Large scale nonconforming domain decomposition methods

Abdoulaye Samaké

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques appliquées**

Arrêté ministériel : du 7 août 2006

Présentée par

Abdoulaye Samaké

Thèse dirigée par **Christophe Prud'homme**

préparée au sein **du Laboratoire Jean Kuntzmann**
et de **l'école doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Méthodes non-conformes de décomposition de domaine à grande échelle

Thèse soutenue publiquement le **08 décembre 2014**,
devant le jury composé de :

M. Eric Blayo
Professeur, Université Joseph Fourier, Président
M. Damien Tromeur-Dervout
Professeur, Université Lyon 1, Rapporteur
M. Zakaria Belhachmi
Professeur, Université de Haute Alsace, Rapporteur
M. Frédéric Nataf
Directeur de recherche, CNRS, Examinateur
M. Christophe Prud'homme
Professeur, Université de Strasbourg, Directeur de thèse
M. Christophe Picard
Maître de Conférences, Grenoble INP, Co-Directeur de thèse

Dedicated to my parents,
my mother Djénéba Samaké
and my late father Moriba Samaké.

Abdoulaye Samaké

Acknowledgments

I would like to express my deepest gratitude to my PhD advisors, Prof. Christophe Prud'homme and Assoc. Prof. Christophe Picard. Thanks to your relevant advice and your support, this thesis has taken place in an especially promising environment for the high-level research in scientific computation and it was a rewarding experience for me.

I am very grateful to Prof. Damien Tromeur-Dervout and Prof. Zakaria Belhachmi for having kindly accepted to be the reviewers of this thesis. My thanks go to Prof. Eric Blayo and Frédéric Nataf, Director of Reasearch CNRS, for agreeing to be jury members.

I want to extend my sincere thanks to Silvia Bertoluzza and Micol Pennacchio for the fruitful collaboration. I am particularly touched by your kindness and warm hospitality during my stays in your research laboratory in Italy.

I am very thankful to Vincent Chabannes, PhD, for many interesting discussions, especially around Feel++, which were useful for the attainment of this project. My thanks also go to Stephane and Morgane.

I want to express my thanks to all the Feel++ team members, specifically Cécile, Marcelas, Ranine, Mourad, Guillaume, Alexandre, Pierre, Christophe T., Vincent D. and Vincent H. It was a real pleasure for me to have worked with you in this great team. I would also like to extend my thanks to the EDP team of Jean Kuntzmann Laboratory (LJK), of which I was a member during this thesis. My thanks go to Charlotte for having been very courteous.

I am sincerely grateful to Hélène, Juana, Catherine and Laurence, the administrative staff of LJK. I thank you very much for the valuable administrative services rendered throughout my PhD thesis. I was deeply moved by your kindness.

I would like to express my heartfelt gratitude to my family, specifically my dearest beloved younger sisters, Alamako and Kamissa. I am heartily thankful to you for your endless love and unwavering support.

Finally, I would like to pay a stirring tribute to my paternal grandmother Samaké Alamako Doumbia and to the memory of my maternal grandmother Samaké Hawa Traoré.

Table of Contents

List of Figures

List of Tables

Notations

Domains and spaces

Symbol		Description
Ω	:	the whole domain
$\partial\Omega$	:	the boundary of Ω
Ω_ℓ	:	the ℓ-th domain
$\gamma_\ell^{(i)}$	:	the i-th side of Ω_ℓ
$\Gamma_{\ell n}$	:	the intersection of $\partial\Omega_\ell$ and $\partial\Omega_n$
H_ℓ	:	the diameter of Ω_ℓ
h_ℓ	:	the mesh characteristic length in Ω_ℓ
p_ℓ	:	the polynomial order in Ω_ℓ
M_h^m	:	the finite dimensional multiplier space on γ_m
$\mathcal{K}^\ell$	:	the family of compatible quasi-uniform shape regular decompositions of Ω_ℓ
$\mathcal{X}_h$	:	the constrained approximation space
$\mathcal{T}_h$	:	the trace space
R_h	:	the discrete lifting operator
$\mathcal{S}$	:	the skeleton of the domain decomposition
$\mathcal{D}(\Omega_\ell)$	:	the space of infinitely differentiable functions with compact support in Ω_ℓ
$L^2(\Omega_\ell)$	:	the square-integrable function space
$H^1(\Omega_\ell)$	:	the Sobolev function space
$H_0^1(\Omega_\ell)$	:	the closure of $\mathcal{D}(\Omega_\ell)$ in $H^1(\Omega_\ell)$
$H_g^1(\Omega_\ell)$	:	the space $H^1(\Omega_\ell)$ with g value on $\partial\Omega_\ell$
$H^{1/2}(\partial\Omega_\ell)$	:	the trace space of $H^1(\Omega_\ell)$ on $\partial\Omega_\ell$
$H^{-1/2}(\partial\Omega_\ell)$	:	the dual space of $H^{1/2}(\partial\Omega_\ell)$
$P_{p_\ell}(K)$	:	the space of polynomials of degree at most p_ℓ
C^0	:	the set of the continuous functions

Chapter 1

Introduction

This chapter introduces the general context of domain decomposition methods. The main contributions and the outline of this thesis are presented respectively in section 1.3 and in section 1.4. The publications are available in section 1.5. We briefly present in section 1.6 the finite element library Feel++ used for implementation of numerical methods and preconditioners studied in this work. The notion of scalability is introduced in section 1.7.

Ce chapitre introduit le contexte général des méthodes de décomposition de domaine. Les principales contributions et le plan de cette thèse sont présentés respectivement dans la section 1.3 et dans la section 1.4. Les publications sont disponibles dans la section 1.5. Nous présentons brièvement la librairie élément fini Feel++ utilisée pour la mise en oeuvre des méthodes numériques et préconditionneurs étudiés dans ce travail. La notion de scalabilité est introduite dans la section 1.7.

Contents

1.1 English version

In scientific computing, tremendous progress in the design and availability of parallel computers over the last decades have allowed large scale simulations for complex scientific and engineering applications. The investigation of efficient and robust numerical methods adapted to modern computer architectures is a major challenge.

For solving very large sparse linear system, direct methods [Dav06; DER86] or iterative methods [Saa03] can be used. Direct methods are robust and accurate for general nonsingular problems, but they scale poorly with the problem size in terms of time and memory complexity, especially for problems resulting from the discretization of Partial Differential Equations (PDEs) in three-dimensional space. In contrast, iterative methods require less storage and fewer operations than direct methods, but their performance depends strongly on the spectral properties of the linear system. The preconditioning techniques can improve the efficiency and the robustness of these methods.

The term preconditioning refers to transformation of the original linear system into another linear system with the same solution, but with more favorable properties for iterative solver. A preconditioner is a matrix that affects such a transformation. In general, the preconditioning aims to improve the spectral properties of the matrix associated with the linear system.

Domain decomposition methods are one of most common paradigms to compute the solution of large scale problems arising from the discretization of Partial Differential Equations (PDEs) in many applications, for example multiscale simulations on massively parallel computer architectures. They combine the strength of both direct and iterative methods, known as hybrid methods, for constructing scalable and robust preconditioners. The central idea of these methods consists in reducing the large problem into a collection of smaller problems, each of which is easier to solve computationally than the original problem, and most or all of which can be solved independently and concurrently. Domain decomposition methods can be categorized mainly into two classes, namely overlapping methods and nonoverlapping or iterative substructuring methods.

In this thesis we investigate a numerical and computational framework for domain decomposition methods (overlapping and nonoverlapping). In the class of overlapping domain decomposition methods, we are interested in Schwarz methods [Gan08] introduced by H. A. Schwarz [Sch70] in the original form, known as alternating Schwarz method, to establish the existence of a solution to the Poisson problem with prescribed boundary conditions in a complex domain. In the class of nonoverlapping domain decomposition methods, we focus mainly on the mortar finite element method, a nonconforming approach of domain decomposition methods introduced in [BMP93a]. The main feature of this method is that the continuity condition at the interfaces between subdomains is ensured in the weak form, i.e. the jump

of the finite element solution on the interfaces should be L^2-orthogonal to a chosen finite element space on the interfaces. Thus, it allows to combine different discretizations and/or methods in different subdomains, which considerably increases the flexibility of this method. We deal with two mortar finite element formulations: the first one is the original formulation [BMP93a] in which the weak matching condition is directly taken into account in the approximation space. This formulation, known as the mortar method with constrained space, leads to a symmetric, positive and definite system allowing the use of efficient preconditioners. In the second formulation, introduced in [Bel99], the weak matching condition is achieved by introducing a Lagrange multiplier. The Lagrange multiplier space should be chosen such that the Ladyzhenskaya-Babuska-Brezzi inf-sup condition is satisfied. The algebraic linear system arising from this formulation is of saddle-point type, symmetric and indefinite. For such formulation problems, iterative methods are known to be less efficient than for symmetric positive definite systems, see [BGL05]. The efficiency of the iterative method to solve the algebraic system with a large scaled matrix depends heavily on the preconditioner used.

This thesis considers the substructuring approach, proposed in [BPS86] in the framework of conforming domain decomposition and extended to nonconforming domain decomposition and a general class of finite elements of any order in [BP07; BP04; Ber+14]. This approach consists in considering a suitable splitting of the nonconforming discretization space in terms of "interior", "edge" and "vertex" degrees of freedom and then using the related block-Jacobi type preconditioners. In the same class of domain decomposition methods, we briefly discuss three-field method introducing in [BM94] and its numerical implementation.

1.2 Version française

En calcul scientifique, d'énormes progrès dans la conception et la disponibilité d'ordinateurs parallèles au cours des dernières décennies ont permis la réalisation de grandes simulations pour des applications scientifiques et d'ingénierie complexes. L'investigation de méthodes numériques efficaces, robustes et adaptées aux architectures modernes d'ordinateurs est un défi majeur.

Pour résoudre un très grand système linéaire creux, les méthodes directes [Dav06 ; DER86] ou les méthodes itératives [Saa03] peuvent être utilisées. Les méthodes directes sont robustes et précises en général pour des problèmes non singuliers, mais elles passent mal à l'échelle avec la taille du problème en termes de complexité en temps et en espace mémoire, en particulier pour des problèmes provenant de la discrétisation d'Équations aux Dérivées Partielles (EDPs) en trois dimensions d'espace. En revanche, les méthodes itératives nécessitent moins de stockage en mémoire et moins d'opérations que les méthodes directes, mais leur performance dépend fortement des propriétés spectrales du système linéaire. Les techniques de préconditionnement peuvent améliorer l'efficacité et la robustesse de ces méthodes.

Le terme préconditionnement se réfère à la transformation du système linéaire original

en un autre système linéaire ayant la même solution, mais avec des propriétés plus favorables pour le solveur itératif. Un préconditionneur est une matrice qui applique une telle transformation. En général, le préconditionnement vise à améliorer les propriétés spectrales de la matrice associées au système linéaire.

Les méthodes de décomposition de domaine sont un des paradigmes les plus courants pour calculer la solution de très grands problèmes provenant d'applications différentes, par exemple les simulations multi-échelles sur des architectures massivement parallèles. L'idée centrale de ces méthodes consiste à réduire le grand problème en une collection de petits problèmes dont chacun est plus facile à résoudre que le problème original. Les méthodes de décomposition de domaine peuvent être catégorisées principalement en deux classes, à savoir les méthodes de décomposition de domaine avec recouvrement et celles sans recouvrement, aussi appelées les méthodes de sous-structuration.

Dans cette thèse nous étudions un framework numérique et de calcul pour les méthodes de décomposition de domaine (avec et sans recouvrement). Dans la classe des méthodes de décomposition de domaine avec recouvrement, nous nous intéressons aux méthodes de Schwarz [Gan08] introduite par H. A. Schwarz [Sch70] dans la forme originale connue sous le nom de l'algorithme de Schwarz alterné, afin d'étudier l'existence d'une solution au problème de Poisson homogène avec des conditions aux limites imposées dans un domaine de calcul complexe. Dans la classe des méthodes de décomposition de domaine sans recouvrement, nous nous focalisons principalement sur la méthode mortar (aussi appelée la méthode des éléments finis joints), une méthode de décomposition de domaine non conforme introduite dans [BMP93a]. La principale attractivité de cette méthode est que la condition de continuité aux interfaces entre sous-domaines est traitée sous forme faible, c'est à dire le saut de la solution élément fini aux interfaces doit être L^2-orthogonal à un espace élément fini défini sur les interfaces. Cela permet de combiner des discrétisations différentes et/ou des méthodes d'approximation différentes dans des sous-domaines différents, ce qui accroît considérablement la flexibilité de cette méthode. Nous traitons deux formulations mortar : la première est la formulation mortar originale [BMP93a] dans laquelle la condition de continuité faible est directement prise en compte dans l'espace d'approximation. Cette formulation connue sous le nom de la formulation mortar avec espace contraint conduit à un système linéaire symétrique, défini et positif permettant l'utilisation des préconditionneurs efficaces. Dans la seconde formulation mortar introduite dans [Bel99], la condition de continuité faible est réalisée en introduisant un multiplicateur de Lagrange. L'espace de multiplicateur de Lagrange doit être bien choisi de telle sorte que la propriété inf-sup de Ladyzhenskaya-Babuska-Brezzi soit satisfaite. Le système algébrique linéaire provenant de cette discrétisation est de type point-selle, symétrique et indéfini. Pour de tels problèmes, les méthodes itératives sont connues pour être moins efficaces que pour les systèmes symétriques, définis et positifs, voir [BGL05]. L'efficacité de la méthode iterative pour résoudre un système algébrique linéaire de grande taille depend fortement du préconditionneur utilisé.

Cette thèse considère l'approche par sous-structuration proposée dans [BPS86] dans le framework sur les méthodes de décomposition de domaine conformes et étendue à celles non conformes et à une classe générale d'éléments finis d'ordre arbitraire dans [BP07 ; BP04 ; Ber+14]. Cette approche consiste à considérer une décomposition appropriée de l'espace de discrétisation non conforme en termes de degrés de liberté "interieurs", ceux sur les "arêtes" et ceux sur les "sommets" et à utiliser des préconditionneurs de type bloc-Jacobi associés. Dans cette même classe de méthodes de décomposition de domaine sans recouvrement, nous discutons brièvement de la méthode three-field introduite par F. Brezzi in [BM94] et de son implémentation numérique.

1.3 Contributions

This thesis aims at the development and the analysis of a generic computational framework for domain decomposition methods and preconditioners in Feel++ programming environment. The domain decomposition methods surveyed in this work include the mortar finite element method, Schwarz methods and three-field method. This thesis contributes to the field of high-performance computing by implementing these methods and preconditioners on massively parallel computer architectures.

1.4 Outline

The body of this dissertation is organized into three parts plus the appendices collecting the essential results used in this thesis. The part I (Chapters 2 to 5) investigates a wide range of domain decomposition methods with a special emphasis placed on mortar element method. The part II (Chapters 6 to 7) develops a generic and flexible implementation framework for various numerical methods described in part I. The part III (Chapters 8 to 9) summarizes the numerical experiments supporting the theoretical results and the scalability property of the parallel numerical algorithms discussed in this work.

Part I

Chapter 2 reviews domain decomposition methods including overlapping methods and sub-structuring methods. In section 2.1, we recall the Schwarz additive and multiplicative algorithms with different artificial boundary conditions namely Dirichlet-Dirichlet, Dirichlet-Neumann, Neumann-Neumann and Robin-Robin. We introduce in section 2.2 the general concept of mortar element method, specially the hybrid formulation using Lagrange multipliers and the formulation with constrained space. In section 2.3, we handle the basic formulation of the three-field method. Finally, we remind The FETI method in section 2.4.

Chapter 3 is devoted to the mortar element method with constrained space for two-dimensional problems. We first introduce the model problem and the basic notations in section 3.1. We define the functional settings in section 3.2. The discretization aspects including the technical

tools required for the construction and the analysis of the substructuring preconditioners for this method are presented in section 3.3. We remind the discrete formulation and the mortar correction operator respectively in section 3.5 and in section 3.6. Finally, we analyze the convergence properties supporting the theoretical estimates in section 3.7.

Chapter 4 deals with the substructuring preconditioners for mortar element method in two-dimensional space. We consider the substructuring approach in section 4.1. We investigate in section 4.2 the vertex block of the preconditioner and emphasize its fundamental role for the good scaling properties of the preconditioners. The algebraic forms for the realization of the preconditioners and discrete Steklov-Poincaré operator are available in section 4.3.

Chapter 5 studies the mortar element method with Lagrange multipliers. We introduce the hybrid formulation for mortar element method in section 5.1. A special computational framework for this method is discussed in section 5.2. Finally, we analyze the convergence properties in accordance with the theoretical results.

Part II

Chapter 6 covers the implementation aspects of the substructuring preconditioners described in Chapter 4. In section 6.1, we define some basic ingredients required for Feel++ implementation. The section 6.2 emphasizes the crucial role of the linear interpolation operator for domain decomposition framework in Feel++. We discuss the geometric and algebraic framework for the realization of the preconditioners in section 6.3. The code design illustrating the robustness and the flexibility of our parallel codes is summarized in section 6.6.

Chapter 7 develops an implementation framework for Schwarz methods, three-field method and mortar element method with Lagrange multipliers. In section 7.1, we discuss two communication approaches for Schwarz methods in Feel++, namely explicit and seamless communications. The section 7.2 briefly presents the assembly of jump matrices in the classical three-field formulation. The section 7.3 handles a special parallel implementation of mortar element method with Lagrange multipliers in 2D and 3D based on the duplication of data at the interfaces between subdomains, in order to reduce the interprocess communications. The Feel++ codes showing the flexibility of the library and in particular its ability to handle domain decomposition methods are reported.

Part III

Chapter 8 summarizes the numerical results for substructuring preconditioners for h-p mortar element method, introduced and analyzed in Chapter 4. We define the problem settings and the computational platforms for all numerical simulations achieved. In section 8.1, we consider the conforming domain decompositions using linear elements and high-order elements. We solve the Schur complement system, the algebraic representation of the Steklov-Poincaré operator defined in Chapter 3, preconditioned by the substructuring preconditioners proposed

in Chapter 4. The numerical results for the examination and validation of the mathematical properties of substructuring preconditioners are reported. In section 8.2, we run the same set of experiments carried out in section 8.1, but by considering nonconforming domain decompositions. The section 8.3 is dedicated to the large scale simulations with Discontinuous Galerkin coarse preconditionner. The section 8.4 analyzes the performance and scalability of the parallel implementation on large scale computer architectures.

Chapter 9 analyzes a framework for basic numerical results for Schwarz methods, three-field method, and mortar element method with Lagrange multipliers. Some numerical experiments for Schwarz methods using explicit and seamless communication approach are presented in section 9.1. A few tests for three-field method in two and three dimensional space are given in section 9.2. The section 9.3 is centered on the scalability and performance analysis of a parallel implementation of mortar element method with Lagrange multipliers in three-dimensional space.

1.5 Publications

Journal papers

[Ber+14] Silvia Bertoluzza, Micol Pennacchio, Christophe Prud'homme, and Abdoulaye Samaké. "substructuring preconditioners for hp mortar FEM". In : *ESAIM : Mathematical Modelling and Numerical Analysis* (2014). Submitted (cit. on pp. 3, 5, 28).

[Sam+12b] Abdoulaye Samaké, Vincent Chabannes, Christophe Picard, and Christophe Prud'Homme. "Domain decomposition methods in Feel++". In : *Springer* (Nov. 2012). published.

[Sam+14] Abdoulaye Samaké, Silvia Bertoluzza, Micol Pennacchio, and Christophe Prud'homme. "Implementation and Numerical Results of Substructuring preconditioners for the h-p Mortar Method". In : *in preparation* (2014) (cit. on p. 124).

Conference papers

[A+12] Samaké A., Chabannes V., Daversin C., Ismail M. Doyeux V., Prud'homme C., Trophime C., and Veys S. "Advances in Feel++ : A domain specific embedded language in C++ for partial differential equations". In : *European Community on Computational Methods in Applied Sciences*. Published. 2012.

[Pru+12a] Christophe Prud'homme, Vincent Chabannes, Vincent Doyeux, Mourad Ismail, Abdoulaye Samaké, and Gonçalo Pena. "Feel++ : A Computational Framework for Galerkin Methods and Advanced Numerical Methods". In : *Cemracs 2011*. Published. EDP Sciences, Jan. 2012 (cit. on p. 67).

[Sam+12a] Abdoulaye Samaké, Silvia Bertoluzza, Micol Pennacchio, Christophe Prud'Homme, and Chady Zaza. "A Parallel Implementation of the Mortar Element Method in 2D and 3D". In : published. 2012 (cit. on pp. 56, 62).

[Sam+12c] Abdoulaye Samaké, Vincent Chabannes, Christophe Picard, and Christophe Prud'Homme. "Implementation of Schwarz methods using Feel++". In : *21st International Conference on Domain Decomposition Methods*. Published. 2012.

Conference talks

[A+12] Samaké A., Chabannes V., Daversin C., Ismail M. Doyeux V., Prud'homme C., Trophime C., and Veys S. "Advances in Feel++ : A domain specific embedded language in C++ for partial differential equations". In : *European Community on Computational Methods in Applied Sciences*. Published. 2012.

[Sam+12c] Abdoulaye Samaké, Vincent Chabannes, Christophe Picard, and Christophe Prud'Homme. "Implementation of Schwarz methods using Feel++". In : *21st International Conference on Domain Decomposition Methods*. Published. 2012.

1.6 Feel++ Library

The numerical implementation of various domain decomposition methods and preconditioners presented in this thesis has been done using Feel++ library. We present in this section a short extract from [Pru+12b] dedicated to Feel++.
Feel++, *Finite Element Embedded Language in C++* [Pru+12b ; Pru07 ; Pru06] is a C++ library for partial differential equation resolution using generalized Galerkin methods such as the finite element method, the h-p finite element method and the spectral element method. It aims at bringing the scientific community a tool for the implementation of advanced numerical methods and high-performance computing.

Two main aspects in the design of the library are to (i) have the syntax, semantics of the library very close to mathematics, and (ii) have a small manageable library that makes use wherever possible of established libraries (for linear system solves, for instance). While the first point at creating a high-level language powerful enough to describe solution strategies in a simple way, the second aspect with the maintenance of the code delegating some procedures to frequently maintained third party libraries.

Feel++ relies on a so-called *domain specific embedded language* (DSEL) designed to closely match the Galerkin mathematical framework. In computer science, DS(E)Ls are used to partition complexity and in our case the DSEL splits low level mathematics and computer science on one side leaving the Feel++ developer to enhance them and high-level mathematics as well as physical applications to the other side which are left to the Feel++ user. This enables using Feel++ for teaching purposes, solving complex problems with multiple physics and scales or rapid prototyping of new methods, schemes or algorithms. The goal is always to hide (ideally all) technical details behind software layers, providing only the relevant components required by the user or programmer and enforce the mathematical language computationnally between the users be they physicists, mathematicians, computer scientists, engineers or

students. The DSEL approach has advantages over generating a specific *external* language : (i) interpreter/compiler construction complexities can be ignored, (ii) libraries can concurrently be used which is often not the case of specific languages which would have to also develop their own libraries and library system, (iii) DSELs inherit the capabilities of the host language (e.g. *C++*).

The DSEL on Feel++ provides access to powerful tools, yet simple and seamless interface, such as interpolation and the clear translation of a wide range of variational formulations into the variational embedded language. Combined with this robust engine, lie also state of the art arbitrary order finite elements including handling high-order geometrical approximations, high-order quadrature formulas and robust nodal configuration sets. The tools at the user's disposal grant the flexibility to implement numerical methods that cover a large combination of choices from meshes, function spaces or quadrature points using the same integrated language and control at each stage of the solution process the numerical approximations.

Feel++ uses advanced *C++* (e.g. template meta-programming) and in particular the latest standard *C++* 11 that provides very useful additions such as type inference `auto` and `decltype` keywords. Feel++ also uses the essential Boost *C++* libraries [Kor11]. The data structures of Feel++ can be customized with respect to MPI communicators. The linear algebra is handled by PETSc library [Bal+04].

1.7 Scalability Analysis

In the context of High-Performance Computing (HPC), the scalability expresses the gain of solving large problems on parallel computers. The performance of a parallel computer depends on a wide number of factors [HRP93] affecting the scalability of a parallel algorithm.

From [HRP93], some basic metrics affecting the scalability of a parallel computer for a parallel algorithm are given by :

- Machine size — the number of processing units employed in a parallel computer. The computational power is a function of machine size.
- Clock rate — the clock rate refers to the frequency of a CPU.
- Problem size — the amount of computational workload used for solving a given problem. The problem size is directly proportional to the *sequential execution time*.
- CPU time — the CPU time (in seconds) elapsed in the execution of a given program on a parallel computer with n processing units. It is the *parallel execution time*.
- I/O demand — the input/output demands when running the program.
- Memory capacity — the amount of main memory (in bytes) used in the execution of a program. The memory demand is affected by the problem size, the algorithms and the data structures used.

- Communication overhead — the amount of time elapsed in interprocess communications, synchronization and remote memory access.
- Computer cost — the total cost of hardware and software resources required to carry out the execution of a program.

The Scalability analysis expresses the ability of a given parallel algorithm to best exploit a parallel computer architecture. The notion of scalability is related to the notions of speedup and efficiency. We introduce below the notion of strong scalability (speedup) and the weak scalability (efficiency).

1.7.1 Speedup

The measure of speedup was used to determine the quality of parallel algorithm running on a parallel computational platform. The speedup is defined as the time to run a problem of size n on one processor, divided by the time it takes to run the same problem using p processors. Generally speaking, let T_1 be the sequential execution time on one processor and T_p be the parallel execution time on p processors. The speedup is written

$$S_p = T_1/T_p.$$

Note that in speedup analysis, the problem size remains fixed but the number of processing units are increased.

1.7.2 Efficiency

The efficiency is defined by

$$E_p = S_p/p.$$

The best possible efficiency is $E_p = 1$. It is reached when the speedup is linear, i.e. $S_p = p$. Note that in efficiency analysis, the problem size assigned to each core remains constant and additional processing units are used for solving a larger problem.

Part I

Numerical Methods

Chapter 2

Review of Domain Decomposition Methods

A Generic numerical framework for domain decomposition methods is studied. Schwarz methods, mortar finite element method, three-field method and feti method are recalled respectively in section 2.1, section 2.2, section 2.3 and section 2.4. We provide a convergence analysis for Schwarz, three-field and mortar element methods with respect to mesh size and polynomial order using Feel++.

Un framework générique pour les méthodes de décomposition de domaine est étudié. Les méthodes de Schwarz, la méthode des éléments finis mortar, la méthode three-field et la méthode feti sont rappelées respectivement dans la section 2.1, section 2.2, section 2.3 et dans la section 2.4. Nous présentons une analyse de convergence pour les méthodes de Schwarz, la méthode three-field et la méthode des éléments finis mortar en fonction de la taille caractéristique du maillage et de l'ordre polynomial en utilisant Feel++.

Contents

2.1 Schwarz Methods

The original form of Schwarz iterative procedure, known as the alternating Schwarz method was introduced by H. A. Schwarz [Sch70] to establish the existence of a solution to the elliptic boundary value problem (2.1) on the union of two subdomains $\Omega = \Omega_1 \cup \Omega_2$, as in FIGURE 2.1.

$$-\Delta u = f \ \text{ in } \ \Omega, \qquad u = g \ \text{ on } \ \partial\Omega. \tag{2.1}$$

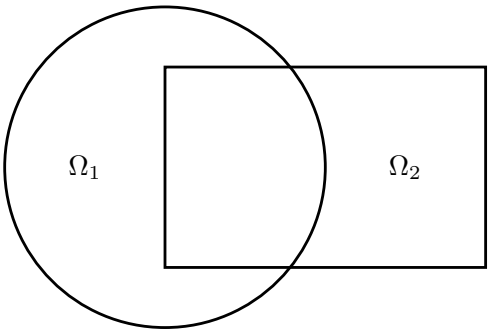


FIGURE 2.1 : Overlapping partition for alternating Schwarz method

This method begins by selecting an initial guess u_2^0, and then define iteratively for $n \geq 0$, two sequences u_1^{n+1} and u_2^{n+1} solutions of

$$\begin{array}{lll} -\Delta u_1^{n+1} = f & \text{in } \Omega_1 & \\ u_1^{n+1} = g & \text{on } \partial\Omega_1 \cap \partial\Omega & \text{and} \\ u_1^{n+1} = u_2^n & \text{on } \partial\Omega_1 \cap \overline{\Omega}_2 & \end{array} \qquad \begin{array}{lll} -\Delta u_2^{n+1} = f & \text{in } \Omega_2 & \\ u_2^{n+1} = g & \text{on } \partial\Omega_2 \cap \partial\Omega & \\ u_2^{n+1} = u_1^{n+1} & \text{on } \partial\Omega_2 \cap \overline{\Omega}_1 & \end{array} \tag{2.2}$$

We present a generic computational framework for the extension of the algorithm (2.2) for more general cases including overlapping and nonoverlapping Schwarz methods (conforming and nonconforming grids). The framework main objectives consist in *(i)* reproducing and comparing easily several of methods of the literature *(ii)* developing a teaching and research programming environment *(iii)* providing the methods at the functional level or at the algebraic level.

2.1.1 Schwarz Methods at the Continuous Level

Let Ω be a domain of $\mathbb{R}^d$, $d = 1\,, 2\,, 3$, and $\partial\Omega$ its boundary. We look for u the solution of the problem :

$$Lu = f \quad \text{in} \quad \Omega, \quad u = g \quad \text{on} \quad \partial\Omega \tag{2.3}$$

where L is a linear partial differential operator, and f and g are given functions. Let $\Omega_i (i = 1, \ldots, N,\ N \in \mathbb{N},\ N \geq 2)$ the subdomain partitions of Ω such that $\overline{\Omega} = \cup_{i=1}^{N} \overline{\Omega}_i$ and $\Gamma_{ij} = \partial\Omega_i \cap \overline{\Omega}_j$ the interface between neighboring subdomains Ω_i and Ω_j. Let $\mathcal{V}_{\Omega_i}$ be the set of neighboring subdomains of Ω_i. In the case of nonoverlapping subdomains, $\Gamma_{ij} = \Gamma_{ji}$. We are interested in the overlapping and nonoverlapping Schwarz methods [QV99 ; BBG04] as solver with the general nonmatching grids and arbitrary number of subdomains. The generic Schwarz additive algorithm is given by (2.4) where u_i^0 is known on Γ_{ij}, $j \in \mathcal{V}_{\Omega_i}$, $k \geq 1$ the Schwarz iteration index and C_i is a partial differential operator.

$$Lu_i^k = f \text{ in } \quad \Omega_i, \quad u_i^k = g \text{ on } \quad \partial\Omega_i \setminus \Gamma_{ij}, \quad C_i u_i^k = C_i u_j^{k-1} \text{on } \quad \Gamma_{ij} \tag{2.4}$$

The algorithm (2.4) extends easily to the multiplicative version of Schwarz methods and to different types of artificial boundary conditions such as Dirichlet-Dirichlet (DD), Dirichlet-Neumann (DN), Neumann-Neumann (NN) and Robin-Robin (RR) [QV99 ; BBG04] according to the choice of the operator C_i that is assumed linear in our case. The above algorithm can also adapt to relaxation techniques [QV99] necessary for the convergence of some types of interface conditions such as DN and NN without overlap.

2.1.2 One-level Schwarz Methods

In general, Schwarz methods are used as a preconditioner for a Krylov subspace method. We consider the following algebraic linear system arising from the discretization of (2.3)

$$Au = f \tag{2.5}$$

A fixed point method for (2.5) is given by : for a given u^0, we look for

$$u^{n+1} = u^n + M^{-1}(f - Au^n). \tag{2.6}$$

We group the unknowns into subsets, $u_j = R_j u$, $j = 1, \ldots, J$, where R_j are rectangular matrices such that each entry u_i of the vector u is contained in at least one u_j, see [SGT07]. For each subdomain, the local matrix is defined by $A_j = R_j A R_j^T$.

The classical multiplicative Schwarz method (MSM) is a preconditioner for (2.6) where M is defined as

$$M_{MSM}^{-1} = \left[I - \prod_{j=1}^{J} \left(I - R_j^T A_j^{-1} R_j A \right) \right] \tag{2.7}$$

The classical additive Schwarz method (ASM) is a preconditioner for (2.6) where M is defined as

$$M_{ASM}^{-1} = \sum_{j=1}^{J} R_j^T A_j^{-1} R_j. \tag{2.8}$$

From [SGT07], the additive Schwarz iteration (2.6), (2.8) does not correspond to the classical iteration per subdomain and does not converge in general.

The restricted additive Schwarz method (RAS) introduced in [CS99], allowing to correct this problem, is a preconditioner for (2.6) defined by

$$M_{RAS}^{-1} = \sum_{j=1}^{J} \widetilde{R}_j^T A_j^{-1} R_j, \tag{2.9}$$

where $\widetilde{R}_j^T$ is the prolongation operator corresponding to a nonoverlapping decomposition.

2.1.3 Two-level Schwarz Methods

In the literature [TW04; QV99], it is well-known that the one-level methods do not scale with the number of subdomain. Achieving a good scalability property requires a coarse space correction [Nat+11].
We consider a coarse mesh $\mathcal{T}^H$ on the domain Ω and a finite element space of continuous, piecewise linear functions on $\mathcal{T}^H$. Let R_0 be the matrix representation of linear interpolation from the coarse grid to fine grid. The coarse matrix A_C is derived from the global matrix A by the relation $A_C = R_0^T A R_0$. A two-level restricted additive Schwarz (RAS2) is a two step preconditioner for (2.6). The iterations are given by

$$u^{n+1/2} = u^n + \sum_{j=1}^{J} \widetilde{R}_j^T A_j^{-1} R_j (f - A u^n), \tag{2.10}$$

$$u^{n+1} = u^{n+1/2} + R_0^T A_0^{-1} R_0 (f - A u^{n+1/2}). \tag{2.11}$$

The development of the scalable Schwarz methods is an active field, for example [Jol+12] proposes a strategy that scales over up to several thousands subdomains for scalar diffusion problems and linear elasticity. For in-depth investigations on overlapping Schwarz methods, we refer the reader to [BBG04; TW04; Lio88; EZ98; CN98; CGL01].

2.1.4 Convergence Analysis

We summarize in FIGURE 2.2 the convergence analysis for a one-level additive Schwarz method (see section 7.1 for more details) by choosing the analytic solution $g = \sin(\pi x)\cos(\pi y)$. We plot the relative L^2 error $\|u - u_h\|_{L^2}$ as a function of the characteristic mesh size h for different

polynomial orders from P_1 to P_5. The artificial boundary condition is Dirichlet-Dirichlet and the number of subdomains is equal to 128.

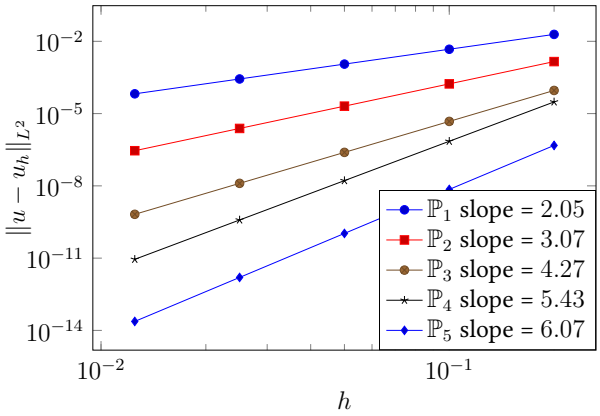


Figure 2.2 : Convergence analysis for additive Schwarz Method in 2D

The Figure 2.2 shows that our framework verifies the best convergence properties in accordance with the finite element theoretical results. The implementation and the numerical experiments of the Schwarz methods described in this section are available respectively in section 7.1 and section 9.1.

2.2 Mortar Element Method

Introduced in the early nineties by Bernardi, Maday and Patera [BMP94] as a tool to couple spectral and finite element method for the solution of second-order elliptic Partial Differential Equations (PDEs), the mortar method has been quickly extended to treat many different application fields [AP95 ; BBM01 ; Pru98 ; BB94 ; Pen04 ; PS08]. It appears to be well suited for parallel implementation and to the coupling of many different approximation spaces. The method has gained a wide popularity, since it offers the possibility to use different, non matching, possibly heterogeneous discretizations in different regions of the domain of definition of the problem at hand.

The mortar approximations involve the weak continuity constraints on the space. Two different approaches can be used to ensure these weak constraints. The original mortar formulation [BMP93b] can be seen as a nonconforming finite element approximation since the weak continuity constraints are directly taken into account in the approximation space. This approach leads to a symmetric positive definite problem. Another mortar formulation introduced in [Bel99] achieves the weak continuity constraints as Lagrange multipliers, leading to a saddle point problem, which is symmetric and indefinite. In either case, efficient iterative methods are essential for the overall performance of the method.

However, in order to make such technique more competitive for real life applications, one has to deal with the problem of the efficient solution of the associated linear system of equations. The design of efficient preconditioners for such linear system is then a fundamental task. Different approaches were considered in the literature : iterative substructuring [AMW99], additive Schwarz with overlap [KW06], FETI-DP [DDP06 ; Kim07 ; KL05] and BDDC [KW06].

A thorough investigation of the mortar element method is available in the remainder of this thesis. We will focus mainly on the formulation with constrained space, see chapter 3. We propose the substructuring preconditioners for the h-p version of such a formulation in chapter 4.

2.3 Three-field Method

The three-field method was inspired by the so-called *hybrid* finite element formulation for elasticity, see for example [Ton70], and has been introduced in the domain decomposition context by F. Brezzi and L.D. Marini [BM94].

The three-field formulation is a nonconforming domain decomposition method for the resolution of second-order elliptic boundary value problems. In this formulation, an independent variable is introduced to represent the trace of the global finite element solution on the skeleton of the domain decomposition. The Lagrange multipliers are used to enforce the weak continuity constraints on the skeleton. In the remainder of this section, we refer to [BFM08].

2.3.1 Three-field Formulation

Let $\Omega \subset \mathbb{R}^d$, $d = 2, 3$, be a convex polygonal domain. We will consider the following problem : given $f \in L^2(\Omega)$, find u satisfying

$$-\sum_{i,j=1}^{d} \frac{\partial}{\partial \mathbf{x}_j}\left(a_{ij}(\mathbf{x})\frac{\partial u}{\partial \mathbf{x}_i}\right) + a_0(\mathbf{x})u = f \ \text{ in } \ \Omega, \qquad u = 0 \ \text{ on } \ \partial\Omega. \tag{2.12}$$

We assume that for almost all $\mathbf{x} \in \Omega$ we have $0 \leq a_0(\mathbf{x}) \leq R$ and that the matrix $a(\mathbf{x}) = (a_{ij}(\mathbf{x}))_{i,j=1,\ldots,d}$, is symmetric positive definite, with smallest eigenvalue $\geq \alpha > 0$ and largest eigenvalue $\leq \alpha'$, $\alpha\,, \alpha'$ independent of $\mathbf{x}$.

In order to discretize the problem, we consider a decomposition of Ω as the union of L subdomains Ω_ℓ,

$$\overline{\Omega} = \bigcup_{\ell=1}^{L} \overline{\Omega}_\ell. \tag{2.13}$$

We set $\Gamma_\ell = \partial\Omega_\ell$, $\Sigma = \cup_\ell \Gamma_\ell$ being the skeleton of the decomposition. The functional setting for the three-field domain decomposition method is given by the following spaces :

$$V = \prod_{\ell=1}^{L} V^\ell, \qquad \text{with} \quad V^\ell = H^1(\Omega_\ell) \tag{2.14}$$

$$\Lambda = \prod_{\ell=1}^{L} \Lambda^\ell, \qquad \text{with} \quad \Lambda^\ell = H^{-1/2}(\partial\Omega_\ell) \tag{2.15}$$

$$\Phi = \left\{\varphi \in L^2(\Sigma) : \text{there exists } u \in H_0^1(\Omega),\ u|_\Sigma = \varphi\right\} \tag{2.16}$$

These functional spaces are respectively equipped with the norms [BM94]

$$\|u\|_V = \left(\sum_\ell \|u_\ell\|^2_{H^1(\Omega_\ell)} \right)^{1/2}, \qquad \|\lambda\|_\Lambda = \left(\sum_\ell \|\lambda_\ell\|^2_{H^{-1/2}(\partial\Omega_\ell)} \right)^{1/2}, \tag{2.17}$$

and

$$\|\varphi\|_\Phi = \inf_{\substack{u \in H^1_0(\Omega) \\ u=\varphi \text{ on } \Sigma}} \|u\|_{H^1(\Omega_\ell)} \simeq \left(\sum_\ell \|\varphi_\ell\|^2_{H^{1/2}(\partial\Omega_\ell)} \right)^{1/2} \tag{2.18}$$

Let $a^\ell : H^1(\Omega_\ell) \times H^1(\Omega_\ell) \longrightarrow \mathbb{R}$ denote the bilinear form corresponding to the linear operator considered

$$a^\ell(u,v) = \int_{\Omega_\ell} \left(\sum_{i,j=1}^{d} a_{ij}(\mathbf{x}) \frac{\partial u}{\partial \mathbf{x}_i} \frac{\partial v}{\partial \mathbf{x}_j} + a_0(\mathbf{x}) uv \right) d\mathbf{x}. \tag{2.19}$$

Under the assumptions made on the matrix $a(\mathbf{x}) = (a_{ij}(\mathbf{x}))_{i,j=1,\dots,d}$ and on a_0, the bilinear forms a^ℓ are uniformly $H^1(\Omega_\ell)$-continuous and semidefinite. More precisely, there exists positive constants R_1 and R_2 independent of ℓ and H_ℓ such that for all $u\,, v \in H^1(\Omega_\ell)$

$$R_1 |w|^2_{H^1(\Omega_\ell)} \leq a^\ell(u,v), \qquad |a^\ell(u,v)| \leq R_2 \|u\|_{H^1(\Omega_\ell)} \|v\|_{H^1(\Omega_\ell)}$$

The three-field formulation of (2.12) reads :

Problem 2.3.1. *find* $(u\,, \lambda\,, \varphi) \in V \times \Lambda \times \Phi$ *such that*

$$\begin{cases} a^\ell(u^\ell, v^\ell) & - \displaystyle\int_{\partial\Omega_\ell} u^\ell \lambda^\ell \, ds = \int_{\Omega_\ell} f v^\ell \, d\mathbf{x} & \forall\, v^\ell \in H^1(\Omega_\ell) \\ \displaystyle\int_{\partial\Omega_\ell} u^\ell \mu^\ell \, ds & - \displaystyle\int_{\partial\Omega_\ell} \mu^\ell \varphi \, ds = 0 & \forall\, \mu^\ell \in H^{-1/2}(\partial\Omega_\ell) \\ \displaystyle\sum_\ell \int_{\partial\Omega_\ell} \lambda^\ell \psi \, ds & = 0 & \forall\, \psi \in \Phi \end{cases} \tag{2.20}$$

From [BM94], the problem (2.20) admits a unique solution $(u\,, \lambda\,, \varphi) \in V \times \Lambda \times \Phi$ where u is the solution of (2.12) and such that

$$\lambda^\ell = \frac{\partial u^\ell}{\partial \nu^\ell} \text{ on } \Gamma_\ell, \qquad \varphi = u \text{ on } \Sigma, \tag{2.21}$$

where ν^ℓ denotes the outer conormal derivative to the subdomain Ω_ℓ.

The problem 2.3.1 can be discretized by a Galerkin method. Let $\mathcal{T}_u^\ell$ be a regular triangulation of Ω_ℓ with mesh size h_u^ℓ and let $\mathcal{T}_\lambda^\ell$ be the decomposition of Γ_ℓ with mesh size h_u^ℓ induced by $\mathcal{T}_u^\ell$ and $\mathcal{T}_\varphi$ the decomposition of Σ with mesh size h_φ^ℓ. Let $V_h^\ell \subseteq V^\ell$ and $\Lambda_h^\ell \subseteq \Lambda^\ell$ two finite element subspaces verifying

$$V_h^\ell \subseteq \Big\{ v \in C^0(\Omega_\ell) : v|_T \in \mathcal{P}_p(T),\ \forall\, T \in \mathcal{T}_u^\ell \Big\}, \tag{2.22}$$

$$\Lambda_h^\ell \subseteq \Big\{ \mu \in L^2(\Gamma_\ell) : \mu|_I \in \mathcal{P}_p(I),\ \forall\, I \in \mathcal{T}_\lambda^\ell \Big\}, \tag{2.23}$$

$$\Phi_h = \Big\{ \psi \in C^0(\Sigma) : \psi|_J \in \mathcal{P}_p(J),\ \forall\, J \in \mathcal{T}_\varphi \Big\}, \tag{2.24}$$

where $\mathcal{P}_p(T)$, $\mathcal{P}_p(I)$ and $\mathcal{P}_p(J)$ are the spaces of polynomials of order at most p on, respectively T, I and J. We denote the functions of the nodal bases of the discrete spaces V_h^ℓ, Λ_h^ℓ and Φ_h by $u_{\ell,i}$, $\lambda_{\ell,i}$ and φ_i respectively so that

$$V_h^\ell = \text{span}\Big\{ u_{\ell,i},\ i = 1, \dots, N_\ell^u \Big\} \tag{2.25}$$

$$\Lambda_h^\ell = \text{span}\Big\{ \lambda_{\ell,i},\ i = 1, \dots, N_\ell^\lambda \Big\} \tag{2.26}$$

$$\Phi_h = \text{span}\Big\{ \varphi_i,\ i = 1, \dots, N^\varphi \Big\} \tag{2.27}$$

Finally we set

$$V_h = \prod_{\ell=1}^{L} V_h^\ell, \qquad \Lambda_h = \prod_{\ell=1}^{L} \Lambda_h^\ell. \tag{2.28}$$

We consider the following problem

Problem 2.3.2. *find* $(u_h\,, \lambda_h\,, \varphi_h) \in V_h \times \Lambda_h \times \Phi_h$ *such that*

$$\begin{cases} a^\ell(u_h^\ell, v_h^\ell) & - \displaystyle\int_{\Gamma_\ell} u_h^\ell \lambda_h^\ell \, ds = \int_{\Omega_\ell} f v_h^\ell \, d\mathbf{x} & \forall v_h^\ell \in V_h^\ell \\ \displaystyle\int_{\Gamma_\ell} u_h^\ell \mu_h^\ell \, ds & - \displaystyle\int_{\Gamma_\ell} \mu_h^\ell \varphi_h \, ds = 0 & \forall \mu_h^\ell \in \Lambda_h^\ell \\ \displaystyle\sum_\ell \int_{\Gamma_\ell} \lambda_h^\ell \psi_h \, ds & = 0 & \forall\, \psi_h \in \Phi \end{cases} \tag{2.29}$$

The existence, uniqueness and stability of the solution of (2.29) rely on the validity of suitable *inf-sup* conditions [BM94]. The discrete bilinear operators associated to the subdomain Ω_ℓ are represented on the previous nodal basis equations (2.25) to (2.27) by

$$
\begin{aligned}
A_\ell = (a^\ell_{i,j}), \qquad & a^\ell_{i,j} := a^\ell(u_{\ell,j}, u_{\ell,i}) \\
B_\ell = (b^\ell_{i,j}), \qquad & b^\ell_{i,j} := \int_{\Gamma_\ell} u_{\ell,j}\lambda_{\ell,i}\, ds \\
C_\ell = (c^\ell_{i,j}), \qquad & c^\ell_{i,j} := \int_{\Gamma_\ell} \varphi_j\lambda_{\ell,i}\, ds
\end{aligned}
$$

and the discrete bilinear operators globally defined on Ω are given by

$$
A = \begin{pmatrix} A_1 & & 0 \\ & \ddots & \\ 0 & & A_L \end{pmatrix}, \qquad B = \begin{pmatrix} B_1 & & 0 \\ & \ddots & \\ 0 & & B_L \end{pmatrix}, \qquad \text{and} \quad C = \begin{bmatrix} C_1, C_2, \ldots, C_L \end{bmatrix}. \tag{2.30}
$$

In view of this notation, the linear system arising from (2.29) reads as follows

$$
\begin{pmatrix} A & B^T & 0 \\ B & 0 & C^T \\ 0 & C & 0 \end{pmatrix} \begin{pmatrix} \underline{u}_h \\ \underline{\lambda}_h \\ \underline{\varphi}_h \end{pmatrix} = \begin{pmatrix} \underline{f}_h \\ 0 \\ 0 \end{pmatrix} \tag{2.31}
$$

where $\underline{u}_h$, $\underline{\lambda}_h$ and $\underline{\varphi}_h$ are the vectors of the coefficients of u_h, λ_h and φ_h in the bases chosen for V_h, Λ_h and Φ_h respectively.

2.3.2 Convergence Analysis

We present in FIGURE 2.3 the convergence analysis of the three-field method described above by choosing the analytic solution $g = \sin(\pi x)\cos(\pi y)$ in two-dimensional space. We plot the relative L^2 error $\|u - u_h\|_{L^2}$ as a function of the characteristic mesh size h for different polynomial orders from P_1 to P_4.

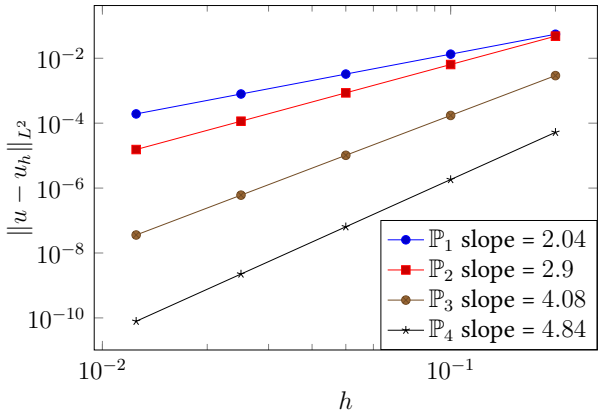


FIGURE 2.3 : Convergence analysis for three-field Method in 2D

The FIGURE 2.3 shows that our framework confirms the best convergence properties expected by the theoretical results. The implementation and the numerical experiments of the three-field formulation described in this section are available respectively in section 7.2 and in section 9.2.

2.4 Feti Method

The Finite Element Tearing and Interconnecting (FETI) method, introduced by Francois-Xavier Roux in [FR91], is a nonoverlapping domain decomposition method using Lagrange multipliers to ensure the continuity of the finite element solution across the subdomain interfaces. Originally, this method was used to solve second order, self-adjoint elliptic equations and it was later been extended to many other problems, e.g., time-dependent problems [FCM95]. In the remainder of this section, we refer to [Ste99, Chapter 5.].

2.4.1 Feti Algorithm

We consider the Poisson equation with dirichlet and Neumann boundary conditions

$$\begin{cases} -\Delta u = f & \text{in} \quad \Omega \\ u = 0 & \text{on} \quad \partial\Omega_D \\ \dfrac{\partial u}{\partial n} = 0 & \text{on} \quad \partial\Omega_N \end{cases} \tag{2.32}$$

The finite element mesh is partitioned along mesh lines into L nonoverlapping subdomains $\Omega_i \subset \Omega\,,\ i = 1\,,\ldots,L$. Since the finite element mesh is conforming, the boundary nodes of the subdomains match across the interface. A subdomain Ω_i is said to be floating if $\partial\Omega_i \cap \partial\Omega_D = \emptyset$, and nonfloating otherwise. Because of the Neumann boundary conditions, the local problems are indefinite in floating subdomains.

For each Ω_i, let K_i and $\hat{f}_i$ be the local stiffness matrix and right hand side, respectively. As in other substructuring methods, the first step of the FETI method consists in eliminating the interior subdomain variables. If K_i is written using blocks obtained by ordering the interior nodes first, and the boundary nodes last, then

$$K_i = \begin{pmatrix} K_{II,i} & K_{IB,i} \\ K_{BI,i} & K_{BB,i} \end{pmatrix} \tag{2.33}$$

where $K_{BI,i}$ is the transpose matrix of $K_{IB,i}$. Similarly,

$$\hat{f}_i = \begin{bmatrix} \hat{f}_{I,i} \\ \hat{f}_{B,i} \end{bmatrix}$$

The Schur complement matrix $S^{(i)}$ and the corresponding right hand side f_i are given by

$$S^{(i)} = K_{BB,i} - K_{BI,i}K_{II,i}^{-1}K_{IB,i}, \qquad f_i = \hat{f}_{B,i} - K_{BI,i}K_{II,i}^{-1}\hat{f}_{I,i}$$

Let $S = diag_{1=1}^{L} S^{(i)}$ be a block–diagonal matrix, and let f be the vector $[f_1\,,\ldots,f_L]$. We denote by u_i the vector of nodal values on $\partial\Omega_i$ and by u the vector $[u_1\,,\ldots,u_L]$.

If Ω_i is floating subdomain, then $S^{(i)}$ is a singular matrix and its kernel is generated by a vector Z_i which is equal to 1 at the nodes of $\partial\Omega_i$ and vanishes at all the other interface nodes. Let Z consisting of all the column vectors Z_i. Then

$$KerS = RangeZ. \tag{2.34}$$

Let B be the matrix of constraints which measures the jump of a given vector u across the interface ; B will also be referred to as the Lagrange multiplier matrix. Each row of the matrix B is associated to two matching nodes across the interface, and has values 1 and -1, respectively at the two nodes, and zero entries everywhere else. A finite element function with corresponding vector values u is continuous if and only if $Bu = 0$.

For a method without redundant constraints and multipliers, the number of pointwise continuity conditions required at crosspoints, i.e., the points that belong to the closure of more than two subdomains, and therefore the number of corresponding rows in the matrix B, is one less than the number of the subdomains meeting at the crosspoint. There exist several different ways of choosing which conditions to enforce at a crosspoint, all of them resulting in algorithms with similar properties. An alternative suggested in [RF97] is to connect all the degrees of freedom at the crosspoints by Lagrange multipliers and use a special scaling, resulting in a method with redundant multipliers.

Let W_i be the space of the degrees of freedom associated with $\partial\Omega_i \backslash \partial\Omega_D$, and let W be the direct sum of all spaces W_i. If $U = RangeB$ is the space of the Lagrange multipliers, then

$$S : W \longrightarrow W, \qquad B : W \longrightarrow U.$$

By introducing Lagrange multipliers λ for the constraint $Bu = 0$, we obtain a saddle point Schur formulation of (2.32),

$$\begin{aligned} Su + B^t\lambda &= f \\ Bu &= 0, \end{aligned} \tag{2.35}$$

where B^t denote the transpose of B.

2.4.2 Algebraic Formulation

In the FETI method, the primal variable u is eliminated from (2.32) and the resulting equation for the dual variable λ is solved by a projected conjugate gradient method. We note that S is singular if there exists at least one floating subdomains among the subdomains $\Omega_i\,,\ i = 1\,,\ldots,N$. Let $S^\dagger : W \longrightarrow W$ be the pseudoinverse of S, such that $S^\dagger b \in RangeS$, for any $b \perp KerS$. A solution for the first equation in (2.32) exists if and only if

$$f - B^t\lambda \perp KerS. \tag{2.36}$$

If (2.36) is satisfied, then

$$u = S^{\dagger}\big(f - B^t\lambda\big) + Z_\alpha, \tag{2.37}$$

where Z_α is an element of $KerS = RangeZ$; see (2.34) to be determined. Let $G = BZ$. Substituting (2.37) into the second equation in (2.35), it follows that

$$BS^{\dagger}B^t\lambda = BS^{\dagger}f + G_\alpha \tag{2.38}$$

An important role in the FETI algorithm is played by $V \subset U$ defined by $V = KerG^t$. In other words,

$$V = KerG^t \perp RangeG = BRangeZ = BKerS. \tag{2.39}$$

Let $P = I - G(G^tG)^{-1}G^t$ be the projection onto V. Since $P(G_\alpha) = 0$, if P is applied to (2.38), then

$$PBS^{\dagger}B^t\lambda = PBS^{\dagger}f. \tag{2.40}$$

From [Ste01], G^tG is nonsingular, by using the fact that

$$KerB \cap RangeZ = KerB \cap KerS = \emptyset.$$

We now return to the necessary condition (2.36). From (2.34), we obtain that (2.36) is equivalent to $f - B^t\lambda \perp RangeZ$, which leads to

$$Z^t\big(f - B^t\lambda\big) = 0,$$

and therefore to

$$G^t\lambda = Z^tf. \tag{2.41}$$

Let $F = BS^{\dagger}B^t$, $d = BS^{\dagger}f$, and $e = Z^tf$. We concluded that we have to solve the dual problem (2.40) for λ, subject to the constraint (2.41) ; with the new notations,

$$PF\lambda = Pd; \tag{2.42}$$
$$G^t\lambda = e \tag{2.43}$$

After that an approximate solution for λ is found, the primal variable u can be obtained as follows : Solving for α in (2.38),

$$\alpha = (G^tG)^{-1}G^t(F\lambda - d).$$

Then u can be obtained from (2.37) after solving a Neumann or a mixed boundary problem on each floating and nonfloating subdomain, respectively, corresponding to a vector multiplication by $S^{\dagger}$.

