GetDDM: an Open Framework for Testing Optimized Schwarz Methods for Time-Harmonic Wave Problems

B. Thierry[1,2], A.Vion[2], S. Tournier[2,3], M. El Bouajaji[4],
D. Colignon[2], N. Marsic[2], X. Antoine[4], C. Geuzaine[2]

January 11, 2017

Contents

Abstract

We present an open finite element framework, called GetDDM, for testing optimized Schwarz domain decomposition techniques for time-harmonic wave problems. After a review of Schwarz domain decomposition methods and associated transmission conditions, we

[1]Laboratoire J. L. Lions (LJLL), University of Paris VI, Paris, France (thierry@ljll.math.umpc.fr).
[2]Université de Liège, Institut Montefiore B28, B-4000 Liège, Belgium (a.vion@ulg.ac.be, d.colignon@ulg.ac.be, nicolas.marsic@ulg.ac.be, cgeuzaine@ulg.ac.be).
[3]Pontificia Universidad Catolica de Chile, Santiago, Chile (simon.tournier@gmail.com).
[4]Institut Elie Cartan de Lorraine, Université de Lorraine, Inria Nancy-Grand Est, F-54506 Vandoeuvre-lès-Nancy Cedex, France. (melbouaj@gmail.com, xavier.antoine@univ-lorraine.fr).

discuss the implementation, based on the open source software GetDP and Gmsh. The solver, along with ready-to-use examples for Helmholtz and Maxwell's equations, is freely available online for further testing.

Keywords: Domain Decomposition Methods; Parallel computing; Finite Element Method; Acoustics; Electromagnetics.

0 Program Summary

Manuscript title: GetDDM: an Open Framework for Testing Optimized Schwarz Methods for Time-Harmonic Wave Problems
Authors: B. Thierry, A.Vion, S. Tournier, M. El Bouajaji, D. Colignon, N. Marsic, X. Antoine, C. Geuzaine.
Program title: GetDDM
Licensing provisions: Standard CPC licence
Programming language: Gmsh (http://gmsh.info) and GetDP (http://getdp.info)
Computer(s) for which the program has been designed: PC, Mac, Tablets, Computer clusters
Operating system(s) for which the program has been designed: Linux, Windows, MacOSX
RAM required to execute with typical data: From 512 Megabytes upwards.
Has the code been vectorised or parallelized?: Yes
Number of processors used: All available.
Keywords: Domain Decomposition Methods; Parallel computing; Finite Element Method; Acoustics; Electromagnetics.
CPC Library Classification: 4.3, 4.12, 6.5, 10
Nature of problem: Computing the solution of large scale time-harmonic acoustic and electromagnetic wave problems.
Solution method: Finite element method with optimized Schwarz domain decomposition method.
Running time: From a few seconds for simple problems to several days for large-scale simulations.

1 Introduction

We present an open-source framework for testing Schwarz-type domain decomposition methods for time-harmonic wave problems. Such problems are known to be computationally challenging, especially in the high-frequency regime. Among the various approaches that can be used to solve them, the Finite Element Method (FEM) with an Absorbing Boundary Condition (ABC) or a Perfectly Matched Layer (PML) is widely used for its ability to handle complex geometrical configurations and materials with non-homogeneous properties. However, the brute-force application of the FEM in the high-frequency regime leads to the solution of very large, complex-valued and possibly indefinite linear systems [47]. Direct sparse solvers do not scale well for such problems, and Krylov subspace iterative solvers exhibit slow convergence or diverge, while efficiently preconditioning proves difficult [28]. Domain decomposition methods provide an alternative, iterating between subproblems of smaller sizes, amenable to sparse direct solvers [57].

Among the different families of domain decomposition techniques, this work focuses on optimized Schwarz methods [29], which are well suited for time-harmonic wave problems [12, 13, 14, 33, 1, 17, 18, 19, 23, 24, 51, 52, 53] and can be used with or without overlap between the subdomains. The convergence rate of these methods strongly depends on the transmission condition enforced on the interfaces between the subdomains. The optimal convergence is obtained by using as transmission condition on each interface the Dirichlet-to-Neumann (DtN) map [50] related to the complementary of the subdomain of interest [49, 48]. For acoustic waves, this DtN map links the normal derivative and the trace of the acoustic pressure on the interface.

For electromagnetic waves, it links the magnetic and the electric surface currents (and is referred to in this case as the Magnetic-to-Electric, or MtE, map) [23]. However, using the DtN leads to a very expensive numerical procedure in practice, as this operator is non-local. A great variety of techniques based on local transmission conditions have therefore been proposed to build practical algorithms, both for the acoustic case [16, 12, 13, 14, 33] and the electromagnetic one [1, 17, 18, 19, 23, 24, 51, 52, 53]. Recently, PMLs have also been used for this same purpose [56, 27, 59, 60].

The aim of the present paper is twofold. First, it aims to provide a concise review of the most common transmission operators for optimized Schwarz methods applied to time-harmonic acoustic and electromagnetic wave problems, with the corresponding mathematical background. Second, it introduces the flexible finite element framework GetDDM ("a General environment for the treatment of Domain Decomposition Methods") to test and compare them, based on the open source software GetDP[1] [21, 22, 35] and Gmsh[2] [39, 40]. While GetDDM is written in C++, all the problem-specific data (geometry description, finite element formulation with appropriate transmission condition, domain decomposition algorithm) are directly written in input ASCII text files, using the code's built-in language. This general implementation allows to solve a wide variety of problems with the same software, and hides all the complexities of the finite element implementation from the end-user (in particular the MPI-based parallelization). Moreover, the software is designed to work both on small- and medium-scale problems (on a workstation, a laptop, a tablet or even a mobile phone) and on large-scale problems on high-performance computing clusters, without changing the input files. The complete implementation of all the techniques reviewed in this paper is freely available online on the web site of the ONELAB project[3] [36, 37], together with various 2D and 3D sample geometries, for both Helmholtz and Maxwell's equations. While other open source codes provide facilities for domain decomposition methods, either linked to finite element kernels (e.g. FreeFem++ [43] or Feel++ [55] *via* the HPDDM framework [44, 45]), or from a purely algebraic perspective (e.g. PETSc [3]), GetDDM adopts a complementary point of view. It focuses on Schwarz methods where the transmission conditions play a central role and provides a simple, flexible and ready-to-use software environment where the weak formulations of these transmission conditions can be automatically transcribed at the discrete level, with a direct link to their symbolic mathematical and physical structure. The authors hope that this approach will help the scientific community to test and compare different optimized Schwarz domain decomposition techniques by focusing on the mathematics of the transmission conditions, while not having to worry (too much) about the implementation.

The paper is organized as follows. The first section describes the acoustic and the electromagnetic scattering problems and the associated optimized Schwarz methods; the main transmission operators and the weak formulations are given, in view of their transcription in GetDDM. The next section describes the main features of GetDDM and provides code samples for the implementation of domain decomposition problems. Finally, the article concludes with some illustrative numerical results and perspectives for further development.

[1] `http://getdp.info`
[2] `http://gmsh.info`
[3] `http://onelab.info/wiki/GetDDM`

2 Optimized Schwarz Methods for Time-Harmonic Wave Propagation

The mathematical framework of optimized Schwarz methods is presented in this section together with a review of the different transmission operators, for both the acoustic and the electromagnetic cases. As the goal is to implement them using a finite element method, the weak formulations of the different problems are also provided.

2.1 Acoustic Waves: Helmholtz Equation

2.1.1 Mono-Domain Problem

Let us consider an open subset Ω^- of $\mathbb{R}^d$, where $d = 1, 2, 3$ is the dimension, with boundary Γ, such that its complementary $\Omega^+ = \mathbb{R}^d \setminus \overline{\Omega^-}$ is connected. When illuminated by a time-harmonic incident wave u^{inc}, the obstacle Ω^- generates a complex-valued scattered field u, solution of the following problem, where the time dependence is implicit and of the form $e^{-i\omega t}$,

$$\left\{ \begin{array}{rcll} (\Delta + k^2)u & = & 0 & \text{in } \Omega^+, \\ u & = & -u^{\text{inc}} & \text{on } \Gamma, \\ u & & \text{outgoing.} & \end{array} \right. \tag{1}$$

The operator $\Delta = \sum_{i=1}^{d} \partial_{x_i}^2$ is the Laplacian operator and $k = \omega/c$ is the real and strictly positive wavenumber ($c = c(\mathbf{x})$ being the local speed of sound in the medium). In what follows $\mathbf{a} \cdot \overline{\mathbf{b}}$ denotes the inner product between two complex-valued vectors $\mathbf{a}$ and $\mathbf{b}$ in $\mathbb{C}^3$, where $\overline{z}$ is the complex conjugate of $z \in \mathbb{C}$. The associated norm is $||\mathbf{a}|| := \sqrt{\mathbf{a} \cdot \overline{\mathbf{a}}}$. Here, a Dirichlet boundary condition on Γ has been set (i.e. we consider a sound-soft obstacle), but other conditions can be studied such as Neumann, Fourier or even penetrable obstacles. The outgoing condition stands for the Sommerfeld radiation condition ($\imath$ being the square root of -1)

$$\lim_{\|\mathbf{x}\| \to \infty} \|\mathbf{x}\|^{\frac{d-1}{2}} \left(\nabla u \cdot \frac{\mathbf{x}}{\|\mathbf{x}\|} - \imath k u \right) = 0,$$

which ensures that, first, problem (1) is uniquely solvable and, second, that the scattered field u is directed from Ω^- to infinity.

To solve problem (1) using the finite element method, the (unbounded) domain Ω^+ must be truncated, using for example a Perfectly Matched Layer (PML) [8, 15] or a fictitious boundary Γ^∞ with an Absorbing Boundary Condition (ABC) [6, 26] (see e.g. [2] for a review of different methods). With an ABC on a fictitious boundary, the aim is then to find the field $\hat{u}$ approximating u on the bounded domain Ω of boundary $\Gamma^\infty \bigcup \Gamma$. After merging the notations $\hat{u}$ and u for simplicity, the problem to be solved can be written as follows:

$$\left\{ \begin{array}{rcll} (\Delta + k^2)u & = & 0 & \text{in } \Omega, \\ u & = & -u^{\text{inc}} & \text{on } \Gamma, \\ \partial_{\mathbf{n}} u + \mathcal{B}u & = & 0 & \text{on } \Gamma^\infty, \end{array} \right. \tag{2}$$

where the unit normal vector $\mathbf{n}$ is directed outside Ω (and thus inside Ω^- on Γ). The simplest local ABC, i.e., the Sommerfeld radiation condition at finite distance (zeroth-order condition), is obtained by setting

$$\mathcal{B}u = -\imath k u. \tag{3}$$

The extension to more accurate ABCs or PMLs is standard [41].

2.1.2 Domain Decomposition and Transmission Operators

Substructured formulation. The domain Ω is now decomposed into N_{dom} disjoint subdomains Ω_i (the *substructures*) without overlap. For every $i = 0, \ldots, N_{\text{dom}} - 1$, let $\Gamma_i = \Gamma \bigcap \partial\Omega_i$, $\Gamma_i^\infty = \Gamma^\infty \bigcap \partial\Omega_i$; for $j = 0, \ldots, N_{\text{dom}} - 1, j \neq i$, the transmission boundary $\Sigma_{ij} = \Sigma_{ji} = \overline{\partial\Omega_i \bigcap \partial\Omega_j}$ is introduced. To simplify, let $D := \{0, \ldots, N_{\text{dom}} - 1\}$ be the set of indices of the subdomains, and for $i \in D$, let $D_i := \{j \in D \text{ such that } j \neq i \text{ and } \Sigma_{ij} \neq \emptyset\}$ be the set of indices of the subdomains sharing at least a point with Ω_i (we will call them the domains *connected* to Ω_i). Finally, for all $i \in D$, the unit normal $\mathbf{n}_i$ is directed into the exterior of Ω_i and thus inside the obstacle Ω^- (if $\Gamma_i \neq \emptyset$).

The additive Schwarz domain decomposition method can be described as follows, at iteration $n+1$:

1. For all $i \in D$, compute u_i^{n+1} solution to

$$\left\{ \begin{array}{rcll} (\Delta + k^2) u_i^{n+1} & = & 0 & \text{in } \Omega_i, \\ u_i^{n+1} & = & -u^{\text{inc}} & \text{on } \Gamma_i, \\ \partial_{\mathbf{n}_i} u_i^{n+1} + \mathcal{B} u_i^{n+1} & = & 0 & \text{on } \Gamma_i^\infty, \\ \partial_{\mathbf{n}_i} u_i^{n+1} + \mathcal{S} u_i^{n+1} & = & g_{ij}^n & \text{on } \Sigma_{ij}, \quad \forall j \in D_i. \end{array} \right. \tag{4}$$

2. For all $i \in D$ and $j \in D_i$, update the interface unknowns according to:

$$g_{ji}^{n+1} = -\partial_{\mathbf{n}_i} u_i^{n+1} + \mathcal{S} u_i^{n+1} = -g_{ij}^n + 2\mathcal{S} u_i^{n+1}, \qquad \text{on } \Sigma_{ij}. \tag{5}$$

The operator $\mathcal{S}$ is an transmission operator, which will be detailed later. The $(n+1)^{\text{th}}$ iteration of the above algorithm can be rewritten in the following more compact form:

1. For all $i \in D$, compute the volume solution u_i^{n+1} of (4), written as $u_i^{n+1} = \mathscr{V}_i(u^{\text{inc}}, g^n)$, where $g^n = (g_{ji}^n)_{i \in D, j \in D_i}$ is the vector collecting all the interface unknowns.

2. For all $i \in D$ and $j \in D_i$, update the surface fields g_{ji}^{n+1} following (5), written as $g_{ji}^{n+1} = \mathscr{T}_{ji}(g_{ij}^n, u_i^{n+1})$.

In (4) we have only considered the case of Dirichlet sources; other kinds such as volumic sources are of course possible and must be treated in the same way through the algorithm. We will refer to them as *physical* sources, as opposed to the *artificial* sources g_{ij}^n on the transmission boundaries.

The algorithm described by (4) and (5) can be interpreted as a Jacobi iteration applied to a linear operator equation. Indeed, for every $n \in \mathbb{N}$ and by linearity, the field u_i^{n+1} can be decomposed as $u_i^{n+1} = v_i^{n+1} + \widetilde{u}_i^{n+1}$, where

$$v_i^{n+1} = \mathscr{V}_i(u^{\text{inc}}, 0) \qquad \text{and} \qquad \widetilde{u}_i^{n+1} = \mathscr{V}_i(0, g^n). \tag{6}$$

The quantity v_i^{n+1} is independent of the iteration number n and can hence be written as $v_i := v_i^n$, $\forall n \in \mathbb{N}, \forall i \in D$. Equation (5) then becomes

$$g_{ji}^{n+1} = \mathscr{T}_{ji}(g_{ij}^n, u_i^{n+1}) = \mathscr{T}_{ji}(g_{ij}^n, \widetilde{u}_i^{n+1}) + 2\mathcal{S} v_i, \qquad \text{on } \Sigma_{ij}. \tag{7}$$

Let us introduce the vector $b = (b_{ji})_{i \in D, j \in D_i}$, with $b_{ji} = 2(\mathcal{S} v_i)|_{\Sigma_{ij}}$, and the operator $\mathcal{A} : g^n \mapsto \mathcal{A} g^n$ such that

$$\forall i \in D \quad \left\{ \begin{array}{l} \widetilde{u}_i^{n+1} = \mathscr{V}_i(0, g^n), \\ (\mathcal{A} g^n)_{ji} = \mathscr{T}_{ji}(g_{ij}^n, \widetilde{u}_i^{n+1}), \ \forall j \in D_i. \end{array} \right. \tag{8}$$

One iteration of the domain decomposition algorithm then corresponds to

$$g^{n+1} = \mathcal{A}g^n + b, \tag{9}$$

which is one iteration of the Jacobi method applied to the system

$$(\mathcal{I} - \mathcal{A})g = b, \tag{10}$$

where $\mathcal{I}$ is the identity operator. Any iterative linear solver can be applied to (10), as for example Krylov subspace methods such as GMRES [54]. When using a Krylov subspace solver, the method is called a substructured preconditioner [30].

