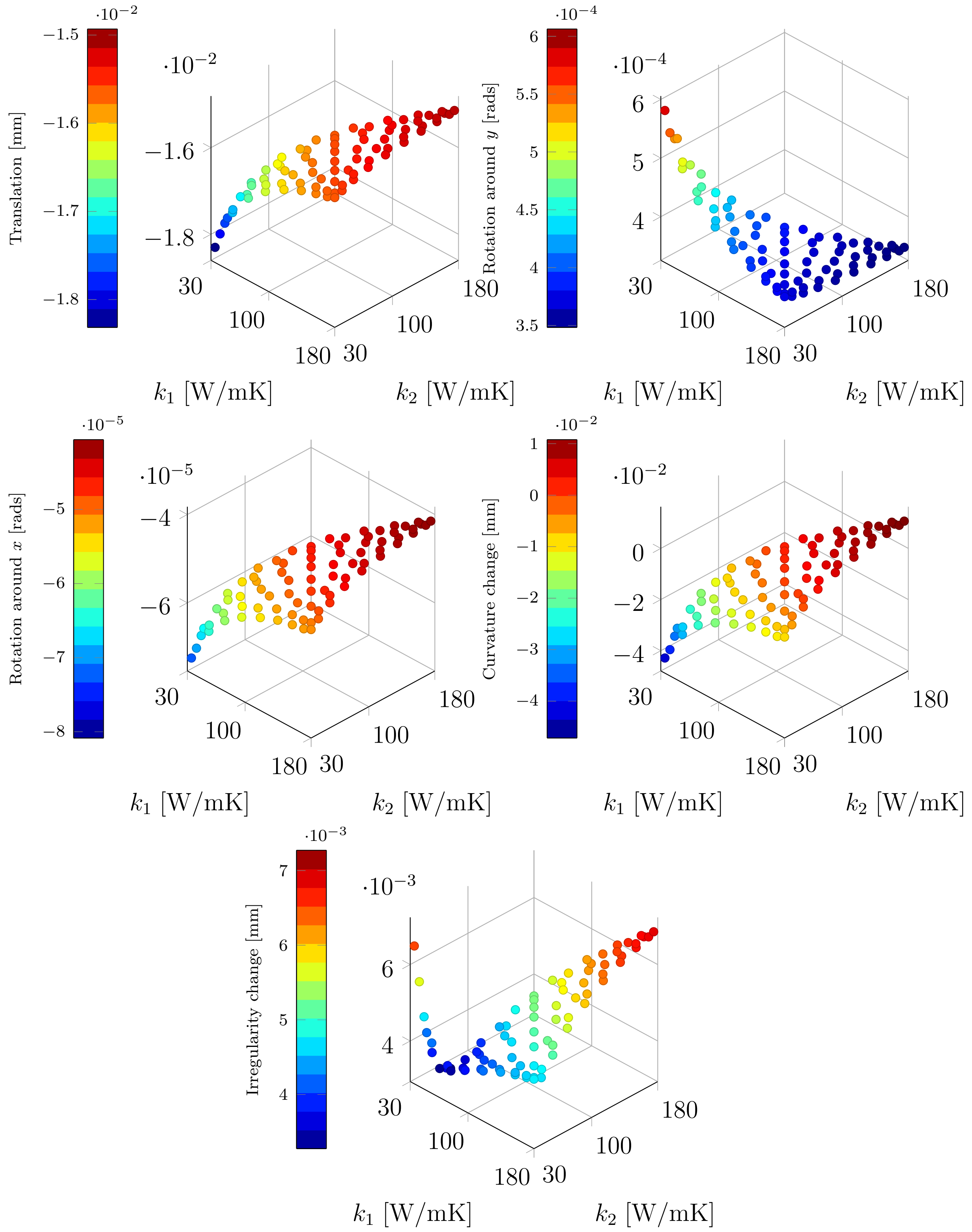


Figure 8.34: Optical quantities of interest for clamping option A.

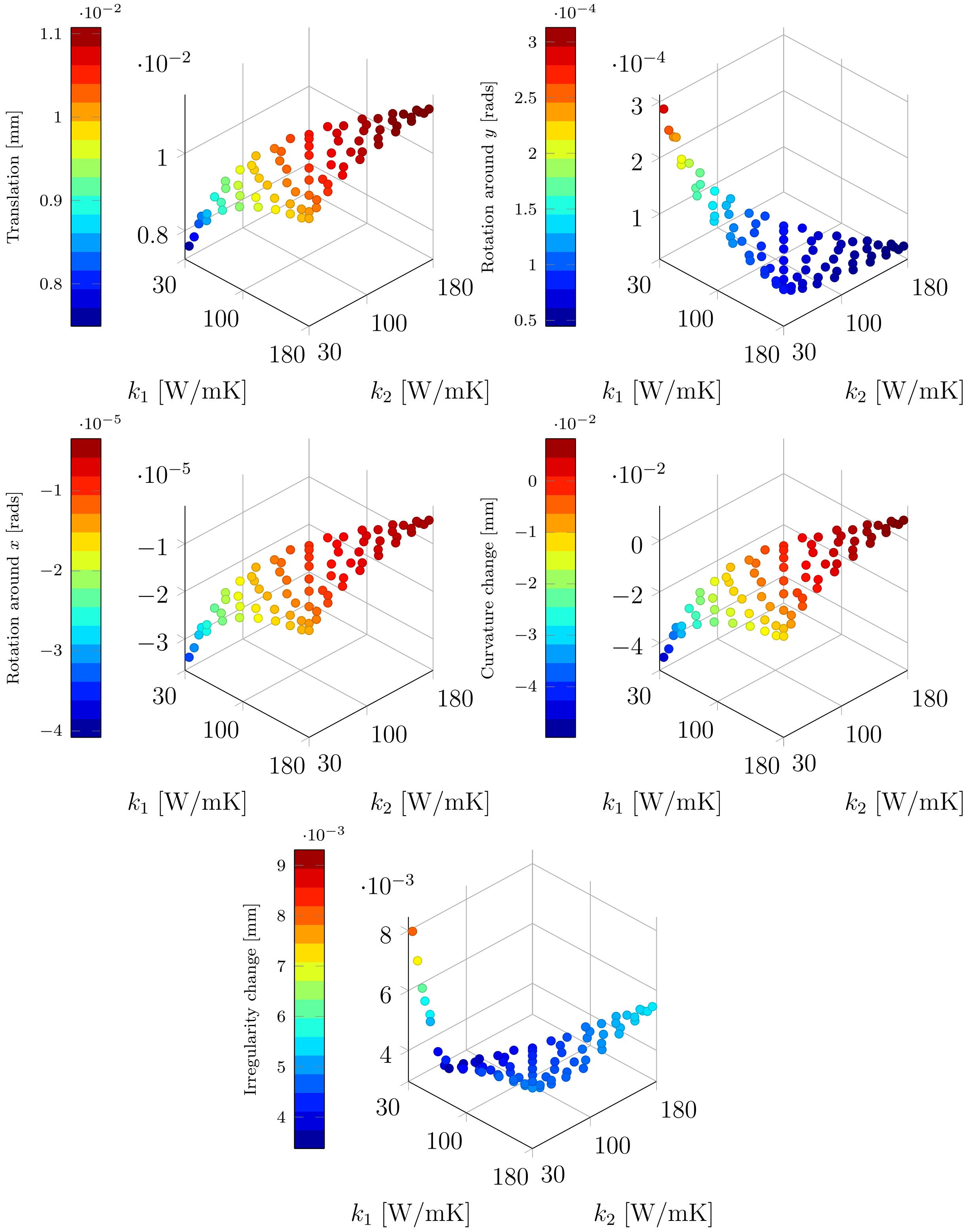


Figure 8.35: Optical quantities of interest for clamping option B.

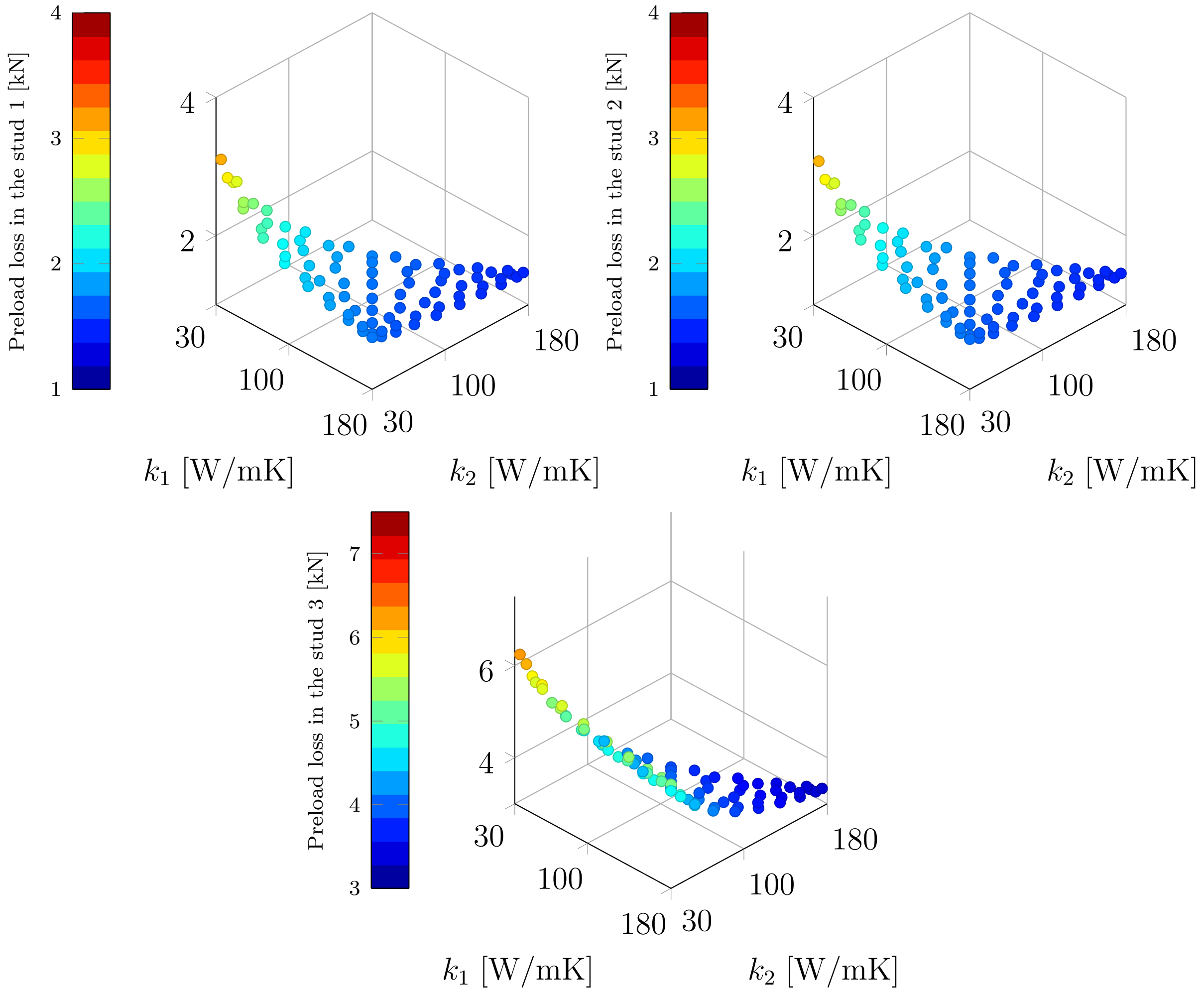


Figure 8.36: Preload loss for clamping option A.

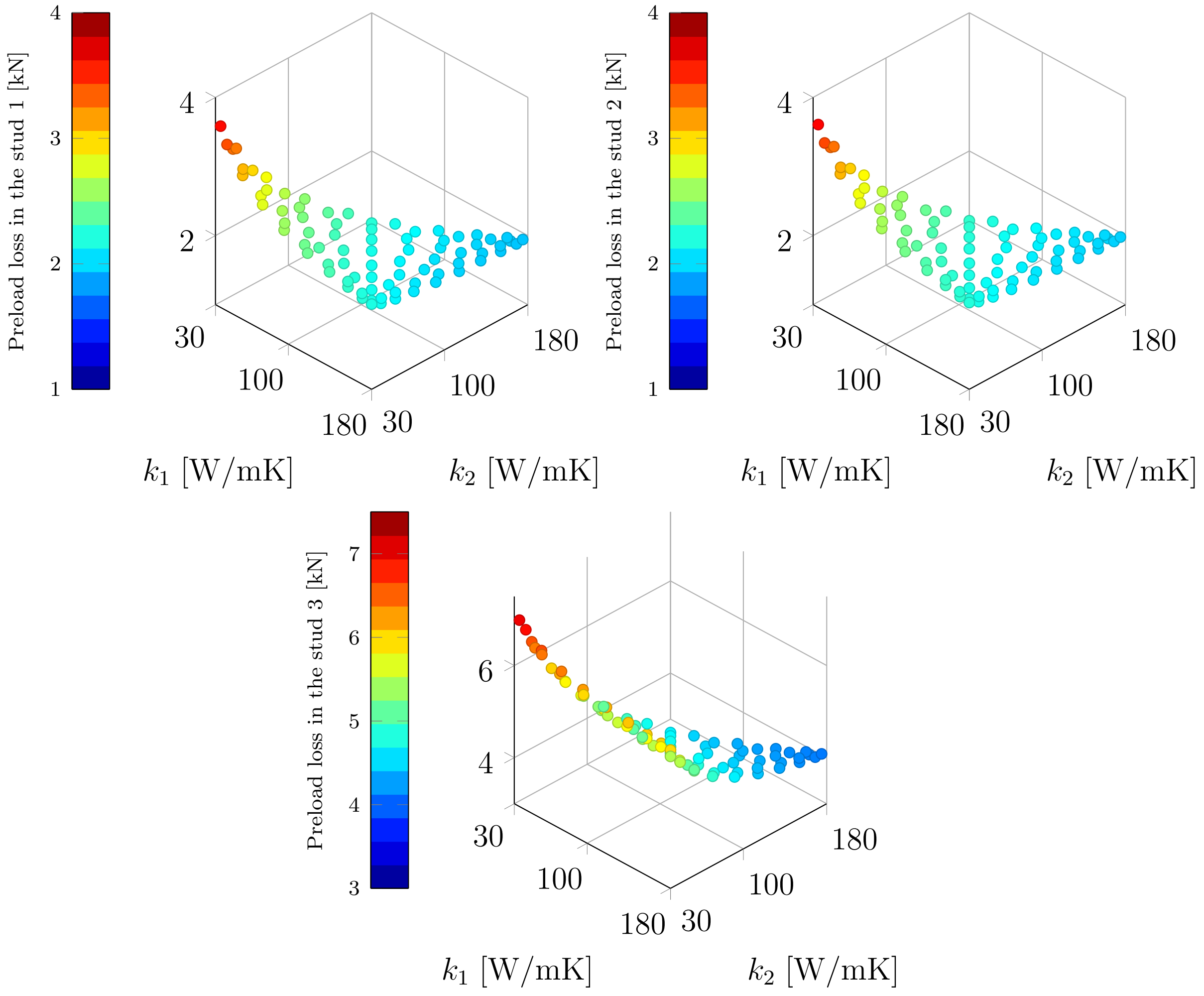


Figure 8.37: Preload loss for clamping option B.