The main part of the FETI algorithm consists of solving (2.42) for the dual variable λ, which is done by a projected conjugate gradient(PCG) method. Since λ must also satisfy the constraint (2.43), let

$$\lambda_0 = G\big(G^tG\big)^{-1}e \tag{2.44}$$

be the initial approximation. Then $G^t\lambda_0 = e$ and $\lambda - \lambda_0 \in KerG^t = V$. If all the increments $\lambda_k - \lambda_{k-1}$, i.e. the search directions, are in V, then (2.43) will be satisfied.

One possible preconditioner for (2.42) is of the form PM, where

$$M = BSB^t. \tag{2.45}$$

When a vector multiplication by M is performed, N independent Dirichlet problems have to be solved at each iteration step. Therefore, M is known as the Dirichlet preconditioner. We note that the Schur complement matrix S is never computed explicitly, since only the action of S on a vector is needed.

From [MT96], the condition number of this FETI method has a condition number which grows polylogarithmically with the number of nodes in each subdomain,

$$\kappa(PMPF) \leq c\Big(1 + \log(\frac{H}{h})\Big)^3,$$

where C is a positive constant independent of h, H. If there are no crosspoints in the partition of Ω, then the bound improves to $\big(1 + \log(H/h)\big)^2$.

We reminded the classical FETI method in this section. In the remainder of this thesis, this method will not be subject of an advanced survey. See [Jol14] for more details on FETI method and its implementation in Feel++.

2.5 Conclusion

We conducted in this chapter a generic review of domain decomposition methods. We review various domain decomposition methods including overlapping domain decomposition methods (Schwarz methods) and substructuring domain decomposition methods (mortar element method, three-field method and FETI method). The theoretical investigation presented in this chapter, except for FETI method, will be followed by implementation aspects and numerical simulations in the next chapters.

Chapter 3

Mortar Element Method with Constrained Space in 2D

This chapter investigates the mortar finite element method with constrained space [BMP93a]. The basic notations, functional settings and the description of the mortar method are given. Some technical tools required in the construction and analysis of the substructuring preconditioners [Ber+14] are recalled.

Ce chapitre étudie la méthode des éléments finis mortar avec espace contraint [BMP93a]. Les notations de base, les paramètres fonctionnels et la description de la méthode mortar sont donnés. Quelques outils techniques nécessaires dans la construction et l'analyse des préconditionneurs par sous-structuration [Ber+14] sont rappelés.

Contents

3.1 Model Problem

We focus, for simplicity, on the following simple model problem, even if the results of this work can be easily extended to a more general situation. Let $\Omega \in \mathbb{R}^2$ be a polygonal domain and a given $f \in L^2(\Omega)$; then we find u satisfying

$$-\sum_{i,j=1}^{2} \frac{\partial}{\partial \mathbf{x}_j}\left(a_{ij}(\mathbf{x})\frac{\partial u}{\partial \mathbf{x}_i}\right) = f \ \text{ in } \ \Omega, \qquad u = 0 \ \text{ on } \ \partial\Omega. \tag{3.1}$$

We assume that for almost all $\mathbf{x} \in \Omega$ the matrix $a(\mathbf{x}) = (a_{ij}(\mathbf{x}))_{i,j=1,2}$ is symmetric positive definite, with smallest eigenvalue λ_{min} and largest eigenvalue λ_{max} satisfying

$$\lambda_{min} \geq \alpha > 0, \qquad \lambda_{max} \leq \alpha', \quad \alpha, \alpha' \text{ independent of } \mathbf{x}.$$

In order to discretize the above problem we start by considering a decomposition of Ω as the union of L subdomains Ω_ℓ,

$$\Omega = \bigcup_{\ell=1,\ldots,L} \Omega_\ell. \tag{3.2}$$

which, for simplicity, we assume to be quadrangles.

In the following we will employ the notation $A \lesssim B$ (resp. $A \gtrsim B$) to say that the quantity A is bounded from above (resp. from below) by cB, with a constant c independent of ℓ, of the H_ℓ's, the diameter of the subdomain Ω_ℓ, as well as of any mesh size parameter and of the polynomial degree p_ℓ. The expression $A \simeq B$ will stand for $A \lesssim B \lesssim A$.

We assume that each subdomain Ω_ℓ satisfies the following assumption : there exists orientation preserving bilinear mappings $B_\ell : [0\,,1]^2 \to \Omega_\ell$ such that there exist a constant H_ℓ with

$$H_\ell^{-1}|J(B_\ell)| \lesssim 1, \qquad H_\ell|J(B_\ell^{-1}) \lesssim 1$$

where J denotes the Jacobian matrix and where H_ℓ is the diameter of the subdomain Ω_ℓ.

We set

$$\Gamma_{\ell n} = \partial\Omega_n \cap \partial\Omega_\ell, \qquad \mathcal{S} = \cup\Gamma_{\ell n} \tag{3.3}$$

and we denote by $\gamma_\ell^{(i)}$ ($i = 1\,,\ldots,4$) the i-th side of the ℓ-th domain :

$$\partial\Omega_\ell = \bigcup_{i=1}^{4} \gamma_\ell^{(i)}.$$

For each subdomain Ω_ℓ, let $x_i^\ell, i = 1\,, \ldots, 4$ be the vertices of the subdomain, which we assume to be ordered consecutively, so that each segment $\gamma_\ell^{(i)} = [x_i^\ell\,, x_{i+1}^\ell]$ (for notational simplicity we also introduce the notation $x_5 = x_1$).

Here we deal with the case of a geometrically conforming decomposition : each edge $\gamma_\ell^{(i)}$ coincides with $\Gamma_{\ell n}$ for some n.

3.2 Functional Spaces

Let us at first introduce the necessary functional setting. For $\hat{\Omega}$ any domain in $\mathbb{R}^d$, $d = 1\,, 2$ we introduce the following unscaled norms and seminorms (with $0 < s < 1$) :

$$\|\hat{u}\|^2_{0,\hat{\Omega}} = \int_{\hat{\Omega}} |\hat{u}|^2, \qquad |\hat{u}|^2_{1,\hat{\Omega}} = \int_{\hat{\Omega}} |\nabla u|^2, \qquad |\hat{u}|_{s,\hat{\Omega}} = \int_{\hat{\Omega}} dx \int_{\hat{\Omega}} dy \frac{|\hat{u}(x) - \hat{u}(y)|^2}{|x-y|^{d+2s}}.$$

We then introduce the following suitably scaled norms and seminorms : for two-dimensional entities

$$\|u\|^2_{H^1(\Omega_\ell)} = H_\ell^{-2} \int_{\Omega_\ell} |u|^2\, dx + \int_{\Omega_\ell} |\nabla u|^2\, dx, \qquad |u|^2_{H^1(\Omega_\ell)} = \int_{\Omega_\ell} |\nabla u|^2\, dx, \tag{3.4}$$

and for one dimensional entities

$$|\eta|^2_{H^s(\partial\Omega_\ell)} = H_\ell^{2s-1} \int_{\partial\Omega_\ell} \int_{\partial\Omega_\ell} \frac{|\eta(x) - \eta(y)|^2}{|x-y|^{2s+1}}\, dx\, dy, \qquad s \in (0,1) \tag{3.5}$$

$$\|\eta\|^2_{H^s(\partial\Omega_\ell)} = |\eta|^2_{H^s(\partial\Omega_\ell)} + H_\ell^{-1} \int_{\partial\Omega_\ell} |\eta|^2\, ds, \qquad s \in (0,1). \tag{3.6}$$

Remark that the above norms are defined in such a way that they are scaling invariant, that is they are preserved when Ω_ℓ is rescaled to the reference domain $]0\,, 1[^2$.

In the following for $\gamma_\ell^{(i)}$ edge of Ω_ℓ we will also make explicit use of the spaces $H_0^s(\gamma_\ell^{(i)})$ and $H_{00}^{1/2}(\gamma_\ell^{(i)})$, which are defined as the subspaces of those functions η of $H^s(\gamma_\ell^{(i)})$ (resp. $H^{1/2}(\gamma_\ell^{(i)})$) such that the function $\hat{\eta}$ defined as $\hat{\eta} = \eta$ on $\gamma_\ell^{(i)}$ and $\hat{\eta} = 0$ on $\partial\Omega \setminus \gamma_\ell^{(i)}$ belongs to $H^s(\partial\Omega)$ (resp. to $H^{1/2}(\partial\Omega)$). The spaces $H_0^s(\gamma_\ell^{(i)})$ and $H_{00}^{1/2}(\gamma_\ell^{(i)})$ are endowed with the norms

$$\|\eta\|_{H_0^s(\gamma_\ell^{(i)})} = \|\hat{\eta}\|_{H^s(\gamma_\ell^{(i)})} \qquad\qquad \|\eta\|_{H_{00}^{1/2}(\gamma_\ell^{(i)})} = \|\hat{\eta}\|_{H^{1/2}(\gamma_\ell^{(i)})}.$$

Let the spaces X and T be defined as

$$X = \prod_{\ell} \{u_\ell \in H^1(\Omega_\ell) |\ \ u_\ell = 0 \text{ on } \partial\Omega \cap \partial\Omega_\ell\}, \qquad T = \prod_{\ell} H_*^{1/2}(\partial\Omega_\ell), \tag{3.7}$$

where $H_*^{1/2}(\Omega_\ell)$ is defined by

$$H_*^{1/2}(\partial\Omega_\ell) = H^{1/2}(\partial\Omega_\ell) \qquad \text{if } \partial\Omega_\ell \cap \partial\Omega = \emptyset$$

and

$$H_*^{1/2}(\partial\Omega_\ell) = \{\eta \in H^{1/2}(\partial\Omega_\ell),\ \eta|_{\partial\Omega_\ell \cap \partial\Omega} \equiv 0\} \sim H_{00}^{1/2}(\partial\Omega_\ell \setminus \partial\Omega)$$

otherwise.

3.3 Discretizations

We consider for each ℓ a family $\mathcal{K}^\ell$ of compatible quasi-uniform shape regular decompositions of Ω^ℓ, each made of open elements K (triangular or quadrilateral) depending on a parameter $h_\ell > 0$. We let $\mathcal{V}_h^\ell \subset H^1(\Omega_\ell)$ be a finite element space defined on the decomposition $\mathcal{K}^\ell$ and satisfying an homogeneous boundary condition on $\partial\Omega \cap \partial\Omega_\ell$. We assume that for some integers $p_\ell\,, p'_\ell$ with $1 \leq p_\ell \leq p'_\ell$ we have

$$\mathcal{V}_h^\ell = \left\{v \in C^0(\bar{\Omega}_\ell) \text{ s.t. } v|_K \in P_{p_\ell}(K),\ K \in \mathcal{K}^\ell\right\} \cap H_0^1(\Omega_\ell),$$

where $P_{p_\ell}(K)$ stands for the space of polynomials of degree at most p_ℓ. We set

$$T_h^\ell = \mathcal{V}_h^\ell|_{\partial\Omega_\ell}, \tag{3.8}$$

and, for each edge $\gamma_\ell^{(i)}$ of the subdomain Ω_ℓ, we define

$$T_{\ell,i} = \left\{\eta :\ \ \eta \text{ is the trace on } \gamma_\ell^{(i)} \text{ of some } u_\ell \in \mathcal{V}_h^\ell\ \right\} \tag{3.9}$$

$$T_{\ell,i}^0 = \left\{\eta \in T_{\ell,i} :\ \ \eta = 0 \text{ at the vertices of } \gamma_\ell^{(i)}\right\}. \tag{3.10}$$

Finally, we set

$$X_h = \prod_{\ell=1}^{L} \mathcal{V}_h^\ell \subset X, \qquad T_h = \prod_{\ell=1}^{L} T_h^\ell \subset T. \tag{3.11}$$

On X and T we introduce the following broken norm and semi-norm :

$$\|u\|_X = \left(\sum_\ell \|u\|^2_{1,\Omega_\ell}\right)^{\frac{1}{2}}, \qquad |u|_X = \left(\sum_\ell |u|^2_{1,\Omega_\ell}\right)^{\frac{1}{2}}, \tag{3.12}$$

$$\|\eta\|_T = \left(\sum_\ell \|\eta_\ell\|^2_{1/2,\partial\Omega_\ell}\right)^{1/2} \qquad |\eta|_T = \left(\sum_\ell |\eta_\ell|^2_{1/2,\partial\Omega_\ell}\right)^{1/2}. \tag{3.13}$$

The spaces considered satisfy classical direct and inverse inequalities, see e.g. [BS94 ; Can+06 ; Sch98]. In view of the scaling (3.4), the direct inequalities take the following form : for $0 \le s \le 1$, $s < r \le p_\ell + 1$

$$\inf_{\eta_h \in T_{\ell,i}} |\eta - \eta_h|_{H^s(\gamma_m)} \lesssim p_\ell^{s-r} \left(\frac{h_\ell}{H_\ell}\right)^{r-s} |\eta|_{H^r(\gamma_m)} \qquad \forall \eta \in H^r(\gamma_m) \tag{3.14}$$

$$\inf_{\eta_h \in T^0_{\ell,i}} |\eta - \eta_h|_{H^s(\gamma_m)} \lesssim p_\ell^{s-r} \left(\frac{h_\ell}{H_\ell}\right)^{r-s} |\eta|_{H^r(\gamma_m)} \qquad \forall \eta \in H^r(\gamma_m) \cap H^1_0(\gamma_m) \tag{3.15}$$

while the inverse inequalities take the form for all $\eta \in T_{\ell,i}$ and for all $s\,, r$ such that $0 \le s < r \le 1$

$$\|\eta\|_{H^r(\gamma_m)} \lesssim {p_\ell}^{2(r-s)} \left(\frac{h_\ell}{H_\ell}\right)^{s-r} \|\eta\|_{H^s(\gamma_m)}, \qquad |\eta|_{H^r(\gamma_m)} \lesssim {p_\ell}^{2(r-s)} \left(\frac{h_\ell}{H_\ell}\right)^{s-r} |\eta|_{H^s(\gamma_m)}, \tag{3.16}$$

and for all $\eta \in T^0_{\ell,i}$ and for all $s\,, r \ne 1/2$ such that $0 \le s < r \le 1$

$$\|\eta\|_{H^r_0(\gamma_m)} \lesssim {p_\ell}^{2(r-s)} \left(\frac{h_\ell}{H_\ell}\right)^{s-r} \|\eta\|_{H^s_0(\gamma_m)}, \qquad |\eta|_{H^r_0(\gamma_m)} \lesssim {p_\ell}^{2(r-s)} \left(\frac{h_\ell}{H_\ell}\right)^{s-r} |\eta|_{H^s_0(\gamma_m)}, \tag{3.17}$$

once again with constants independent of r, s. For $s = 1/2$ or $r = 1/2$ (3.17) holds with H^s_0 (resp. H^r_0) replaced by $H^{1/2}_{00}$.

In the following it will be convenient to introduce the following notation :

$$H = H_{\ell^*}, \qquad h = h_{\ell^*}, \qquad p = p_{\ell^*}$$

with

$$\ell^* = \arg\max_\ell \frac{H_\ell p_\ell^2}{h_\ell}, \quad \text{and} \quad \hat{p} = \max_\ell p_\ell.$$

3.4 Classical Bounds

With the chosen scaling, saveral classical bounds hold with constants independent of H_ℓ. In particular we have :

Trace bound

For all $u \in H^1(\Omega_\ell)$ we have, see [LM72]

$$\|u\|_{H^{1/2}(\partial\Omega_\ell)} \lesssim \|u\|_{H^1(\Omega_\ell)}, \qquad |u|_{H^{1/2}(\Omega_\ell)} \lesssim |u|_{H^1(\Omega_\ell)}. \tag{3.18}$$

Injection of H^s in L^∞ for $s > 1/2$

For all $\eta \in H^s(\gamma)$, $s > 1/2$, γ being either $\gamma_\ell^{(i)}$ or $\partial\Omega_\ell$,

$$\|\eta\|_{L^\infty(\gamma)} \lesssim \frac{1}{\sqrt{2s-1}} \|\eta\|_{H^s(\gamma)}. \tag{3.19}$$

Poincaré type inequalities

for all $\eta \in H_0^s(\gamma_\ell^{(i)})$ it holds that

$$\|\eta\|_{H_0^s(\gamma_\ell^{(i)})} \lesssim |\eta|_{H_0^s(\gamma_\ell^{(i)})} \tag{3.20}$$

and for all η with $\displaystyle\int_\gamma \eta = 0$, γ being either $\gamma_\ell^{(i)}$ or $\partial\Omega_\ell$, it holds that

$$\|\eta\|_{H^s(\gamma)} \lesssim |\eta|_{H^s(\gamma)}. \tag{3.21}$$

Injection of H^s in H_0^s for $s < 1/2$

We recall that for $s < 1/2$ the spaces $H^s(\gamma_\ell^{(i)})$ and $H_0^s(\gamma_\ell^{(i)})$ coincide as sets and have equivalent norms. However, the constants in the norm equivalence goes to infinity as s tends to $1/2$. For all $\varphi \in H^s(\gamma_\ell^{(i)})$ the following bound can be shown, see [BF11] : for $\beta \in \mathbb{R}$ arbitrary it holds that

$$|\varphi|_{H_0^s(\gamma_\ell^{(i)})} \lesssim \frac{1}{1/2-s} \|\varphi - \beta\|_{H^{1/2}(\gamma_\ell^{(i)})} + \frac{1}{\sqrt{1/2-s}} |\beta|. \tag{3.22}$$

If φ is linear, the bound (3.22) can be improved to

$$|\varphi|_{H_0^s(\gamma_\ell^{(i)})} \lesssim \frac{1}{\sqrt{1/2-s}} (\|\varphi - \beta\|_{H^{1/2}(\gamma_\ell^{(i)})} + \frac{1}{\sqrt{1/2-s}} |\beta|). \tag{3.23}$$

Technical tools

We now revise some technical tools that will be required in the construction and analysis of our preconditioner. We observe the following result, that corresponds to the hp-version of [Ber03, Lemma 3.1] and of [BPS86, Lemma 3.4], see e.g. [GC96] for the proof.

Lemma 3.4.1. *The following bounds hold :*

- *for all $\xi \in T_h^\ell$ and γ being either $\gamma_\ell^{(i)}$ or $\partial\Omega_\ell$, it holds*

$$\|\xi\|^2_{L^\infty(\gamma)} \lesssim \left(1 + \log\left(\frac{H_\ell p_\ell^2}{h_\ell}\right)\right) \|\xi\|^2_{H^{1/2}(\gamma)}; \tag{3.24}$$

- *for all $\xi \in T_h^\ell$ such that $\xi(P) = 0$ for some $P \in \gamma$, γ being either $\gamma_\ell^{(i)}$ or $\partial\Omega_\ell$, it holds*

$$\|\xi\|^2_{L^\infty(\gamma)} \lesssim \left(1 + \log\left(\frac{H_\ell p_\ell^2}{h_\ell}\right)\right) |\xi|^2_{1/2,\gamma}; \tag{3.25}$$

- *for all $\xi \in T_h^\ell$, letting x_i^ℓ and x_{i+1}^ℓ denote the two extrema of the segment $\gamma_\ell^{(i)}$, we have*

$$(\xi(x_i^\ell) - \xi(x_{i+1}^\ell))^2 \lesssim \left(1 + \log\left(\frac{H_\ell p_\ell^2}{h_\ell}\right)\right) |\xi|^2_{H^{1/2}(\gamma_\ell^{(i)})}; \tag{3.26}$$

- *for all $\xi \in T^0_{\ell,i}$ it holds*

$$\|\xi\|^2_{H^{1/2}_{00}(\gamma_\ell^{(i)})} \lesssim \left(1 + \log\left(\frac{H_\ell p_\ell^2}{h_\ell}\right)\right)^2 |\xi|^2_{H^{1/2}(\gamma_\ell^{(i)})}. \tag{3.27}$$

- *for all $\zeta_L \in H^{1/2}(\partial\Omega_\ell)$, ζ_L linear on each edge of Ω_ℓ, we have*

$$|\zeta_L|^2_{H^{1/2}(\partial\Omega_\ell)} \lesssim \sum_{i=1}^{4} (\zeta_L(x_{i+1}^\ell) - \zeta_L(x_i^\ell))^2. \tag{3.28}$$

The following result is a generalization to the h-p version of Lemmas 3.2, 3.4 and 3.5 of [BPS86].

Lemma 3.4.2. *Let $\xi \in T_h^\ell$ such that $\xi(x_i^\ell) = 0$, $i = 1, \ldots, 4$, and let $\zeta_L \in H^{1/2}(\partial\Omega_\ell)$, ζ_L linear on each edge of Ω_ℓ. Then it holds*

$$\sum_{i=1}^{4} \|\xi\|^2_{H^{1/2}_{00}(\gamma_\ell^{(i)})} \lesssim \left(1 + \log\left(\frac{H_\ell p_\ell^2}{h_\ell}\right)\right)^2 \|\xi + \zeta_L\|^2_{H^{1/2}(\partial\Omega_\ell)}. \tag{3.29}$$

The following lemma holds.

Lemma 3.4.3. *Let $\sigma : \mathbb{R}^L \times \mathbb{R}^L \to \mathbb{R}$ be defined as*

$$\sigma(\alpha, \beta) = \sum_{\ell,n:|\Gamma_{\ell n}|>0} (\alpha_\ell - \alpha_n)(\beta_\ell - \beta_n). \tag{3.30}$$

For $\eta \in T$ let $\bar{\eta}$ be defined by

$$\bar{\eta} = (\bar{\eta}_\ell)_{\ell=1,\dots,L}, \qquad \bar{\eta}^\ell = |\partial\Omega_\ell|^{-1} \int_{\Omega_\ell} \eta^\ell \tag{3.31}$$

Then, if $\eta \in T$ verifies

$$\int_{\gamma_m} [\eta] = 0, \quad \forall m = (\ell, i) \in I, \tag{3.32}$$

we have

$$\sigma(\bar{\eta}, \bar{\eta}) \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right) |\eta|_T^2. \tag{3.33}$$

See section E in Appendices for the proof of Lemma 3.4.3.

3.5 Mortar Problem

Let $a_X : X \times X \longrightarrow \mathbb{R}$ be a composite bilinear form defined as follows :

$$a_X(u, v) = \sum_\ell \int_{\Omega_\ell} \sum_{i,j} a_{ij}(x) \frac{\partial u_\ell}{\partial x_i} \frac{\partial v_\ell}{\partial x_j} \, dx. \tag{3.34}$$

The bilinear form a_X is clearly not coercive on X. In order to obtain a well posed problem, we will consider proper subspaces of X, consisting of functions satisfying a suitable weak continuity constraint.

According to the mortar method, for defining such weak continuity constraint, we start by choosing for each segment $\Gamma_{\ell,\ell'} = \gamma_\ell^{(i)} = \gamma_{\ell'}^{(i')}$, one side (for example ℓ) to be the master side, while the other side to be the 'multiplier side" (in the usual terminology these are called "non mortars" or "slave sides"). More precisely, we choose an index set $I \subset \{1, \dots, L\} \times \{1, \dots, 4\}$ such that,

$$\mathcal{S} = \bigcup_{(\ell,i)\in I} \gamma_\ell^{(i)}, \qquad \begin{matrix} (\ell_1, i_1), (\ell_2, i_2) \in I, \\ (\ell_1, i_1) \neq (\ell_2, i_2) \end{matrix} \Rightarrow \gamma_{\ell_1}^{(i_1)} \cap \gamma_{\ell_2}^{(i_2)} = \emptyset. \tag{3.35}$$

Furthermore we will denote by $I^* \subset \{1, \dots, L\} \times \{1, \dots, 4\}$ the index-set corresponding to "trace sides" ("mortars" or "master sides" in the usual terminology), which is defined in such a way that $I^* \cap I = \emptyset$ and $\mathcal{S} = \cup_{(\ell,i)\in I^*} \gamma_\ell^{(i)}$.

For each $m = (\ell\,, i) \in I$, let $a_0 = x_i^\ell < a_1 < \ldots < a_{M-1} < a_M = x_{i+1}^\ell$ denotes the one dimensional mesh induced on γ_m by the two dimensional mesh $\mathcal{K}_\ell$. Let $e_i = (a_{i-1}\,, a_i)$ and let the finite dimensional multiplier space M_h^m on γ_m, be defined as

$$M_h^m = \big\{v \in C^0(\gamma_m),\ v|_{e_i} \in P_{p_\ell}(e_i), i \neq 1, M,\ v|_{e_1} \in P_{p_\ell - 1}(e_1),\ v|_{e_M} \in P_{p_\ell - 1}(e_M)\big\}. \quad (3.36)$$

Remark that $dim(M_h^m) = dim(T_m^0)$.
We set :

$$M_h = \big\{\eta \in H^{-1/2}(\mathcal{S}),\ \forall m \in I\ \ \eta|_{\gamma_m} \in M_h^m\big\} \sim \prod_{m \in \mathcal{I}} M_m. \quad (3.37)$$

The constrained approximation and trace spaces $\mathcal{X}_h$ and $\mathcal{T}_h$ are then defined as follows :

$$\mathcal{X}_h = \big\{v_h \in X_h, \quad \int_S [v_h]\lambda\, ds = 0,\ \forall \lambda \in M_h\big\} \quad (3.38)$$

$$\mathcal{T}_h = \big\{\eta \in T_h, \int_S [\eta]\lambda\, ds = 0,\ \forall \lambda \in M_h\big\}. \quad (3.39)$$

where, on $\gamma_\ell^{(i)} = \gamma_n^{(j)}$, $(\ell\,, i) \in I$, we set $[\eta] = \eta_\ell - \eta_n$.

We can now introduce the following discrete problem :

Problem 3.5.1. *Find $u_h \in \mathcal{X}_h$ such that for all $v_h \in \mathcal{X}_h$*

$$a_X(u_h, v_h) = \int_\Omega f v_h\, dx. \quad (3.40)$$

It is known that Problem 3.5.1 admits a unique solution u_h which satisfies the following error estimate, see [BSS00].

Theorem 3.5.1. *Assume that the exact solution $u \in H^1(\Omega)$ of the problem 3.5.1 is such that $u_h \in H^{\tau_\ell}(\Omega_\ell)$, $\tau_\ell \geq 1$. Then, for any $\varepsilon > 0$ there exists $C(\varepsilon) > 0$ such that the solution u_h of* (3.1) *satisfies*

$$\|u - u_h\|_X \leq C(\varepsilon) \sum_{\ell=1}^{L} \frac{h_\ell^{\eta_\ell - 1}}{{p_\ell}^{\tau_\ell - 1}} {p_\ell}^{\varepsilon} \left(\|u_h\|_{H^{\tau_\ell}(\Omega_\ell)} + \|f_\ell\|_{L^2(\Omega_\ell)}\right)$$

where $\eta_\ell = \min(\tau_\ell\,, p_\ell + 1)$.

3.6 Mortar Correction Operator

For all $m = (\ell\,, i) \in I$ ($\gamma_\ell^{(i)}$ slave side), we let $\pi_m : L^2(\gamma_m) \longrightarrow T_m^0$ be the bounded projector defined as

$$\int_{\gamma_m} (\eta - \pi_m \eta)\lambda = 0, \quad \forall \lambda \in M_m. \tag{3.41}$$

The projection π_m is well defined and satisfies (see [SS00 ; SS98])

Theorem 3.6.1. *For $m = (\ell\,, i) \in I$ it holds :*

$$\|\pi_m \eta\|_{L^2(\gamma_m)} \lesssim p_\ell^{\frac{1}{2}} \|\eta\|_{L^2(\gamma_m)} \quad \forall \eta \in L^2(\gamma_m) \tag{3.42}$$

$$|\pi_m \eta|_{H^1(\gamma_m)} \lesssim p_\ell \, |\eta|_{H^1(\gamma_m)} \quad \forall \eta \in H_0^1(\gamma_m). \tag{3.43}$$

Remark 3.6.2. The problem of whether (3.42) and (3.43) are optimal was studied in [SS98], where, through an eigenvalue analysis the dependence on p to the power $1/2$ and 1 of the norm of the projector appearing in (3.42) and (3.43) were confirmed. This dependence does not seem to affect the asymptotic rate of the error, which, as observed in [SS98] seems to be only slightly suboptimal (loss of a factor $C(\varepsilon)p^\varepsilon$ for ε arbitrarily small). In [BSS00] this good behavior of the error was proven, for sufficiently smooth solutions, thanks to an interpolation argument.

By space interpolation and using the Poincaré inequality we immediately get the following corollary

Corollary 3.6.3. *For all s, $0 < s < 1$, $s \neq 1/2$, for all $\eta \in H_0^s(\gamma_m)$ we have*

$$|\pi_m \eta|_{H_0^s(\gamma_m)} \lesssim p_\ell^{(1+s)/2} \, |\eta|_{H_0^s(\gamma_m)}, \tag{3.44}$$

uniformly in s. For all $\eta \in H_{00}^{1/2}(\gamma_m)$ we have

$$|\pi_m \eta|_{H_{00}^{1/2}(\gamma_m)} \lesssim p_\ell^{3/4} \, |\eta|_{H_{00}^{1/2}(\gamma_m)}. \tag{3.45}$$

We now define a global linear operator

$$\pi_h : \prod_{\ell=1}^{L} L^2(\partial\Omega_\ell) \longrightarrow \prod_{\ell=1}^{L} L^2(\partial\Omega_\ell)$$

as follows : for $\eta = (\eta_\ell)_{\ell=1,\dots,L} \in \Pi_\ell L^2(\partial\Omega_\ell)$, we set $\pi_h(\eta) = (\eta_\ell^*)_{\ell=1,\dots,L}$, where $\eta_\ell^* \in T_h^\ell$ is defined on multiplier sides as π_m applied to the jump of η, while it is set identically zero on trace sides and on the external boundary $\partial\Omega$: on $\gamma_m = \gamma_\ell^{(i)} = \gamma_n^{(j)}$, $(\ell\,, i) \in I$, $(n\,, j) \in I^*$ (ℓ slave side)

$$\eta_\ell^*|_{\gamma_m} = \pi_m([\eta]|_{\gamma_m}), \qquad \eta_n^*|_{\gamma_m} = 0,$$

and for all ℓ

$$\eta_\ell^* = 0 \text{ on } \partial\Omega_\ell \cap \partial\Omega.$$

The following bound holds

Lemma 3.6.4. *For all $\eta = (\eta_\ell)_{\ell=1,\dots,L} \in T$ and for all $\alpha = (\alpha_\ell)_{\ell=1,\dots,L}$, α_ℓ constant in Ω_ℓ, it holds*

$$|(Id - \pi_h)(\eta)|_T^2 \lesssim \hat{p}^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2 \|\eta - \alpha\|_T^2 + \hat{p}^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)\sigma(\alpha,\alpha). \tag{3.46}$$

where we recall that the bilinear form σ is defined in (3.30).
If, in addition, each η_ℓ is linear on each $\gamma_\ell^{(i)}$, then the bound can be improved to

$$|(Id - \pi_h)(\eta)|_T^2 \lesssim \hat{p}^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)\left(\|\eta - \alpha\|_T + \sigma(\alpha,\alpha)\right). \tag{3.47}$$

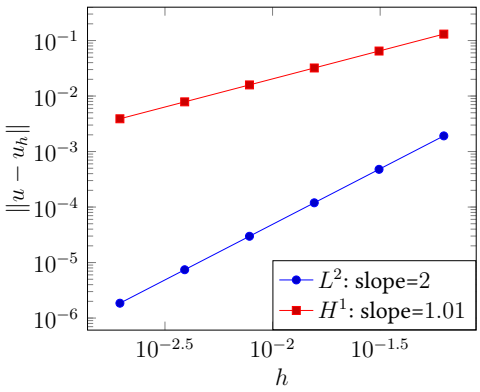
Corollary 3.6.5. *Let $\eta \in T$ and let $\bar{\eta} = (\bar{\eta}_\ell)_{\ell=1,\dots,L}$ be defined by* (3.31). *Then*

$$|\pi_h(\eta)|_T^2 \lesssim \hat{p}^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2 |\eta|_T^2 + \hat{p}^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)\sigma(\bar{\eta},\bar{\eta}).$$

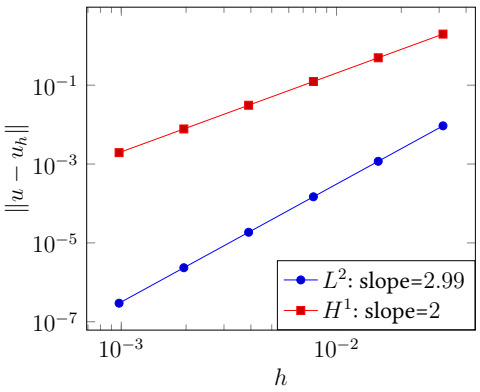
See section E in Appendices for the proof of Lemma 3.6.4.

3.7 Convergence Analysis

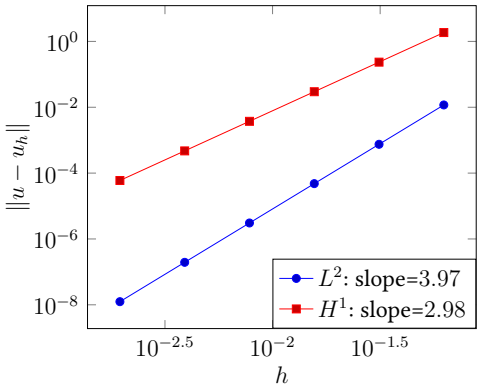
We analyze the convergence properties of the mortar finite element method described in this chapter in accordance with the theoretical error estimations for finite element method [EG04 ; Can+06]. We report in FIGURE 3.1 and FIGURE 3.2 the L^2-norm error $\|u - u_h\|_{L^2}$ and H^1-norm error $\|u - u_h\|_{H^1}$ as a function of the characteristic mesh size h in logarithmic axis for different Lagrange polynomial orders $p = 1, 2, 3, 4, 5$. We conduct this convergence analysis by considering conforming domain decompositions (see FIGURE 3.1) and nonconforming domain decompositions (see FIGURE 3.2). The number of subdomains is equal to 64 fixed. The chosen analytical solution is $\sin(15\pi x)\sin(10\pi y)$.



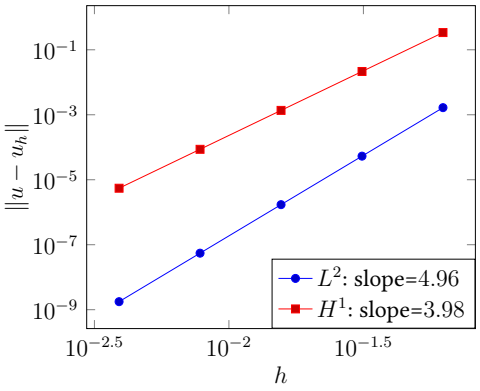
3.1.1. P1 mortar FEM



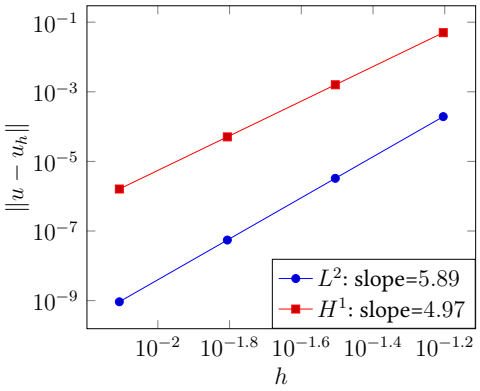
3.1.2. P2 mortar FEM



3.1.3. P3 mortar FEM

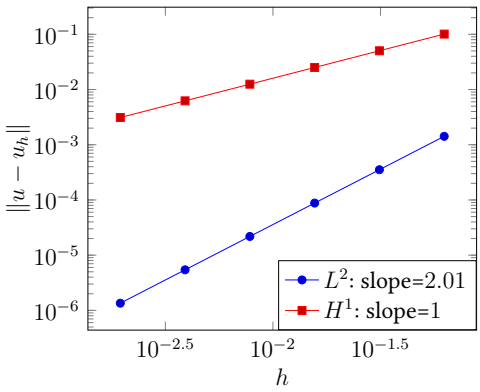


3.1.4. P4 mortar FEM

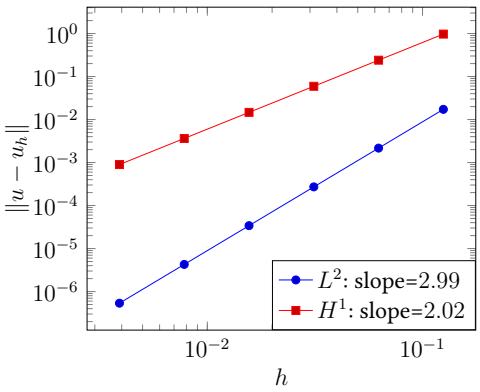


3.1.5. P5 mortar FEM

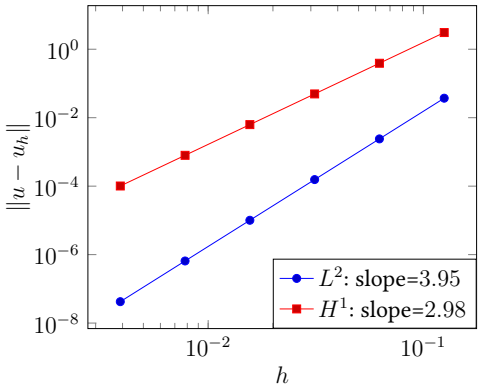
Figure 3.1 : Convergence analysis with conforming domain decompositions for $p = 1, 2, 3, 4, 5$



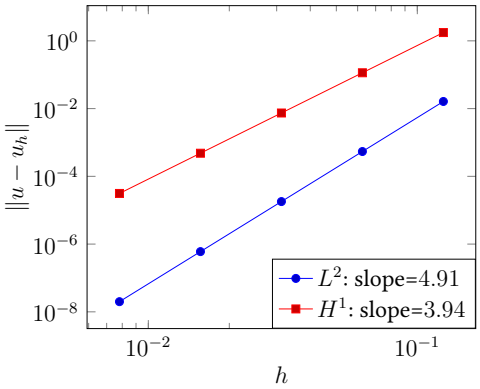
3.2.1. P1 mortar FEM



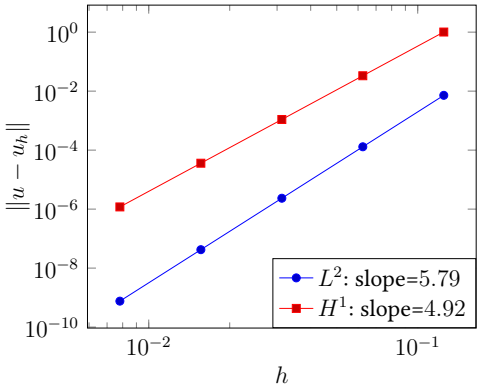
3.2.2. P2 mortar FEM



3.2.3. P3 mortar FEM



3.2.4. P4 mortar FEM



3.2.5. P5 mortar FEM

Figure 3.2 : Convergence analysis with nonconforming domain decompositions for $p = 1, 2, 3, 4, 5$

The plots in Figure 3.1 and in Figure 3.2 show that the convergence properties (L^2 and H^1) of our mortar finite element framework are consistent with the finite element theoretical error estimations for both conforming and nonconforming domain decompositions. These

convergence results clearly indicate that this framework properly supports the linear finite elements and the high-order finite elements.

3.8 Conclusion

In this chapter, we studied the h-p mortar element method with constrained space, previously introduced briefly in chapter 2. We reminded the mortar formulation including the functional settings, the discretization aspects and the mortar correction operator. As we recalled in chapter 2, the efficient preconditioners are essential for solving the linear system arising from this discretization. For this purpose, we revised in this chapter some technical tools that will be required in the construction and analysis of our proposed substructuring preconditioners which will be presented in chapter 4.

The theoretical results discussed in this chapter will be followed by the numerical implementation in chapter 6 and the numerical simulations supporting the mathematical properties in chapter 8. The scalability analysis including the speedup and the efficiency of the parallel algorithms will be also available in the same chapter.

Chapter 4

Substructuring Preconditioners for Mortar Element Method in 2D

The construction of the substructuring preconditioners for h-p mortar finite element method is analyzed. The substructuring approach, whose the main idea was proposed in [BP04] for the case of linear finite elements, is considered in section 4.1. Particular emphasis is placed on the vertex block of the preconditioners in section 4.2. The algebraic forms for the realization of the preconditioners and discrete Steklov-Poincaré operator are given in section 4.3.

La construction des préconditionneurs par sous-structuration pour la méthode des éléments finis h-p mortar est analysée. L'approche de sous-structuration, dont l'idée a été proposée dans [BP04] pour le cas des éléments finis linéaires est considérée dans la section 4.1. Une attention particulière est placée sur le bloc vertex des préconditionneurs dans la section 4.2. Les formes algébriques pour la réalisation des préconditionneurs et de l'opérateur Steklov-Poincaré discret sont données dans la section 4.3.

Contents

In this chapter, we deal with the construction of preconditioners for the h-p mortar finite element method. We start by considering the approach proposed in the framework of conforming domain decomposition by J.H. Bramble, J.E. Pasciak and A.H. Schatz [BPS86], which has already been extended to the h version of the Mortar method by Achdou, Maday, Widlund [AMW99]. In doing this we will extend to the h-p version some tools that are common to the analysis of a wide range of substructuring preconditioner. This approach consists in considering a suitable splitting of the nonconforming discretization space in terms of "interior", "edge" and "vertex" degrees of freedom and then using the related block-Jacobi type preconditioners. While the "interior" and the "edge" blocks can be treated essentially as in the conforming case, the treatment of the vertex block deserves some additional considerations.

In fact, a problem that, in our opinion, has not until now been tackled in a satifactory way for the mortar method, is the design of the coarse vertex block of the preconditioner, which is responsible for the good scaling properties of the preconditioners considered. Indeed, when building preconditioners for the Mortar method we have to deal with the fact that the coarse space depends on the fine discretization, via the the action of the "mortar projection operator". Moreover, the design of such block is further complicated by the the presence of multiple degrees of freedom at each cross point (we recall, in fact, that in the definition of the mortar method, continuity at cross points is not required). The solution considered in [AMW99] is to use as a coarse preconditioner the vertex block of the Schur complement. This is clearly not efficient, since it implies actually assembling at least a block of the Schur complement (which is a task that we would like to avoid) and, for a high number of subdomains, it is definitely not practically feasible. Here we propose two different coarse preconditioners. The first one is the vertex block of the Schur complement for a fixed auxiliary order one mesh with a small number of degrees of freedom per subdomain. This idea was presented in [BP04] for the case of linear finite elements. We combine it, here, with a suitable balancing between vertex and edge component, yielding a better estimate for the condition number of the preconditioned matrix. This alternative makes it possible to avoid the need of recomputing the coarse block of the preconditioner when refining the mesh. It still demands assembling a Schur complement matrix (though starting from a coarse mesh) and it is therefore quite expensive, at least when considering a large number of subdomains. In order to be able to tackle this kind of configuration, and obtain a feasible, scalable method even in a massively parallel environment we propose here, as a further alternative, to build the coarse preconditioner by giving up weak continuity and use, as a coarse preconditioner, a (non consistent) Discontinuous Galerkin type interior penalty method defined on the coarse mesh whose elements are the (quadrangular) subdomains. This approach turns out to be quite efficient even for a very a large number of subdomains, as we show in the numerical tests section.

By applying the theoretical approach first presented in [Ber04], that allows to provide a much more general analysis than [BPS86 ; AMW99], we are able to prove, for both choices of the coarse preconditioner, a condition number bound for the preconditioned matrix of the

form

$$Cond(P^{-1}S) \lesssim p^{3/2}(1+\log\left(Hp^2/h\right)^2,$$

where H, h and p are the subdomain mesh-size, the fine mesh-size and the polynomial order respectively, see Corollary 4.2.2 and Theorem 4.2.5. The numerical experiments seem, however, to indicate that this bound might not be optimal. The condition number appears to behave in a polylogarithmic way, and there is no numerical evidence of the presence of the factor $p^{3/2}$. The same kind of behavior, loss of a power of p in the theoretical estimate that does not appear in the numerical tests, was observed also for the first error estimates for the h-p mortar method. Such estimate was then improved by applying an interpolation argument [BSS00] that, unfortunately, cannot be applied for the type of bound that we are considering. The factor $p^{3/2}$ in the theoretical estimate derives from the boundedness estimates for the mortar projector, which were shown to be sharp in [SS98]. We observe that the norm of such projection operator also comes into play in the analysis of other preconditioners (like, for instance, the FETI method) so that a generalization of the related theoretical estimates to the h-p version would also suffer of the loss of a factor $p^{3/2}$.

4.1 Substructuring Approach

The main idea of substructuring preconditioners consists in distinguishing three types of degrees of freedom : *interior* degrees of freedom (corresponding to basis functions vanishing on the skeleton and supported on one sub-domain), *edge* degrees of freedom, and *vertex* degrees of freedom. Then, we can split the functions $u \in \mathcal{X}_h$ as the sum of three suitably defined components : $u = u^0 + u^E + u^V$ and, when expressed in a basis related to such a splitting, substructuring preconditioners can be written in a block diagonal form.

Consequently, given any discrete function $w = (w_\ell)_{\ell=1,\dots,L} \in X_h$ we can split it in a unique way as the sum of an *interior* function $w^0 \in \mathcal{X}_h^0$ and a discrete lifting, performed subdomain-wise of its trace $\eta(w) = (w^\ell|_{\Omega_\ell})_{\ell=1,\dots,L}$ which for notational simplicity we denote by $R_h(w)$ (rather than $R_h(\eta(w))$) :

$$w = w^0 + R_h(w), \qquad w^0 \in \mathcal{X}_h^0,$$

with $R_h(w) = (R_h^\ell(w_\ell))_{\ell=1,\dots,L}$, $R_h^\ell(w_\ell)$ being the unique element in $\mathcal{V}_h^\ell$ satisfying

$$R_h^\ell(w_\ell) = w_\ell \text{ on } \Gamma_\ell, \qquad a_l(R_h^\ell(w_\ell), v_h^\ell) = 0, \quad \forall v_h \in \mathcal{V}_h^\ell \cap H_0^1(\Omega_\ell).$$

Thus the spaces X_h of unconstrained functions and $\mathcal{X}_h$ of constrained functions can be split as direct sums of an interior and of a (respectively unconstrained or constrained) trace component :

$$X_h = X_h^0 \oplus R_h(T_h), \qquad \mathcal{X}_h = \mathcal{X}_h^0 \oplus R_h(\mathcal{T}_h). \tag{4.1}$$

We can easily verify that $a_X : X_h \times X_h \to \mathbb{R}$ satisfies

$$a_X(w, v) = a_X(w^0, v^0) + a_X(R_h(w), R_h(v)) := a_X(w^0, v^0) + s(\eta(w), \eta(v)), \tag{4.2}$$

where the *discrete Steklov-Poincaré* operator $s : T_h \times T_h \to \mathbb{R}$ is defined by

$$s(\xi, \eta) := \sum_{\ell} a_\ell(R_h^\ell(\xi)), R_h^\ell(\eta)). \tag{4.3}$$

Finally, it is well known that

$$\|R_h^\ell \eta\|_{H^1(\Omega_\ell)} \simeq \|\eta\|_{1/2,\partial\Omega_\ell}, \qquad |R_h^\ell \eta|_{H^1(\Omega_\ell)} \simeq |\eta|_{1/2,\partial\Omega_\ell}. \tag{4.4}$$

see [BS94 ; SS00], whence

$$\|R_h(\eta)\|_X \simeq \|\eta\|_T, \qquad |R_h(\eta)|_X \simeq |\eta|_T. \tag{4.5}$$

The following result for the *Steklov–Poincaré* operator follows easily from the definition of $s(\,\cdot\,,\,\cdot\,)$, the continuity and coercivity of $a_X(\,\cdot\,,\,\cdot\,)$ and (4.5).

Corollary 4.1.1. *For all $\xi \in T_h$, it holds*

$$s(\xi, \xi) \simeq |\xi|_T^2, \tag{4.6}$$

The problem of preconditioning the matrix A associated to the discretization of a_X, reduces to finding good preconditioners for the matrices A_0 and $\mathbf{S}$ corresponding respectively to the bilinear forms a_X restricted to $\mathcal{X}_h^0$ and s. Here we concentrate only on the discrete Steklov-Poincaré operator s assuming to have good preconditioners for the stiffness matrix A_0.

We start by observing that the space of constrained skeleton functions $\mathcal{T}_h$ can be further split as the sum of *vertex* and *edge* functions. More specifically, if we denote by $\mathfrak{L}$ the space

$$\mathfrak{L} = \{(\eta_\ell)_{\ell=1,\dots,L},\ \eta_\ell \in C^0(\partial\Omega_\ell) \text{ is linear on each edge of } \Omega_\ell\}, \tag{4.7}$$

then we can define the space of constrained *vertex* functions as

$$\mathcal{T}_h^V = (Id - \pi_h)\mathfrak{L}. \tag{4.8}$$

We observe that $\mathfrak{L} \subset T_h$, which yields $\mathcal{T}_h^V \subset \mathcal{T}_h$.
We then introduce the space of constrained *edge* functions $\mathcal{T}_h^E \subset \mathcal{T}_h$ defined by

$$\mathcal{T}_h^E = \left\{\eta = (\eta_\ell)_{\ell=1,\dots,L} \in \mathcal{T}_h,\ \eta_\ell(x_i^\ell) = 0,\ i = 1, \dots, 4\right\} \tag{4.9}$$

and we can easily verify that

$$\mathcal{T}_h = \mathcal{T}_h^V \oplus \mathcal{T}_h^E. \tag{4.10}$$

Moreover it is quite simple to check that a function in $\mathcal{T}_h^E$ is uniquely defined by its value on trace edges, the value on multiplier edges being forced by the constraint.

It will be useful in the following to introduce the linear interpolation operator $\Lambda : T_h \to \mathfrak{L}$ defined as

$$\Lambda\eta = (\Lambda^\ell \eta_\ell)_{\ell=1,\dots,L}, \qquad \Lambda^\ell \eta^\ell(x_i^\ell) = \eta(x_i^\ell),\ i = 1,\dots,4.$$

For $\eta \in \mathcal{T}_h$ we observe that $(1-\pi_h)\Lambda\eta \in \mathcal{T}_h^V$ and $\eta - (1-\pi_h)\Lambda\eta \in \mathcal{T}_h^E$. The following Lemma holds [GC96].

Lemma 4.1.2. *For all $\eta = (\eta_\ell)_\ell \in T_h$, it holds*

$$|\Lambda\eta|_T^2 \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)|\eta|_T^2, \qquad \|\Lambda\eta\|_T^2 \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)\|\eta\|_T^2 \tag{4.11}$$

The preconditioner that we consider is built by introducing bilinear forms :

$$b^E : \mathcal{T}_h^E \times \mathcal{T}_h^E \to \mathbb{R}, \qquad b^V : \mathcal{T}_h^V \times \mathcal{T}_h^V \to \mathbb{R}.$$

Let us start by introducing the bilinear form relative to the edges : for any trace side $\gamma_\ell^{(i)}$, $m = (\ell\,, i) \in I^*$, let $b_{\ell,i} : T_{\ell,i}^0 \times T_{\ell,i}^0 \longrightarrow \mathbb{R}$ be a symmetric bilinear form satisfying for all $\eta \in T_{\ell,i}^0$

$$b_{\ell,i}(\eta,\eta) \simeq \|\eta\|^2_{H_{00}^{1/2}(\gamma_\ell^{(i)})}. \tag{4.12}$$

Then, the edge block diagonal global bilinear form $b^E : \mathcal{T}_h^E \times \mathcal{T}_h^E \longrightarrow \mathbb{R}$ here considered is defined by

$$b^E(\eta,\xi) = \sum_{(\ell,i)\in I^*} b_{\ell,i}(\eta_\ell, \xi_\ell). \tag{4.13}$$

Applying Lemma 3.4.2 we easily get

$$b^E(\eta^E,\eta^E) \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2 s(\eta,\eta). \tag{4.14}$$

Moreover, using the fact that η^E verifies the weak continuity constraint and thar η_ℓ^E vanishes at the cross points we immediately get that for $m = (\ell\,, i) \in I$ and $k = (n\,, k) \in I^*$, we have $\eta_\ell^E|_{\gamma_\ell^{(i)}} = \pi_m(\eta_n^E|_{\gamma_n^{(j)}})$

$$|\eta_\ell^E|_{H_{00}^{1/2}(\gamma_\ell^{(i)})} \lesssim \hat{p}^{3/2}|\eta_\ell^E|_{H_{00}^{1/2}(\gamma_n^{(j)})},$$

which allows us to write

$$|\eta^E|_T^2 \lesssim \sum_{m=(\ell,i)\in I^*} |\eta_\ell^E|^2_{H_{00}^{1/2}(\gamma_\ell^{(i)})} + \sum_{m=(\ell,i)\in I} |\eta_\ell^E|^2_{H_{00}^{1/2}(\gamma_\ell^{(i)})} \tag{4.15}$$

$$\lesssim \hat{p}^{3/2} \sum_{m=(\ell,i)\in I^*} |\eta_\ell^E|^2_{H_{00}^{1/2}(\gamma_\ell^{(i)})} \lesssim \hat{p}^{3/2} b^E(\eta^E,\eta^E). \tag{4.16}$$

The construction of the vertex block of the preconditioner in the mortar method framework is not standard, since we need to take into account the weak continuity constraint. In the P1 framework Achdou [AMW99], Maday and Widlund propose to use

$$b_0^V(\eta^V,\zeta^V) = s(\eta^V,\zeta^V). \tag{4.17}$$

This choice immediately yields the bound

$$s(\eta,\eta) \lesssim b_0^V(\eta^V,\eta^V) + \hat{p}^{3/2} b^E(\eta^E,\eta^E).$$

Let us bound $b_0^V(\eta^V,\eta^V)$ in terms of $s(\eta,\eta)$. Let $\bar{\eta} = (\bar{\eta}_\ell)_{\ell=1,\dots,L}$ be defined as in (3.31). Using Lemma 3.6.4 (and in particular (3.47)) we can write

$$b_0^V(\eta^V,\eta^V) \lesssim |(1-\pi_h)\Lambda\eta|_T^2 \lesssim \hat{p}^{3/2}\left(1+\log\left(\frac{Hp^2}{h}\right)\right)\left(\|\Lambda(\eta-\bar{\eta})\|_T^2 + \sigma(\bar{\eta},\bar{\eta})\right).$$

(where we used that $\Lambda\bar{\eta} = \bar{\eta}$). We now use a Poincaré inequality, Lemma 4.1.2 and Lemma 3.4.3, and obtain

$$b_0^V(\eta^V,\eta^V) \lesssim \hat{p}^{3/2}\left(1+\log\left(\frac{Hp^2}{h}\right)\right)^2 |\eta|_T^2.$$

Then we have

$$b_0^V(\eta^V,\eta^V) + \hat{p}^{3/2} b^E(\eta^E,\eta^E) \lesssim \hat{p}^{3/2}\left(1+\log\left(\frac{Hp^2}{h}\right)\right)^2 s(\eta,\eta).$$

This bound would suggest to choose, as a preconditioner for the matrix $\mathbf{S}$, the matrix $\mathbf{P}_0$ corresponding to the bilinear form

$$s_0(\eta,\zeta) = b_0^V(\eta^V,\zeta^V) + \hat{p}^{3/2} b^E(\eta^E,\zeta^E).$$

With this choice we would have the bound

$$Cond(\mathbf{P}_0^{-1}\mathbf{S}) \lesssim \hat{p}^{3/2}\left(1+\log\left(\frac{Hp^2}{h}\right)\right)^2. \tag{4.18}$$

4.2 Vertex Block of the Preconditioner

Building the vertex block of the preconditioner according to (4.17) for fine meshes turns out to be quite expensive, since it implies assembling at least a portion of the Schur complement matrix **S**. In the the present section we propose two more efficient alternatives.

4.2.1 A Coarse Vertex Block Preconditioner

The first option that we considered is to build the vertex block of the preconditioner using a fixed auxiliary coarse mesh, independent of the space discretisation and of the polynomial degree. This idea was presented in [BP04] for the case of P1 finite elements. We combine it here with a suitable balancing between vertex and edge component, yielding a better estimate for the condition number of the preconditioned matrix.

Let n_c be a fixed small integer. We build coarse auxiliary quasi-uniform triangular meshes $\mathcal{K}_\delta^\ell$ with mesh size $\delta = \delta_\ell = \dfrac{H_\ell}{\mathsf{n}_c} \geq h_\ell$. We do not assume that $\mathcal{K}_\delta^\ell$ and $\mathcal{K}_h^\ell$ are nested. We define the coarse auxiliary P1 discretization spaces $\mathcal{V}_\delta^\ell \subset H^1(\Omega_\ell) \cap C^0(\bar{\Omega}_\ell)$ by

$$\mathcal{V}_\delta^\ell = \left\{ v \in C^0(\bar{\Omega}_\ell) \text{ s.t. } v|_K \in P_1(K),\ K \in \mathcal{T}_\delta^\ell \right\} \cap H_0^1(\Omega_\ell).$$

For each $m = (\ell, i) \in I$ we also consider the corresponding auxiliary multiplier space $M_\delta^m \subset L^2(\gamma_m)$, defined analogously to (3.36).