It is worth noting that the iteration unknowns in (9), (10) are the surface quantities g and not the volume unknowns u. Obtaining the volume quantities from the surface unknowns corresponds to solving $u_i = \mathscr{V}_i(u^{\text{inc}}, g)$, on every subdomain Ω_i. A summary of the Schwarz method with Krylov solver is given in Algorithm 1.

Algorithm 1 Schwarz algorithm with Krylov solver.

1. Compute the right hand side b:

$$\begin{cases} \forall i \in D, \quad v_i = \mathscr{V}_i(u^{\text{inc}}, 0), \\ \forall i \in D, \forall j \in D_i, \quad b_{ji} = \mathscr{T}_{ji}(0, v_i). \end{cases}$$

2. Solve system $(\mathcal{I} - \mathcal{A})g = b$ iteratively using a Krylov subspace solver, where $\mathcal{A}$ is defined by (8).

3. At convergence, compute the final solution: $\forall i \in D, \quad u_i = \mathscr{V}_i(u^{\text{inc}}, g)$.

Transmission operator. The convergence rate of the iterative solver is strongly linked to the choice of the transmission operator $\mathcal{S}$ [13]. The optimal transmission operator would be the Dirichlet-to-Neumann (DtN) map for the complement of each subdomain [49, 48], but this operator is however non-local, which makes it very expensive to use computationally. Different local approximations have hence been proposed, based on polynomial or rational approximations of the total symbol of the surface free-space DtN, or by using a volume representation through Perfectly Matched Layers. Among those approximations, four are detailed below and are implemented in GetDDM for a generic transmission boundary Σ:

- *Evanescent Modes Damping Algorithm* [12, 14]:

$$\mathcal{S}_{\text{IBC}(\chi)}u = (-\imath k + \chi)u,$$

  where χ is a real constant. This zeroth-order polynomial approximation is a generalization of the Després condition [17], for which $\chi = 0$. In what follows, we will denote this family of impedance transmission conditions as IBC(χ).

- *Optimized second-order transmission condition* [33]:

$$\mathcal{S}_{\text{GIBC}(a,b)}u = au + b\Delta_\Sigma u, \tag{11}$$

  where Δ_Σ is the Laplace-Beltrami operator on Σ, and a and b are two complex numbers obtained by solving a min-max optimization problem on the rate of convergence. This

condition corresponds to a second-order polynomial approximation of the DtN symbol. In what follows, we will denote this family of generalized impedance transmission conditions as GIBC(a, b). A zeroth-order optimized condition can be constructed in a similar way.

- *Padé-localized square-root transmission condition* [13]:

$$\mathcal{S}_{\mathrm{GIBC}(N_p,\,\alpha,\,\varepsilon)}u = -\imath k C_0 u - \imath k \sum_{\ell=1}^{N_p} A_\ell \mathrm{div}_\Sigma \left(\frac{1}{k_\varepsilon^2}\nabla_\Sigma\right)\left(\mathcal{I} + B_\ell \mathrm{div}_\Sigma\left(\frac{1}{k_\varepsilon^2}\nabla_\Sigma\right)\right)^{-1} u, \quad (12)$$

where

$$k_\varepsilon = k + i\varepsilon, \quad (13)$$

with $\varepsilon = 0.39 k^{1/3}\mathcal{H}^{2/3}$, $\mathcal{H}$ being the local mean curvature of the interface. The coefficients C_0, A_ℓ and B_ℓ are given by

$$C_0 = e^{\imath\alpha/2} R_{N_p}\left(e^{-\imath\alpha} - 1\right), \quad A_\ell = \frac{e^{-\frac{\imath\alpha}{2}} a_\ell}{(1 + b_\ell(e^{-\imath\alpha} - 1))^2}, \quad B_\ell = \frac{e^{-\imath\alpha} b_\ell}{1 + b_\ell(e^{-\imath\alpha} - 1)}, \quad (14)$$

where α is a rotation angle in the complex plane (usually taken as $\pi/4$) and R_{N_p} are the standard real-valued Padé approximation of order N_p of $\sqrt{1+z}$:

$$R_{N_p}(z) = 1 + \sum_{\ell=1}^{N_p} \frac{a_\ell z}{1 + b_\ell z},$$

with

$$a_\ell = \frac{2}{2N_p + 1}\sin^2\left(\frac{\ell\pi}{2N_p+1}\right) \qquad \text{and} \qquad b_\ell = \cos^2\left(\frac{\ell\pi}{2N_p+1}\right). \quad (15)$$

This transmission condition corresponds to a rational approximation of the DtN symbol. In what follows, we will denote this family of generalized impedance transmission conditions as GIBC(N_p, α, ε).

- *PML transmission condition* [56, 27, 59, 60]: The operator $\mathcal{S}_{\mathrm{PML}(\sigma)}$ is constructed by appending a layer Ω^{PML} to the transmission interface, in which a PML transformation with absorption profile σ is applied. For example, in cartesian coordinates, the profile

$$\sigma(x_{\mathrm{PML}}) = \frac{1}{k(x_{\mathrm{PML}} - \delta)}$$

can be used, where δ is the thickness of the PML layer and x_{PML} is the local coordinate inside the PML [9, 46].

All these methods are referred to as optimized Schwarz domain decomposition methods. Note that GIBC(N_p, α, ε) and PML(σ) have in common that they introduce additional unknowns, whereas the other two transmission conditions do not. Also, the first three transmission conditions can be formulated explicitly through sparse surface equations (see e.g. the weak formulations (18)–(23) below), while a sparse formulation of the PML transmission condition requires a volume representation (see e.g. (24)–(25)), a surface representation being dense [58].

2.1.3 Weak Formulations

The finite element method is based on the weak formulations of the partial differential equations. Two different kinds of PDEs are considered when using optimized Schwarz methods: a volume system (here the Helmholtz equation) represented by the operators $\mathscr{V}_i$, and a surface system on the transmission boundaries, represented by $\mathscr{T}_{ji}$. For the sake of clarity, the weak formulations are first given for a generic transmission operator $\mathcal{S}$. For conciseness, we develop the case where there is no contribution on $\partial\Sigma_{ij}$ through integration by parts. Nevertheless, in some situations (e.g. when $\Sigma_{ij} \bigcap \Gamma^\infty \neq \emptyset$), care should be taken to include these terms into the weak formulations.

Generic weak formulations. Without loss of generality, only the case of a particular subdomain Ω_i, for $i \in D$, with no incident wave (homogeneous Dirichlet boundary condition) is detailed. We consider the general setting where PML layers $\Omega_i^{\mathrm{PML}} = \cup_{j\in D_i}\Omega_{ij}^{\mathrm{PML}}$ are potentially appended to the artificial interfaces Σ_{ij}, and define $\Omega_i^* := \Omega_i \cup \Omega_i^{\mathrm{PML}}$. In what follows, the space $H^1(\Omega_i^*) := \{\widetilde{u}_i \in L^2(\Omega_i^*)$ such that $\nabla\widetilde{u}_i \in (L^2(\Omega_i^*))^3\}$ is the classical Sobolev space and $H_0^1(\Omega_i^*)$ is the space of functions $\widetilde{u}_i \in H^1(\Omega_i^*)$ such that $\widetilde{u}_i|_{\Gamma_i} = 0$, which slightly differs from its usual definition (the Dirichlet condition is here set only on part of $\partial\Omega_i^*$).

- The volume PDE $\widetilde{u}_i^{n+1} = \mathscr{V}_i(0, g^n)$ has the following weak formulation:

$$\left\{\begin{array}{l} \text{Find } \widetilde{u}_i^{n+1} \text{ in } H_0^1(\Omega_i^*) \text{ such that, for every } \widetilde{u}_i' \in H_0^1(\Omega_i^*): \\ \displaystyle\int_{\Omega_i} \nabla\widetilde{u}_i^{n+1}\cdot\nabla\widetilde{u}_i' \,\mathrm{d}\Omega_i - \int_{\Omega_i} k^2\widetilde{u}_i^{n+1}\widetilde{u}_i' \,\mathrm{d}\Omega_i + \int_{\Gamma_i^\infty} \mathcal{B}\widetilde{u}_i^{n+1}\widetilde{u}_i' \,\mathrm{d}\Gamma_i^\infty \\ \qquad\qquad + \displaystyle\sum_{j\in D_i}\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}\widetilde{u}_i' \,\mathrm{d}\Sigma_{ij} = \sum_{j\in D_i}\int_{\Sigma_{ij}} g_{ij}^n\widetilde{u}_i' \,\mathrm{d}\Sigma_{ij}. \end{array}\right. \tag{16}$$

- And the surface PDE $g_{ji}^{n+1} = \mathscr{T}_{ji}(g_{ij}^n, \widetilde{u}_i^{n+1})$ has the following one:

$$\left\{\begin{array}{l} \text{Find } g_{ji}^{n+1} \text{ in } H^1(\Sigma_{ij}) \text{ such that, for every } g_{ji}' \in H^1(\Sigma_{ij}): \\ \displaystyle\int_{\Sigma_{ij}} g_{ji}^{n+1}g_{ji}' \,\mathrm{d}\Sigma_{ij} = -\int_{\Sigma_{ij}} g_{ij}^n g_{ji}' \,\mathrm{d}\Sigma_{ij} + 2\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}g_{ji}' \,\mathrm{d}\Sigma_{ij}. \end{array}\right. \tag{17}$$

On the transmission boundaries. Depending on the choice of the transmission operator $\mathcal{S}$, the quantities $\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}\widetilde{u}_i' \,\mathrm{d}\Sigma_{ij}$ and $\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}g_{ji}' \,\mathrm{d}\Sigma_{ij}$ expand as follows:

- IBC(χ):

$$\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}\widetilde{u}_i' \,\mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} (-\imath k + \chi)\widetilde{u}_i^{n+1}\widetilde{u}_i' \,\mathrm{d}\Sigma_{ij}; \tag{18}$$

$$\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}g_{ji}' \,\mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} (-\imath k + \chi)\widetilde{u}_i^{n+1}g_{ji}' \,\mathrm{d}\Sigma_{ij}. \tag{19}$$

- GIBC(a, b):

$$\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}\widetilde{u}_i' \,\mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} a\widetilde{u}_i^{n+1}\widetilde{u}_i' \,\mathrm{d}\Sigma_{ij} - \int_{\Sigma_{ij}} b\nabla\widetilde{u}_i^{n+1}\cdot\nabla\widetilde{u}_i' \,\mathrm{d}\Sigma_{ij}; \tag{20}$$

$$\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}g_{ji}' \,\mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} a\widetilde{u}_i^{n+1}g_{ji}' \,\mathrm{d}\Sigma_{ij} - \int_{\Sigma_{ij}} b\nabla\widetilde{u}_i^{n+1}\cdot\nabla g_{ji}' \,\mathrm{d}\Sigma_{ij}. \tag{21}$$

- GIBC(N_p, α, ε):

$$\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}\widetilde{u}'_i \,\mathrm{d}\Sigma_{ij} := -\imath k C_0 \int_{\Sigma_{ij}} \widetilde{u}_i^{n+1}\widetilde{u}'_i \,\mathrm{d}\Sigma_{ij} + \imath k \sum_{\ell=1}^{N_p} A_\ell \int_{\Sigma_{ij}} \frac{1}{k_\varepsilon^2} \nabla_{\Sigma_{ij}}\varphi_\ell \cdot \nabla_{\Sigma_{ij}}\widetilde{u}'_i \,\mathrm{d}\Sigma_{ij}, \tag{22}$$

where, for every $\ell = 1, \ldots, N_p$, the function φ_ℓ is obtained through the resolution of

$$\begin{cases} \text{Find } \varphi_\ell \text{ in } H^1(\Sigma_{ij}) \text{ such that, for every } \varphi'_\ell \in H^1(\Sigma_{ij}): \\ -\displaystyle\int_{\Sigma_{ij}} \widetilde{u}_i^{n+1}\varphi'_\ell \,\mathrm{d}\Sigma_{ij} - B_\ell \int_{\Sigma_{ij}} \frac{1}{k_\varepsilon^2}\nabla_{\Sigma_{ij}}\varphi_\ell \cdot \nabla_{\Sigma_{ij}}\varphi'_\ell \,\mathrm{d}\Sigma_{ij} + \int_{\Sigma_{ij}} \varphi_\ell \cdot \varphi'_\ell \,\mathrm{d}\Sigma_{ij} = 0; \end{cases}$$

$$\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1} g'_{ji} \,\mathrm{d}\Sigma_{ij} := -\imath k C_0 \int_{\Sigma_{ij}} \widetilde{u}_i^{n+1} g'_{ji} \,\mathrm{d}\Sigma_{ij} - \imath k \sum_{\ell=1}^{N_p} \frac{A_\ell}{B_\ell} \int_{\Sigma_{ij}} (\widetilde{u}_i^{n+1} - \varphi_\ell) g'_{ji} \,\mathrm{d}\Sigma_{ij}. \tag{23}$$

- PML(σ):

$$\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1}\widetilde{u}'_i \,\mathrm{d}\Sigma_{ij} := \int_{\Omega_{ij}^{\mathrm{PML}}} D\nabla\widetilde{u}_i^{n+1} \cdot \nabla\widetilde{u}'_i \,\mathrm{d}\Omega_{ij}^{\mathrm{PML}} - \int_{\Omega_{ij}^{\mathrm{PML}}} k^2 E\, \widetilde{u}_i^{n+1}\widetilde{u}'_i \,\mathrm{d}\Omega_{ij}^{\mathrm{PML}}; \tag{24}$$

$$\int_{\Sigma_{ij}} \mathcal{S}\widetilde{u}_i^{n+1} g'_{ji} \,\mathrm{d}\Sigma_{ij} := \int_{\Omega_{ij}^{\mathrm{PML}}} D\nabla\widetilde{u}_i^{n+1} \cdot \nabla g'_{ji} \,\mathrm{d}\Omega_{ij}^{\mathrm{PML}} - \int_{\Omega_{ij}^{\mathrm{PML}}} k^2 E\, \widetilde{u}_i^{n+1} g'_{ji} \,\mathrm{d}\Omega_{ij}^{\mathrm{PML}}, \tag{25}$$

where $D = \mathrm{diag}(\frac{1}{\gamma_x}, \gamma_x, \gamma_x)$ and $E = \gamma_x$, with $\gamma_x(x_{\mathrm{PML}}) = 1 + \frac{\imath}{\omega}\sigma_x(x_{\mathrm{PML}})$, that is, we consider a 1D PML with an absorption function that grows only in the direction normal to the interface. In (25) the domain of definition of the test functions g'_{ji} on Σ_{ij} is extended to the neighboring PML layer $\Omega_{ij}^{\mathrm{PML}}$, effectively resulting at the discrete level in the integration of the functions associated with the nodes of the interface in the layer of volume elements connected to the interface.