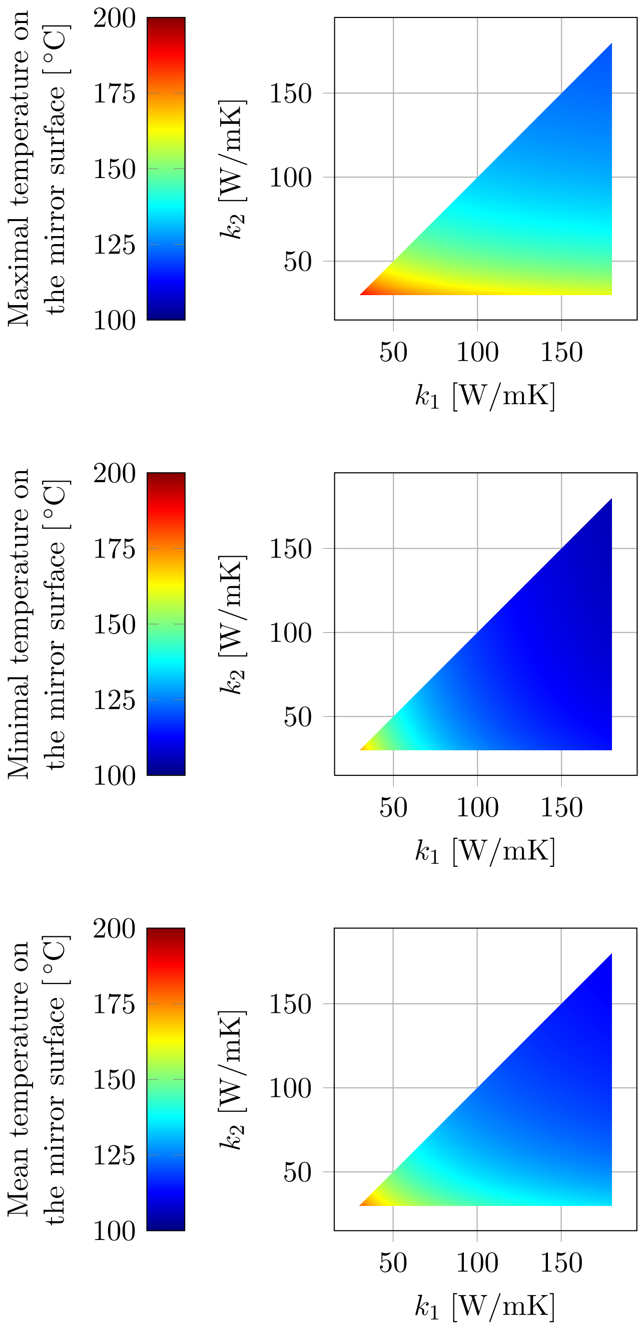


Figure 8.38: Surrogate models of the maximal, minimal, and mean temperature on the mirror surface.

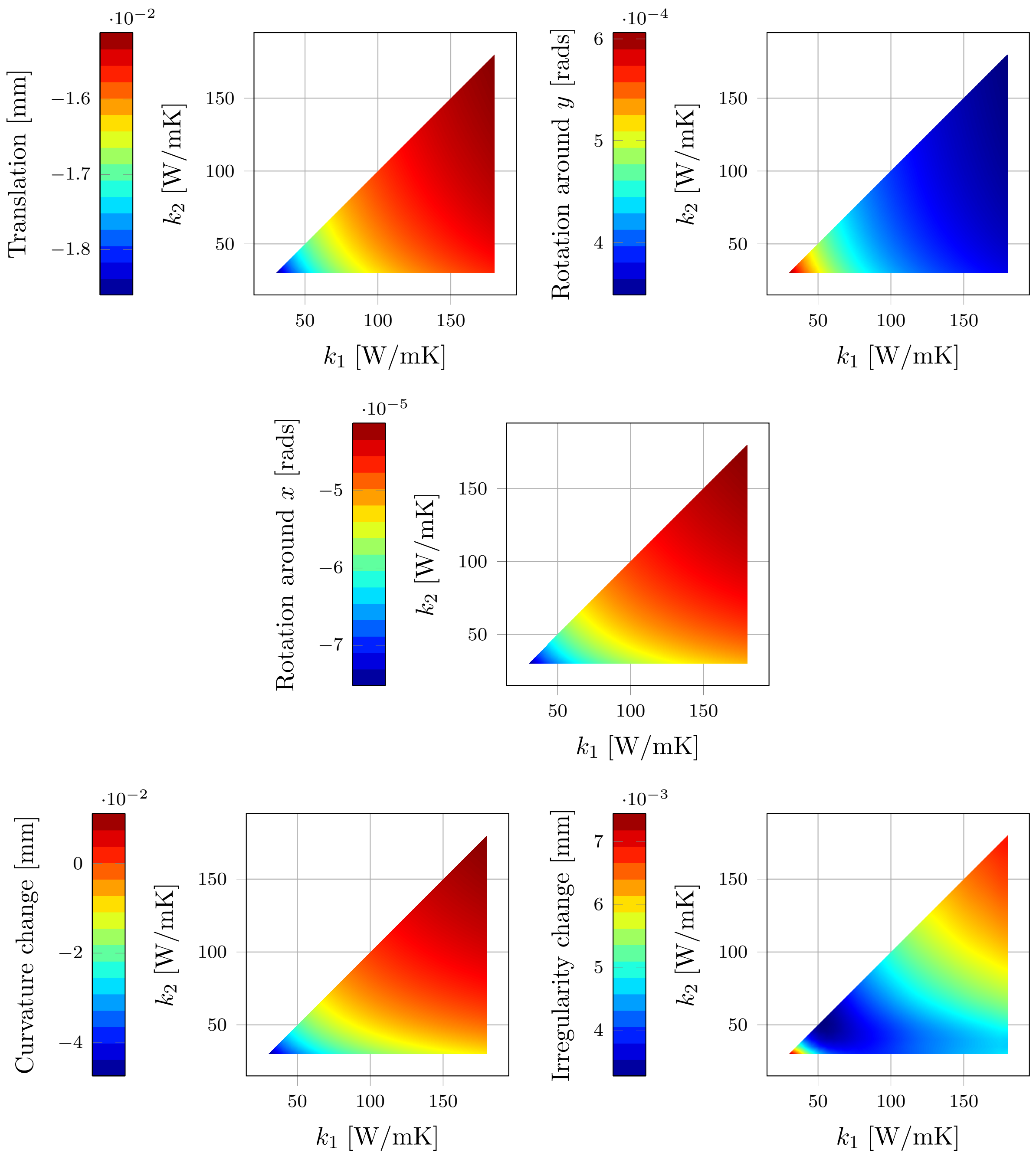


Figure 8.39: Surrogate models for the optical quantities of interest for clamping option A.

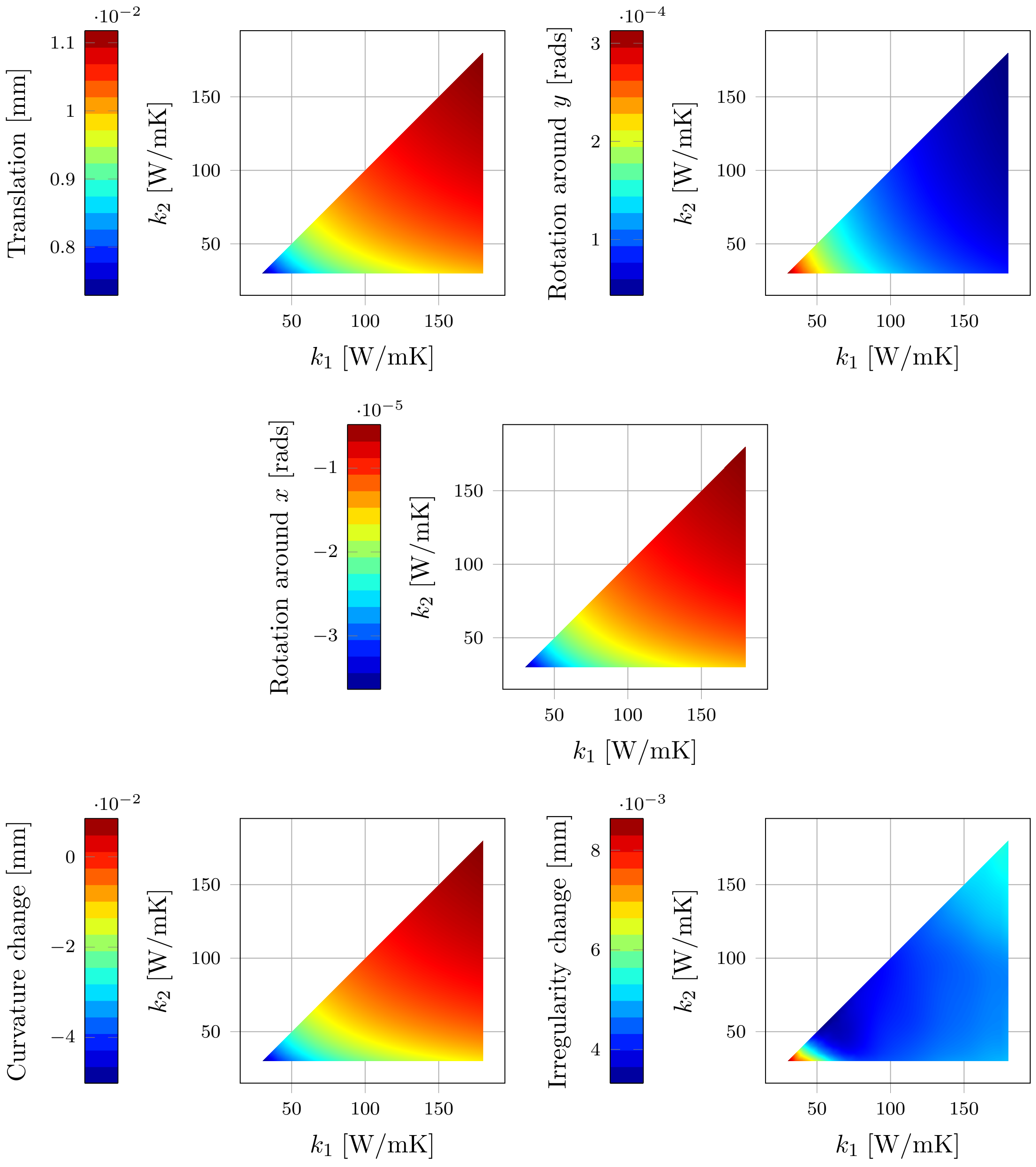


Figure 8.40: Surrogate models for the optical quantities of interest for clamping option B.

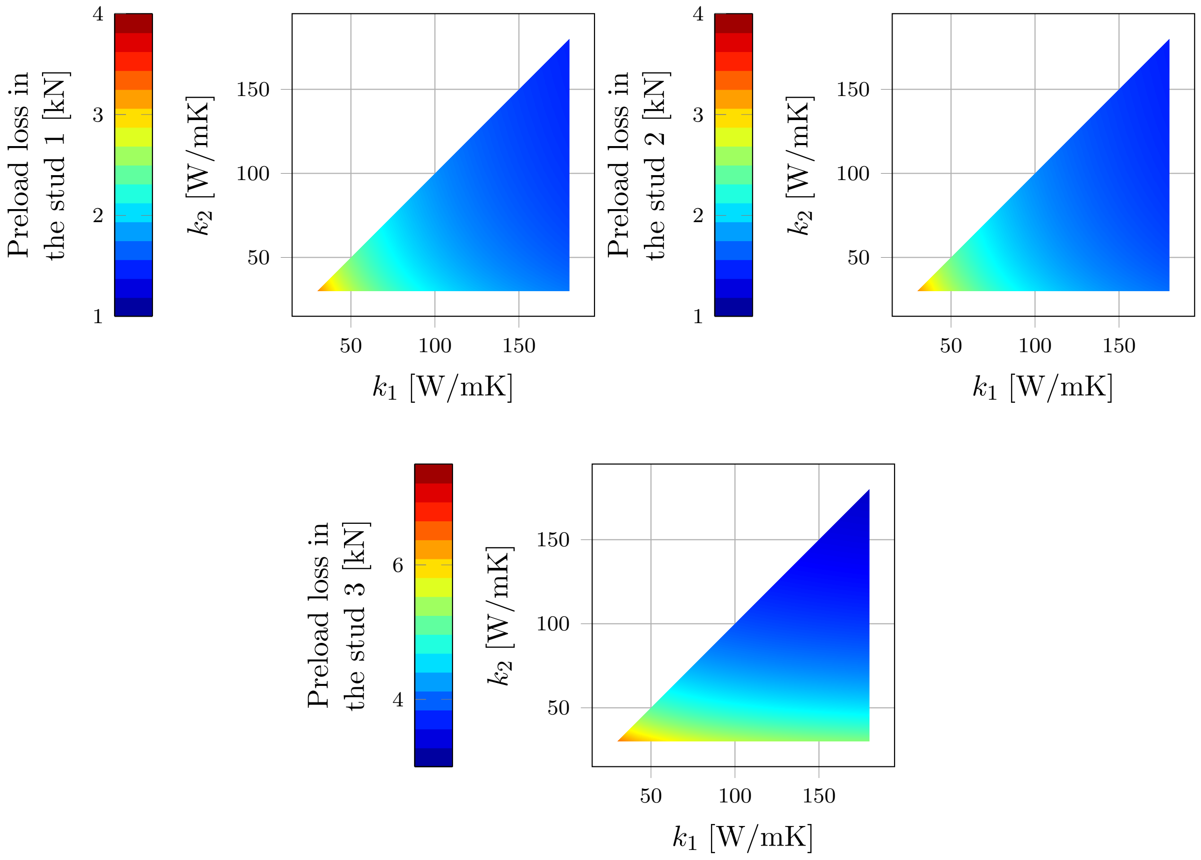
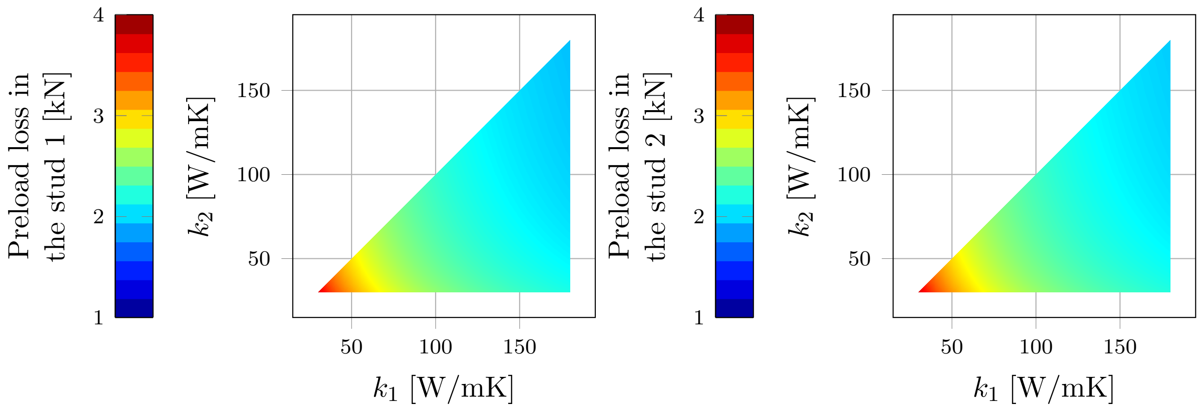


Figure 8.41: Surrogate models for preload loss for clamping option A.



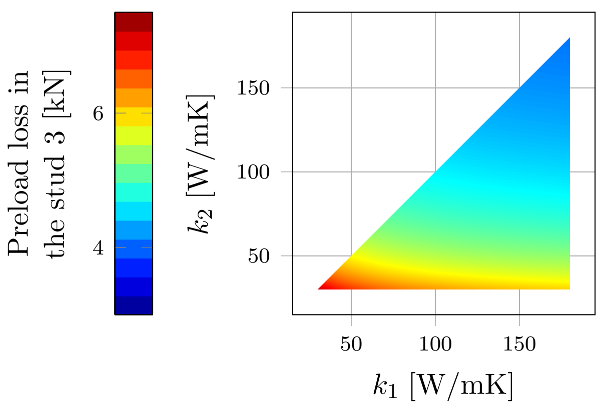


Figure 8.42: Surrogate models for preload loss for clamping option B.

