
B. Thierry\textsuperscript{1,2}, A. Vion\textsuperscript{2}, S. Tournier\textsuperscript{2,3}, M. El Bouajaji\textsuperscript{4}, D. Colignon\textsuperscript{2}, N. Marsic\textsuperscript{2}, X. Antoine\textsuperscript{4}, C. Geuzaine\textsuperscript{2}

June 2, 2015

Contents

1 Introduction ............................................. 2

2 Optimized Schwarz Methods for Time-Harmonic Wave Propagation ............... 3

  2.1 Acoustic Waves: Helmholtz Equation ........................................ 3

      2.1.1 Mono-Domain Problem .................................................. 3

      2.1.2 Domain Decomposition and Transmission Operators ..................... 4

      2.1.3 Weak Formulations ...................................................... 7

  2.2 Electromagnetic Waves: Maxwell’s Equations ................................... 9

      2.2.1 Mono-Domain Problem .................................................. 9

      2.2.2 Domain Decomposition and Transmission operators ..................... 9

      2.2.3 Weak Formulations ...................................................... 11

3 Implementation in GetDDM ............................................. 12

  3.1 Weak Formulations and Finite Element Discretization ....................... 14

  3.2 Domain Decomposition Solver ............................................ 19

      3.2.1 Distributing the Subdomains and the Unknowns ....................... 20

      3.2.2 Enhancing the Communication Process ................................ 22

      3.2.3 Handling the Iterative Solver ...................................... 23

  3.3 Other Features ...................................................... 23

4 Complete Examples ............................................. 28

5 Conclusion ..................................................... 31

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 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.

\textsuperscript{1}Laboratoire J. L. Lions (LJLL), University of Paris VI, Paris, France (thierry@ljll.math.upmc.fr).
\textsuperscript{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).
\textsuperscript{3}Pontificia Universidad Catolica de Chile, Santiago, Chile (simon.tournier@gmail.com).
\textsuperscript{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).
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 such problems, 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 [45]. 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 [55].

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, 17, 18, 19, 23, 24, 49, 50, 51] 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 [48] related to the complementary of the subdomain of interest [47, 46]. 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 [17, 18, 19, 23, 24, 49, 50, 51]. Recently, PMLs have also been used for this same purpose [54, 27, 57, 58].

The aim of this 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 [21, 22, 35] and Gmsh [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 [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

\[ \text{http://getdp.info} \]
\[ \text{http://gmsh.info} \]
\[ \text{http://onelab.info/wiki/GetDDM} \]
methods, either linked to finite element kernels (e.g. FreeFem++ [43] or Feel++ [53] via the HPDDM framework), 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.

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

\[
\begin{align*}
(\Delta + k^2)u &= 0 \quad \text{in } \Omega^+, \\
u &= -u^{\text{inc}} \quad \text{on } \Gamma, \\
u &\text{ outgoing.}
\end{align*}
\]

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 (\( c = c(x) \) being the local speed of sound in the medium). In what follows \( a \cdot \overline{b} \) denotes the inner product between two complex-valued vectors \( a \) and \( b \) in \( \mathbb{C}^3 \), where \( \overline{z} \) is the complex conjugate of \( z \in \mathbb{C} \). The associated norm is \( ||a|| := \sqrt{a \cdot \overline{a}} \). Here, a Dirichlet boundary condition on \( \Gamma \) 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 (\( i \) being the square root of \(-1\))

\[
\lim_{||x|| \to \infty} \frac{||x||^{d-1}}{2\pi^2} \left( \nabla u \cdot \frac{x}{||x||} - iku \right) = 0,
\]

which ensures that, first, problem \([I]\) is uniquely solvable and, second, that the scattered field \( u \) is directed from \( \Omega^- \) to infinity.
To solve problem 1 using the finite element method, the (unbounded) domain \( \Omega^+ \) must be truncated, using for example a Perfectly Matched Layer (PML) [8, 13] or a fictitious boundary \( \Gamma^\infty \) with an Absorbing Boundary Condition (ABC) [6, 20] (see e.g. [2] for a review of different methods). Therefore, the aim is now to find the field \( \hat{u} \) approximating \( u \) on the bounded domain \( \Omega \) of boundary \( \Gamma^\infty \cup \Gamma \). To simplify, notations \( \hat{u} \) and \( u \) are merged. The problem is now to solve the following problem:

\[
\begin{cases}
(\Delta + k^2)u = 0 & \text{in } \Omega, \\
u = -u^{\text{inc}} & \text{on } \Gamma, \\
\partial_n u + B u = 0 & \text{on } \Gamma^\infty.
\end{cases}
\]

The unit normal vector \( n \) is directed outside \( \Omega \) (and thus inside \( \Omega^- \) on \( \Gamma \)). For conciseness we choose the simplest local ABC, i.e., the Sommerfeld radiation condition at finite distance (zeroth-order condition):

\[Bu = -iku.\]  