The spaces X_δ, M_δ, $\mathcal{X}_\delta$, and T_δ^ℓ, T_δ, $\mathcal{T}_\delta$ are built starting from the $\mathcal{V}_\delta^\ell$'s and the M_δ^m's in the same way as the spaces X_h, M_h, $\mathcal{X}_h$ and T_h^ℓ, T_h, $\mathcal{T}_h$ by using definitions similar to (3.8), (3.9) (3.10) (3.37), (3.38).

Analogously to π_h we can define the operator $\pi_\delta : \prod_{\ell=1}^L L^2(\partial\Omega_\ell) \longrightarrow T_\delta$. Using Lemma 3.6.4 we obtain for all $\eta \in T$ and $\alpha = (\alpha_\ell)_{\ell=1,\dots,L} \in T$, with α_ℓ constant,

$$|(Id - \pi_\delta)\eta|_T^2 \lesssim (1 + \log(\mathsf{n}_c))^2 \, \|\eta - \alpha\|_T^2 + (1 + \log(\mathsf{n}_c)) \, \sigma(\alpha, \alpha), \tag{4.19}$$

and for $\eta \in \mathfrak{L}$

$$|(Id - \pi_\delta)\eta|_T^2 \lesssim (1 + \log(\mathsf{n}_c)) \, (\|\eta - \alpha\|_T^2 + \sigma(\alpha, \alpha)). \tag{4.20}$$

Moreover, Lemma 4.1.2 yields that for all $\eta \in T_\delta$

$$|\Lambda\eta|_T^2 \lesssim (1 + \log(\mathsf{n}_c)) \, |\eta|_T^2. \tag{4.21}$$

Analogously to R_h^ℓ we can define a local coarse lifting operator R_δ^ℓ. By standard arguments it verifies, for all $\eta \in T_\delta$,

$$\|R_\delta \eta\|_X \simeq \|\eta\|_T, \qquad |R_\delta \eta|_X \simeq |\eta|_T. \tag{4.22}$$

We define the vertex block of the preconditioner $b_1^V : \mathcal{T}_h^V \times \mathcal{T}_h^V \to \mathbb{R}$ as

$$b_1^V(\eta^V, \xi^V) := \sum_\ell \int_{\Omega_\ell} a(\mathbf{x}) \nabla(R_\delta^\ell (1 - \pi_\delta)\Lambda \eta^V) \cdot \nabla(R_\delta^\ell (1 - \pi_\delta)\Lambda \xi^V). \tag{4.23}$$

The second preconditioner we propose is then :

$$\begin{aligned} &s_1 : \mathcal{T}_h \times \mathcal{T}_h \longrightarrow \mathbb{R}, \\ &s_1(\eta, \xi) = b^E(\eta^E, \xi^E) + \left(1 + \log\left(\frac{Hp^2}{h}\right)\right) b_1^V(\eta^V, \xi^V). \end{aligned} \tag{4.24}$$

Remark that $(1 - \pi_\delta)\Lambda \mathcal{T}_h^V = \mathcal{T}_\delta^V$. In view of this identity it is not difficult to realize that computing the vertex block of this preconditioner only implies assembling the Schur complement matrix for an auxiliary mortar problem corresponding to the coarse dicretization. This is then independent of the mesh size h.

The following theorem holds :

Theorem 4.2.1. *For all $\eta \in \mathcal{T}_h$ we have :*

$$\hat{p}^{-3/2} s(\eta, \eta) \lesssim s_1(\eta, \eta) \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2 s(\eta, \eta). \tag{4.25}$$

See section E in Appendices for the proof of Theorem 4.2.1.

Let $\mathbf{S}$ and $\mathbf{P}_1$ be the matrices obtained by discretizing respectively s and s_1 then, by using the lower and upper bounds for the eigenvalues of $\mathbf{P}_1^{-1}\mathbf{S}$ given by Theorem 4.2.1, we obtain :

Corollary 4.2.2. *The condition number of the preconditioned matrix $\mathbf{P}_1^{-1}\mathbf{S}$ satisfies :*

$$Cond(\mathbf{P}_1^{-1}\mathbf{S}) \lesssim \hat{p}^{3/2} \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2. \tag{4.26}$$

4.2.2 A Discontinuous Galerkin Vertex Block Preconditioner

As a further alternative, we propose to construct the vertex block of the preconditioner, by completely giving up weak continuity and by replacing it with a Discontinuous Galerkin interior penalty method as coarse problem.

More precisely, letting $\mathcal{H}_\ell : H^{1/2}(\partial\Omega_\ell) \to H^1(\Omega_\ell)$ denotes the harmonic lifting, we set

$$b_\#^V(\eta_\ell^V, \zeta_\ell^V) = \sum_\ell a_\ell(\mathcal{H}_\ell \Lambda^\ell \eta^V, \mathcal{H}_\ell \Lambda^\ell \zeta^V), \tag{4.27}$$

$$b_{[]}^V(\eta^V, \eta^V) = \sum_{m\in I} |\gamma_m|^{-1} \int_{\gamma_m} |[\Lambda\eta]|^2. \tag{4.28}$$

Then, as vertex block of the preconditioner, we consider :

$$b_2^V(\eta, \eta) = \beta\, b_\#^V(\eta_\ell^V, \eta_\ell^V) + \gamma\, b_{[]}^V(\eta_\ell^V, \eta_\ell^V) \tag{4.29}$$

with $\beta, \gamma > 0$ constant.

The global preconditioner is then assembled as follow :

$$s_2(\eta, \eta) = b^E(\eta^E, \eta^E) + \left(1 + \log\left(\frac{Hp^2}{h}\right)\right) b_2^V(\eta^V, \eta^V). \tag{4.30}$$

We have the following theorem.

Theorem 4.2.3. *For all $\eta \in \mathcal{T}_h$ we have :*

$$\hat{p}^{-3/2} s(\eta, \eta) \lesssim s_2(\eta, \eta) \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2 s(\eta, \eta). \tag{4.31}$$

See section E in Appendices for the proof of Theorem 4.2.3.

Remark 4.2.4. We observe that if the Ω_ℓ's are rectangles, for $\eta \in \mathfrak{L}$ we have that $\mathcal{H}_\ell \eta_\ell$ is the Q_1 function (polynomial of degree ≤ 1 in each of the two unknowns) coinciding with η_ℓ at the four vertices of Ω_ℓ. The local matrix corresponding to the block b_1^V can then be replaced by the elementary Q1 stifness matrix for the problem considered.

Let $\mathbf{S}$ and $\mathbf{P}_2$ be the matrices obtained by discretizing respectively s and $\hat{s}$ then, by using the lower and upper bounds for the eigenvalues of $\mathbf{P}_2^{-1}\mathbf{S}$ given by Theorem 4.2.3, we obtain :

Corollary 4.2.5. *The condition number of the preconditioned matrix $\mathbf{P}_2^{-1}\mathbf{S}$ satisfies :*

$$Cond(\mathbf{P}_2^{-1}\mathbf{S}) \lesssim \hat{p}^{3/2} \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2. \tag{4.32}$$

4.3 Algebraic Forms

We start by deriving the matrix form of the discrete Steklov-Poincaré operator s defined in (4.3). Let us assume that the nodes are numbered as interior degrees of freedom first (grouped subdomain-wise), then degrees of freedom associated to nodes that lives on master edges, the degrees of freedom corresponding to the crosspoints of the subdomains and finally the degrees of freedom corresponding to slave edges.

We let $n_{\mathtt{I}}$, $n_{\mathtt{E}}$, $n_{\mathtt{V}}$ and $n_{\mathtt{S}}$ be the number of interior, master edge, crosspoints and slave edge degrees of freedom, respectively, and set n the number of degree of freedom, i.e. $n = n_{\mathtt{E}} + n_{\mathtt{V}}$.

With this notation, the vector of unknown $\mathbf{u}$, the matrix A and the vector $\mathbf{F}$ associated respectively to the discretization of a_X and of $\int_\Omega f\,dx$ can be written as :

$$A = \begin{pmatrix} \mathbf{A}_{\mathtt{II}} & \mathbf{A}_{\mathtt{IB}} \\ \mathbf{A}_{\mathtt{BI}} & \mathbf{A}_{\mathtt{BB}} \end{pmatrix}, \qquad \mathbf{u} = \begin{pmatrix} \mathbf{u}_{\mathtt{I}} \\ \mathbf{u}_{\mathtt{B}} \end{pmatrix}, \qquad \mathbf{F} = \begin{pmatrix} \mathbf{f}_{\mathtt{I}} \\ \mathbf{f}_{\mathtt{B}} \end{pmatrix} \tag{4.33}$$

where $\mathbf{u}_{\mathtt{I}}$ represents the unknown component associated to interior nodes and $\mathbf{u}_{\mathtt{B}}$ the unknown component associated to boundary nodes. The local Schur complement system is written

$$\Sigma \mathbf{u}_{\mathtt{B}} = \mathbf{g}_{\mathtt{B}}, \qquad \Sigma = -\mathbf{A}_{\mathtt{BI}}\mathbf{A}_{\mathtt{II}}^{-1}\mathbf{A}_{\mathtt{IB}} + \mathbf{A}_{\mathtt{BB}} \quad \text{and} \quad \mathbf{g}_{\mathtt{B}} = \mathbf{f}_{\mathtt{B}} - \mathbf{A}_{\mathtt{BI}}\mathbf{A}_{\mathtt{II}}^{-1}\mathbf{f}_{\mathtt{I}} \tag{4.34}$$

4.3.1 Constraint Matrix

From the mortar condition, it follows that the interior nodes of the slave edges are not associated with genuine degrees of freedom in the finite element space. Indeed, the value of those coefficients corresponding to basis functions "living" on slave edges is uniquely determined by the remaining coefficients through the jump condition, and can be eliminated from the global vector $\mathbf{u}_{\mathtt{B}} = (\mathbf{u}_{\mathtt{V}}, \mathbf{u}_{\mathtt{M}}, \mathbf{u}_{\mathtt{S}})^T$ of the local Schur complement system (4.34). The mortar condition is given by

$$C_{\mathtt{S}}\mathbf{u}_{\mathtt{S}} = C_{\mathtt{M}}\mathbf{u}_{\mathtt{M}} - C_{\mathtt{V}}\mathbf{u}_{\mathtt{V}}. \tag{4.35}$$

The constrained coefficients $\mathbf{u}_{\mathtt{S}}$ are uniquely determined through the condition (4.35), i.e.

$$\mathbf{u}_{\mathtt{S}} = C_{\mathtt{S}}^{-1}C_{\mathtt{M}}\mathbf{u}_{\mathtt{M}} - C_{\mathtt{S}}^{-1}C_{\mathtt{V}}\mathbf{u}_{\mathtt{V}} = Q_{\mathtt{M}}\mathbf{u}_{\mathtt{M}} + Q_{\mathtt{V}}\mathbf{u}_{\mathtt{V}} \tag{4.36}$$

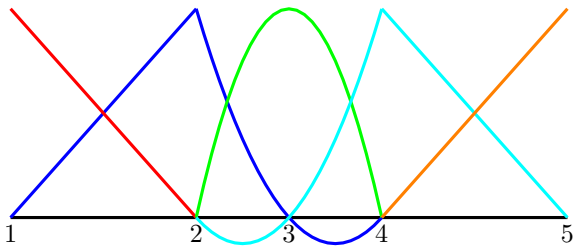
where $Q_{\mathtt{M}} = C_{\mathtt{S}}^{-1}C_{\mathtt{M}}$ and $Q_{\mathtt{V}} = -C_{\mathtt{S}}^{-1}C_{\mathtt{V}}$. The entries of the mass matrices $C_{\mathtt{S}}$, $C_{\mathtt{M}}$ and $C_{\mathtt{V}}$ are given by

$$c_{i,j} := \int_{\gamma_m} [\phi_j]\lambda_i \, ds$$

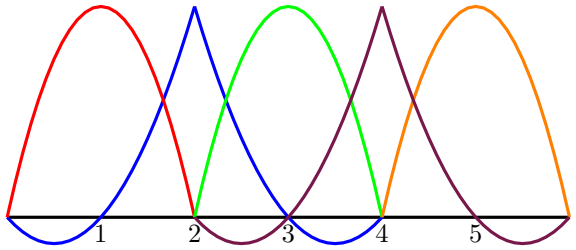
with $\lambda_i \in M_h$ and ϕ_j corresponding to the different nodal basis functions associated with the slave side, the master side and with the cross-points. More specifically, we have

$$\begin{aligned}
C_{\mathsf{S}}: \quad & c^{\mathsf{S}}_{i,j} = \int_{\gamma_m} [\phi_j]\lambda_i \, ds \qquad i,j = 1,2,\ldots,n_{\mathsf{S}} \\
C_{\mathsf{M}}: \quad & c^{\mathsf{M}}_{i,j} = \int_{\gamma_m} [\phi_j]\lambda_i \, ds \qquad i = 1,2,\ldots,n_{\mathsf{S}} \quad j = 1,2,\ldots,n_{\mathsf{M}} \\
C_{\mathsf{V}}: \quad & c^{\mathsf{V}}_{i,j} = \int_{\gamma_m} [\phi_j]\lambda_i \, ds \qquad i = 1,2,\ldots,n_{\mathsf{S}} \quad j = 1,2,\ldots,n_{\mathsf{V}}.
\end{aligned}$$

We note that C_{S} is a square matrix whereas C_{M} and C_{V} are rectangular matrices. The FIGURE 4.1 shows the profile of the test basis functions λ_i (see FIGURE 4.1.1.) and the trial basis functions ϕ_i in P2 finite element approximation (see FIGURE 4.1.2.).



4.1.1. Test basis functions in P_2



4.1.2. Trial basis functions in P_2

FIGURE 4.1 : Second-order basis functions for mortar finite element method

Remark 4.3.1. The test basis functions in FIGURE 4.1.1. contain the mortar modifications, i.e. the basis functions are P_k polynomials on all the interior elements, whereas they are P_{k-1} polynomials on the extremal elements. The trial basis functions in FIGURE 4.1.2. are the standard Lagrange polynomials.

From (4.36), the solution vector $\mathbf{u}_{\mathsf{B}}$ of (4.34) can be written

$$\mathbf{u}_{\mathsf{B}} = \begin{pmatrix} \mathbf{u}_{\mathsf{V}} \\ \mathbf{u}_{\mathsf{M}} \\ Q_{\mathsf{M}}\mathbf{u}_{\mathsf{M}} + Q_{\mathsf{V}}\mathbf{u}_{\mathsf{V}} \end{pmatrix} = \begin{pmatrix} \mathbf{I}_{\mathsf{V}} & 0 \\ 0 & \mathbf{I}_{\mathsf{M}} \\ Q_{\mathsf{V}} & Q_{\mathsf{M}} \end{pmatrix} \begin{pmatrix} \mathbf{u}_{\mathsf{V}} \\ \mathbf{u}_{\mathsf{M}} \end{pmatrix} = Q \begin{pmatrix} \mathbf{u}_{\mathsf{V}} \\ \mathbf{u}_{\mathsf{M}} \end{pmatrix} \tag{4.37}$$

From (4.37), the system (4.34) becomes

$$\mathbf{S}\mathbf{u} = \mathbf{g} \qquad \text{with} \quad \mathbf{S} = Q^T \Sigma Q, \quad \mathbf{u} = \begin{pmatrix} \mathbf{u}_{\mathsf{V}} \\ \mathbf{u}_{\mathsf{M}} \end{pmatrix}, \quad \text{and} \quad \mathbf{g} = Q^T \mathbf{g}_{\mathsf{B}} \tag{4.38}$$

The Schur complement $\mathbf{S}$ represents the matrix form of the Steklov-Poincaré operator $s(\,\cdot\,,\,\cdot\,)$ defined in (4.3).

In order to implement the preconditioner introduced in this chapter, we need to represent algebraically the splitting of the trace space given by (4.10). As defined in (4.7), we consider the space $\mathfrak{L}$ of functions that are linear on each subdomain edge, and introduce the matrix representation of the injection of $\mathfrak{L}$ into T.

Let $\Xi = \left\{\mathbf{x}_i,\ i = 1, \dots n_{\mathsf{V}}, n_{\mathsf{V}} + 1 \dots n_{\mathsf{V}} + n_{\mathsf{E}}\right\}$ be the set of edge and vertex nodes. For any vertex node $\mathbf{x}_j\,,\ j = 1\,, \dots n_{\mathsf{V}}$, let $\phi_j(\,\cdot\,)$ be the piecewise polynomial that is linear on each subdomain edge and that satisfies $\phi_j(\mathbf{x}_k) = \delta_{j,k},\ j, k = 1, \dots n_{\mathsf{V}}$. Let $\mathbf{R}_{\mathsf{V}} \in \mathbb{R}^{n \times n_{\mathsf{V}}}$ be the matrix realizing the linear interpolation of vertex values and let $\mathbf{R} \in \mathbb{R}^{n \times n}$ be the matrix defined as

$$\mathbf{R} = \begin{pmatrix} \begin{pmatrix} 0 \\ \mathbf{I}_e \end{pmatrix} & \mathbf{R}_{\mathsf{V}} \end{pmatrix} \tag{4.39}$$

Let now $\widehat{\mathbf{S}}$ be the matrix obtained after applying the change of basis corresponding to switching from the standard nodal basis to the basis related to the splitting (4.10), that is

$$\widehat{\mathbf{S}} = \mathbf{R}^T \mathbf{S} \mathbf{R} = \begin{pmatrix} \widehat{\mathbf{S}}_{\mathsf{VV}} & \widehat{\mathbf{S}}_{\mathsf{VE}} \\ \widehat{\mathbf{S}}_{\mathsf{EV}} & \widehat{\mathbf{S}}_{\mathsf{EE}} \end{pmatrix} \tag{4.40}$$

From now on, we focus on finding efficient preconditioners for the transformed Schur complement system

$$\widehat{\mathbf{S}}\widehat{\mathbf{u}} = \widehat{\mathbf{g}}, \qquad \widehat{\mathbf{u}} = \mathbf{R}^{-1}\mathbf{u} \quad \text{and} \quad \widehat{\mathbf{g}} = \mathbf{R}^T \mathbf{g} \tag{4.41}$$

The preconditioner for $\widehat{\mathbf{S}}$ will be of block-Jacobi type : one block for the master edges and another one for the vertices.

For the edge block of the preconditioner, we deal with the matrix counterpart of (4.13). In the literature, it is possible to find different ways to build bilinear forms $b^E(\,\cdot\,,\,\cdot\,)$ that satisfy (4.13)-(4.12). The choice followed here for defining $b^E(\,\cdot\,,\,\cdot\,)$ is the one proposed in [BPS86] and it is based on an equivalence result for the $H^{1/2}_{00}$ norm, see [BW86] and [Ant+14] for a detailed description of its construction.

For $\eta^{\mathsf{E}} \in T^0_{\ell,i}$ we denote by $\boldsymbol{\eta}^{\mathsf{E}}$ its vector representation. Then, it can be verified that, for each $\gamma^{(i)}_\ell \subset \partial\Omega_\ell$, we have (see [BW86] pag. 1110 and [Dry82])

$$(l_0^{1/2}\eta^{\mathsf{E}}, \eta^{\mathsf{E}})_{\gamma_\ell^{(i)}} = \boldsymbol{\eta}^{\mathsf{E}T}\widehat{\mathbf{K}}_{\mathsf{E}}\boldsymbol{\eta}^{\mathsf{E}}$$

where $\widehat{\mathbf{K}}_{\mathsf{E}} = \mathbf{M}_{\mathsf{E}}^{1/2}(\mathbf{M}_{\mathsf{E}}^{-1/2}\mathbf{R}_{\mathsf{E}}\mathbf{M}_{\mathsf{E}}^{-1/2})^{1/2}\mathbf{M}_{\mathsf{E}}^{1/2}$ and $\mathbf{M}_{\mathsf{E}}$ and $\mathbf{R}_{\mathsf{E}}$ are the mass and stiffness matrices associated to the discretization of the operator $-\mathrm{d}^2/\mathrm{ds}^2$ (in $T^0_{\ell,i}$) with homogeneous Dirichlet boundary conditions at the extrema a and b of $\gamma_\ell^{(i)}$. Thus, the edge block of the preconditioner can be written as :

$$\mathbf{P}_{\mathsf{E}} = \begin{pmatrix} \widehat{\mathbf{K}}_{\mathsf{E}_1} & 0 & 0 & 0 \\ 0 & \widehat{\mathbf{K}}_{\mathsf{E}_2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \widehat{\mathbf{K}}_{\mathsf{E}_\mathtt{M}} \end{pmatrix} \tag{4.42}$$

with one block for each mortar edge where $\mathtt{M}$ is the number of mortar edges.

Remark 4.3.2. $\widehat{\mathbf{K}}_{\mathsf{E}}$ can be approximated as the square root of $h\mathbf{R}_{\mathsf{E}}$. The computation of the square root of a matrix can be quite expensive. Therefore, we use the Lanczos Algorithm for SVD (Singular Value Decomposition) to compute the matrix square root $\mathbf{R}_{\mathsf{E}}^{1/2}$.

The preconditioner $\mathbf{P}$ that we propose is obtained as the matrix counterpart of (4.30) and of (4.24) defined by

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_{\mathsf{V}} & \\ & \mathbf{P}_{\mathsf{E}} \end{pmatrix} \tag{4.43}$$

where $\mathbf{P}_{\mathsf{V}}$ and $\mathbf{P}_{\mathsf{E}}$ are the vertex and egde blocks of the preconditioner respectively.

4.3.2 Preconditioner $\mathbf{P}_1$

Concerning the vertex block of our preconditioner, following subsection 4.2.1, we introduce a coarse auxiliary mesh in each subdomain made up of 3×3 elements and we fix the polynomial order $p = 1$.

Let now consider the associated Schur complement system and let $\widehat{\mathbf{S}}^c$ be the matrix obtained after applying the change of basis, that is

$$\widehat{\mathbf{S}}^c = \begin{pmatrix} \widehat{\mathbf{S}}^c_{\mathsf{VV}} & \widehat{\mathbf{S}}^c_{\mathsf{VE}} \\ \widehat{\mathbf{S}}^c_{\mathsf{EV}} & \widehat{\mathbf{S}}^c_{\mathsf{EE}} \end{pmatrix} \tag{4.44}$$

The preconditioner $\mathbf{P}_1$, described in section 4.2.1, can then be written as :

$$\mathbf{P}_1 = \begin{pmatrix} \mathbf{P}_{\mathsf{V}}^c & 0 \\ 0 & \mathbf{P}_{\mathsf{E}} \end{pmatrix} \tag{4.45}$$

where $\mathbf{P}_{\mathsf{V}}^c = \left(1 + \log\left(\frac{Hp^2}{h}\right)\right) \widehat{\mathbf{S}}_{\mathsf{VV}}^c$.

4.3.3 Preconditioner $\mathbf{P}_2$

Let $\mathbf{P}_{\#}$ and $\mathbf{P}_{[]}$ be the matrix counterparts respectively of (4.27) and of (4.28) in section 4.2.2 and let $\mathbf{P}_{\mathsf{V}}^{\mathsf{DG}} = \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)(\beta\mathbf{P}_{\#} + \gamma\mathbf{P}_{[]})$. Then the new preconditioner we propose writes :

$$\mathbf{P}_2 = \begin{pmatrix} \mathbf{P}_{\mathsf{V}}^{\mathsf{DG}} & 0 \\ 0 & \mathbf{P}_{\mathsf{E}} \end{pmatrix}. \tag{4.46}$$

4.4 Conclusion

In this chapter, we analyzed the substructuring preconditioners for h-p mortar finite element method described in chapter 3. We introduced the general concept of the substructuring approach of domain decomposition methods. We focused on the preconditioning of the discrete Steklov-Poincaré operator defined on the skeleton. We proposed two vertex block preconditioners, responsible for the good scaling properties of the preconditioners considered. The main contribution of this chapter was the construction of a coarse preconditioner based on the Discontinuous Galerkin type interior penalty method defined on the coarse mesh. We proved that the condition number of the preconditioned Schur complement system behaves in a polylogarithmic way. A parallel implementation framework for the preconditioners will be developed in chapter 6. The numerical results to be presented in the chapter 8 will conform the optimality and the efficiency of the preconditioners for very large number of subdomains. The strong and weak scalability analyzed in the same chapter will indicate the performance of our parallel algorithms on large scale computer architectures.

Chapter 5

Mortar Element Method with Lagrange Multipliers

We discuss the domain decomposition method based on an approximation by the mortar finite element method with Lagrange multipliers proposed in [Bel99]. The hybrid formulation is reminded in section 5.1. A computational framework already proposed in [Sam+12a] is emphasized in section 5.2. The convergence analysis is presented in section 5.3.

Nous discutons la méthode de décomposition de domaine basée sur une approximation par la méthode des éléments finis mortar avec multiplicateurs de Lagrange proposée dans [Bel99]. La formulation hybride est rappelée dans la section 5.1. Un framework de calcul déjà proposé dans [Sam+12a] est souligné dans la section 5.2. L'analyse de convergence est présentée dans la section 5.3.

Contents

5.1 Hybrid Formulation for Mortar Element Method

Let Ω be a bounded domain of $\mathbb{R}^d\,, d = 2\,, 3$. We consider the following Dirichlet boundary value problem : find u satisfying

$$-\Delta u = f \ \text{ in } \ \Omega, \qquad u = g \ \text{ on } \ \partial\Omega, \tag{5.1}$$

where $f \in L^2(\Omega)$ and $g \in H^{1/2}(\partial\Omega)$ are given functions. We consider a decomposition of Ω as the union of L subdomains Ω_ℓ,

$$\Omega = \bigcup_{\ell=1}^{L} \Omega_\ell. \tag{5.2}$$

We assume that the domain decomposition (5.2) is geometrically conforming which means that if $\Gamma_{\ell n} := \overline{\Omega}_\ell \cap \overline{\Omega}_n \ (\ell \neq n) \neq \emptyset$, then $\Gamma_{\ell n}$ must either be a common vertex of Ω_ℓ and Ω_n, or a common edge, or a common face if $d = 3$. Note that $\Gamma_{\ell n} = \Gamma_{n\ell}$. The usual variational formulation of (5.1) reads

Problem 5.1.1. *Find $u \in H^1_g(\Omega)$ such that*

$$\int_\Omega \nabla u \cdot \nabla v \, dx = \int_\Omega f v \, dx \qquad \forall v \in H^1_0(\Omega). \tag{5.3}$$

We define two product spaces :

$$V = \prod_{\ell=1}^{L} H^1(\Omega_\ell), \quad \Lambda = \prod_{\ell=1}^{L} \prod_{\substack{0\le n<\ell \\ |\Gamma_{\ell n}|\neq 0}} \Big(H^{1/2}(\Gamma_{\ell n})\Big)'. \tag{5.4}$$

The space Λ will be a trial space for the weak continuity conditions on the interfaces. We introduce the bilinear forms $a : V \times V \longrightarrow \mathbb{R}$, $b : V \times \Lambda \longrightarrow \mathbb{R}$ and the linear functional $f : V \longrightarrow \mathbb{R}$:

$$a(u,v) = \sum_{\ell=1}^{L} a_\ell(u,v), \quad a_\ell(u,v) = \int_{\Omega_\ell} \nabla u_\ell \cdot \nabla v_\ell \, dx, \tag{5.5}$$

$$b(\lambda, v) = \sum_{\ell=1}^{L} \sum_{\substack{n=0 \\ |\Gamma_{n\ell}|\neq 0}}^{L} b_{\ell n}(\lambda, v), \quad b_{\ell n}(\lambda, v) = \langle \lambda_{\ell n}, v_\ell \rangle|_{\Gamma_{\ell n}}, \tag{5.6}$$

$$f(v) = \sum_{\ell=1}^{L} \int_{\Omega_\ell} f v_\ell \, dx, \tag{5.7}$$

where $\lambda_{\ell n} = -\lambda_{n\ell}\ \langle\cdot\,,\cdot\rangle|_{\Gamma_{kl}}$ stands for the duality product between $\left(H^{1/2}(\Gamma_{\ell n})\right)'$ and $H^{1/2}(\Gamma_{\ell n})$. The mortar formulation with Lagrange multipliers reads as

Problem 5.1.2. *Find* $(u\,,\lambda) \in V \times \Lambda$ *such that* $\forall (v\,,\mu) \in V \times \Lambda$

$$a(u,v) + b(\lambda,v) = f(v), \tag{5.8}$$
$$b(\mu,u) = 0. \tag{5.9}$$

If $\underline{\mathbf{u}}$ and $\underline{\lambda}$ denote the vectors of the components of u and λ the discrete system associated to the problem 5.1.2 is equivalent to the following saddle-point system :

$$\mathcal{A}\begin{pmatrix}\underline{\mathbf{u}}\\ \underline{\lambda}\end{pmatrix} = \begin{pmatrix} f \\ 0\end{pmatrix} \tag{5.10}$$

with

$$\mathcal{A} = \begin{pmatrix} A & B^T \\ B & 0\end{pmatrix}, \qquad A = \begin{pmatrix} A_1 & & 0 \\ & \ddots & \\ 0 & & A_L\end{pmatrix}, \qquad B^T = \begin{pmatrix} B_1^T \\ \vdots \\ B_L^T\end{pmatrix}$$

$A_\ell\,,\ \ell = 1\,,\dots L\,$, corresponds to the stiffness matrix in the subdomain Ω_ℓ and B_ℓ denotes the matrix associated to the discrete form of the mortar weak continuity constraint in the subdomain Ω_ℓ.

5.2 Computational Framework

We want to solve the saddle-point algebraic linear system (5.10) using an iterative Krylov subspace method in parallel. Finding a good preconditioner for such problems is a delicate issue as the matrix $\mathcal{A}$ is indefinite and any preconditioning matrix $\mathcal{P}$ acting on the jump matrices $B_{\ell n}$ would involve communications.

A survey on block diagonal and block triangular preconditioners for this type of saddle-point system (5.10) can be found in [BGL05 ; ESW05]. The matrix $\mathcal{A}$ arising in saddle-point problems is known to be spectrally equivalent to the block diagonal matrix :

$$\mathcal{P} = \begin{pmatrix} A & 0 \\ 0 & -S\end{pmatrix} \tag{5.11}$$

where S is the Schur complement $-BA^{-1}B^T$, see [MGW00]. While not being an approximate inverse of $\mathcal{A}$, the matrix $\mathcal{P}$ is an ideal preconditioner. Indeed it can be shown that

$P(X) = X(X-1)(X^2-X-1)$ is an annihilating polynomial of the matrix $\mathcal{T} = \mathcal{P}^{-1}\mathcal{A}$. Therefore, assuming $\mathcal{T}$ non singular, the matrix $\mathcal{T}$ has only three eigenvalues $\{1\,,(1\pm\sqrt{5})/2\}$. Thus an iterative solver using the Krylov subspaces constructed with $\mathcal{T}$ would converge within three iterations. In practice, computing the inverse of the exact preconditioner $\mathcal{P}$ is too expensive. Instead, one would rather look for an *inexact* inverse $\widehat{\mathcal{P}}^{-1}$. When applying the preconditioner, the inexact inverse $\widehat{\mathcal{P}}^{-1}$ would be determined following an iterative procedure for solving the linear system $\mathcal{P}\mathbf{x} = \mathbf{y}$. It requires a class of iterative methods qualified as flexible inner-outer preconditioned solvers [Saa93] or inexact inner-outer preconditioned solvers [GY99].

The outer iterations for solving the main problem involve inner iterations for computing an inexact and non-constant preconditioner. Finding the relevant convergence parameters to this inner iterative procedure is a critical issue. On one hand, $\widehat{\mathcal{P}}^{-1}$ has to be computed in few iterations : the total number of iterations including the inner iterations should be less than without preconditioner. On the other hand for ensuring the stability of the outer iterations, it would be preferable to solve the inner iterations with as much accuracy as possible in order to keep an almost constant preconditioner. We refer the reader to [CMZ12] and references therein for theoretical results and experimental assessment with respect to the influence of the perturbation to the preconditioner. In this context, the choice a good preconditioner for solving the inner iterations can have a significant impact on the convergence of the outer iterations.

The outer iterations will be carried out with a Flexible Preconditioned Biconjugate Gradient Stabilized Method (FBICGSTAB) [Saa03] and the Flexible Preconditioned Generalized Minimal Residual Method with restart `m` (FGMRES(`m`)) [Saa93].

Algorithm 5.1. FBICGSTAB : Solve $\mathcal{A}\mathbf{x} = \mathbf{b}$

$\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0$ `# initialize the residual` $\mathbf{r}_0$
$\tilde{\mathbf{r}}_0 = \mathbf{r}_0$
$\mathbf{p}_0 = \mathbf{r}_0$
$\mathbf{v}_0 = \mathbf{r}_0$
$\rho_0 = \alpha = \omega_0 = 1$
for $j = 0\,,1\,,\ldots,$ maxiter **do**
  $\rho_{j+1} = (\tilde{\mathbf{r}}_0\,,\mathbf{r}_j)$
  $\beta = (\rho_{j+1}/\rho_j) \times (\alpha/\omega_j)$
  $\mathbf{p}_{j+1} = \mathbf{r}_j + \beta(\mathbf{p}_j - \omega_j\mathbf{v}_j)$
  solve $\mathcal{P}\hat{\mathbf{p}} = \mathbf{p}_{j+1}$ `# apply the preconditioner` $\mathcal{P}$
  $\mathbf{v}_{j+1} = \mathcal{A}\hat{\mathbf{p}}$
  $\alpha = \rho_{j+1}/(\hat{\mathbf{r}}_0\,,\mathbf{v}_{j+1})$
  $\mathbf{s} = \mathbf{r}_j - \alpha\mathbf{v}_{j+1}$
  solve $\mathcal{P}\hat{\mathbf{s}} = \mathbf{s}$
  $\mathbf{t} = \mathcal{A}\hat{\mathbf{s}}$
  $\omega_{j+1} = (\mathbf{t}\,,\mathbf{s})/(\mathbf{t}\,,\mathbf{t})$
  $\mathbf{x}_{j+1} = \mathbf{x}_j + \alpha\hat{\mathbf{p}} + \omega\hat{\mathbf{s}}$
  $\mathbf{r}_{j+1} = \mathbf{s} - \omega_{j+1}\mathbf{t}$
end for

Algorithm 5.2. FGMRES(m) : Solve $\mathcal{A}\mathbf{x} = \mathbf{b}$

for $k = 1, 2, \ldots,$ maxiter **do**
  $\mathbf{r}_0 = \mathbf{b} - \mathcal{A}\mathbf{x}_0$ `# initialize the residual` $\mathbf{r}_0$
  $\beta = \|\mathbf{r}_0\|_2$
  $\mathbf{v}_1 = \mathbf{r}_0/\beta$
  $\mathbf{p} = \beta \mathbf{e}_1$
  for $j = 0, 1, \ldots,$ m **do**
    solve $\mathcal{P}\mathbf{z}_j = \mathbf{v}_j$ `# apply the preconditioner` $\mathcal{P}$
    $\mathbf{w} = \mathcal{A}\mathbf{z}_j$
    for $i = 1, 2, \ldots, j$ **do**
      $h_{i,j} = (\mathbf{w}, \mathbf{v}_i)$
      $\mathbf{w} = \mathbf{w} - h_{i,j}\mathbf{v}_i$
    end for
    $h_{j+1,j} = \|\mathbf{w}\|_2$
    $\mathbf{v}_{j+1} = \mathbf{w}/h_{j+1,j}$
    for $i = 1, 2, \ldots, j-1$ **do**
      $h_{i,j} = c_i h_{i,j} + s_i h_{i+1,j}$
      $h_{i+1,j} = -s_i h_{i,j} + c_i h_{i+1,j}$
    end for
    $\gamma = \sqrt{h_{j,j}^2 + h_{j+1,j}^2}$
    $c_j = h_{j,j}/\gamma; \quad s_j = h_{j+1,j}/\gamma$
    $h_{j,j} = \gamma; \quad h_{j+1,j} = 0$
    $p_j = c_j p_j; \quad p_{j+1} = -s_j p_j$
    if $|p_{j+1}| \leq \varepsilon$ **then**
      exit loop
    end if
  end for
  $\mathcal{Z}^m \longleftarrow [\mathbf{z}_1 \cdots \mathbf{z}_m]$
  $\mathcal{H}^m \longleftarrow (h_{i,j})_{1 \leq i \leq j+1; 1 \leq j \leq m}$
  $\mathbf{y} = \text{Argmin}_{\mathbf{q}} \|\mathbf{p} - \mathcal{H}^m \mathbf{q}\|_2$
  $\mathbf{x} = \mathbf{x}_0 + \mathcal{Z}^m \mathbf{y}$
  if $|p_{j+1}| \leq \varepsilon$ **then**
    exit loop
  else
    $\mathbf{x}_0 = \mathbf{x}$
  end if
end for

Regarding the preconditioning we will focus on two approximations of $\mathcal{P}$:

$$\mathcal{P}_I = \begin{pmatrix} A & 0 \\ 0 & I \end{pmatrix} \qquad \text{and} \qquad \mathcal{P}_S = \begin{pmatrix} A & 0 \\ 0 & -\widehat{S} \end{pmatrix} \tag{5.12}$$

In the first preconditioner the exact inverse of $\mathcal{P}_I$ is computed at each iteration using the $(I)LU$ factorization of the block diagonal matrix A. This preconditioner only acts on the diagonal blocks A_ℓ. As a result, solving the linear system $\mathcal{P}_I \mathbf{x} = \mathbf{y}$ does not involve any communication between the subdomains. However, since $\mathcal{P}_I$ does not act on the jump matrices $B_{\ell n}$, it is very likely to become less effective as the number of subdomains increases. In the second preconditioner the exact inverse of the block diagonal matrix A is also computed so that the exact Schur complement $S = -BA^{-1}B^T$ is readily available. Instead of taking S we choose an approximation $\widehat{S}$ such that $\mathbf{x} = \widehat{S}^{-1}\mathbf{y}$ is an approximate solution to the linear system $S\mathbf{x} = \mathbf{y}$ following an iterative procedure. This inner procedure is also carried out with a BICGSTAB algorithm preconditioned with the diagonal of S (Jacobi preconditioner $\mathcal{M}_J$) or with $\mathcal{M}_S^{-1} = BAB^T$.

Remark 5.2.1. The Krylov methods FBICGSTAB and FGMRES(m) presented respectively in Algorithm 5.1. and Algorithm 5.2. are both adapted to our saddle-point problem, but the only major difference between these methods is that FBICGSTAB presents sometimes breakdowns unlike FGMRES(m).

5.3 Convergence Analysis

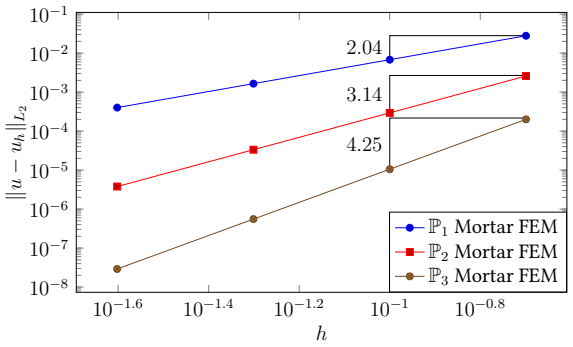
We summarize in TABLE 1 and TABLE 2 the behavior of L_2 and H^1 errors of the numerical solution relative to the analytical solution $g = \sin(\pi x)\cos(\pi y)\cos(\pi z)$ in 3D, see (5.1). The tests are performed in the case of nonconforming decompositions where the characteristic mesh size in subdomain Ω_ℓ is $h_{\Omega_\ell} = h + \delta_\ell\,,\ \ell = 1, \ldots, L$, with $\delta_\ell = 0\,.001$ the small perturbation. All the tests are achieved with 2, 4, 8 and 16 number of subdomains. We denote by u the exact solution of our problem and u_h^N the discrete solution obtained by using the characteristic mesh size equal to h and the piecewise polynomials of degree less than or equal to N. We denote $\|\cdot\|_0$ the L^2-norm and $\|\cdot\|_1$ the H^1-norm.

TABLE 1 : L^2-convergence analysis for hybrid mortar finite element formulation

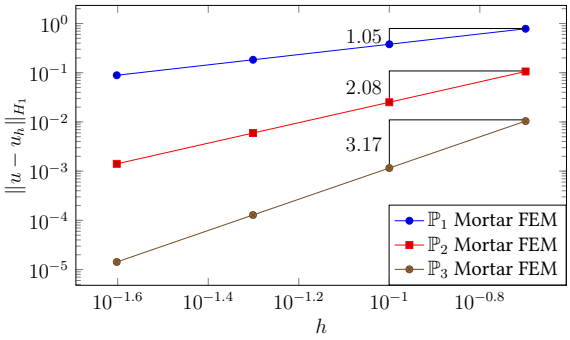
h	$\|u - u_h^1\|_0$	$\|u - u_h^2\|_0$	$\|u - u_h^3\|_0$
$2 \cdot 10^{-1}$	$2.80 \cdot 10^{-2}$	$2.67 \cdot 10^{-3}$	$2.16 \cdot 10^{-4}$
$1 \cdot 10^{-1}$	$6.69 \cdot 10^{-3}$	$2.83 \cdot 10^{-4}$	$9.58 \cdot 10^{-6}$
$5 \cdot 10^{-2}$	$1.66 \cdot 10^{-3}$	$3.24 \cdot 10^{-5}$	$5.29 \cdot 10^{-7}$
$2.5 \cdot 10^{-2}$	$4.00 \cdot 10^{-4}$	$3.91 \cdot 10^{-6}$	$3.10 \cdot 10^{-8}$

Table 2 : H^1-convergence analysis for hybrid mortar finite element formulation

h	$\|u - u_h^1\|_1$	$\|u - u_h^2\|_1$	$\|u - u_h^3\|_1$
$2 \cdot 10^{-1}$	$7.92 \cdot 10^{-1}$	$1.09 \cdot 10^{-1}$	$1.10 \cdot 10^{-2}$
$1 \cdot 10^{-1}$	$3.72 \cdot 10^{-1}$	$2.44 \cdot 10^{-2}$	$1.08 \cdot 10^{-3}$
$5 \cdot 10^{-2}$	$1.83 \cdot 10^{-1}$	$5.88 \cdot 10^{-3}$	$1.25 \cdot 10^{-4}$
$2.5 \cdot 10^{-2}$	$8.93 \cdot 10^{-2}$	$1.43 \cdot 10^{-3}$	$1.49 \cdot 10^{-5}$



5.1.1. L^2 convergence analysis



5.1.2. H^1 convergence analysis

Figure 5.1 : Convergence analysis for polynomial orders $p = 1, 2, 3$

The convergence results reported in Table 1, Table 2 and the plots in Figure 5.1 show that the mortar formulation presented in this chapter satisfies the convergence properties certified by the finite element theoretical estimations.

5.4 Conclusion

We discussed in this chapter the mortar finite element method with Lagrange multipliers. We recalled the hybrid formulation leading to a saddle-point type linear system, symmetric and infinite. We handled a computational framework, already proposed in [Sam+12a] for efficient solving of such a linear system. The parallel implementation and the numerical experiments including strong and weak scalability analysis for the mortar formulation studied in this chapter will be presented respectively in section 7.3 and in chapter 9.3.

Part II

Numerical Implementation

Chapter 6

Substructuring Preconditioners in 2D

This chapter deals with the implementation of the substructuring preconditioners for h-p mortar element finite method described in chapter 4. The external libraries needed for the implementation are presented in section 6.1. The interpolation operator framework for domain decomposition methods is briefly introduced in section 6.2. The parallel implementation and the code design are presented respectively in section 6.3 and in section 6.6.

Ce chapitre traite la mise en oeuvre des préconditionneurs par sous-structuration pour la méthode des éléments finis h-p mortar décrits dans le chapitre 4. Les librairies externes nécessaires pour la mise en oeuvre sont présentées dans la section 6.1. Un framework d'opérateurs d'interpolation pour les méthodes de décomposition de domaine est brièvement introduit dans la section 6.2. La mise en oeuvre en parallèle et la conception du code de calcul sont présentées respectivement dans la section 6.3 et dans la section 6.6.

Contents

6.1 Essential Ingredients

The implementation of numerical methods described in this thesis has been performed using Feel++ library, introduced in section 1.6, and the Message Passing Interface (MPI) library [Sni+96]. For the parallel computing, the interprocess communications are handled implicitly and/or explicitly using Boost.MPI and Boost.Serialization [Kor11].

We use PETSc library [Bal+04] for sparse linear algebra, such as matrices, vectors, numerical solvers, and related algorithms. The Eigen library [G+10] involves in dense linear algebra, such as the computation of matrix square root, the condition number estimates from conjugate gradient coefficients.

Feel++ uses Gmsh [GR09] to generate meshes in one, two and three dimensional spaces.

6.1.1 MPI

MPI is a standardized message-passing system for distributed-memory in parallel computing. The MPI standards provide portable, efficient, and flexible library routines for writing message-passing programs in the Fortran, C, and C++ programming languages.

We deal with two different approaches for MPI communications for domain decomposition framework in Feel++, namely explicit communications and seamless communications. In the first approach, the interprocess communications are handled explicitly and independently of Feel++, whereas in the second one, they are managed directly by Feel++. In this second approach, the parallelism is fully transparent.

6.1.2 PETSc

The Portable, Extensible Toolkit for Scientific Computation is a large suite of data structures and routines providing parallel, efficient application programs for the solution of problems in scientific computation, especially the solution of Partial Differential Equations (PDEs). All of its features are directly usable with the programming language C. The PETSc package is designed around two main concepts, namely data encapsulation and software layering.

In the explicit communication approach introduced in section 6.1.1, we use PETSc sequentially even though the code is parallel. It requires explicitly sending and receiving complex data structures such as trace mesh data structures, elements of function space (traces), PETSc vectors.

6.1.3 Gmsh

Gmsh [GR09] is a three-dimensional finite element mesh generator with a build-in Computer-aided design (CAD) engine and post-processor. It aims to provide a fast, light and user-friendly meshing tool with parametric input and advanced visualization capabilities. There exists four

main modules in GMSH : geometry, mesh, solver and post-processing. All instructions are prescribed either interactively using the graphical user interface (GUI) or in text files.

6.2 Linear Interpolation Operator

In the context of domain decomposition methods, the interpolation operator is a crucial tool. It allows the transfer of information between adjacent meshes (with overlapping or not, conforming or not) at subdomain interfaces.

We are interested in the algebraic representation of the linear interpolation operator, more precisely the associated matrix. Let X_h and Y_h be two function spaces defined respectively on meshes $\mathcal{T}_h^1$ and $\mathcal{T}_h^2$. The interpolation operator $\mathcal{I}_{X_h \to Y_h}$ from X_h to Y_h is defined as

$$\begin{aligned} \mathcal{I}_{X_h \to Y_h} \ : \ X_h &\longrightarrow Y_h \\ u &\longmapsto v = \mathcal{I}_{X_h \to Y_h}(u). \end{aligned}$$

$\mathcal{I}_{X_h \to Y_h}(u) \in Y_h$ is called the interpolant of $u \in X_h$. Let A be the algebraic representation matrix of $\mathcal{I}_{X_h \to Y_h}$. The application of $\mathcal{I}_{X_h \to Y_h}$ to an element $u \in X_h$ is equivalent to the simple matrix-vector multiplication Au. In addition, A is a sparse matrix and its coefficients are independent of u. This feature is particularly interesting since it enables to apply the matrix A as many times as necessary without having to rebuild it.

The interpolation operator is based on two fundamental tools namely localization tool using a kd-tree data structure for fast localization and the inverse geometrical transformation, see [Cha13 ; Pru+12a].

In FEEL++, the operator interpolation is defined as

Listing 6.1 : Definition of linear interpolation operator in FEEL++

```
// define two function spaces Xh and Yh
auto Xh = Pch<2>(mesh1);
auto Yh = Pch<3>(mesh2);
// define linear interpolation operator from Xh to Yh
auto opI = opInterpolation(_domainSpace=Xh,_imageSpace=Yh);
```

In Listing 6.1, the matrix A is computed and stored in opI. The interpolant $v \in Y_h$ of $u \in X_h$ is computed as follows

Listing 6.2 : Application of linear interpolation operator

```
// project the function cos(πx)sin(πy) on Xh
auto u = vf::project(_space=Xh,_expr=cos(pi*Px())*sin(pi*Py()));
// define an element v of Yh
auto v = Yh->element();
// compute the interpolant v of u
opI->apply(u,v);
```

Remark 6.2.1. Other options of linear interpolation operator are available, for example the possibility to interpolate on a subspace, the boundary or a portion of the boundary of the image space.

In the remainder of this part dedicated to the implementation aspects of various numerical methods presented in previous chapters, the linear interpolation operator will play an essential role.

6.3 Parallel Implementation

The parallel implementation is performed using Feel++ library and the Message Passing Interface (MPI) library. We use the explicit communication approach introduced above for the interprocess communications. The explicit sending and receiving of complex data structures such as mesh data structures and elements of function space (traces) are handled by using Boost.MPI and Boost.Serialization.

Boost.MPI

Boost.MPI is a C++ layer on top of MPI allowing for simpler mpi usage in particular most standard C++ containers can be sent or received without having to do anything.

Listing 6.3 : Send and receive data structures

```
mpi::communicator world;
std::vector<double> x(100);
// do something with x
// send x from 0 to 1 with tag == 1
if ( world.rank() == 0 ) world.send( 1, 1, x );
if ( world.rank() == 1 ) world.recv( 0, 1, x );
```

Boost.Serialization

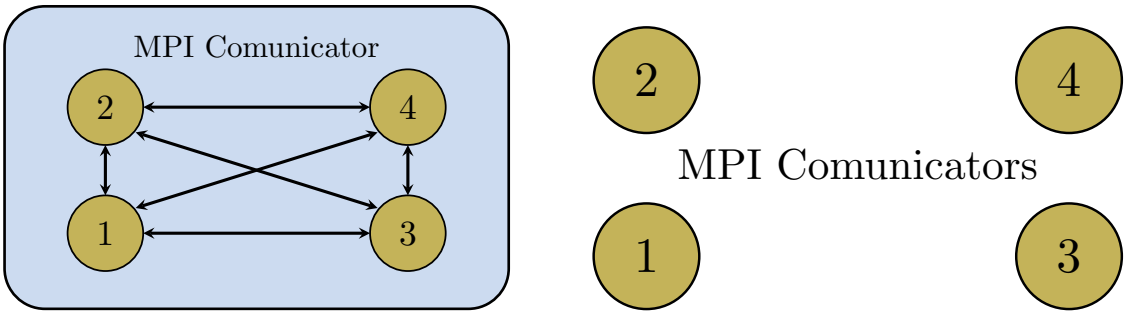
Boost.Serialization provides a simple interface to serialize data structures, used typically to archive data structures (persistence) but also by Boost.MPI to send and receive complex data structures.

Listing 6.4 : Example using Boost.Serialization

```
class A {  public:
std::vector<double> x,y,z;
template<class Archive>
void save( Archive & ar, const unsigned int /*version*/ ) const {
ar  & x; ar & y; ar & z;   }
template<class Archive>
void load( Archive & ar, const unsigned int /*version*/ ) {
ar  & x; ar & y; ar & z;  }
BOOST_SERIALIZATION_SPLIT_MEMBER()  };
```

MPI Communicators

We define three different types of MPI communicators : (i) a global communicator for interprocess communications between subdomains (e.g. FIGURE 6.1.1.) (ii) the local communicator (sub-communicator) that activates only the subdomain within (e.g. FIGURE 6.1.2.) (iii) the subgroup communicator for interprocess communications between selected subgroups responsible for the parallel solution of the coarse preconditioner defined in (4.29), see FIGURE 6.4.



6.1.1. Global MPI communicator

6.1.2. Local MPI communicators

FIGURE 6.1 : MPI Communicators

One interesting aspect of the parallel implementation of the h-p mortar finite element method is the absence of communication at cross-points (in 2D and 3D) and cross-edges (in 3D), which reduces significantly the interprocess communications between subdomains, and therefore the computational cost.

6.3.1 Geometric Framework

6.3.1.1 Polygonal Domain

We describe in this section the domain decomposition and the mesh generation for the mortar finite element discretization. First, we consider a geometrical discretization $\Omega_\ell^{h_\ell}$ of Ω_ℓ, $\Omega_\ell^{h_\ell} \subset \Omega_\ell$. Let $\mathcal{T}_{h_\ell}^\ell$ be a finite collection of nonempty, disjoint open simplices or hypercubes forming a partition of $\Omega_\ell^{h_\ell}$ such that $h_\ell = \max_{K \in \mathcal{T}_h^\ell} \left\{ h_\ell^K \right\}$, with h_ℓ^K denoting the diameter of the element $K \in T_h^\ell$. Note that in the case of nonconforming domain decomposition, $h_\ell \neq h_k$ for $\ell \neq k$. The mesh $\mathcal{T}_{h_\ell}^\ell$ is generated independently and sequentially in each subdomain Ω_ℓ, which means that it does not require communication. FEEL++ mesh data structure is defined through the type of geometrical entities (simplex or hypercube) and the geometrical transformation associated. The mesh entities (elements, faces, edges, nodes) are indexed either by their ids, corresponding to the process id to which they belong, their markers or their location. FEEL++ uses Boost.Multi-index to retrieve pairs of iterators over the containers of the entities depending on the usage context and the pairs of iterators are then turned into a range to be manipulated by the tools for submesh generation, the interpolation, the integration and the projection. The mesh data structure allows us to determine the neighboring subdomains as well as edge (in 2D and 3D) and face (in 3D) shared by neighboring subdomains.

Listing 6.5 : Local mesh generation

```
auto mesh = createGMSHMesh( _mesh=new mesh_type(CommSelf),
                    _prefix="fine",
                    _update=MESH_CHECK|MESH_UPDATE_FACES|MESH_UPDATE_EDGES,
                    _desc=domain( _name="localmesh",
                              _addmidpoint=false,
                              _usenames=false,
                              _shape="Hypercube",
                              _dim=Dim,
                              _h=(isMaster)?hsize1:hsize2,
                              _convex=convex,
                              _xmin=xmin,
                              _xmax=xmax,
                              _ymin=ymin,
                              _ymax=ymax,
                              _zmin=zmin,
                              _zmax=zmax,
                              _substructuring=true
                              ),
                    _structured=(grid=="fine")?structured:2,
                    _partitions=CommSelf.localSize(),
                    _worldcomm=CommSelf );
```

In Listing 6.5, the communicator `CommSelf` is purely local, see FIGURE 6.1.2.. Two different mesh characteristic sizes `hsize1` and `hsize2` are used according respectively to master and slave subdomains in order to process nonconforming meshes. The parameter `convex` represents the type of elements which can be simplex or hypercube.

Remark 6.3.1. The number of mesh files to generate is equal to the number of subdomains. Writing these files on the disk is very expensive for large number of subdomains and heavily penalizes the Input/Output (IO) filesystem on computational platforms. Thanks to the recent advances in FEEL++, a new feature that consists in generating mesh files directly in memory without writing to disk is available. This feature is of an utmost importance for the achieving of very large simulations on parallel computing platforms.

6.3.1.2 Complex Domain

We discuss the extension of the polygonal domain decomposition presented in section 6.3.1.1 to the general computational domain. Let $\Omega \subset \mathbb{R}^2$ be a bounded domain. First, we generate the coarse mesh $\mathcal{T}^H$ as in FIGURE 6.2 with hypercube elements (for the regularity at subdomain interfaces). Each element of the coarse mesh is considered as a subdomain of the domain decomposition. In our strategy of parallelization, only one subdomain is assigned to each processing unit. The coarse mesh $\mathcal{T}^H$ is loaded by all the processing units and each subdomain extracts its corresponding element K using the keyword `createSubmesh(elementId)`, where `elementId` means the id of the element in the coarse mesh $\mathcal{T}^H$. The generation of the fine mesh $\mathcal{T}_h^K$ is made locally in subdomain K using the GMSH interface of FEEL++. For the parallel resolution of the Discontinuous Galerkin coarse problem, the coarse mesh $\mathcal{T}^H$ is partitioned by using the graph partitioner Metis or Chaco.