With such surrogate models, we can illustrate how probability of having a blurred image can be computed. The resulting computed probability listed in this section have no engineering meaning as they are computed based on a model problem which does not include all the complexity of the mirror assembly and as the probability density functions of the input parameters are rather arbitrary. To do so, we generate random samples with uniform probability density function on the triangle, evaluate the surrogate model of each optical related quantity of interest. Based on those values, we can evaluate the optical criteria to test whether each sample is blurred or not. After that, the probability of having a blurred image is approximated by the ratio between the number of samples with a blurred image over the total number of drawn samples. With such a strategy, we have a converged probability for a number of 100000 samples on the triangle. The converged probability are 0.915% for option A and 1.410% for option B. Once again, those probabilities have no engineering meaning and are just discussed in order to illustrate the use of the surrogate models.

As previously announced, we observe from Fig. 8.39 and 8.40, that the curvature change and the irregularity change are not very sensitive to the choice of the clamping option. However, the translation and the two rotations are sensitive to the choice of the clamping option. To evaluate position of the reflective surface more precisely, it is required to use a more complex model such as the one as discussed in section 8.5.

8.4 Speed-up results

Finally, we discuss in this section the speed-up of applying ensemble propagation to evaluate the 79 samples. In contrast to previous examples of the use of ensemble propagation discussed in this work, we have here a number of samples which is fixed by the degree d of the considered polynomials. As we have chosen $d = 10$, we have a total of 79 samples which is not a multiple of 32. Therefore, we will discuss speed-up over the first 64 samples and, afterwards, discuss how to evaluate the remaining 15 samples.

8.4.1 Speed-up of the 64 first samples

In this section, we discuss the speed-up of using ensemble GMRES for the clamping option A on 2 compute nodes with two MPI processes per node (one per CPU) of the Blake system described in Appendix A without hyper-threading. This is the first example of this thesis with more than 1 MPI process.

First, as done in the previous examples, we illustrate the speed-up of the different parts of GMRES per iteration in Fig. 8.43. We observe that the speed-ups of the matrix-vector product and the orthogonalization process are consistent with the results of section 7.2: the speed-up of the sparse matrix-vector product is about 1.3 and the speed-up of the orthogonalization process is about 1. However, the speed-up of applying the preconditioner is not directly explained by section 7.2 as the preconditioner of section 7.2 is just some threaded Gauss-Seidel iterations without mutigrid, whereas the mirror example uses a 3-level multigrid preconditioner with a direct solver on the coarsest level. It is, therefore, interesting to illustrate the speed-up of the smoothers on each level as shown in Fig. 8.44. Those curves can be explained with section 7.2: the speed-up of the threaded Gauss-Seidel highly depends on the multigrid level as the multigrid level impacts the matrix size and the matrix size impacts the speed-up of the threaded Gauss-Seidel as discussed in section 7.2. Moreover, we observe that the speed-ups are smaller than the speed-ups of the mesh-tying problem of section 7.3 which is consistent with the fact that the speed-ups decrease with the problem size as illustrated in section 7.2. Another observation is that the speed-up of the smoother on the level 0 decreases from $s = 8$ to $s = 32$ but this is not the case on the level 1. This is linked to the fact that we observed a reduced memory bandwidth for larger memory size. Finally, we observe that the total speed-up of the preconditioner of Fig. 8.43 is mainly deduced by the speed-up of the smoother on the finest level. The speed-up of the two costs outside of GMRES are listed in Fig. 8.45.

We can now discuss the effect of ensemble reduction on the convergence. Once again, ensemble GMRES with reduction increases the required number of iterations to fulfill the stopping criteria compared to ensemble GMRES without ensemble reduction as illustrated in Fig. 8.46. This increased number of iterations results in higher CPU cost for ensemble GMRES with ensemble reduction compared to ensemble GMRES without reduction as listed in table 8.11 and reduced speed-up as listed in table 8.12 and Fig. 8.47. The speed-up of the full simulation is 2.021, 2.032, and 1.763 for $s = 8$, $s = 16$, and $s = 32$ respectively, without ensemble reduction.

In this particular example, we observe that the ensemble size leading to the best speed-up is $s = 16$. This is due to the observed reduced memory bandwidth for larger memory size. If the memory bandwidth was fully independent of the memory size, the speed-up of $s = 32$ would be larger and at least as good as the speed-up of $s = 16$.

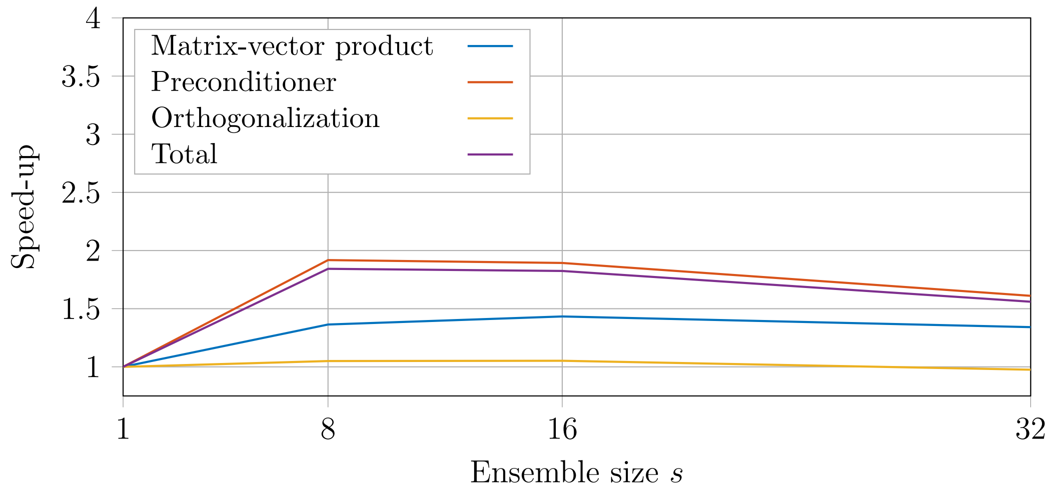


Figure 8.43: Speed-up of the different steps of GMRES

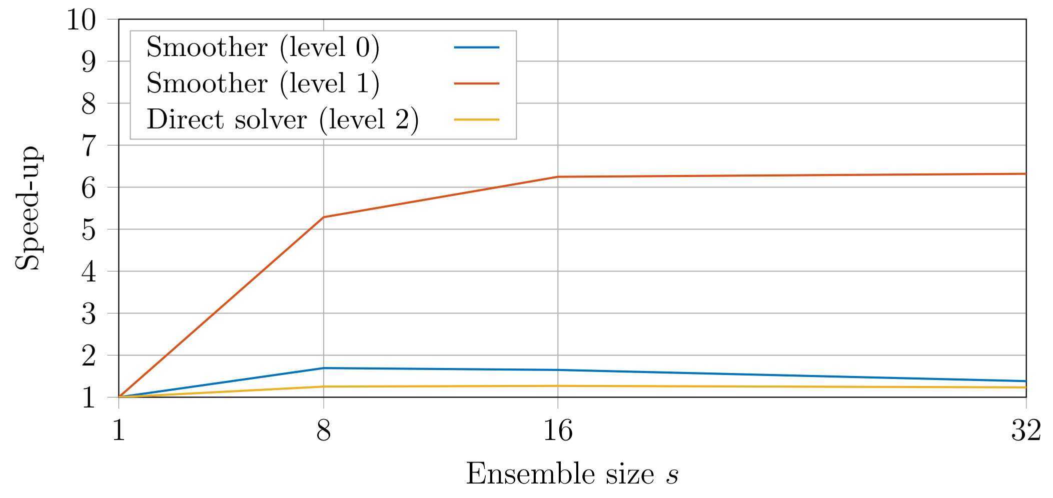


Figure 8.44: Speed-up of the smoothers at the different levels of the preconditioner.

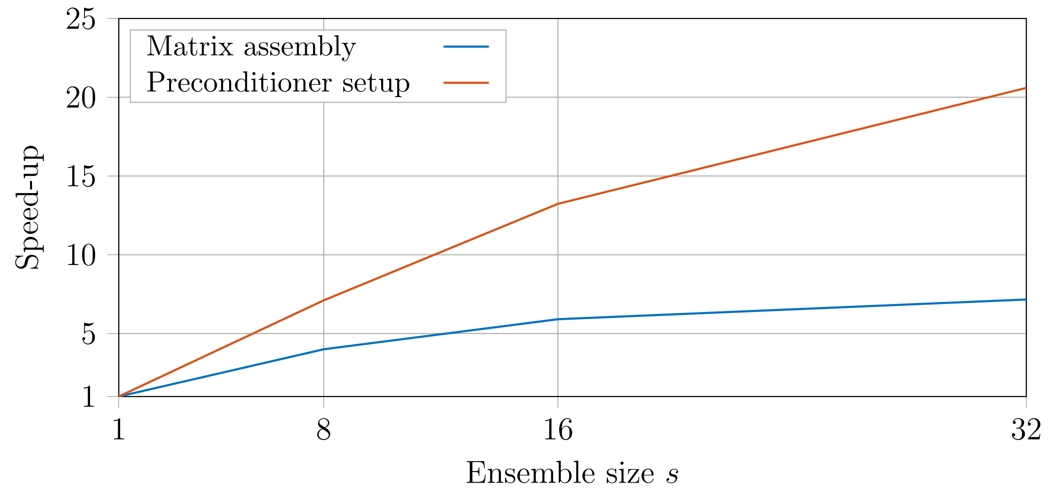


Figure 8.45: Speed-up of the fixed cost.

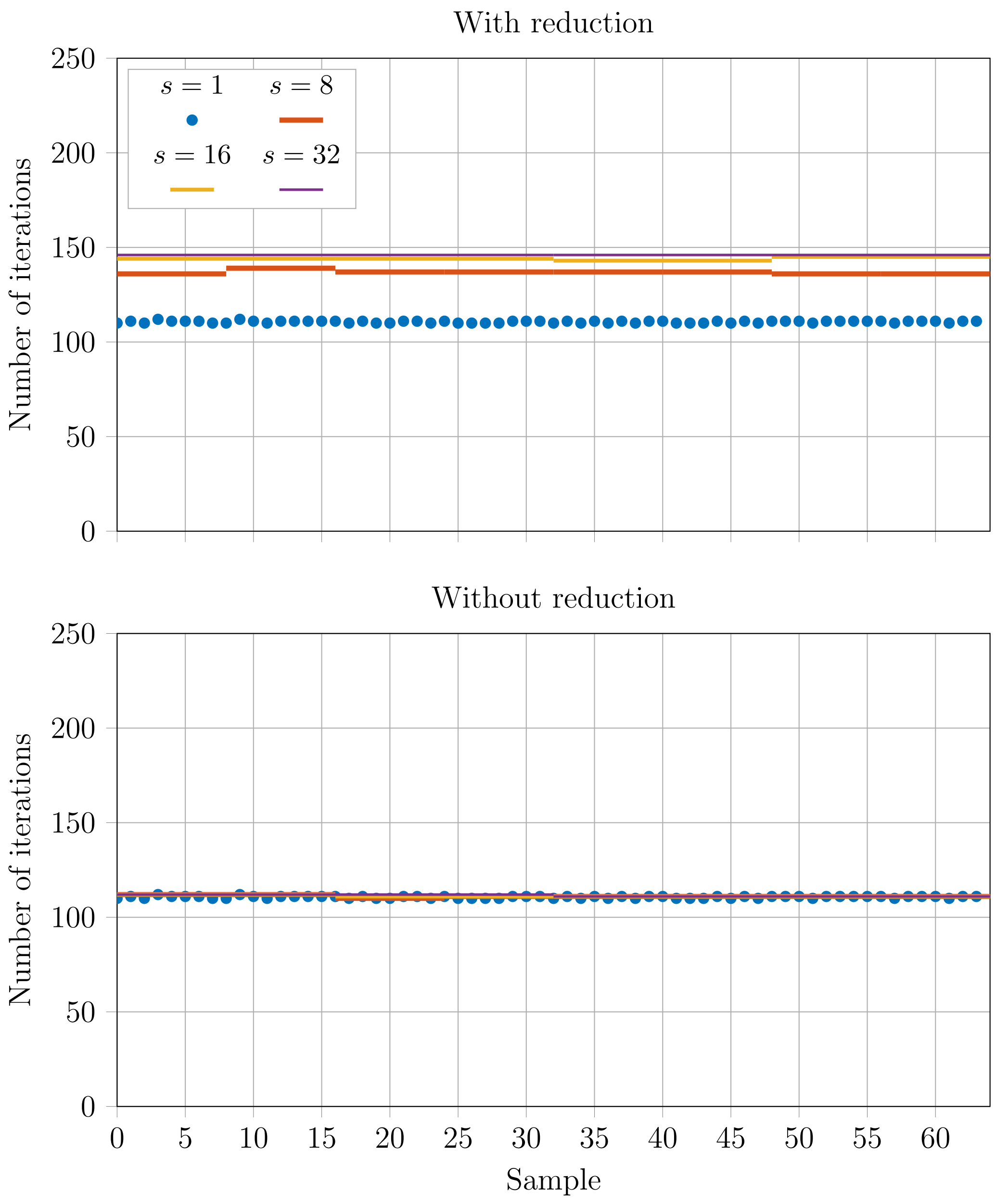


Figure 8.46: Number of iterations to converge with and without ensemble reduction. The blue dots represent the required number of iterations of GMRES to converge for each sample propagated alone. The orange lines, yellow lines, and purple lines represent the required number of iterations of ensemble GMRES to converge with ensembles of size 8, 16, and 32 respectively. We observe that ensemble GMRES without ensemble reduction converges faster than ensemble GMRES with ensemble reduction for all ensembles.

	Tag		With reduction			Without reduction		
Ensemble size		1	8	16	32	8	16	32
Matrix assembly								
Total	II	1.406	2.763	3.850	6.4	2.825	3.900	6.300
Preconditioner setup								
Total	III	0.985	1.124	1.234	1.545	1.123	1.219	1.536
GMRES								
Orthogonalization	IV.X	0.699	8.900	22.083	50.400	5.325	10.655	22.926
Matrix-vector product	IV.IV	0.643	4.702	9.820	21.583	3.788	7.230	15.471
Preconditioner	IV.III	21.880	115.349	253.274	610.662	91.654	186.211	438.057
Level 0 smoother		16.270	96.994	217.273	535.488	76.819	159.257	383.200
Level 1 smoother		3.793	7.171	13.169	26.962	5.659	9.710	19.343
Level 2 smoother		0.575	4.440	9.350	19.592	3.679	7.286	15.012
Total	IV	23.416	131.188	290.821	695.161	101.707	205.985	480.326
Total		27.017	136.513	297.600	705.700	106.950	212.750	490.350

Table 8.11: Average wall-clock time in second per ensemble.

	Tag		With reduction			Without reduction		
Ensemble size		1	8	16	32	8	16	32
Matrix assembly								
Total	II	1.	4.072	5.844	7.031	3.982	5.769	7.143
Preconditioner setup								
Total	III	1.	7.015	12.776	20.409	7.022	12.936	20.536
GMRES								
Orthogonalization	IV.X	1.	0.628	0.506	0.444	1.050	1.049	0.975
Matrix-vector product	IV.IV	1.	1.095	1.048	0.954	1.359	1.424	1.331
Preconditioner	IV.III	1.	1.517	1.382	1.147	1.910	1.880	1.598
Level 0 smoother		1.	1.342	1.198	0.972	1.694	1.635	1.359
Level 1 smoother		1.	4.232	4.609	4.502	5.362	6.251	6.275
Level 2 smoother		1.	1.035	0.983	0.939	1.249	1.262	1.225
Total	IV	1.	1.428	1.288	1.078	1.842	1.819	1.560
Total		1.	1.583	1.453	1.225	2.021	2.032	1.763

Table 8.12: Speed-up.