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

\[2.1.2 \text{ Domain Decomposition and Transmission Operators}\]

\textbf{Substructured formulation.} The domain \( \Omega \) is now decomposed into \( N_{\text{dom}} \) disjoint subdomains \( \Omega_i \) (the \textit{substructures}) without overlap. For every \( i = 0, \ldots, N_{\text{dom}} - 1 \), let \( \Gamma_i = \Gamma \cap \partial \Omega_i \), \( \Gamma_i^\infty = \Gamma^\infty \cap \partial \Omega_i \); for \( j = 0, \ldots, N_{\text{dom}} - 1, j \neq i \), the transmission boundary \( \Sigma_{ij} = \Sigma_{ji} = \partial \Omega_i \cap \partial \Omega_j \) 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 \( \Omega_i \) (we will call them the \textit{connected} domains of \( \Omega_i \)). Finally, for all \( i \in D \), the unit normal \( n_i \) is directed into the exterior of \( \Omega_i \) and thus inside the obstacle \( \Omega^- \) (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

\[
\begin{cases}
(\Delta + k^2)u_i^{n+1} = 0 & \text{in } \Omega_i, \\
u_i^{n+1} = -u_i^{\text{inc}} & \text{on } \Gamma_i, \\
\partial_n u_i^{n+1} + B u_i^{n+1} = 0 & \text{on } \Gamma_i^\infty, \\
\partial_n u_i^{n+1} + S u_i^{n+1} = g_{ji}^n & \text{on } \Sigma_{ij}, \quad \forall j \in D_i.
\end{cases}
\]  

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

\[g_{ji}^{n+1} = -\partial_n u_i^{n+1} + Su_i^{n+1} = -g_{ij}^n + 2Su_i^{n+1}, \quad \text{on } \Sigma_{ij}.
\]  

The operator \( S \) is the invertible 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 \( u_i^{n+1} = V_i(u^{\text{inc}}, g^n) \), where \( g^n = (g^n_{ji})_{i \in D, j \in D_i} \) 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 \( g_{ji}^{n+1} = J_{ji}(g_{ij}^n, u_i^{n+1}). \)
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 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 $u_i^{n+1} = v_i^{n+1} + \tilde{u}_i^{n+1}$, where

$$v_i^{n+1} = \mathcal{V}_i(u^{\text{inc}}, 0) \quad \text{and} \quad \tilde{u}_i^{n+1} = \mathcal{V}_i(0, g^n).$$

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

$$g_{ji}^{n+1} = \mathcal{J}_{ji}(g_{ij}^n, u_i^{n+1}) = \mathcal{J}_{ji}(g_{ij}^n, \tilde{u}_i^{n+1}) + 2S v_i, \quad \text{on } \Sigma_{ij}. \quad (6)$$

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

$$\forall i \in D \quad \begin{cases} \tilde{u}_i^{n+1} = \mathcal{V}_i(0, g^n), \\ (Ag^n)_{ji} = \mathcal{J}_{ji}(g_{ij}^n, \tilde{u}_i^{n+1}), \forall j \in D_i. \end{cases} \quad (7)$$

One iteration of the domain decomposition algorithm then corresponds to

$$g^{n+1} = Ag^n + b, \quad (9)$$

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

$$(I - A)g = b, \quad (10)$$

where $I$ is the identity operator. Any iterative linear solver can be applied to (10), as for example Krylov subspace methods such as GMRES [52]. 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 $u_i = \mathcal{V}_i(u^{\text{inc}}, g)$, on every subdomain $\Omega_i$. 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 $b$:

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

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

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

5
Transmission operator. The convergence rate of the iterative solver is strongly linked to the choice of the transmission operator \( S \) \([13]\). The optimal transmission operator would be the Dirichlet-to-Neumann (DtN) map for the complement of each subdomain \([47, 46]\), 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 \( \Sigma \):

- **Evanescent Modes Damping Algorithm** \([12, 14]\):
  \[
  S_{\text{IBC}(\chi)} u = (-ik + \chi)u,
  \]
  where \( \chi \) is a real constant. This zeroth-order polynomial approximation is a generalization of the Desprès condition \([17]\), for which \( \chi = 0 \). In what follows, we will denote this family of impedance transmission conditions as IBC(\( \chi \)).