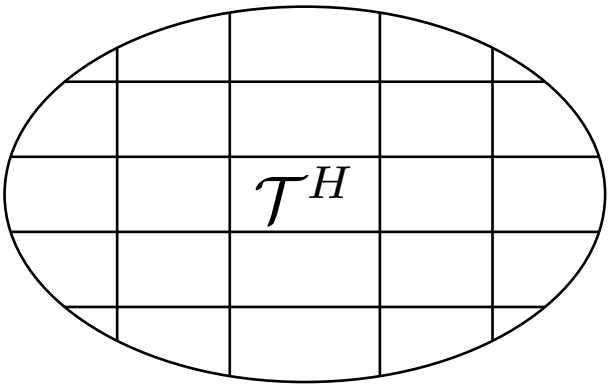


FIGURE 6.2 : Coarse mesh partition

6.3.1.3 Trace Meshes

In the mortar finite formulation method, the assembly of transfer matrices C_M and C_S presented in section 4.3.1 requires the exchange of trace mesh data structures between neighboring subdomains. We extract the trace meshes locally in each subdomain by using the FEEL++ submesh generation tool, the so-called createSubmesh [Cha13]. The global MPI communicator is used for the swap of extracted trace meshes between master and slave subdomains.

Algorithm 6.3. Local extraction of trace meshes

Require: mesh `# input local mesh`

1: **for** f $\in$ interfaces **do**
2: tracemesh[f] $\leftarrow$ createSubmesh(mesh,f) `# extraction of trace mesh`
3: **end for**

Listing 6.6 : Communications for exchange of trace meshes

```
std::vector<mpi::request> reqs;
for (std::string const& face : interface_flags )
 {
       const int pid = procid;
       const int npid = neighborProc(face);

       auto req1 = comm.isend(pid,pid,*(trace_send[face]));
       auto req2 = comm.irecv(npid,npid,*(trace_recv[face]));

       reqs.push_back(req1);
       reqs.push_back(req2);
 }
mpi::wait_all(reqs.begin(),reqs.end());
```

6.3.2 Algebraic Framework

The Algebraic representations are handled using a so-called backend which is a wrapper class that encapsulates several algorithms as well as data structures like vectors and matrices. It provides all the algebraic data structure behind function spaces, operators and forms. In the case of linear functionals, the representation is a vector and, in the case of linear operators and bilinear forms, the representation is a matrix. The backend abstraction allows to write code that is independent of the libraries used in the assembly process or to solve the linear systems involved, thus hiding all the details of that algebraic part under the hood of the backend.

In this work, we use two backends that provide an interface to PETSc for sparse matrices and Eigen for dense matrices which will be used in particular for storing the edge block preconditioners and the tridiagonal matrix associated with the Lanczos algorithm to estimate the condition number using the conjugate gradient coefficients.

Our purpose is to solve the reduced Schur complement system (4.41) preconditioned by the substructuring preconditioners presented in chapter 4. Traditionally, the Schur complement matrix is almost never assembled explicitly, since its expression depends on the local inverse matrices, see (4.34). We use the common technique that consists in computing the action of the Schur complement matrix $\widehat{\mathbf{S}}$ on a vector, only needed for the use of Krylov subspace method [Saa03] for solving the linear system (4.41). This technique is generally known as "shell" or "matrix-free" operations.

6.3.2.1 Index Sets for Substructuring

The principle of the substructuring approach is to consider a nonoverlapping splitting of the nonconforming discretization space in terms of interior, edge and vertex degrees of freedom. At the algebraic level, this consists in considering a nonoverlapping partition of the unknowns of local algebraic linear system into subsets corresponding to interior, edge and vertex unknowns. In our implementation strategy, we first consider the nonoverlapping splitting of the unknown $\mathbf{u}$ of the local Schur complement system (4.33) into $\mathbf{u}_{\mathtt{I}}$ and $\mathbf{u}_{\mathtt{B}}$ respectively the unknown components associated to interior nodes and the boundary nodes. Let $\mathtt{IS}_{\mathtt{I}}$ and $\mathtt{IS}_{\mathtt{B}}$ be the index sets representing the location of interior nodes $\mathbf{u}_{\mathtt{I}}$ and boundary nodes $\mathbf{u}_{\mathtt{B}}$ in the global solution vector $\mathbf{u}$. Feel++ provides a feature, the so-called `markerToDof` allowing to extract the degrees of freedom from geometric entity (vertex, edge, surface) markers. From the keyword "substructuring" of Feel++, we can mark the entities during the mesh generation. This information is stored in Feel++ mesh data structures and it is available after creating the function spaces.

The index sets $\mathtt{IS}_{\mathtt{I}}$ and $\mathtt{IS}_{\mathtt{B}}$ are essential for the extraction of submatrices $\mathbf{A}_{\mathtt{II}}$, $\mathbf{A}_{\mathtt{IB}}$, $\mathbf{A}_{\mathtt{BI}}$ and $\mathbf{A}_{\mathtt{BB}}$ needed for the local Schur complement system, see (4.33) and (4.34). These submatrices are extracted by using the Feel++ function, the so-called `createSubmatrix`, a interface for the `MatCreateSubMatrix` function of PETSc. The input parameters required for this function include the index sets for rows and columns corresponding to the submatrices in the initial

matrix. The Listing 6.7 illustrates the examples of submatrice extraction in FEEL++.

Listing 6.7 : Extraction of submatrices

```
// define local function space
auto Xh = Pch<2>(mesh);
// define index sets
auto IS_I = Xh->dof()->markerToDof("Interior");
auto IS_B = Xh->dof()->markerToDof("Boundary");
// create submatrices A_II, A_IB and A_BB from the global matrix A
auto A_II = createSubmatrix(A,IS_I,IS_I);
auto A_IB = createSubmatrix(A,IS_I,IS_B);
// create the transpose matrix A_BI of A_IB without assembly
auto A_BI = transpose(A_IB,MATRIX_TRANSPOSE_UNASSEMBLED);
auto A_BB = createSubmatrix(A,IS_B,IS_B);
```

Remark 6.3.2. In Listing 6.7, we do not require the assembled transpose matrix $\mathbf{A}_{\mathtt{BI}}$ of $\mathbf{A}_{\mathtt{IB}}$ but rather the ability to do the multiplication of $\mathbf{A}_{\mathtt{BI}}$ on a vector. This feature allows to gain a lot in terms of storage and operations.

The next step is the definition of the index sets $\mathtt{IS}_{\mathtt{V}}$ and $\mathtt{IS}_{\mathtt{E}}$ representing the location of vertex nodes $\mathbf{u}_{\mathtt{V}}$ and edge nodes $\mathbf{u}_{\mathtt{E}}$ in the solution vector $\mathbf{u}_{\mathtt{B}}$ of the Schur complement system (4.34). This decomposition is needed for the application of mortar constraint, see (4.37) and (4.38).

6.3.2.2 Application of the Schur Complement Matrix

The operation $\widehat{\mathbf{v}} = \widehat{\mathbf{S}}\widehat{\mathbf{u}}$ can be performed in the following steps :

$$1)\ \mathbf{w} = \mathbf{R}\widehat{\mathbf{u}} \quad 2)\ \mathbf{x} = Q\mathbf{w} \quad 3)\ \mathbf{y} = \Sigma\mathbf{x} \quad 4)\ \mathbf{z} = Q^T\mathbf{y} \quad 5)\ \widehat{\mathbf{v}} = \mathbf{R}^T\mathbf{z}.$$

Algorithm 6.4. Application of the Schur complement $\widehat{\mathbf{S}}$

Require: $\widehat{\mathbf{u}}, \widehat{\mathbf{v}}$ # input and output vectors
1: $\mathbf{w} \leftarrow \mathbf{R}\widehat{\mathbf{u}}$ # no communication
2: $\mathbf{x} \leftarrow Q\mathbf{w}$ # require communications from master to slave
3: $\mathbf{y} \leftarrow \Sigma\mathbf{x}$ # no communication
4: $\mathbf{z} \leftarrow Q^T\mathbf{y}$ # require communications from slave to master
5: $\widehat{\mathbf{v}} \leftarrow \mathbf{R}^T\mathbf{z}$ # no communication
6: **return** $\widehat{\mathbf{v}}$ # return output vector

The vectors $\mathbf{x}$ and $\mathbf{z}$ in the Algorithm 6.4. are given by

$$\mathbf{x} = \begin{pmatrix} \mathbf{w}_{\mathtt{V}} \\ \mathbf{w}_{\mathtt{M}} \\ Q_{\mathtt{V}}\mathbf{w}_{\mathtt{V}} + Q_{\mathtt{M}}\mathbf{w}_{\mathtt{M}} \end{pmatrix} \quad \text{and} \quad \mathbf{z} = \begin{pmatrix} \mathbf{y}_{\mathtt{V}} + Q_{\mathtt{V}}^T\mathbf{y}_{\mathtt{S}} \\ \mathbf{y}_{\mathtt{M}} + Q_{\mathtt{V}}^T\mathbf{y}_{\mathtt{S}} \end{pmatrix}, \tag{6.1}$$

where $Q_{\mathtt{M}}\mathbf{w}_{\mathtt{M}} = C_{\mathtt{S}}^{-1}C_{\mathtt{M}}\mathbf{w}_{\mathtt{M}}$ and $Q_{\mathtt{V}}^{T}\mathbf{y}_{\mathtt{S}} = C_{\mathtt{S}}^{-1}C_{\mathtt{V}}^{T}\mathbf{y}_{\mathtt{S}}$.

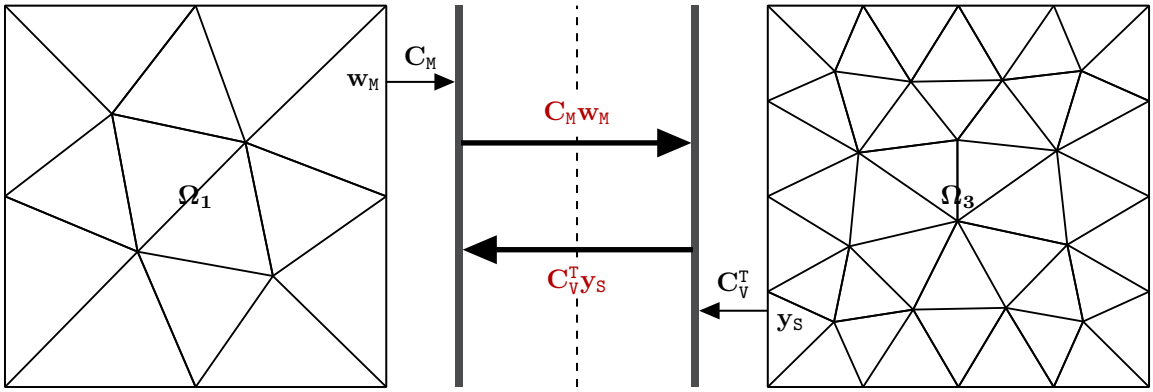


Figure 6.3 : MPI communications for transfer matrix-vector multiplication

The terms in color in (6.1) are those requiring interprocess communications between neighboring subdomains, see Figure 6.3.

6.3.2.3 Application of the Constraint Matrix Q

The application of the constraint matrix Q implies the operations $Q_{\mathtt{M}}\mathbf{w}_{\mathtt{M}} = C_{\mathtt{S}}^{-1}C_{\mathtt{M}}\mathbf{w}_{\mathtt{M}}$ and $Q_{\mathtt{V}}^{T}\mathbf{y}_{\mathtt{S}} = C_{\mathtt{S}}^{-1}C_{\mathtt{V}}^{T}\mathbf{y}_{\mathtt{S}}$, see (6.1). The interpolation from local function space to trace function spaces is needed in these operations since $C_{\mathtt{V}}$ and $\mathbf{y}_{\mathtt{S}}$ do not live on the same mesh. Let $\mathcal{I}_{S_h \to T_h^K}$ be the interpolation operator from the Schur function space S_h to trace function space T_h and let $\mathcal{I}_{T_h^K \to S_h}$ be the adjoint operator of $\mathcal{I}_{S_h \to T_h^K}$, $K = \{M\,, S\}$, depending on whether the side is master or slave.

Algorithm 6.5. Compute of $C_{\mathtt{S}}^{-1}C_{\mathtt{M}}\mathbf{w}_{\mathtt{M}}$

Require: $\mathbf{w}_{\mathtt{M}}$, $C_{\mathtt{S}}$, $C_{\mathtt{M}}$ `# input vector and matrices`
1: $\mathbf{x}^{\mathcal{I}} \leftarrow \mathcal{I}_{S_h \to T_h^K}(\mathbf{w}_{\mathtt{M}})$ `# apply interpolation operator`
2: $\mathbf{y} \leftarrow C_{\mathtt{M}}\mathbf{x}^{\mathcal{I}}$ `# apply` $C_{\mathtt{M}}$
3: Solve $C_{\mathtt{S}}\mathbf{z} = \mathbf{y}$ `# use direct solver`
4: $\mathbf{v}_{\mathtt{S}} \leftarrow \mathcal{I}_{T_h \to S_h^K}(\mathbf{z})$ `# apply inverse interpolation operator`
5: **return** $\mathbf{v}_{\mathtt{S}}$ `# return output vector`

Listing 6.8 : Interpolation operators

```
// operator interpolation I_{S_h -> T_h^K}, K = {M, S}
opI = opInterpolation( _domainSpace=Sh,
                       _imageSpace=Th,
                       _range=elements(Th->mesh()),
                       _backend=M_backend);
```

```
// operator interpolation transpose I_{T_h^K -> S_h}, K = {M, S}
opI->matPtr()->transpose(opIMATTRANS, MATRIX_TRANSPOSE_UNASSEMBLED);
```

Remark 6.3.3. As in remark 6.3.2, the transpose `opIMATTRANS` of the matrix associated to the linear interpolation operator `opI` in Listing 6.8 is not assembled explicitly but only its action on a vector is required.

6.3.2.4 Application of the Change of Basis Matrix R

The application of the change of basis matrix defined in (4.39) is done locally for each subdomain as

$$\mathbf{R}\begin{pmatrix}\mathbf{u}_{\mathsf{V}}\\ \mathbf{u}_{\mathsf{E}}\end{pmatrix} = \begin{pmatrix}0\\ \mathbf{u}_{\mathsf{E}}\end{pmatrix} + \mathbf{R}_{\mathsf{V}}\begin{pmatrix}\mathbf{u}_{\mathsf{V}}\\ \mathbf{u}_{\mathsf{E}}\end{pmatrix} \tag{6.2}$$

The matrix-vector multiplication in (6.2) does not require communication and involves mainly the action of the matrix $\mathbf{R}_{\mathsf{V}}$ on a given vector. Given as the matrix $\mathbf{R}_{\mathsf{V}}$ is the algebraic representation of the piecewise polynomial linear interpolation operator from a coarse mesh (with two simplex elements) to the local mesh, its construction is done by using FEEL++ linear interpolation framework discussed in section 6.2.

Listing 6.9 : Construction of the matrix $\mathbf{R}_{\mathsf{V}}$

```
// Schur function space
auto Sh = Pch<2>(tracemesh);
// function space defined on the coarse mesh with two simplex elements
auto Ch = Pch<1>(coarsemesh);
// operator interpolation I_{C_h -> S_h}
opI = opInterpolation( _domainSpace=Ch,
                       _imageSpace=Sh,
                       _range=elements(Sh->mesh()),
                       _backend=M_backend);

// matrix R_V associated to opI
auto Rv = opI->matPtr();
```

6.3.2.5 Application of the Preconditioner P

The substructuring preconditioner $\mathbf{P}$ is applied locally for each subdomain, see Algorithm 6.6.. The matrix-vector multiplication $\mathbf{P}_{\mathsf{E}}\mathbf{u}_{\mathsf{E}}$ is a purely local operation performed homogeneously for each subdomain. This operations do not require communication (see Algorithm 6.7.) since the block preconditioners $\widehat{\mathbf{K}}_{\mathsf{E}_j}$, $j = 1, \ldots, \mathtt{M}$, are thoroughly independent, see (4.42).

Algorithm 6.6. Application of the preconditioner $\mathbf{P}$

Require: $\mathbf{u}$, $\mathbf{v}$ # input and output vectors
1: $\mathbf{v}_V \leftarrow \mathbf{P}_V \mathbf{u}_V$ # require communications between neighboring subdomains
2: $\mathbf{v}_E \leftarrow \mathbf{P}_E \mathbf{u}_E$ # no communication
3: **return** $\mathbf{v}$ # return output vector

Algorithm 6.7. Application of the edge block preconditioner $\mathbf{P}_E$

Require: $\mathbf{u}_E$, $\mathbf{v}_E$ # input and output vectors
1: **for** $i = 1, \ldots, M$ **do**
2: $\mathbf{v}_{E_i} \leftarrow \widehat{\mathbf{K}}_{E_i} \mathbf{u}_{E_i}$ # local matrix-vector product on the i^{th} master edge
3: **end for**
4: **return** $\mathbf{v}_E$ # return output vector

6.3.2.6 Vertex Block Preconditioner

We develop two different implementations of the vertex block preconditioner $\mathbf{P}_V^{DG}$. In the first one, we solve the coarse problem sequentially on a master process using the Discontinuous Galerkin (DG) formulation in Feel++ (see Listing 6.12). Each subdomain retrieves its contribution on its processor rank through the MPI communications for the local application of the coarse component of the substructuring preconditioner. In the second approach, we solve the coarse problem in parallel on a group of master processes (see Figure 6.4.1.). The communications are handled explicitly inside subgroups (see Figure 6.4.2.) and implicitly between master processes (see Figure 6.4.1.).

Let `nmp` be the number of selected processors for solving the coarse problem in parallel. First, we load the coarse mesh sequentially one processor, for example the process rank 0 and we partition it into `nmp` partitions by using the mesh partitioner Metis or Chaco from Gmsh. We select for example the first element id in each partition as master processes. On the master process ranks, we define a master MPI communicator from the global MPI communicator by using Boost.MPI `group`, a interface allowing the creation of communicators for subgroups of processors, as shown Listing 6.10. Let `masterRanks` be the iterator range of master ranks and let `worldcomm` the global MPI communicator, see Figure 6.1.1..

Listing 6.10 : Construction of master communicator

```
// define a group including only master processes
auto masterGroup = worldcomm().group().include(masterRanks.begin(),masterRanks.end());
// define a master communicator
auto masterComm = mpi::communicator(worldcomm,masterGroup);
```

Analogously, we define a subcommunicator that activates only the processes within each partition, see Listing 6.11. Let `subRanks` be the iterator range of process ranks activated in the partition.

Listing 6.11 : Construction of subgroup communicators

```
// define a group including only subgroup processes
auto subGroup = worldcomm().group().include(subRanks.begin(),subRanks.end());
// define a subgroup communicator
auto subComm = mpi::communicator(worldcomm,subGroup);
```

We design the Discontinuous Galerkin formulation assigned to the master processes and fully supported by Feel++.

Listing 6.12 : Discontinuous Galerkin formulation in Feel++

```
// define coarse mesh in parallel on master processes
auto mesh = loadMesh( _mesh=new Mesh<Hypercube<2> >,
                      _partitions=masterComm.globalSize(),
                      _worldcomm=masterComm );

// define DG function space of piecewise linear polynomials
auto Vh = Pdh<1>( mesh, true );
auto u = Vh->element();
auto v = Vh->element();

// assemble DG problem
auto dgform = form2( _trial=Vh, _test=Vh,
                     _pattern=size_type(Pattern::EXTENDED) );

dgform = integrate(_range=elements(mesh),
                   _expr=β*gradt(u)*trans(grad(v)) ); // ∫_Ω β∇u · ∇v

dgform += integrate( internalfaces( mesh ),
                    + γ*( trans( jumpt( idt( u ) ) )*jump( id( v ) ) )/hFace() ); // γ⟦u⟧.⟦v⟧/h
```

The application of the coarse DG preconditioner defined in Listing 6.12 is performed in three steps and requires collective communications (`gather` and `scatter`) in each subgroup, see Figure 6.4.2.. These communications enable the vertex data from the fine grid to the coarse grid and vice versa thanks to the simple pattern of the table of degrees of freedom of Discontinuous Galerkin problems.

The first step consists in sending the solution vector of each subgroup process to the subgroup master process by using Boost.MPI `gather`, see Listing 6.13.

Listing 6.13 : Communications from subgroup processes to subgroup master process

```
// subgroup master processs
int subMasterRank = subGroup.masterRank();
// sending the solution vectors to subgroup master processs
if (subGroup.rank()==subMasterRank)
{
   mpi::gather(subGroup, inputvec, 4, gathervec, subMasterRank);
}
```

```
else
{
   mpi::gather(subGroup, inputvec, 4, subMasterRank);
}
```

The second step is dedicated to the resolution of the coarse DG problem in parallel using a direct solver, for example `mumps`, see Listing 6.14.

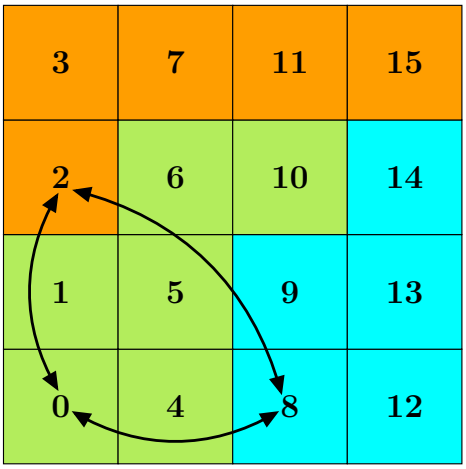
Listing 6.14 : Solve DG problem in parallel

```
// solve DG problem using direct solver mumps
dgform.solve(_solution=scattervec,
             _rhs=gathervec,
             _pcfactormatsolverpackage=''mumps'' );
```

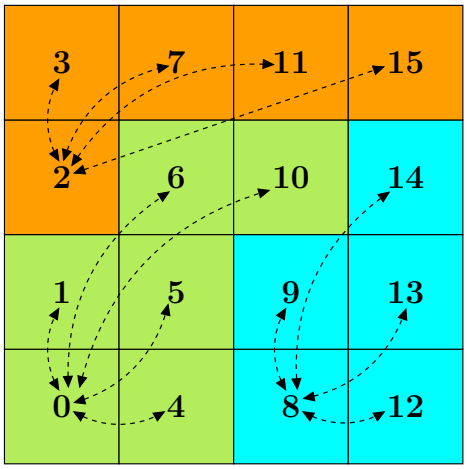
The last step consists in sending the DG problem solution vector from subgroup master process to each subgroup process by using Boost.MPI `scatter`, see 6.15.

Listing 6.15 : Communications from subgroup master process to subgroup processes

```
// receive subgroup process contributions from subgroup master process
if (subGroup.rank()==subMasterRank)
{
   mpi::scatter(subGroup, scattervec, outvec, 4, subMasterRank);
}
else
{
   mpi::scatter(subGroup, outputvec, 4, subMasterRank);
}
```



6.4.1. Master communications



6.4.2. Communications inside subgroups

Figure 6.4 : MPI communications for vertex block preconditioner $\mathbf{P}_V^{DG}$

Remark 6.3.4. The parallel implementation of the vertex block preconditioner $\mathbf{P}_V^{DG}$ will improve the load balancing and therefore the performance of our preconditioner on very large

scale architectures since the coarse problem size is four times the number of subdomains (number of cores) in two-dimensional space.

6.3.2.7 Condition Number Estimate

We solve the transformed Schur complement system (4.41) by using the Preconditioned Conjugate Gradient (PCG) method [Saa03]. The condition number of the (preconditioned) Schur complement matrix is estimated from the (preconditioned) conjugate gradient coefficients by using the relationship between Lanczos technique and the PCG method, see [GV96] for more details.

Algorithm 6.8. PCG algorithm for Schur complement system (4.41)

Compute $\mathbf{r}_0 = \widehat{\mathbf{g}} - \widehat{\mathbf{S}}\mathbf{x}_0$ `# initialize the residual` $\mathbf{r}_0$
for $i = 1, 2, \ldots$ maxiter **do**
  solve $\mathbf{P}\mathbf{z}_{i-1} = \mathbf{r}_{i-1}$ `# apply the preconditioner` **P**
  $\rho_{i-1} = (\mathbf{r}_{i-1}, \mathbf{z}_{i-1})$
  if $i = 1$ **then**
    $\mathbf{p}_1 = \mathbf{z}_0$
  else
    $\beta_{i-1} = \dfrac{\rho_{i-1}}{\rho_{i-2}}$
    $\mathbf{p}_i = \mathbf{z}_{i-1} + \beta_{i-1}\mathbf{p}_{i-1}$
  end if
  $\mathbf{q}_i = \widehat{\mathbf{S}}\mathbf{p}_{i-1}$
  $\alpha_i = \dfrac{\rho_{i-1}}{(\mathbf{p}_i, \mathbf{q}_i)}$
  $\mathbf{x}_i = \mathbf{x}_{i-1} + \alpha_i\mathbf{p}_i$
  $\mathbf{r}_i = \mathbf{r}_{i-1} + \alpha_i\mathbf{q}_i$
  check convergence ; continue if necessary
end for

The central idea of the conjugate gradient method is to construct the minimum error solution in $\widehat{\mathbf{S}}$-norm over the Krylov space $\mathcal{K}_i = \text{span}\{\mathbf{r}_0, \widehat{\mathbf{S}}\mathbf{r}_0, \ldots, \widehat{\mathbf{S}}\mathbf{r}_{i-1}\}$, see Algorithm 6.8. A rich literature devoted to the iterative Krylov subspace method is available in [Saa03].

The Lanczos algorithm [GV96] applied to the matrix $\widehat{\mathbf{S}}$ constructs a tridiagonal matrix T whose smallest and largest eigenvalues converge to the smallest and largest eigenvalues of $\widehat{\mathbf{S}}$ respectively denoted by $\lambda_{min}(\widehat{\mathbf{S}})$ and $\lambda_{max}(\widehat{\mathbf{S}})$. Let T_m be the tridiagonal matrix associated with the m-th step of the Lanczos algorithm [Saa03, p. 214-215, Algorithm 6.15]. T_m is defined by $T_m = \text{tridiag}[\eta_i^m, \delta_i^m, \eta_{i+1}^m]$. The coefficients η_i^m and δ_i^m are functions of conjugate gradient coefficients α_i and β_i defined in Algorithm 6.8.. The tridiagonal matrix entries η_i^m and δ_i^m are defined by

$$\eta_{i+1}^m = \frac{\sqrt{\beta_{i-1}}}{\alpha_{i-1}} \quad \text{and} \quad \delta_{i+1}^m = \begin{cases} \dfrac{1}{\alpha_i} & \text{for } i = 0 \\ \dfrac{1}{\alpha_i} + \dfrac{\beta_{i-1}}{\alpha_{i-1}} & \text{for } i > 0 \end{cases}$$

This technique to obtain condition number estimate is valid for the preconditioned conjugate gradient method (see Algorithm 6.8.), i.e., if we apply the Lanczos algorithm to the preconditioned matrix $\mathbf{P}^{-1}\widehat{\mathbf{S}}$, the extremal eigenvalues of the tridiagonal matrix T_m converge to the extremal eigenvalues of $\mathbf{P}^{-1}\widehat{\mathbf{S}}$. An estimation of the condition number $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$ of the preconditioned Schur complement matrix is defined by

$$\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}}) = \frac{\lambda_{max}(\mathbf{P}^{-1}\widehat{\mathbf{S}})}{\lambda_{min}(\mathbf{P}^{-1}\widehat{\mathbf{S}})},$$

where $\lambda_{min}(\mathbf{P}^{-1}\widehat{\mathbf{S}})$ and $\lambda_{max}(\mathbf{P}^{-1}\widehat{\mathbf{S}})$ are estimated respectively by smallest and largest eigenvalues of the Lanczos tridiagonal matrix T_m.

In our implementation, the symmetric m-dimensional Lanczos tridiagonal matrix T_m is stored in dense format. We compute its smallest and largest eigenvalues by using the EIGEN [G+10] library.

6.4 Notes on Implementation in 3D

The extension of the substructuring preconditioners for two-dimensional mortar finite element method presented in this thesis to three-dimensional problems is an ongoing work. The principal ingredients which are under development include the construction of the mortar projector and the coarse grid operator on the wirebasket (the union of the edges and vertices). The construction of the mortar projection operator is almost done for first-order approximation ($p = 1$) and its extension to any polynomial order is a highly technical task which is in progress. Other points required for the realization of the preconditioners in three-dimensional space are available : the *(i)* construction of the Schur complement system and *(ii)* the efficient computation of matrix square root, essential for the face block preconditioning.

6.5 Complexity Analysis

6.5.1 Data Structure

For our parallel implementation, we chose the non-clusterization strategy, i.e. one subdomain per processing unit. The data related to each subdomain, e.g. local mesh, local function space, local stiffness matrix, local linear interpolation operator are stored in the local memory. The

assembly of the transfer matrix $C_{\mathtt{M}}$ requires the exchange of trace meshes between neighboring subdomains. The exchange of trace solutions is needed for the application of the constraint matrix Q and its transpose Q^T. These shared data are copied between processing units and therefore increase the memory complexity.

6.5.2 Communication

One of interesting points of the mortar finite element method is that continuity at the cross-points (in 2D and 3D) and the cross-edges (in 3D) is not required. In the implementation view point, this means that there is no communication between neighboring subdomains through the cross-points, which significantly reduces the time complexity. Solving and the application of the coarse grid preconditioner have a significant effect on the communication cost. In this framework, the size of the coarse problem is four times the number of processing units (number of subdomains), that can be solved in sequential or in parallel depending on the configurations, specially the number of processing units employed for solving the overall problem. The main rule we have adopted is that the communications should not be too many compared with the workload in order to not adversely affect the computational cost. Indeed, we have the ability to intuitively choose the suitable number of processing units for solving the coarse grid problem.

6.5.3 Load Balancing

The load balancing has a great influence on the performance of a parallel algorithm. The mortar finite element method naturally involves imbalance since the master and the slave subdomains are not handled in the same way, e.g. the assembly of transfer matrix $C_{\mathtt{M}}$. The solution of the coarse grid problem adds another source of imbalance since it is performed on a selection of processing units among all resources dedicated to the overall problem. An equilibrated mortar approach introduced in [JMN13] and based on Schwarz type methods with Robin interface conditions can improve the load balancing.

6.5.4 Synchronization

The synchronization manages the sequence of work and the tasks execution for parallel algorithms. It is an important factor that can affect the performance of a parallel application. The process synchronization point is a point where all processing units must arrive before starting at the same time a given task. In our framework, this feature is used mostly in the compute of inner products in the Krylov subspace iterations.

6.5.5 Scalability

The concept of scalability was introduced in section 1.7.

6.6 Code Design

For the numerical implementation of the methodology described in chapter 3 and in chapter 4, we developed the code in C++11. Without going into detailed descriptions, we will just give a brief overview to the four main classes defined in the implementation code : `class Subdomain`, `class LocalProblemBase`, `class LocalProblem` and `class Mortar`.

6.6.1 Class Subdomain

It is a virtual class, from which the `class LocalProblemBase` is derived. This class handles the domain decomposition (one subdomain per processor core) and provides all the ids of the neighboring subdomains from the different edges. It also provides the following main methods :

- `xmin` : returns the minimum length of the subdomain in X-direction
- `xmax` : returns the maximum length of the subdomain in X-direction
- `ymin` : returns the minimum length of the subdomain in Y-direction
- `ymax` : returns the maximum length of the subdomain in Y-direction
- `zmin` : returns the minimum length of the subdomain in Z-direction
- `zmax` : returns the maximum length of the subdomain in Z-direction
- `isInterior` : return true or false depending on whether the current subdomain is an interior subdomain or not
- `isOnBoundary` : return true or false depending on whether the current subdomain is on boundary or not
- `isMaster` : return true or false depending on whether the current subdomain is a master subdomain or not
- `isSlave` : return true or false depending on whether the current subdomain is a slave subdomain or not

Listing 6.16 : Class Subdomain

```
template<int Dim>
class Subdomain
{
 public:
  /*
     @param pid        process id for the current subdomain
     @param nx         number of subdomain in X-direction
     @param ny         number of subdomain in Y-direction
```

```
    @param nz          number of subdomain in Z-direction
    @param vm          variables map for options
*/

 // constructor
 Subdomain( int pid, int nx, int ny, int nz, po::variables_map const& vm );

 // virtual destructor
 virtual ~Subdomain(){}

};
```

6.6.2 Class LocalProblemBase

This class is derived from `class Subdomain` and allows the `shell` construction of the constrained Schur complement system (4.38), the transformed Schur complement system (4.41) and the change of basis operators (4.39). It provides the following main methods :

- `createMesh` : create the local meshes(fine and coarse). The exchange of trace meshes between neighboring subdomain is also performed in this function
- `createFunctionSpaces` : create the local function spaces(fine, coarse, traces and mortar)
- `interpolation` : create interpolation operators (i) from domain space to trace space (ii) from domain space to corse space
- `createIndices` : create indices corresponding to the decomposition $u = u^0 + u^E + u^V$. This indices are used for the extraction of submatrices needed for the construction Schur complement system and substructuring preconditioners
- `assembleProblem` : assemble local stiffness matrix A, local right hand side vector F and the transfer matrices $C_{\mathtt{M}}$, $C_{\mathtt{S}}$ and $C_{\mathtt{V}}$
- `schurComplement` : create `shell` operator for the Schur complement system (4.34)
- `switchingMatrix` : apply the switching matrix Q, see (4.38)
- `switchingMatrixTrans` : apply the switching matrix transpose Q^T, see (4.38)
- `multVector` : create `shell` operator for the constraint Schur complement system (4.38)
- `rightHandSide` : create `shell` operator for the right hand side corresponding to the constraint Schur complement system (4.38)
- `referenceVertexPrecondApply` : apply the reference coarse preconditioner (4.17)
- `changeBasis` : apply the change of basis operator **R**, see (4.39)
- `changeBasisTrans` : apply the change of basis operator transpose (4.40).

Listing 6.17 : Class LocalProblemBase

```
template<int Dim,int Order>
class LocalProblemBase: public Subdomain<Dim>
{
 public:
  /*
     @param pid               process id for the current subdomain
     @param nx                number of subdomain in X-direction
     @param ny                number of subdomain in Y-direction
     @param nz                number of subdomain in Z-direction
     @param vm                variables map for options
     @param worldcomm         mpi communicator
     @param grid              fine/coarse grid
 */

  // constructor
  LocalProblemBase( int pid, int nx, int ny, int nz, po::variables_map const& vm,
                    WorldComm const& worldcomm, std::string grid=''fine'' )
     :
     Subdomain<Dim>( pid, nx, ny, nz, vm )
  {}

};
```

6.6.3 Class LocalProblem

This class is derived from `class LocalProblemBase` and allows the `shell` construction of the substructuring preconditioners $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$ described in chapter 4, the parallel/sequential assembly of the Discontinuous Galerkin coarse preconditioners presented in section 4.2.2. The solvers (`CG`, `BICGSTAB`, `MINRES`) used for solving the (preconditioned) Schur complement system are implemented in this class. the `class LocalProblemBase` provides the methods for the post processing operations which are (i) the resolution of the local linear system with the Schur complement solution as boundary conditions (ii) the compute of the numerical errors (L^2 and H^1) (iii) the exporting of the numerical results.. The main methods are

- `schurMatrixPrecond` : apply the substructuring preconditioner
- `vertexPrecondApply` : apply the coarse preconditioner
- `vertexDGPrecondApply` : apply the DG coarse preconditioner in sequential
- `vertexParallelDGPrecondApply` : apply the DG coarse preconditioner in parallel
- `solve` : solve the (preconditioned) Schur complement system
- `l2Error` : compute the numerical L^2 error
- `h1Error` : compute the numerical H^1 error
- `exportResults` : export the numerical results

Listing 6.18 : Class LocalProblem

```
template<int Dim,int Order>
class LocalProblem: public LocalProblemBase<Dim,Order>
{
 public:
  /*
     @param pid               process id for the current subdomain
     @param nx                number of subdomain in X-direction
     @param ny                number of subdomain in Y-direction
     @param nz                number of subdomain in Z-direction
     @param vm                variables map for options
     @param worldcomm         mpi communicator
     @param cs                coarse Schur complement for preconditioner
 */

  // constructor
  LocalProblem( int i, int nx, int ny, int nz, po::variables_map const& vm,
                WorldComm const& worldcomm, coarseproblem_ptrtype cs  )
     :
     LocalProblemBase<Dim,Order>( i, nx, ny, nz, vm, worldcomm )
  {}

};
```

6.6.4 Class Mortar

This class is derived from `class LocalProblem` and the `class Simget` of Feel++. It mainly provides methods for automatic convergence analysis (in h, in H and in p) and automatic scalability analysis (strong and weak). The main methods are

- `convergenceStudy` : exports data files for automatic convergence analysis
- `strongScalingStudy` : exports data files for automatic strong scalability analysis
- `weakScalingStudy` : exports data files for automatic weak scalability analysis

Listing 6.19 : Class Mortar

```
template<int Dim,int Order>
class Mortar: public std::map<int, boost::shared_ptr<LocalProblem<Dim,Order> > >,
              public Simget
{
 public:

  // constructor
  Mortar()
     :
     Simget()
  {}

};
```

Listing 6.20 : Main funtion

```
int main(int argc, char** argv )
{
  // use Feel namespace
  using namespace Feel;

  // Initialize Feel++ Environment
  Environment env( _argc=argc,
                   _argv=argv,
                   _desc=makeOptions(),
                   _about=about(_name=''mortar'',
                                _author=''Abdoulaye Samake'',
                                _email=''abdoulaye.samake@imag.fr'') );

  // create an application
  Application app;

  // instanciate Mortar in 2D
  app.add( new Mortar<2,FEELPP_ORDER>() );

  // run the application
  app.run();
}
```

6.7 Conclusion

In this chapter, we introduced the numerical implementation of the substructuring preconditioners for mortar element method in two dimensional space, previously described in chapter 4. We first recalled the essential ingredients including MPI, PETSc and Gmsh. We placed a special emphasis on linear interpolation operator which is a crucial tool for domain decomposition methods in Feel++. We presented two different MPI communication approaches namely explicit and seamless approach. For the first approach, we introduced Boost.MPI and Boost.Serialization that provide a simple interface for MPI usage. We discussed a generic geometric and algebraic framework for mortar discretization and the construction and analysis of the substructuring preconditioners. This implementation will be followed by the numerical experiments supporting the theoretical properties of the preconditioners and the performance of our parallel algorithms in chapter 6.

Chapter 7

Generic Implementation Framework

This chapter discusses the implementation aspects for schwarz methods in section 7.1, three-field method in section 7.2 and mortar element method with lagrange multipliers in section 7.3.

Ce chapitre aborde les aspects de mise en oeuvre des méthodes de schwarz dans la section 7.1, de la méthode three-field dans la section 7.2 et de la méthode des éléments finis mortar avec multiplicateurs de lagrange dans la section 7.3.

Contents

7.1 Schwarz Methods

In this section, we discuss two different approaches for Schwarz methods in Feel++ using explicit communications (see section 7.1.1) and seamless communications (see section 7.1.2). In the first approach, we deal with different types of Schwarz methods(Additive, Multiplicative, with(out) Relaxation) with different artificial boundary conditions (DD, DN, NN, RR) while having the ability to process (non-)conforming meshes as well as being able to control the size of the overlap between neighboring subdomains. In the second approach, we use the parallel data structures of Feel++ and the algebraic domain decomposition framework provided by PETSc.

7.1.1 Explicit Communication Approach

Schwarz methods are used as solvers and the communications are handled explicitly. We use PETSc sequentially even though the code is parallel using MPI communicators. It requires explicitly sending and receiving complex data structures such as mesh data structures and elements of functions space (traces). A sequential interpolation operator is also used to make the transfer between the grids (overlapping or not, conforming or not). In this case each subdomain creates locally its mesh and its function space, the matrices and vectors associated to the discretization process are completely local.

The variational formulation of the problem (2.4) in the simplest form ($L := -\Delta$) in the subdomain Ω_i at iteration number k using Nitsche's method [Nit71] for applying weakly the Dirichlet-Dirichlet artificial boundary conditions ($C_i = C_j = Id\,,\ j \in \mathcal{V}_{\Omega_i}$) is given by : find $u_i^k \in H^1(\Omega_i)$ such that $a(u_i^k\,, v) = l(v)\ \forall v \in H^1(\Omega_i)$ where

$$a(u_i^k, v) := \int_{\Omega_i} \nabla u_i^k \cdot \nabla v + \int_{\partial\Omega_i} -\frac{\partial u_i^k}{\partial n} v - \frac{\partial v}{\partial n} u_i^k + \frac{\gamma}{h} u_i^k v \tag{7.1}$$

$$l(v) := \int_{\Omega_i} fv + \int_{\partial\Omega_i \setminus \Gamma_{ij}} \Big(-\frac{\partial v}{\partial n} + \frac{\gamma}{h} v \Big) g + \sum_{j \in \mathcal{V}_{\Omega_i}} \int_{\Gamma_{ij}} \Big(-\frac{\partial v}{\partial n} + \frac{\gamma}{h} v \Big) u_j^{k-1} \tag{7.2}$$

with γ a penalization parameter and h the maximum mesh size.

Other variants of artificial boundary conditions such as Dirichlet-Neumann ($C_i = Id\,,\ C_j = \partial/\partial n\,,\ j \in \mathcal{V}_{\Omega_i}$), Neumann-Neumann ($C_i = C_j = \partial/\partial n\,,\ j \in \mathcal{V}_{\Omega_i}$) and Robin-Robin ($C_i = C_j = (\partial/\partial n) + Id\,,\ j \in \mathcal{V}_{\Omega_i}$) are also treated. In the above variational formulation, only the terms colored in red in (7.2) requires communications between neighboring subdomains for each Schwarz iteration and interpolation between the grids. Note that the assembly of the other terms of the variational formulation is done once and is purely local.

The listing 7.1 illustrates some aspects of Schwarz algorithm using the Feel++ language.

Listing 7.1 : Feel++ snippet code for parallel Schwarz algorithm

```
// Create local mesh and function space on subdomain number i
```

```
auto mesh = createGMSHMesh(_mesh=mesh_type, ...);
auto Xh = space_type::New(mesh);
std::vector<mpi::request> reqs; // vector of Boost.MPI requests
for(int j=0, j< Nneighbors, ++j){
 // Extract trace mesh on interface number j
 trace_mesh_send[j]=mesh->trace(markedfaces(mesh,j));
 // Exchange trace mesh with neighbor subdomain number j
 auto req1=comm.isend( j,i,trace_mesh_send[j] );
 auto req2=comm.irecv( j,j,trace_mesh_recv[j] );
 reqs.push_back(req1); reqs.push_back(req2);
} mpi::wait_all(reqs.begin(), reqs.end());// wait all requests
for(int j=0, j< Nneighbors, ++j){
 // Create trace function space for interface number j
 TXh[j] = trace_space_type::New(trace_mesh_recv[j]);
 // Create interpolation operator from Xh to TXh[j]
 opI[j]=operatorInterpolation(Xh,TXh[j]); }
while(!convergence) { // Schwarz iterations
 reqs.clear();
 for(int j=0, j< Nneighbors, ++j){
  // Non conforming interpolation for interface number j
  opI[j]->apply(solution,trace_solution_send[j]);
  // Exchange trace solution with neighbor subdomain number j
  auto req1=comm.isend( j,i,trace_solution_send[j] );
  auto req2=comm.irecv( j,j,trace_solution_recv[j] );
  reqs.push_back(req1); reqs.push_back(req2);
 } mpi::wait_all(reqs.begin(), reqs.end());// wait all requests
  // Update right hand side for each schwarz iteration
 for(int j=0, j< Nneighbors, ++j){
  form1( _test=Xh,_vector=F ) +=
   integrate(elements(trace_mesh_send[j]),
    -grad(v)*N()*idv(trace_solution_recv[j])
    +penaldir*idv(trace_solution_recv[j])*id(v)/hFace()); }
 solve(); }
```

7.1.2 Seamless Communication Approach

Here we consider the domain decomposition methods with seamless communications in Feel++. We provide a parallel data framework : we start with automatic mesh partitioning using Gmsh (Chaco/Metis) — adding information about ghosts cells with communication between neighbor partitition ; — then Feel++ data structures are parallel such as meshes, (elements of) function spaces — create a parallel degrees of freedom table with local and global views ; — and finally we use the PETSc Krylov subspace solvers (KSP) coupled with PETSc preconditioners such as Block-Jacobi, ASM, GASM. The last preconditioner is an additive variant of the Schwarz alternating method for the case of many subregion, see [BBG04]. For each sub-preconditioners (in the subdomains), PETSc allows to choose in the wide range of sequential preconditioners such, ilu, jacobi, ml.

To illustrate this, we perform a strong scalability test with a Laplace problem in 3D using P3 Lagrange elements (about 8 Millions degrees of freedom). The listing 7.2 corresponds to the code that allowed us to realize this test.

Listing 7.2 : Laplacian Solver in parallel

```
/* Create parallel function space and some associated elements */
auto Xh = space_type::New( _mesh=mesh );
/* Create the parallel matrix and vector of linear system */
auto A = backend()->newMatrix(_test=Xh, _trial=Xh);
auto F = backend()->newVector(Xh);
/* Parallel assembly of the right hand side */
form1( _test=Xh, _vector=F )=
    integrate( _range=elements( mesh ), _expr=f*id( v ) )
/* Parallel assembly of the global matrix */
form2( _test=Xh, _trial=Xh, _matrix=A ) =
    integrate( _range=elements( mesh ),
               _expr=gradt(u)*trans(grad(v)) );
/* Apply Dirichlet boundary conditions strongly */
form2( _test=Xh, _trial=Xh, _matrix=A ) +=
  on( _range=boundaryfaces(mesh),
      _element=u,_rhs=F, _expr=g );
/* solve system using PETSc parallel solvers/preconditioners */
backend()->solve( _matrix=A, _solution=u, _rhs=F );
```

7.2 Three-field Method

In Listing 7.3, we display the terms corresponding to the jump matrices B_1 , C_1 , B_2 and C_2 in (2.31).

Listing 7.3 : Assembly of the jump terms in global matrix

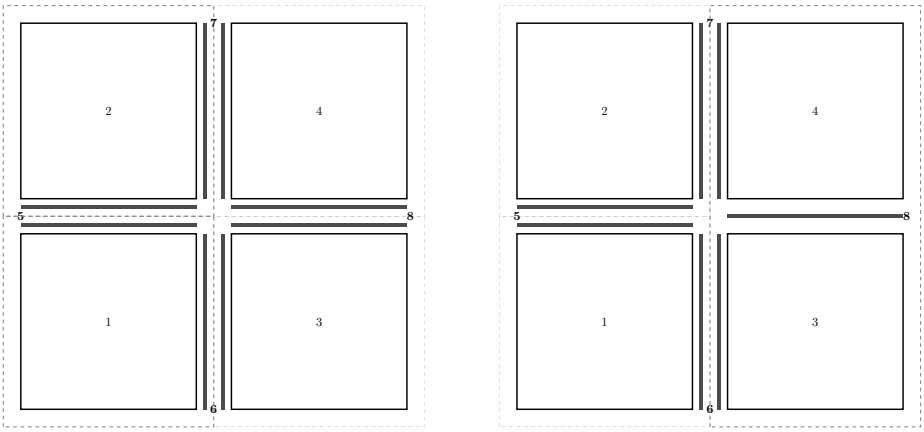
```
// Product function spaces Xh_1 × Λ_1h × Λ_h × Λ_2h × Xh_2 for Ω_1 × Γ_1 × Γ × Γ_2 × Ω_2
 typedef meshes<msh1_t,tr1_t,tr_t,tr2_t,msh2_t> mesh_type;
 typedef bases<PSet1,PSet1,PSet3,PSet2,PSet2> basis_t;
 typedef FunctionSpace< mesh_type, basis_t > space_type;
 auto mesh = fusion::make_vector(msh1,t_msh1,t_msh,t_msh2,msh2);
 auto Xh = space_type::New( mesh );
 auto u = Xh->element();
 auto u1 = u.element<0>();
 auto mu1 = u.element<1>();
 auto mu = u.element<2>();
 auto mu2 = u.element<3>();
 auto u2 = u.element<4>();

 // Initialize the bilinear form associated to the global matrix A
 auto A = backend->newMatrix( _trial=Xh, _test=Xh );
 form2( _trial=Xh, _test=Xh, _matrix=A);

 // Assembly the stiffness terms in Ω_1
 form2( _trial=Xh, _test=Xh, _matrix=A ) +=
 integrate( elements(Xh->template mesh<0>()), gradt(u1)*trans(grad(u1)) );

 // Assembly the stiffness terms in Ω_2
 form2( _trial=Xh, _test=Xh, _matrix=A ) +=
 integrate( elements(Xh->template mesh<4>()), gradt(u2)*trans(grad(u2)) );

 // Add the jump terms in the global matrix A
 form2( _trial=Xh, _test=Xh, _matrix=A) +=
```

```
integrate( elements(Xh->template mesh<2>()), idt(u1)*id(mu1)+id(u1)*idt(mu1)
                                    +idt(mu)*id(mu1)+idt(mu1)*id(mu)
                                    -idt(u2)*id(mu2)-id(u2)*idt(mu2)
                                    -idt(mu)*id(mu2)-idt(mu2)*id(mu));
```

7.3 Mortar Element Method with Lagrange Multipliers

The parallel implementation is designed using the Message Passing Interface (MPI) and Feel++ libraries. The objective of the parallel implementation is to minimize the amount of communications with respect to the parallel operations involved in the linear solver, namely matrix-vector products and dot products. One of interests of this mortar parallel implementation is that there's no communication at cross-points (in 2D and 3D) and cross-edges (in 3D), which reduces considerably communications between subdomains.

Assuming a constant number of internal dofs in each subdomain, it is rather straightforward to bind a subdomain to each process. Each process would own its subdomain mesh $\mathcal{T}_{h_\ell}$, functional space X_{h_ℓ}, stiffness matrix A_ℓ and unknown u_ℓ. Regarding the mortars, the choice is less obvious. In order to decrease the amount of communications in the matrix-vector products, we have used technique developed in [Abd+99] which consists in duplicating the data at the interfaces between subdomains. If $\Gamma_{\ell n}$ is such an interface, then the Lagrange multiplier vector $\lambda_{\ell n}$ and its associated trace mesh $\mathcal{T}_{h_\ell,n}$ and trace space $\mathcal{M}_{h_\ell,n}$ are stored in both the processors dealing Ω_ℓ and Ω_n. Although the data storage is increased a little bit, the communications will be reduced significantly.



7.1.1. One subdomain per cluster

7.1.2. Subdomains 3 and 4 on the same cluster

Figure 7.1 : Domain decompositions

As an example, consider the splitting of the unit square into four little squares, as in Figure 7.1, where the dash rectangles denote clusters and the bold segments correspond to the mortar interfaces. Note, that when neighboring subdomains belong to different clusters, there

are two copies of the mortar interface variables stored in different clusters. Consider the interfaces as shown in the picture. The matrix $\mathcal{A}$ has the following form :

$$\mathcal{A} = \left(\begin{array}{cccc|cccc} A_1 & & & & B_{15}^T & B_{16}^T & \mathbf{0} & \mathbf{0} \\ & A_2 & & & B_{25}^T & \mathbf{0} & B_{27}^T & \mathbf{0} \\ & & A_3 & & \mathbf{0} & B_{36}^T & \mathbf{0} & B_{38}^T \\ & & & A_4 & \mathbf{0} & \mathbf{0} & B_{47}^T & B_{48}^T \\ \hline B_{15} & B_{25} & \mathbf{0} & \mathbf{0} & & & & \\ B_{16} & \mathbf{0} & B_{36} & \mathbf{0} & & & & \\ \mathbf{0} & B_{27} & \mathbf{0} & B_{38} & & & & \\ \mathbf{0} & \mathbf{0} & B_{47} & B_{48} & & & & \end{array}\right) \tag{7.3}$$

Let us consider the matrix-vector multiplication procedure with the matrix $\mathcal{A}$ and the vector $(\mathbf{u}, \lambda)$, where $\mathbf{u}$ and λ have the following component-wise representation, according to the decomposition and the enumeration in FIGURE 7.1.1. : $\mathbf{u} = (u_1^T, u_2^T, u_3^T, u_4^T)^T$ and $\lambda = (\lambda_5^T, \lambda_6^T, \lambda_7^T, \lambda_8^T)^T$. The resulting vector $(\mathbf{v}, \mu) = \mathcal{A} \cdot (, \lambda)$ can be computed as

$$\left(\begin{array}{c} v_1^{(1)} \\ v_2^{(2)} \\ v_3^{(3)} \\ v_4^{(4)} \\ \hline \mu_5^{(1,2)} \\ \mu_6^{(1,3)} \\ \mu_7^{(2,3)} \\ \mu_8^{(3,4)} \end{array}\right) = \left(\begin{array}{c} A_1 u_1^{(1)} + B_{15}^T \lambda_5^{(1)} + B_{16}^T \lambda_6^{(1)} \\ A_2 u_2^{(2)} + B_{25}^T \lambda_5^{(2)} + B_{27}^T \lambda_7^{(2)} \\ A_3 u_3^{(3)} + B_{36}^T \lambda_6^{(3)} + B_{38}^T \lambda_8^{(3)} \\ A_4 u_4^{(4)} + B_{47}^T \lambda_7^{(4)} + B_{48}^T \lambda_8^{(4)} \\ \hline B_{15} u_1^{(1)} + B_{25} u_2^{(2)} \\ B_{16} u_1^{(1)} + B_{36} u_3^{(3)} \\ B_{27} u_2^{(2)} + B_{47} u_4^{(4)} \\ B_{38} u_3^{(3)} + B_{48} u_4^{(4)} \end{array}\right) \tag{7.4}$$

where the upper indices denote the cluster (the processor), in which this variable is stored. Two upper indices mean that this variable is stored in both processors. Note that $\lambda_i^{(\ell)} \equiv \lambda_i^{(n)}$ and so far we need communications only when computing μ_i. For example

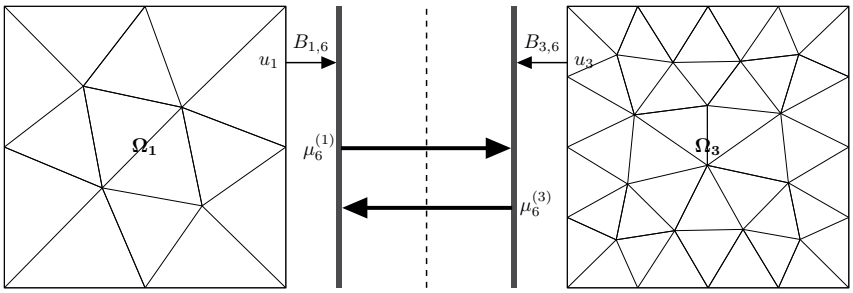


FIGURE 7.2 : Communications for jump matrix-vector multiplication

$$\mu_6 = \mu_6^{(1)} + \mu_6^{(3)}, \qquad \mu_6^{(1)} = B_{16}u_1 \text{ and } \mu_6^{(3)} = B_{36}u_3.$$

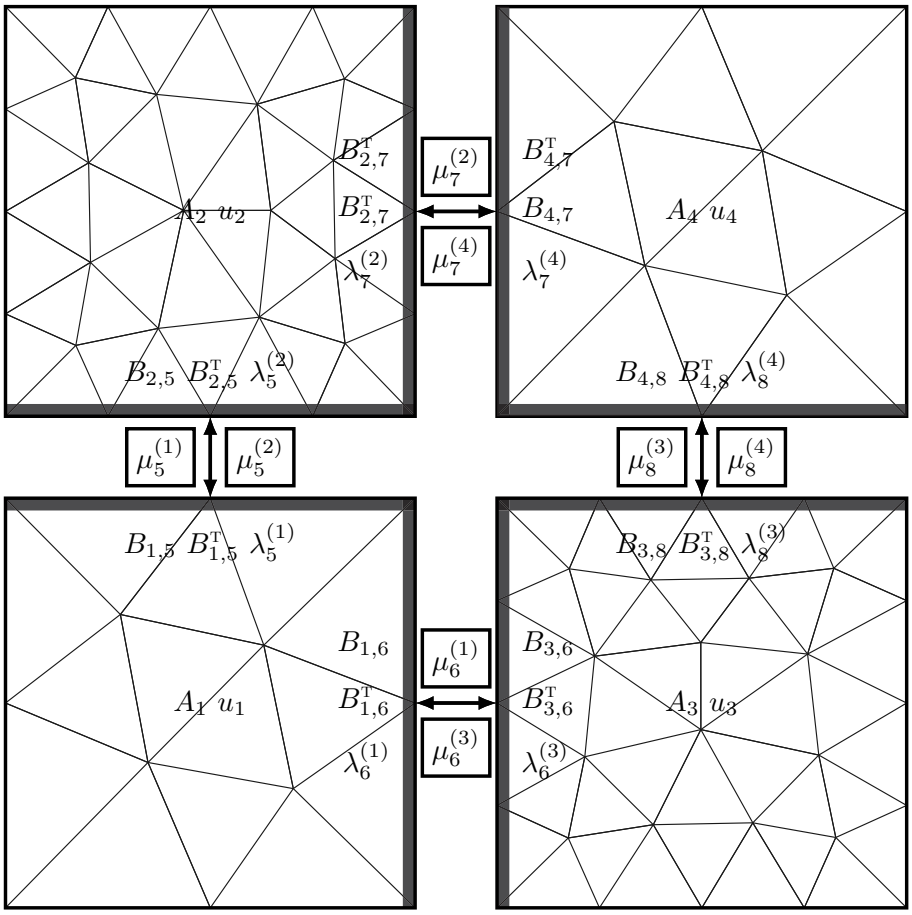


Figure 7.3 : Communications for parallel matrix-vector multiplication

We see that $\mu_6^{(1)}$ and $\mu_6^{(3)}$ are computed in parallel, and then should be interchanged and summed, see the representations in Figure 7.3 and more explicitly in Figure 7.2.