2.2 Electromagnetic Waves: Maxwell's Equations

2.2.1 Mono-Domain Problem

The case of an electromagnetic wave is now considered for an obstacle Ω^- with a smooth boundary Γ and a three dimensional medium. When illuminated by an incident electric field $\mathbf{E}^{\mathrm{inc}}$, a perfectly conducting body Ω^- generates a scattered field $\mathbf{E}$, solution of the following exterior electromagnetic scattering problem:

$$\begin{cases} \mathbf{curl}\ \mathbf{curl}\ \mathbf{E} - k^2\mathbf{E} &= 0, & \text{in } \Omega^+, \\ \gamma^T(\mathbf{E}) &= -\gamma^T(\mathbf{E}), & \text{on } \Gamma, \\ \displaystyle\lim_{\|\mathbf{x}\|\to\infty} \|\mathbf{x}\| \left(\frac{\mathbf{x}}{\|\mathbf{x}\|} \times \mathbf{curl}\ \mathbf{E} + \imath k \mathbf{E} \right) &= 0, \end{cases} \tag{26}$$

where $k := 2\pi/\lambda$ is again the wavenumber and λ the wavelength, $\mathbf{n}$ is the outward unit normal to Ω^+ (thus, inward to the obstacle) and γ^T is the tangential component trace operator

$$\gamma^T : \mathbf{v} \longmapsto \mathbf{n} \times (\mathbf{v} \times \mathbf{n}).$$

The **curl** operator is defined by $\mathbf{curl}\ \mathbf{a} := \nabla \times \mathbf{a}$, for a complex-valued vector field $\mathbf{a} \in \mathbb{C}^3$, and the notation $\mathbf{a} \times \mathbf{b}$ designates the cross product between two complex-valued vectors $\mathbf{a}$ and

b. The last equation of system (26), which is the so-called Silver-Müller radiation condition at infinity, provides the uniqueness of the solution to the scattering boundary-value problem (26).

As for the acoustic case, numerically solving problem (26) with a volume discretization method requires the truncation of the exterior propagation domain with a PML or with an ABC on a fictitious boundary Γ^∞ surrounding Ω^-. For an ABC the problem to be solved is then defined on the bounded domain Ω, with boundaries Γ and Γ^∞:

$$\left\{\begin{array}{rcll} \textbf{curl curl}\,\mathbf{E} - k^2\mathbf{E} & = & 0, & \text{in } \Omega, \\ \gamma^T(\mathbf{E}) & = & -\gamma^T(\mathbf{E}), & \text{on } \Gamma, \\ \gamma^t(\textbf{curl}\,\mathbf{E}) + \mathcal{B}(\gamma^T(\mathbf{E})) & = & 0, & \text{on } \Gamma^\infty, \end{array}\right. \tag{27}$$

with γ^t the tangential trace operator:

$$\gamma^t : \mathbf{v} \longmapsto \mathbf{n} \times \mathbf{v}.$$

As above, the unit normal $\mathbf{n}$ is outwardly directed to Ω and, to simplify, the solution of the above problem is still designated by $\mathbf{E}$. The operator $\mathcal{B}$ is an approximation of the Magnetic-to-Electric (MtE) operator. The well-known Silver-Müller ABC at finite distance is obtained with $\mathcal{B} = \imath k$, similar to (3) for acoustics modulo the sign (due to the trace operator definitions). The extension to more accurate ABCs or PMLs is standard.

2.2.2 Domain Decomposition and Transmission operators

Substructured formulation. The optimized Schwarz domain decomposition without overlap is now considered for the Maxwell problem (27). The domain Ω is decomposed as described in paragraph 2.1.2 and the same notations are used (recalling that $D = \{0, \ldots, N_{\text{dom}} - 1\}$ and, for $i \in D$, $D_i = \{j \in D \text{ s.t. } j \neq i \text{ and } \Sigma_{ij} \neq \emptyset\}$). The iterative Jacobi algorithm for the computation of the electric fields $(\mathbf{E}_i^{n+1})_{i\in D}$ at iteration $n+1$ involves, first, the solution of the N_{dom} following problems

$$\left\{\begin{array}{rcll} \textbf{curl curl}\,\mathbf{E}_i^{n+1} - k^2\,\mathbf{E}_i^{n+1} & = & \mathbf{0}, & \text{in } \Omega_i, \\ \gamma_i^T(\mathbf{E}_i^{n+1}) & = & -\gamma_i^T(\mathbf{E}^{\text{inc}}), & \text{on } \Gamma_i, \\ \gamma_i^t(\textbf{curl}\,\mathbf{E}_i^{n+1}) + \mathcal{B}(\gamma_i^T(\mathbf{E}_i^{n+1})) & = & \mathbf{0}, & \text{on } \Gamma_i^\infty, \\ \gamma_i^t(\textbf{curl}\,\mathbf{E}_i^{n+1}) + \mathcal{S}(\gamma_i^T(\mathbf{E}_i^{n+1})) & = & \mathbf{g}_{ij}^n, & \text{on } \Sigma_{ij},\ \forall j \in D_i, \end{array}\right. \tag{28}$$

and then forming the quantities $\mathbf{g}_{ji}^{n+1}$ through

$$\mathbf{g}_{ji}^{n+1} = \gamma_i^t(\textbf{curl}\,\mathbf{E}_i^{n+1}) + \mathcal{S}(\gamma_i^T(\mathbf{E}_i^{n+1})) = -\mathbf{g}_{ij}^n + 2\mathcal{S}(\gamma_i^T(\mathbf{E}_i^{n+1})), \quad \text{on } \Sigma_{ij}, \tag{29}$$

where, for $i \in D$, $\mathbf{E}_i = \mathbf{E}_{|\Omega_i}$, $\mathcal{S}$ is a transmission operator through the interfaces Σ_{ij} and γ_i^t and γ_i^T are the local tangential trace and tangential component trace operators:

$$\gamma_i^t : \mathbf{v}_i \longmapsto \mathbf{n}_i \times \mathbf{v}_{i|\partial\Omega_i} \quad \text{and} \quad \gamma_i^T : \mathbf{v}_i \longmapsto \mathbf{n}_i \times (\mathbf{v}_{i|\partial\Omega_i} \times \mathbf{n}_i),$$

with $\mathbf{n}_i$ the outward-pointing unit normal to Ω_i.

Following the same procedure as in section 2.1.2, we introduce the two families of operators $(\mathscr{V}_i)_{i\in D}$ and $(\mathscr{T}_{ji})_{i\in D, j\in D_i}$ as:

1. $\mathbf{E}_i^{n+1} = \mathscr{V}_i(\mathbf{E}^{\text{inc}}, \mathbf{g}^n) \Longleftrightarrow \mathbf{E}_i^{n+1}$ is solution of problem (28), where $\mathbf{g}^n = (\mathbf{g}_{ji}^n)_{i\in D, j\in D_i}$ collects all the unknowns at iteration n;
2. $\mathbf{g}_{ji}^{n+1} = \mathscr{T}_{ji}(\mathbf{g}_{ij}^n, \mathbf{E}_i^{n+1}) \Longleftrightarrow \mathbf{g}_{ji}^{n+1}$ is solution of problem (29).

By linearity, we decompose the field $\mathbf{E}_i^{n+1}$ as $\mathbf{E}_i^{n+1} = \mathbf{F}_i^{n+1} + \widetilde{\mathbf{E}}_i^{n+1}$, where

$$\mathbf{F}_i^{n+1} = \mathscr{V}_i(\mathbf{E}^{\text{inc}}, 0) \qquad \text{and} \qquad \widetilde{\mathbf{E}}_i^{n+1} = \mathscr{V}_i(0, \mathbf{g}^n). \tag{30}$$

The quantity $\mathbf{F}_i^{n+1}$ is independent of the iteration number n and can hence be written as $\mathbf{F}_i := \mathbf{F}_i^n,\ \forall n \in \mathbb{N}, \forall i \in D$. The whole algorithm can then be recast into a linear system:

$$(\mathcal{I} - \mathcal{A})\,\mathbf{g} = \mathbf{b}, \tag{31}$$

that can be solved by a Krylov subspace solver.

As in the acoustic case, for a vector $\mathbf{g}^n$, the quantity $\mathcal{A}\mathbf{g}^n$ is given by, for $i \in D$ and $j \in D_i$, $(\mathcal{A}\mathbf{g}^n)_{ji} = \mathscr{T}_{ji}\left(\mathbf{g}_{ij}^n, \widetilde{\mathbf{E}}_i^{n+1}\right)$. The information about the incident wave is contained in the right-hand side: $\mathbf{b}_{ji} = \mathscr{T}_{ji}\,(0, \mathbf{F}_i)$. The domain decomposition algorithm for the Maxwell system is then exactly the same as the one described in Algorithm 1 for the Helmholtz equation, by formally replacing v_i, u^{inc}, g and u_i by $\mathbf{F}_i, \mathbf{E}^{\text{inc}}, \mathbf{g}$ and $\mathbf{E}_i$, respectively.

Transmission operator. Similarly to the acoustic case, optimal convergence of the domain decomposition algorithm would be achieved by using the (non-local) MtE operator as transmission condition. Local approximations based on polynomial or rational approximations of the total symbol of the surface free-space MtE have been proposed, as well as volume representations through Perfectly Matched Layers. Among those approximations, four are detailed below and are implemented in GetDDM for a generic transmission boundary Σ:

- *Zeroth-order transmission condition* [17]:

$$\mathcal{S}_{\text{IBC}(0)}(\gamma^T(\mathbf{E})) = \imath k \gamma^T(\mathbf{E}). \tag{32}$$

- *Optimized second-order transmission condition* [53]:

$$\mathcal{S}_{\text{GIBC}(a,b)}(\gamma^T(\mathbf{E})) = \imath k \left(\mathcal{I} + \frac{a}{k^2}\nabla_\Sigma \text{div}_\Sigma\right)^{-1} \left(\mathcal{I} - \frac{b}{k^2}\mathbf{curl}_\Sigma \text{curl}_\Sigma\right)(\gamma^T(\mathbf{E})), \tag{33}$$

  where the curl operator is the dual operator of **curl** and where a and b are chosen so that an optimal convergence rate is obtained for the (TE) and (TM) modes; see [53] for the expression of a and b in the half-plane case. A zeroth-order optimized transmission condition using a single second-order operator was proposed in [1].

- *Padé-localized square-root transmission condition* [23, 25]:

$$\mathcal{S}_{\text{GIBC}(N_p,\,\alpha,\,\varepsilon)}(\gamma^T(\mathbf{E})) = \imath k \left(C_0 + \sum_{\ell=1}^{N_p} A_\ell X \left(\mathcal{I} + B_\ell X\right)^{-1}\right)^{-1} \left(\mathcal{I} - \mathbf{curl}_\Sigma \frac{1}{k_\varepsilon^2}\text{curl}_\Sigma\right)(\gamma^T(\mathbf{E})), \tag{34}$$

  with $X := \nabla_\Sigma \frac{1}{k_\varepsilon^2}\text{div}_\Sigma - \mathbf{curl}_\Sigma \frac{1}{k_\varepsilon^2}\text{curl}_\Sigma$, and where k_ε, C_0, A_ℓ and B_ℓ are defined by (13) and (14). This transmission condition corresponds to a rational approximation of the MtE symbol, generalizing the polynomial approximations underlying (32) and (33).

- *PML transmission condition* [59, 60]: The operator $\mathcal{S}_{\text{PML}(\sigma)}$ is constructed by appending a layer Ω^{PML} to the transmission interface, into which a PML transformation with absorption profile σ is applied in the same way as for the acoustic case.

2.2.3 Weak Formulations

Generic weak formulations. Without loss of generality, only the case of a particular subdomain Ω_i, for $i \in D$, with no incident wave (homogeneous Dirichlet boundary condition) is detailed. We consider the same general setting as in the acoustic case, i.e., where PML layers $\Omega_i^{\mathrm{PML}} = \cup_{j\in D_i}\Omega_{ij}^{\mathrm{PML}}$ are potentially appended to the artificial interfaces Σ_{ij}, and define $\Omega_i^* := \Omega_i \cup \Omega_i^{\mathrm{PML}}$. The space of complex-valued curl-conforming vector fields on Ω_i^* is denoted by $\mathbf{H}(\mathbf{curl}, \Omega_i^*) := \{\mathbf{W} \in (L^2(\Omega_i^*))^3 \text{ such that } \mathbf{curl}(\mathbf{W}) \in (L^2(\Omega_i^*))^3\}$. The functional space $\mathbf{H}_0(\mathbf{curl}, \Omega_i^*)$ is the space of functions $\mathbf{W}_i$ in $\mathbf{H}(\mathbf{curl}, \Omega_i^*)$ such that $\gamma_i^T(\mathbf{W}_i) = 0$ on $\Gamma_i = 0$ (the boundary condition is only imposed on a part $\partial\Omega_i^*$).

- The volume PDE $\widetilde{\mathbf{E}}_i^{n+1} = \mathscr{V}_i(0, \mathbf{g}^n)$ has the following weak formulation:

$$\left\{\begin{array}{l} \text{Find } \widetilde{\mathbf{E}}_i^{n+1} \in \mathbf{H}_0(\mathbf{curl}, \Omega_i) \text{ such that, for every } \widetilde{\mathbf{E}}_i' \in \mathbf{H}_0(\mathbf{curl}, \Omega_i): \\ \displaystyle\int_{\Omega_i} \mathbf{curl}\, \widetilde{\mathbf{E}}_i^{n+1} \cdot \mathbf{curl}\, \widetilde{\mathbf{E}}_i' \,\mathrm{d}\Omega_i - \int_{\Omega_i} k^2 \widetilde{\mathbf{E}}_i^{n+1} \cdot \widetilde{\mathbf{E}}_i' \,\mathrm{d}\Omega_i - \int_{\Gamma_i^\infty} \mathcal{B}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \widetilde{\mathbf{E}}_i' \,\mathrm{d}\Gamma_i^\infty \\ \qquad\qquad - \displaystyle\sum_{j\in D_i} \int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \widetilde{\mathbf{E}}_i' \,\mathrm{d}\Sigma_{ij} = -\sum_{j\in D_i} \int_{\Sigma_{ij}} \mathbf{g}_{ij}^n \cdot \widetilde{\mathbf{E}}_i' \,\mathrm{d}\Sigma_{ij}. \end{array}\right. \tag{35}$$

- The surface PDE $\mathbf{g}_{ji}^{n+1} = \mathscr{T}_{ji}(\mathbf{g}_{ij}^n, \widetilde{\mathbf{E}}_i^{n+1})$ has the following one:

$$\left\{\begin{array}{l} \text{Find } \mathbf{g}_{ji}^{n+1} \text{ in } \mathbf{H}(\mathbf{curl}, \Sigma_{ij}) \text{ such that, for every } \mathbf{g}_{ji}' \in \mathbf{H}(\mathbf{curl}, \Sigma_{ij}): \\ \displaystyle\int_{\Sigma_{ij}} \mathbf{g}_{ji}^{n+1} \cdot \mathbf{g}_{ji}' \,\mathrm{d}\Sigma_{ij} = -\int_{\Sigma_{ij}} \mathbf{g}_{ij}^n \cdot \mathbf{g}_{ji}' \,\mathrm{d}\Sigma_{ij} + 2\int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \mathbf{g}_{ji}' \,\mathrm{d}\Sigma_{ij}. \end{array}\right.$$

On the transmission boundaries.