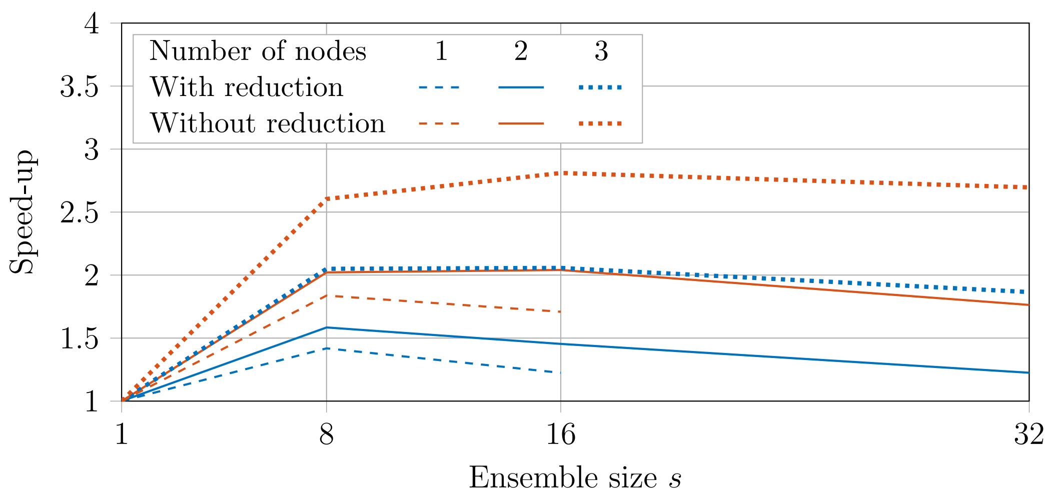


Figure 8.47: Total speed-up of the parametric computation of the mirror problem with and without ensemble reduction on 1, 2, and 4 compute nodes of the Blake system.

8.4.2 Influence of the number of compute nodes

For a fixed problem size, the number of compute nodes allocated for the evaluation of one ensemble influences the speed-up of the simulation. Increasing the number of nodes decreases the problem size per node which results in a smaller memory size used per node and better speed-up of the different components as illustrated in section 7.2. However, the fact that the speed-up increases while increasing the number of compute nodes is not sufficient to decide that more nodes should be used per ensemble. Indeed, it depends on the parallel efficiency E_p [Eijkhout, 2013] of the simulation without ensemble propagation:

$$E_p = \frac{T_1}{pT_p}, \tag{8.12}$$

where p is the number of compute nodes and T_1 and T_p are the wall-clock time to solve the problem on 1 and p nodes respectively. If E_p is small, the speed-up increase will not be sufficient to improve the throughput. Solving one ensemble of size s on p compute nodes is faster than solving ps samples without ensemble propagation on p nodes with an embarrassingly parallel strategy where each node computes s samples one by one if:

$$E_p\, S_{sp} > 1, \tag{8.13}$$

where S_{sp} is the speed-up associated to ensemble propagation which depends both on the ensemble size s and the number of nodes p.

To illustrate that, we have done the parametric computations with $p = 1$, 2, and 4 and list the wall-clock time per second per ensemble in table 8.13. It has not been possible to run the simulation with $s = 32$ and $p = 1$; the simulation has not been able to allocate enough contiguous memory to store the Arnoldi vectors of ensemble GMRES. Based on those wall-clock time measurements, we can compute the parallel efficiency E_p without ensemble propagation and the speed-up S of the different ensemble sizes. Those results are listed in table 8.14 and the speed-ups are illustrated in Fig. 8.47. If we only use the parallel efficiency to select the number of nodes, we would have chosen $p = 1$ whereas, if we only use the speed-up to select the number of nodes, we would have chosen $p = 4$ or even higher. However, taking both information into account leads to the results of table 8.15 where we observe that the optimal strategy to follow here is $p = 2$ with $s = 16$ and without ensemble reduction. Finally, even if we do not choose the optimal parameters $p = 2$ and $s = 16$, using embedded ensemble propagation reduces the total wall-clock time as all the speed-up of table 8.14 are greater than 1.

		E_p	With reduction			Without reduction		
p \ s	1		8	16	32	8	16	32
1	52.128	-	294.550	682.325	-	227.550	489.275	-
2	27.017	0.965	136.513	297.600	705.700	106.95	212.75	490.350
4	19.072	0.683	75.525	149.575	327.200	58.588	108.700	226.400

Table 8.13: Average total wall-clock time in seconds per ensemble.

		E_p	With reduction			Without reduction		
p \ s	1		8	16	32	8	16	32
1	1	-	1.416	1.222	-	1.833	1.705	-
2	1	0.965	1.583	1.453	1.225	2.021	2.032	1.763
4	1	0.683	2.020	2.040	1.865	2.604	2.807	2.696

Table 8.14: Speed-up S_{sp} and the efficiency E_p is indicated in the third column.

		With reduction			Without reduction		
p \ s	1	8	16	32	8	16	32
1	1	1.416	1.222	-	1.833	1.705	-
2	0.965	1.528	1.402	1.182	1.950	**1.961**	1.701
4	0.683	1.380	1.393	1.274	1.779	1.917	1.841

Table 8.15: Product of the efficiency and the speed-up E_pS_{sp}. We observe that the fastest strategy to solve this parametric computation is to use $s = 16$ and 2 compute nodes per ensemble.

		With reduction			Without reduction		
p \ s	1	8	16	32	8	16	32
1	834.048	589.017	682.527	-	455.018	489.178	-
2	864.298	545.843	594.899	705.624	705.624	425.318	490.328
4	1221.154	604.383	598.742	654.669	468.830	435.080	453.041

Table 8.16: Total wall-clock time in seconds to evaluate the 64 samples on a system with 4 compute nodes. The total time is equal to $52.128 \times 64/(4 \times E_pS_{sp})$.

8.4.3 Strategy to solve the remaining samples

Given a fixed ensemble size s and a total number of samples n, there are two ways to evaluate the $\text{mod}(n, s)$ last samples. Either, we can loop over them one by one propagating them without ensemble propagation, or we can add $s - \text{mod}(n, s)$ identical samples to complete an ensemble of size s. The most effective approach depends on $\text{mod}(n, s)$ and the speed-up S. If the speed-up S is smaller than the ratio between s and $\text{mod}(n, s)$, the additional cost of evaluating the identical samples is not amortised by the speed-up of ensemble propagation and it is more effective to propagate each sample one by one. Otherwise, if the speed-up S is larger than the ratio between s and $\mathrm{mod}(n, s)$, it is better to propagate those samples with some identical samples.

If s is sufficiently large such that smaller ensemble can be propagated, we may want to reduce the ensemble size. For example, if s is 32 and $\mathrm{mod}(n, s)$ is 8 or 16, it will probably be more effective to propagate the last remaining samples inside an ensemble of smaller size. Given s and n, we can compute the most effective ensemble size $s^\star$ for the remaining samples as follows:

$$s^\star = \arg\min_{x\in[1,s]} \frac{\left\lceil \frac{\mathrm{mod}(n,s)}{x} \right\rceil x/\mathrm{mod}(n, s)}{S(x)}, \tag{8.14}$$

where $\left\lceil \frac{\mathrm{mod}(n,s)}{x} \right\rceil$ is the number of ensembles of size x needed to solve the $\mathrm{mod}(n, s)$ last samples and $S(x)$ is the speed-up of ensemble of size x.

In practice, we typically do not have access to $S(x)$ beforehand, therefore, to deduce $s^\star$ it is required either to run some tests with different ensemble sizes or to use data from previously run examples.

8.5 Speed-up on the full assembly

In this section, we consider embedded ensemble propagation on the full assembly illustrated in Fig. 8.2 in order to illustrate the performance on a larger problem. The holder and the pipes are in Stainless steel 55316 (Table 8.3) and the bolts are in Inconel 718 (Table 8.6). The mesh used has 2 288 050 nodes leading to 9 152 200 degrees of freedom per sample and 12.8 millions of tetrahedra. As the problem has about 7 times more degrees of freedom compared to the previous case, we have tested ensemble GMRES on 16 compute nodes instead of 2 to increase the available memory. The temperature field before irradiation is illustrated in Fig. 8.48.

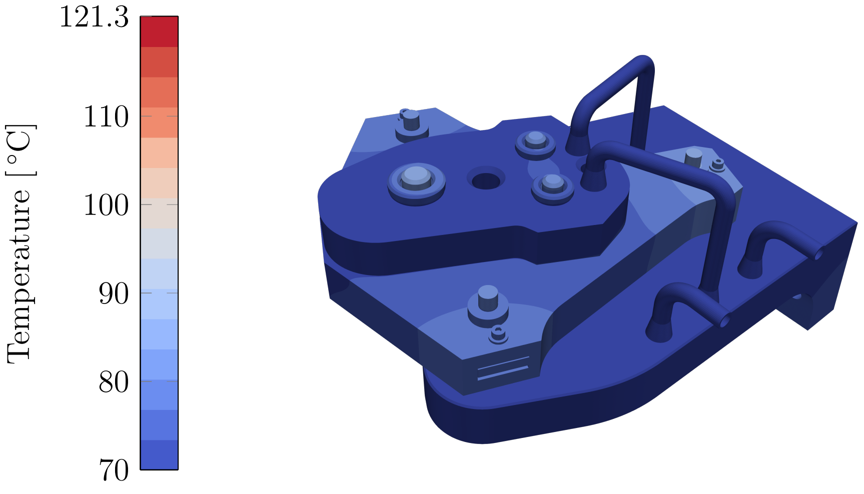


Figure 8.48: Temperature of the full assembly before irradiation.

For this example, we use the preconditioner as described in section 8.2.6 and in table 8.17. The tolerance of the stopping criteria of the three first criteria have been updated to 10^{-5}, 10^{-7}, and 10^{-5} and the last criterion has be dropped.

level	rows	non-zero entries	non-zero entries per row	coarsening ratio	level smoother
0	9 152 200	435 215 794	47.55		BGS
1	509 636	23 592 647	46.29	17.96	BGS
2	35 304	1 470 196	41.64	14.44	Klu

Table 8.17: Multigrid information of the preconditioner.

The resulting wall-clock time in second on the first 32 samples are listed in table 8.18 without ensemble reduction and the speed-ups are listed in table 8.19 and are illustrated in Fig. 8.49. We observe that the speed-ups are similar with the speed-ups of the previous case: the speed-up of the orthogonalization process is about 1, the speed-up of the sparse matrix-vector product is about 1.35 , but the speed-up of the preconditioner is a bit smaller for this example. This is explained by the fact that the size of the coarsest level is larger in this case leading to a relative larger wall-clock time spend applying the Klu solver as listed in table 8.18 which has the poorest speed-up as illustrated in table 8.19. The speed-up of this example can be improved using a 4–level multigrid preconditioner instead of this 3–level multigrid preconditioner. The maximal total speed-up is close to 2, as in the previous case, but with $p = 16$ and $s = 8$.

	Tag				
Ensemble size		1	8	16	32
Matrix assembly					
Total	II	1.600	3.000	3.600	7.200
Preconditioner setup					
Total	III	1.167	1.274	1.404	1.619
GMRES					
Orthogonalization	IV.X	2.699	22.045	43.623	88.367
Matrix-vector product	IV.IV	1.654	9.574	18.507	39.423
Preconditioner	IV.III	67.049	313.855	709.347	1405.960
Level 0 smoother		40.043	197.311	475.509	953.720
Level 1 smoother		13.477	19.612	30.640	58.476
Level 2 smoother		9.850	81.298	166.614	337.892
Total	IV	72.518	351.332	792.731	1557.679
Total		89.500	369.900	811.300	1582.800

Table 8.18: Average wall-clock time in second per ensemble for the full mirror assembly.

	Tag				
Ensemble size		1	8	16	32
Matrix assembly					
Total	II	1.	4.267	7.111	7.111
Preconditioner setup					
Total	III	1.	7.327	13.296	23.051
GMRES					
Orthogonalization	IV.X	1.	0.979	0.990	0.977
Matrix-vector product	IV.IV	1.	1.382	1.430	1.342
Preconditioner	IV.III	1.	1.709	1.512	1.526
Level 0 smoother		1.	1.624	1.347	1.344
Level 1 smoother		1.	5.497	7.038	7.375
Level 2 smoother		1.	0.969	0.946	0.933
Total	IV	1.	1.651	1.464	1.490
Total		1.	1.936	1.765	1.809

Table 8.19: Speed-up for the full mirror assembly.

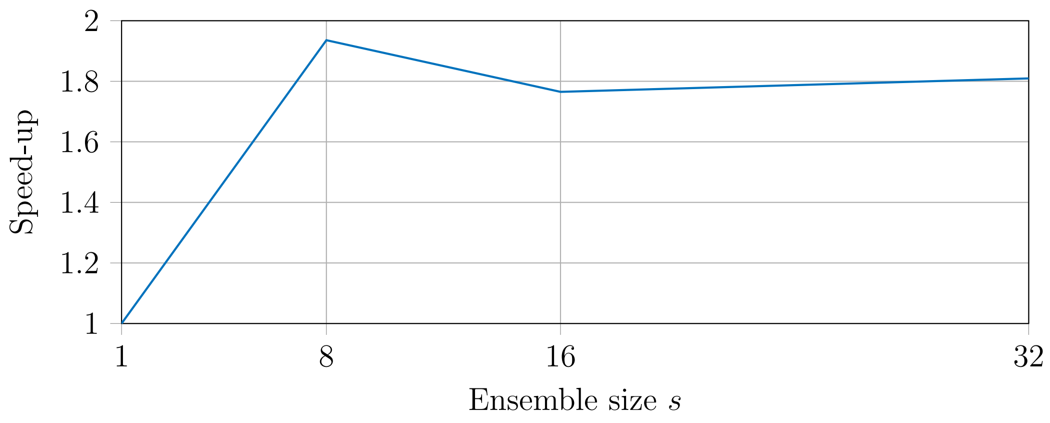


Figure 8.49: Total speed-up of the full mirror assembly.

8.6 Conclusions

In this chapter, we have investigated the effect of the irradiation of AlN ceramic on the optical measurement using a model problem and a surrogate-based approach. Moreover, we have used ensemble propagation in order to reduce the wall-clock time of the largest CPU cost of the method: the wall-clock time needed to evaluate the high-fidelity model to train and build the surrogate model. With ensemble propagation, we have divided the CPU cost by a factor 2 and, therefore, have multiplied the throughput of the parametric computation by 2 taking as reference a GMRES without ensemble propagation with state-of-the-art computational kernels. We have discussed the impact of the number of MPI processes on the speed-up. We have illustrated similar speed-up on a larger example with about 10^7 degrees of freedom.

Finally, we have shown that the two clamping options provide similar deformation of the mirror surface but different position of this surface. We have discussed the need of using a more complex model to evaluate this position more precisely.

Chapter 9

Conclusions

The work presented in this thesis aimed to accelerate the parametric computations of problems with non-symmetric or indefinite matrices on high performance computing architectures. To do so, we have contributed to embedded ensemble propagation by introducing ensemble GMRES without reduction, implementing it, and comparing it with the already existing ensemble GMRES with reduction.

9.1 Summary

In this work, we have extended ensemble GMRES not to use ensemble reduction. To do so, we started by describing the possible occurrence of ensemble divergence inside ensemble GMRES. After that, we described formally the already existing approach to manage those occurrences of ensemble divergence: ensemble GMRES with ensemble reduction. In particular, we discussed how the reduction impacts the convergence of ensemble GMRES. Thereafter, we introduced an alternative approach: ensemble GMRES without ensemble reduction. For this second approach, we tackled both the function-call divergence and the control-flow divergence of ensemble GMRES without reduction. The function call divergence was solved implementing the tensor contraction as a GEMV for the ensemble type. We developed a simple high-performing GEMV implementation for ensembles. We highlighted the impact of ensemble propagation on GEMV and, in particular, the impact of the ensemble size on the optimal tile size. This resulted in an equivalent wall-clock time per iteration of GMRES for both methods, i.e. with and without reduction.

The control-flow divergence was solved using masks to perform masked assignment and logical reduction. Alongside this work, we revisited the mathematical formulation of ensemble propagation using tensor notation.