Algorithm 7.9. Parallel matrix-vector multiplication : $Y = \mathcal{A}X$

Require: X , Y `# input and output vectors`
  $u =$ subvect(X,mapDom) `# subdomain component of` X
  $v =$ subvect(Y,mapDom) `# subdomain component of` Y
  for $f \in$ interfaces **do**
    $\lambda_f =$ subvect(X,mapLag) `# multiplier component of` X
    $\mu_f =$ subvect(Y,mapLag) `# multiplier component of` Y
  end for
  $v \leftarrow Au$ `#` $v = Au$
  for $f \in$ interfaces **do**
    $v \leftarrow B_f^t \lambda_f$ `#` $v = v + B_f^t \lambda_f$
    $\mu_f \leftarrow B_f \lambda_f$ `#` $\mu_f = \mu_f + B_f \lambda_f$
  end for
  for $f \in$ interfaces **do**
    mpi_isend(neighbor(f),μ_f) `# send` μ_f `to neighbor` f
    mpi_irecv(neighbor(f),μ_{fr}) `# receve` μ_f `from neighbor` f
  end for
  mpi_waitall
  for $f \in$ interfaces **do**
    $\mu_f \leftarrow \mu_{fr}$ `#` $\mu_f = \mu_f + \mu_{fr}$
  end for
  return Y `# return output vector`

The Algorithm 7.9. represents the general case of the parallel matrix-vector multiplication for the saddle-point matrix $\mathcal{A}$ for the arbitrary number of subdomains.

7.4 Conclusion

In this chapter, we first discussed a Feel++ implementation framework for Schwarz methods including seamless and explicit MPI communication approach. We handled different variants of Schwarz methods namely additive and multiplicative algorithms and different artificial boundary conditions such as Dirichlet-Dirichlet, Dirichlet-Neumann, Neumann-Neumann and Robin-Robin. Then, we briefly interested in the assembly of jump matrices in the three-field formulation. Finally, we considered a special parallel implementation of mortar finite element method with Lagrange multipliers in 2D and 3D based on the duplication of data at the interfaces between subdomains in order to reduce the communications. We presented some Listings illustrating the flexibility of Feel++ for domain decomposition methods.

Part III

Numerical Experiments

Chapter 8

Substructuring Preconditioners in 2D

This chapter summarizes the numerical results for substructuring preconditioners for h-p mortar element method introduced in chapter 4. The numerical experiments for conforming and nonconforming domain decompositions are presented respectively in section 8.1 and 8.2. Some results obtained with large number of processor cores are given in section 8.3. The scalability analysis including strong and weak scalability is available in section 8.4.

Ce chapitre résume les résultats numériques pour les préconditionneurs par sous-structuration pour la méthode des éléments finis h-p mortar introduits dans le chapitre 4. Les expérimentations numériques pour les méthodes de décomposition de domaine conforme et non conforme sont présentées respectivement dans la section 8.1 et 8.2. Quelques résultats obtenus avec un très grand nombre de processeurs sont donnés dans la section 8.3. L'analyse de scalabilité comprenant la scalabilité forte et faible est disponible dans la section 8.4.

Contents

In this chapter, we analyze the properties of the preconditioners previously proposed in chapter 4 and investigated numerically in chapter 6. We perform a p-, H- and h-convergence study and the scalability analysis of our parallel algorithms. We consider the model problem

$$-\Delta u = f \quad \text{in } \Omega =]0,1[^2, \qquad u = 0 \quad \text{on } \partial\Omega \tag{8.1}$$

Unless otherwise stated, for all the following simulations we set $f = 1$. We consider the geometrically conforming domain decomposition and we split the domain Ω in $N = 4^\ell$ subdomains, $\ell > 1$, with simplex or hypercube elements.

The simulations were partly performed at MesoCentre@Strasbourg on hpc-login. MesoCentre is a supercomputer with 288 compute nodes interconnected by an infiniband QDR network. The system is Scientific Linux based on Intel Xeon Ivy Bridge processors with 16 cores and 64 GB of RAM running at 2 .6 Ghz. MesoCentre has a theoretical peak performance of 70 TFLOP/s. The simulations on a large number of cores, more than or equal to 1024, were done on Curie at the TGCC, a TIER-0 system which is part of PRACE. Curie has 5040 B510 bullx nodes and for each node a 2 Eight-Core Intel processors Sandy Bridge cadenced at 2.7 GHz with 64 GB.

We present the numerical results including the theoretical properties and the performance of the parallel implementation of substructuring preconditioners for h-p mortar element method proposed in chapter 4. the numerical tests relate the following three preconditioners $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$ for the transformed Schur complement system (4.41). All the tests presented relate to $\beta = 1/10$ and $\gamma = 2$. The relative tolerance of the Preconditioned Conjugate Gradient(PCG) solver is set to 10^{-6}.

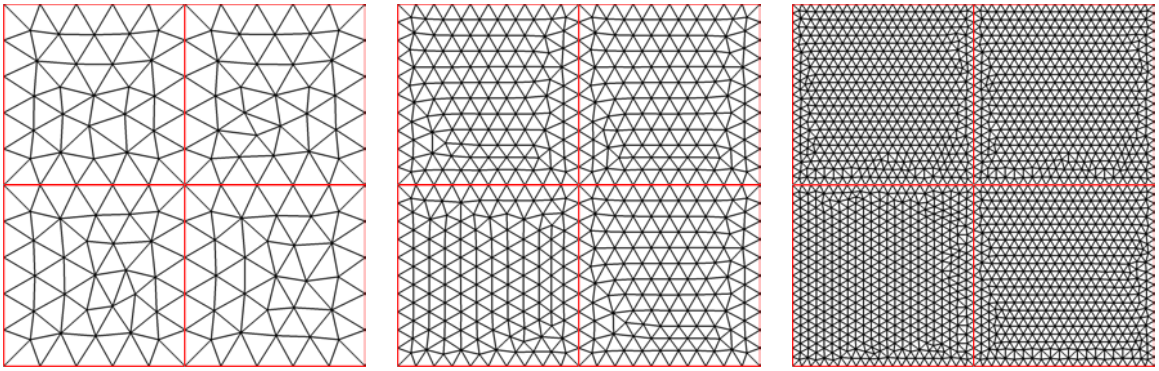
We report the condition number estimate of the preconditioned Schur complement matrix $\kappa(\widehat{\mathbf{P}}^{-1}\widehat{\mathbf{S}})$ where $\widehat{\mathbf{P}}$ is one of the preconditioners $\mathbf{P}_0$, $\mathbf{P}_1$ or $\mathbf{P}_2$, the number of iterations and the following two ratios :

$$R_2 = \frac{\kappa(\widehat{\mathbf{P}}^{-1}\widehat{\mathbf{S}})}{\left(1+\log\left(\frac{Hp^2}{h}\right)\right)^2} \qquad \text{and} \qquad R_{2p} = \frac{R_2}{p^{3/2}} \tag{8.2}$$

where H is the coarse mesh-size, h the fine mesh-size and p the polynomial order.

8.1 Conforming Domain Decompositions

We split the domain Ω in $N = 4^\ell$ subdomains, $\ell = 2\,,3\,,4\,$, with $n \times n$ mesh in each subdomain. These results were obtained on a sequence of triangular grids like the ones shown in FIGURE 8.1. The example in FIGURE 8.1 relate to the first three levels of refinements for unstructured triangular grids on a subdomain partition made of four squares.



8.1.1. Level 1 8.1.2. Level 2 8.1.3. Level 3

Figure 8.1 : Conforming domain decompositions with unstructured meshes

8.1.1 Linear Elements

In the first set of experiments, we consider the piecewise linear elements ($p = 1$), and we report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of iterations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.1.1.1 Unpreconditioned Schur Complement

We report in Table 3 and Table 4 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$.

Table 3 : Unpreconditioned Schur complement - number of iterations for $p = 1$

$N\backslash n$	5	10	20	40	80	160	320
16	44	59	84	105	155	240	354
64	59	77	109	150	213	298	468
256	81	99	127	178	250	327	484

Table 4 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 1$

$N\backslash n$	5	10	20	40	80	160	320
16	$6.63e{+}1$	$1.28e{+}2$	$3e{+}2$	$7.3e{+}2$	$1.78e{+}3$	$4.3e{+}3$	$1.02e{+}4$
64	$1.76e{+}2$	$2.14e{+}2$	$3.56e{+}2$	$8.29e{+}2$	$2.02e{+}3$	$4.37e{+}3$	$1.15e{+}4$
256	$6.49e{+}2$	$7.22e{+}2$	$8.02e{+}2$	$9.94e{+}2$	$2.09e{+}3$	$4.99e{+}3$	$1.19e{+}4$

In TABLE 3 and TABLE 4, we observe that the number of iterations and the condition number estimate increase significantly with the number of subdomains (number of processor cores) N and with $n = H/h$. These results support the requirement to use an efficient preconditioner for solving such a linear system, as explained in previous chapters.

8.1.1.2 Preconditioned Schur Complement

We report in TABLE 5 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

TABLE 5 : Ratio R_2 and number of iterations (between parenthesis) for $p = 1$

	$N\backslash n$	5	10	20	40	80	160	320
$\mathbf{P}_0$	16	3.33 (25)	2.50 (27)	2.04 (29)	1.72 (30)	1.52 (31)	1.37 (31)	1.27 (30)
	64	3.23 (25)	2.44 (27)	1.97 (28)	1.67 (29)	1.48 (29)	1.34 (30)	1.24 (31)
	256	3.05 (22)	2.34 (24)	1.91 (25)	1.62 (26)	1.45 (27)	1.26 (27)	1.15 (28)
$\mathbf{P}_1$	16	3.25 (26)	2.42 (27)	2.02 (28)	1.81 (31)	1.70 (33)	1.63 (34)	1.59 (36)
	64	3.16 (24)	2.39 (27)	2.01 (29)	1.77 (31)	1.67 (33)	1.59 (35)	1.56 (36)
	256	2.99 (21)	2.23 (23)	1.89 (25)	1.71 (28)	1.62 (30)	1.57 (33)	1.54 (35)
$\mathbf{P}_2$	16	3.59 (23)	2.57 (24)	2.16 (26)	1.86 (28)	1.65 (31)	1.51 (33)	1.41 (35)
	64	3.57 (22)	2.56 (23)	2.19 (26)	1.94 (29)	1.74 (31)	1.60 (33)	1.50 (35)
	256	3.53 (20)	2.55 (21)	2.22 (23)	1.96 (26)	1.76 (28)	1.63 (30)	1.52 (33)

As the theoretical estimates (4.18), (4.26) and (4.32), the TABLE 5 shows that for $n = H/h$ fixed, the ratio R_2 remains constant. The same behavior is observed for the number of iterations.

8.1.2 Second-order Elements

We consider the second-order elements ($p = 2$), and we report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of iterations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.1.2.1 Unpreconditioned Schur Complement

We report in TABLE 6 and TABLE 7 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$.

Table 6 : Unpreconditioned Schur complement - number of iterations for $p = 2$

$N\backslash n$	5	10	20	40	80	160
16	66	90	114	164	256	373
64	87	123	167	229	330	507
256	110	138	196	272	356	533

Table 7 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 2$

$N\backslash n$	5	10	20	40	80	160
16	$1.75e{+}2$	$3.96e{+}2$	$9.42e{+}2$	$2.25e{+}3$	$5.34e{+}3$	$1.25e{+}4$
64	$2.87e{+}2$	$4.73e{+}2$	$1.07e{+}3$	$2.55e{+}3$	$5.43e{+}3$	$1.42e{+}4$
256	$9.36e{+}2$	$1.01e{+}3$	$1.26e{+}3$	$2.65e{+}3$	$6.23e{+}3$	$1.46e{+}4$

In Table 6 and Table 7, we observe that the number of iterations and the condition number estimate increase significantly with the number of subdomains N and with $n = H/h$. The Table 7 shows that $\kappa(\widehat{\mathbf{S}})$ is growing faster than in the case of linear elements presented in section 8.1.1.1. These results support the requirement to use an efficient preconditioner for solving such a linear system.

8.1.2.2 Preconditioned Schur Complement

We report in Table 8 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

Table 8 : Ratio R_2 and number of iterations (between parenthesis) for $p = 2$

	$N\backslash n$	5	10	20	40	80	160
$\mathbf{P}_0$	16	1.51 (26)	1.32 (27)	1.20 (29)	1.12 (30)	1.06 (30)	1.02 (30)
	64	1.50 (25)	1.30 (27)	1.18 (27)	1.09 (29)	1.07 (31)	0.99 (32)
	256	1.43 (22)	1.26 (24)	1.15 (26)	1.06 (28)	1.03 (30)	0.98 (31)
$\mathbf{P}_1$	16	1.56 (27)	1.33 (27)	1.22 (28)	1.17 (30)	1.14 (31)	1.13 (32)
	64	1.55 (25)	1.33 (27)	1.20 (28)	1.13 (30)	1.11 (32)	1.10 (33)
	256	1.48 (22)	1.28 (24)	1.16 (25)	1.11 (29)	1.09 (31)	1.08 (30)
$\mathbf{P}_2$	16	1.47 (23)	1.33 (26)	1.23 (29)	1.18 (31)	1.14 (32)	1.12 (33)
	64	1.54 (22)	1.42 (25)	1.32 (28)	1.25 (31)	1.21 (33)	1.17 (35)
	256	1.57 (21)	1.44 (23)	1.34 (26)	1.27 (28)	1.23 (32)	1.19 (33)

In accordance with the theoretical estimates (4.18), (4.26) and (4.32), the TABLE 8 shows that for $n = H/h$ fixed, the ratio R_2 remains constant. We observe the same behavior for the number of iterations.

8.1.3 Third-order Elements

The third-order elements ($p = 3$) is considered in this experiments, and we report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of iterations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.1.3.1 Unpreconditioned Schur Complement

We report in TABLE 9 and TABLE 10 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$.

TABLE 9 : Unpreconditioned Schur complement - number of iterations for $p = 3$

$N\backslash n$	5	10	20	40	80	160
16	88	117	150	239	344	446
64	114	160	218	323	468	679
256	137	179	256	323	495	703

TABLE 10 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 3$

$N\backslash n$	5	10	20	40	80	160
16	$3.59e{+}2$	$8.21e{+}2$	$1.92e{+}3$	$4.48e{+}3$	$1.04e{+}4$	$2.41e{+}4$
64	$4.55e{+}2$	$9.41e{+}2$	$2.18e{+}3$	$5.08e{+}3$	$1.18e{+}4$	$2.73e{+}4$
256	$1.17e{+}3$	$1.32e{+}3$	$2.29e{+}3$	$5.24e{+}3$	$1.22e{+}4$	$2.77e{+}4$

In TABLE 9 and TABLE 10, we observe that the number of iterations and the condition number estimate increase significantly with the number of subdomains N and with $n = H/h$. The TABLE 9 shows that $\kappa(\widehat{\mathbf{S}})$ is growing faster than in the case of linear and second-order elements presented respectively in section 8.1.1.1 and in section 8.1.2.1. These results support the requirement to use an efficient preconditioner for solving such a linear system.

8.1.3.2 Preconditioned Schur Complement

We report in TABLE 11 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

Table 11 : Ratio R_2 and number of iterations (between parenthesis) for $p = 3$

	$N\backslash n$	5	10	20	40	80	160
$\mathbf{P}_0$	16	1.19 (27)	1.09 (28)	1.03 (29)	1.00 (31)	0.97 (31)	0.94 (32)
	64	1.17 (26)	1.07 (27)	1.01 (29)	0.96 (31)	0.97 (32)	0.91 (33)
	256	1.13 (23)	1.04 (25)	0.99 (28)	0.96 (30)	0.97 (32)	0.91 (32)
$\mathbf{P}_1$	16	1.21 (26)	1.10 (27)	1.07 (29)	1.04 (32)	1.03 (32)	1.02 (33)
	64	1.20 (27)	1.10 (28)	1.04 (30)	1.01 (32)	1.00 (34)	0.99 (34)
	256	1.16 (23)	1.06 (25)	1.01 (27)	0.99 (30)	0.99 (32)	0.98 (32)
$\mathbf{P}_2$	16	1.19 (26)	1.13 (28)	1.10 (30)	1.07 (33)	1.06 (33)	1.05 (35)
	64	1.27 (25)	1.20 (28)	1.16 (31)	1.13 (33)	1.11 (35)	1.09 (40)
	256	1.30 (23)	1.22 (25)	1.18 (28)	1.14 (31)	1.12 (34)	1.10 (36)

As the theoretical estimates (4.18), (4.26) and (4.32) and similarly to the results obtained with the linear elements ($p = 1$) and the second-order elements ($p = 2$) presented respectively in 8.1.1.2 and 8.1.2.2, the Table 11 shows that for $n = H/h$ fixed, the ratio R_2 remains constant. We observe the same behavior for the number of iterations.

8.1.4 Fourth-order Elements

The simulations are performed using the fourth-order elements ($p = 4$), and we report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of iterations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.1.4.1 Unpreconditioned Schur Complement

We report in Table 12 and Table 13 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$. These results support the use of preconditioners for an efficient solution of the Schur complement system.

Table 12 : Unpreconditioned Schur complement - number of iterations for $p = 4$

$N\backslash n$	5	10	20	40	80
16	109	135	198	308	444
64	147	196	284	400	613
256	173	233	316	432	631

Table 13 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 4$

$N\backslash n$	5	10	20	40	80
16	$6.47e{+2}$	$1.47e{+3}$	$3.38e{+3}$	$7.79e{+3}$	$1.78e{+4}$
64	$7.64e{+2}$	$1.67e{+3}$	$3.83e{+3}$	$7.91e{+3}$	$2.01e{+4}$
256	$1.51e{+3}$	$1.91e{+3}$	$3.97e{+3}$	$9.07e{+3}$	$2.07e{+4}$

8.1.4.2 Preconditioned Schur Complement

We report in Table 14 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

Table 14 : Ratio R_2 and number of iterations (between parenthesis) for $p = 4$

	$N\backslash n$	5	10	20	40	80
$\mathbf{P}_0$	16	1.06 (27)	1.00 (28)	0.96 (29)	0.94 (32)	0.93 (32)
	64	1.04 (27)	0.98 (28)	0.94 (30)	0.91 (32)	0.91 (33)
	256	1.02 (25)	0.96 (26)	0.93 (29)	0.91 (31)	0.93 (32)
$\mathbf{P}_1$	16	1.07 (27)	1.01 (29)	0.98 (30)	0.97 (32)	0.96 (33)
	64	1.06 (27)	0.99 (29)	0.95 (31)	0.94 (33)	0.92 (34)
	256	1.03 (24)	0.97 (26)	0.94 (28)	0.92 (31)	0.93 (33)
$\mathbf{P}_2$	16	1.09 (28)	1.07 (30)	1.05 (31)	1.04 (35)	1.03 (38)
	64	1.15 (28)	1.12 (30)	1.10 (32)	1.08 (35)	1.07 (40)
	256	1.18 (25)	1.13 (27)	1.11 (29)	1.09 (33)	1.08 (36)

Similarly to the results obtained with the linear elements ($p = 1$), the second-order elements ($p = 2$) and the third-order elements ($p = 3$) presented respectively in 8.1.1.2, 8.1.2.2 and 8.1.3.2, the Table 14 shows that for $n = H/h$ fixed, the ratio R_2 remains constant. The same behavior is valid for the number of iterations.

8.1.5 Fifth-order Elements

We consider the fifth-order elements ($p = 5$), and we report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of iterations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.1.5.1 Unpreconditioned Schur Complement

We report in TABLE 15 and TABLE 16 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$. These results motivate the need to use the preconditioning techniques for an efficient solution of such a linear system.

TABLE 15 : Unpreconditioned Schur complement - number of iterations for $p = 5$

$N\backslash n$	5	10	20	40	80
16	137	166	245	393	533
64	180	247	353	525	765
256	201	287	382	539	787

TABLE 16 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 5$

$N\backslash n$	5	10	20	40	80
16	$1.13e{+}3$	$2.54e{+}3$	$5.73e{+}3$	$1.29e{+}4$	$2.9e{+}4$
64	$1.3e{+}3$	$2.87e{+}3$	$6.47e{+}3$	$1.44e{+}4$	$3.28e{+}4$
256	$2.03e{+}3$	$3.07e{+}3$	$6.66e{+}3$	$1.5e{+}4$	$3.35e{+}4$

8.1.5.2 Preconditioned Schur Complement

We report in TABLE 17 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

TABLE 17 : Ratio R_2 and number of iterations (between parenthesis) for $p = 5$

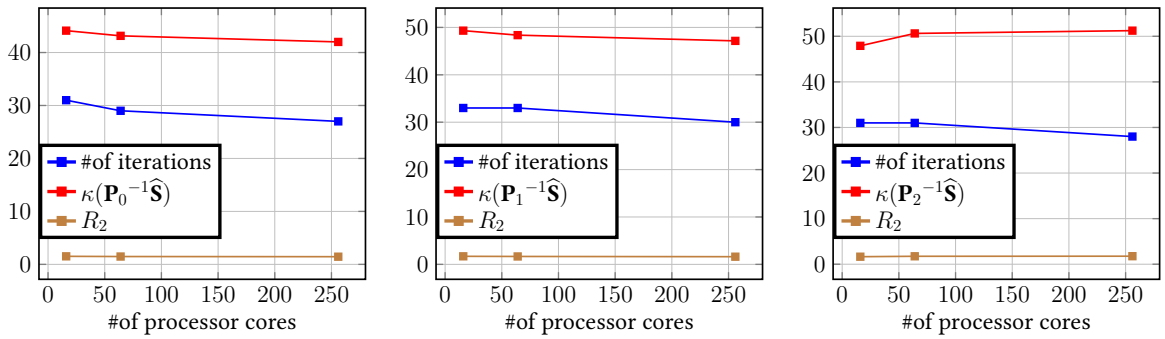
	$N\backslash n$	5	10	20	40	80
$\mathbf{P}_0$	16	0.99 (27)	0.95 (29)	0.92 (30)	0.91 (33)	0.91 (33)
	64	0.97 (28)	0.93 (30)	0.90 (31)	0.88 (33)	0.89 (34)
	256	0.95 (25)	0.92 (27)	0.89 (29)	0.87 (31)	0.90 (33)
$\mathbf{P}_1$	16	0.99 (28)	0.96 (29)	0.94 (30)	0.94 (33)	0.93 (34)
	64	0.98 (28)	0.94 (31)	0.92 (32)	0.91 (34)	0.90 (34)
	256	0.95 (25)	0.92 (27)	0.90 (29)	0.90 (32)	0.90 (33)
$\mathbf{P}_2$	16	1.06 (30)	1.04 (31)	1.03 (32)	1.02 (38)	1.02 (39)
	64	1.11 (29)	1.09 (31)	1.08 (34)	1.07 (40)	1.07 (42)
	256	1.13 (26)	1.10 (29)	1.08 (33)	1.07 (35)	1.06 (40)

As the theoretical estimates (4.18), (4.26) and (4.32), the TABLE 17 clearly indicates that for $n = H/h$ fixed, the ratio R_2 remains constant. The same properties are observed for the number of iterations.

8.1.6 Dependence on Number of Subdomains

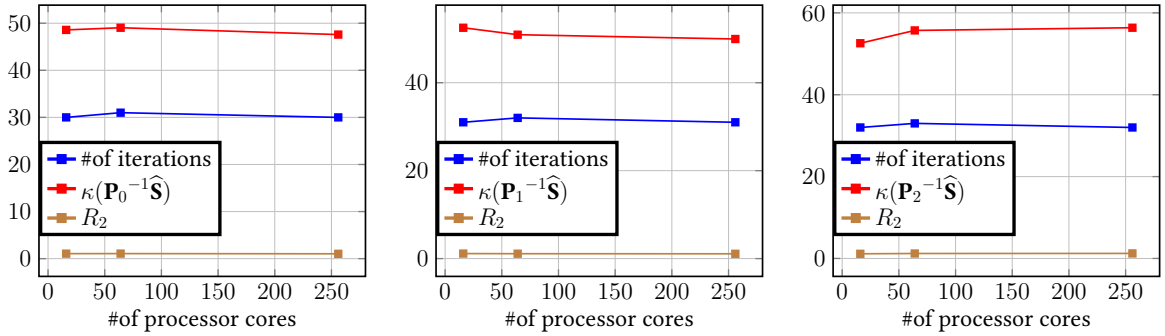
To analyze the dependence of the substructuring preconditioners in coarse mesh-size H, we plot in figures FIGURE 8.2 to FIGURE 8.6 the number of iterations required for solving the linear system (4.41) preconditioned by $\widehat{\mathbf{P}} \in \{\mathbf{P}_0\,, \mathbf{P}_1\,, \mathbf{P}_2\}$, the condition number $\kappa(\widehat{\mathbf{P}}^{-1}\widehat{\mathbf{S}})$ and the ratio R_2 as a function of number of subdomains (number of processor cores) N for $H/h = 80$. The simulations are performed with linear elements ($p = 1$) and high-order elements ($2 \leq p \leq 5$).

In accordance with the theoretical estimates (4.18), (4.26) and (4.32), for each polynomial order $p = 1\,, 2\,, 3\,, 4\,, 5$, a logarithmic growth is clearly observed for all the preconditioners $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$. These results remain valid for other values of H/h as reported in tables TABLE 5, TABLE 8, TABLE 11, TABLE 14 and TABLE 17.



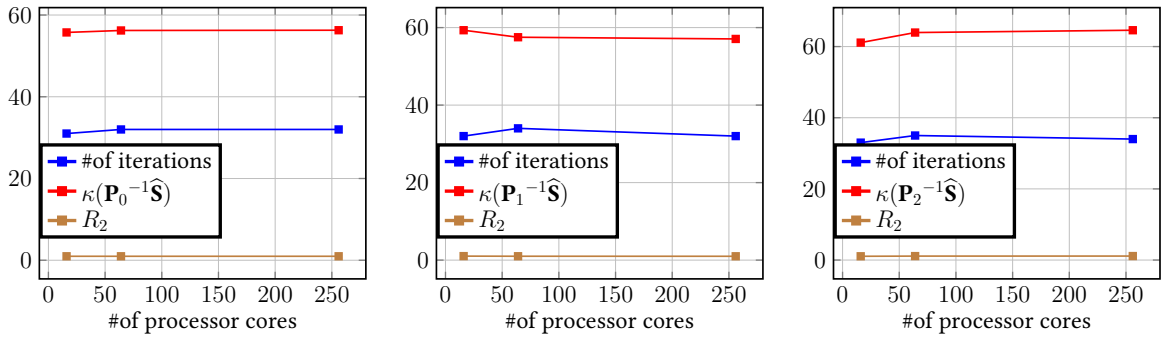
8.2.1. Preconditioner $\mathbf{P}_0$ 8.2.2. Preconditioner $\mathbf{P}_1$ 8.2.3. Preconditioner $\mathbf{P}_2$

FIGURE 8.2 : Behavior in number of subdomains for $p = 1$ and $H/h = 80$



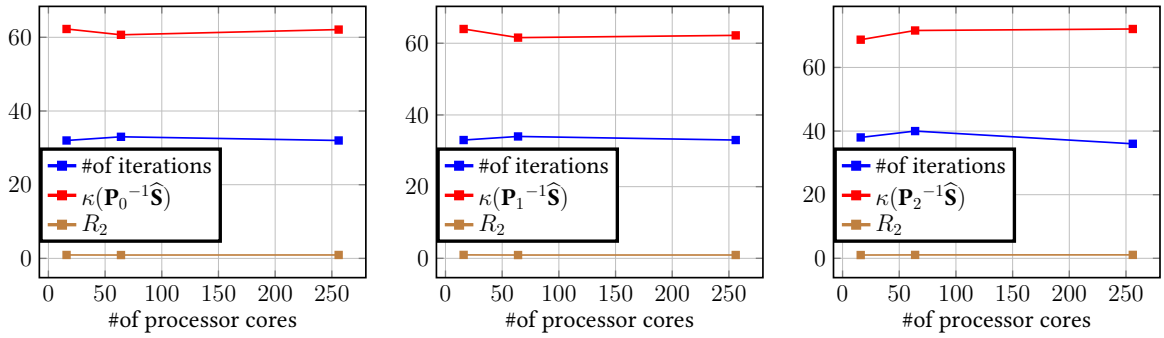
8.3.1. Preconditioner $\mathbf{P}_0$ 8.3.2. Preconditioner $\mathbf{P}_1$ 8.3.3. Preconditioner $\mathbf{P}_2$

FIGURE 8.3 : Behavior in number of subdomains for $p = 2$ and $H/h = 80$



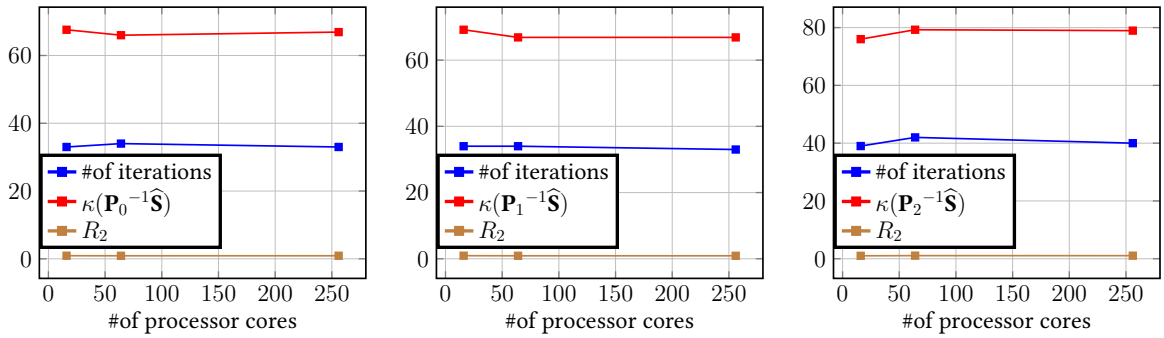
8.4.1. Preconditioner $\mathbf{P}_0$ 8.4.2. Preconditioner $\mathbf{P}_1$ 8.4.3. Preconditioner $\mathbf{P}_2$

Figure 8.4 : Behavior in number of subdomains for $p = 3$ and $H/h = 80$



8.5.1. Preconditioner $\mathbf{P}_0$ 8.5.2. Preconditioner $\mathbf{P}_1$ 8.5.3. Preconditioner $\mathbf{P}_2$

Figure 8.5 : Behavior in number of subdomains for $p = 4$ and $H/h = 80$



8.6.1. Preconditioner $\mathbf{P}_0$ 8.6.2. Preconditioner $\mathbf{P}_1$ 8.6.3. Preconditioner $\mathbf{P}_2$

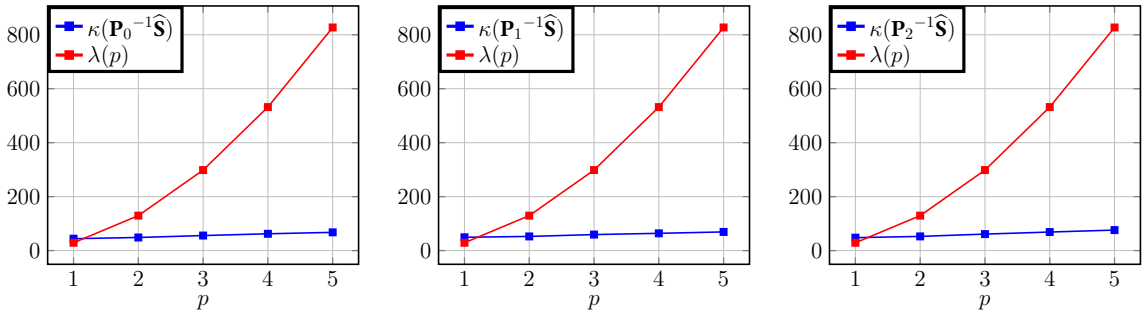
Figure 8.6 : Behavior in number of subdomains for $p = 5$ and $H/h = 80$

8.1.7 Dependence on Polynomial Order

To study the dependence on p, we report the condition number estimate of the preconditioned system as a function of p with H/h constant. Let the function λ be defined as

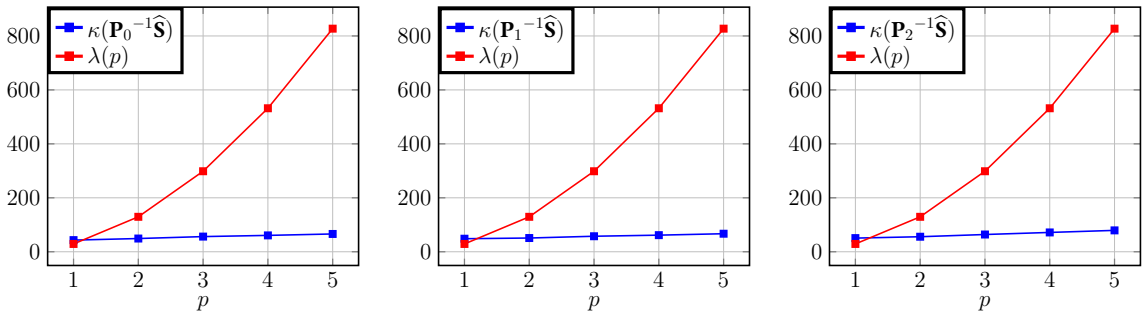
$$\lambda(p) = p^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2.$$

In FIGURE 8.7, FIGURE 8.8 and FIGURE 8.9, we plot the condition number of the transformed Schur system preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.



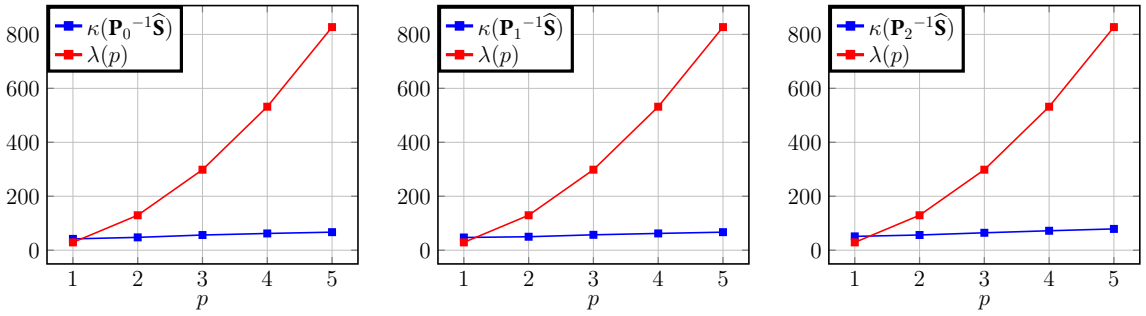
8.7.1. Preconditioner $\mathbf{P}_0$ 8.7.2. Preconditioner $\mathbf{P}_1$ 8.7.3. Preconditioner $\mathbf{P}_2$

FIGURE 8.7 : $\kappa(\widehat{\mathbf{P}}^{-1}\widehat{\mathbf{S}})$ as a function of p with #of subdomains=16 and $H/h = 80$



8.8.1. Preconditioner $\mathbf{P}_0$ 8.8.2. Preconditioner $\mathbf{P}_1$ 8.8.3. Preconditioner $\mathbf{P}_2$

FIGURE 8.8 : $\kappa(\widehat{\mathbf{P}}^{-1}\widehat{\mathbf{S}})$ as a function of p with #of subdomains=64 and $H/h = 80$



8.9.1. Preconditioner $\mathbf{P}_0$ 8.9.2. Preconditioner $\mathbf{P}_1$ 8.9.3. Preconditioner $\mathbf{P}_2$

Figure 8.9 : $\kappa(\widehat{\mathbf{P}}^{-1}\mathbf{S})$ as a function of p with #of subdomains=256 and $H/h = 80$

The Figure 8.7, Figure 8.8 and Figure 8.9 show that, for increasing values of p, our preconditioners behaves similarly to the linear case $p = 1$. To highlight the dependence on p of our preconditioners, we report in Table 18 the ratio R_2 for $H/h = 80$ fixed and increasing values of the polynomial order p. We clearly do not see the factor $\hat{p}^{3/2}$ which appears in the theoretical estimates (4.18), (4.26) and (4.32) (which, we recall, stems the mortar projector operator) since for fixed H, the ratio R_2 does not depend on this factor as shown Table 18. Indeed, the numerical results seem to show an even better behavior than the polylogarithmic dependence on Hp^2/h.

Table 18 : Ratio R_2 and number of iterations (between parenthesis) for $H/h = 80$

	$N\backslash p$	1	2	3	4	5
$\mathbf{P}_0$	16	1.52 (31)	1.06 (30)	0.97 (31)	0.93 (32)	0.91 (33)
	64	1.48 (29)	1.07 (31)	0.97 (32)	0.91 (33)	0.89 (34)
	256	1.45 (27)	1.03 (30)	0.97 (32)	0.93 (32)	0.90 (33)
$\mathbf{P}_1$	16	1.70 (33)	1.14 (31)	1.03 (32)	0.96 (33)	0.93 (34)
	64	1.67 (33)	1.11 (32)	1.00 (34)	0.92 (34)	0.90 (34)
	256	1.62 (30)	1.09 (31)	0.99 (32)	0.93 (33)	0.90 (33)
$\mathbf{P}_2$	16	1.65 (31)	1.14 (32)	1.06 (33)	1.03 (38)	1.02 (39)
	64	1.74 (31)	1.21 (33)	1.11 (35)	1.07 (40)	1.07 (42)
	256	1.76 (28)	1.23 (32)	1.12 (34)	1.08 (36)	1.06 (40)

Table 19 : $\kappa(\widehat{\mathbf{S}})$ and number of iterations (between parenthesis) for $H/h = 80$

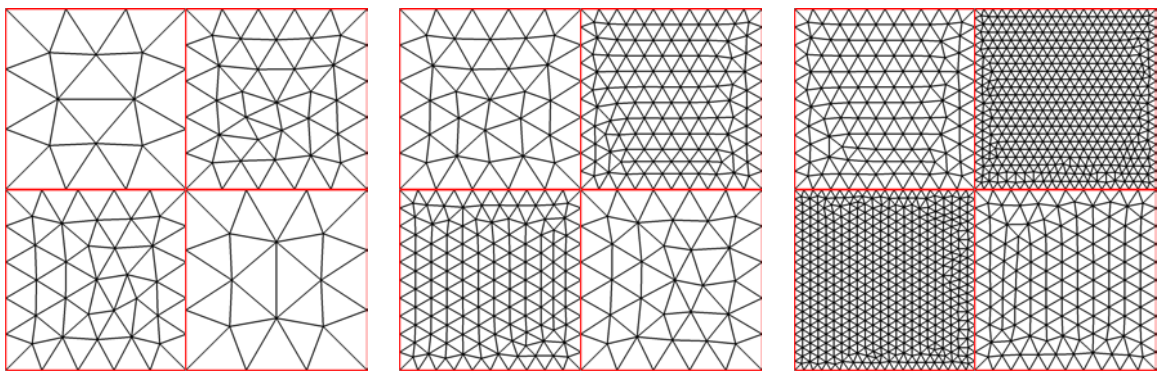
$N\backslash p$	1	2	3	4	5
16	1.78e+3 (155)	5.34e+3 (256)	1.04e+4 (344)	1.78e+4 (444)	2.9e+4 (533)
64	2.02e+3 (213)	5.43e+3 (330)	1.18e+4 (468)	2.01e+4 (613)	3.28e+4 (765)
256	2.09e+3 (250)	6.23e+3 (356)	1.22e+4 (495)	2.07e+4 (631)	3.35e+4 (787)

8.1.8 Conclusion

The numerical results presented in this section dedicated to the confirming domain decompositions support the mathematical properties including the p-, H- and h-convergence of substructuring preconditioners for h-p mortar finite element method. These results hold for linear elements ($p = 1$) and high order elements ($2 \leq p \leq 5$).

8.2 Nonconforming Domain Decompositions

The tests performed until now deal with decomposition with matching grid (though the solution is non conforming, due to the lack of continuity at the cross points). We now turn to the numerical results for nonconforming decompositions. As in section 8.1, we split the domain Ω in $N = 4^\ell$, $\ell = 2\,, 3\,, 4$ but now we take quasiuniform meshes with two different mesh sizes : $h_{\text{fine}} = 1/(2n)$ and $h_{\text{coarse}} = 1/n$. We deliberately choose embedded grids (see Figure 8.10) in order to ensure exact numerical integration for the constraints. On the interface, the master subdomains are chosen to be the ones corresponding to the coarser mesh.



8.10.1. Level 1 8.10.2. Level 2 8.10.3. Level 3

Figure 8.10 : Nonconforming domain decompositions with unstructured meshes

8.2.1 Linear Elements

We start as in section 8.1 with the linear finite elements ($p = 1$). We report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of

iterations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.2.1.1 Unpreconditioned Schur Complement

We report in TABLE 20 and TABLE 21 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$.

TABLE 20 : Unpreconditioned Schur complement - number of iterations for $p = 1$

$N\backslash n$	5	10	20	40	80	160	320
16	29	33	47	64	86	116	163
64	43	53	73	104	143	187	290
256	67	79	97	123	171	229	306

TABLE 21 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 1$

$N\backslash n$	5	10	20	40	80	160	320
16	$4.57e{+}1$	$6.43e{+}1$	$1.21e{+}2$	$2.87e{+}2$	$7.24e{+}2$	$1.82e{+}3$	$4.51e{+}3$
64	$1.41e{+}2$	$1.71e{+}2$	$2.16e{+}2$	$3.53e{+}2$	$8.28e{+}2$	$2.07e{+}3$	$4.75e{+}3$
256	$5.28e{+}2$	$6.23e{+}2$	$7.23e{+}2$	$8.35e{+}2$	$1.04e{+}3$	$2.16e{+}3$	$5.27e{+}3$

As for the conforming domain decompositions with linear elements presented in section 8.1, we observe that the number of iterations and the condition number estimate increase significantly with the number of subdomains (number of processing units) N and with $n = H/h$. These results support the requirement to use an efficient preconditioner for solving such a linear system, as explained in previous chapters.

8.2.1.2 Preconditioned Schur Complement

We report in TABLE 22 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

Table 22 : Ratio R_2 and number of iterations (between parenthesis) for $p = 1$

	$N\backslash n$	5	10	20	40	80	160	320
$\mathbf{P}_0$	16	1.10 (16)	0.80 (17)	0.77 (19)	0.75 (20)	0.74 (20)	0.73 (22)	0.72 (23)
	64	1.11 (15)	0.82 (17)	0.78 (18)	0.76 (20)	0.74 (21)	0.72 (23)	0.70 (24)
	256	1.05 (14)	0.80 (15)	0.76 (16)	0.73 (17)	0.71 (17)	0.70 (19)	0.69 (20)
$\mathbf{P}_1$	16	0.95 (15)	0.83 (18)	0.81 (20)	0.80 (22)	0.82 (22)	0.84 (24)	0.88 (24)
	64	1.00 (15)	0.83 (17)	0.81 (20)	0.80 (22)	0.82 (25)	0.85 (27)	0.87 (29)
	256	0.97 (13)	0.80 (15)	0.77 (17)	0.77 (20)	0.77 (21)	0.80 (22)	0.83 (23)
$\mathbf{P}_2$	16	1.46 (15)	0.87 (16)	0.73 (17)	0.73 (20)	0.74 (22)	0.75 (23)	0.76 (28)
	64	1.47 (16)	0.88 (16)	0.74 (17)	0.75 (20)	0.76 (22)	0.79 (25)	0.80 (29)
	256	1.43 (15)	0.87 (14)	0.71 (16)	0.76 (19)	0.77 (21)	0.79 (23)	0.80 (25)

As the theoretical estimates (4.18), (4.26) and (4.32), the Table 22 shows that for $n = H/h$ fixed, the ratio R_2 remains constant. The same behavior is observed for the number of iterations.

8.2.2 Second-order Elements

We consider the second-order elements ($p = 2$), and we report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of iterations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.2.2.1 Unpreconditioned Schur Complement

We report in Table 23 and Table 24 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$.

Table 23 : Unpreconditioned Schur complement - number of iterations for $p = 2$

$N\backslash n$	5	10	20	40	80	160
16	36	49	69	96	132	181
64	63	82	114	159	210	317
256	89	104	137	188	257	335

Table 24 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 2$

$N\backslash n$	5	10	20	40	80	160
16	$9.07e{+}1$	$1.56e{+}2$	$3.62e{+}2$	$8.89e{+}2$	$2.19e{+}3$	$5.33e{+}3$
64	$2.1e{+}2$	$2.6e{+}2$	$4.38e{+}2$	$1.02e{+}3$	$2.49e{+}3$	$5.57e{+}3$
256	$7.43e{+}2$	$8.43e{+}2$	$9.57e{+}2$	$1.22e{+}3$	$2.59e{+}3$	$6.23e{+}3$

In Table 23 and Table 24, we observe that the number of iterations and the condition number estimate increase significantly with the number of subdomains N and with $n = H/h$. The Table 7 shows that $\kappa(\widehat{\mathbf{S}})$ is growing faster than in the case of linear elements presented in section 8.2.1.1. These results support the requirement to use an efficient preconditioner for solving such a linear system.

8.2.2.2 Preconditioned Schur Complement

We report in Table 25 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

Table 25 : Ratio R_2 and number of iterations (between parenthesis) for $p = 2$

	$N\backslash n$	5	10	20	40	80	160
$\mathbf{P}_0$	16	0.70 (18)	0.65 (19)	0.65 (20)	0.65 (21)	0.65 (22)	0.65 (23)
	64	0.72 (18)	0.66 (19)	0.65 (20)	0.65 (22)	0.64 (24)	0.64 (26)
	256	0.70 (15)	0.63 (16)	0.63 (17)	0.62 (18)	0.63 (20)	0.63 (21)
$\mathbf{P}_1$	16	0.70 (19)	0.65 (21)	0.65 (22)	0.65 (22)	0.65 (24)	0.66 (25)
	64	0.71 (18)	0.66 (21)	0.65 (22)	0.65 (24)	0.65 (26)	0.65 (28)
	256	0.69 (16)	0.63 (18)	0.63 (19)	0.63 (20)	0.63 (21)	0.63 (22)
$\mathbf{P}_2$	16	0.64 (17)	0.62 (19)	0.65 (21)	0.67 (23)	0.70 (27)	0.73 (29)
	64	0.68 (17)	0.65 (19)	0.68 (22)	0.70 (25)	0.73 (28)	0.76 (30)
	256	0.67 (16)	0.65 (18)	0.68 (20)	0.70 (22)	0.72 (25)	0.74 (29)

In accordance with the theoretical estimates (4.18), (4.26) and (4.32), the Table 25 shows that for $n = H/h$ fixed, the ratio R_2 remains constant. We observe the same behavior for the number of iterations.

8.2.3 Third-order Elements

The third-order elements ($p = 3$) is considered in this experiments, and we report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of it-

erations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.2.3.1 Unpreconditioned Schur Complement

We report in TABLE 9 and TABLE 10 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$.

TABLE 26 : Unpreconditioned Schur complement - number of iterations for $p = 3$

$N\backslash n$	5	10	20	40	80	160
16	51	63	87	118	168	236
64	86	107	144	203	291	415
256	113	130	172	236	316	455

TABLE 27 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 3$

$N\backslash n$	5	10	20	40	80	160
16	$1.7e{+}2$	$3.03e{+}2$	$7.2e{+}2$	$1.74e{+}3$	$4.21e{+}3$	$1e{+}4$
64	$2.87e{+}2$	$3.91e{+}2$	$8.29e{+}2$	$1.98e{+}3$	$4.76e{+}3$	$1.13e{+}4$
256	$9.32e{+}2$	$1.01e{+}3$	$1.19e{+}3$	$2.09e{+}3$	$4.91e{+}3$	$1.17e{+}4$

In TABLE 9 and TABLE 10, we observe that the number of iterations and the condition number estimate increase significantly with the number of subdomains N and with $n = H/h$. The TABLE 9 shows that $\kappa(\widehat{\mathbf{S}})$ is growing faster than in the case of linear and second-order elements presented respectively in section 8.1.1.1 and in section 8.1.2.1. These results support the requirement to use an efficient preconditioner for solving such a linear system.

8.2.3.2 Preconditioned Schur Complement

We report in TABLE 28 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

Table 28 : Ratio R_2 and number of iterations (between parenthesis) for $p = 3$

	$N\backslash n$	5	10	20	40	80	160
$\mathbf{P}_0$	16	0.76 (22)	0.62 (19)	0.62 (20)	0.62 (22)	0.63 (23)	0.63 (24)
	64	0.77 (21)	0.62 (20)	0.62 (23)	0.62 (25)	0.62 (27)	0.62 (29)
	256	0.74 (19)	0.59 (18)	0.59 (18)	0.60 (20)	0.60 (21)	0.61 (22)
$\mathbf{P}_1$	16	0.76 (22)	0.62 (21)	0.62 (22)	0.62 (24)	0.63 (24)	0.63 (24)
	64	0.77 (23)	0.63 (22)	0.62 (23)	0.62 (25)	0.63 (27)	0.62 (28)
	256	0.74 (19)	0.59 (18)	0.59 (19)	0.60 (20)	0.60 (21)	0.61 (22)
$\mathbf{P}_2$	16	0.71 (22)	0.62 (19)	0.64 (24)	0.68 (27)	0.71 (28)	0.73 (29)
	64	0.73 (21)	0.64 (22)	0.68 (25)	0.72 (28)	0.75 (30)	0.77 (32)
	256	0.74 (19)	0.65 (19)	0.68 (23)	0.70 (26)	0.74 (30)	0.76 (30)

As the theoretical estimates (4.18), (4.26) and (4.32) and similarly to the results obtained with the linear elements ($p = 1$) and the second-order elements ($p = 2$) presented respectively in 8.2.1.2 and 8.2.2.2, the Table 28 shows that for $n = H/h$ fixed, the ratio R_2 remains constant. We observe the same behavior for the number of iterations.

8.2.4 Fourth-order Elements

The simulations are performed using the fourth-order elements ($p = 4$), and we report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of iterations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.2.4.1 Unpreconditioned Schur Complement

We report in Table 29 and Table 30 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$. These results support the use of preconditioners for an efficient solution of the Schur complement system.

Table 29 : Unpreconditioned Schur complement - number of iterations for $p = 4$

$N\backslash n$	5	10	20	40	80
16	69	78	108	149	215
64	104	131	181	241	356
256	127	154	208	287	389

Table 30 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 4$

$N\backslash n$	5	10	20	40	80
16	$2.74e{+}2$	$5.12e{+}2$	$1.21e{+}3$	$2.87e{+}3$	$6.82e{+}3$
64	$3.82e{+}2$	$6.1e{+}2$	$1.37e{+}3$	$3.25e{+}3$	$7.14e{+}3$
256	$1.09e{+}3$	$1.21e{+}3$	$1.59e{+}3$	$3.38e{+}3$	$7.94e{+}3$

8.2.4.2 Preconditioned Schur Complement

We report in Table 31 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

Table 31 : Ratio R_2 and number of iterations (between parenthesis) for $p = 4$

	$N\backslash n$	5	10	20	40	80
	16	0.64 (19)	0.60 (20)	0.61 (22)	0.61 (23)	0.62 (24)
$\mathbf{P}_0$	64	0.65 (21)	0.61 (22)	0.62 (25)	0.61 (26)	0.61 (29)
	256	0.63 (19)	0.59 (20)	0.58 (20)	0.59 (21)	0.60 (24)
	16	0.64 (21)	0.60 (22)	0.61 (23)	0.62 (24)	0.62 (24)
$\mathbf{P}_1$	64	0.65 (22)	0.61 (23)	0.61 (25)	0.61 (27)	0.62 (29)
	256	0.62 (18)	0.58 (19)	0.58 (20)	0.59 (22)	0.60 (22)
	16	0.66 (21)	0.64 (23)	0.68 (25)	0.70 (28)	0.73 (28)
$\mathbf{P}_2$	64	0.68 (22)	0.67 (24)	0.71 (26)	0.74 (29)	0.76 (31)
	256	0.69 (21)	0.67 (22)	0.69 (25)	0.72 (28)	0.76 (31)

Similarly to the results obtained with the linear elements ($p = 1$), the second-order elements ($p = 2$) and the third-order elements ($p = 3$) presented respectively in 8.2.1.2, 8.2.2.2 and 8.2.3.2, the Table 31 shows that for $n = H/h$ fixed, the ratio R_2 remains constant. The same behavior is valid for the number of iterations.

8.2.5 Fifth-order Elements

We consider the fifth-order elements ($p = 5$), and we report the condition number estimate of the (preconditioned) Schur complement matrix $\kappa(\mathbf{P}^{-1}\widehat{\mathbf{S}})$, the number of iterations required by the PCG solver and the ratio R_2 when varying the number of subdomains N and the number of elements n of the fine mesh.

8.2.5.1 Unpreconditioned Schur Complement

We report in TABLE 32 and TABLE 33 respectively the number of iterations required for solving the transformed Schur complement system (4.41) and the condition number estimate $\kappa(\widehat{\mathbf{S}})$. These results motivate the need to use the preconditioning techniques for an efficient solution of such a linear system.

TABLE 32 : Unpreconditioned Schur complement - number of iterations for $p = 5$

$N\backslash n$	5	10	20	40	80
16	79	95	135	190	268
64	124	156	215	305	445
256	151	177	245	337	475

TABLE 33 : Unpreconditioned Schur complement - $\kappa(\widehat{\mathbf{S}})$ for $p = 5$

$N\backslash n$	5	10	20	40	80
16	$4.39e{+}2$	$8.11e{+}2$	$1.87e{+}3$	$4.35e{+}3$	$1.01e{+}4$
64	$5.5e{+}2$	$9.38e{+}2$	$2.12e{+}3$	$4.92e{+}3$	$1.12e{+}4$
256	$1.33e{+}3$	$1.48e{+}3$	$2.28e{+}3$	$5.08e{+}3$	$1.17e{+}4$

8.2.5.2 Preconditioned Schur Complement

We report in TABLE 34 the ratio R_2 and the number of iterations required for solving the transformed Schur complement system (4.41) preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.

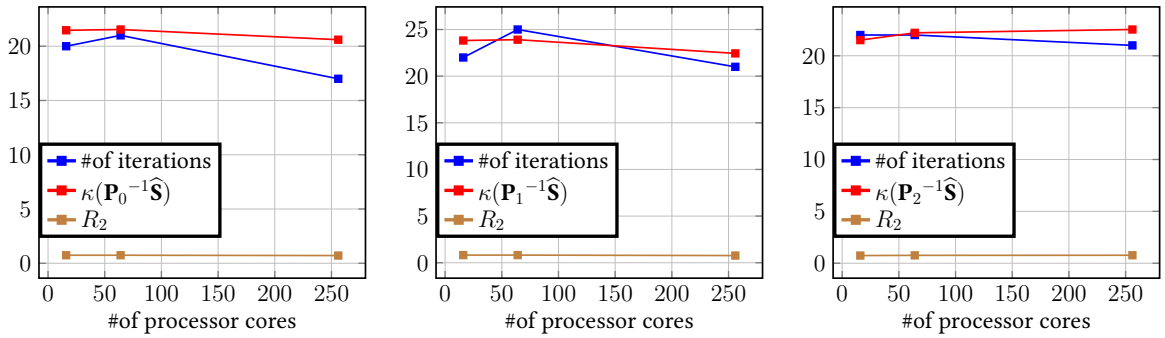
TABLE 34 : Ratio R_2 and number of iterations (between parenthesis) for $p = 5$

	$N\backslash n$	5	10	20	40	80
$\mathbf{P}_0$	16	0.63 (19)	0.60 (20)	0.60 (22)	0.61 (23)	0.61 (24)
	64	0.64 (21)	0.60 (22)	0.61 (25)	0.61 (26)	0.61 (29)
	256	0.62 (19)	0.59 (20)	0.58 (20)	0.59 (21)	0.60 (24)
$\mathbf{P}_1$	16	0.63 (22)	0.60 (23)	0.60 (23)	0.61 (25)	0.62 (26)
	64	0.64 (23)	0.61 (24)	0.61 (26)	0.61 (28)	0.61 (29)
	256	0.61 (19)	0.57 (20)	0.58 (21)	0.59 (22)	0.59 (23)
$\mathbf{P}_2$	16	0.67 (23)	0.66 (24)	0.69 (26)	0.71 (29)	0.74 (28)
	64	0.71 (24)	0.70 (25)	0.73 (28)	0.75 (30)	0.77 (32)
	256	0.71 (22)	0.68 (24)	0.72 (26)	0.75 (29)	0.78 (31)

As the theoretical estimates (4.18), (4.26) and (4.32), the TABLE 34 shows that for $n = H/h$ fixed, the ratio R_2 remains constant. The same properties are observed for the number of iterations.