- IBC(0):

$$\int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \widetilde{\mathbf{E}}_i' \,\mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} \imath k(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \widetilde{\mathbf{E}}_i' \,\mathrm{d}\Sigma_{ij}; \tag{36}$$

$$\int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \mathbf{g}_{ji}' \,\mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} \imath k(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \mathbf{g}_{ji}' \,\mathrm{d}\Sigma_{ij}. \tag{37}$$

- GIBC(a, b):

$$\int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \widetilde{\mathbf{E}}_i' \,\mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} \imath k\mathbf{r} \cdot \widetilde{\mathbf{E}}_i' \,\mathrm{d}\Sigma_{ij}, \tag{38}$$

where the function $\mathbf{r} \in \mathbf{H}(\mathbf{curl}, \Sigma_{ij})$ is obtained through the solution of

$$\left\{\begin{array}{l} \text{Find } \mathbf{r} \text{ in } \mathbf{H}(\mathbf{curl}, \Sigma_{ij}) \text{ and } \rho \text{ in } H^1(\Sigma_{ij}) \text{ such that } \forall \mathbf{r}' \in \mathbf{H}(\mathbf{curl}, \Sigma_{ij}) \\ \text{and } \forall \rho' \in H^1(\Sigma_{ij}): \\ \displaystyle -\int_{\Sigma_{ij}} \frac{a}{k^2} \nabla_{\Sigma_{ij}}\rho \cdot \mathbf{r}' \,\mathrm{d}\Sigma_{ij} - \int_{\Sigma_{ij}} \mathbf{r} \cdot \mathbf{r}' \,\mathrm{d}\Sigma_{ij} + \int_{\Sigma_{ij}} \gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1}) \cdot \mathbf{r}' \,\mathrm{d}\Sigma_{ij} \\ \qquad\qquad - \displaystyle\int_{\Sigma_{ij}} \frac{b}{k^2} \mathrm{curl}_{\Sigma_{ij}}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \,\mathrm{curl}_{\Sigma_{ij}}\mathbf{r}' \,\mathrm{d}\Sigma_{ij} = 0, \\ \displaystyle\int_{\Sigma_{ij}} \rho\rho' \,\mathrm{d}\Sigma_{ij} + \int_{\Sigma_{ij}} \mathbf{r} \cdot \nabla_{\Sigma_{ij}}\rho' \,\mathrm{d}\Sigma_{ij} = 0; \end{array}\right. \tag{39}$$

$$\int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \mathbf{g}_{ji}' \,\mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} \imath k\mathbf{r} \cdot \mathbf{g}_{ji}' \,\mathrm{d}\Sigma_{ij}. \tag{40}$$

- GIBC(N_p, α, ε):

$$\int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \widetilde{\mathbf{E}}_i' \, \mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} \imath k \mathbf{r} \cdot \widetilde{\mathbf{E}}_i' \, \mathrm{d}\Sigma_{ij}, \tag{41}$$

where the function $\mathbf{r} \in \mathbf{H}(\mathbf{curl}, \Sigma_{ij})$ is obtained through the solution of

$$\begin{cases} \text{Find } \mathbf{r} \text{ in } \mathbf{H}(\mathbf{curl}, \Sigma_{ij}), \text{ and for } \ell = 1, \dots, N_p, \boldsymbol{\varphi}_\ell \text{ in } \mathbf{H}(\mathbf{curl}, \Sigma_{ij}) \text{ and } \rho_\ell \text{ in } H^1(\Sigma_{ij}) \\ \text{such that } \forall \mathbf{r}' \in \mathbf{H}(\mathbf{curl}, \Sigma_{ij}), \forall \boldsymbol{\varphi}_\ell' \in \mathbf{H}(\mathbf{curl}, \Sigma_{ij}) \text{ and } \forall \rho_\ell' \in H^1(\Sigma_{ij}): \\ \displaystyle\int_{\Sigma_{ij}} C_0 \mathbf{r} \cdot \mathbf{r}' \, \mathrm{d}\Sigma_{ij} - \int_{\Sigma_{ij}} \gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1}) \cdot \mathbf{r}' \, \mathrm{d}\Sigma_{ij} + \int_{\Sigma_{ij}} \frac{1}{k_\varepsilon^2} \mathrm{curl}_{\Sigma_{ij}}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \, \mathrm{curl}_{\Sigma_{ij}} \mathbf{r}' \, \mathrm{d}\Sigma_{ij} \\ \qquad + \displaystyle\sum_{\ell=1}^{N_p} A_\ell \left[\int_{\Sigma_{ij}} \nabla_{\Sigma_{ij}} \rho_\ell \cdot \mathbf{r}' \, \mathrm{d}\Sigma_{ij} - \int_{\Sigma_{ij}} \frac{1}{k_\varepsilon^2} \mathrm{curl}_{\Sigma_{ij}} \boldsymbol{\varphi}_\ell \, \mathrm{curl}_{\Sigma_{ij}} \mathbf{r}' \, \mathrm{d}\Sigma_{ij} \right] = 0, \\ \displaystyle\int_{\Sigma_{ij}} \boldsymbol{\varphi}_\ell \cdot \boldsymbol{\varphi}_\ell' \, \mathrm{d}\Sigma_{ij} + B_\ell \left[\int_{\Sigma_{ij}} \nabla_{\Sigma_{ij}} \rho_\ell \cdot \boldsymbol{\varphi}_\ell' \, \mathrm{d}\Sigma_{ij} - \int_{\Sigma_{ij}} \frac{1}{k_\epsilon^2} \mathrm{curl}_{\Sigma_{ij}} \boldsymbol{\varphi}_\ell \, \mathrm{curl}_{\Sigma_{ij}} \boldsymbol{\varphi}_\ell' \, \mathrm{d}\Sigma_{ij} \right] \\ \qquad - \displaystyle\int_{\Sigma_{ij}} \mathbf{r} \cdot \boldsymbol{\varphi}_\ell' \, \mathrm{d}\Sigma_{ij} = 0, \quad \ell = 1, \dots, N_p, \\ \displaystyle\int_{\Sigma_{ij}} \rho_\ell \rho_\ell' \, \mathrm{d}\Sigma_{ij} + \int_{\Sigma_{ij}} \frac{1}{k_\varepsilon^2} \boldsymbol{\varphi}_\ell \cdot \nabla_{\Sigma_{ij}} \rho_\ell' \, \mathrm{d}\Sigma_{ij} = 0, \quad \ell = 1, \dots, N_p; \end{cases} \tag{42}$$

$$\int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \mathbf{g}_{ji}' \, \mathrm{d}\Sigma_{ij} := \int_{\Sigma_{ij}} \imath k \mathbf{r} \cdot \mathbf{g}_{ji}' \, \mathrm{d}\Sigma_{ij}. \tag{43}$$

- PML(σ):

$$\begin{aligned} \int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \widetilde{\mathbf{E}}_i' \, \mathrm{d}\Sigma_{ij} := & \int_{\Omega_{ij}^{\mathrm{PML}}} D^{-1} \, \mathbf{curl} \, \widetilde{\mathbf{E}}_i^{n+1} \cdot \mathbf{curl} \, \widetilde{\mathbf{E}}_i' \, \mathrm{d}\Omega_{ij}^{\mathrm{PML}} \\ & - \int_{\Omega_{ij}^{\mathrm{PML}}} D \, k^2 \widetilde{\mathbf{E}}_i^{n+1} \cdot \widetilde{\mathbf{E}}_i' \, \mathrm{d}\Omega_{ij}^{\mathrm{PML}}; \end{aligned} \tag{44}$$

$$\begin{aligned} \int_{\Sigma_{ij}} \mathcal{S}(\gamma_i^T(\widetilde{\mathbf{E}}_i^{n+1})) \cdot \mathbf{g}_{ji}' \, \mathrm{d}\Sigma_{ij} := & \int_{\Omega_{ij}^{\mathrm{PML}}} D^{-1} \, \mathbf{curl} \, \widetilde{\mathbf{E}}_i^{n+1} \cdot \mathbf{curl} \, \mathbf{g}_{ji}' \, \mathrm{d}\Omega_{ij}^{\mathrm{PML}} \\ & - \int_{\Omega_{ij}^{\mathrm{PML}}} D \, k^2 \widetilde{\mathbf{E}}_i^{n+1} \cdot \mathbf{g}_{ji}' \, \mathrm{d}\Omega_{ij}^{\mathrm{PML}}, \end{aligned} \tag{45}$$

where the tensor D is defined as for the acoustic case and the test functions $\mathbf{g}_{ji}'$ are again extended to the volume of the PML layers.

3 Implementation in GetDDM

GetDDM is based on the open source finite element solver GetDP (`http://getdp.info`) and the open source mesh generator Gmsh (`http://gmsh.info`). The complete implementation of all the techniques reviewed in this paper is freely available online on the web site of the ONELAB project [36, 37], at the following address: `http://onelab.info/wiki/GetDDM`. Various 2D and 3D test-cases are provided online (see also Section 4) for both Helmholtz and Maxwell's equations, as well as detailed instructions on how to build the software for parallel computer architectures. Pre-compiled, serial versions of the software for Windows, MacOS and Linux are also available for development and testing.

By default, GetDDM uses Gmsh [39, 40] for geometry description, mesh generation and data visualization. While any other CAD system and mesh generator can also be used, Gmsh provides a tight integration with GetDP through the ONELAB interface, which permits a seamless modification of the various model and solver parameters. In what follows, as we focus on the behavior of the domain decomposition methods with various optimized transmission conditions, we will consider relatively simple geometrical configurations, where the subdomain partitioning is carried out at the level of the CAD. Also, no special treatment of cross-points is considered. More complex configurations can however be treated—see Section 4.

Once a suitable finite element mesh is generated, GetDDM uses GetDP to assemble and solve the finite element problem in parallel. GetDP (a "General environment for the Treatment of Discrete Problems") [21, 22, 35] uses mixed elements to discretize de Rham-type complexes in one, two and three dimensions. Its main feature is the closeness between the input data defining discrete problems and the symbolic mathematical expressions of these problems, translated into ten inter-dependent objects. As mentioned in the introduction, while GetDP is written in C++, the problem definition is directly written in input ASCII text files (`.pro` files), using GetDP's own problem definition language. This allows for example to write weak forms of (28) together with either (32), (33) or (34) directly in the input data files, and use the natural mixed finite element spaces suitable for discretization [34, 23]. In practice, a problem definition written in `.pro` input files is usually split between the objects defining data particular to a given problem, such as geometry, physical characteristics and boundary conditions (i.e., the `Group`, `Function` and `Constraint` objects), and those defining a resolution method, such as unknowns, equations and related objects (i.e., the `Jacobian`, `Integration`, `FunctionSpace`, `Formulation`, `Resolution` and `PostProcessing` objects). The processing cycle ends with the presentation of the results, using the `PostOperation` object. This decomposition points out the possibility of building black boxes adapted to the treatment of general classes of problems that share the same resolution methods—typically what is done in GetDDM, where, as will be explained below, the `Schwarz.pro` and `Decomposition.pro` files contain the generic domain decomposition algorithms, and the `Helmholtz.pro` and `Maxwell.pro` contain the physics-specific function spaces and weak formulations, used in all the other particular problem definition files (e.g. `waveguide3d.pro` or `marmousi.pro`). Of particular significance is that the input `.pro` files are not interpreted at run-time: they are analyzed once before the computation is run; then the computation (finite element assembly, system solution, etc.) is carried out fully automatically in compiled code, the parallel linear algebra being handled by the open-source PETSc solvers [3, 4, 5].

The remainder of this section covers in detail the implementation in the GetDP language (in `.pro` files) of the domain decomposition algorithms presented in Section 2, from the weak formulations to the parallel iterative solution of the resulting linear systems. The focus is on advanced techniques required to implement the domain decomposition algorithms efficiently; for an introduction to GetDP and its various more elementary features, the reader is referred to the GetDP reference manual [20] and is encouraged to explore the numerous examples available on the website of the ONELAB project (`http://onelab.info`). In a similar way, the elementary geometry and meshing features of Gmsh are not described here: the reader is referred to the Gmsh reference manual [38] and the various `.geo` files provided on the ONELAB site for additional information.

3.1 Weak Formulations and Finite Element Discretization

Before describing the weak formulations, we briefly introduce the discrete functional spaces based on the appropriate finite element functions available in GetDDM. For the Helmholtz problem

```
FunctionSpace {
  { Name Hgrad_u~{idom}; Type Form0;
    BasisFunction {
      { Name sn; NameOfCoef un; Function BF_Node;
        Support Region[ {Omega~{idom}, Pml~{idom}, Sigma~{idom},
                         GammaInf~{idom}} ];
        Entity NodesOf[ All ];
      }
    }
    Constraint {
      { NameOfCoef un; EntityType NodesOf;
        NameOfConstraint Dirichlet_u~{idom}; }
      { NameOfCoef un; EntityType NodesOf;
        NameOfConstraint Dirichlet_u0~{idom}; }
    }
  }
}
```

Listing 1: Approximation space of $H^1(\Omega_i^*)$ with Dirichlet boundary conditions, using standard P1 basis functions (associated with the nodes of the finite element mesh). (Code extracted from the file `Helmholtz.pro`.)

```
      { Name sn2; NameOfCoef un2; Function BF_Node_2E;
        Support Region[ {Omega~{idom}, Pml~{idom}, Sigma~{idom},
                         GammaInf~{idom}} ];
        Entity EdgesOf[ All ];
      }
```

Listing 2: Additional edge-based basis functions for a second-order, hierarchical approximation space of $H^1(\Omega_i^*)$.

with a Dirichlet boundary condition, we naturally consider the classical linear finite element (P1) space, using the built-in `BF_Node` basis functions (the `Form0` type refers to functions in H^1). The boundary conditions are imposed strongly as constraints on the coefficients in the expansion of the approximate solution in terms of the finite element basis: see Listing 1. Going to second-order using hierarchical polynomials would simply consist in adding another `BasisFunction` in the `FunctionSpace`; in addition to the nodal degrees of freedom, the finite element expansion now also uses degrees of freedom associated with the edges of the mesh: see Listing 2. Discrete function spaces for surface unknowns are written similarly, the only difference being the `Support`. For example, the approximation space for the auxiliary functions φ_ℓ in $H^1(\Sigma_{ij})$ in (22) is simply constructed by specifying `Support Sigma~{idom}` in the `FunctionSpace`.