The two approaches, i.e. the existing ensemble GMRES with ensemble reduction and ensemble GMRES without ensemble reduction as introduced in this work, have been compared on four academic problems and a thermomechanical simulation. The examples showed that the wall-clock time of one iteration of ensemble GMRES was independent of the use of the reduction and illustrated a faster convergence of ensemble GMRES without reduction. Those two points led to improved performance of ensemble propagation without ensemble reduction as ensemble reduction increases the number of iterations required to converge.

Finally, we have illustrated the speed-up of ensemble GMRES on one industrial problem relevant for the design of an optomechanical system for ITER and have illustrated an increased throughput due to the use of ensemble GMRES compared to not using ensemble propagation. Using ensemble GMRES allowed to reduce the wall-clock time of evaluating the high-fidelity model for the different samples by up to 50%.

9.2 Directions for future work

- Preconditioner for the mesh-tying problem: in the mesh-tying problem, we have used Gauss-Seidel iterations as smoother for the stiffness matrix which is not positive definite but is positive semi-definite. For this kind of matrix, we should investigate better options such as using an augmented Lagrange formulation to have a symmetric positive definite block 00 matrix [Benzi and Wathen, 2008]. This would enforce the convergence of the Gauss-Seidel iterations used as smoother.

- Memory bandwidth: we observe a reduced memory bandwidth when streaming large arrays on Blake running the STREAM Triad Memory Bandwidth benchmark [McCalpin, 1995] with large vectors. This effect varies from one node to another and starts typically when streaming arrays of about 30 Gb. It would be interesting to better understand this observation as it has a direct impact on the performance of embedded ensemble propagation for large problems.

- Investigate preconditioner strategies for contact problems with singular stiffness matrix. This investigation will allow to consider the contact constraints in the mirror example and to investigate the contact losses. This is closely related to the first direction presented in this section.

9.3 Broader directions for future work

9.3.1 Grouping strategy for non-linear problems

As illustrated with the beam contact problem, grouping strategies can improve the performance of ensemble GMRES. In fact, this might even be more important for ensemble GMRES than the ensemble CG as extra iterations of ensemble GMRES are getting more and more expensive where the extra iterations of the ensemble CG have a fixed cost per iteration. Therefore, grouping strategies for ensemble GMRES should be studied as a way to improve even further the performances.

To apply efficiently embedded ensemble propagation requires the study of the grouping of samples in ensembles of size s that minimizes ensemble divergence as discussed in [D'Elia et al., 2020] for linear systems. However, when we consider non-linear problems or transient problems, there is more than one linear system per evaluation of the model. Such a study requires first the definition of a metric that measures ensemble divergence of a given ensemble. As a starting point, we suggest associating to each sample ℓ a vector $\mathbf{c}_\ell$ in $\mathbb{N}^n$ where $\mathbb{N}$ is the set of natural numbers and n is the maximal number of linear systems in the simulation (the maximum number of time steps times the maximum number of non-linear steps per time step) such that its entries $c_{\ell 1}, \ldots, c_{\ell n}$ correspond to the number of iterations of the Krylov method associated with each corresponding linear solve. Using those vectors, given two samples ℓ_1 and ℓ_2, it is possible to compute the total number of iterations of the Krylov method where only one of ℓ_1 and ℓ_2 has not converged by computing $\|\mathbf{c}_{\ell_1} - \mathbf{c}_{\ell_2}\|_1$ where $\|.\|_1$ is the l_1–norm. As a consequence, we can introduce a first measure of ensemble divergence d of a given ensemble of samples $\{\ell_1, \dots, \ell_s\}$ as:

$$d = \max_{(i,j)\in\{1,\dots,s\}^2} \|\mathbf{c}_{\ell_i} - \mathbf{c}_{\ell_j}\|_1. \tag{9.1}$$

Using this first definition, we can see the grouping problem as a discrete minimization problem with l_1–norm during which we seek a combination of the samples that minimizes the average divergence.

However, the vector $\mathbf{c}_\ell$ is not known a priori and, therefore, we suggest using a surrogate model to estimate $\mathbf{c}_\ell$ based on the previous computation as done in [D'Elia et al., 2020] for linear systems.

A difficulty with this approach is that the estimation of $\mathbf{c}_\ell$ for the first samples are not accurate as the surrogate model use for the estimation has not been trained with a sufficiently large number of samples. Therefore, a possibility would be to not fix the ensemble size s during the simulation, starting with a small s while the surrogate model is imprecise to reduce the probability of having a bad speed-up due to ensemble divergence, and increasing the ensemble size at the end when the surrogate model predicts ensemble divergence accurately.

9.3.2 GMRES

It would be interesting to investigate more the influence of the preconditioner on the convergence of ensemble GMRES with ensemble reduction. In particular, it would be interesting to draw recommendations on the choices of the preconditioner and its parameters to reduce the differences between the spectra to improve the convergence of the coupled samples. It would help to understand better when ensemble reduction has a critical impact on the convergence of ensemble GMRES.

Another interesting work is to study the link between ensemble GMRES without reduction with the block GMRES method [Saad, 2003] which uses the Krylov subspace:

$$\mathcal{K}_m(\boldsymbol{A}, \boldsymbol{R}^{(0)}) \equiv \text{span}\left\{\boldsymbol{R}^{(0)}, \boldsymbol{A}\boldsymbol{R}^{(0)}, \dots, \boldsymbol{A}^{m-1}\boldsymbol{R}^{(0)}\right\}. \tag{9.2}$$

In ensemble GMRES without reduction, we use the Krylov subspace (3.24) (not taking into account a preconditioner):

$$\mathcal{K}_m(\mathscr{A}, \boldsymbol{R}^{(0)}) \equiv \text{span}\left\{\boldsymbol{R}^{(0)}, \mathcal{A}(\boldsymbol{R}^{(0)}), \dots, (\mathcal{A})^{m-1}(\boldsymbol{R}^{(0)})\right\}. \tag{9.3}$$

These subspaces are identical if $\boldsymbol{A}$ is not sample dependent. It would be interesting to understand the link between the two subspaces when $\boldsymbol{A}$ is sample dependent. This would allow the possibility to use existing developments studied in the context of the block GMRES method for ensemble GMRES with ensemble reduction.

9.3.3 Implementation of the remaining kernels

There are other kernels than the GEMV that require to be implemented efficiently for ensemble propagation such as the level 3 BLAS function GEMM or LAPACK functions. To have a portable and efficient implementation of those functions is important to allow the user of embedded ensemble propagation use algorithms with ensemble reduction and without ensemble reduction without having to implement themselves the kernels. Such research and implementation work should be carried out at the KokkosKernels level and is closely related to the Compact BLAS research [Kim et al., 2017]. This will allow the efficient use of other Krylov methods such as the block GMRES with ensemble propagation.

9.3.4 Scheduling and load balancing

In this thesis, we have highlighted that ensemble propagation improves the throughput of parametric computations. Moreover, we have illustrated that the saved wall-clock times depend on the size of the problems per compute node and on the number of threads used. Therefore, another direction for future research is, for a given parametric computation and a given computational system, to study the optimal strategy to evaluate the samples. Is it better to have a larger speed-up reducing the size of the computational model per node or is it better to have more instances of the computational model running in parallel?

Such a study and the need of grouping strategy are both opportunities to study the scheduling strategy of the instances of the computational models with ensemble propagation. Künzner et al. [2019] proposed a strategy to minimize the idle CPU time of the embarrassingly parallel evaluations of a computational model, without ensemble propagation, using a surrogate model to predict the wall-clock time of a new evaluation of the model. They have illustrated a speed-up of about 2 on a real-world evacuation scenario in pedestrian dynamics. It would be interesting to combine such a strategy with the use of embedded ensemble propagation with grouping strategy to combine both speed-ups.

9.3.5 Adaptive ensemble size

As discussed in this work, ensemble divergence can occur during loops such as time integration. It is possible that at some point the iterative method has converged for a subset of the samples but not for the remaining. A possible way to improve the performance of the remaining iterations is to drop the already converged samples and to use an ensemble of smaller size or even to propagate one sample alone. Doing so has, however, an overhead cost due to the need of reindexing the samples of the ensemble and modifying the layout of the data accordingly. Whether the fact that this strategy is interesting or not in the sense of the wall-clock time will depend on the amount of the memory that has to be reordered and on the number of remaining iterations. It would be interesting to study criteria to decide whether this strategy is interesting or not for a given architecture.

9.3.6 Other non-linear problems

The first direction for future work is to consider other non-linear problems or other types of non-linearity with ensemble propagation. For example, in the example of the front mirror of the ITER diagnostic system, we might want to take into account influence of the temperature on the material properties; this results in a non-linear thermomechanical problem. Typically a Newton–Raphson strategy is typically used to solve those non-linear problems. Therefore, it is necessary to compute the derivative of functions such as the constitutive laws which are functions of the temperature. The computation of such derivatives with ensemble propagation can be done using automatic differentiation [Phipps and Pawlowski, 2012]. There is, however, one challenge doing so with ensemble propagation; they are new possible occurrences of ensemble divergence. The influence of the temperature on the material properties are typically given to FEM software defining a table of empirical experiments with some temperature and the value of the material properties at those temperatures. The FEM software can, using those data, create a piecewise-defined linear function which interpolate those data to have a function that can be evaluated for any temperature. Such approach leads to control-flow divergence as samples of an ensemble might not be in the same interval of the function's domain.

9.3.7 Time integration

Another direction for future work is to consider time integration scheme with ensemble propagation. In the simplest cases, the samples of an ensemble can share the same time discretization and the time integration with ensemble propagation will just add an extra loop without any extra ensemble divergence. However, if time adaptive methods are considered some questions are risen: should we enforce the same time step for all the samples and therefore take the smallest of the adapted time step? Should we allow the algorithm to have different time steps for the different samples? Those questions impact the total number of linear systems to solve and, therefore, the speed-up of using ensemble propagation.

9.3.8 Contact problem with friction

Using frictional contact conditions, two bodies at a closed surface can either be sticking or sliding in the tangential direction. There are two conditional statements: Is the contact locally closed? Is the locally closed contact sliding or sticking? Therefore, considering frictional contact conditions introduce a new occurrence of ensemble divergence due to this last conditional statement.

9.3.9 Contact problem with large displacement

In the case of the large sliding contact problems, the fact that potential contact interfaces are a priori unknown and sample dependent adds another occurrence of such ensemble divergence. This ensemble divergence is challenging as it is the first case where meshes are sample dependent and is challenging for both the computation of the entries and their storage.

The computation of the Mortar matrices is based on three steps: the global search of segment-to-segment interactions, the projection and the clipping of a segment on the other, and the integration on the clipped polygon. These processes introduce ensemble divergence as two elements can have a segment-to-segment interaction for a given sample but not for another one or the clipped polygon can have different shapes for every sample.

Following the same approach as the one in section 5.1.7 for the storing of the entries is doable but it will fill a large amount of the memory with zeros as the feasible graphs are more different.

The first step is to study the global search for contact element pairs that can be touching. This step can be efficiently implemented using a strategy based on hierarchical global search trees and bounding volumes as described in [Yang and Laursen, 2008; Popp et al., 2013a; Hansen et al., 2016]. Moreover, there is already such an implementation based on Kokkos at Sandia National Laboratories made by Hansen et al. [2016]. Such a strategy can be extended to ensemble propagation as samples will have the same tree topology, but different tree node information as the node connectivity of the mesh is assumed to be sample independent. Therefore, it makes sense to build one tree per contact face for one ensemble. The search across those trees can then be either computed for each sample individually or using ensembles but dealing with ensemble divergence. The first approach would minimize the amount of work that has to be done while the second would possibly improve the throughput. Those tree strategies are used in fields such as ray tracing and video games and are well suited to be computed on GPU. Thus, the computation of Mortar matrices can be seen as an opportunity to use the heterogeneity of the compute node: while the CPU is computing the stiffness matrices of the bodies, the GPU can compute the Mortar matrices.

The next steps, the projection, the clipping, the triangularization, and the integration are challenging too in the context of ensemble propagation as all of them are impacted by ensemble divergence. However, those steps can be implemented with small size tensor computations with ensemble propagation.

9.3.10 Ice sheet problems

Another direction for future work is to apply the developed ensemble GMRES without ensemble reduction on ice sheet problems seen as contact problems [Bosten, 2019]. Those problems are of particular interest for the research group of the supervisor of this thesis.

Appendices

Appendix A

Blake

Blake is a HPC server located at Sandia National Laboratories, Albuquerque. It consists in 40 compute nodes connected with Intel OmniPath and made of 2 Intel(R) Xeon(R) Platinum 8160 CPU at 2.10GHz per node [Hammond et al., 2018]. Each Intel(R) Xeon(R) Platinum 8160 CPU has 24 cores which supports hyper-threading and AVX-512. Each node has 192GB of 2666MT/s DDR4 memory (96GB with 6 memory channels per CPU).

The maximal memory bandwidth is therefore $2.666 \times 8 \times 6 = 127.97$ GB/s per NUMA region.

The cache hierarchy is listed in Table A.1.

Level	Size	Associativity	Distributed	Total size
L1	32kB	8-way	No	768kB
L2	1MB	16-way	No	24MB
L3	33MB	11-way	Yes	33MB

Table A.1: Cache hierarchy

Appendix B

Discussion on the speed-up

In this section, we review the definition of the speed-up of ensemble propagation, discuss how it can be compared to the embarrassingly parallel strategy, and illustrate it on an example.

The reduction of the total wall-clock time due to embedded ensemble propagation is measured by the notion of speed-up defined in [Phipps et al., 2017] as:

$$S = \frac{\sum_{\ell=1}^{s} T^{(\ell)}}{T^{(e)}}, \tag{B.1}$$

where, for a given ensemble, s is the ensemble size, $T^{(\ell)}$ is the wall-clock time of evaluating the sample ℓ alone, and $T^{(e)}$ the wall-clock time of evaluating the ensemble. As discussed in section 2.1, this definition assumes that the CPU costs $T^{(\ell)}$ and $T^{(e)}$ have been measured on the same CPU power, i.e. the same number of compute nodes, with the same number of MPI process per node, and the same number of threads per MPI process. Therefore, this speed-up does not bring information on how faster embedded ensemble propagation is compared to running multiple samples in parallel on a same compute node for instance.

In this section, we illustrate that embedded ensemble propagation is still faster than the previously mentioned strategy on an example. The considered example is the example of section 7.2 (tensile test on a cube) with a $10 \times 10 \times 300$ mesh and 200 iterations of GMRES.

We considered 4 strategies on the Blake system:

1. The serial reference test: running 1 sample on 1 CPU of a compute node of the Blake system with 1 thread,

2. The threaded reference test: running 1 sample on 1 CPU of a compute node of the Blake system with 24 threads,

3. Embarrassingly parallel strategy: running 1 sample per core on 1 CPU of a compute node of the Blake system with 1 thread,

4. Embedded ensemble propagation: running 1 ensemble of size 8 on 1 CPU of a compute node of the Blake system with 24 threads.

The wall-clock time of the different parts are listed in table B.1 and the corresponding speed-up, with the second strategy as the reference, are listed in table B.2. We observe that the embarrassingly parallel strategy and embedded ensemble propagation have nearly the same speed-up for the orthogonalization; this was expected as this computation is limited by the memory bandwidth and because both approaches have to stream the same amount of memory. However, the speed-up of all the remaining parts are larger for the ensemble propagation approach. In particular, the matrix-vector product has a larger speed-up for ensemble propagation even if this kernel is limited by the memory bandwidth too. This is explained by the data of the graph reused as discussed in section 7.2.1; even if both approaches, i.e. the embarrassingly parallel strategy and embedded ensemble propagation, are both limited by the memory bandwidth, embedded ensemble propagation is faster as it needs to stream less data than the embarrassingly parallel strategy. The speed-up of parts of the code which are less impacted by the memory bandwidth such as the matrix assembly or the preconditioner setup are larger for embedded ensemble propagation.

Overall, although the orthogonalization process reduces the difference between the wall-clock time of the embarrassingly parallel strategy and embedded ensemble propagation, embedded ensemble propagation is faster on the Blake system for this test.