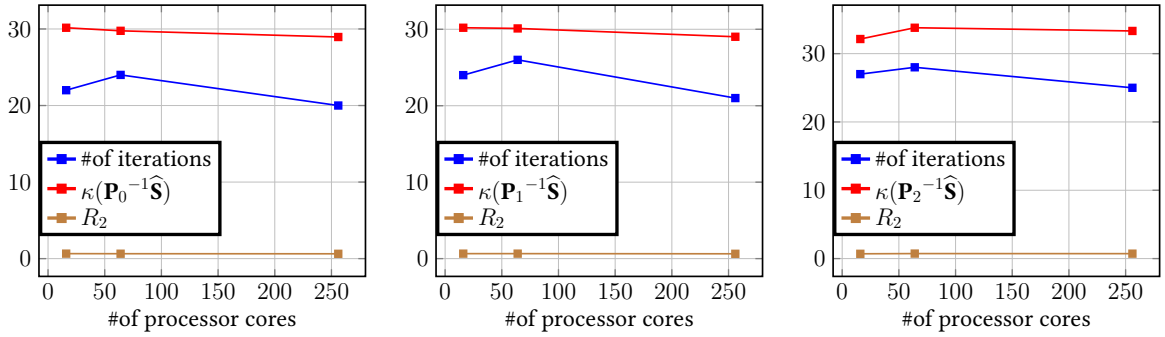
8.2.6 Dependence on Number of Subdomains

To analyze the dependence of the substructuring preconditioners in coarse mesh-size H, we plot in figures FIGURE 8.11 to FIGURE 8.15 the number of iterations required for solving the linear system (4.41) preconditioned by $\widehat{\mathbf{P}} \in \{\mathbf{P}_0\,, \mathbf{P}_1\,, \mathbf{P}_2\}$, the condition number $\kappa(\widehat{\mathbf{P}}^{-1}\widehat{\mathbf{S}})$ and the ratio R_2 as a function of number of subdomains (number of processor cores) N for $H/h = 80$. The simulations are performed with linear elements ($p = 1$) and high-order elements ($2 \leq p \leq 5$).



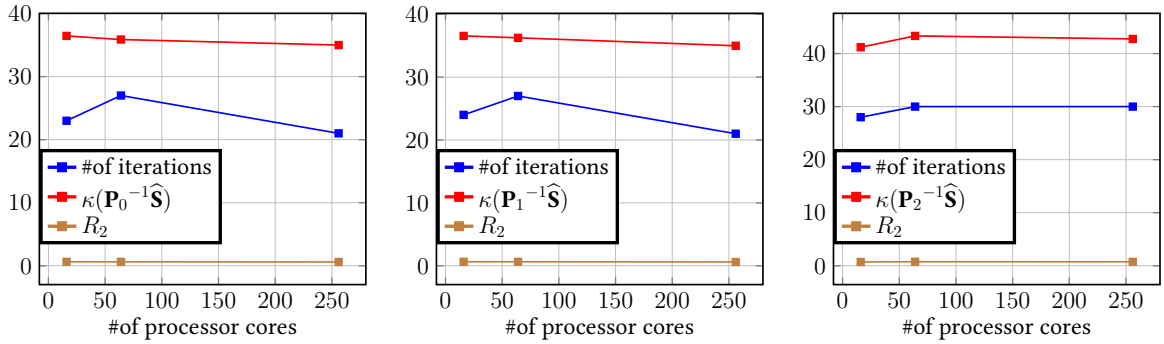
8.11.1. Preconditioner $\mathbf{P}_0$ 8.11.2. Preconditioner $\mathbf{P}_1$ 8.11.3. Preconditioner $\mathbf{P}_2$

FIGURE 8.11 : Behavior in number of subdomains for $p = 1$ and $H/h = 80$



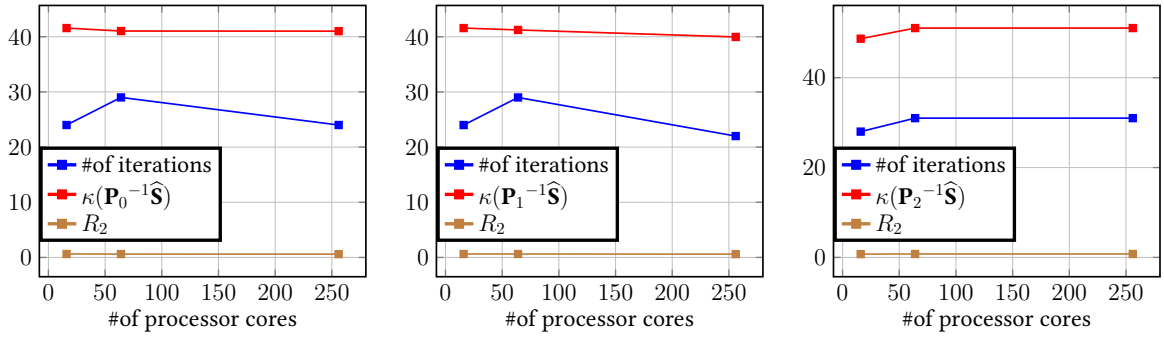
8.12.1. Preconditioner $\mathbf{P}_0$ 8.12.2. Preconditioner $\mathbf{P}_1$ 8.12.3. Preconditioner $\mathbf{P}_2$

FIGURE 8.12 : Behavior in number of subdomains for $p = 2$ and $H/h = 80$



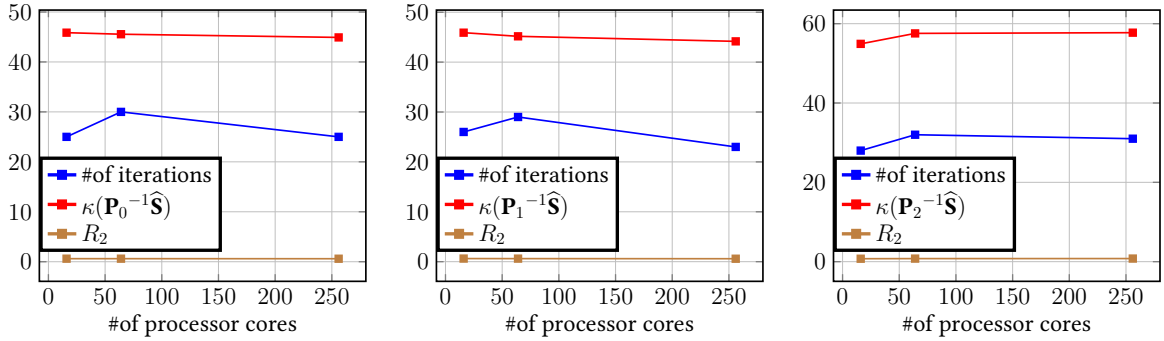
8.13.1. Preconditioner $\mathbf{P}_0$ 8.13.2. Preconditioner $\mathbf{P}_1$ 8.13.3. Preconditioner $\mathbf{P}_2$

Figure 8.13 : Behavior in number of subdomains for $p = 3$ and $H/h = 80$



8.14.1. Preconditioner $\mathbf{P}_0$ 8.14.2. Preconditioner $\mathbf{P}_1$ 8.14.3. Preconditioner $\mathbf{P}_2$

Figure 8.14 : Behavior in number of subdomains for $p = 4$ and $H/h = 80$



8.15.1. Preconditioner $\mathbf{P}_0$ 8.15.2. Preconditioner $\mathbf{P}_1$ 8.15.3. Preconditioner $\mathbf{P}_2$

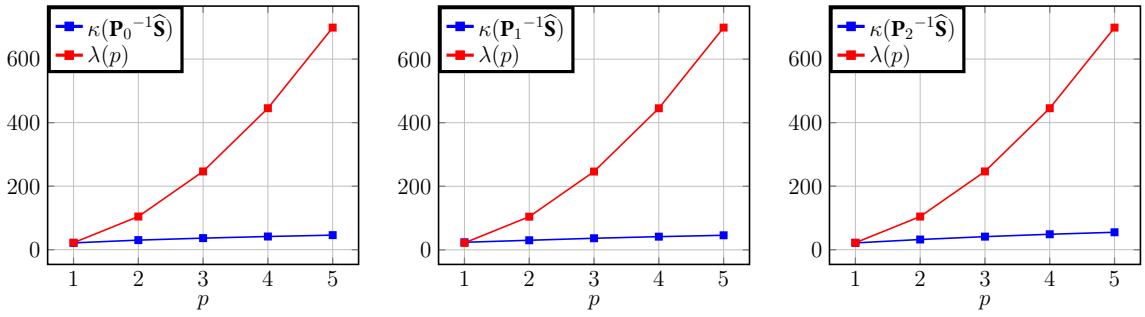
Figure 8.15 : Behavior in number of subdomains for $p = 5$ and $H/h = 80$

In accordance with the theoretical estimates (4.18), (4.26) and (4.32), for each polynomial order $p = 1\,, 2\,, 3\,, 4\,, 5$, a logarithmic growth is clearly observed for all the preconditioners $\mathbf{P}_0$,

$\mathbf{P}_1$ and $\mathbf{P}_2$. These results remain valid for other values of H/h as reported in tables TABLE 22, TABLE 25, TABLE 28, TABLE 31 and TABLE 34.

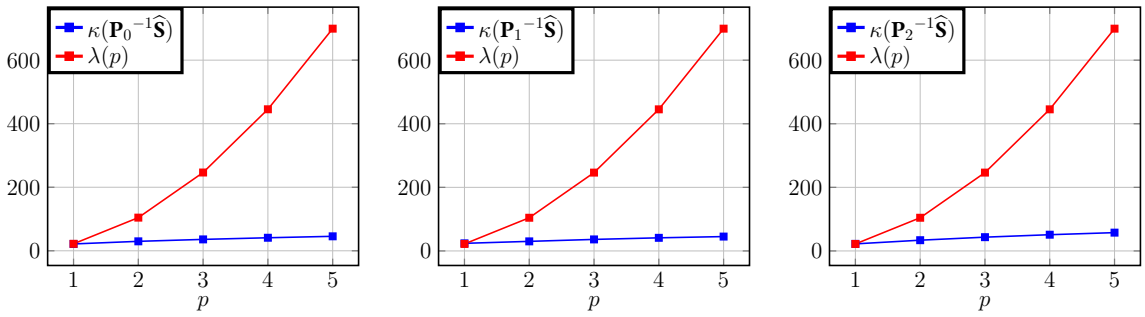
8.2.7 Dependence on Polynomial Order

As in section 8.1.7 devoted to the conforming domain decompositions, we study the dependence on p by reporting the condition number estimate of the preconditioned system as a function of p with H/h constant. Let the function λ be defined as $\lambda(p) = p^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2$. In FIGURE 8.16, FIGURE 8.17 and FIGURE 8.18 below, we plot the condition number of the Schur system preconditioned by $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$.



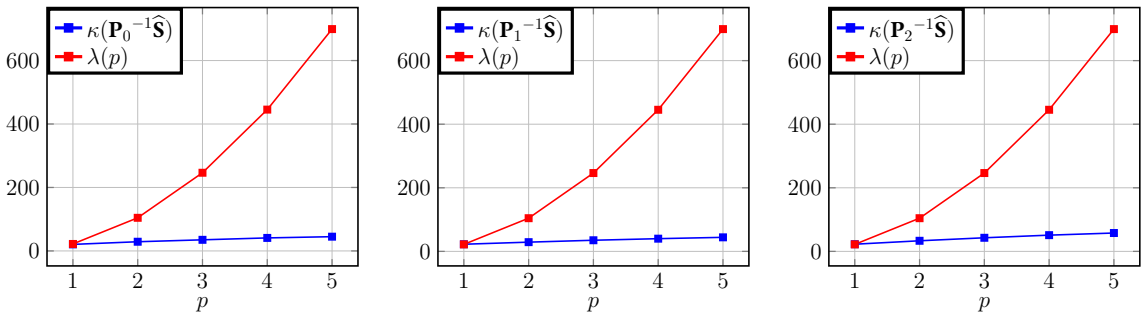
8.16.1. Preconditioner $\mathbf{P}_0$ 8.16.2. Preconditioner $\mathbf{P}_1$ 8.16.3. Preconditioner $\mathbf{P}_2$

FIGURE 8.16 : $\kappa(\widehat{\mathbf{P}}^{-1}\widehat{\mathbf{S}})$ as a function of p with #of subdomains=16 and $H/h = 80$



8.17.1. Preconditioner $\mathbf{P}_0$ 8.17.2. Preconditioner $\mathbf{P}_1$ 8.17.3. Preconditioner $\mathbf{P}_2$

FIGURE 8.17 : $\kappa(\widehat{\mathbf{P}}^{-1}\widehat{\mathbf{S}})$ as a function of p with #of subdomains=64 and $H/h = 80$



8.18.1. Preconditioner $\mathbf{P}_0$ 8.18.2. Preconditioner $\mathbf{P}_1$ 8.18.3. Preconditioner $\mathbf{P}_2$

Figure 8.18 : $\kappa(\widehat{\mathbf{P}}^{-1}\mathbf{S})$ as a function of p with #of subdomains=256 and $H/h = 80$

The Figure 8.16, Figure 8.17 and Figure 8.18 show that, for increasing values of p, our preconditioners behaves similarly to the linear case $p = 1$. To highlight the dependence on p of our preconditioners, we report in Table 35 the ratio R_2 for $H/h = 80$ fixed and increasing values of the polynomial order p. We clearly do not see the factor $\hat{p}^{3/2}$ which appears in the theoretical estimates (4.18), (4.26) and (4.32) (which, we recall, stems the mortar projector operator) since for fixed H, the ratio R_2 does not depend on this factor as shown Table 35. Indeed, the numerical results seem to show an even better behavior than the polylogarithmic dependence on Hp^2/h.

Table 35 : Ratio R_2 and number of iterations (between parenthesis) for $H/h = 80$

	$N\backslash p$	1	2	3	4	5
$\mathbf{P}_0$	16	0.74 (20)	0.65 (22)	0.63 (23)	0.62 (24)	0.61 (25)
	64	0.74 (21)	0.64 (24)	0.62 (27)	0.61 (29)	0.61 (30)
	256	0.71 (17)	0.63 (20)	0.60 (21)	0.60 (24)	0.60 (25)
$\mathbf{P}_1$	16	0.82 (22)	0.65 (24)	0.63 (24)	0.62 (24)	0.62 (26)
	64	0.82 (25)	0.65 (26)	0.63 (27)	0.62 (29)	0.61 (29)
	256	0.77 (21)	0.63 (21)	0.60 (21)	0.60 (22)	0.59 (23)
$\mathbf{P}_2$	16	0.74 (22)	0.70 (27)	0.71 (28)	0.73 (28)	0.74 (28)
	64	0.76 (22)	0.73 (28)	0.75 (30)	0.76 (31)	0.77 (32)
	256	0.77 (21)	0.72 (25)	0.74 (30)	0.76 (31)	0.78 (31)

Table 36 : $\kappa(\widehat{\mathbf{S}})$ and number of iterations (between parenthesis) for $H/h = 80$

$N\backslash p$	1	2	3	4	5
16	$7.24e{+}2$ (86)	$2.19e{+}3$ (132)	$4.21e{+}3$ (168)	$6.82e{+}3$ (215)	$1.01e{+}4$ (268)
64	$8.28e{+}2$ (143)	$2.49e{+}3$ (210)	$4.76e{+}3$ (291)	$7.14e{+}3$ (356)	$1.12e{+}4$ (445)
256	$1.04e{+}3$ (171)	$2.59e{+}3$ (257)	$4.91e{+}3$ (316)	$7.94e{+}3$ (389)	$1.17e{+}4$ (475)

8.2.8 Conclusion

As the results related to the conforming domain decompositions discussed in section 8.1, the numerical results presented in this section dedicated to the nonconforming domain decompositions support the mathematical properties including the p-, H- and h-convergence of the substructuring preconditioners for h-p mortar element method. These results hold for both linear elements ($p = 1$) and high order elements ($2 \leq p \leq 5$).

8.3 Large Scale Simulations

We present the numerical results for large number of subdomains obtained with the Discontinuous Galerkin coarse preconditionner $\mathbf{P}_2$ for conforming domain decompositions. We report the condition number estimate $\kappa(\mathbf{P}_2^{-1}\widehat{\mathbf{S}})$, the number of iterations required by PCG solver and the ratio R_2 as a function of p for $H/h = 80$ fixed and for increasing number of subdomains N. We plot the number of degrees of freedom as a function of p for $H/h = 80$ with 4096, 16384, 22500 and 40000 subdomains.

Table 37 : Ratio R_2 and number of iterations (between parenthesis) for $H/h = 80$

$N\backslash p$	1	2	3	4	5
16	1.65 (31)	1.14 (32)	1.06 (33)	1.03 (38)	1.02 (39)
64	1.74 (31)	1.21 (33)	1.11 (35)	1.07 (40)	1.07 (42)
256	1.76 (28)	1.23 (32)	1.12 (34)	1.08 (36)	1.06 (40)
1,024	1.78 (27)	1.23 (29)	1.12 (31)	1.08 (32)	1.06 (34)
4,096	1.79 (25)	1.23 (28)	1.12 (29)	1.08 (31)	1.06 (31)
16,384	1.52 (20)	0.88 (22)	0.91 (26)	0.94 (27)	0.96 (28)
22,500	1.52 (19)	0.88 (20)	0.69 (22)	0.95 (26)	0.99 (27)
40,000	1.52 (17)	0.88 (20)	0.69 (22)	0.68 (23)	–

The Table 37 shows that the behavior observed in the analysis of the dependence on p of our preconditionner $\mathbf{P}_2$ in section 8.1 for the medium number of subdomains (from 16 to 256)

holds for very large number of subdomains, i.e. the numerical results seem to show an even better behavior than the polylogarithmic dependence on Hp^2/h.

In TABLE 37, for fixed H/h and p, the ratio R_2 seems to be slightly decreasing rather than constant. We believe that there are different causes for this behavior. First of all the problem chosen has a quite regular solution which, for a large number of subdomains, is already well approximated at the coarse level. Moreover, as the coarse mesh becomes finer and finer and the polynomial degree increases, round-off errors might become more significant and they might pollute the numerical results. This issue will be investigated in a forthcoming paper [Sam+14].

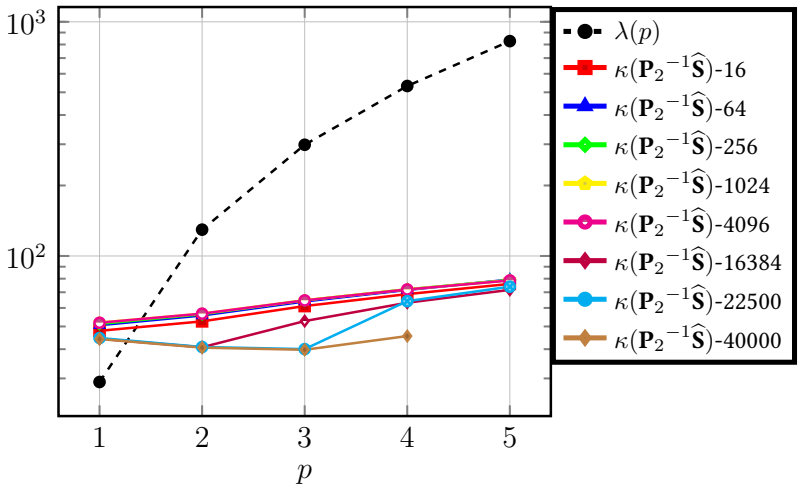


FIGURE 8.19 : Condition number of the preconditioned system as a function of p with 16, 64, 256, 1024, 4096, 16384, 22500, 40000 subdomains and $H/h = 80$

8.4 Scalability Analysis

In this section, we analyze the performance and the scalability of our parallel algorithms for solving the Schur complement system (4.41) preconditioned by the substructuring preconditioner $\mathbf{P}_2$ introduced in chapter 4. For this end, we perform the strong and weak scalability experiments, already introduced in section 1.7. The simulations were achieved on Curie, a Tier-0 computational platform for PRACE, previously described at the beginning of this chapter. The tests are carried out with first-order and high-order mortar finite element approximations.

8.4.1 Strong Scalability

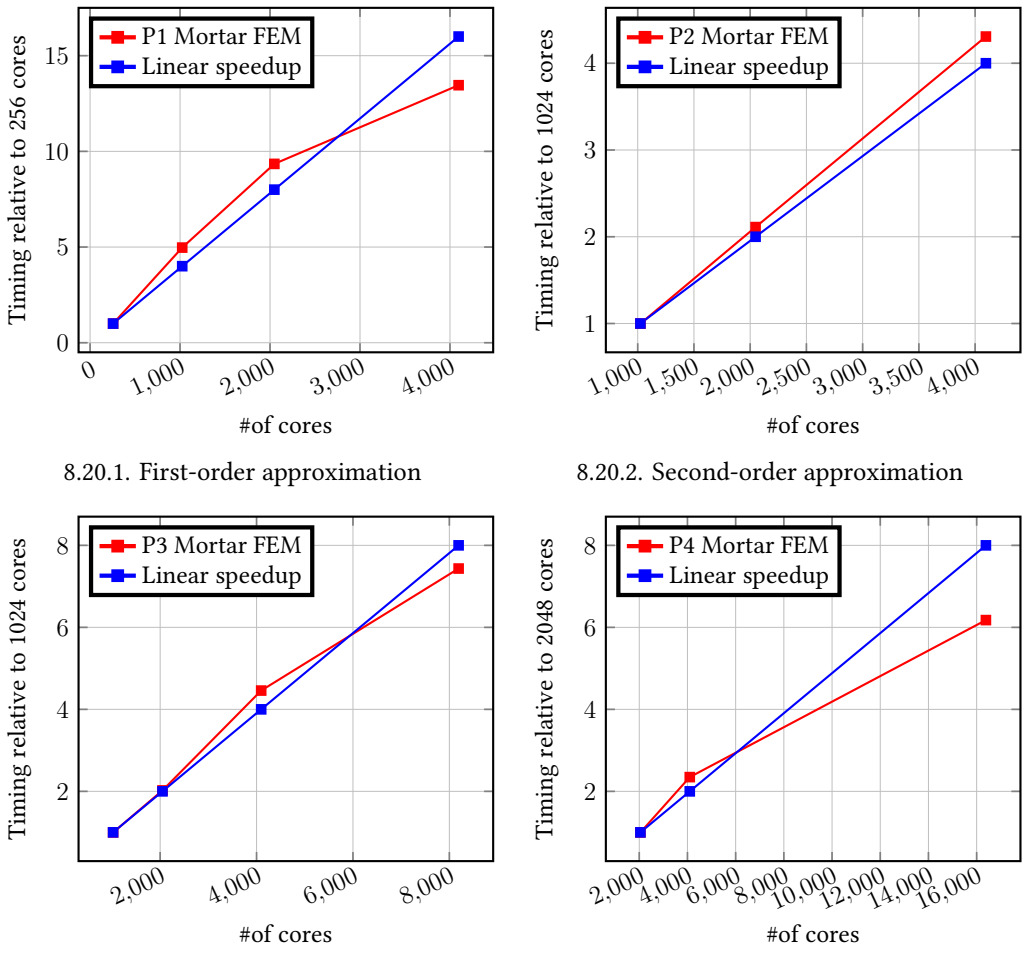
To measure the speedup of the preconditionner $\mathbf{P}_2$, we perform the strong scalability analysis for $p = 1, 2, 3, 4$. We consider the conforming domain decompositions with simplex elements, as in FIGURE 8.1. We compute the time required for solving the Schur complement system (4.41) preconditioned by $\mathbf{P}_2$ when increasing the number of processor cores, while maintaining the overall problem size constant. We perform the experiments by using 256, 1024, 2048, 4096 and 16384 processor cores, depending to the polynomial order used.

For the first-order mortar approximation ($p = 1$), see FIGURE 8.20.1., the total number of unknowns is approximately equal to 30 millions and we use respectively 256, 1024, 2048 and 4096 processor cores for solving the system. Until 2048 cores, we obtain a nice speedup relative to 256 cores. We observe a deterioration of the speedup from 4096 cores, more precisely, the speedup relative to 256 cores is 13.45 while the linear speedup is 16. This deterioration is mainly caused by the low workload in the subdomains while the interprocess communications significantly increase.

For the second-order mortar approximation ($p = 2$), see FIGURE 8.20.2., the total number of unknowns is about 122 millions and the simulations are performed using respectively 1024, 2048 and 4096 processor cores. A good speedup is obtained, as shown the plots in FIGURE 8.20.2.. More specifically, using 4096 cores, the speedup relative to 1024 cores is 4.31 while the linear speedup is 4. This over-linearity of our speedup is due to the use of direct solver for the solution of local problems although the workload becomes small in subdomains.

For the third-order mortar approximation ($p = 3$), see FIGURE 8.20.3., the total number of unknowns is roughly equal to 273 millions and the experiments are achieved using respectively 1024, 2048, 4096, 8192 processor cores. Using 8192 cores, the speedup relative to 1024 cores is 7.44, which is near to the linear speedup equal to 8.

For the fourth-order mortar approximation ($p = 4$), see FIGURE 8.20.4., the total number of degree of freedom varies between 485 and 486 millions and the simulations are realized using respectively 2048, 4096 and 16384 processor cores. Using 16384 cores, the speedup relative to 2048 cores is 6.20 whereas the linear speedup is 8. The principal reason of this deterioration of the speedup is a significant increase by about 5 millions of the number of unknowns between 2048 and 16384 cores.



8.20.1. First-order approximation

8.20.2. Second-order approximation

8.20.3. Third-order approximation

8.20.4. Fourth-order approximation

Figure 8.20 : Strong scalability analysis for $p = 1\,, 2\,, 3\,, 4$

8.4.2 Weak Scalability

To evaluate the efficiency of the preconditionner $\mathbf{P}_2$ using the Discontinuous Galerkin interior penalty method as coarse problem, we perform the weak scalability analysis with the same problem settings as for the results reported in Table 37 and in Figure 8.19. We consider the first-order and high-order mortar finite element approximations and the conforming domain decompositions with simplex elements, as in Figure 8.1. We are interested in the computational time required for solving the Schur complement system (4.41) preconditioned by $\mathbf{P}_2$ when increasing the number of processor cores, while maintaining the local problem size constant. The experiments are achieved by using 1024, 4096, 16384, 22500 and 40000 processor cores.

For the fourth-order mortar approximation ($p = 4$), the number of degrees of freedom per subdomain is approximately equal to 120000. Using 1024 cores, the total number of unknowns is approximately equal to 122 millions and the system is solved in 121.98s. Employing 40000

cores, we reach about 5 billions of unknowns and the system is solved in 132.88s, i.e. with an efficiency relative to 1024 cores equal to 92%, as shown FIGURE 8.21.4..

For the third-order mortar approximation ($p = 3$), the number of degrees of freedom per subdomain is about 67000. Using 1024 cores, the total number of unknowns is approximately equal to 68 millions and the system is solved in 64.30s. Employing 40000 cores, the number of unknowns of the overall problem is roughly equal to 3 billions and the system is solved in 68.80s, i.e. with an efficiency relative to 1024 cores equal to 93.46%, as shown FIGURE 8.21.3..

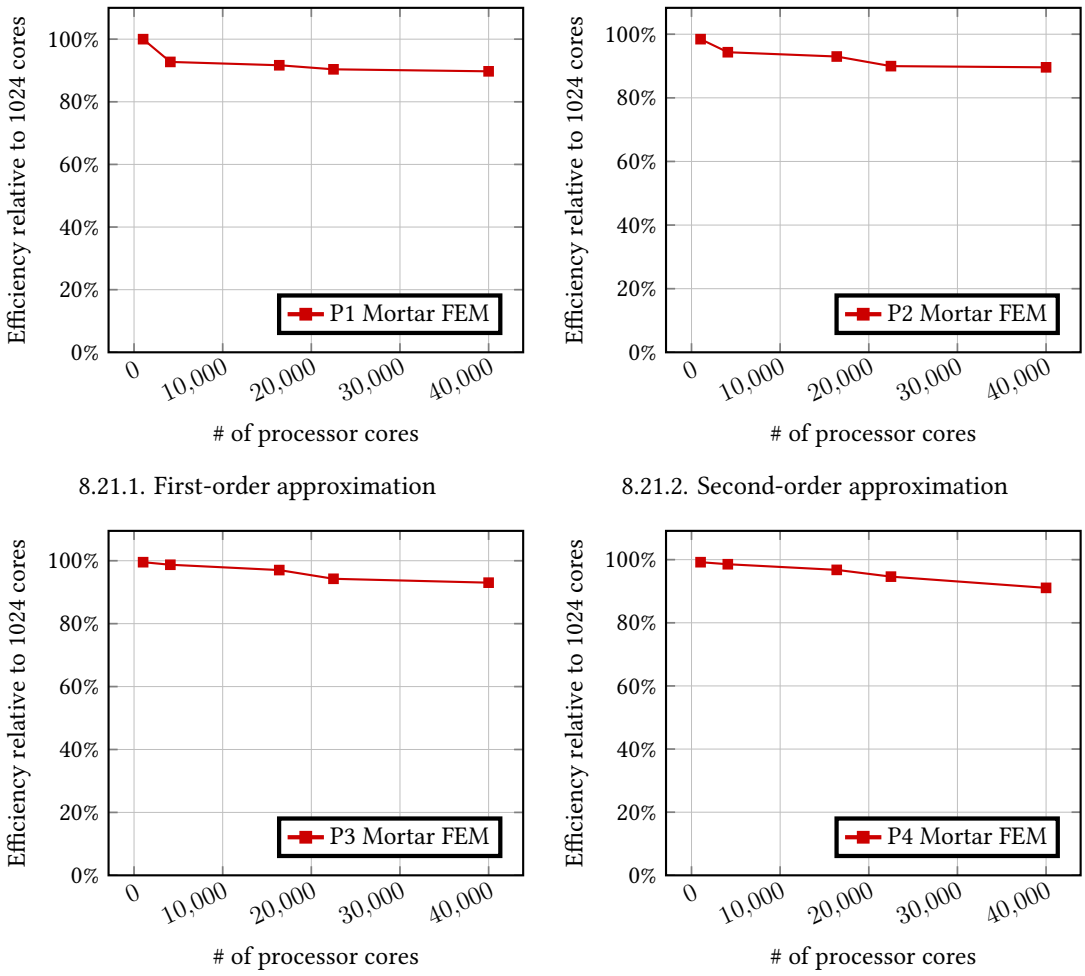
For the second-order mortar approximation ($p = 2$), the number of degrees of freedom per subdomain is approximately equal to 30000. Using 1024 cores, the total number of unknowns is about 30 millions and the system is solved in 26.41s. Employing 40000 cores, the number of unknowns of the global problem is roughly equal to 1.2 billions and the system is solved in 29.02s, i.e. with an efficiency relative to 1024 cores equal to 91%, as shown FIGURE 8.21.2..

For the first-order mortar approximation ($p = 1$), the number of degrees of freedom per subdomain is roughly equal to 7400. Using 1024 cores, the total number of unknowns is about 7.5 millions and the system is solved in 6.38s. Employing 40000 cores, the number of unknowns of the global problem is approximately equal to 300 millions and the system is solved in 7.11s, i.e. with an efficiency relative to 1024 cores equal to 89.73%, as shown FIGURE 8.21.1..

To highlight the efficiency of our parallel algorithms, we summarize in TABLE 38 the efficiency relative to 1024 processor cores for increasing values of polynomial order $p = 1\,, 2\,, 3\,, 4$.

TABLE 38 : Efficiency relative to 1024 processor cores

$N \backslash p$	1	2	3	4
1,024	100%	100%	100%	100%
4,096	92.73%	95.83%	99.18%	99.35%
16,384	91.67%	94.46%	97.48%	97.54%
22,500	90.37%	91.39%	94.7%	95.39%
40,000	89.73%	91.01%	93.46%	91.8%



8.21.1. First-order approximation

8.21.2. Second-order approximation

8.21.3. Third-order approximation

8.21.4. Fourth-order approximation

Figure 8.21 : Weak scalability analysis for $p = 1\,, 2\,, 3\,, 4$

The efficiency analysis shows that the preconditionner $\mathbf{P}_2$ scales well on large scale computer architectures.

The resolution of the coarse problem at each PCG iteration has a significant effect on the computational cost. The number of degrees of freedom of the coarse problem is four times the number of subdomains and it is solved in parallel on three number of processor cores in this weak scalability study. When we further increase the number of cores, for example reaching hundreds of thousands of processors, the resolution of the coarse grid problem in parallel on more cores will be essential, since it will become particularly consequent in term of size.

The coarse grid problem being responsible of the scalability for large number of subdomains, the construction, analysis and the implementation of the Discontinuous Galerkin coarse problem was decisive for the performance of our parallel algorithms.

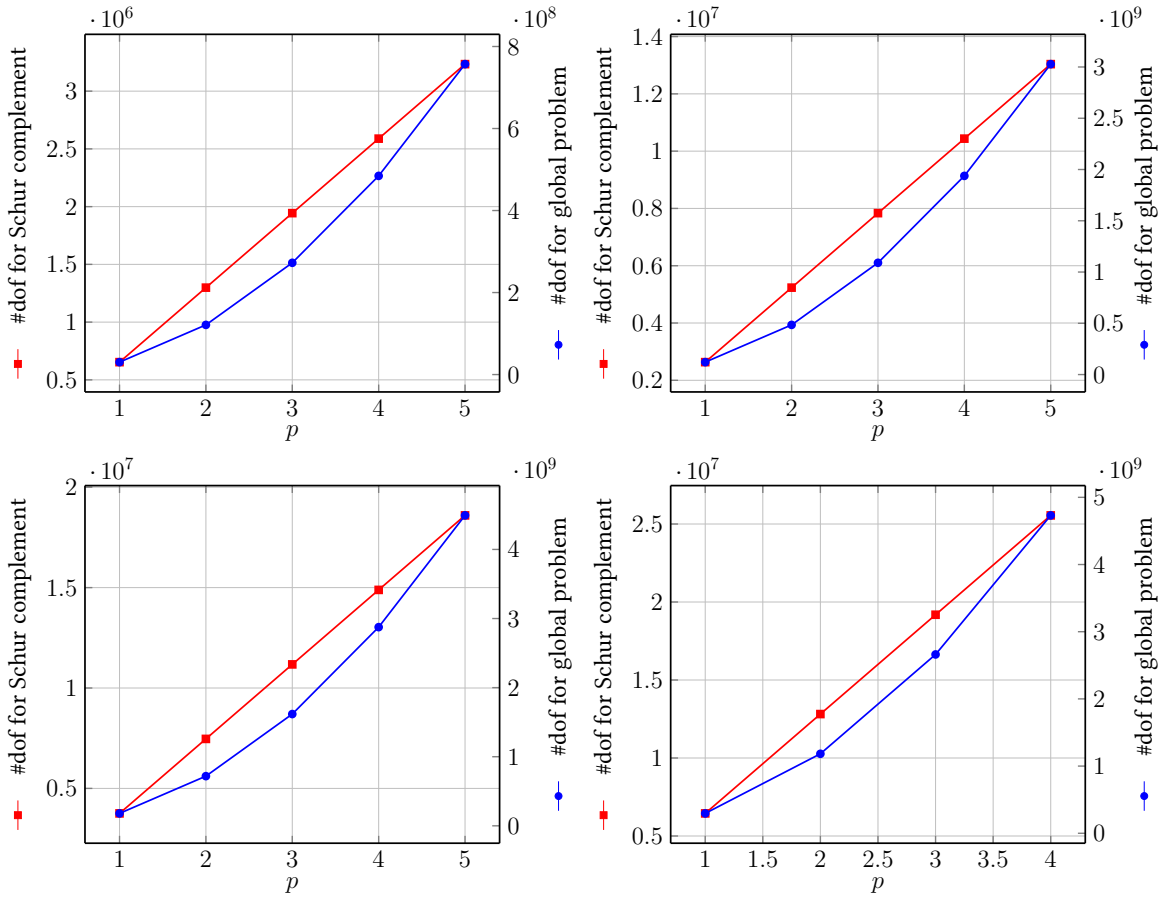


Figure 8.22 : Number of degrees of freedom as a function of p with 4096, 16384, 22500 and 40000 subdomains and $H/h = 80$

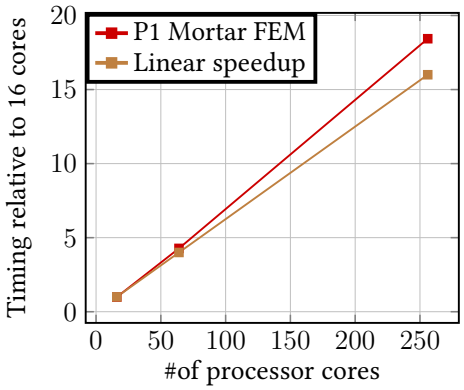
8.5 Scalability Analysis on Medium Scale Architectures

In this section, we analyze the performance and the scalability on medium scale architectures (from 16 to 256 processor cores) of our parallel algorithms for solving the Schur complement system (4.41) preconditioned by the substructuring preconditioner $\mathbf{P}_2$. We consider the same problem settings as for the scalability analysis on large scale architectures presented in section 8.4 and the experiments are carried out with first-order and high-order mortar finite element approximations. The simulations were achieved at MesoCentre@Strasbourg on hpc-login, a supercomputer previously described at the beginning of this chapter.

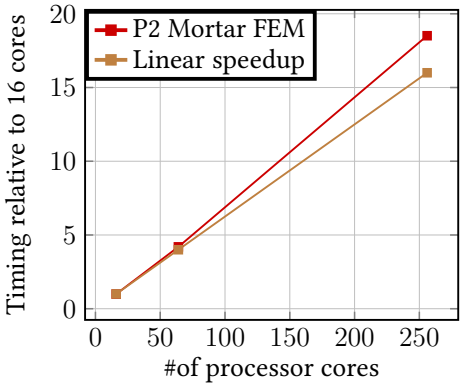
8.5.1 Strong Scalability

We compute the time required for solving the Schur complement system (4.41) preconditioned by $\mathbf{P}_2$ when increasing the number of processor cores, while maintaining the overall problem size constant. We perform the experiments by using 16, 64 and 256 processor cores for increasing values of polynomial order $p = 1, 2, 3, 4, 5$. We obtain a nice speedup relative to

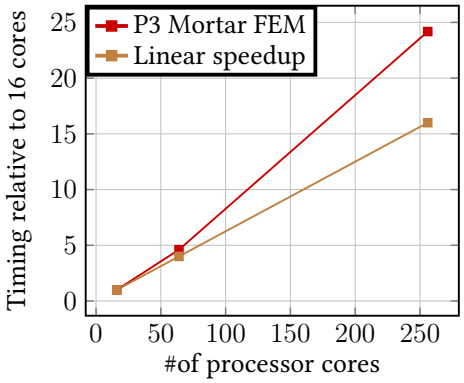
16 cores for all mortar finite element approximations considered, as shown FIGURE 8.23. The over-linearity observed in the speedup analysis is due to the use of direct solver for the solution of local problems although the subdomains become smaller.



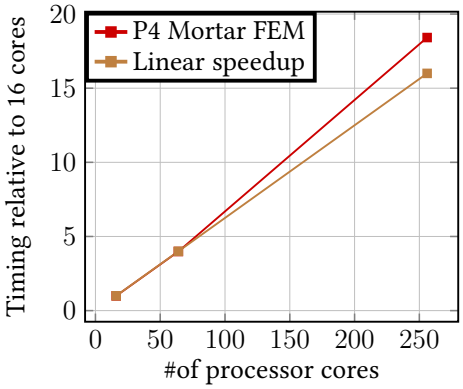
8.23.1. First-order approximation



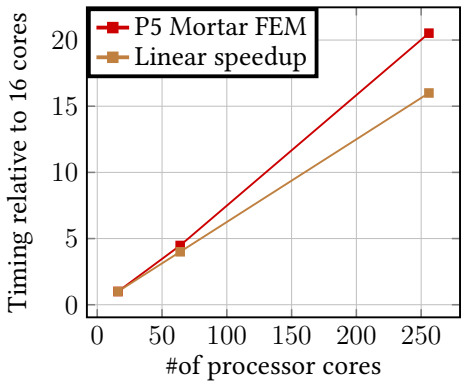
8.23.2. Second-order approximation



8.23.3. Third-order approximation



8.23.4. Fourth-order approximation

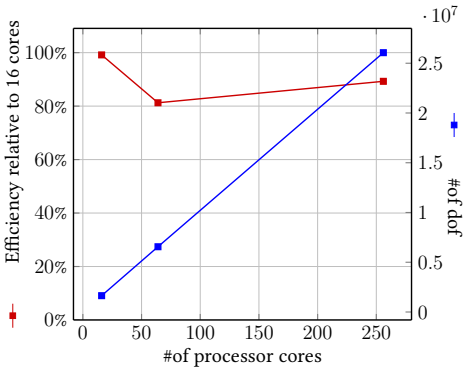


8.23.5. Fifth-order approximation

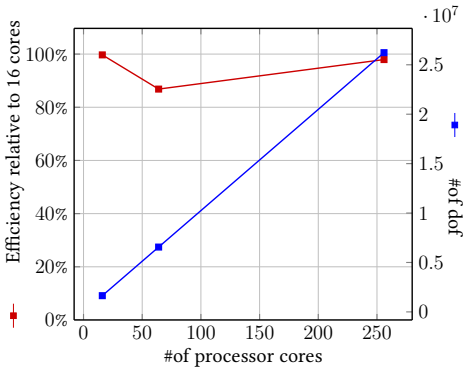
FIGURE 8.23 : Strong scalability analysis

8.5.2 Weak Scalability

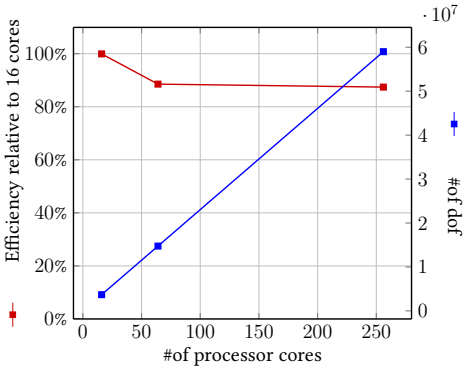
We compute the time required for solving the Schur complement system (4.41) preconditioned by $\mathbf{P}_2$ when increasing the number of processor cores, while maintaining the local problem size constant. We perform the experiments by using 16, 64 and 256 processor cores for increasing values of polynomial order $p = 1, 2, 3, 4, 5$.



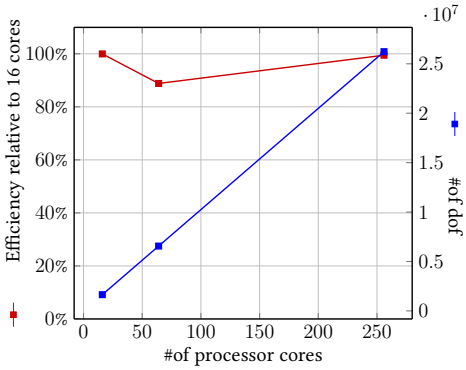
8.24.1. First-order approximation



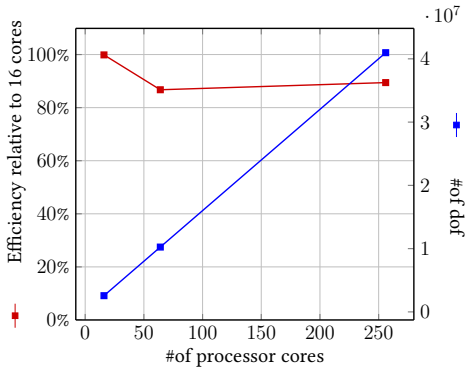
8.24.2. Second-order approximation



8.24.3. Third-order approximation



8.24.4. Fourth-order approximation



8.24.5. Fifth-order approximation

Figure 8.24 : Weak scalability analysis

The efficiency analysis shows that the preconditionner $\mathbf{P}_2$ scales well on medium scale computer architectures for all mortar finite element approximations considered, $p = 1\,, 2\,, 3\,, 4\,, 5$ as shown FIGURE 8.24.

8.6 Conclusion

In this chapter, we presented the numerical experiments for substructuring preconditioners for h-p mortar element method in different configurations, including conforming and nonconforming domain decompositions, linear and high-order finite elements. The mathematical properties of three different substructuring preconditioners proposed in this work were analyzed by performing a p-, H- and h-convergence study. The number of iterations required for the Preconditioned Conjugate Gradient (PCG) method for solving the preconditioned Schur complement system, the condition number estimate and the ratio between the condition number estimate and its bound for the preconditioned matrix were reported for each preconditioner considered. As the theoretical results, a logarithmic growth was observed for all preconditioners surveyed with conforming and nonconforming domain decompositions and the linear finite elements. Indeed, in the case of high-order elements, the numerical results indicated an even better behavior than the polylogarithmic dependence on Hp^2/h and the main reasons for this behavior were discussed. To evaluate the performance of our parallel algorithms, the strong and weak scalability were analyzed with the Discontinuous Galerkin coarse grid preconditioner. The best scalability (strong and weak) properties were obtained on medium scale computational platforms (from 16 to 256 processor cores on hpc-login) and on large scale computer architectures (from 1024 to 40.000 processor cores on Curie). These scalability results hold for linear finite elements ($p = 1$) and for high-order finite elements ($2 \leq p \leq 5$).

Chapter 9

Collecting Framework for Numerical Results

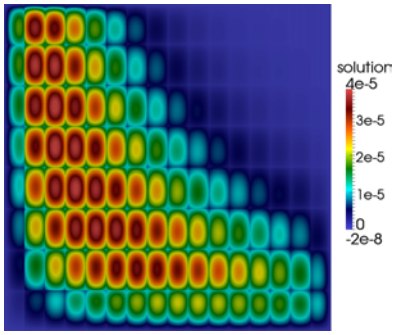
This chapter presents the some numerical results for Schwarz and three-field methods described previously in this thesis and summarizes some results for mortar element formulation using Lagrange multipliers. The numerical experiments for Schwarz methods in parallel and for three-field method are given respectively in 9.1 and in section 9.2. The numerical results for mortar element method with Lagrange multipliers including the strong and weak scalability analysis are summarized in section 9.3.

Ce chapitre présente quelques résultats numériques des méthodes de Schwarz et de three-field décrites précédemment dans cette thèse et résume quelques résultats pour la méthode des éléments finis mortar utilisant les multiplicateurs de Lagrange. Les expériences numériques pour les méthodes de Schwarz en parallèle et celles pour la méthode three-field sont données respectivement dans la section 9.1 et dans la section 9.2. Les résultats numériques pour la méthode mortar avec multiplicateurs de Lagrange comprenant l'analyse de scalabilité forte et faible sont résumés dans la section 9.3.

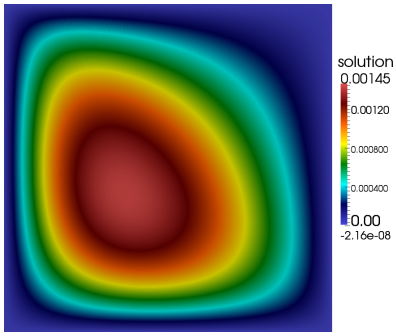
Contents

9.1 Numerical Results for Schwarz Methods

To illustrate our implementation of Schwarz methods, we consider the problem (2.3) over a partition over the domain $\Omega = [0\,,1]^2$ into 128 overlapping subdomains (16 subdomains in x-axis direction and 8 subdomains in y-axis direction) with non-matching meshes. The Dirichlet boundary condition is given by and $u = g(x\,,y) = 0$ on $\partial\Omega$ and the chosen source term is written $f(x\,,y) = \exp(-10xy)\cos(\frac{3\pi}{8})\sin(xy)$.



9.1.1. First Schwarz iteration



9.1.2. Solution at convergence

Figure 9.1 : Numerical solutions obtained by parallel additive Schwarz algorithm in 2D on 128 processor cores(1 subdomain/processor core)

The numerical solutions in Figure 9.1 are obtained using $\mathbb{P}_2$ Lagrange elements. The tolerance of the numerical solver is fixed to $1e-07$. The characteristic mesh size is $0\ .01$ in each subdomain and the size of the overlap is $0\ .02$. The grids may be nonconforming. The number of Schwarz iterations to convergence is 130 and the relative L^2 error $\|u-u_h\|_{L^2}$ is $1\ .164901e-06$.

The speedup displayed in Table 39 corresponds to the assembly plus the solve times. We can see that the scaling is good except for the last configuration where the local problems are too small.

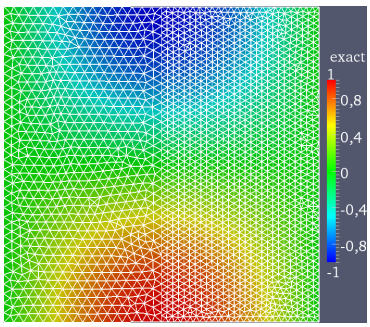
Table 39 : Strong scalability analysis for Schwarz methods

Number of Cores	Absolute Times	Speedup
1,024	41.2	1
2,048	18.2	2.26
4,096	10	4.12
8,192	7	5.88

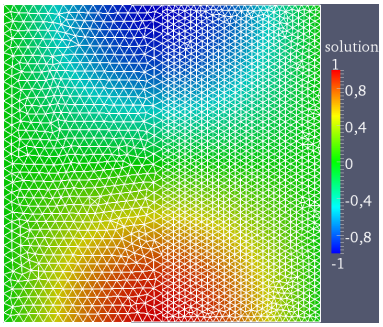
9.2 Numerical Results for Three-field Method

We present in numerical solutions corresponding to partition of Ω into two nonoverlapping subdomains Ω_1 and Ω_2 with the following configurations (i) $g(x\,,y) = \sin(\pi x)\cos(\pi y)$ is the exact solution (ii) $f(x\,,y) = 2\pi^2 g$ is the right hand side of the equation (iii) $\mathbb{P}_2$, $\mathbb{P}_1$ and $\mathbb{P}_3$ approximations respectively in Ω_1, Ω_2 and Γ (iv) we set $h_{\Omega_1} = 0\ .03$, $h_{\Omega_2} = 0\ .02$ and $h_\Gamma = 0\ .01$ in 2D (v) we set $h_{\Omega_1} = 0\ .05$, $h_{\Omega_2} = 0\ .07$ and $h_\Gamma = 0\ .02$ in 3D.

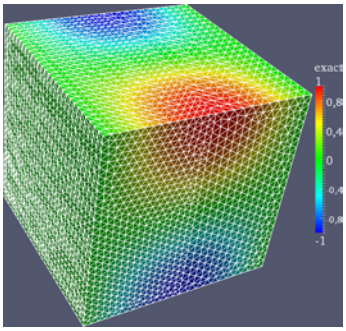
Table 40 summarizes several error quantities for both problems studied and the global solution u_h defined on Ω as $u_h = u_{1,h}$ on Ω_1 and $u_h = u_{2,h}$ on Ω_2. Let $e^0_{i,h} = \|u_{i,h} - g\|_{0,\Omega_i}$ be the L^2-error norm in subdomain Ω_i and let $e^1_{i,h} = \|u_{i,h} - g\|_{1,\Omega_i}$ be the H^1-error norm in subdomain Ω_i, $i = 1\,,2$. We denote by $e^0_{\Gamma,h} = \|\lambda_h - g\|_{0,\Gamma}$ the L^2-error norm at subdomain interface Γ and by e^1_h the H^1-error norm in Ω.



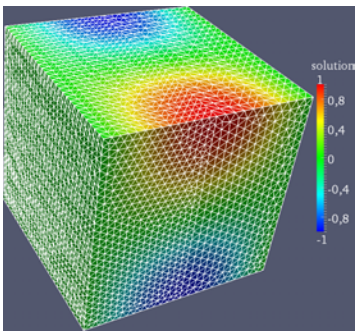
9.2.1. Exact solution in 2D



9.2.2. Numerical solution in 2D



9.2.3. Exact solution in 3D



9.2.4. Numerical solution in 3D

Figure 9.2 : Numerical solutions for three-field method in 2D and 3D

Table 40 : Numerical results for three-field method in 2D and 3D

	$e^0_{1,h}$	$e^0_{2,h}$	$e^1_{1,h}$	$e^1_{2,h}$	$e^0_{\Gamma,h}$	e^1_h
2D	$6.22\cdot10^{-11}$	$6.24\cdot10^{-11}$	$1.15\cdot10^{-8}$	$1.61\cdot10^{-8}$	$4.87\cdot10^{-12}$	$1.98\cdot10^{-8}$
3D	$2.65\cdot10^{-7}$	$2.24\cdot10^{-7}$	$2.21\cdot10^{-5}$	$1.39\cdot10^{-5}$	$2.45\cdot10^{-8}$	$2.61\cdot10^{-5}$

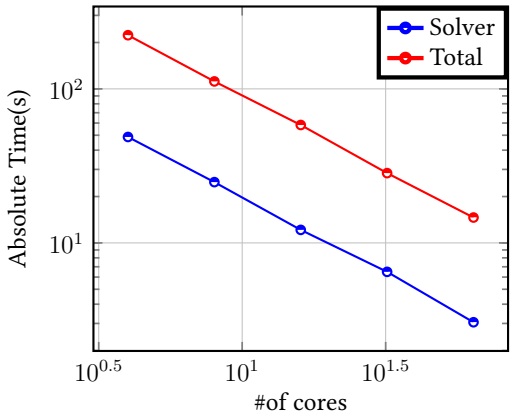
9.3 Numerical Results for Mortar Element Method with Lagrange Multipliers

We present in this section the numerical results for parallel implementation of mortar element method using Lagrange multipliers described in chapter 5 and investigated in implementation view point in section 7.3. We consider the problem (5.1) in 3D with the chosen analytical solution $g = \sin(\pi x)\cos(\pi y)\cos(\pi z)$ and $f = -\Delta g = 3\pi^2 g$ the corresponding source term. The problem is solved in the parallelepiped $\Omega = [0\,, L_x] \times [0\,, L_y] \times [0\,, L_z]\,,\ L_x\,, L_y\,, L_z > 0$. The following numerical results are obtained using P_2 finite element approximations. The tolerance of the Krylov solver is $\varepsilon = 10^{-7}$.

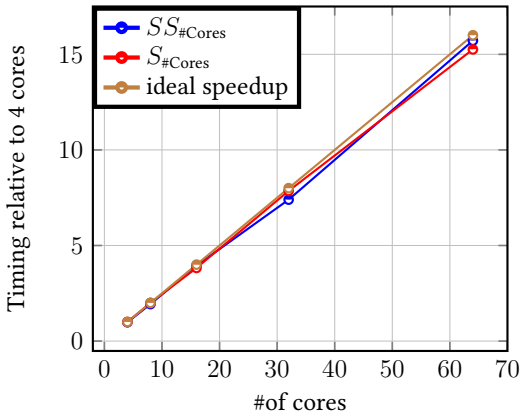
The simulations have been performed at Leibniz Supercomputing Centre (LRZ) on SuperMUC. SuperMUC is the Tier-0 supercomputer with 155.656 processor cores in 9400 compute nodes which provides resource to PRACE via the German Gauss Centre. The system is an IBM System x iDataPlex based on Intel Sandy Bride EP processors. SuperMUC has a peak performance of 3.2 PFLOP/s consisting of 18 islands, each combining 512 compute nodes with 16 physical cores and 32 GB per node. The nodes are connected by a non-blocking fat tree, based on Infiniband FDR10.

9.3.1 Strong Scalability Analysis

We analyze the strong scalability results corresponding to the partition of the global domain Ω into $L_x \times L_y \times L_z$ subdomains (1 subdomain per core) with the fixed lengths $L_x = L_y = L_z = 1$. We plot in FIGURE 9.3.1. the absolute solve time and the total time as a function of number of processor cores in logarithmic axis. In FIGURE 9.3.2., we present the speedup and ideal speedup as a function of number of cores. The total number of degrees of freedom is approximately equal to 500000 and all the measured timings are expressed in seconds.