Even if the Maxwell problem is more complicated, its implementation in GetDDM is also quite straightforward. Indeed, the natural approximation space for the vector unknowns is the Whitney edge element space [11], which is directly available through the `BF_Edge` basis functions (with type `Form1`, referring to functions in $\mathbf{H}(\mathbf{curl})$), for both the volume and surface unknowns (see Listing 3). More details about the functional framework for the Maxwell problem can be found in [25]. Let us also remark that, for the sake of conciseness, we choose to present the standard weak formulations for the Helmholtz and Maxwell's equations. Other formulations (dual, mixed), associated finite element spaces (Raviart-Thomas, discontinuous, ...) and other equations (elastodynamics, Schrödinger, ...) could be developed as well in the same GetDDM

```
FunctionSpace {
  { Name Hcurl_e~{idom}; Type Form1;
    BasisFunction {
      { Name se; NameOfCoef ee; Function BF_Edge;
        Support Region[ {Omega~{idom}, Pml~{idom}, Sigma~{idom},
                         GammaInf~{idom}} ];
        Entity EdgesOf[ All ]; }
    }
    Constraint {
      { NameOfCoef ee; EntityType EdgesOf;
        NameOfConstraint Dirichlet_e0~{idom}; }
    }
  }
}
```

Listing 3: Approximation space of $\mathbf{H}(\mathbf{curl}, \Omega_i^*)$. (Code extracted from the file `Maxwell.pro`.)

framework.

Following Algorithm 1, for every subdomain, the weak formulation solved at each iteration remains the same, up to the right-hand side. The input file contains only one weak formulation, indexed by the subdomain number `idom` and encapsulated in a `For` loop. All the considered transmission conditions are also regrouped in the same file and selected according to the `TC_TYPE` parser variable[4]. The weak formulation for the volume system (16) is presented in Listing 4 without the transmission boundary condition terms (18)–(24), which are detailed in Listings 5–8 respectively. In the `Formulation` object `[.,.]` inside a `Galerkin` term denotes an inner product for building linear or bilinear forms, with test functions by convention to the right of the comma. Unknown quantities (in bilinear forms) are marked with the `Dof{}` operator; `d` represents the exterior derivative. The `$PhysicalSource` run-time variable (resp. `$ArtificalSource`) specifies whether the physical (resp. transmission) boundary condition is set or not. The `iSide` parser variable is used to indicate the "left" or the "right" transmission boundaries of a given subdomain (see Section 3.2.1), which is necessary for sweeping-type preconditioners (see Section 3.3) [59]. In the absence of preconditioning, the "left"/"right" distinction is not mandatory, and all the quantities and domains can be regrouped in either one. In the same way as for the volume part, the surface equation (17) is also directly transcribed as a `Formulation` in the input file: the relevant `Equation` is presented in Listing 9. Note that, for the GIBC(N_p, α, ε) condition (22), the auxiliary quantities φ_ℓ have already been computed during the volume resolution (Listing 7) and are here re-used (without `Dof{}`). The quantities `g_in` refer to the (incoming) functions g_{ji} (16) (see Section 3.2.2). Finally, for conciseness, the part of the implementation which is standard in GetDP is not detailed here, that is for example the definition of functions (`Function` objects) and the construction of integration rules (`Jacobian` and `Integration` objects). As an example, for a circular absorbing boundary condition of radius `R_EXT`, the two functions `alphaBT[]` and `betaBT[]` in Listing 4 are defined as:

```
alphaBT[]  =  1/(2*R_EXT) - I[]/(8*k*R_EXT^2*(1+I[]/(k*R_EXT)));
 betaBT[]  =  - 1/(2*I[]*k*(1+I[]/(k*R_EXT)));
```

where the function `I[]` is defined as `I[] = Complex[0, 1]`.

```
Formulation {
  { Name Vol~{idom}; Type FemEquation;
    Quantity {
      { Name u~{idom}; Type Local; NameOfSpace Hgrad_u~{idom}; }
    }
    Equation {
      Galerkin { [ Dof{d u~{idom}}, {d u~{idom}} ];
        In Omega~{idom}; Jacobian JVol; Integration I1; }
      Galerkin { [ - k[]^2 * Dof{u~{idom}}, {u~{idom}} ];
        In Omega~{idom}; Jacobian JVol; Integration I1; }

      // artificial sources on transmission boundaries (iSide split only
      // useful for sweeping-type preconditioners)
      For iSide In {0:1}
      Galerkin { [ - ($ArtificialSource~{iSide} ? g_in~{idom}~{iSide}[] : 0),
        {u~{idom}} ];
        In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
      EndFor

      // Bayliss-Turkel absorbing boundary condition
      Galerkin { [ - I[] * k[] * Dof{u~{idom}} , {u~{idom}} ];
        In GammaInf~{idom}; Jacobian JSur; Integration I1; }
      Galerkin { [ alphaBT[] * Dof{u~{idom}} , {u~{idom}} ];
        In GammaInf~{idom}; Jacobian JSur; Integration I1; }
      Galerkin { [ betaBT[] * Dof{d u~{idom}} , {d u~{idom}} ];
        In GammaInf~{idom}; Jacobian JSur; Integration I1; }

      // transmission condition (see next 4 Listings)
      ⟨···⟩
    }
  }
}
```

Listing 4: Volume system of the Helmholtz equation (16). (Code extracted from the file `Helmholtz.pro`. For conciseness the declaration of the auxiliary functions `phi` in `Quantity` is omitted.)

```
      If(TC_TYPE == 0) // IBC
        Galerkin { [ - I[] * kIBC[] * Dof{u~{idom}} , {u~{idom}} ];
          In Sigma~{idom}; Jacobian JSur; Integration I1; }
      EndIf
```

Listing 5: Impedance boundary condition IBC(χ) transmission condition (18). (Code extracted from the file `Helmholtz.pro`.)

```
      If(TC_TYPE == 1) // GIBC(a, b)
        Galerkin { [ a[] * Dof{u~{idom}} , {u~{idom}} ];
          In Sigma~{idom}; Jacobian JSur; Integration I1; }
        Galerkin { [ - b[] * Dof{d u~{idom}} , {d u~{idom}} ];
          In Sigma~{idom}; Jacobian JSur; Integration I1; }
      EndIf
```

Listing 6: GIBC(a, b) transmission condition (20). (Code extracted from the file `Helmholtz.pro`.)

```
If(TC_TYPE == 2) // GIBC(NP_OSRC, theta_branch, eps)
  Galerkin { [ - I[] * k[] * OSRC_C0[]{NP_OSRC,theta_branch} * Dof{u~{idom}} ,
      {u~{idom}} ];
    In Sigma~{idom}; Jacobian JSur; Integration I1; }
  For iSide In {0:1}
    For j In{1:NP_OSRC}
      Galerkin { [ I[] * k[] * OSRC_Aj[]{j,NP_OSRC,theta_branch} / keps[]^2 *
          Dof{d phi~{j}~{idom}~{iSide}} , {d u~{idom}} ];
        In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }

      Galerkin { [ - Dof{u~{idom}} , {phi~{j}~{idom}~{iSide}} ];
        In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
      Galerkin { [ - OSRC_Bj[]{j,NP_OSRC,theta_branch} / keps[]^2 *
          Dof{d phi~{j}~{idom}~{iSide}} , {d phi~{j}~{idom}~{iSide}} ];
        In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
      Galerkin { [ Dof{phi~{j}~{idom}~{iSide}} , {phi~{j}~{idom}~{iSide}} ];
        In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
    EndFor
  EndFor
EndIf
```

Listing 7: GIBC(N_p, α, ε) transmission condition (22). (Code extracted from the file `Helmholtz.pro`.)

```
If (TC_TYPE == 3) // PML
  For iSide In {0:1}
    Galerkin { [D[] * Dof{d u~{idom}}, {d u~{idom}}];
      In Pml~{idom}~{iSide}; Jacobian JVol; Integration I1;}
    Galerkin { [ - kPml~{idom}~{iSide}[]^2 * E[] * Dof{u~{idom}}, {u~{idom}}];
      In Pml~{idom}~{iSide}; Jacobian JVol; Integration I1;}
  EndFor
EndIf
```

Listing 8: Perfectly Matched Layer transmission condition (24). (Code extracted from the file `Helmholtz.pro`.)

```
Galerkin { [ Dof{g_out~{idom}~{iSide}} , {g_out~{idom}~{iSide}} ];
  In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
Galerkin { [ $ArtificialSource~{iSide} ? g_in~{idom}~{iSide}[] : 0 ,
    {g_out~{idom}~{iSide}} ];
  In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }

If(TC_TYPE == 0) // IBC
  Galerkin { [ 2 * I[] * kIBC[] * {u~{idom}} , {g_out~{idom}~{iSide}} ];
    In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
EndIf

If(TC_TYPE == 1) // GIBC(a, b)
  Galerkin { [ - 2 * a[] * {u~{idom}} , {g_out~{idom}~{iSide}} ];
    In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
  Galerkin { [ 2 * b[] * {d u~{idom}} , {d g_out~{idom}~{iSide}} ];
    In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
EndIf

If(TC_TYPE == 2) // GIBC(NP_OSRC, theta_branch, eps)
  Galerkin { [ 2 * ( I[] * k[] * OSRC_C0[]{NP_OSRC,theta_branch} *
        {u~{idom}} ) , {g_out~{idom}~{iSide}} ];
    In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
  For j In {1:NP_OSRC}
    Galerkin { [ 2 * ( I[] * k[] * OSRC_Aj[]{j,NP_OSRC,theta_branch} /
          OSRC_Bj[]{j,NP_OSRC,theta_branch} *
          ({u~{idom}} - {phi~{j}~{idom}~{iSide}})) , {g_out~{idom}~{iSide}} ];
      In Sigma~{idom}~{iSide}; Jacobian JSur; Integration I1; }
  EndFor
EndIf

If (TC_TYPE == 3) // PML
  Galerkin { [ -2 * D[] *  {d u~{idom}}, {d g_out~{idom}~{iSide}}];
    In Pml~{idom}~{iSide}; Jacobian JVol; Integration I1;}
  Galerkin { [ 2 * kPml~{idom}~{iSide}[]^2 *E[] * {u~{idom}},
      {g_out~{idom}~{iSide}}];
    In Pml~{idom}~{iSide}; Jacobian JVol; Integration I1;}
EndIf
```

Listing 9: `Equation` part of the surface formulation (17) for the Helmholtz problem. (Code extracted from the file `Helmholtz.pro`.)

```
Formulation {
  { Name Vol~{idom}; Type FemEquation;
    Quantity {
      { Name e~{idom}; Type Local; NameOfSpace Hcurl_e~{idom}; }
    }
    Equation {
      // volume terms
      Galerkin { [ Dof{d e~{idom}}, {d e~{idom}} ];
        In Omega~{idom}; Integration I1; Jacobian JVol; }
      Galerkin { [ -k[]^2 * Dof{e~{idom}} , {e~{idom}} ];
        In Omega~{idom}; Integration I1; Jacobian JVol; }
      Galerkin { [ -I[] * k[] * N[] /\ (Dof{e~{idom}} /\ N[]) , {e~{idom}} ];
        In GammaInf~{idom}; Integration I1; Jacobian JSur; }

      // artificial sources on transmission boundaries (iSide split only
      // useful for sweeping-type preconditioners)
      For iSide In {0:1}
        Galerkin { [ $ArtificialSource~{iSide} ? g_in~{idom}~{iSide}[] :
          Vector[0,0,0] , {e~{idom}} ];
          In Sigma~{idom}~{iSide}; Integration I1; Jacobian JSur; }
      EndFor

      // transmission condition
      ⟨···⟩
    }
  }
}
```

Listing 10: Volume system of the Maxwell equation (35). (Code extracted from the file `Maxwell.pro`.)

The implementation of the weak formulations for the Maxwell problem is similar. For example, the volume PDE formulation is presented in Listing 10. For the detailed implementation of the transmission conditions, see the `Maxwell.pro` source file.

3.2 Domain Decomposition Solver

The domain decomposition method is naturally suited for parallel computation on distributed memory architectures, using e.g. the Message Passing Interface (MPI). A classical and efficient approach is to use, when possible, as many MPI processes as subdomains. The i^{th} process is then associated for example to Ω_i and is then in charge of solving the volume PDE on Ω_i and computing every outgoing data $(g_{ji})_{j\in D_i}$. (Each MPI process can of course be multi-threaded in order to benefit from shared memory parallelism—this is handled automatically in GetDDM through the use of appropriate BLAS libraries.) On the other hand, solving these problems requires the knowledge of the incoming quantities $(g_{ij})_{j\in D_i}$ and, as the memory is distributed, an exchange of information with each process of rank $j \in D_i$ must be achieved, leading to Algorithm 2. The inter-process communications are handled transparently by GetDDM in the `IterativeLinearSolver` function (see Section 3.2.3) and *via* the explicit `SetCommSelf`/`SetCommWorld` commands. By default, GetDDM uses a broadcast to exchange information between the processes; this can be replaced by a more efficient communication pattern by explicitly specifying the subdomain connections, as explained in Section 3.2.2. To summarize,

[4]Parser variables are evaluated when the input `.pro` file is analyzed. They cannot be changed during the actual computation, contrary to run-time variables (prefixed with a `$`).

solving the domain decomposition problem with GetDDM involves these main steps:

1. Create a mesh for each subdomain.
2. Write the weak formulations.
3. Distribute the subdomains and the unknowns to the MPI processes.
4. Optional: specify the topology to improve communications.
5. Setup the iterative solver.

Points 1 and 2 have been described in the previous section and are unchanged compared to standard (non-parallel) programs, for which many examples can be found online on `http://onelab.info`. Only the new points 3, 4 and 5 are hence detailed in the following paragraphs. Each point will first be described, then illustrated by an example and a code listing, based on a simple waveguide-like structure divided into slices.

Algorithm 2 Domain decomposition algorithm from the point of view of the local i^{th} MPI process (Helmholtz case).

---

1. Solve the local volume problem: $v_i = \mathscr{V}_i(u^{\text{inc}}, 0)$.
2. Compute the local contribution to the right hand side: $b_{ji} = \mathscr{T}_{ji}(0, v_i)$.
3. Exchange information about the (geometric) topology between processes.
4. Enter the parallel iterative Krylov subspace solver for the system $(\mathcal{I} - \mathcal{A})g = b$; at each iteration, compute the local contribution $(\mathcal{A}g^n)_{ji}$ through the following sequence of operations:
   - Send quantity g^n_{ji} to and receive g^n_{ij} from connected processes $j \in D_i$.
   - Solve the local volume problem: $\widetilde{u}^{n+1}_i = \mathscr{V}_i(0, g^n)$.
   - Solve the local surface problems: $(\mathcal{A}g^n)_{ji} = \mathscr{T}_{ji}(g^n_{ij}, \widetilde{u}^{n+1}_i), \quad \forall j \in D_i$.
5. Send quantity g_{ji} to and receive g_{ij} from connected processes $j \in D_i$.
6. Compute the local contribution to the final solution by solving the volume problem: $u_i = \mathscr{V}_i(u^{\text{inc}}, g)$.