Finally, as illustrated in this example the value of the speed-up does not tell us how fast is embedded ensemble propagation compared with the embarrassingly parallel strategy. It does tell us how fast is embedded ensemble propagation compared with running each sample one at a time with the same CPU power. Taking the embarrassingly parallel strategy as the reference for the speed-up computation would reduce the speed-up from 1.577 to 1.287 in this case. Therefore, using the embarrassingly parallel strategy as the reference could reduce the measured speed-up included in this thesis, however, this strategy requires more memory than both the threaded reference strategy and the embedded ensemble propagation. Depending on the memory requirement of the test considered, this can increase the speed-up.

	Tag				
Strategy number		1	2	3	4
Strategy		Serial reference test	Threaded reference test	Embarrassingly parallel strategy	Embedded ensemble propagation
Matrix assembly					
Total	II	432.000	28.800	18.300	12.300
Preconditioner setup					
Total	III	335.650	19.212	16.947	2.635
GMRES					
Orthogonalization	IV.X	503.182	62.767	55.010	57.087
Matrix-vector product	IV.IV	161.054	18.168	15.966	13.619
Preconditioner	IV.III	955.258	136.867	122.182	97.046
Total	IV	1626.334	223.277	196.811	168.880
Total		2474.400	295.200	240.900	187.200

Table B.1: Average wall-clock time in second to evaluate 24 samples.

	Tag				
Strategy number		1	2	3	4
Matrix assembly					
Total	II	0.067	1.	1.574	2.341
Preconditioner setup					
Total	III	0.057	1.	1.134	7.290
GMRES					
Orthogonalization	IV.X	0.125	1.	1.141	1.100
Matrix-vector product	IV.IV	0.113	1.	1.138	1.334
Preconditioner	IV.III	0.143	1.	1.120	1.410
Total	IV	0.137	1.	1.134	1.322
Total		0.119	1.	1.225	1.577

Table B.2: Speed-up of the four strategies with the second strategy as the reference.

Appendix C

Multigrid preconditioners

This appendix describes in more detail the multigrid preconditioners used in this thesis.

The main motivation to use multigrid methods is the observation that relaxation-based methods such as Richardson iteration, Jacobi iteration, or Gauss-Seidel iteration typically converge slowly for error modes with low-frequency. The idea is then to construct a hierarchy of increasingly smaller linear systems called levels and to apply those relaxation-based methods and damp different error modes on each level.

A multigrid method has two phases: the setup phase, discussed in section C.1, where the hierarchy of levels is constructed and the solve phase, discussed in section C.2, where the multigrid is applied to a given vector.

C.1 Preconditioner setup

When the multigrid method is used as a preconditioner for GMRES, the setup is typically done once before starting GMRES and the multigrid grid is called once per iteration.

The setup phase consists in the construction of the hierarchy of levels. The multigrid method starts from the matrix of the linear system solved with GMRES $\boldsymbol{A}_0$ where the subscript 0 is used to specify that $\boldsymbol{A}_0$ is the level matrix of the level 0, the finest level.

There are two types of multigrid methods to generate the hierarchy of levels: the geometric (GMG) and algebraic multigrid methods (AMG). The GMG creates the hierarchy using different meshes for each level defined by the user. The AMG creates the hierarchy from the graph of the operator of the finest level, i.e. from the graph of the linear system of interest.

In this work, we only used aggregation-based AMG methods that rely on the computation of aggregates, a node of the next level using interaction of nodes on the current level.

When those aggregates have been constructed for all levels, the coarse level matrix $\boldsymbol{A}_{l+1}$ can be constructed using two operators: the prolongation $\boldsymbol{P}_{l+1}$ and restriction $\boldsymbol{R}_{l+1}$ operators:

$$\boldsymbol{A}_{l+1} = \boldsymbol{R}_{l+1}\, \boldsymbol{A}_l\, \boldsymbol{P}_{l+1}, \tag{C.1}$$

which leads to the multigrid hierarchy $\boldsymbol{A}_0$, $\boldsymbol{A}_1$, ..., $\boldsymbol{A}_{L-1}$ where L is the total number of levels as illustrated in Fig. C.1.

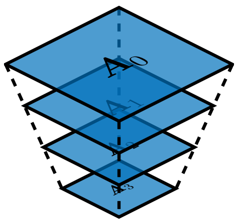


Figure C.1: Illustration of a multigrid hierarchy with 4 levels.

C.2 Preconditioner application

Now that the multigrid hierarchy is computed, we can apply the multigrid strategy. This application is based on a cycle which visits the different levels of the hierarchy starting and finishing from the finest level. On each level, the preconditioner applies a so-called *smoother*, a strategy used to damp some of the error modes such as a relaxation-based method, before moving to the next level of the cycle. There are different multigrid cycles such as the V-cycle, the W-cycle, or the F-cycle. In this work, we have limited ourselves to the V-cycle which can be seen as a recursive process written as follows:

```
Procedure V-cycle($\boldsymbol{x}_l, \boldsymbol{b}_l, l, L$)
if $l == L-1$ then
    $\boldsymbol{x}_l = \text{Coarse}(\boldsymbol{x}_l, \boldsymbol{b}_l)$                Coarsest smoother
    return $\boldsymbol{x}_l$
else
    $\boldsymbol{x}_l = \text{preS}(\boldsymbol{x}_l, \boldsymbol{b}_l)$                  Pre-smoother
    $\boldsymbol{r}_l = \boldsymbol{A}_l \boldsymbol{x}_l - \boldsymbol{b}_l$                    Residual computation
    $\boldsymbol{r}_{l+1} = \boldsymbol{R}_{l+1} \boldsymbol{r}_l$                         Restriction
    $\boldsymbol{e}_{l+1} = \text{V-cycle}\ (\boldsymbol{0}, \boldsymbol{r}_{l+1}, l+1, L)$      Coarse correction
    $\boldsymbol{x}_l = \boldsymbol{x}_l - \boldsymbol{P}_{l+1} \boldsymbol{e}_{l+1}$                Prolongation
    $\boldsymbol{x}_l = \text{postS}(\boldsymbol{x}_l, \boldsymbol{b}_l)$                 Post-smoother
    return $\boldsymbol{x}_l$
```

where $\text{Coarse}(\boldsymbol{x}_l, \boldsymbol{b}_l)$, $\text{preS}(\boldsymbol{x}_l, \boldsymbol{b}_l)$, and $\text{postS}(\boldsymbol{x}_l, \boldsymbol{b}_l)$ are the coarse smoother, the pre-smoother, and the post-smoother respectively. Those are the 3 smoothers used in the V-cycle:

- The pre-smoother is the smoother applied before restricting the residual on the coarser level,
- The post-smoother is the smoother applied after prolongating the correction computed on the coarser level,
- The coarse smoother is the smoother applied on the coarsest level. Typically, as the coarsest level matrix $\boldsymbol{A}_{L-1}$ has a smaller size, it can be useful to use a direct solver as the coarsest smoother: when the V-cycle reaches the coarsest level, a direct solver can be used to compute the solution of the system on the coarsest level.

As already said, the idea is to start with the fine level problem, apply a smoother (called pre-smoother because it happens before the restriction operator) such as some

Gauss-Seidel iterations, then compute the residual of the system and restrict it to the coarser level. Then the algorithm moves to the coarser level, if this is the last level, the coarse level smoother is used and return the solution to the one level finer. Otherwise, once again, a pre-smoother is applied and algorithm goes one level coarser. When moving from a coarser level to a finer level, the prolongation of the solution on the coarser level is used to update the solution on the current level, as this process can reintroduce higher frequency error mode, a post-smoother is used before sending the solution to the next finer level.

In this work, the right preconditioner $\boldsymbol{M}^{-1}$ used in Algo. 2 is used as follows:

$$\boldsymbol{M}^{-1}\,\boldsymbol{v}_{:j} := \text{V-cycle}(\boldsymbol{0}, \boldsymbol{v}_{:j}, 0, L). \tag{C.2}$$

C.3 Multigrid for blocked matrices

The content of this section comes from [Wiesner, 2015; Verdugo and Wall, 2016]. When the matrix $\boldsymbol{A}_0$ is a blocked matrix, for instance due to the discretization of a system of PDEs, there are two main approaches to use multigrid methods:

- Monolithic AMG preconditioner [Verdugo and Wall, 2016] also called Full multigrid approach in [Wiesner, 2015]: this approach consists in applying the multigrid on the blocked matrix to create a hierarchy of blocked matrices and, as level smoothers, use blocked smoothers such as block Gauss Seidel, block Jacobi, or SIMPLE for instance. This approach has the advantage that the coupling between the different fields is done on all the levels of the hierarchy which has improved convergence of strongly coupled problems in [Verdugo and Wall, 2016].

- Nested multigrid approach [Wiesner, 2015]: this second approach consists in decoupling the fields on the finest level using a block smoother such as a block Gauss Seidel and, then, using multigrid methods as the smoother of the block. This strategy is typically easier to implement as discussed in [Verdugo and Wall, 2016] but the main drawback is that the different fields are treated separately except on the finest level.

C.3.1 Block Gauss-Seidel

The block Gauss-Seidel is an iterative solver used to solve system with $N \times N$ block structure such as:

$$\underbrace{\begin{bmatrix} \boldsymbol{A}_{11} & \cdots & \boldsymbol{A}_{1N} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{N1} & \cdots & \boldsymbol{A}_{NN} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_N \end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_N \end{bmatrix}}_{\boldsymbol{b}}. \tag{C.3}$$

The idea is to use an approximation of the inverse of a triangular block matrix, either the lower block triangular matrix in the case of the forward BGS or the upper block triangular matrix in the case of the backward BGS:

$$\boldsymbol{M}_{\text{BGS}}^{\text{f}} := \begin{bmatrix} \boldsymbol{A}_{11} & \cdots & \boldsymbol{0} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}_{N1} & \cdots & \boldsymbol{A}_{NN} \end{bmatrix}, \qquad \boldsymbol{M}_{\text{BGS}}^{\text{b}} := \begin{bmatrix} \boldsymbol{A}_{11} & \cdots & \boldsymbol{A}_{1N} \\ \vdots & \ddots & \vdots \\ \boldsymbol{0} & \cdots & \boldsymbol{A}_{NN} \end{bmatrix}. \tag{C.4}$$

The strategy is then to apply the iterations:

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \boldsymbol{M}_{\text{BGS}}^{-1}\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k-1)}\right), \quad k = 1, \dots, k_{\max}, \tag{C.5}$$

where $\boldsymbol{x}^{(0)}$ is the initial guess.

In practice, the inverse of $\boldsymbol{M}_{\text{BGS}}$ is not explicitly computed, the approach is to use, in the case of the forward BGS:

$$\boldsymbol{x}_i^{(k)} = \boldsymbol{A}_{ii}^{-1}\left[\boldsymbol{b}_i - \sum_{j=1}^{i-1}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k)} + \sum_{j=i+1}^{N}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k-1)}\right] \quad \text{for } i = 1, \dots, N, \tag{C.6}$$

and, in the case of the backward BGS:

$$\boldsymbol{x}_i^{(k)} = \boldsymbol{A}_{ii}^{-1}\left[\boldsymbol{b}_i - \sum_{j=i+1}^{N}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k)} + \sum_{j=1}^{i-1}\boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k-1)}\right] \quad \text{for } i = 1, \dots, N. \tag{C.7}$$

Once again, the inverse of the matrices $\boldsymbol{A}_{ii}$ are not computed explictly and are approximated with a smoother such as Gauss-Seidel iterations.

In this thesis, we use a backward block Gauss-Seidel as the level smoother for thermomechanical simulation. It is more interesting to use a backward approach as the thermomechanical system block matrix is an upper block triangular matrix:

$$\begin{bmatrix}\boldsymbol{K} & \boldsymbol{S}\\ \boldsymbol{0} & \boldsymbol{L}\end{bmatrix}\begin{bmatrix}\boldsymbol{u}\\ \boldsymbol{T}\end{bmatrix} = \begin{bmatrix}\boldsymbol{f}\\ \boldsymbol{l}\end{bmatrix}. \tag{C.8}$$

C.3.2 SIMPLE

The SIMPLE smoother is a block smoother for systems with 2×2 block structure such as:

$$\underbrace{\begin{bmatrix}\boldsymbol{A}_{11} & \boldsymbol{A}_{12}\\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22}\end{bmatrix}}_{\boldsymbol{A}}\underbrace{\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix}\boldsymbol{b}_1\\ \boldsymbol{b}_2\end{bmatrix}}_{\boldsymbol{b}}. \tag{C.9}$$

The iterative process is as follows:

$$\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k-1)} + \boldsymbol{M}_{\text{SIMPLE}}^{-1}\left(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}^{(k-1)}\right), \quad k = 1, \dots, k_{\max}, \tag{C.10}$$

with $\boldsymbol{x}^{(0)}$ the initial guess and the matrix $\boldsymbol{M}_{\text{SIMPLE}}$ is the following block factorization of $\boldsymbol{A}$:

$$\boldsymbol{M}_{\text{SIMPLE}} := \begin{bmatrix}\boldsymbol{A}_{11} & \boldsymbol{0}\\ \boldsymbol{A}_{21} & \boldsymbol{R}\end{bmatrix}\begin{bmatrix}\boldsymbol{I} & \left(\widetilde{\boldsymbol{A}}_{11}\right)^{-1}\boldsymbol{A}_{12}\\ \boldsymbol{0} & \boldsymbol{I}\end{bmatrix}, \tag{C.11}$$

where $\widetilde{\boldsymbol{A}}_{11}$ is an approximation of $\boldsymbol{A}_{11}$ which is easy to invert (such as the diagonal of $\boldsymbol{A}_{11}$ for instance) and $\boldsymbol{R}$ is the corresponding approximation of the Schur complement associated with this $\widetilde{\boldsymbol{A}}_{11}$:

$$\boldsymbol{R} := \boldsymbol{A}_{22} - \boldsymbol{A}_{21}\left(\widetilde{\boldsymbol{A}}_{11}\right)^{-1}\boldsymbol{A}_{12}. \tag{C.12}$$

Once again, the computation of the inverse of $\boldsymbol{M}_{\text{SIMPLE}}$ is not done explicitly and, instead, four steps are computed:

$$\begin{aligned}
\Delta\boldsymbol{x}_1^{(k-1)} &= \boldsymbol{A}_{11}^{-1}\left[\boldsymbol{b}_1 - \boldsymbol{A}_{12}\boldsymbol{x}_2^{(k-1)}\right], && \text{(Predictor equation)}\\
\Delta\boldsymbol{x}_2^{(k-1)} &= \boldsymbol{R}^{-1}\left[\boldsymbol{b}_2 - \boldsymbol{A}_{21}\Delta\boldsymbol{x}_1^{(k-1)} - \boldsymbol{A}_{22}\boldsymbol{x}_2^{(k-1)}\right], && \text{(Schur equation)}\\
\boldsymbol{x}_2^{(k)} &= \boldsymbol{x}_2^{(k-1)} + \Delta\boldsymbol{x}_2^{(k-1)}, && \text{(Update)}\\
\boldsymbol{x}_1^{(k)} &= \boldsymbol{x}_1^{(k-1)} - \left(\widetilde{\boldsymbol{A}}_{11}\right)^{-1}\boldsymbol{A}_{12}\Delta\boldsymbol{x}_2^{(k-1)}. && \text{(Update)}
\end{aligned} \tag{C.13}$$

The SIMPLE smoother for contact problems

$$\boldsymbol{A} := \begin{bmatrix} \boldsymbol{K} & \boldsymbol{B}_1 \\ \boldsymbol{B}_2^{\mathrm{T}} & \boldsymbol{C} \end{bmatrix}, \tag{C.14}$$

has block preconditioning matrix $\boldsymbol{M}$:

$$\boldsymbol{M} := \begin{bmatrix} \boldsymbol{K} & \boldsymbol{0} \\ \boldsymbol{B}_2^{\mathrm{T}} & \boldsymbol{R} \end{bmatrix}\begin{bmatrix} \boldsymbol{I} & \boldsymbol{D}^{-1}\boldsymbol{B}_1 \\ \boldsymbol{0} & \frac{1}{\beta}\boldsymbol{I} \end{bmatrix} = \begin{bmatrix} \boldsymbol{K} & \boldsymbol{K}\boldsymbol{D}^{-1}\boldsymbol{B}_1 \\ \boldsymbol{B}_2^{\mathrm{T}} & \frac{1}{\alpha\beta}\boldsymbol{C} + (1 - \frac{1}{\alpha\beta})\boldsymbol{B}_2^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{B}_1 \end{bmatrix}, \tag{C.15}$$

where $\boldsymbol{D}$ is an approximation of $\boldsymbol{K}$, $\boldsymbol{R} := \frac{1}{\alpha}\boldsymbol{C} - \frac{1}{\alpha}\boldsymbol{B}_2^{\mathrm{T}}\boldsymbol{D}^{-1}\boldsymbol{B}_1$ is an approximation of the Schur complement, and α and β are damping factors chosen in the interval $]0,1]$ discussed in [Elman et al., 2008]. In this example, we use the matrix diagonal of $\boldsymbol{K}$ as the approximation $\boldsymbol{D}$.