9.3.1. Absolute time versus #of cores



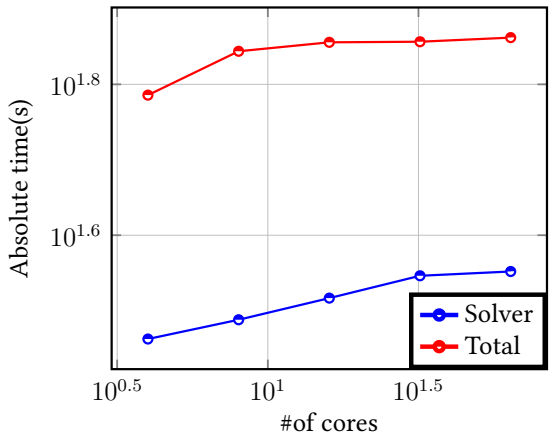
9.3.2. Speedup versus #of cores

FIGURE 9.3 : Strong scalability analysis

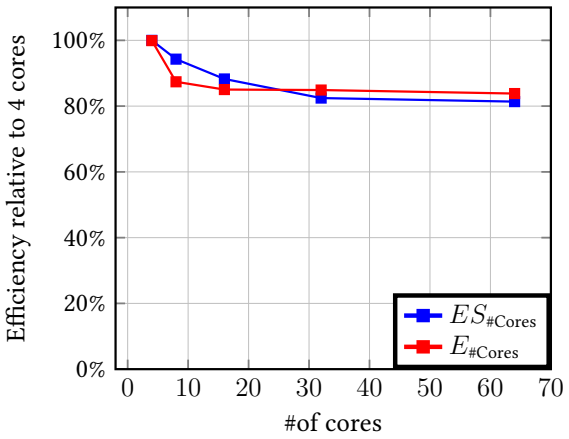
We obtain a nice speedup related to the total computational time and to the solver time, as shown FIGURE 9.3.2..

9.3.2 Weak Scalability Analysis

We present the weak scalability results corresponding to the partition of the global domain Ω into $L_x \times L_y \times L_z$ subdomains (1 subdomain per processor) with $L_x \times L_y \times L_z =$ #of cores. We plot in FIGURE 9.4.1. the absolute solve time and total time as a function of the number of cores in logarithmic axis. In FIGURE 9.4.2., we present the efficiency relative to four cores as a function of the number of cores. We denote by E_p the efficiency relative to four cores for the total time on p cores and by ES_p the efficiency relative to four cores for the solver time on p cores. The total number of degrees of freedom is approximately equal to 30000 per subdomain and all the measured timings are expressed in seconds.



9.4.1. Absolute time versus #of cores



9.4.2. Efficiency versus #of cores

FIGURE 9.4 : Weak scalability analysis

The strong and weak scalability analysis presented in FIGURE 9.3 and FIGURE 9.4 clearly show that our parallel computational framework for solving the algebraic linear system of saddle-point type arising from the discretization of mortar finite element method with Lagrange multipliers in 2D and 3D perform well on the small size computer architectures.

9.4 Conclusion

We summarized in this chapter the numerical simulations for Schwarz methods, three-field method and mortar element method with Lagrange multipliers. These results confirm the theoretical properties of these methods described in the part I and investigated in the implementation view point in part II. Regarding the mortar finite element method with Lagrange

multipliers, the scalability analysis (strong and weak scalability) supports the best performance property of our parallel algorithms.

Conclusion

The significant advances in terms of large numerical simulations for complex scientific and engineering problems have always been related to major levels reached by the technologies of high-performance computing. The numerical methods able to best exploit these modern computational platforms are challenging and booming research topics in the field of scientific computing, specifically domain decomposition methods. In this thesis, we investigated a numerical and computational framework for diverse domain decomposition methods and preconditioners.

In the Part I devoted to the numerical methods, we first reviewed the overlapping Schwarz methods and the iterative substructuring methods such as the mortar finite element method, the three-field method and the FETI method. The classical formulations for these methods were recalled and we reported the convergence analysis supporting the theoretical estimates. The main subject discussed in this work was the mortar finite element method, for which we introduced two different formulations. The first one was the original mortar finite element formulation, in which the mortar weak matching condition is directly taken into account in the approximation space. One of our principal motivations for this formulation is that it lead to a sparse, positive and definite linear system allowing the use of efficient preconditioners. The substructuring preconditioners for this mortar formulation in the h-p finite element framework have been handled. A particular emphasis was placed on the construction and the analysis of the coarse grid preconditioner and its fundamental role for the good scaling properties. We analyzed two variants of coarse preconditioner, whose the first one is an improved version of a coarse preconditioner already existing in the literature. The second is our main proposed version based on a Discontinuous Galerkin interior penalty method as coarse problem. The second mortar formulation studied in work was the approach using Lagrange multiplier for ensure the mortar weak continuity constraints. The algebraic linear system of saddle-point type arising from such a formulation was revised. For solving this indefinite saddle-point linear system, the block diagonal preconditioners involving the local preconditioners for subdomains and the algebraic Schur complement on the Lagrange multiplier were analyzed.

In the Part II dedicated to the implementation of various numerical methods and preconditioners described in Part I, we developed an implementation framework for the substructuring preconditioners and the Schur complement system, the algebraic representation of the Steklov-Poincaré operator in two dimensional space. We defined some basic ingredients re-

quired for Feel++ implementation and emphasized the crucial role of the linear interpolation operator for domain decomposition framework in Feel++. The geometric and algebraic aspects for the realization of the preconditioners were studied and the code design illustrating the genericity and the flexibility of our parallel algorithms was summarized. In the same part, the implementation of Schwarz methods, three-field method and mortar element method with Lagrange multipliers was surveyed. The Schwarz and three-field methods were considered in the purpose to establish a teaching and research programming environment in Feel++ for a wide range of these methods.

In the Part III centered on the numerical experiments, the numerical results for various numerical methods and preconditioners investigated in Part I and in Part II are summarized. First, the problem settings and the computational platforms for all numerical simulations for substructuring preconditioners achieved were defined. We presented the results related to the substructuring preconditioners in different configurations, including conforming and nonconforming domain decompositions, linear and high-order finite elements. The mathematical properties of three different substructuring preconditioners proposed in this work were analyzed by performing a p-, H- and h-convergence study. The number of iterations required for the Preconditioned Conjugate Gradient (PCG) method for solving the preconditioned Schur complement system, the condition number estimates and the ratio between the condition number estimates and its bound for the preconditioned matrix were reported for each preconditioner considered. As the theoretical results, a logarithmic growth was observed for all preconditioners surveyed with conforming and nonconforming domain decompositions and the linear finite elements. Indeed, in the case of high-order elements, the numerical results indicated an even better behavior than the polylogarithmic dependence on Hp^2/h and the main reasons for this behavior were discussed. To evaluate the performance of our parallel algorithms, the strong and weak scalability were analyzed with the Discontinuous Galerkin coarse grid preconditioner. The best scalability (strong and weak) properties were obtained on medium scale computational platforms (from 16 to 256 processor cores) and on large scale computer architectures (from 1024 to 40.000 processor cores). These scalability results hold for linear finite elements ($p = 1$) and high-order finite elements ($p = 2\,, 3\,, 4\,, 5$). In the same part, some basic numerical results for Schwarz methods including seamless and explicit communication approach, and for three-field method were presented. Regarding the mortar finite element method with Lagrange multipliers, the three-dimensional simulations performed on SuperMUC, a Tier-0 supercomputer for PRACE located at Leibniz Supercomputing Centre (LRZ) were exposed. These simulations include the strong and weak scalability analysis showing the best performance properties of our parallel algorithms.

Ongoing work and Perspectives

The goal we have set ourselves is to extend the framework for substructuring preconditioners for h-p mortar finite element method presented in this work to harder problems and complex computational domains. The long-term objective would be to extend the current parallel

implementation framework for this mortar method and preconditioners to hybrid computer architectures GPU/GPGPU in Feel++ programming environment.

Extension to Complex Domains

The numerical simulations presented in the Chapter 8 dedicated to the substructuring preconditioners are obtained with conforming and nonconforming domain decompositions of polygonal computational domains in two-dimensional space. The work is underway to extend our implementation to complex domains, for which the necessary ingredients are already available in Feel++. The main idea is to generate the coarse mesh with hypercube elements from an arbitrary computational domain, with a particular attention for the regularity along the element interfaces. The implementation details of this extension have already been discussed in section 6.3 of this dissertation.

Implementation in 3D

In the literature dedicated to the preconditioning techniques, particular difficulties are related to the three-dimensional problems. We are currently working to extend our implementation to three-dimensional preconditioner framework for mortar finite element method. This work requires some developments in Feel++, especially in the table of degrees of freedom for taking into account the mortar conditions for the processing of the cross-points and cross-edges across the subdomain interfaces.

Extension to Multiscale Problems

Multiscale problems are prevalent in industrial applications. A simple model of such problems is the diffusion equation with highly heterogeneous coefficients, which is used for example for predicting the presence of oil and gas in a porous medium. The extension of our preconditioner framework to these problems is an interesting perspective of this work.

Appendices

This appendices collect the essential results used in this thesis. In section A, the Sobolev spaces are recalled. The finite element approximations are reviewed in section B. A theoretical framework for the solution of algebraic linear systems is given in section C. Fast algorithms for the computing of matrix square root proposed in [HHT08] are available in section D. A special emphasis is placed on the proofs of theorems and lemmas proposed in this thesis, specifically for the mortar element method with constrained space and the substructuring preconditioners in section E. Some tools in Feel++ for domain decomposition methods, especially for the substructuring preconditioners are presented in section H. The Aitken acceleration procedure of Schwarz methods is reminded in section I. Some numerical results for substructuring preconditioners are available in section J.

Ces appendices collectionnent les résultats essentiels utilisés dans cette thèse. Dans la section A, les espaces de Sobolev sont rappelés. Les approximations par éléments finis sont révisées dans la section B. Un framework théorique pour la résolution des systèmes linéaires algébriques est donné dans la section C. Les algorithmes rapides de calcul de la racine carrée de matrice proposés dans [HHT08] sont disponibles dans la section D. Un accent particulier est mis sur les preuves des théorèmes et des lemmes proposés dans cette thèse, spécifiquement pour la méthode des éléments finis mortar avec espace contraint et les préconditionneurs par sous-structuration dans la section E. Quelques outils dans Feel++ pour les méthodes de décomposition de domaine, particulièrement pour les préconditionneurs par sous-structuration sont présentés dans la section H. La procédure d'accélération Aitken des méthodes de Schwarz est rappelée dans la section I. Quelques résultats numériques pour les préconditionneurs par sous-structuration sont disponibles dans la section J.

A Sobolev Spaces

Sobolev spaces are essential tools in solving elliptic partial differential equations. Using the variational formulation of the PDE, the existence of a generalized solution in the appropriate Sobolev space can be established by using variational methods, in particular the Lax-Milgram lemma. Regularity results are also expressed by bounding the Sobolev norm of the solution of the PDE in terms of the Sobolev norm of the boundary data and the right hand side. For a description of the general spaces and their properties, see [AF03 ; MS11].

Let $\Omega \subset \mathbb{R}^d$, $d = \{1\,, 2\,, 3\}$, be a bounded domain with smooth boundary. The space $L^2(\Omega)$ is defined as the space of square integrable functions,

$$L^2(\Omega) = \left\{ u : \|u\|_{L^2(\Omega)} = \int_\Omega \left(|u|^2 \, dx \right)^{1/2} < \infty \right\}.$$

Let k be a positive integer. The Sobolev space $H^k(\Omega)$ is the Hilbert space of functions with weak derivatives of all orders less than and equal to k in the space $L^2(\Omega)$. In particular, the inner product on $H^1(\Omega)$ is

$$(u, v)_{H^1(\Omega)} = \int_\Omega uv \, dx + \int_\Omega \nabla u \cdot \nabla v \, dx.$$

The H^1-seminorm and norm of $u \in H^1(\Omega)$ are respectively,

$$|u|^2 = \int_\Omega |\nabla u|^2 \, dx \quad \text{and} \quad \|u\|_{H^1(\Omega)} = |u|^2 + \|u\|_{L^2(\Omega)}\,.$$

Of particular interest for domain decomposition methods is the scaled norm obtained by dilation of a domain of unit diameter,

$$\|u\|^2_{H^1(\Omega)} = |u|_{H^1(\Omega)} + \frac{1}{diam(\Omega)^2} \|u\|^2_{L^2(\Omega)}\,, \tag{A.1}$$

where $diam(\Omega)$ is the diameter of Ω.

The Sobolev spaces can also be defined as the closure of $C^\infty(\Omega)$ in the corresponding norm, e.g., $H^1(\Omega)$ is the closure of $C^\infty(\Omega)$ with respect to $\|\cdot\|_{H^1(\Omega)}$. Let $C_0^\infty(\Omega) \subset C^\infty(\Omega)$ be the set of smooth functions with compact support. The subspace $H_0^1(\Omega) \subset H^1(\Omega)$ is the closure of $C^\infty(\Omega)$ with respect to $\|\cdot\|_{H^1(\Omega)}$, and consists of all the functions from $H^1(\Omega)$ which vanish on $\partial\Omega$ in the L^2 sense.

Trace Theorems

The trace theorems are results concerning the restriction of elements of Sobolev spaces on a domain to the boundary of the domain. Their duals are the extension theorems. The following

trace theorem will be useful later on ; see [AF03] for the general theory.

Theorem A.1. *If Ω is a Lipschitz domain and $u \in H^s\Omega\,,\ 1/2 < s \leq 1$, then*

$$\gamma_0 u = u_{|\partial\Omega} \in H^{s-1/2}(\partial\Omega).$$

Moreover, the restriction operator from $H^s(\Omega)$ to $H^{s-1/2}(\partial\Omega)$ is onto and continuous,

$$\|\gamma_0 u\|_{H^{s-1/2}(\partial\Omega)} \leq C(s,\Omega)\,\|u\|_{H^s(\Omega)}$$

The next theorem is a variant of Theorem A.1 for functions in $H^1(\Omega)$. We consider the norms given by (A.1), such that the dependence of the constants on the domain Ω can be specified.

Theorem A.2. *If Ω is a Lipschitz domain, then*

$$|u|^2_{H^{1/2}(\partial\Omega)} \leq C|u|_{H^1(\Omega)}$$

and

$$\|u\|^2_{L^2(\partial\Omega)} \leq C\left(diam(\Omega)|u|^2_{H^1(\Omega)} + \frac{1}{diam(\Omega)^2}\,\|u\|^2_{L^2(\Omega)}\right).$$

Poincaré and Friedrichs Inequalities

The Poincaré and Friedrichs inequalities provide simple equivalent norms for spaces like H^1_0 and H^1, and are used to derive convergence and condition number estimates for finite element methods. They can be proven using the Rellich compactness theorem and the completeness of Sobolev spaces, see [Cia78]. We are interested in formulations of the inequalities specifying the dependence of the constants on the domain Ω. Let $\widehat{\Omega} \subset \mathbb{R}^d\,,\ d = \{2\,,3\}$ be a reference Lipschitz domain of unit diameter and let Ω be a domain of diameter $diam(\Omega)$ obtained by a uniform dilation of $\widehat{\Omega}$.

Theorem A.3 (Poincaré Inequality)**.** *There exists a constant C that depends only on $\widehat{\Omega}$ such that*

$$\|u\|^2_{L^2(\Omega)} \leq C\left(diam(\Omega)^2|u|^2_{H^1(\Omega)} + \frac{1}{c^2 diam(\Omega)^d}\Big|\int_\Omega u\,dx\Big|^2\right), \quad \forall u \in H^1(\Omega)$$

Theorem A.4 (Friedrichs Inequality)**.** *Let $c > 0$ and let $\Lambda \subset \partial\Omega$ such that $c\mu(\partial\Omega) \leq \mu(\Lambda)$, where μ is the Lebesgue measure. Then,*

$$\|u\|^2_{L^2(\Omega)} \leq C\left(diam(\Omega)^2|u|^2_{H^1(\Omega)} + \frac{1}{c^2 diam(\Omega)^{d-2}}\Big|\int_\Lambda u\,d\sigma\Big|^2\right)$$

where C is a constant that does not depend on $u\,,\ \Omega\,,\ \Lambda\,,$ or c.

B Finite Element Approximations

Triangulations

Let $\Omega \subset \mathbb{R}^d\,,\ d = 2\,,3\,,$ be a bounded polygonal or polyhedral domain with Lipschitz continuous boundary. A triangulation (or, equivalently mesh) is a nonoverlapping partitions of Ω into elements. We consider meshes consisting of triangles or affinely mapped rectangles in two dimensions, and of tetrahedra or affinely mapped parallelepipeds in three dimensions. More precisely, let the reference triangle (tetrahedron) have vertices $(0\,,0)\,,\ (0\,,1)\,,\ (1\,,1)$ or $((0\,,0\,,0)\,,\ (0\,,0\,,1)\,,\ (0\,,1\,,0)\,,\ (1\,,0\,,0)\,,$ respectively). The reference square and cube are $(-1\,,1)^d$. Troughout of this monograph, a reference element $\hat{K}$ is one of the four regions defined above and elements are always open sets. An affine mapping from $\hat{K}$ onto an element K is defined by

$$F_K : \hat{K} \longrightarrow K, \quad F_K(x) = B_K x + \mathbf{b}_K,$$

with B_K a linear mapping and $\mathbf{b}_K$ a constant vector. We define a family of triangulations $\mathcal{T}_h\,,\ h > 0$:

Definition B.1. Let $h > 0$. A family of triangulations of Ω is a partition of Ω

$$\mathcal{T}_h = \big\{K = F_K(\hat{K})\big\},$$

such that, $\bigcup_{K\in\mathcal{T}_h} \overline{K} = \overline{\Omega}$; $\quad K \cap K' = \emptyset \quad$ if $K \neq K'$; $\quad \hat{K}$ is a reference element and F_K is an affine mapping ; $h = \max_{K\in\mathcal{T}_h} h_K, \quad h_K = diam(K)$. h is called the diameter of of $\mathcal{T}_h$. The family $\mathcal{T}_h$ is called geometrically conforming(briefly, conforming), if the intersection between the closure of two differents elements is either empty, a vertex, an edge, or a face that is common to both elements.

We consider particular triangulations.

Definition B.2. A family of triangulations $\mathcal{T}_h$ is called shape-regular if there exists a constant independent of h, such that $h_K \leq C\rho_K, \quad K \in \mathcal{T}_h$, where ρ_K is the radius of the largest circle or sphere containing K. The ratio h_K/ρ_K is called the aspect ratio of K.

Definition B.3. A family of triangulations $\mathcal{T}_h$ is called quasi-uniform if it is shape-regular and if there exists a constant independent of h, such that $h_K \geq Ch, \quad K \in \mathcal{T}_h$.

Finite Element Spaces

Given an open set $\mathcal{D} \in \mathbb{R}^d\,,\ d = 1\,,2\,,3\,,$ we now define some polynomial spaces. Let $\mathbb{P}_k(\mathcal{D})\,,\ k \geq 0$, be the set of polynomials of total degree at most k defined on $\mathcal{D}$, and let $\mathbb{P}_k(\mathcal{D})^d$ for $d = 2\,,3\,,$ be the set of vector of $\mathbb{R}^d$, the components of which belong to $\mathbb{P}_k(\mathcal{D})$. In addition, let $\mathcal{Q}_k(\mathcal{D})$ be the set of polynomials of degree at most k in each variable. Let $\mathcal{T}_h$ be a conforming triangulation. We have the following result, cf., e.g., [QV08, Pr. 3.2.1].

Lemma B.4. *A function $u : \Omega \longrightarrow \mathbb{R}$ belongs to $H^1(\Omega)$ if and only if the restriction of u to every $K \in \mathcal{T}_h$ belongs to $H^1(K)$, and for each common face(or edge in two dimensions) $\overline{f} = \overline{K_1} \cap \overline{K_2}$, we have $u_{|_{K_1}} = u_{|_{K_2}}$, on f.*

Finite element spaces of continuous, piecewise polynomial functions are therefore contained in $H^1(\Omega)$. For $k \geq 1$, we define (see [QV08, Sect. 3.2])

$$V^h = V_k^h(\Omega) := \Big\{ u \in C^0(\Omega),\ u_{|_K} \in \mathbb{P}_k(K),\ K \in \mathcal{T}_h \Big\}, \qquad V_0^h = V_{k;0}^h(\Omega) := V_k^h(\Omega) \cap H_0^1(\Omega),$$

if $\mathcal{T}_h$ consists of triangles or tetrahedra, and

$$V^h = V_k^h(\Omega) := \Big\{ u \in C^0(\Omega),\ u_{|_K} \in \mathbb{Q}_k(K),\ K \in \mathcal{T}_h \Big\}, \qquad V_0^h = V_{k;0}^h(\Omega) := V_k^h(\Omega) \cap H_0^1(\Omega),$$

if $\mathcal{T}_h$ is made of affinely mapped rectangles or parallelepipeds.

For a fixed polynomial degree k, the set of Lagrangian basis functions ϕ_i^h associated to a set of nodes $\{P_i\}$ of the triangulation can be introduced. The degree of freedom are then the values of a function at these nodes. We have

$$u(x) = \sum_i u(P_i)\phi_i^h(x), \quad u \in V^h,$$

and the basis functions are uniquely defined by $\phi_i(P_j) = \delta_{ij}$.

There is of course a one-to-one correspondence between functions in V^h and vectors of degrees of freedom. Troughout this monograph, we use the same notation of finite element functions u and vectors of degrees of freedom, and for finite element spaces and spaces of vectors of degrees of freedom. The support of the nodal basis function ϕ_i^h is contained in the union of the elements that share the node P_i. A scaling argument allows us to prove the following property ; see [QV08, Prop. 3.4.1].

Lemma B.5. *Let ϕ_i^h be a basis function associated to a node of $K \in \mathcal{T}_h$. Then there exists constants independent of h_K, and h, such that*

$$\begin{aligned} c_1 h_K^d &\leq \left\|\phi_i^h\right\|^2_{L^2(K)} \leq C_1 h_K^d \\ c_2 h_K^{d-2} &\leq |\phi_i^h|^2_{H^1(K)} \leq C_2 h_K^{d-2} \\ c_2 h_K^{d-1} &\leq |\phi_i^h|^2_{H^{1/2}(K)} \leq C_3 h_K^{d-2} \end{aligned}$$

where C_1 is independent of the aspect ratio of K.

A nodal interpolation operator $I^h = I_K^h$ can be defined for functions that are continuous in $\overline{\Omega}$ by

$$I^h u = \sum_i u(P_i)\phi_i^h, \quad u \in C^0(\overline{\Omega})$$

Error estimates can be found ; see [QV08, Sect. 3.4.1].

Lemma B.6. *Given a mesh $\mathcal{T}_h$, for $u \in H^s(\Omega)$ and $K \in \mathcal{T}_h$, $\frac{d}{2} < s \leq k+1$, $0 \leq m \leq s$, there exists a constant, depending on m , s and the aspect ratio of K, such that,*

$$|u - I_k^h u|_{H^m(K)} \leq C h_K^{s-m} |u|_{H^s(K)}.$$

If $\mathcal{T}_h$ is conforming and shape-regular, we have

$$|u - I_k^h u|_{H^m(K)} \leq C h^{s-m} |u|_{H^s(K)}.$$

We also need some finite element spaces that consist of discontinuous functions and are conforming in $L^2(\Omega)$. For $k \geq 0$, see [QV08, Sect. 3.2],

$$\begin{aligned} Q^h &= Q_k^h(\Omega) := \Big\{ u \in L^2(\Omega) \; u_{|_K} \in \mathbb{P}_k(K), \; K \in \mathcal{T}_h \Big\}, \\ Q_0^h &= Q_{k,0}^h(\Omega) := Q_k^h(\Omega) \cap L_0^2(\Omega), \end{aligned}$$

if $\mathcal{T}_h$ is made of triangles or tetrahedra, and an analogous definition holds if $\mathcal{T}_h$ is made of rectangles or parallelepipeds.

Positive Definite Problems

Let $\Omega \subset \mathbb{R}^d$, $d = 2\,, 3$, be a Lipschitz region of unit diameter. We consider a general second order elliptic scalar partial differential equation involving the operator : find $u \in H_0^1(\Omega)$ such that

$$-\sum_{i,j=1}^{d} \frac{\partial}{\partial x_j} \left(a_{ij}(x) \frac{\partial u}{\partial x_i} \right) = f \;\text{ in }\; \Omega, \qquad u = 0 \;\text{ on }\; \partial\Omega. \tag{B.1}$$

The weak form of (B.1) is given by : find $u \in H_0^1(\Omega)$ such that

$$a(u, v) = (f, v), \quad v \in H_0^1(\Omega) \tag{B.2}$$

where $a(u,v) = \displaystyle\sum_{i,j=1}^{d} \int_\Omega a_{i,j}(x) \frac{\partial u}{\partial x_j} \frac{\partial u}{\partial x_i} \, dx$ and $(f, v) = \displaystyle\int_\Omega f v \, dx.$

We consider the finite element spaces $V^k = V_k^h(\Omega)$ and $V_0^h = V_{k;0}^h(\Omega)$, defined in section . Given the variational formulation (B.2), we consider a conforming approximation in the subspace V_0^h : find $u \in V_0^h$, such that,

$$a(u_h, v_h) = (f, v_h), \quad v_h \in V_0^h. \tag{B.3}$$

The weel-posedness of (B.3) is ensured by the Lax-Milgram Lemma, see [QV08].

Theorem B.7. *Let $\mathcal{T}_h$ be a conforming triangulation of Ω and $k \geq 1$. Then, The problem* (B.3) *is weel-posed : there exists a unique solution such that*

$$\|u_h\|_{H^1(\Omega)} \leq C_1 \|f\|_{H^{-1}(\Omega)}, \quad \|u_h\|_a \leq C_2 \|f\|_{H^{-1}(\Omega)}$$

where $\|\cdot\|$ is the norm associated to the bilinear form $a(\cdot,\cdot)$. The finite element solution satisfies

$$a(u - u_h, v_h) = 0, \quad v_h \in V_0^h \tag{B.4}$$

If the mesh $\mathcal{T}_h$ is shape-regular, we have

$$|u - u_h|_{H^1(\Omega)} \leq Ch^{s-1}|u|_{H^s(\Omega)}, \quad \frac{n}{2} < s \leq k+1$$

and if, in addition Ω is convex

$$\|u - u_h\|_{L^2(\Omega)} \leq Ch^s|u|_{H^s(\Omega)}.$$

C Solution of Algebraic Linear Systems

We consider the solution of linear systems

$$Au = b \tag{C.1}$$

with $u, b \in \mathbb{R}^n$, and A an $n \times n$, real, invertible matrix. We use the notation $\langle u\,, v\rangle = u^t v$, for $u, v \in \mathbb{R}^n$.

Eigenvalues and Condition Number

We recall that, given matrix $A \in \mathbb{R}^n \times \mathbb{R}^n$, its eigenvalues $\lambda \in \mathbb{C}$ and eigenvectors $u \in \mathbb{C}^n \backslash \{0\}$ are solution of

$$Au = \lambda u.$$

The set of eigenvalues of A, also called spectrum, is denoted by $\sigma(A)$. The spectrum radius $\rho(A)$ is defined as

$$\rho(A) := \max_{\lambda \in \sigma(A)} \{|\lambda|\}.$$

Given a matrix norm $\|\cdot\|$, we define the condition number of an invertible matrix A by

$$\kappa(A) := \|A\| \left\|A^{-1}\right\|.$$

In the same way, given a second matrix M, we can consider the generalized eigenproblem

$$Au = \lambda Mu.$$

A matrix A is said positive definite if all its eigenvalues have positive real part or, equivalently, if $u^t Au$ has positive real part for $u \in \mathbb{C}\backslash\{0\}$. We note that in this case

$$u^t Au = u^t \frac{A + A^t}{2} > 0, \quad u \in \mathbb{R}^n\backslash\{0\}. \tag{C.2}$$

If in addition A is symmetric, then its eigenvalues are real and strictly positive. Troughout this section, we make use of the following property.

Lemma C.1. *Let A and M be two symmetric, positive definite matrices of order n. For an arbitrary matrix $B \in \mathbb{R}^{n\times n}$, let*

$$\|B\|_A := \sup_{u\in\mathbb{R}^n} \frac{\|Bu\|_A}{\|u\|_A},$$

with $\|u\|_A^2 := u^t Au$, and similarly for $\|B\|_M$. Then,

1. *The following eigenvalue problems have the same n eigenvalues*

$$Au = \lambda Mu, \tag{C.3}$$
$$M^{-1}Au = \lambda u, \tag{C.4}$$
$$(M^{-1/2}AM^{-1/2})u = \lambda u, \tag{C.5}$$
$$(A^{1/2}M^{-1}A^{1/2})u = \lambda u. \tag{C.6}$$
$$\tag{C.7}$$

   They are all real and strictly positive.

The smallest and largest eigenvalues of the problems above satisfy

$$\lambda_{min} = \inf_{u\in\mathbb{R}^n} \frac{u^t Au}{u^t Mu}, \quad \lambda_{max} = \sup_{u\in\mathbb{R}^n} \frac{u^t Au}{u^t Mu}. \tag{C.8}$$

We have

$$\left\|M^{-1}A\right\|_A = \left\|M^{-1}A\right\|_M = \lambda_{max} = \rho(M^{-1}A),$$
$$\left\|(M^{-1}A)^{-1}\right\|_A = \left\|(M^{-1}A)^{-1}\right\|_M = 1/\lambda_{min},$$

and thus

$$\kappa_A(M^{-1}A) = \kappa_2(M^{-1/2}AM^{-1/2}) = \lambda_{max}/\lambda_{min}$$

We have

$$(u^t Au \geq cu^t Mu,\ u \in \mathbb{R}^n) \longrightarrow \lambda_{min} \geq c.$$

Analogously,

$$(u^t Au \leq Cu^t Mu,\ u \in \mathbb{R}^n) \longrightarrow \lambda_{max} \leq C,$$

and thus

$$\kappa_A(M^{-1}A) \leq C/c.$$

We use the notation

$$\kappa M^{-1}A = \kappa_A M^{-1}A = \kappa_M M^{-1}A.$$

We note that since $M^{-1}A$ is not symmetric, its norm $\|M^{-1}A\|_2$ is not in general equal to the largest eigenvalue. However, $M^{-1}A$ is symmetric with respect to the scalar product induced by A and M.

The Lemma C.1 allows to prove the following corollary. It basically ensures that a good preconditioner of a good preconditioner remains a good preconditioner.

Corollary C.2. *Let A, B, and C be three positive definite symmetric matrices. Then,*

$$\kappa(C^{-1}A) \leq \kappa(C^{-1}B)\kappa(B^{-1}C)$$

D Algorithms for Matrix Square Root

The edge block of substructuring preconditioners described in chapter 6 involves the square root of matrices, see (4.42). As explained above, we use the Preconditioned Conjugate Gradient(PCG) method for solving the Schur complement system (4.41). Then, each iteration of PCG requires the solution of the system

$$\widehat{\mathbf{K}}_{\mathsf{E}} z = g, \qquad \widehat{\mathbf{K}}_{\mathsf{E}} = \mathbf{M}_{\mathsf{E}}^{1/2}\left(\mathbf{M}_{\mathsf{E}}^{-1/2}\mathbf{R}_{\mathsf{E}}\mathbf{M}_{\mathsf{E}}^{-1/2}\right)^{1/2}\mathbf{M}_{\mathsf{E}}^{1/2}. \tag{D.1}$$

We can work with the lumped mass matrix $\mathbf{M}_{\mathsf{E},L}$ instead of $\mathbf{M}_{\mathsf{E}}$, hence in (D.1) we can substitute $\mathbf{M}_{\mathsf{E}} = (m_{ij})$ with $\mathbf{M}_{\mathsf{E},L}$ defined as

$$\mathbf{M}_{\mathsf{E},L} = diag(m_{ii}^L), \qquad m_{ii}^L = \sum_{ij} m_{ij}. \tag{D.2}$$

Therefore we may compute $\widetilde{\mathbf{K}} = \mathbf{M}_{\mathsf{E},L}^{1/2}\mathbf{R}_{\mathsf{E}}\mathbf{M}_{\mathsf{E},L}^{1/2}$ and then $\widehat{\mathbf{K}}_{\mathsf{E}} = \widetilde{\mathbf{K}}^{1/2}$.

Remark D.1. When high order fem are used, the process of mass-lumping (D.2) may produce singular matrices. More specifically, the process of the mass-lumping can be applied to the two-dimensional case(i.e. when the interface is a face) when linear elements are used. For

quadratic finite elements, the above procedure would generate a singular mass matrix $\mathbf{M}_{\mathsf{E},L}$. An alternative diagonalization strategy consists in using the matrix

$$\widehat{\mathbf{M}} = diag(\hat{m}_{ii}), \qquad \hat{m}_{ii} = \frac{m_{ii}}{\sum_j m_{jj}}. \tag{D.3}$$

In the one-dimensional case(i.e. the interface is an edge), for linear and quadratic finite elements, matrices $\widehat{\mathbf{M}}$ and $\mathbf{M}_{\mathsf{E},L}$ coincide, while they differ for cubic elements. Note that $\widehat{\mathbf{M}}$ is non-singular also for Lagrangian finite elements of high order, while it can turn out to be singular when using non-lagrangian finite elements, for instance when using hierarchical basis, see [Qua10] for more details.

In the case of use of lagrangian basis functions, we consider the standard lumped mass matrix $\mathbf{M}_{\mathsf{E},L}$ defined in (D.2). Then we compute $\widetilde{\mathbf{K}}^{1/2}$ once, before the application of PCG, by using standard technique such as `sqrt` in EIGEN [G+10] library. Otherwise we may apply the technique proposed in [HHT08] and compute, for each PCG iteration, the action $\widetilde{\mathbf{K}}^{-1/2}$ that is

$$z = \widetilde{\mathbf{K}}^{-1/2} g. \tag{D.4}$$

The technique proposed in [HHT08] is based on conbining contour integrals evaluated by periodic trapezoid rule with conformal maps involving Jacobi elliptic functions. This procedure is particularly effective when the matrix $\widetilde{\mathbf{K}}$ is such that the systems of equations $\left(zI - \widetilde{\mathbf{K}}\right) = b$ b can be solved efficiently by sparse direct methods. An example is the matrix associated to the finite element discretization of the Laplacian in 2D or 3D, that is our case.

The procedure is iterative and requires an approximation of the minimum and maximum eigenvalues of $\widetilde{\mathbf{K}}$ and the solution of N systems of equations with matrix $\left(zI - \widetilde{\mathbf{K}}\right)$. N has to be chosen depending on the accuracy wanted.

E Proofs of Theorems and Lemmas

Proof of Lemma 3.4.3. For each edge $\Gamma_{\ell n}$ we introduce the constant

$$\bar{\eta}_{\ell,n} = \frac{1}{|\Gamma_{\ell n}|}\int_{\Gamma_{\ell n}} \eta_\ell = \frac{1}{|\Gamma_{\ell n}|}\int_{\Gamma_{\ell n}} \eta_n,$$

(the last identity is a consequence of (3.32)). For $\gamma_\ell^{(i)} = \Gamma_{\ell n}$ we also introduce the notation $\bar{\eta}_\ell^{(i)} = \bar{\eta}_{\ell,n}$. We have

$$\begin{aligned}
\sigma(\bar{\eta},\bar{\eta}) &= \sum_{\ell,n:|\Gamma_{\ell n}|>0} |\bar{\eta}_\ell - \bar{\eta}_{\ell,n} - (\bar{\eta}_n - \bar{\eta}_{\ell,n})|^2 \lesssim \sum_\ell \sum_{n:|\Gamma_{\ell n}|>0} |\bar{\eta}_\ell - \bar{\eta}_{\ell,n}|^2 \\
&= \sum_\ell \sum_{i\in\mathcal{E}_\ell} |\bar{\eta}_\ell - \bar{\eta}_\ell^{(i)} + \eta(x_i^\ell) - \eta(x_i^\ell)|^2 \\
&\lesssim \sum_\ell \sum_{i\in\mathcal{E}_\ell} |\eta(x_i^\ell) - \bar{\eta}_\ell|^2 + \sum_\ell \sum_{i\in\mathcal{E}_\ell} |\eta(x_i^\ell) - \bar{\eta}_\ell^{(i)}|^2,
\end{aligned}$$

where, for each ℓ, we let $\mathcal{E}_\ell = \{i : \ \gamma_\ell^{(i)} \text{ is an interior edge}\}$.

We have

$$|\bar{\eta}_\ell - \eta(x_i^\ell)|^2 \lesssim \|\eta - \bar{\eta}_\ell\|^2_{L^\infty(\Gamma_\ell)}.$$

We observe that $\displaystyle\int_{\partial\Omega_\ell} \eta_\ell - \bar{\eta}_\ell = 0$, which, since $\eta_\ell - \bar{\eta}_\ell \in C^0(\partial\Omega_\ell)$, implies that $\eta_\ell - \bar{\eta}_\ell$ vanishes at some point of $\partial\Omega_\ell$. We can then apply bound (3.25), which yields

$$|\bar{\eta}_\ell - \eta(x_i^\ell)|^2 \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right) |\eta_\ell|^2_{1/2,\partial\Omega_\ell}.$$

The term $|\eta_\ell(x_i^\ell) - \bar{\eta}_\ell^{(i)}|^2$ is bound analogously. The thesis is obtained since the cardinality of the set $\mathcal{E}_\ell$ is bounded. □

Proof of Lemma 3.6.4. We have

$$\begin{aligned} |\pi_h(\eta)|_T^2 &\lesssim \sum_{m=(\ell,i)\in I} |\pi_m([\eta])|^2_{H_{00}^{1/2}(\gamma_m)} \\ &\lesssim \sum_{m=(\ell,i)\in I} H_\ell^{2\varepsilon} p_\ell^{4\varepsilon} h_\ell^{-2\varepsilon} |\pi_m([\eta])|^2_{H_0^{1/2-\varepsilon}(\gamma_m)} \\ &\lesssim \hat{p}^{3/2} \sum_{m=(\ell,i)\in I} h_\ell^{-2\varepsilon} H_\ell^{2\varepsilon} p_\ell^{4\varepsilon} |[\eta]|^2_{H_0^{1/2-\varepsilon}(\gamma_m)}. \end{aligned} \tag{E.1}$$

We now observe that, for $m = (\ell\,, i) \in I$, $\gamma_\ell^{(i)} = \Gamma_{\ell,n}$ we have (see (3.22))

$$|[\eta]|^2_{H_0^{1/2-\varepsilon}(\gamma_m)} \lesssim \frac{1}{\varepsilon^2} \|[\eta - \alpha]\|^2_{H^{1/2}(\gamma_m)} + \frac{1}{\varepsilon} |\alpha_\ell - \alpha_n|^2. \tag{E.2}$$

Then we obtain

$$|\pi_h(\eta)|_T^2 \lesssim \hat{p}^{3/2} \frac{H^{2\varepsilon} p^{4\varepsilon}}{h^{2\varepsilon}} \left(\frac{1}{\varepsilon^2} \sum_{m=(\ell,i)\in I} \|[\eta - \alpha]\|^2_{H^{1/2}(\gamma_m)} + \frac{1}{\varepsilon} \sigma(\alpha, \alpha) \right).$$

Observing that, for $\gamma_m = \Gamma_{\ell,n}$ it holds that

$$\|[\eta - \alpha]\|^2_{H^{1/2}(\gamma_m)} \leq \|\eta_\ell - \alpha_\ell\|^2_{H^{1/2}(\gamma_m)} + \|\eta_n - \alpha_n\|^2_{H^{1/2}(\gamma_m)},$$

by choosing $\varepsilon = 1/\log(Hp^2/h)$, we get

$$|\pi_h(\eta)|_T^2 \lesssim \hat{p}^{3/2} \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2 \|\eta - \alpha\|_T^2 + \hat{p}^{3/2} \left(1 + \log\left(\frac{Hp^2}{h}\right)\right) \sigma(\alpha, \alpha).$$

The bound (3.46) follows easily by observing that

$$|(Id - \pi_h(\eta))|_T^2 \lesssim |\eta|_T^2 + |\pi_h(\eta)|_T^2 = |\eta - \alpha|_T^2 + |\pi_h(\eta)|_T^2 \lesssim \|\eta - \alpha\|_T^2 + |\pi_h(\eta)|_T^2.$$

The bound (3.47) is obtained by observing that, if each η_ℓ is linear on each $\gamma_\ell^{(i)}$, thanks to (3.23), the bound (E.2) can be improved to

$$|[\eta]|^2_{H_0^{1/2-\varepsilon}(\gamma_m)} \lesssim \frac{1}{\varepsilon}(\|[\eta - \alpha]\|^2_{H^{1/2}(\gamma_m)} + |\alpha_\ell - \alpha_n|^2).$$

□

Proof of Theorem 4.2.1. By using (4.15) we get

$$s(\eta, \eta) \lesssim |\eta^E|_T^2 + |\eta^V|_T^2 \lesssim \hat{p}^{3/2} b^E(\eta^E, \eta^E) + |\eta^V|_T^2. \tag{E.3}$$

Concerning $|\eta^V|_T^2$, let $\eta_\delta^V = (1 - \pi_\delta)\Lambda\eta$. We have $\eta^V = (1 - \pi_h)\Lambda\eta_\delta^V$. We introduce $\tilde{\eta} = (\tilde{\eta}_\ell)_{\ell=1,\dots,L} \in T$ with $\tilde{\eta}_\ell$ constant on $\partial\Omega_\ell$ defined as

$$\tilde{\eta}_\ell = |\partial\Omega_\ell|^{-1} \int_{\partial\Omega_\ell} \eta_\delta^V.$$

Using Lemma 3.4.3 and (4.21), as well as (3.21), we have ($\Lambda\tilde{\eta} = \tilde{\eta}$)

$$\begin{aligned}
|\eta^V|_T^2 = |(Id - \pi_h)\Lambda\eta_\delta^V|_T^2 &\lesssim \hat{p}^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)\left(\|\Lambda(\eta_\delta^V - \tilde{\eta})\|_T^2 + \sigma(\tilde{\eta}, \tilde{\eta})\right) \\
&\lesssim \hat{p}^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)(1 + \log(\mathsf{n}_c))\left(\|\eta_\delta^V - \tilde{\eta}\|_T^2 + |\eta_\delta^V|_T^2\right) \\
&\lesssim \hat{p}^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right)(1 + \log(\mathsf{n}_c))\,|\eta_\delta^V|_T^2 \\
&\lesssim \hat{p}^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right) b_1^V(\eta^V, \eta^V),
\end{aligned}$$

where the last bound holds since n_c is a constant independent of h , p and H. Then we have

$$s(\eta, \eta) \lesssim |\eta^E|_T^2 + |\eta^V|_T^2 \lesssim \hat{p}^{3/2} b^E(\eta^E, \eta^E) + p^{3/2}\left(1 + \log\left(\frac{Hp^2}{h}\right)\right) b_1^V(\eta^V, \eta^V) = p^{3/2} s_1(\eta, \eta),$$

that is the first part of the theorem.

$$s(\eta, \eta) \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2 \hat{s}(\eta, \eta).$$

Let us now bound $s_1(\eta\,,\eta)$ in terms of $s(\eta\,,\eta)$. We have, for $\bar{\eta}$ defined by (3.31),

$$\begin{aligned} b_1^V(\eta^V,\eta^V) &\lesssim |(Id-\pi_\delta)\Lambda\eta|_T^2 \lesssim (1+\log(\mathsf{n}_c))\,(\|\Lambda(\eta-\bar{\eta})\|_T^2+\sigma(\bar{\eta},\bar{\eta})) \\ &\lesssim (1+\log(\mathsf{n}_c))\left(1+\log\left(\frac{Hp^2}{h}\right)\right)|\eta|_T^2 \lesssim \left(1+\log\left(\frac{Hp^2}{h}\right)\right)s(\eta,\eta), \end{aligned}$$

where we used (4.20) and (3.21).

Thanks to (4.14) and the definition of (4.24) we get that

$$s_1(\eta,\eta)=b^E(\eta^E,\eta^E)+\left(1+\log\left(\frac{Hp^2}{h}\right)\right)b_1^V(\eta^V,\eta^V)\lesssim\left(1+\log\left(\frac{Hp^2}{h}\right)\right)^2 s(\eta,\eta)$$

that concludes the proof of the Theorem 4.2.1. □

Proof of Theorem 4.2.3. Thanks to Lemma 4.1.2 we have

$$b_\#^V(\eta^V,\eta^V)\lesssim\left(1+\log\left(\frac{Hp^2}{h}\right)\right)|\eta|_T^2.$$

Let us then bound $b_{[\,]}^V(\eta^V\,,\eta^V)$. For each slave side γ_m with $\gamma_m=\Gamma_{\ell n}$ we introduce the constant

$$\bar{\eta}_m=\frac{1}{|\gamma_m|}\int_{\gamma_m}\eta_\ell^V=\frac{1}{|\gamma_m|}\int_{\gamma_m}\eta_n^V$$

(the last identity is a consequence of the weak continuity constraint). For $\gamma_m=\gamma_\ell^{(i)}=\gamma_n^{(j)}$, we also introduce the notation $\bar{\eta}_\ell^{(i)}=\bar{\eta}_n^{(j)}=\bar{\eta}_m$.

Letting a_m and b_m denote the two extrema of γ_m we can write

$$b_{[\,]}^V(\eta^V,\eta^V)=\sum_{m\in I}|\gamma_m|^{-1}\int_{\gamma_m}|[\Lambda\eta]|^2\simeq\sum_{m\in I}(|[\Lambda\eta](a_m)|^2+|[\Lambda\eta](b_m)|^2).$$

Observing that for $(\ell\,,i)$, $(n\,,j)$ such that $\gamma_m=\gamma_\ell^{(i)}=\gamma_n^{(j)}$ and for $x\in\bar{\gamma}_m$ we have that

$$|[\Lambda\eta](x)|^2=|\eta_\ell(x)-\eta_n(x)|^2=|\eta_\ell(x)-\bar{\eta}_\ell^{(i)}-(\eta_n(x)-\bar{\eta}_n^{(j)})|^2\lesssim|\eta_\ell(x)-\bar{\eta}_\ell^{(i)}|^2+|\eta_n(x)-\bar{\eta}_n^{(j)}|^2,$$

we immediately obtain that

$$b_1^V(\eta^V,\eta^V)\lesssim\sum_\ell\sum_{i=1}^4|\eta(x_i^\ell)-\bar{\eta}_\ell^{(i)}|^2.$$

Now, reasoning as in the proof of Lemma 3.4.3 we obtain

$$|\eta_\ell(x_i^\ell) - \bar{\eta}_\ell(i)|^2 \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right) |\eta|^2_{H^{1/2}(\partial\Omega_\ell)}.$$

Putting all together we obtain

$$b_2^V(\eta^V, \eta^V) \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right) s(\eta, \eta). \tag{E.4}$$

Combining (E.4), (4.14) with (4.30), we obtain

$$\hat{s}(\eta, \eta) \lesssim \left(1 + \log\left(\frac{Hp^2}{h}\right)\right)^2 s(\eta, \eta).$$

Let us now bound $s(\eta\,, \eta)$. We let $\bar{\eta} \in L^2(\Sigma)$ denote the (single valued) function assuming the value $\bar{\eta}_m$ on γ_m for $m \in I$. We have

$$s(\eta, \eta) \lesssim |\eta^V|^2_T + |\eta^E|^2_T.$$

Let us now consider $s(\eta^V\,, \eta^V)$. We have

$$s(\eta^V, \eta^V) = |\eta^V|^2_T = |(1 - \pi_h)\Lambda\eta|^2_T = |\Lambda\eta|^2_T + |\pi_h\Lambda\eta|^2_T.$$

We bound the two terms on the right hand side separately. We have (see [BPS86])

$$|\Lambda\eta|^2_T \lesssim \sum_\ell |\mathcal{H}_\ell\Lambda^\ell\eta|^2_{H^1(\Omega_\ell)} \lesssim b_\#^V(\eta^V, \eta^V).$$

As far as the second term is concerned, we can write

$$\begin{aligned}
|\pi_h(\Lambda\eta)|^2_T &\lesssim \sum_{m=(\ell,i)\in I} |\pi_m([\Lambda\eta])|^2_{H_{00}^{1/2}(\gamma_m)} \\
&\lesssim \sum_{m=(\ell,i)\in I} H_\ell^{2\varepsilon} p^{4\varepsilon} h_\ell^{-2\varepsilon} |\pi_m([\Lambda\eta])|^2_{H_0^{1/2-\varepsilon}(\gamma_m)} \\
&\lesssim \hat{p}^{3/2} \sum_{m=(\ell,i)\in I} h^{-2\varepsilon} H^{2\varepsilon} p^{3\varepsilon} \|[\Lambda\eta]\|^2_{H_0^{1/2-\varepsilon}(\gamma_m)} \\
&\lesssim \hat{p}^{3/2} \sum_{m=(\ell,i)\in I} \frac{H^{2\varepsilon} p^{3\varepsilon}}{h^{2\varepsilon}} \frac{1}{\varepsilon} \|[\Lambda\eta]\|^2_{H^{1/2-\varepsilon}(\gamma_m)} \\
&\lesssim \hat{p}^{3/2} \left(1 + \log\left(\frac{Hp^{3/2}}{h}\right)\right) \sum_{m=(\ell,i)\in I} \|[\Lambda\eta]\|^2_{H^{1/2-\varepsilon}(\gamma_m)}.
\end{aligned} \tag{E.5}$$

Now we have (recall that $\|\cdot\|_{L^2(\Gamma_\ell)}$ is the scaled L^2 norm)

$$\begin{aligned}\|[\Lambda\eta]\|^2_{H^{1/2-\varepsilon}(\gamma_m)} &= \|[\Lambda\eta]\|^2_{L^2(\gamma_m)} + |[\Lambda\eta]|^2_{H^{1/2-\varepsilon}(\gamma_m)} \\ &\lesssim \|[\Lambda\eta]\|^2_{L^2(\gamma_m)} + |[\Lambda\eta]|^2_{H^{1/2}(\gamma_m)} \lesssim |\gamma_m|^{-1}\|[\Lambda\eta]\|^2_{L^2(\gamma_m)},\end{aligned}$$

where the last inverse type inequality is obtained by a scaling argument and using the linearity of $\Lambda\eta$ on γ_m.

Combining the bounds on the two contributions we obtain

$$s(\eta^V,\eta^V) \lesssim \hat{p}^{3/2}\left(1+\log\left(\frac{Hp^2}{h}\right)\right) b_2^V(\eta^V,\eta^V).$$

which finally yields

$$s(\eta,\eta) \lesssim \hat{p}^{3/2} s_2(\eta,\eta).$$

□

F A Two Domain Overlapping Schwarz Method

We consider the following laplacian boundary value problem

$$\begin{cases} -\Delta u = f & \text{in} \quad \Omega \\ \quad u = g & \text{on} \quad \partial\Omega \end{cases} \tag{F.1}$$

where $\Omega \subset \mathbb{R}^d\,, d = 2\,, 3$ and g is the dirichlet boundary value.

Schwarz Algorithms

The Schwarz overlapping multiplicative algorithm with dirichlet interface conditions for this problem on two subdomains Ω_1 and Ω_2 at n^{th} iteration is given by

$$\begin{cases} -\Delta u_1^n = f & \text{in} \quad \Omega_1 \\ \quad u_1^n = g & \text{on} \quad \partial\Omega_1^{ext} \\ \quad u_1^n = u_2^{n-1} & \text{on} \quad \Gamma_1 \end{cases} \quad \text{and} \quad \begin{cases} -\Delta u_2^n = f & \text{in} \quad \Omega_2 \\ \quad u_2^n = g & \text{on} \quad \partial\Omega_2^{ext} \\ \quad u_2^n = u_1^n & \text{on} \quad \Gamma_2 \end{cases} \tag{F.2}$$

Variational Formulations

$$\int_{\Omega_i} \nabla u_i \cdot \nabla v = \int_{\Omega_i} fv \quad \forall\, v,\ i = 1,2.$$

Feel++ Implementation

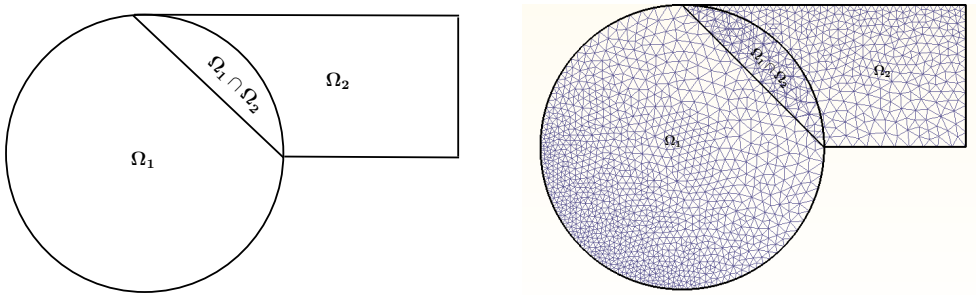
Listing F.1 : Example with 2 subdomains

```
template<Expr>
void
localProblem(element_type& u, Expr expr)
{
      // Assembly of the right hand side ∫_Ω fv
   auto F = backend->newVector(Xh);
   form1( _test=Xh,_vector=F, _init=true ) =
         integrate( elements(mesh), f*id(v) );
   F->close();
   // Assembly of the left hand side ∫_Ω ∇u · ∇v
   auto A = backend->newMatrix( Xh, Xh );
   form2( _test=Xh, _trial=Xh, _matrix=A, _init=true ) =
         integrate( elements(mesh), gradt(u)*trans(grad(v)) );
   A->close();
   // Apply the dirichlet boundary conditions
   form2( Xh, Xh, A ) +=
         on( markedfaces(mesh, "Dirichlet") ,u,F,g);
   // Apply the dirichlet interface conditions
   form2( Xh, Xh, A ) +=
         on( markedfaces(mesh, "Interface") ,u,F,expr);

   // solve the linear system Au = F
   backend->solve(_matrix=A, _solution=u, _rhs=F );
}

unsigned int cpt = 0;
double tolerance = 1e − 8;
double maxIterations = 20;
double l2erroru_1 = 1.;
double l2erroru_2 = 1;

// Iteration loop

while( (l2erroru_1 +l2erroru_2) > tolerance && cpt <= maxIterations)
 {
       // call the localProblem on the first subdomain Ω_1
       localProblem(u_1, idv(u_2));
       // call the localProblem on the first subdomain Ω_2
       localProblem(u_2, idv(u_1));
       // compute L2 errors on each subdomain
       L2erroru_1 = l2Error(u_1);
       L2erroru_2 = l2Error(u_2);
       // increment the cunter
       ++cpt;
 }
```

Numerical Results in 2D

The numerical results presented in the following table correspond to the partition of the global domain Ω in two subdomains Ω_1 and Ω_2 (see FIGURE F.1) and the following configuration :

1. $g(x\,,y) = \sin(\pi x)\cos(\pi y)$: the exact solution
2. $f(x\,,y) = 2\pi^2 g$: the right hand side of the equation
3. $\mathbb{P}_2$ approximation : the lagrange polynomial order
4. hsize $= 0\ .02$: the mesh size
5. tol $= 1e-9$: the tolerance



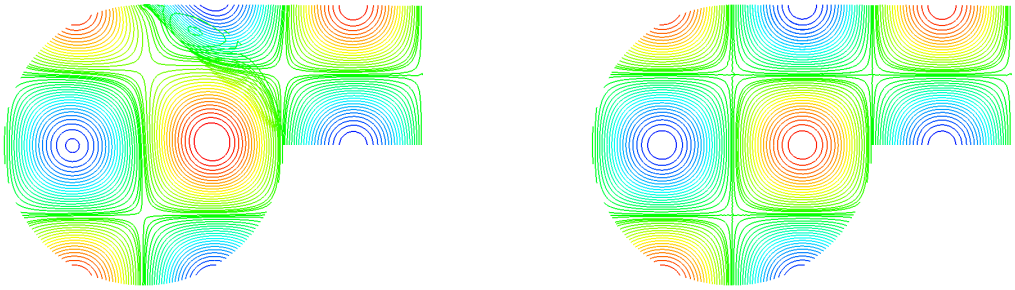
F.1.1. Two overlapping subdomains F.1.2. Two overlapping meshes

FIGURE F.1 : Geometry in 2D

TABLE F.1 : L^2 errors in Ω_1 and Ω_2

Nomber of iterations	$\|\mathbf{u_1} - \mathbf{u_{ex}}\|_{\mathbf{L_2}}$	$\|\mathbf{u_2} - \mathbf{u_{ex}}\|_{\mathbf{L_2}}$
11	2.52e-8	2.16e-8

Numerical Solutions in 2D



F.2.1. First iteration F.2.2. 10^{th} iteration

FIGURE F.2 : Isovalues of solution in 2D

G Eigenmodes of Dirichlet to Neumann Operator

Problem Description and Variational Formulation

We consider at the continuous level the Dirichlet-to-Neumann (DtN) map on Ω, denoted by DtN_Ω.