3.2.1 Distributing the Subdomains and the Unknowns

In GetDDM the unknown g_{ji} is called a `Field`, which is a function that can be interpolated using a finite element basis on its domain of definition (and which is set to zero elsewhere). Distributing the subdomains to the different MPI processes is straightforward. Special consideration is however needed for the `Fields`, as they must be indexed using a global numbering scheme. This numbering is chosen by the user: it amounts to choosing a unique integer identifier for every pair (j, i). Once the numbering is decided upon, two local (per MPI process) lists must be constructed containing respectively the indices of the subdomain(s) and the (global) indices of the `Fields` that the current process is in charge of. In this paper, these two lists are

named respectively `ListOfSubdomains` and `ListOfFields`. A generic implementation for simple layered-like decompositions is provided in the `Decomposition.pro` file.

Let this point be clarified by a simple example on a one-directional waveguide, divided into N_{dom} subdomains, from "left" (0) to "right" ($N_{\text{dom}} - 1$) as depicted in the table below (each cell being a subdomain):

0	1	2	...	$N_{\text{dom}} - 1$

To simplify, the number of MPI processes is set to N_{dom} and the domain Ω_i is handled by the MPI process of rank i, which is done in practice by an `If` statement on the MPI rank. Now, about the unknown g_{ji}: a subdomain has two connected subdomains except the first and the last one (Ω_0 and $\Omega_{N_{\text{dom}}-1}$), which have only one adjacent subdomain. Every process will hence be in charge of either 2 `Fields` (the "interior" subdomains) or 1 `Field` (the first and last one). A simple choice for the global numbering of these `Fields` is to go from 0 to $2N_{\text{dom}} - 3$ with a step of 1, as follows:

Subdomain number	0	1		2		...	$N_{\text{dom}} - 1$
Indices (j, i) (of g_{ji})	$(1, 0)$	$(0, 1)$	$(2, 1)$	$(1, 2)$	$(2, 3)$	...	$(N_{\text{dom}} - 2, N_{\text{dom}} - 1)$
Global index of (j, i)	0	1	2	3	4	...	$2N_{\text{dom}} - 3$

The contents of the local lists `ListOfSubdomains` and `ListOfFields` are then (the brackets $[\cdot]$ recall the list nature of the quantities):

MPI Process	`ListOfSubdomains`	`ListOfFields`
0	$[0]$	$[0]$
1	$[1]$	$[1, 2]$
2	$[2]$	$[3, 4]$
$\vdots$	$\vdots$	$\vdots$
$N_{\text{dom}} - 1$	$[N_{\text{dom}} - 1]$	$[2N_{\text{dom}} - 3]$

A generalized input `.pro` file for this example is presented in Listing 11 (taken from the file `Decomposition.pro`). Let us point out the following remarks about the above implementation:

- The operation `List += a` appends `a` at the end of the list `List`; `#List` returns the size of the list `List`.
- The sizes of `ListOfSubdomains` and `ListOfFields` can differ from one MPI process to another.
- Listing 11 is not restricted to N_{dom} MPI processes. Indeed and for example, for 4 subdomains and 3 MPI processes, the processes 1 and 2 will be in charge of subdomains 1 and 2, but the process of rank 0 will be in charge of subdomains 0 and 3. In that case and for process 0, `ListOfSubdomains` $= [0, 3]$ and `ListOfFields` $= [0, 5]$. Sequential execution (1 MPI process) is therefore also automatically handled.
- Generally and as in the example above, a `Field` represents one and only one unknown g_{ji}. However, there is no restriction and a `Field` can actually represent two or more unknowns. This can lead to a simpler algorithmic implementation, but can degrade parallel performance as it increases the amount of communication (e.g. i^{th} process would send the field g_{ki} to process j instead of only g_{ji}, for $j, k \in D_i$, $j \neq k$).

```
ListOfSubdomains = {}; // the subdomains that I'm in charge of
ListOfFields = {}; // my fields
For idom In {0:N_DOM-1}
  If (idom % MPI_Size == MPI_Rank)
    If(idom == 0)
      myFieldLeft = {};
      myFieldRight = {0};
    EndIf
    If(idom == N_DOM-1)
      myFieldLeft = {2*idom-1};
      myFieldRight = {};
    EndIf
    If(idom > 0 && idom < N_DOM-1)
      myFieldLeft = {2*idom-1};
      myFieldRight = {2*idom};
    EndIf
    ListOfSubdomains += idom;
    ListOfFields += {myFieldLeft(), myFieldRight()};
  EndIf
EndFor
```

Listing 11: Distribution of the subdomains and the `Fields` to the different MPI processes; the number of MPI processes can be different from the number of subdomains. (Code extracted from the file `Decomposition.pro`.)

3.2.2 Enhancing the Communication Process

By default the communication of the `Fields` between the MPI processes is global and realized through an `MPI_Broadcast`. This is clearly inefficient for large scale problems and can be remedied by explicitly specifying the connections between the `Fields`, in which case efficient asynchronous `MPI_Isend` and `MPI_Irecv` communication is used. Two `Fields` are said to be "connected" when one is needed to compute the other. For example, the `Fields` storing g_{ij} and g_{ji} are connected (through equation (5)). For the simple one-directional waveguide, the following table provides the connectivity of the `Fields`:

Subdomain number	0	1		2		...	$N_{\text{dom}} - 1$
Index of `Field` storing g_{ji}	0	1	2	3	4	...	$2N_{\text{dom}} - 3$
Index of connected `Field` storing g_{ij}	1	0	3	2	5	...	$2N_{\text{dom}} - 4$

In practice, the indices of these connected `Fields` are contained in a third list, named here `ListOfConnectedFields`. As suggested above, a `Field` can however store more than one unknown g_{ji}. In the waveguide example, the two unknowns $g_{0,1}$ and $g_{2,1}$ could be stored in a single `Field` numbered 1, without changing the other `Fields`' indices. In that case, `Field` 1 would be connected to the two `Fields` 0 and 3. `ListOfConnectedFields` is actually a concatenation of sublists, each one being preceded by its size.

This point is clarified by the following example, which completes the one from Section 3.2.1. The two lists `ListOfSubdomains` and `ListOfFields` remain unchanged and a new list, named `ListOfConnectedFields`, is added, composed of the number of connected `Fields` (written in

bold) followed by their indices. Every `Field` has hence only one connected `Field`.

MPI Process	`ListOfSubdomains`	`ListOfFields`	Connected	`ListOfConnectedFields`
0	$[0]$	$[0]$	$[[1]]$	$[\mathbf{1}, 1]$
1	$[1]$	$[1, 2]$	$[[0], [3]]$	$[\mathbf{1}, 0, \mathbf{1}, 3]$
2	$[2]$	$[3, 4]$	$[[2], [5]]$	$[\mathbf{1}, 2, \mathbf{1}, 5]$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$N_{\text{dom}} - 1$	$[N_{\text{dom}} - 1]$	$[2N_{\text{dom}} - 3]$	$[[2N_{\text{dom}} - 4]]$	$[\mathbf{1}, 2N_{\text{dom}} - 4]$

A general implementation for layered decomposition is provided in Listing 12, taken from the file `Decomposition.pro`. This code is again written for an arbitrary number of MPI processes. Note that, to be used in the weak formulations, a `Field` must be called through the command `ComplexScalarField` or `ComplexVectorField`, depending on its nature. In Listing 12 and to simplify, the connected `Fields` are stored in the quantities `g_in` (see also Listing 4).

3.2.3 Handling the Iterative Solver

The iterative algorithm uses the built-in GetDDM function `IterativeLinearSolver`, which takes as argument the operations that implement the matrix-vector product. (Internally, `IterativeLinearSolver` is based on PETSc, which allows to transparently interface a large collection of parallel iterative solvers. In Listing 13, the solver is specified by the string variable `SOLVER` (line 29), which can select any of the PETSc `KSP_TYPE` Krylov method (e.g. `"gmres"` for GMRES or `"bcgs"` for bi-CGStab[5]), followed by standard parameters such as the tolerance and the maximum number of iterations.

Listing 13 presents the `Resolution` object from the generic `Schwarz.pro` example file, using the lists described above. The implementation uses a series of `Macros` (called with `Call`), defined in the `SchwarzMacros.pro` file. The macros `EnablePhysicalSources` and `DisablePhysicalSources` modify the run-time variable `$PhysicalSource` introduced in Section 3.1 (and update the values of the physical constraints accordingly); the macros `EnableArtificialSources` and `DisableArtificialSources` modify the run-time variable `$ArtificialSource`. The three other macros (`SolveVolumePDE` and `SolveSurfacePDE` and `UpdateSurfaceFields`) solve the volume and the surface problems and update the `Fields`, respectively. They are presented in Listings 14, 15 and 16.

3.3 Other Features

As seen above, GetDDM provides quite a general environment for the solution of Schwarz-type domain decomposition problems: the flexibility in the specification of discrete function spaces and weak formulations allows to handle a variety of physical problems and geometrical configurations; and the generic linear algebra (through `IterativeLinearSolver`) and communication mechanism (using `Fields`) permits to implement different variants of Schwarz methods. In addition to the basic functionality described above, the following features are also available:

[5] See `http://www.mcs.anl.gov/petsc/petsc-current/docs/manualpages/KSP/KSPType.html` for a complete list.

```
ListOfConnectedFields = {}; // fields connected to my fields
For idom In {0:N_DOM-1}
  If (idom % MPI_Size == MPI_Rank)
    If(idom == 0)
      // fields to exchange with
      connectedFieldLeft = {};
      connectedFieldRight = {1};
      // as many "blocks" as connected fields
      ListOfConnectedFields += #connectedFieldRight();
      ListOfConnectedFields += connectedFieldRight();
    EndIf
    If(idom == N_DOM-1)
      connectedFieldLeft = {2*(idom-1)};
      connectedFieldRight = {};
      ListOfConnectedFields += #connectedFieldLeft();
      ListOfConnectedFields += connectedFieldLeft();
    EndIf
    If(idom > 0 && idom < N_DOM-1)
      connectedFieldLeft = {2*(idom-1)};
      connectedFieldRight = {2*idom+1};
      // 2 "blocks"
      ListOfConnectedFields += #connectedFieldLeft();
      ListOfConnectedFields += connectedFieldLeft();
      ListOfConnectedFields += #connectedFieldRight();
      ListOfConnectedFields += connectedFieldRight();
    EndIf
    // definition of artificial source fields
    If(ANALYSIS == 0) // Helmholtz (scalar-valued)
      g_in~{idom}~{0}[Sigma~{idom}~{0}] = ComplexScalarField[XYZ[]]{
          connectedFieldLeft()};
      g_in~{idom}~{1}[Sigma~{idom}~{1}] = ComplexScalarField[XYZ[]]{
          connectedFieldRight()};
    EndIf
    If(ANALYSIS == 1) // Maxwell (vector-valued)
      g_in~{idom}~{0}[Sigma~{idom}~{0}] = ComplexVectorField[XYZ[]]{
          connectedFieldLeft()};
      g_in~{idom}~{1}[Sigma~{idom}~{1}] = ComplexVectorField[XYZ[]]{
          connectedFieldRight()};
    EndIf
  EndIf
EndFor
```

Listing 12: Creation of the list `ListOfConnectedFields` for a layered decomposition. (Code extracted from the file `Decomposition.pro`.)

```
Resolution {
  { Name DDM;
    System {
      For i In {0:#ListOfSubdomains()-1}
        idom = ListOfSubdomains(i);
        { Name Vol~{idom}; NameOfFormulation Vol~{idom};
          Type Complex; NameOfMesh Sprintf[StrCat[FILE, "%g.msh"], idom]; }
        For iSide In {0:1}
          { Name Sur~{idom}~{iSide}; NameOfFormulation Sur~{idom}~{iSide};
            Type Complex; NameOfMesh Sprintf[StrCat[FILE, "%g.msh"], idom]; }
        EndFor
      EndFor
    }
    Operation {
      // compute local part of distributed rhs b for Krylov solver using
      // physical sources only, and update surface data
      Call EnablePhysicalSources;
      Call DisableArtificialSources;
      Call SolveVolumePDE;
      Call SolveSurfacePDE;
      Call UpdateSurfaceFields;

      // launch distributed Krylov solver using artificial sources only.
      // IterativeLinearSolver solves (I-A) g = b: ListOfFields() initially
      // stores the local part of b; then stores each local part of iterate
      // g^n.
      Call DisablePhysicalSources;
      Call EnableArtificialSources;
      IterativeLinearSolver["I-A", SOLVER, TOL, MAXIT, RESTART,
                            {ListOfFields()}, {ListOfConnectedFields()}, {}]
      {
        // compute local part of (A g^n) and stores the result in
        // ListOfFields()
        Call SolveVolumePDE;
        Call SolveSurfacePDE;
        Call UpdateSurfaceFields;
      }

      // build final volume solution after convergence on own cpu, using both
      // physical and artificial sources
      Call EnablePhysicalSources;
      Call EnableArtificialSources;
      Call SolveVolumePDE;
      Call SaveVolumeSolutions;
    }
  }
}
```

Listing 13: Resolution of the domain decomposition method using GetDDM. Macros are used to simplify the code. (Code extracted from the file `Schwarz.pro`.)

```
Macro SolveVolumePDE
  // work on own cpu
  SetCommSelf;
  For ii In {0:#ListOfSubdomains()-1}
    idom = ListOfSubdomains(ii);
    // solve the volume PDE on each subdomain
    If(GenerateVolFlag~{idom})
      // the matrix is already factorized, only regenerate the RHS
      GenerateRHS[Vol~{idom}]; SolveAgain[Vol~{idom}];
    EndIf
    If(GenerateVolFlag~{idom} == 0)
      // first time generation and factorization of the matrix
      Generate[Vol~{idom}]; Solve[Vol~{idom}];
      GenerateVolFlag~{idom} = 1;
    EndIf
  EndFor
  // go back to parallel mode
  SetCommWorld;
Return
```

Listing 14: Macro used to solve the volume problem. The two commands `SetCommSelf` and `SetCommWorld` let GetDDM respectively go into sequential mode and come back to parallel mode. (Code extracted from the file `SchwarzMacros.pro`.)

```
Macro SolveSurfacePDE
  SetCommSelf;
  // compute g_in for next iteration
  For ii In {0:#ListOfSubdomains()-1}
    idom = ListOfSubdomains(ii);
    // solve the surface PDE on the boundaries of each subdomain
    For iSide In {0:1}
      If(NbrRegions[Sigma~{idom}~{iSide}])
        If(GenerateSurFlag~{idom}~{iSide})
          // the matrix is already factorized, only regenerate the RHS
          GenerateRHS[Sur~{idom}~{iSide}]; SolveAgain[Sur~{idom}~{iSide}];
        EndIf
        If(GenerateSurFlag~{idom}~{iSide} == 0)
          // first time generation and factorization of the matrix
          Generate[Sur~{idom}~{iSide}]; Solve[Sur~{idom}~{iSide}];
          GenerateSurFlag~{idom}~{iSide} = 1;
        EndIf
      EndIf
    EndFor
  EndFor
  SetCommWorld;
Return
```

Listing 15: Macro `SolveSurfacePDE` that solves the surface problem. (Code extracted from the file `SchwarzMacros.pro`.)

```
Macro UpdateSurfaceFields
  SetCommSelf;
  // store g in ListOfFields()
  For ii In {0:#ListOfSubdomains()-1}
    idom = ListOfSubdomains(ii);
    For iSide In {0:1}
      PostOperation[g_out~{idom}~{iSide}];
    EndFor
  EndFor
  SetCommWorld;
Return
```

Listing 16: Macro to update the surface fields. (Code extracted from the file `SchwarzMacros.pro`.)