Appendix D

Impact of ensemble propagation on the false sharing

In this section, we discuss the impact of ensemble propagation on the false sharing. To do so, we first start by reviewing what the false sharing is and illustrate it by an example and how to avoid it. After that, we revisit the example with ensemble propagation and show how it has impacted the results.

This appendix is related to section 7.2.2 where we observed the speed-up of the threaded Gauss-Seidel iterations. We observed a better speed-up than expected, this can be partially explained by the impact of ensemble propagation on the false sharing discussed in this appendix.

D.1 False sharing

False sharing is a performance issue on shared memory multiprocessor, where each processor has a local cache. The performance issue happens when at least two threads on different cores (and therefore which have different local cache) modify different variables stored on a same cache line. This is called false sharing as the two threads do not modify a shared data but they modify a same cache line. This results in a performance issue as the coordination required to ensure the cache line consistency can be avoided having the data on different cache lines.

D.2 Example of false sharing

As an example of false sharing, we compute the sum of all the entries of a matrix in such a way that each thread computes the sum over a tile of the matrix and store it in a local temporary variable. A first implementation is illustrated in Listing D.1 which presents false sharing due to the fact that the entries `result[p]` are located on the same cache lines. A way to avoid the issue is to use a padding strategy: to allocate more memory for the array `result` such that the local entries are not on the same cache line. For architecture with cache line of 64 bytes and with data stores on 8 bytes, we can allocate 8 times more data as proposed in Listing D.2.

The wall-clock time and speed-up of the threaded implementation are illustrated in Fig. D.1 and Fig. D.2 respectively. We observe, as expected, that the first implementation is not significantly faster using more threads than one as false sharing occurs. This impacts the speed-up of the threaded implementation which remains close to 1. The second implementation has a better speed-up as the false sharing is removed.

Those results have been generated with a code compiled without any optimization as the compiler is able to find some false sharing and pad the data automatically.

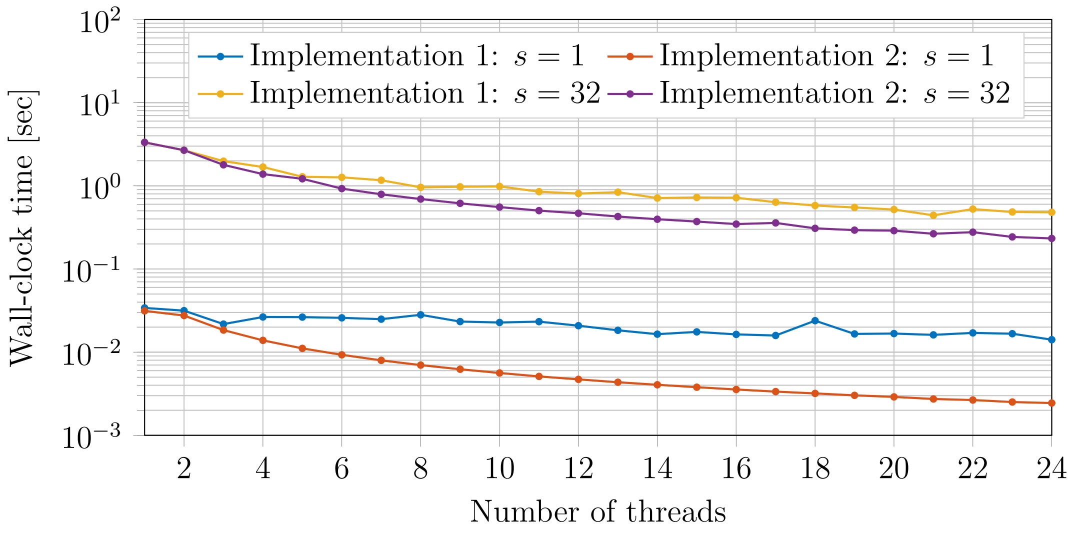


Figure D.1: Wall-clock time of the example without ensemble propagation ($s = 1$) and with ensemble propagation with one ensemble of size 32 ($s = 32$).

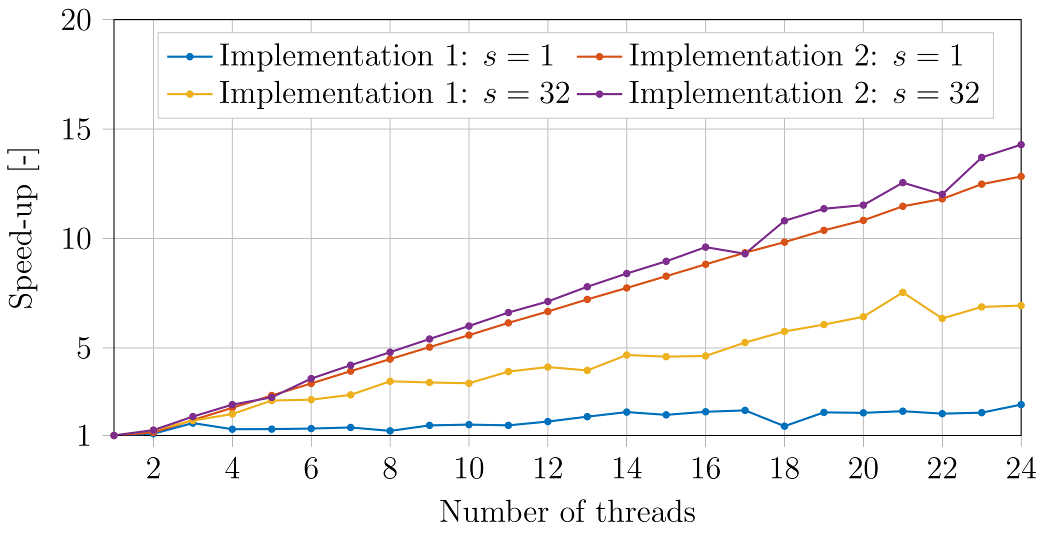


Figure D.2: Speed-up of the example without ensemble propagation ($s = 1$) and with ensemble propagation with one ensemble of size 32 ($s = 32$).

D.3 Impact of ensemble propagation

Due to the fact the ensemble size is chosen to be equal to 8, 16, or 32, there is no possibility for two ensembles to be stored on a same cache line. Therefore, this prevents the possibility of false sharing even in the first implementation.

```cpp
template <typename scalar>
void example(scalar *matrix, const int DIM, const int NUM_THREADS, scalar &sum)
{
    scalar result[NUM_THREADS];

#pragma omp parallel num_threads(NUM_THREADS)
    {
        int p = omp_get_thread_num();

        result[p] = 0;
        int chunkSize = DIM / NUM_THREADS + 1;
        int myStart = p * chunkSize;
        int myEnd = min(myStart + chunkSize, DIM);
        for (int i = myStart; i < myEnd; ++i)
            for (int j = 0; j < DIM; ++j)
                result[p] += matrix[i * DIM + j];
    }

    sum = 0;
    for (int p = 0; p < NUM_THREADS; ++p)
        sum += result[p];
}
```

Listing D.1: Example of false sharing.

```cpp
template <typename scalar>
void example(scalar *matrix, const int DIM, const int NUM_THREADS, scalar &sum)
{
    scalar result[NUM_THREADS][8];

#pragma omp parallel num_threads(NUM_THREADS)
    {
        int p = omp_get_thread_num();

        result[p][0] = 0;
        int chunkSize = DIM / NUM_THREADS + 1;
        int myStart = p * chunkSize;
        int myEnd = min(myStart + chunkSize, DIM);
        for (int i = myStart; i < myEnd; ++i)
            for (int j = 0; j < DIM; ++j)
                result[p][0] += matrix[i * DIM + j];
    }

    sum = 0;
    for (int p = 0; p < NUM_THREADS; ++p)
        sum += result[p][0];
}
```

Listing D.2: Example with padding to avoid false sharing.

References

Adams, B. M., Eldred, M. S., Geraci, G., Hooper, R. W., Jakeman, J. D., Maupin, K. A., Monschke, J. A., Rushdi, A. A., Stephens, J. A., Swiler, L. P., Wildey, T. M., Bohnhoff, W. J., Dalbey, K. R., Ebeida, M. S., Eddy, J. P., Hough, P. D., Khalil, M., Hu, K. T., Ridgway, E. M., Vigil, D. M., and Winokur, J. G. (2019). DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis: version 6.11 user's manual. Technical report, Sandia National Laboratories, Albuquerque, NM.

Arnst, M. and Ponthot, J.-P. (2014). An overview of nonintrusive characterization, propagation, and sensitivity analysis of uncertainties in computational mechanics. *International Journal for Uncertainty Quantification*, 4(5):387–421.

Baert, L., Beauthier, C., Leborgne, M., and Lepot, I. (2015). Surrogate-based optimisation for a mixed-variable design space: Proof of concept and opportunities for turbomachinery applications. In *Turbo Expo: Power for Land, Sea, and Air*, volume 56659. American Society of Mechanical Engineers.

Baker, C. G. and Heroux, M. A. (2012). Tpetra, and the use of generic programming in scientific computing. *Scientific Programming*, 20(2):115–128.

Barabash, V. (2013). *Materials Design Limit Data.* ITER_D_222RLN, ITER.

Bavier, E., Hoemmen, M., Rajamanickam, S., and Thornquist, H. (2012). Amesos2 and Belos: Direct and iterative solvers for large sparse linear systems. *Scientific Programming*, 20(3):241–255.

Beaucaire, P., Beauthier, C., and Sainvitu, C. (2019a). Exploitation of multiple surrogate models in multi-point infill sampling strategies. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 245–246.

Beaucaire, P., Beauthier, C., and Sainvitu, C. (2019b). Multi-point infill sampling strategies exploiting multiple surrogate models. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1559–1567.

Beauthier, C., Beaucaire, P., and Sainvitu, C. (2017). A surrogate-based evolutionary algorithm for highly constrained design problems. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1613–1613.

Beauthier, C., Mahajan, A., Sainvitu, C., Hendrick, P., Sharifzadeh, S., and Verstraete, D. (2014). Hypersonic cryogenic tank design using mixed-variable surrogate-based optimization. In *Engineering Optimization IV—Proceedings of the 4th International Conference on Engineering Optimization, ENGOPT*, pages 543–549.

Beazley, D. M. (2003). Automated scientific software scripting with SWIG. *Future Generation Computer Systems*, 19(5):599–609.

Belgacem, F. B. (1999). The Mortar finite element method with Lagrange multipliers. *Numerische Mathematik*, 84(2):173–197.

Benzi, M., Golub, G. H., and Liesen, J. (2005). Numerical solution of saddle point problems. *Acta numerica*, 14:1–137.

Benzi, M. and Wathen, A. J. (2008). Some preconditioning techniques for saddle point problems. In *Model order reduction: theory, research aspects and applications*, pages 195–211. Springer.

Bernardi, C., Maday, Y., and Patera, A. T. (1993). Domain decomposition by the Mortar element method. In *Asymptotic and numerical methods for partial differential equations with critical parameters*, pages 269–286. Springer.

Blackford, L. S., Petitet, A., Pozo, R., Remington, K., Whaley, R. C., Demmel, J., Dongarra, J., Duff, I., Hammarling, S., Henry, G., et al. (2002). An updated set of basic linear algebra subprograms (BLAS). *ACM Transactions on Mathematical Software*, 28(2):135–151.

Boman, E., Devine, K., Heaphy, R., Hendrickson, B., Heroux, M., and Preis, R. (2004). LDRD report: Parallel repartitioning for optimal solver performance. Technical report, SAND2004–0365, Sandia National Laboratories, Albuquerque, NM.

Booth, J. D., Ellingwood, N. D., Thornquist, H. K., and Rajamanickam, S. (2017). Basker: Parallel sparse LU factorization utilizing hierarchical parallelism and data layouts. *Parallel Computing*, 68:17–31.

Bosten, A. (2019). Towards a contact formulation for efficient numerical simulation of marine ice-sheet instabilities. Master Thesis, University of Liège.

Cassagne, A., Le Gal, B., Leroux, C., Aumage, O., and Barthou, D. (2015). An efficient, portable and generic library for successive cancellation decoding of polar codes. In *Languages and Compilers for Parallel Computing*, pages 303–317. Springer.

Coutinho, B., Sampaio, D., Pereira, F. M. Q., and Meira Jr, W. (2011). Divergence analysis and optimizations. In *2011 International Conference on Parallel Architectures and Compilation Techniques*, pages 320–329. IEEE.

Dahlgren, T., Domyancic, D., Brandon, S., Gamblin, T., Gyllenhaal, J., Nimmakayala, R., and Klein, R. (2015). Scaling uncertainty quantification studies to millions of jobs. In *Proceedings of the 27th ACM/IEEE International Conference for High Performance Computing and Communications Conference (SC)*.

Daniel, J. W., Gragg, W. B., Kaufman, L., and Stewart, G. W. (1976). Reorthogonalization and stable algorithms for updating the Gram-Schmidt QR factorization. *Mathematics of Computation*, 30(136):772–795.

D'Elia, M., Phipps, E. T., Edwards, H., Hu, J. J., and Rajamanickam, S. (2018). Ensemble grouping strategies for embedded stochastic collocation methods applied to anisotropic diffusion problems. *SIAM/ASA Journal on Uncertainty Quantification*, 6(1):87–117.

D'Elia, M., Phipps, E. T., Rushdi, A., and Ebeida, M. (2020). Surrogate-based ensemble grouping strategies for embedded sampling-based uncertainty quantification. In *Quantification of Uncertainty: Improving Efficiency and Technology*, pages 41–66. Springer.

Demmel, J., Grigori, L., Hoemmen, M., and Langou, J. (2008). Communication-avoiding parallel and sequential QR factorizations. *CoRR abs/0806.2159*.

Deveci, M., Boman, E. G., Devine, K. D., and Rajamanickam, S. (2016a). Parallel graph coloring for manycore architectures. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 892–901. IEEE.

Deveci, M., Rajamanickam, S., Kim, K., Bradley, A. M., Trott, C. R., Hoemmen, M. F., and Boman, E. G. (2016b). KokkosKernels introduction: Design API and performance. Technical report, Sandia National Laboratories, Albuquerque, NM.

Draper, N. R. and Smith, H. (1998). *Applied regression analysis*, volume 326. John Wiley & Sons.

Edwards, H. C., Sunderland, D., Porter, V., Amsler, C., and Mish, S. (2012). Manycore performance-portability: Kokkos multidimensional array library. *Scientific Programming*, 20(2):89–114.

Edwards, H. C., Trott, C. R., and Sunderland, D. (2014). Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202–3216.

Eijkhout, V. (2013). *Introduction to High Performance Scientific Computing.* www.lulu.com.

Elman, H., Howle, V. E., Shadid, J., Shuttleworth, R., and Tuminaro, R. (2008). A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier–Stokes equations. *Journal of Computational Physics*, 227(3):1790–1808.