Let $u : \Gamma \longmapsto \mathbb{R}$,

$$\text{DtN}_\Omega(u) = \kappa \frac{\partial v}{n}\Big|_\Gamma$$

where v satisfies

$$\begin{cases} \mathcal{L}(v) := (\eta - \text{div}(\kappa\nabla))v = 0 & \text{dans} \quad \Omega, \\ v = u & \text{sur} \quad \Gamma \end{cases} \tag{G.1}$$

where Ω is a bounded domain of $\mathbb{R}^d$ (d=2 or 3), and Γ it border, κ is a positive diffusion function which can be discontinuous, and $\eta \geq 0$. The eigenmodes of the Dirichlet-to-Neumann operator are solutions of the following eigenvalues problem

$$\text{DtN}_\Omega(u) = \lambda\kappa u \tag{G.2}$$

To obtain the discrete form of the DtN map, we consider the variational form of (G.1). let's define the bilinear form $a : H^1(\Omega) \times H^1(\Omega) \longrightarrow \mathbb{R}$,

$$a(w, v) := \int_\Omega \eta w v + \kappa \nabla w \cdot \nabla v.$$

With a finite element basis $\{\phi_k\}$, the coefficient matrix of a Neumann boundary value problem in Ω is

$$A_{kl} := \int_\Omega \eta \phi_k \phi_l + \kappa \nabla \phi_k \cdot \nabla \phi_l.$$

A variational formulation of the flux reads

$$\int_\Gamma \kappa \frac{\partial v}{\partial n}\phi_k = \int_\Omega \eta v\phi_k + \kappa\nabla v\cdot\nabla\phi_k \quad \forall\, \phi_k.$$

So the variational formulation of the eigenvalue problem (G.2) reads

$$\int_\Omega \eta v\phi_k + \kappa\nabla v\cdot\nabla\phi_k = \lambda\int_\Gamma \kappa v\phi_k \quad \forall\, \phi_k. \tag{G.3}$$

Let B be the weighted mass matrix

$$(B)_{kl} = \int_\Gamma \kappa\phi_k\phi_l$$

The compact form of (G.3) is

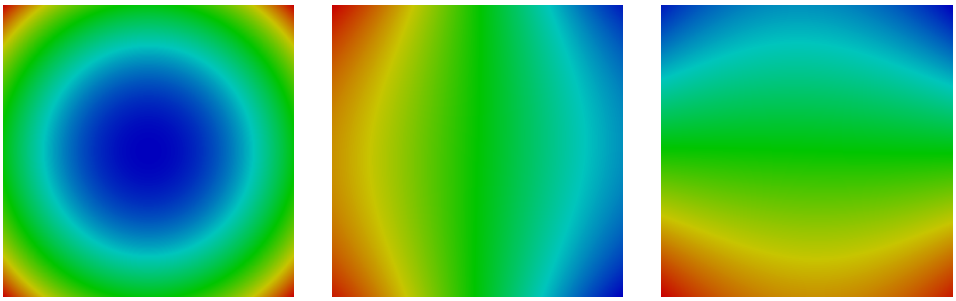
$$Av = \lambda Bv \tag{G.4}$$

Feel++ Implementation

Listing G.1 : Eigenvalue solver

```
// Assembly of the right hand side B = ∫_Γ κvw
auto B = backend->newMatrix( _test=Xh, _trial=Xh ) ;
form2( _test=Xh, _trial=Xh, _matrix=B, _init=true );
for( int const& marker : flags )
 {
       form2( _test=Xh, _trail=Xh, _matrix=B ) +=
             integrate( markedfaces(mesh,marker), κ*idt(u)*id(v) );
 }
B->close();
// Assembly of the left hand side A = ∫_Ω ηvw + κ∇v·∇w
auto A = backend->newMatrix( Xh, Xh ) ;
form2( _test=Xh, _trial=Xh, _matrix=A, _init=true ) =
      integrate( elements(mesh), κ*gradt(u)*trans(grad(v))
                           + ν*idt(u)*id(v) );
A->close();
// eigenvalue solver options
int nev = doption("solvereigen-nev");
int ncv = doption("solvereigen-ncv)";
// definition of the eigenmodes
SolverEigen<double>::eigenmodes_type modes;
// solve the eigenvalue problem Av = λBv
 modes=
     eigs( _matrixA=A,
          _matrixB=B,
          _nev=nev,
          _ncv=ncv,
          _transform=SINVERT,
          _spectrum=SMALLEST_MAGNITUDE,
          _verbose = true );
```

Numerical Solutions



G.1.1. First mode G.1.2. Second mode G.1.3. Third mode

Figure G.1 : Three eigenmodes

The numerical solutions in Figure G.1 correspond to the following configuration :

1. $\mathbb{P}_2$ approximation : the lagrange polynomial order
2. hsize $= 0\ .02$: the mesh size
3. $\mu = \kappa = 1$.

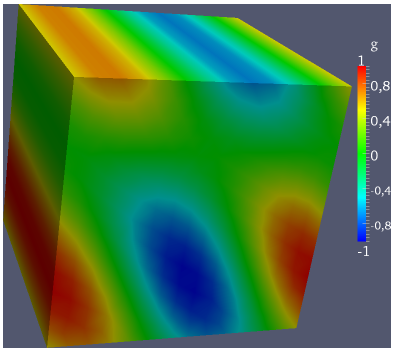
H Tools in Feel++ : trace and lift

Trace of trace and lift of lift operations are needed in the construction of a substructuring preconditioner e.g. for the mortar method presented above in this thesis, which we are currently developping in 2D and 3D. Let Ω be a domain of $\mathbb{R}^3$, $\Sigma \subset \partial\Omega$ an open and nonempty subset and $\Gamma := \partial\Sigma$. We also recall that the trace space of $V := H^1(\Omega)$ on Σ is denoted by $H^{1/2}(\Sigma)$ and the trace space of $W := H^{1/2}(\Sigma)$ on Γ is indicated by $\Lambda := H^{1/2}_{\Sigma}(\Gamma)$ that is the trace space of trace space of V on Γ. It is necessary in this part to be able to manipulate the objects of real dimension equal to d and topological dimension ranging from 1 to d back and forth. Let $u \in V$, first we compute $v = u_{|\Sigma} \in W$ the trace of u and then $w = v_{|\Gamma} \in \Lambda$ the trace of v that is also the trace of trace of u. Reciprocally let $w \in \Lambda$. The extension of w by its mean $c := \dfrac{1}{|\Gamma|}\displaystyle\int_{\Gamma} w$ in W is given by $v \in W$ such that $v = w$ on Γ and $v = c$ in Σ. Now we compute the harmonic extension of v in V that is given by $u \in V$ such that $-\Delta u = 0$ in Ω and $u = v$ on Σ.

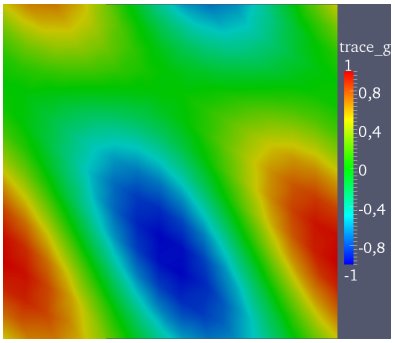
Listing H.1 : Trace of trace and lift and lift implementation

```
auto Xh = space_type::New(mesh);
// trace function space associated to trace(mesh)
auto TXh = trace_space_type::New(mesh->trace(markedfaces(mesh,marker)));
// trace function space associated to trace(trace(mesh))
auto TTXh = trace_trace_space_type::New(TXh->mesh()->trace(
```

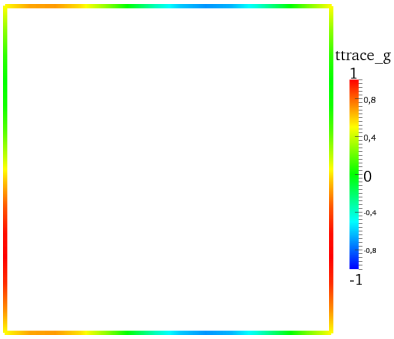
```
                                                boundaryfaces(TXh->mesh())));
// Let g be an function given on 3D mesh
auto g = sin(pi*(2*Px()+Py()+1./4))*cos(pi*(Py()-1./4));
/* trace and trace of trace of g  */
// trace of g on the 2D trace_mesh
auto trace_g = vf::project( TXh, elements(TXh->mesh()), g );
// trace of g on the 1D trace_trace_mesh
auto trace_trace_g = vf::project( TTXh, elements(TTXh->mesh()), g);
/* lift and lift of lift of trace_trace_g  */
// extension of trace_trace_g by zero on 2D trace_mesh
auto zero_extension = vf::project( TXh,boundaryfaces(TXh->mesh()),
                                   idv(trace_trace_g));
// extension of trace_trace_g by the mean of trace_trace_g on trace_mesh
auto const_extension = vf::project( TXh, boundaryfaces(TTXh->mesh()),
                                    idv(trace_trace_g)-mean );
const_extension += vf::project( TXh, elements(TXh->mesh()), cst(mean) );
// harmonic extension of const_extension on 3D mesh
auto op_lift = operatorLift(Xh);
auto glift = op_lift->lift(_range=markedfaces(mesh,marker),
                           _expr=idv(const_extension));
```



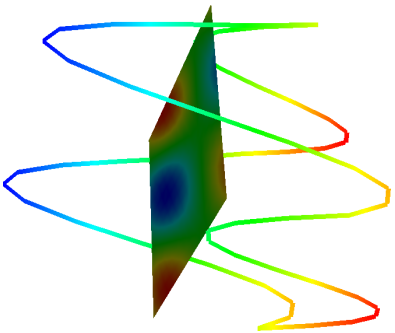
H.1.1. Volume and the function g



H.1.2. Trace mesh and trace of g



H.1.3. Wirebasket and trace of trace of g



H.1.4. Warp with respect the function

Figure H.1 : Volume and wirebasket

I Aitken Acceleration

Let Ω be a domain of $\mathbb{R}^d\,,\ d=1\,,2\,,3\,,$ and $\partial\Omega$ its boundary. We look for u the solution of the problem :

$$\begin{cases} Lu = f & \text{in} \quad \Omega \\ u = g & \text{on} \quad \partial\Omega \end{cases} \tag{I.1}$$

where L is a partial differential operator, and the functions f and g are given. For the sake of exposition we refer to the case of a domain Ω partitioned into two subdomains Ω_1 and Ω_2 such that $\overline{\Omega} = \overline{\Omega}_1 \cup \overline{\Omega}_2$. We denote $\Gamma_1 := \partial\Omega_1 \cap \overline{\Omega}_2$ and $\Gamma_2 := \partial\Omega_2 \cap \overline{\Omega}_1$, in the case of two overlapping subdomains, and $\Gamma := \partial\Omega_1 \cap \partial\Omega_2$, in the case of two nonoverlapping subdomains. The norm of $H^1(\Phi)$ will be denoted by $\|\cdot\|_{1,\Phi}$, while $\|\cdot\|_{0,\Phi}$ will indicate the norm of $L^2(\Phi)$ for all nonempty subset $\Phi \subseteq \Omega$.

First we are interested in the overlapping and nonoverlapping Schwarz methods [QV99]. The overlapping multiplicative Schwarz algorithm with Dirichlet interface conditions at $(k+1)^{th}$ iteration, $k \geq 0$, is given by (I.2) where u_2^0 is known on Γ_1. The additive version of this algorithm is obtained by changing the interface condition $u_2^{k+1} = u_1^{k+1}$ on the second subdomain Ω_2 to $u_2^{k+1} = u_1^k$ in the second system of (I.2).

$$\begin{cases} Lu_1^{k+1} = f & \text{in} \quad \Omega_1 \\ u_1^{k+1} = g & \text{on} \quad \partial\Omega_1 \setminus \Gamma_1 \\ u_1^{k+1} = u_2^k & \text{on} \quad \Gamma_1 \end{cases} \qquad \begin{cases} Lu_2^{k+1} = f & \text{in} \quad \Omega_2 \\ u_2^{k+1} = g & \text{on} \quad \partial\Omega_2 \setminus \Gamma_2 \\ u_2^{k+1} = u_1^{k+1} & \text{on} \quad \Gamma_2 \end{cases} \tag{I.2}$$

The nonoverlapping Schwarz algorithm with Dirichlet and Neumann interface conditions at $(k+1)^{th}$ iteration, $k \geq 0$, are given by

$$\begin{cases} Lu_1^{k+1} = f & \text{in} \quad \Omega_1 \\ u_1^{k+1} = g & \text{on} \quad \partial\Omega_1 \setminus \Gamma \\ u_1^{k+1} = \lambda^k & \text{on} \quad \Gamma \end{cases} \qquad \begin{cases} Lu_2^{k+1} = f & \text{in} \quad \Omega_2 \\ u_2^{k+1} = g & \text{on} \quad \partial\Omega_2 \setminus \Gamma \\ \dfrac{\partial u_2^{k+1}}{\partial n} = \dfrac{\partial u_1^{k+1}}{\partial n} & \text{on} \quad \Gamma \end{cases} \tag{I.3}$$

where $\lambda^{k+1} := \theta u_{2|\Gamma}^{k+1} + (1-\theta)\lambda^k$, θ being a positive acceleration parameter which can be computed for example by an Aitken procedure, see listing I.2, and λ^0 is given on Γ. The additive Schwarz method requires generally more iterations than the multiplicative method and is naturally parallelizable. The generalization of these algorithms to many subdomains is immediate.

Listing I.1 : Fixed point algorithm using Aitken acceleration for (I.2) and (I.3)

```
enum DDMethod {  DD = 0,/*Dirichlet-Dirichlet*/
                 DN = 1 /*Dirichlet-Neumann*/ };
```

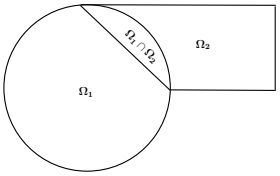
```
auto accel =  aitken( _space=Xh2 );
accel.initialize( _residual=residual,_currentElt=lambda);
double maxIteration = 20;
while( !accel.isFinished() &&
       accel.nIterations() < maxIteration)
{
    // call the localProblem on the first subdomain Ω1
    localProblem(u1, idv(u2), DDMethod::DD);
    lambda = u2;
    // call the localProblem on the first subdomain Ω2
    if( ddmethod = DDMethod::DD )
        localProblem(u2, idv(u1), DDMethod::DD);
    else
        localProblem(u2, gradv(u1)*N(), DDMethod::DN);
    residual = u2-lambda;
    u2 = accel.apply( _residual=residual,_currentElt=u2);
    ++accel;
}
```

The numerical solutions in FIGURE I.1 correspond to the partition of Ω into two subdomains Ω_1 and Ω_2 and the following configuration : (i) $g(x\,,y) = \sin(\pi x)\cos(\pi y)$ is the exact solution (ii) $f(x\,,y) = 2\pi^2 g$ is the right hand side of the equation (iii) we use $\mathbb{P}_2$ Lagrange approximation (iv) we use the maximal number of iteration equal to 10.

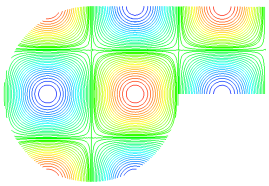
Table TABLE I.1 summarizes The L^2 and H^1 errors for both problems studied.

TABLE I.1 : Numerical results for Schwarz methods

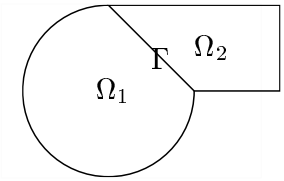
	$\|\mathbf{u_1}-\mathbf{g}\|_{0,\Omega_1}$	$\|\mathbf{u_2}-\mathbf{g}\|_{0,\Omega_2}$	$\|\mathbf{u_1}-\mathbf{g}\|_{1,\Omega_1}$	$\|\mathbf{u_2}-\mathbf{g}\|_{1,\Omega_2}$
With overlap	2.52e-8	2.16e-8	4.07e-6	3.89e-6
Without overlap	1.37e-9	2.04e-8	1.32e-6	6.71e-6



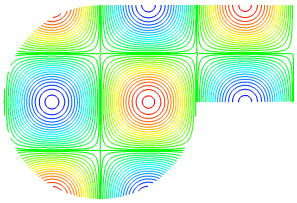
I.1.1. Geometry



I.1.2. Numerical solution



I.1.3. Geometry



I.1.4. Numerical solution

Figure I.1 : Overlapping and nonoverlapping Schwartz methods in 2D

Listing I.2 : Local problems for (3.40) and (6.1)

```
template<Expr> void localProblem(element_type& u, Expr expr, DDMethod ddmethod)
{
  auto Xh=u.functionSpace();
  auto mesh=Xh->mesh();
  auto v=Xh->element();
  auto F = M_backend->newVector(Xh);
  auto A = M_backend->newMatrix( Xh, Xh );

  // Assembly of the right hand side ∫_Ω fv
  form1( _test=Xh,_vector=F ) = integrate( elements(mesh), f*id(v) );

  // Assembly of the left hand side ∫_Ω ∇u · ∇v
  form2( _test=Xh, _trial=Xh, _matrix=A ) =
      integrate( elements(mesh), gradt(u)*trans(grad(v)) );

  // Add Neumann contribution
  if ( ddmethod == DDMethod::DN )
      form1( _test=Xh,_vector=F ) += integrate( markedfaces(mesh,"Interface"),
                                                _expr=expr*id(v));
  else if( ddmethod == DDMethod::DD )
      form2( Xh, Xh, A ) += on( markedfaces(mesh,"Interface") , u,F,expr);

  // Apply the Dirichlet boundary conditions
  form2( Xh, Xh, A ) += on( markedfaces(mesh, "Dirichlet"), u,F,g);

  // Apply the Dirichlet interface conditions
  // solve the linear system Au = F
  M_backend->solve(_matrix=A,_solution=u,_rhs=F );
}
```

J Some Results for Substructuring Preconditioners

We present some numerical results related to the set of experiments presented in section 8.1 and in section 8.2 respectively for conforming and nonconforming domain decompositions. We report the number of iterations required for solving the linear system (4.41) preconditioned by $\widehat{\mathbf{P}} \in \{\mathbf{P}_0\,, \mathbf{P}_1\,, \mathbf{P}_2\}$, the condition number $\kappa(\widehat{\mathbf{P}}^{-1}\widehat{\mathbf{S}})$ when varying the number of subdomains N and the number of elements n of the fine mesh.

Conforming Domain Decompositions

Table J.1 : Condition number and number of iterations (between parenthesis) for $p = 1$

	$N\backslash n$	5	10	20	40	80	160	320
$\mathbf{P}_0$	16	22.72 (25)	27.33 (27)	32.62 (29)	37.96 (30)	44.13 (31)	50.82 (31)	58.29 (30)
	64	22.05 (25)	26.67 (27)	31.59 (28)	36.84 (29)	43.15 (29)	49.68 (30)	56.82 (31)
	256	20.82 (22)	25.57 (24)	30.56 (25)	35.83 (26)	42.00 (27)	46.51 (27)	52.84 (28)
$\mathbf{P}_1$	16	22.16 (26)	26.47 (27)	32.34 (28)	39.93 (31)	49.32 (33)	60.27 (34)	73.22 (36)
	64	21.52 (24)	26.12 (27)	32.10 (29)	39.03 (31)	48.37 (33)	58.92 (35)	71.48 (36)
	256	20.38 (21)	24.38 (23)	30.26 (25)	37.77 (28)	47.16 (30)	58.27 (33)	70.75 (35)
$\mathbf{P}_2$	16	24.48 (23)	28.07 (24)	34.61 (26)	41.01 (28)	47.91 (31)	55.91 (33)	64.99 (35)
	64	24.37 (22)	27.94 (23)	35.06 (26)	42.66 (29)	50.63 (31)	59.37 (33)	68.91 (35)
	256	24.04 (20)	27.88 (21)	35.48 (23)	43.11 (26)	51.24 (28)	60.31 (30)	69.78 (33)

Table J.2 : Condition number and number of iterations (between parenthesis) for $p = 2$

	$N\backslash n$	5	10	20	40	80	160
$\mathbf{P}_0$	16	24.18 (26)	29.10 (27)	34.95 (29)	41.52 (30)	48.58 (30)	57.13 (30)
	64	23.95 (25)	28.74 (27)	34.19 (27)	40.37 (29)	49.04 (31)	55.64 (32)
	256	22.90 (22)	27.71 (24)	33.37 (26)	39.41 (28)	47.59 (30)	54.66 (31)
$\mathbf{P}_1$	16	24.92 (27)	29.45 (27)	35.58 (28)	43.29 (30)	52.50 (31)	63.23 (32)
	64	24.76 (25)	29.26 (27)	34.99 (28)	42.01 (30)	50.92 (32)	61.35 (33)
	256	23.65 (22)	28.25 (24)	33.65 (25)	41.12 (29)	49.94 (31)	60.18 (30)
$\mathbf{P}_2$	16	23.59 (23)	29.29 (26)	35.80 (29)	43.68 (31)	52.58 (32)	62.46 (33)
	64	24.71 (22)	31.25 (25)	38.34 (28)	46.46 (31)	55.71 (33)	65.58 (35)
	256	25.10 (21)	31.69 (23)	39.00 (26)	47.21 (28)	56.37 (32)	66.33 (33)

Table J.3 : Condition number and number of iterations (between parenthesis) for $p = 3$

	$N\backslash n$	5	10	20	40	80	160
	16	27.60 (27)	33.11 (28)	39.63 (29)	47.46 (31)	55.74 (31)	64.70 (32)
$\mathbf{P}_0$	64	27.18 (26)	32.65 (27)	38.98 (29)	45.87 (31)	56.22 (32)	62.93 (33)
	256	26.31 (23)	31.57 (25)	38.34 (28)	45.75 (30)	56.28 (32)	62.49 (32)
	16	27.97 (26)	33.51 (27)	41.04 (29)	49.63 (32)	59.32 (32)	70.22 (33)
$\mathbf{P}_1$	64	27.87 (27)	33.33 (28)	40.04 (30)	47.92 (32)	57.51 (34)	68.19 (34)
	256	26.90 (23)	32.26 (25)	38.99 (27)	47.23 (30)	57.07 (32)	67.49 (32)
	16	27.51 (26)	34.34 (28)	42.25 (30)	51.15 (33)	61.11 (33)	72.07 (35)
$\mathbf{P}_2$	64	29.52 (25)	36.55 (28)	44.69 (31)	53.85 (33)	63.94 (35)	75.10 (40)
	256	30.12 (23)	37.05 (25)	45.26 (28)	54.35 (31)	64.59 (34)	75.38 (36)

Table J.4 : Condition number and number of iterations (between parenthesis) for $p = 4$

	$N\backslash n$	5	10	20	40	80
	16	30.73 (27)	37.09 (28)	44.17 (29)	52.57 (32)	62.26 (32)
$\mathbf{P}_0$	64	30.38 (27)	36.27 (28)	43.25 (30)	51.19 (32)	60.68 (33)
	256	29.76 (25)	35.53 (26)	42.88 (29)	50.86 (31)	62.10 (32)
	16	31.08 (27)	37.33 (29)	44.96 (30)	54.05 (32)	63.98 (33)
$\mathbf{P}_1$	64	30.73 (27)	36.66 (29)	43.94 (31)	52.60 (33)	61.57 (34)
	256	29.94 (24)	35.84 (26)	43.16 (28)	51.74 (31)	62.20 (33)
	16	31.85 (28)	39.55 (30)	48.23 (31)	58.03 (35)	68.73 (38)
$\mathbf{P}_2$	64	33.58 (28)	41.39 (30)	50.64 (32)	60.61 (35)	71.62 (40)
	256	34.24 (25)	42.04 (27)	51.20 (29)	61.19 (33)	72.09 (36)

Table J.5 : Condition number and number of iterations (between parenthesis) for $p = 5$

	$N\backslash n$	5	10	20	40	80
$\mathbf{P}_0$	16	33.73 (27)	40.82 (29)	48.22 (30)	57.10 (33)	67.60 (33)
	64	33.14 (28)	39.69 (30)	47.13 (31)	55.47 (33)	65.99 (34)
	256	32.56 (25)	39.23 (27)	46.60 (29)	54.91 (31)	66.91 (33)
$\mathbf{P}_1$	16	33.79 (28)	41.09 (29)	49.35 (30)	58.79 (33)	69.18 (34)
	64	33.44 (28)	40.10 (31)	47.96 (32)	57.10 (34)	66.87 (34)
	256	32.45 (25)	39.23 (27)	47.33 (29)	56.54 (32)	66.87 (33)
$\mathbf{P}_2$	16	36.12 (30)	44.40 (31)	53.75 (32)	64.28 (38)	76.00 (39)
	64	38.04 (29)	46.68 (31)	56.24 (34)	66.96 (40)	79.26 (42)
	256	38.51 (26)	47.15 (29)	56.54 (33)	67.16 (35)	78.95 (40)

Nonconforming Domain Decompositions

Table J.6 : Condition number and number of iterations (between parenthesis) for $p = 1$

	$N\backslash n$	5	10	20	40	80	160	320
$\mathbf{P}_0$	16	7.52 (16)	8.75 (17)	12.38 (19)	16.56 (20)	21.47 (20)	27.00 (22)	33.09 (23)
	64	7.58 (15)	8.99 (17)	12.48 (18)	16.81 (20)	21.54 (21)	26.76 (23)	32.32 (24)
	256	7.20 (14)	8.78 (15)	12.25 (16)	16.12 (17)	20.60 (17)	25.88 (19)	31.81 (20)
$\mathbf{P}_1$	16	6.52 (15)	9.05 (18)	12.95 (20)	17.64 (22)	23.82 (22)	31.33 (24)	40.46 (24)
	64	6.83 (15)	9.11 (17)	13.03 (20)	17.68 (22)	23.91 (25)	31.40 (27)	40.28 (29)
	256	6.62 (13)	8.80 (15)	12.30 (17)	16.95 (20)	22.44 (21)	29.62 (22)	38.41 (23)
$\mathbf{P}_2$	16	9.96 (15)	9.50 (16)	11.78 (17)	16.05 (20)	21.51 (22)	27.76 (23)	35.04 (28)
	64	10.06 (16)	9.66 (16)	11.91 (17)	16.68 (20)	22.21 (22)	29.29 (25)	36.82 (29)
	256	9.74 (15)	9.51 (14)	11.38 (16)	16.85 (19)	22.53 (21)	29.37 (23)	36.78 (25)

Table J.7 : Condition number and number of iterations (between parenthesis) for $p = 2$

	$N\backslash n$	5	10	20	40	80	160
$\mathbf{P}_0$	16	11.24 (18)	14.32 (19)	18.91 (20)	24.10 (21)	30.16 (22)	36.55 (23)
	64	11.64 (18)	14.59 (19)	19.04 (20)	24.27 (22)	29.76 (24)	35.94 (26)
	256	11.21 (15)	14.00 (16)	18.25 (17)	23.22 (18)	28.95 (20)	35.25 (21)
$\mathbf{P}_1$	16	11.24 (19)	14.43 (21)	19.03 (22)	24.29 (22)	30.20 (24)	37.14 (25)
	64	11.40 (18)	14.58 (21)	19.00 (22)	24.28 (24)	30.11 (26)	36.72 (28)
	256	11.14 (16)	14.06 (18)	18.27 (19)	23.26 (20)	29.02 (21)	35.60 (22)
$\mathbf{P}_2$	16	10.36 (17)	13.72 (19)	18.95 (21)	24.89 (23)	32.14 (27)	40.84 (29)
	64	10.94 (17)	14.41 (19)	19.93 (22)	26.17 (25)	33.79 (28)	42.86 (30)
	256	10.81 (16)	14.48 (18)	19.89 (20)	26.19 (22)	33.32 (25)	41.44 (29)

Table J.8 : Condition number and number of iterations (between parenthesis) for $p = 3$

	$N\backslash n$	5	10	20	40	80	160
	16	17.77 (22)	18.82 (19)	23.91 (20)	29.76 (22)	36.45 (23)	43.59 (24)
$\mathbf{P}_0$	64	17.91 (21)	18.94 (20)	24.14 (23)	29.81 (25)	35.88 (27)	42.71 (29)
	256	17.11 (19)	18.02 (18)	22.94 (18)	28.66 (20)	35.00 (21)	42.49 (22)
	16	17.78 (22)	18.84 (21)	23.99 (22)	29.82 (24)	36.55 (24)	43.60 (24)
$\mathbf{P}_1$	64	17.92 (23)	19.11 (22)	24.16 (23)	29.86 (25)	36.25 (27)	42.85 (28)
	256	17.14 (19)	17.98 (18)	22.96 (19)	28.64 (20)	34.99 (21)	41.99 (22)
	16	16.43 (22)	18.89 (19)	24.92 (24)	32.62 (27)	41.19 (28)	50.58 (29)
$\mathbf{P}_2$	64	17.07 (21)	19.65 (22)	26.34 (25)	34.48 (28)	43.32 (30)	53.06 (32)
	256	17.18 (19)	19.79 (19)	26.15 (23)	33.36 (26)	42.76 (30)	52.65 (30)

Table J.9 : Condition number and number of iterations (between parenthesis) for $p = 4$

	$N\backslash n$	5	10	20	40	80
	16	18.60 (19)	22.43 (20)	28.17 (22)	34.49 (23)	41.57 (24)
$\mathbf{P}_0$	64	18.97 (21)	22.64 (22)	28.45 (25)	34.19 (26)	41.03 (29)
	256	18.52 (19)	22.13 (20)	27.01 (20)	33.17 (21)	41.01 (24)
	16	18.64 (21)	22.49 (22)	28.20 (23)	34.53 (24)	41.58 (24)
$\mathbf{P}_1$	64	18.94 (22)	22.68 (23)	28.34 (25)	34.42 (27)	41.24 (29)
	256	17.97 (18)	21.54 (19)	27.00 (20)	33.20 (22)	39.97 (22)
	16	19.24 (21)	23.62 (23)	31.16 (25)	39.39 (28)	48.68 (28)
$\mathbf{P}_2$	64	19.89 (22)	24.95 (24)	32.86 (26)	41.56 (29)	51.05 (31)
	256	20.24 (21)	24.81 (22)	32.00 (25)	40.31 (28)	51.05 (31)

Table J.10 : Condition number and number of iterations (between parenthesis) for $p = 5$

	$N\backslash n$	5	10	20	40	80
$\mathbf{P}_0$	16	21.41 (19)	25.64 (20)	31.71 (22)	38.35 (23)	45.86 (24)
	64	21.76 (21)	25.93 (22)	31.91 (25)	38.16 (26)	45.55 (29)
	256	21.27 (19)	25.14 (20)	30.42 (20)	36.97 (21)	44.91 (24)
$\mathbf{P}_1$	16	21.47 (22)	25.69 (23)	31.74 (23)	38.38 (25)	45.90 (26)
	64	21.92 (23)	25.95 (24)	31.99 (26)	38.40 (28)	45.17 (29)
	256	20.72 (19)	24.58 (20)	30.46 (21)	37.00 (22)	44.16 (23)
$\mathbf{P}_2$	16	23.06 (23)	28.35 (24)	36.23 (26)	44.99 (29)	54.91 (28)
	64	24.21 (24)	29.90 (25)	38.29 (28)	47.36 (30)	57.55 (32)
	256	24.31 (22)	29.12 (24)	37.60 (26)	47.23 (29)	57.73 (31)

Bibliography

[Abd+99] G.S. Abdoulaev, Y. Achdou, Y.A. Kuznetsov, and C. Prud'homme. "On a parallel implementation of the Mortar element method". In : *RAIRO-M2AN Modelisation Math et Analyse Numerique-Mathem Modell Numerical Analysis* 33.2 (1999), pp. 245–260 (cit. on p. 91).

[AF03] R.A. Adams and J.J.F. Fournier. *Sobolev Spaces*. Pure and Applied Mathematics. Elsevier Science, 2003. ISBN : 9780080541297. URL : http://books.google.fr/books?id=R5A65Koh-EoC (cit. on pp. 143, 144).

[AMW99] Y. Achdou, Y. Maday, and O. Widlund. "Substructuring preconditioners for the mortar method in dimension two". In : *SIAM J. Numer. Anal.* (1999), pp. 551–580 (cit. on pp. 18, 43, 47).

[Ant+14] P.F. Antonietti, B. Ayuso de Dios, S. Bertoluzza, and M. Pennacchio. "Substructuring preconditioners for an h-p domain decomposition method with interior penalty mortaring". English. In : *Calcolo* (2014), pp. 1–28. ISSN : 0008-0624. DOI : 10.1007/s10092-014-0117-9. URL : http://dx.doi.org/10.1007/s10092-014-0117-9 (cit. on p. 53).

[AP95] Y. Achdou and O. Pironneau. "A Fast Solver for Navier–Stokes Equations in the Laminar Regime Using Mortar Finite Element and Boundary Element Methods". In : *SIAM Journal on Numerical Analysis* 32.4 (1995), pp. 985–1016 (cit. on p. 18).

[Bal+04] Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. *PETSc Users Manual*. Tech. rep. ANL-95/11 - Revision 2.1.5. Argonne National Laboratory, 2004 (cit. on pp. 9, 66).

[BB94] Z. Belhachmi and C. Bernardi. "The mortar spectral element method for fourth-order problems." In : *Comp. Methods in Applied Mech. and Eng.* 116 (1994), pp. 53–58 (cit. on p. 18).

[BBG04] Smith Barry, Petter Bjorstad, and William Gropp. *Domain decomposition : parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press, 2004 (cit. on pp. 15, 16, 89).

[BBM01] F. Ben Belgacem, A. Buffa, and Y. Maday. "The Mortar Finite Element Method for 3D Maxwell Equations : First Results". In : *SIAM Journal on Numerical Analysis* 39.3 (2001), pp. 880–901 (cit. on p. 18).

[Bel99] Faker Ben Belgacem. "The Mortar finite element method with Lagrange multipliers". English. In : *Numerische Mathematik* 84.2 (1999), pp. 173–197. ISSN : 0029-599X. DOI : 10.1007/s002110050468. URL : http://dx.doi.org/10.1007/s002110050468 (cit. on pp. 3, 4, 18, 56).

[Ber+14] Silvia Bertoluzza, Micol Pennacchio, Christophe Prud'homme, and Abdoulaye Samaké. "substructuring preconditioners for hp mortar FEM". In : *ESAIM : Mathematical Modelling and Numerical Analysis* (2014). Submitted (cit. on pp. 3, 5, 28).

[Ber03] S. Bertoluzza. "Substructuring preconditioners for the three fields domain decomposition methods". In : *Mathematics of computation* 73 (2003), pp. 659–689 (cit. on p. 34).

[Ber04] Silvia Bertoluzza. "Substructuring preconditioners for the three fields domain decomposition method". In : *Math. Comp.* 73.246 (2004), 659–689 (electronic). ISSN : 0025-5718. DOI : 10.1090/S0025-5718-03-01550-3. URL : http://dx.doi.org/10.1090/S0025-5718-03-01550-3 (cit. on p. 43).

[BF11] Silvia Bertoluzza and Silvia Falletta. "Analysis of some injection bounds for Sobolev spaces by wavelet decomposition". English. In : *C. R., Math., Acad. Sci. Paris* 349.7-8 (2011), pp. 421–423. DOI : 10.1016/j.crma.2011.02.015 (cit. on p. 33).

[BFM08] Silvia Bertoluzza, Silvia Falletta, and Gianmarco Manzini. "Efficient design of residual-based stabilization techniques for the three fields domain decomposition method". In : *Mathematical Models and Methods in Applied Sciences* 18.07 (2008), pp. 973–999 (cit. on p. 19).

[BGL05] M. Benzi, G.H. Golub, and J. Liesen. "Numerical solution of saddle point problems". In : *Acta numerica* 14.1 (2005), pp. 1–137 (cit. on pp. 3, 4, 58).

[BM94] F. Brezzi and L.D. Marini. "A three-fields domain decomposition method". In : *Domain Decomposition Methods in Science and Engineering* (1994), pp. 27–34 (cit. on pp. 3, 5, 19–21).

[BMP93a] C. Bernardi, Y. Maday, and A. Patera. "A new nonconforming approach to domain decomposition :the mortar element method". In : *Nonlinear Partial Differential Equations and their Applications* (1993) (cit. on pp. 2–4, 28).

[BMP93b] Christine Bernardi, Yvon Maday, and Anthony T. Patera. "Domain decomposition by the mortar element method". In : *Asymptotic and Numerical Methods for Partial Differential Equations with Critical Parameters*. Ed. by H.G. Kaper ans M. Garbey. N.A.T.O. ASI. Kluwer Academic Publishers, 1993, pp. 269–286 (cit. on p. 18).

[BMP94] Christine Bernardi, Yvon Maday, and Anthony T. Patera. "A New Non Conforming Approach to Domain Decomposition : The Mortar Element Method". In : *Collège de France Seminar*. Ed. by Haim Brezis and Jacques-Louis Lions. This paper appeared as a technical report about five years earlier. Pitman, 1994 (cit. on p. 18).

[BP04] S. Bertoluzza and M. Pennacchio. "Preconditioning the Mortar Method by Substructuring : The High Order Case." In : *Applied Numerical Analysis & Computational Mathematics* 1.2 (2004), pp. 434–454. ISSN : 1611-8189. DOI : 10.1002/anac.200410008. URL : http://dx.doi.org/10.1002/anac.200410008 (cit. on pp. 3, 5, 42, 43, 48).

[BP07] Silvia Bertoluzza and Micol Pennacchio. "Analysis of substructuring preconditioners for mortar methods in an abstract framework". In : *Applied Mathematics Letters* 20.2 (2007), pp. 131–137. ISSN : 0893-9659. DOI : 10.1016/j.aml.2006.02.029. URL : http://www.sciencedirect.com/science/article/pii/S0893965906001108 (cit. on pp. 3, 5).

[BPS86] James H Bramble, Joseph E Pasciak, and Alfred H Schatz. "The construction of preconditioners for elliptic problems by substructuring. I". In : *Mathematics of Computation* 47.175 (1986), pp. 103–134 (cit. on pp. 3, 5, 34, 43, 53, 155).

[BS94] Ivo Babuška and Manil Suri. "The p and h-p versions of the finite element method, basic principles and properties". In : *SIAM review* 36.4 (1994), pp. 578–632 (cit. on pp. 32, 45).

[BSS00] Faker Ben Belgacem, Padmanabhan Seshaiyer, and Manil Suri. "Optimal convergence rates of hp mortar finite element methods for second-order elliptic problems". In : *ESAIM : Mathematical Modelling and Numerical Analysis* 34.03 (2000), pp. 591–608 (cit. on pp. 36, 37, 44).

[BW86] Petter E Bjørstad and Olof B Widlund. "Iterative methods for the solution of elliptic problems on regions partitioned into substructures". In : *SIAM Journal on Numerical Analysis* 23.6 (1986), pp. 1097–1120 (cit. on p. 53).

[Can+06] C.G. Canuto, M.Y. Hussaini, A. Quarteroni, and T.A. Zang. *Spectral Methods*. Scientific Computation. Springer, 2006 (cit. on pp. 32, 38).

[CGL01] Luiz M Carvalho, Luc Giraud, and Patrick Le Tallec. "Algebraic two-level preconditioners for the Schur complement method". In : *SIAM Journal on Scientific Computing* 22.6 (2001), pp. 1987–2005 (cit. on p. 16).

[Cha13] Vincent Chabannes. "Vers la simulation des écoulements sanguins". PhD thesis. Université de Grenoble, 2013 (cit. on pp. 67, 71).

[Cia78] Philippe G Ciarlet. *The finite element method for elliptic problems*. Elsevier, 1978 (cit. on p. 144).

[CMZ12] Jie Chen, L.C. McInnes, and H. Zhang. *Analysis and practical use of flexible BICGSTAB*. Tech. rep. ANL/MCS-P3039-0912. Argonne National Laboratory, 2012 (cit. on p. 59).

[CN98] Philippe Chevalier and Frédéric Nataf. "An optimized order 2 (OO2) method for the Helmholtz equation". In : *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics* 326.6 (1998), pp. 769–774 (cit. on p. 16).

[CS99] Xiao-Chuan Cai and Marcus Sarkis. "A restricted additive Schwarz preconditioner for general sparse linear systems". In : *SIAM Journal on Scientific Computing* 21.2 (1999), pp. 792–797 (cit. on p. 16).

[Dav06] Timothy A Davis. *Direct methods for sparse linear systems*. Vol. 2. Siam, 2006 (cit. on pp. 2, 3).

[DDP06] N. Dokeva, M. Dryja, and W. Proskurowski. "A FETI-DP Preconditioner with A Special Scaling for Mortar Discretization of Elliptic Problems with Discontinuous Coefficients". In : *SIAM Journal on Numerical Analysis* 44.1 (2006), pp. 283–299 (cit. on p. 18).

[DER86] Iain S Duff, Albert Maurice Erisman, and John Ker Reid. *Direct methods for sparse matrices*. Clarendon Press Oxford, 1986 (cit. on pp. 2, 3).

[Dry82] M. Dryja. "A capacitance matrix method for Dirichlet problem on polygon region". English. In : *Numerische Mathematik* 39.1 (1982), pp. 51–64. ISSN : 0029-599X. DOI : 10.1007/BF01399311. URL : http ://dx.doi.org/10.1007/BF01399311 (cit. on p. 53).

[EG04] A. Ern and J.L. Guermond. *Theory and Practice of Finite Elements*. Applied Mathematical Sciences vol. 159. Springer, 2004. ISBN : 9780387205748. URL : http ://books.google.fr/books ?id=CCjm79FbJbcC (cit. on p. 38).

[ESW05] H.C. Elman, D.J. Silvester, and A.J. Wathen. *Finite Elements and Fast Iterative Solvers :with Applications in Incompressible Fluid Dynamics : with Applications in Incompressible Fluid Dynamics*. Numerical Mathematics and Scientific Computation. OUP Oxford, 2005. ISBN : 9780198528678. URL : http ://books.google.fr/books ?id=XTHhkpUkOhsC (cit. on p. 58).

[EZ98] Bjorn Engquist and Hong-Kai Zhao. "Absorbing boundary conditions for domain decomposition". In : *IUTAM Symposium on Computational Methods for Unbounded Domains*. Springer. 1998, pp. 315–324 (cit. on p. 16).

[FCM95] Charbel Farhat, Po-Shu Chen, and Jan Mandel. "A scalable Lagrange multiplier based domain decomposition method for time-dependent problems". In : *International Journal for Numerical Methods in Engineering* 38.22 (1995), pp. 3831–3853 (cit. on p. 24).

[FR91] Charbel Farhat and Francois-Xavier Roux. "A method of finite element tearing and interconnecting and its parallel solution algorithm". In : *International Journal for Numerical Methods in Engineering* 32.6 (1991), pp. 1205–1227 (cit. on p. 24).

[G+10] Gaël Guennebaud, Benoît Jacob, et al. *Eigen v3*. http ://eigen.tuxfamily.org. 2010 (cit. on pp. 66, 80, 151).

[Gan08] Martin J. Gander. "Schwarz methods over the course of time." eng. In : *ETNA. Electronic Transactions on Numerical Analysis [electronic only]* 31 (2008), pp. 228–255. URL : http ://eudml.org/doc/130616 (cit. on pp. 2, 4).

[GC96] Benqi Guo and Weiming Cao. "A preconditioner for the h-p version of the finite element method in two dimensions". English. In : *Numerische Mathematik* 75.1 (1996), pp. 59–77. ISSN : 0029-599X. DOI : 10.1007/s002110050230. URL : http://dx.doi.org/10.1007/s002110050230 (cit. on pp. 34, 46).

[GR09] Christophe Geuzaine and Jean-François Remacle. "Gmsh : A 3-D finite element mesh generator with built-in pre-and post-processing facilities". In : *International Journal for Numerical Methods in Engineering* 79.11 (2009), pp. 1309–1331 (cit. on p. 66).

[GV96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations (3rd Ed.)* Baltimore, MD, USA : Johns Hopkins University Press, 1996. ISBN : 0-8018-5414-8 (cit. on p. 79).

[GY99] G.H. Golub and Q. Ye. "Inexact preconditioned conjugate gradient method with inner-outer iteration". In : *SIAM Journal on Scientific Computing* 21.4 (1999), pp. 1305–1320 (cit. on p. 59).

[HHT08] Nicholas Hale, Nicholas J Higham, and Lloyd N Trefethen. "Computing A^α, $\log(A)$, and related matrix functions by contour integrals". In : *SIAM Journal on Numerical Analysis* 46.5 (2008), pp. 2505–2523 (cit. on pp. 142, 151).

[HRP93] Kai Hwang, A Ramachandran, and R Purushothaman. *Advanced computer architecture : parallelism, scalability, programmability*. Vol. 199. McGraw-Hill New York, 1993 (cit. on p. 9).

[JMN13] Caroline Japhet, Yvon Maday, and Frédéric Nataf. "A New Interface Cement Equilibrated Mortar (NICEM) method with Robin interface conditions : the P1 finite element case". In : *Mathematical Models and Methods in Applied Sciences* 23.12 (2013), pp. 2253–2292 (cit. on p. 81).

[Jol+12] Pierre Jolivet, Victorita Dolean, Frédéric Hecht, Frédéric Nataf, Christophe Prud'Homme, and Nicole Spillane. "High performance domain decomposition methods on massively parallel architectures with FreeFem++". In : *Journal of Numerical Mathematics* 20.3-4 (2012), pp. 287–302 (cit. on p. 16).

[Jol14] Pierre Jolivet. "Méthodes de décomposition de domaine. Application au calcul haute performance." PhD thesis. Université de Grenoble, 2014 (cit. on p. 27).

[Kim07] H Kim. "A FETI-DP preconditioner for mortar methods in three dimensions". In : *Electron. Trans. Numer. Anal.* 26 (2007), pp. 103–120 (cit. on p. 18).

[KL05] H. Kim and C. Lee. "A Preconditioner for the FETI-DP Formulation with Mortar Methods in Two Dimensions". In : *SIAM Journal on Numerical Analysis* 42.5 (2005), pp. 2159–2175 (cit. on p. 18).

[Kor11] Sandeep Koranne. "Boost c++ libraries". In : *Handbook of Open Source Tools*. Springer, 2011, pp. 127–143 (cit. on pp. 9, 66).

[KW06] H. H. Kim and O. B. Widlund. "Two-level Schwarz algorithms with overlapping subregions for mortar finite elements". In : *SIAM Journal on Numerical Analysis* 44 (2006), 1514 ?1534 (cit. on p. 18).

[Lio88] Pierre-Louis Lions. "On the Schwarz alternating method. I". In : *First international symposium on domain decomposition methods for partial differential equations*. Paris, France. 1988, pp. 1–42 (cit. on p. 16).

[LM72] J.L. Lions and E. Magenes. *Non-homogeneous boundary value problems and applications*. English. Vol. I. Die Grundlehren der mathematischen Wissenschaften. Springer-Verla, 1972 (cit. on p. 33).

[MGW00] M.F. Murphy, G.H. Golub, and A.J. Wathen. "A note on preconditioning for indefinite linear systems". In : *SIAM Journal on Scientific Computing* 21.6 (2000), pp. 1969–1972 (cit. on p. 58).

[MS11] Vladimir Maz'ya and Tatyana O Shaposhnikova. *Sobolev spaces : with applications to elliptic partial differential equations*. Vol. 342. Springer, 2011 (cit. on p. 143).

[MT96] Jan Mandel and Radek Tezaur. "Convergence of a substructuring method with Lagrange multipliers". In : *Numerische Mathematik* 73.4 (1996), pp. 473–487 (cit. on p. 27).

[Nat+11] Frédéric Nataf, Hua Xiang, Victorita Dolean, and Nicole Spillane. "A coarse space construction based on local Dirichlet-to-Neumann maps". In : *SIAM Journal on Scientific Computing* 33.4 (2011), pp. 1623–1642 (cit. on p. 16).

[Nit71] J. Nitsche. "Uber ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind". German. In : *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 36.1 (1971), pp. 9–15. ISSN : 0025-5858. DOI : 10.1007/BF02995904. URL : http://dx.doi.org/10.1007/BF02995904 (cit. on p. 88).

[Pen04] Micol Pennacchio. "The mortar finite element method for the cardiac "bidomain" model of extracellular potential." In : *J. Sci. Comput.* 20.2 (2004), pp. 191–210 (cit. on p. 18).

[Pru+12a] Christophe Prud'homme, Vincent Chabannes, Vincent Doyeux, Mourad Ismail, Abdoulaye Samaké, and Gonçalo Pena. "Feel++ : A Computational Framework for Galerkin Methods and Advanced Numerical Methods". In : *Cemracs 2011*. Published. EDP Sciences, Jan. 2012 (cit. on p. 67).

[Pru+12b] Christophe Prud'homme, Vincent Chabannes, Vincent Doyeux, Mourad Ismail, Abdoulaye Samaké, and Gonçalo Pena. *Feel++ : A Computational Framework for Galerkin Methods and Advanced Numerical Methods*. Rapport de recherche hal-00662868. HAL, Jan. 2012 (cit. on p. 8).

[Pru06] Christophe Prud'homme. *A domain specific embedded language in C++ for automatic differentiation, projection, integration and variational formulations*. Scientific Programming, 14(2) :81-110. 2006 (cit. on p. 8).

[Pru07] Christophe Prud'homme. *Life : Overview of a unified C++ implementation of the finite and spectral element methods in 1d, 2d and 3d.* In Workshop On State-Of-The-Art In Scientific And Parallel Computing, Lecture Notes in Computer Science, page 10. Springer-Verlag. 2007 (cit. on p. 8).

[Pru98] C. Prud'homme. "A strategy for the resolution of the tridimensionnal incompressible Navier-Stokes equations". In : *Méthodes itératives de décomposition de domaines et communications en calcul parallèle. Calculateurs Parallèles Réseaux et Systèmes répartis.* Vol. 10. Hermes, 1998, pp. 371–380 (cit. on p. 18).

[PS08] Micol Pennacchio and Valeria Simoncini. "Substructuring preconditioners for mortar discretization of a degenerate evolution problem". In : *J. Sci. Comput.* 36.3 (2008), pp. 391–419. ISSN : 0885-7474. DOI : 10.1007/s10915-008-9195-7. URL : http://dx.doi.org/10.1007/s10915-008-9195-7 (cit. on p. 18).

[Qua10] A. Quarteroni. *Numerical Models for Differential Problems.* MS&A. Springer, 2010. ISBN : 9788847010710. URL : http://books.google.fr/books?id=0gCRhwrnEO4C (cit. on p. 151).

[QV08] Alfio Quarteroni and Alberto Valli. *Numerical approximation of partial differential equations.* Vol. 23. Springer, 2008 (cit. on pp. 145–147).

[QV99] A. Quarteroni and A. Valli. "Domain decomposition methods for partial Differential equations". In : Numerical Mathematics and Scientific Computation. New York : Oxford University Press, July 1999. Chap. The Mathematical Fundation of Domain Decomposition Methods, pp. 1–39 (cit. on pp. 15, 16, 163).

[RF97] Daniel Rixen and Charbel Farhat. "Preconditioning the FETI method for problems with intra-and inter-subdomain coefficient jumps". In : *Ninth International Conference on Domain Decomposition Methods.* 1997, pp. 472–479 (cit. on p. 25).

[Saa03] Y. Saad. *Iterative methods for sparse linear systems.* Society for Industrial and Applied Mathematics, 2003 (cit. on pp. 2, 3, 59, 72, 79).

[Saa93] Y. Saad. "A flexible inner-outer preconditioned GMRES algorithm". In : *SIAM Journal on Scientific Computing* 14.2 (1993), pp. 461–469 (cit. on p. 59).

[Sam+12a] Abdoulaye Samaké, Silvia Bertoluzza, Micol Pennacchio, Christophe Prud'Homme, and Chady Zaza. "A Parallel Implementation of the Mortar Element Method in 2D and 3D". In : published. 2012 (cit. on pp. 56, 62).

[Sam+12b] Abdoulaye Samaké, Vincent Chabannes, Christophe Picard, and Christophe Prud'Homme. "Domain decomposition methods in Feel++". In : *Springer* (Nov. 2012). published.

[Sam+14] Abdoulaye Samaké, Silvia Bertoluzza, Micol Pennacchio, and Christophe Prud'homme. "Implementation and Numerical Results of Substructuring preconditioners for the *h-p* Mortar Method". In : *in preparation* (2014) (cit. on p. 124).

[Sch70] H.A. Schwarz. *Ueber einen Grenzübergang durch alternirendes Verfahren.* Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich. Zürcher u. Furrer, 1870. URL : http://books.google.fr/books?id=tMcxYAAACAAJ (cit. on pp. 2, 4, 14).

[Sch98] C. Schwab. *p- and hp- Finite Element Methods : Theory and Applications in Solid and Fluid Mechanics*. G.H.Golub and others. Clarendon Press, 1998. ISBN : 9780198503903. URL : http://books.google.fr/books?id=Wq%5C_J7M0tOW8C (cit. on p. 32).

[SGT07] Amik St-Cyr, Martin J Gander, and Stephen J Thomas. "Optimized multiplicative, additive, and restricted additive Schwarz preconditioning". In : *SIAM Journal on Scientific Computing* 29.6 (2007), pp. 2402–2425 (cit. on pp. 15, 16).

[Sni+96] M Snir, S Otto, S Huss-Lederman, D Walker, and J Dongarra. "MPI : THE COMPETE REFERENCE." In : *Computers & Mathematics with Applications* 31.11 (1996), p. 140 (cit. on p. 66).

[SS00] Padmanabhan Seshaiyer and Manil Suri. "Uniform hp convergence results for the mortar finite element method". In : *Mathematics of Computation of the American Mathematical Society* 69.230 (2000), pp. 521–546 (cit. on pp. 37, 45).

[SS98] Padmanabhan Seshaiyer and Manil Suri. "Convergence Results for Non-Conforming hp Methods : The Mortar Finite Element Method". In : *Contemporary Mathematics* 218 (1998) (cit. on pp. 37, 44).

[Ste01] Dan Stefanica. "A numerical study of FETI algorithms for mortar finite element methods". In : *SIAM Journal on Scientific Computing* 23.4 (2001), pp. 1135–1160 (cit. on p. 26).

[Ste99] Dan Stefanica. "Domain decomposition methods for mortar finite elements". PhD thesis. New York University, 1999 (cit. on p. 24).

[Ton70] Pin Tong. "New displacement hybrid finite element models for solid continua". In : *International Journal for Numerical Methods in Engineering* 2.1 (1970), pp. 73–83 (cit. on p. 19).

[TW04] A. Toselli and O. Widlund. "Domain decomposition methods : Algorithms and theory". In : Springer Series in Computational Mathematics. Berlin Heidelberg : Springer-Verlag, Nov. 2004. Chap. Introduction, pp. 1–34 (cit. on p. 16).

Author Index

H

J

K

L

M

N

O

P

Q

R

S

T

V

W

X

Y

Z

List of Titles

Symboles

A

B

C

D

E

F

G

H

I

L

M

N

O

P

S

T

U

V

Abdoulaye Samaké

Large Scale Nonconforming Domain Decomposition Methods

Abstract

This thesis investigates domain decomposition methods, commonly classified as either overlapping Schwarz methods or iterative substructuring methods relying on nonoverlapping subdomains. We mainly focus on the mortar finite element method, a nonconforming approach of substructuring method involving weak continuity constraints on the approximation space. We introduce a finite element framework for the design and the analysis of the substructuring preconditioners for an efficient solution of the linear system arising from such a discretization method. Particular consideration is given to the construction of the coarse grid preconditioner, specifically the main variant proposed in this work, using a Discontinuous Galerkin interior penalty method as coarse problem. Other domain decomposition methods, such as Schwarz methods and the so-called three-field method are surveyed with the purpose of establishing a generic teaching and research programming environment for a wide range of these methods. We develop an advanced computational framework dedicated to the parallel implementation of numerical methods and preconditioners introduced in this thesis. The efficiency and the scalability of the preconditioners, and the performance of parallel algorithms are illustrated by numerical experiments performed on large scale parallel architectures.

Keywords : domain decomposition, mortar finite element method, substructuring preconditioner, high-performance computing.

Résumé

Cette thèse étudie les méthodes de décomposition de domaine généralement classées soit comme des méthodes de Schwarz avec recouvrement ou des méthodes par sous-structuration s'appuyant sur des sous-domaines sans recouvrement. Nous nous focalisons principalement sur la méthode des éléments finis joints, aussi appelée la méthode mortar, une approche non conforme des méthodes par sous-structuration impliquant des contraintes de continuité faible sur l'espace d'approximation. Nous introduisons un framework élément fini pour la conception et l'analyse des préconditionneurs par sous-structuration pour une résolution efficace du système linéaire provenant d'une telle méthode de discrétisation. Une attention particulière est accordée à la construction du préconditionneur grille grossière, notamment la principale variante proposée dans ce travail utilisant la méthode de Galerkin Discontinue avec pénalisation intérieure comme problème grossier. D'autres méthodes de décomposition de domaine, telles que les méthodes de Schwarz et la méthode dite three-field sont étudiées dans l'objectif d'établir un environnement de programmation générique d'enseignement et de recherche pour une large gamme de ces méthodes. Nous développons un framework de calcul avancé et dédié à la mise en oeuvre parallèle des méthodes numériques et des préconditionneurs introduits dans cette thèse. L'efficacité et la scalabilité des préconditionneurs, ainsi que la performance des algorithmes parallèles sont illustrées par des expériences numériques effectuées sur des architectures parallèles à très grande échelle.

Mots-clefs : décomposition de domaine, méthode des éléments finis joints, préconditionneur par sous-structuration, calcul haute performance.