Preconditioning. The `IterativeLinear` solver can handle both left and right preconditioners: they can be specified in between a second pair of braces after the operations implementing the main matrix-vector product—see Listing 13. Preconditioners are a topic of high interest for domain decomposition methods as it is well-known that a DDM without preconditioner (*one-level* DDM) is not scalable with respect to the number of subdomains. Such preconditioners are often referred to as *coarse grids*, a vocabulary coming historically from mechanics where, indeed, a coarse grid is built with e.g. one point per subdomain, and a global PDE is solved on it in order to transfer low-frequency information over the whole domain. While efficient preconditioners are well-known for Laplace-type PDEs, these classical coarse grids fail for wave-type problems. Designing preconditioners for high-frequency time-harmonic wave problems is an active research field, with recent promising advances in sweeping-type approaches [27, 56, 59]. GetDDM can be used to test these preconditioners, and several variants of sweeping preconditioners are implemented in the `Schwarz.pro` file, with various degrees of parallelization [60][6].

Overlapping decompositions. Using overlaps between the subdomains can significantly improve the convergence of DDMs [32, 29], albeit at the additional cost of larger volume subproblems. Such overlapping decompositions are directly handled with GetDDM at the level of the geometrical description and the mesh; the only modification in the formulations is at the level of the surface field updates like (5), where the Neumann data must be evaluated explicitly since it can no longer be eliminated, the interfaces Σ_{ij} and Σ_{ji} being distinct from each other. Listing 17 shows the part of the script that is modified to treat overlapping decompositions.

Non-conforming meshes. GetDDM makes it easy to handle non-conforming meshes (*Mortar methods* [7, 10] or *NICEM method* [31, 42]), as the data between subdomains is exchanged in the form of `Fields`, which are interpolable. Non-conforming meshes can be useful in a variety of settings, e.g. for multi-physics couplings or adaptive simulations. Another practical interest is linked to parallel mesh generation: relaxing the conformity on interfaces allows for coarse-grained, embarrassingly parallel mesh generation, where each subdomain is handled by its separate CAD representation. The `waveguide3d.geo` file provides an example of such an approach.

Other formulations and physical problems. In addition to the standard formulations pre-

[6] GetDDM currently only handles matrix-free preconditioners: implementing explicit coarse space corrections requires a modification of the C++ source code.

```
1  Galerkin { [ Dof{g_out~{idom}~{iSide}} , {g_out~{idom}~{iSide}} ];
2    In Sigma_e~{idom}~{iSide}; Jacobian JSur; Integration I1; }
3  Galerkin { [ - {d u~{idom}} , {d g_out~{idom}~{iSide}} ];
4    In Omega_o~{idom}~{iSide}; Jacobian JVol; Integration I1; }
5  Galerkin { [ k[]^2 * {u~{idom}} , { g_out~{idom}~{iSide}} ];
6    In Omega_e~{idom}~{iSide}; Jacobian JVol; Integration I1; }
7
8  If(TC_TYPE == 0) // IBC
9    Galerkin { [ I[] * kIBC[] * {u~{idom}} , {g_out~{idom}~{iSide}} ];
10     In Sigma_e~{idom}~{iSide}; Jacobian JSur; Integration I1; }
11 EndIf
```

Listing 17: Modification of lines 1–10 of Listing 9 to treat a decomposition with overlap: `Sigma_e` represents the part of the boundary of Ω_j embedded in Ω_i; `Omega_o` represents the overlap $\Omega_i \cap \Omega_j$.

sented in this paper for the Helmholz and the Maxwell problems, various other formulations can be readily implemented in the `.pro` files: mixed formulations (e.g. pressure-velocity formulations for Helmholtz), dual formulations (e.g. in terms of the magnetic field for Maxwell), etc. Time-dependent and non-linear problems can also be investigated using standard features of the underlying software, as can be applications to other physical problems (e.g. elastodynamics or Schrödinger). Finally, several coupled, multi-physics problems (e.g. electro-mechanical) have been tested as well.

4 Complete Examples

Several complete examples for both Helmholtz and Maxwell problems are available online. Here are the simple steps to run these examples interactively in serial mode (see Figure 1):

1. Download and uncompress the GetDDM code bundle:
   - Windows: `http://onelab.info/files/gmsh-getdp-Windows64.zip`
   - MacOS: `http://onelab.info/files/gmsh-getdp-MacOSX.zip`
   - Linux: `http://onelab.info/files/gmsh-getdp-Linux64.zip`
   - Source code: `http://onelab.info/files/gmsh-getdp-source.zip`
2. Launch Gmsh (e.g. double-click on the `gmsh.exe` executable on Windows).
3. Open the `models/ddm_waves/main.pro` file with the `File>Open` menu.
4. Click on `Run`.

Different test-cases can be chosen by selecting the appropriate `Problem` in the menu to the left of the graphics window. Top-level parameters (type of transmission condition, number of subdomains, frequency, etc.) can be changed interactively in the menu as well.

This interactive use of GetDDM is useful for testing, demonstration and visualization purposes. For actual, parallel computations, you will need to recompile the GetDDM source code for your particular computer architecture and MPI implementation. Detailed installation instructions are available on the website of the GetDDM project, at the address: `http://onelab.info/wiki/GetDDM`. Once compiled, GetDDM is then called from the command line, using (depending on your MPI setup), commands similar to the following (here for the `waveguide3d` test-case, on 100 CPUs):

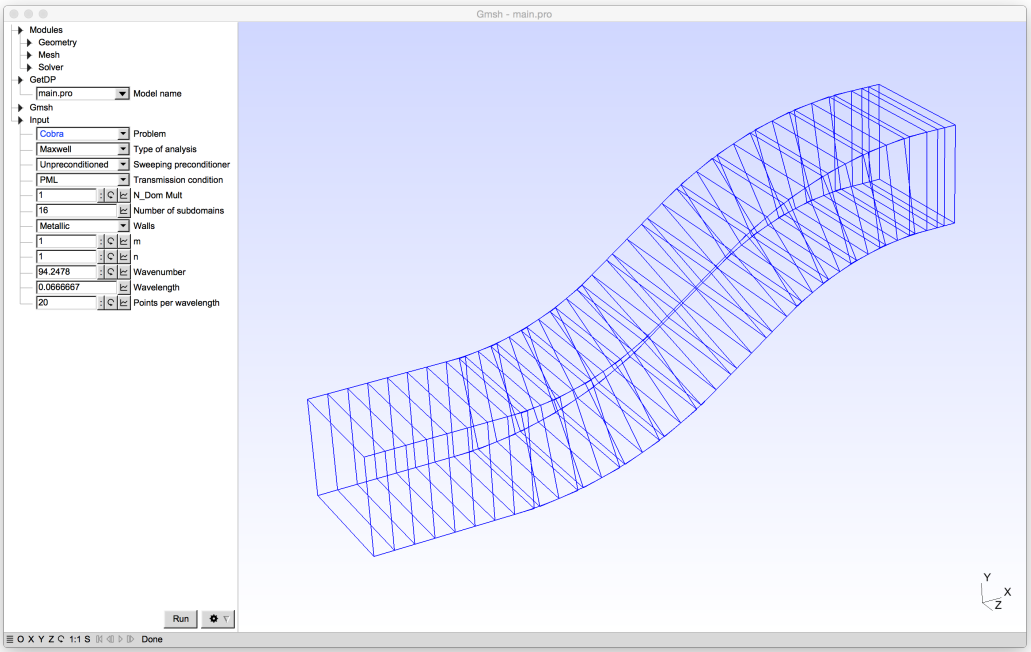


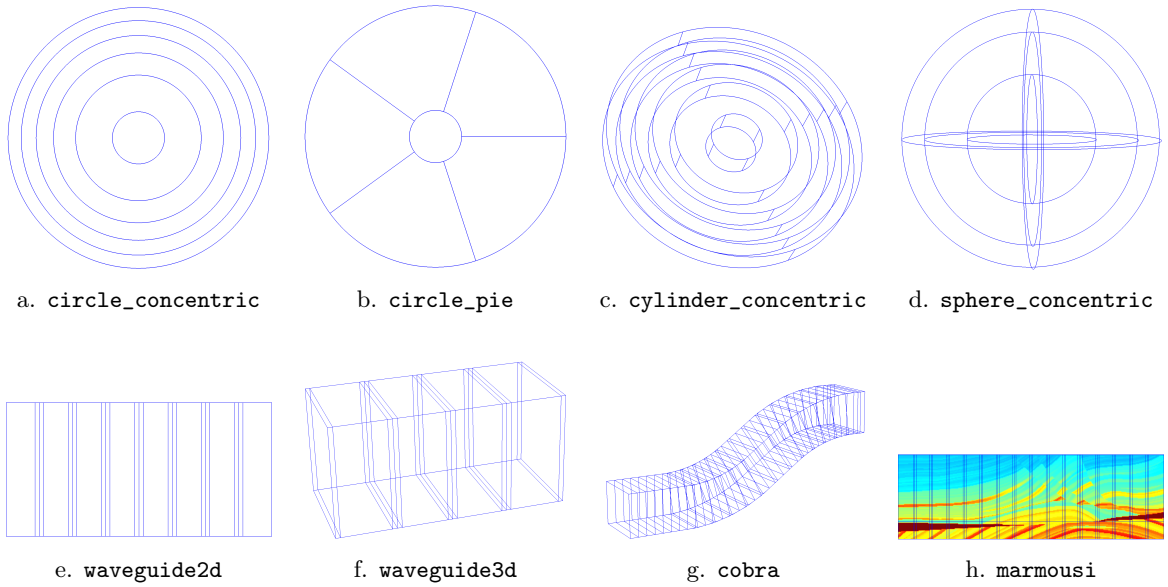
Figure 1: Graphical user interface of GetDDM. (Displayed test-case is `cobra`, with PML transmission conditions.)

```
mpirun -np 100 gmsh -setnumber N_DOM 100 waveguide3d.geo -
mpirun -np 100 getdp -setnumber N_DOM 100 waveguide3d.pro -solve DDM
```

Sample scripts for running large scale computations on computer clusters using the SLURM or PBS job schedulers are also provided in the `models/ddm_waves` directory.

In all cases (interactive, serial or parallel), the input files and the workflow are the same: a geometry (*`file`*`.geo`) is constructed, meshed and partitioned into subdomains using Gmsh; then the problem definition (*`file`*`.pro`) is analyzed and processed by GetDP. The geometry and problem definition files usually include a data file (*`file_data`*`.geo`) containing parameters common to Gmsh and GetDP. Figure 2 lists some of the complete examples available online, together with references to published work where additional information about the test-cases can be found. Some of these examples can be solved in both the Helmholtz and Maxwell setting (e.g. `waveguide3d` or `cylinder_concentric`); others are only designed for the Helmholtz case (e.g. `marmousi`). With the default set of parameters, all these test-cases will run on standard laptop or desktop computers. But they can all also be scaled up to test the algorithms on massively parallel computers. For example, a Helmholtz problem on the `marmousi` example was run at frequency $\frac{\omega}{2\pi} = 700$ (about 4000 wavelengths in the domain, uniformly discretized by a mesh density of 20 points per the smallest wavelength), with 358 subdomains for a total of 4296 CPUs (cores). The size of the full problem exceeded 2.3 billions unknowns. A Maxwell simulation of the `waveguide3d` model was performed with 3,500 subdomains on 3,500 CPUs (cores), with a total number of unknowns exceeding 300 million.

All the specific problem files include the same generic implementation of the Schwarz domain decomposition algorithm, contained in the following 5 files:



a. `circle_concentric` b. `circle_pie` c. `cylinder_concentric` d. `sphere_concentric`

e. `waveguide2d` f. `waveguide3d` g. `cobra` h. `marmousi`

Figure 2: Sample models available online at `http://onelab.info/wiki/GetDDM`. a., b., c., d.: acoustic or electromagnetic (c. and d. only) scattering by cylindrical or spherical obstacles, with concentric or radial subdomains [13, 25]. e., f.: guided acoustic or electromagnetic waves in rectangular waveguides [59]. g.: guided acoustic or electromagnetic waves in the COBRA benchmark defined by the JINA98 workgroup [60]. h.: acoustic waves in the underground Marmousi model [56].

```
Decomposition.pro
Schwarz.pro
SchwarzMacros.pro
Helmholtz.pro
Maxwell.pro
```

As explained in Sections 3.2.1 and 3.2.2, `Decomposition.pro` defines the subdomains and the communication layout. The files `Schwarz.pro` and `SchwarzMacros.pro` that implement the iterative linear solver have been presented in Section 3.2.3. And finally `Helmholtz.pro` and `Maxwell.pro` contain the physics-specific function spaces and weak formulations, detailed in Section 3.1.

For illustration purposes, Figure 3 presents some other cases that have been solved using GetDDM. Published references are provided, which contain further information about the specific test cases, mathematical models and numerical results.

5 Conclusion

This article introduced a new open source software, called GetDDM, for testing optimized Schwarz domain decomposition methods for time-harmonic wave problems. The software aims to provide a simple, flexible and ready-to-use environment where the weak formulations of the wave propagation problems and associated transmission conditions for the Schwarz methods are transcribed naturally, with a direct link to their symbolic mathematical representation. The code and a variety of examples for both Helmholtz and Maxwell problems are freely available

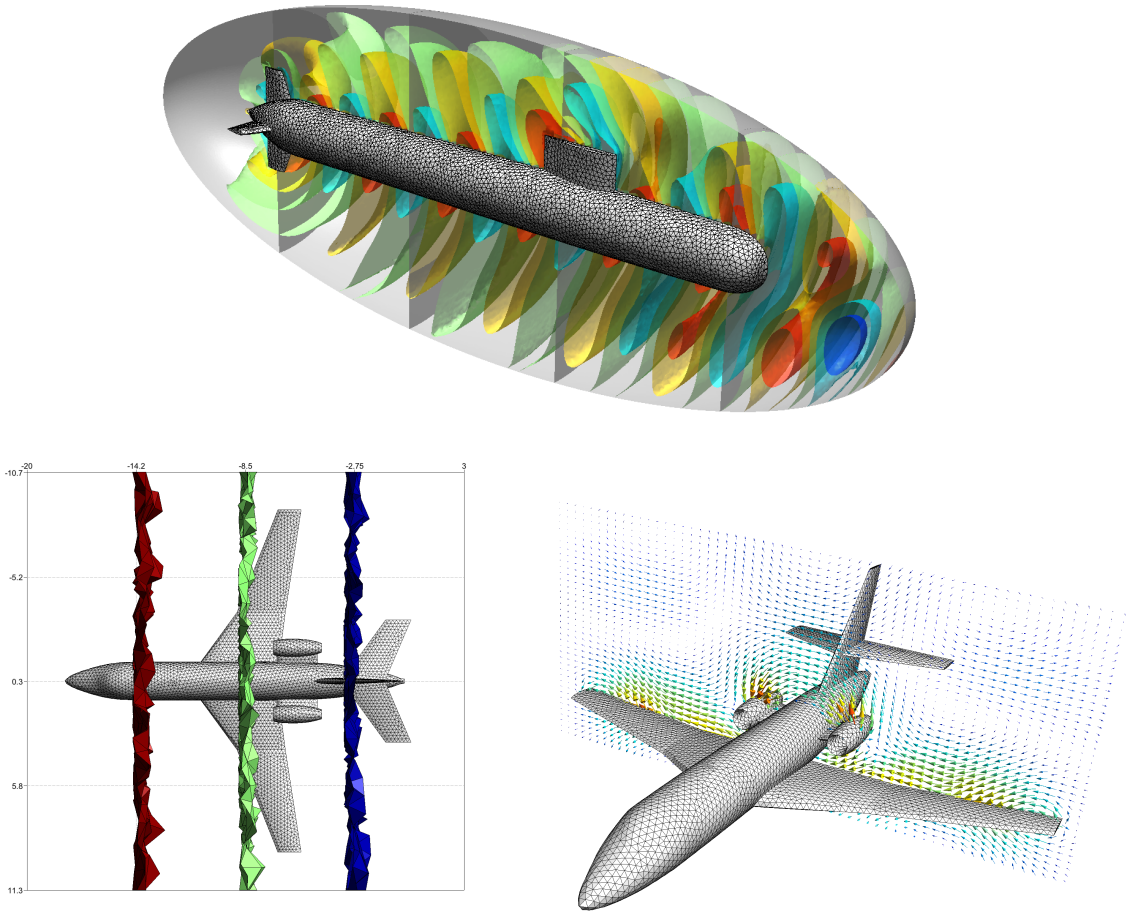


Figure 3: Sample models solved with GetDDM. Top: acoustic waves around a submarine (image reproduced from [13]). Bottom: electromagnetic waves around a Falcon aircraft (images reproduced from [25]).

online for further testing, at the address: `http://onelab.info/wiki/GetDDM`. Depending on the chosen parameters (frequency, mesh refinement, number of subdomains, etc.) the same files can run with a few thousands of unknowns on laptop computers or even tablets; or with billions of unknowns on massively parallel computer clusters.