Estérie, P., Falcou, J., Gaunard, M., and Lapresté, J.-T. (2014). Boost.SIMD: generic programming for portable SIMDization. In *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing*, pages 1–8.

Ewart, T., Delalondre, F., and Schürmann, F. (2014). Cyme: a library maximizing SIMD computation on user-defined containers. In *International Supercomputing Conference*, pages 440–449. Springer.

Flemisch, B., Puso, M. A., and Wohlmuth, B. I. (2005). A new dual Mortar method for curved interfaces: 2D elasticity. *International Journal for Numerical Methods in Engineering*, 63(6):813–832.

Flemisch, B. and Wohlmuth, B. I. (2007). Stable Lagrange multipliers for quadrilateral meshes of curved interfaces in 3D. *Computer Methods in Applied Mechanics and Engineering*, 196(8):1589–1602.

Foster, I., Ainsworth, M., Allen, B., Bessac, J., Cappello, F., Choi, J. Y., Constantinescu, E., Davis, P. E., Di, S., Di, W., et al. (2017). Computing just what you need: Online data analysis and reduction at extreme scales. In *European Conference on Parallel Processing*, pages 3–19. Springer.

Geuzaine, C. and Remacle, J.-F. (2009). Gmsh: A 3-D finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331.

Ghanem, R. G., Higdon, D., and Owhadi, H. (2017). *Handbook of Uncertainty Quantification*. Springer.

Ghanem, R. G. and Spanos, P. D. (1990). Polynomial chaos in stochastic finite elements. *Journal of Applied Mechanics*, 57:197,202.

Ghanem, R. G. and Spanos, P. D. (1991). *Stochastic finite elements: a spectral approach*. Springer.

Glowinski, R. (1984). *Numerical methods for nonlinear variational problems*. Springer series in computational physics. Springer.

Golub, G. H. and Van Loan, C. F. (2012). *Matrix computations*, volume 3. JHU Press.

Goto, K. and van de Geijn, R. A. (2008). Anatomy of high-performance matrix multiplication. *ACM Transactions on Mathematical Software (TOMS)*, 34(3):12.

Griewank, A. (1989). On automatic differentiation. *Mathematical Programming: recent developments and applications*, 6(6):83–107.

Gyllenhaal, J., Gamblin, T., Bertsch, A., and Musselman, R. (2014). Enabling high job throughput for uncertainty quantification on BG/Q. *ser. IBM HPC Systems Scientific Computing User Group (SCICOMP)*.

Hadjidoukas, P. E., Angelikopoulos, P., Kulakova, L., Papadimitriou, C., and Koumoutsakos, P. (2015). Exploiting task-based parallelism in bayesian uncertainty quantification. In *European Conference on Parallel Processing*, pages 532–544. Springer.

Hadjidoukas, P. E., Lappas, E., and Dimakopoulos, V. V. (2012). A runtime library for platform-independent task parallelism. In *2012 20th Euromicro international conference on parallel, distributed and network-based processing*, pages 229–236. IEEE.

Hammond, S., Vaughan, C., and Hughes, C. (2018). Evaluating the Intel Skylake Xeon processor for HPC workloads. In *2018 International Conference on High Performance Computing & Simulation (HPCS)*, pages 342–349. IEEE.

Hansen, G., Mish, S. P., and Xavier, P. G. (2016). An MPI+X implementation of contact global search using Kokkos. *Engineering with Computers*, 32(2):295–311.

Hennessy, J. L. and Patterson, D. A. (2011). *Computer architecture: a quantitative approach*. Elsevier.

Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A. B., and Stanley, K. S. (2005). An overview of the Trilinos project. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):397–423.

Intel (2019). Intel® C++ Compiler 19.1 Developer Guide and Reference. `https://software.intel.com/en-us/cpp-compiler-developer-guide-and-reference`.

Jeffers, J., Reinders, J., and Sodani, A. (2016). *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*. Morgan Kaufmann.

Karpiński, P. and McDonald, J. (2017). A high-performance portable abstract interface for explicit SIMD vectorization. In *Proceedings of the 8th International Workshop on Programming Models and Applications for Multicores and Manycores*, pages 21–28.

Karypis, G. (2011). METIS and ParMETIS. *Encyclopedia of parallel computing*, pages 1117–1124. Springer.

Karypis, G. and Kumar, V. (1998a). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392.

Karypis, G. and Kumar, V. (1998b). Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129.

Kikuchi, N. and Oden, J. T. (1988). *Contact problems in elasticity: a study of variational inequalities and finite element methods*. SIAM.

Kim, K., Costa, T. B., Deveci, M., Bradley, A. M., Hammond, S. D., Guney, M. E., Knepper, S., Story, S., and Rajamanickam, S. (2017). Designing vector-friendly compact BLAS and LAPACK kernels. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 55. ACM.

Kirsch, C. (1898). Die Theorie der Elastizität und die Bedürfnisse der Festigkeitslehre. *Zeitschrift des Vereines Deutscher Ingenieure*, 42:797–807.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM review*, 51(3):455–500.

Koornwinder, T. (1975). Two-variable analogues of the classical orthogonal polynomials. In *Theory and application of special functions*, pages 435–495. Elsevier.

Krasikov, Y., Panin, A., Litnovsky, A., Mertens, P., and Schrader, M. (2017). Specific design and structural issues of single crystalline first mirrors for diagnostics. *Fusion Engineering and Design*, 124:548–552.

Krasikov, Y., Panin, A., Wolgang, B., Krimmer, A., Litnovsky, A., Mertens, P., Neubauer, O., and Schrader, M. (2015). Major aspects of the design of a first mirror for the ITER core CXRS diagnostics. *Fusion engineering and design*, 96:812–816.

Kretz, M. (2015). *Extending C++ for explicit data-parallel programming via SIMD vector types*. PhD thesis, Johann Wolfgang Goethe-Universität.

Kretz, M. (2018). Data-parallel vector types & operations. *ISO/IEC C++ Standards Committee Paper P0214R8 (2018).*

Kretz, M. and Lindenstruth, V. (2012). Vc: A C++ library for explicit vectorization. *Software: Practice and Experience*, 42(11):1409–1430.

Krimmer, A., Balboa, I., Conway, N. J., De Bock, M., Friese, S., Le Guern, F., Hawkes, N. C., Kampf, D., Krasikov, Y., Mertens, P., et al. (2019). Design status of the ITER core CXRS diagnostic setup. *Fusion Engineering and Design*, 146:228–231.

Künzner, F., Neckel, T., and Bungartz, H.-J. (2019). Prediction and reduction of runtime in non-intrusive forward UQ simulations. *SN Applied Sciences*, 1(9):1038.

Le Maître, O. and Knio, O. M. (2010). *Spectral methods for uncertainty quantification: with applications to computational fluid dynamics*. Springer Science & Business Media.

Li, C. and Vuik, C. (2004). Eigenvalue analysis of the simple preconditioning for incompressible flow. *Numerical Linear Algebra with Applications*, 11(5-6):511–523.

Liegeois, K. (2015). Investigation of a first mirror concept and its cooling capability during operation and baking in the ITER environment. Master Thesis, University of Liège.

Liegeois, K., Boman, R., Phipps, E. T., Wiesner, T. A., and Arnst, M. (2020). GMRES with embedded ensemble propagation for the efficient solution of parametric linear systems in uncertainty quantification of computational models. *Computer Methods in Applied Mechanics and Engineering*.

Lomont, C. (2011). Introduction to Intel Advanced Vector Extensions. *Intel White Paper*, pages 1–21.

Lorie, R. A. and Strong Jr, H. R. (1984). Method for conditional branch execution in SIMD vector processors. US Patent 4,435,758.

Martin, K. and Hoffman, B. (2010). *Mastering CMake: a cross-platform build system*. Kitware.

Martinelli, M., Dervieux, A., Hascoët, L., Pascual, V., and Belme, A. (2010). AD-based perturbation methods for uncertainties and errors. volume 2, pages 65–74. Inderscience Publishers.

McCalpin, J. D. (1995). Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25.

Mertens, P. (2018). The core-plasma CXRS diagnostic for ITER: An introduction to the current design. *Journal of Fusion Energy*.

Mertens, P., Boman, R., Dickheuer, S., Krasikov, Y., Krimmer, A., Leichtle, D., Liegeois, K., Linsmeier, C., Litnovsky, A., Marchuk, O., Rasinski, M., and De Bock, M. (2019). On the use of rhodium mirrors for optical diagnostics in ITER. *Fusion Engineering and Design*, 146:2514–2518.

Meurant, G. and Tebbens, J. D. (2015). The role eigenvalues play in forming GMRES residual norms with non-normal matrices. *Numerical Algorithms*, 68(1):143–165.

Naujoks, D. (2006). *Plasma-material interaction in controlled fusion.* Number 39 in Springer series on atomic, optical, and plasma physics. Springer.

Ozik, J., Collier, N. T., Wozniak, J. M., and Spagnuolo, C. (2016). From desktop to large-scale model exploration with Swift/T. In *2016 Winter Simulation Conference (WSC)*, pages 206–220. IEEE.

Pantuso, D., Bathe, K.-J., and Bouzinov, P. A. (2000). A finite element procedure for the analysis of thermo-mechanical solids in contact. *Computers & Structures*, 75(6):551–573.

Pawlowski, R. P., Phipps, E. T., and Salinger, A. G. (2012a). Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, part i: Template-based generic programming. *Scientific Programming*, 20(2):197–219.

Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Owen, S. J., Siefert, C. M., and Staten, M. L. (2012b). Automating embedded analysis capabilities and managing software complexity in multiphysics simulation, part ii: Application to partial differential equations. *Scientific Programming*, 20(3):327–345.

Peterson, J. L., Anirudh, R., Athey, K., Bay, B., Bremer, P.-T., Castillo, V., Di Natale, F., Fox, D., Gaffney, J. A., Hysom, D., et al. (2019). Merlin: Enabling machine learning-ready HPC ensembles. *arXiv preprint arXiv:1912.02892.*

Phipps, E. T. (2015). Stokhos stochastic Galerkin uncertainty quantification methods.

Phipps, E. T., D'Elia, M., Edwards, H., Hoemmen, M., Hu, J. J., and Rajamanickam, S. (2017). Embedded ensemble propagation for improving performance, portability, and scalability of uncertainty quantification on emerging computational architectures. *SIAM Journal on Scientific Computing*, 39(2):C162–C193.

Phipps, E. T., Edwards, H. C., Hu, J., and Ostien, J. T. (2014). Exploring emerging manycore architectures for uncertainty quantification through embedded stochastic Galerkin methods. *International Journal of Computer Mathematics*, 91(4):707–729.

Phipps, E. T. and Pawlowski, R. (2012). Efficient expression templates for operator overloading-based automatic differentiation. In *Recent Advances in Algorithmic Differentiation*, pages 309–319. Springer.

Poirion, F. and Soize, C. (1995). Numerical methods and mathematical aspects for simulation of homogeneous and non homogeneous Gaussian vector fields. In *Probabilistic Methods in Applied Physics*, pages 17–53. Springer.

Popp, A., Gee, M. W., and Wall, W. A. (2013a). Mortar methods for single-and multi-field applications in computational mechanics. In *Sustained Simulation Performance 2012*, pages 133–154. Springer.

Popp, A., Seitz, A., Gee, M. W., and Wall, W. A. (2013b). Improved robustness and consistency of 3D contact algorithms based on a dual Mortar approach. *Computer Methods in Applied Mechanics and Engineering*, 264:67–80.

Prokopenko, A., Hu, J. J., Wiesner, T. A., Siefert, C. M., and Tuminaro, R. S. (2014). MueLu user's guide 1.0. Technical report, SAND2014–18874, Sandia National Laboratories, Albuquerque, NM.

Prokopenko, A., Siefert, C., Hu, J. J., Hoemmen, M. F., and Klinvex, A. M. (2016). Ifpack2 user's guide 1.0. Technical report, Sandia National Laboratories, Albuquerque, NM.

Puso, M. A. (2004). A 3D Mortar method for solid mechanics. *International Journal for Numerical Methods in Engineering*, 59(3):315–336.

Puso, M. A. and Laursen, T. A. (2004a). A Mortar segment-to-segment contact method for large deformation solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 193(6–8):601–629.

Puso, M. A. and Laursen, T. A. (2004b). A Mortar segment-to-segment frictional contact method for large deformations. *Computer Methods in Applied Mechanics and Engineering*, 193(45–47):4891–4913.

Robert, C. and Casella, G. (2013). *Monte Carlo Statistical Methods*. Springer.

Saad, Y. (2003). *Iterative methods for sparse linear systems*. SIAM.

Saad, Y. and Schultz, M. H. (1986). GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7(3):856–869.

Schloegel, K., Karypis, G., and Kumar, V. (1998). Multilevel algorithms for multi-constraint graph partitioning. In *Proceedings of Supercomputing '98*.

Schroeder, W. J., Lorensen, B., and Martin, K. (2004). *The visualization toolkit: an object-oriented approach to 3D graphics*. Kitware.

Smith, T. M., Van De Geijn, R., Smelyanskiy, M., Hammond, J. R., and Van Zee, F. G. (2014). Anatomy of high-performance many-threaded matrix multiplication. In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pages 1049–1059. IEEE.

Smolyak, S. A. (1963). Quadrature and interpolation formulas for tensor products of certain classes of functions. In *Doklady Akademii Nauk*, volume 148, pages 1042–1045. Russian Academy of Sciences.

Soize, C. (2017). *Uncertainty quantification*. Springer.

Sousedík, B., Ghanem, R. G., and Phipps, E. T. (2014). Hierarchical Schur complement preconditioner for the stochastic Galerkin finite element methods: Dedicated to Professor Ivo Marek on the occasion of his 80th birthday. *Numerical Linear Algebra with Applications*, 21(1):136–151.

Stroustrup, B. (2000). *The C++ programming language.* Pearson Education India.

Terpstra, D., Jagode, H., You, H., and Dongarra, J. (2010). Collecting performance data with PAPI-C. In *Tools for High Performance Computing 2009*, pages 157–173. Springer.

Ullmann, E. (2010). A Kronecker product preconditioner for stochastic Galerkin finite element discretizations. *SIAM Journal on Scientific Computing*, 32(2):923–946.

Van Zee, F. G. and Van De Geijn, R. A. (2015). BLIS: A framework for rapidly instantiating BLAS functionality. *ACM Transactions on Mathematical Software (TOMS)*, 41(3):14.

Veldhuizen, T. (1995). Expression templates. *C++ Report*, 7(5):26–31.

Verdugo, F. and Wall, W. A. (2016). Unified computational framework for the efficient solution of n-field coupled problems with monolithic schemes. *Computer Methods in Applied Mechanics and Engineering*, 310:335–366.

Wiesner, T. A. (2015). *Flexible aggregation-based algebraic multigrid methods for contact and flow problems.* PhD thesis, Technische Universität München.

Wilde, M., Hategan, M., Wozniak, J. M., Clifford, B., Katz, D. S., and Foster, I. (2011). Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652.

Wohlmuth, B. I. (2000). A Mortar finite element method using dual spaces for the Lagrange multiplier. *SIAM Journal on Numerical Analysis*, 38(3):989–1012.

Wohlmuth, B. I. (2001). *Discretization Methods and Iterative Solvers Based on Domain Decomposition.* Lecture Notes in Computational Science and Engineering. Springer.

Wohlmuth, B. I. (2011). Variationally consistent discretization schemes and numerical algorithms for contact problems. *Acta Numerica*, 20:569–734.

Wriggers, P. (2006). *Computational Contact Mechanics.* Springer Science & Business Media.

Xiao, H. and Gimbutas, Z. (2010). A numerical algorithm for the construction of efficient quadrature rules in two and higher dimensions. *Computers & Mathematics with Applications*, 59(2):663–676.

Yang, B. and Laursen, T. A. (2008). A contact searching algorithm including bounding volume trees applied to finite sliding Mortar formulations. *Computational Mechanics*, 41(2):189–205.