In addition to the application of GetDDM to other physical problems like elastodynamics or the Schrödinger equation, ongoing work includes the development of a better interface to automatic mesh partitioning tools, as well as the investigation of techniques to handle cross-points and cross-edges in a scalable manner. These new developments will undoubtedly result in new interesting applications, and further online examples on the GetDDM web site.

Acknowledgements

This work was supported in part by the Belgian Science Policy (PAI grant P7/02), the Walloon Region (WIST3 grants ONELAB and ALIZEES), the French ANR (grant MicroWave NT09 460489 "Programme Blanc") and the "EADS Foundation" (High-BRID project, grant 089-1009-1006).

Computational resources have been provided by CÉCI, funded by F.R.S.-FNRS (Fonds de la Recherche Scientifique) under grant n°2.5020.11, and the Tier-1 supercomputer of the Fédération Wallonie-Bruxelles, funded by the Walloon Region under grant n°1117545.

References

[1] A. Alonso-Rodriguez and L. Gerardo-Giorda. New nonoverlapping domain decomposition methods for the harmonic Maxwell system. *SIAM J. Sci. Comput.*, 28(1):102–122, 2006.

[2] X. Antoine, C. Geuzaine, and K. Ramdani. *Wave Propagation in Periodic Media - Analysis, Numerical Techniques and Practical Applications*, volume 1, chapter Computational Methods for Multiple Scattering at High Frequency with Applications to Periodic Structures Calculations, pages 73–107. Progress in Computational Physics, 2010.

[3] S. Balay, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.4, Argonne National Laboratory, 2013.

[4] S. Balay, M. F. Adams, J. Brown, P. Brune, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, K. Rupp, B. F. Smith, and H. Zhang. PETSc Web page. `http://www.mcs.anl.gov/petsc`, 2015.

[5] S. Balay, W. D. Gropp, L. Curfman McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[6] A. Bayliss, M. Gunzburger, and E. Turkel. Boundary conditions for the numerical solution of elliptic equations in exterior regions. *SIAM J. Appl. Math.*, 42(2):430–451, 1982.

[7] F. Ben Belgacem, A. Buffa, and Y. Maday. The mortar finite element method for 3D Maxwell equations: First results. *SIAM Journal on Numerical Analysis*, 39(3):880–901, 2001.

[8] J.-P. Berenger. A perfectly matched layer for the absorption of electromagnetic waves. *J. Comput. Phys.*, 114(2):185–200, 1994.

[9] A. Bermúdez, L. Hervella-Nieto, A. Prieto, and R. Rodríguez. An optimal perfectly matched layer with unbounded absorbing function for time-harmonic acoustic scattering problems. *J. Comput. Phys.*, 223(2):469–488, 2007.

[10] C. Bernardi, Y. Maday, and A.T. Patera. A new nonconforming approach to domain decomposition: the mortar element method. In Pitman, editor, *Collège de France Seminar XI*, pages 13–51. H. Brezis and J. Lions, 1994.

[11] A. Bossavit. *Computational Electromagnetism. Variational Formulations, Edge Elements, Complementarity.* Academic Press, 1998.

[12] Y. Boubendir. An analysis of the BEM-FEM non-overlapping domain decomposition method for a scattering problem. *J. Comput. Appl. Math.*, 204(2):282–291, 2007.

[13] Y. Boubendir, X. Antoine, and C. Geuzaine. A quasi-optimal non-overlapping domain decomposition algorithm for the Helmholtz equation. *Journal of Computational Physics*, 231(2):262 – 280, 2012.

[14] Y. Boubendir, A. Bendali, and M. B. Fares. Coupling of a non-overlapping domain decomposition method for a nodal finite element method with a boundary element method. *Internat. J. Numer. Methods Engrg.*, 73(11):1624–1650, 2008.

[15] F. Collino and P. Monk. The perfectly matched layer in curvilinear coordinates. *SIAM J. Sci. Comput.*, 19(6):2061–2090 (electronic), 1998.

[16] B. Després. *Méthodes de décomposition de domaine pour les problèmes de propagation d'ondes en régime harmonique. Le théorème de Borg pour l'équation de Hill vectorielle.* PhD thesis, Rocquencourt, 1991. Thèse, Université de Paris IX (Dauphine), Paris, 1991.

[17] B. Després, P. Joly, and J. E. Roberts. A domain decomposition method for the harmonic Maxwell equations. In *Iterative methods in linear algebra (Brussels, 1991)*, pages 475–484, Amsterdam, 1992. North-Holland.

[18] V. Dolean, J. M. Gander, S. Lanteri, J.-F. Lee, and Z. Peng. Optimized Schwarz methods for curl-curl time-harmonic Maxwell's equations. 2013.

[19] V. Dolean, M. J. Gander, and L. Gerardo-Giorda. Optimized Schwarz methods for Maxwell's equations. *SIAM J. Sci. Comput.*, 31(3):2193–2213, 2009.

[20] P. Dular and C. Geuzaine. *GetDP Reference Manual: The documentation for GetDP 2.5, A General environment for the treatment of Discrete Problems.* University of Liège.

[21] P. Dular and C. Geuzaine. GetDP Web page, http://getdp.info, 2015. [online]. available: http://getdp.info.

[22] P. Dular, C. Geuzaine, F. Henrotte, and W. Legros. A general environment for the treatment of discrete problems and its application to the finite element method. *IEEE Transactions on Magnetics*, 34(5):3395–3398, September 1998.

[23] M. El Bouajaji, X Antoine, and C. Geuzaine. Approximate local magnetic-to-electric surface operators for time-harmonic Maxwell's equations. *Journal of Computational Physics*, 279(15):241–260, 2014.

[24] M. El Bouajaji, V. Dolean, M. Gander, and S. Lanteri. Optimized Schwarz methods for the time-harmonic Maxwell equations with damping. *SIAM Journal on Scientific Computing*, 34(4):A2048–A2071, 2012.

[25] M. El Bouajaji, B. Thierry, X. Antoine, and C. Geuzaine. A quasi-optimal domain decomposition algorithm for the time-harmonic Maxwell's equations. *Journal of Computational Physics*, 294(1):38–57, 2015.

[26] B. Engquist and A. Majda. Absorbing boundary conditions for the numerical simulation of waves. *Math. Comp.*, 31(139):629–651, 1977.

[27] B. Engquist and L. Ying. Sweeping preconditioner for the Helmholtz equation: moving perfectly matched layers. *Multiscale Model. Simul.*, 9(2):686–710, 2011.

[28] O.G. Ernst and M.J. Gander. Why it is difficult to solve Helmholtz problems with classical iterative methods. In Ivan G. Graham, Thomas Y. Hou, Omar Lakkis, and Robert Scheichl, editors, *Numerical Analysis of Multiscale Problems*, volume 83 of *Lecture Notes in Computational Science and Engineering*, pages 325–363. Springer Berlin Heidelberg, 2012.

[29] M. Gander. Optimized schwarz methods. *SIAM Journal on Numerical Analysis*, 44(2):699–731, 2006.

[30] M. Gander and L. Halpern. Méthode de décomposition de domaine. Encyclopédie électronique pour les ingénieurs, 2012.

[31] M. Gander, C. Japhet, Y. Maday, and F. Nataf. A new cement to glue nonconforming grids with robin interface conditions: The finite element case. In *Domain Decomposition Methods in Science and Engineering*, volume 40 of *Lecture Notes in Computational Science and Engineering*, pages 259–266. Springer Berlin Heidelberg, 2005.

[32] M. J. Gander, L. Halpern, and F. Nataf. Optimized Schwarz methods. In *Proceedings of the 12th International Conference on Domain Decomposition, ddm.org*, 2000.

[33] M. J. Gander, F. Magoulès, and F. Nataf. Optimized Schwarz methods without overlap for the Helmholtz equation. *SIAM J. Sci. Comput.*, 24(1):38–60 (electronic), 2002.

[34] C. Geuzaine. *High order hybrid finite element schemes for Maxwell's equations taking thin structures and global quantities into account.* PhD thesis, Université de Liège, Belgium, 2001.

[35] C. Geuzaine. GetDP: a general finite-element solver for the de Rham complex. In *PAMM Volume 7 Issue 1. Special Issue: Sixth International Congress on Industrial Applied Mathematics (ICIAM07) and GAMM Annual Meeting, Zürich 2007*, volume 7, pages 1010603–1010604. Wiley, 2008.

[36] C. Geuzaine, F. Henrotte, E. Marchandise, J.-F. Remacle, P. Dular, and R. Vazquez Sabariego. ONELAB: Open Numerical Engineering LABoratory. *Proceedings of the 7th European Conference on Numerical Methods in Electromagnetism (NUMELEC2012)*, 2012.

[37] C. Geuzaine, F. Henrotte, E. Marchandise, J.-F. Remacle, and R. Vazquez Sabariego. ONELAB Web page, http://onelab.info, 2015. [online]. available: http://onelab.info.

[38] C. Geuzaine and J.-F. Remacle. *Gmsh Reference Manual: The documentation for Gmsh 2.9, A finite element mesh generator with built-in pre- and post-processing facilities.*

[39] C. Geuzaine and J.-F. Remacle. Gmsh Web page, http://gmsh.info, 2015. [online]. available: http://gmsh.info.

[40] C. Geuzaine and J.-F. Remacle. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Internat. J. Numer. Methods Engrg.*, 79(11):1309–1331, 2009.

[41] D. Givoli. Computational absorbing boundaries. In Steffen Marburg and Bodo Nolte, editors, *Computational Acoustics of Noise Propagation in Fluids - Finite and Boundary Element Methods*, pages 145–166. Springer Berlin Heidelberg, 2008.

[42] C. Japhet, Y. Maday, and F. Nataf. A new interface cement equilibrated mortar (nicem) method with robin interface conditions: the P_1 finite element case. *Mathematical Models and Methods in Applied Sciences*, 23(12):2253–2292, 2013.

[43] P. Jolivet, V. Dolean, F. Hecht, F. Nataf, C. Prud'Homme, and N. Spillane. High performance domain decomposition methods on massively parallel architectures with freefem++. *Journal of Numerical Mathematics*, 20(3-4):287–302, 2012.

[44] P. Jolivet, F. Hecht, F. Nataf, and C. Prud'homme. Scalable domain decomposition preconditioners for heterogeneous elliptic problems. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, pages 80:1–80:11, New York, NY, USA, 2013. ACM.

[45] P. Jolivet and F. Nataf. Hpddm: High-Performance Unified framework for Domain Decomposition methods, MPI-C++ library. https://github.com/hpddm/hpddm.

[46] A. Modave, E. Delhez, and C. Geuzaine. Optimizing perfectly matched layers in discrete contexts. *International Journal for Numerical Methods in Engineering*, 99(6):410–437, 2014.

[47] A. Moiola and E. A. Spence. Is the Helmholtz equation really sign-indefinite? *SIAM Rev.*, 56(2):274–312, 2014.

[48] F. Nataf. Interface connections in domain decomposition methods. *NATO Science Series II*, 75, 2001.

[49] F. Nataf and F. Nier. Convergence rate of some domain decomposition methods for overlapping and nonoverlapping subdomains. *Numer. Math.*, 75:357–377, 1997.

[50] J.-C. Nédélec. *Acoustic and electromagnetic equations*, volume 144 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2001. Integral representations for harmonic problems.

[51] Z. Peng and J. Lee. A scalable nonoverlapping and nonconformal domain decomposition method for solving time-harmonic Maxwell equations in $\mathbb{R}^3$. *SIAM Journal on Scientific Computing*, 34(3):A1266–A1295, 2012.

[52] Z. Peng, V. Rawat, and J.-F. Lee. One way domain decomposition method with second order transmission conditions for solving electromagnetic wave problems. *Journal of Computational Physics*, 229(4):1181 – 1197, 2010.

[53] V. Rawat and J.-F. Lee. Nonoverlapping domain decomposition with second order transmission condition for the time-harmonic Maxwell's equations. *SIAM J. Scientific Computing*, 32(6):3584–3603, 2010.

[54] Y. Saad and M. H. Schultz. GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 7(3):856–869, 1986.

[55] A. Samake, V. Chabannes, C. Picard, and C. Prud'Homme. Domain decomposition methods in feel++. In *Domain Decomposition Methods in Science and Engineering XXI*, pages 397–405. Springer, 2014.

[56] C. Stolk. A rapidly converging domain decomposition method for the Helmholtz equation. *Journal of Computational Physics*, 241(0):240 – 252, 2013.

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

[58] A. Vion, R. Bélanger-Rioux, L. Demanet, and C. Geuzaine. A DDM double sweep preconditioner for the Helmholtz equation with matrix probing of the DtN map. In *Mathematical and Numerical Aspects of Wave Propagation WAVES 2013*, June 2013.

[59] A. Vion and C. Geuzaine. Double sweep preconditioner for optimized schwarz methods applied to the Helmholtz problem. *Journal of Computational Physics*, 266(0):171 – 190, 2014.

[60] A. Vion and C. Geuzaine. Parallel double sweep preconditioner for the optimized Schwarz algorithm applied to high frequency Helmholtz and Maxwell equations. In *LNCSE, Proc. of DD22*, 2014.