- **Optimized second-order transmission condition** \([33]\):
  \[
  S_{\text{GIBC}(a, b)} u = au + b\Delta_\Sigma u,
  \]
  where \( \Delta_\Sigma \) is the Laplace-Beltrami operator on \( \Sigma \), 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** \([13]\):
  \[
  S_{\text{GIBC}(N_p, \alpha, \varepsilon)} u = -ikC_0 u - ik \sum_{\ell=1}^{N_p} A_\ell \text{div}_\Sigma \left( \frac{1}{k^2} \nabla_\Sigma \right) \left( I + B_\ell \text{div}_\Sigma \left( \frac{1}{k^2} \nabla_\Sigma \right) \right)^{-1} u,
  \]
  where \( k_\varepsilon = k + i\varepsilon \),
  \[
  \varepsilon = 0.39k^{1/3}H^{2/3},
  \]
  \( H \) being the local mean curvature of the interface. The coefficients \( C_0, A_\ell \) and \( B_\ell \) are given by
  \[
  C_0 = e^{\alpha/2}R_{N_p}(e^{-\alpha} - 1), \quad A_\ell = \frac{e^{-\alpha} a_\ell}{1 + b_\ell(e^{-\alpha} - 1)}, \quad B_\ell = \frac{e^{-\alpha} b_\ell}{1 + b_\ell(e^{-\alpha} - 1)},
  \]
  where \( \alpha \) 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
  \[
  a_\ell = \frac{2}{2N_p + 1} \sin^2 \left( \frac{\ell \pi}{2N_p + 1} \right) \quad \text{and} \quad b_\ell = \cos^2 \left( \frac{\ell \pi}{2N_p + 1} \right).
  \]
  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(\( N_p, \alpha, \varepsilon \)).
- **PML transmission condition** \[54, 27, 57, 58\]: The operator \( S_{\text{PML}(\sigma)} \) is constructed by appending a layer \( \Omega_{\text{PML}} \) to the transmission interface, in which a PML transformation with absorption profile \( \sigma \) is applied. For example, in cartesian coordinates, the profile

\[
\sigma(x_{\text{PML}}) = \frac{1}{k(x_{\text{PML}} - \delta)}
\]

can be used, where \( \delta \) is the thickness of the PML layer and \( x_{\text{PML}} \) is the local coordinate inside the PML [9, 43].

All these methods are referred to as optimized Schwarz domain decomposition methods. Note that GIBC(\( N_p, \alpha, \varepsilon \)) and PML(\( \sigma \)) 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 [50].

### 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 \( \mathcal{V}_i \), and a surface system on the transmission boundaries, represented by \( \mathcal{S}_{ij} \). For the sake of clarity, the weak formulations are first given for a generic transmission operator \( S \). 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 \( \Sigma_{ij} \cap \Gamma_{\infty} \neq \emptyset \)), 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 \( \Omega_i \), for \( i \in D \), with no incident wave (homogeneous Dirichlet boundary condition) is detailed. We consider the general setting where PML layers \( \Omega^PML_i = \bigcup_{j \in D_i} \Omega^PML_{ij} \) are potentially appended to the artificial interfaces \( \Sigma_{ij} \), and define \( \Omega^PML_i := \Omega_i \cup \Omega^PML_i \). In what follows, the space \( H^1(\Omega^PML_i) := \{ \tilde{u}_i \in L^2(\Omega^PML_i) \mid \nabla \tilde{u}_i \in (L^2(\Omega^PML_i))^3 \} \) is the classical Sobolev space and \( H^1_0(\Omega^PML_i) \) is the space of functions \( \tilde{u}_i \in H^1(\Omega^PML_i) \) such that \( \tilde{u}_{i|\Gamma_{\infty}} = 0 \), which slightly differs from its usual definition (the Dirichlet condition is here set only on part of \( \partial \Omega^PML_i \)).

- The volume PDE \( \tilde{u}_{i}^{n+1} = \mathcal{V}_i (0, g^n) \) has the following weak formulation:

\[
\begin{cases}
\text{Find } \tilde{u}_{i}^{n+1} \text{ in } H^1_0(\Omega^PML_i) \text{ such that, for every } \tilde{u}_i' \in H^1_0(\Omega^PML_i): \\
\int_{\Omega_i} \nabla \tilde{u}_{i}^{n+1} \cdot \nabla \tilde{u}_i' \, d\Omega_i - \int_{\Omega_i} k^2 \tilde{u}_{i}^{n+1} \tilde{u}_i' \, d\Omega_i + \int_{\Gamma_{i\sigma}} B \tilde{u}_{i}^{n+1} \tilde{u}_i' \, d\Gamma_{i\sigma} \\
\quad + \sum_{j \in D_i} \int_{\Sigma_{ij}} S \tilde{u}_{i}^{n+1} \tilde{u}_j' \, d\Sigma_{ij} = \sum_{j \in D_i} \int_{\Sigma_{ij}} g_{ji}^n \tilde{u}_j' \, d\Sigma_{ij}.
\end{cases}
\]

(16)

- And the surface PDE \( g_{ji}^{n+1} = \mathcal{S}_{ji} (g_{ij}^{n}, \tilde{u}_{i}^{n+1}) \) has the following one:

\[
\begin{cases}
\text{Find } g_{ji}^{n+1} \text{ in } H^1(\Sigma_{ij}) \text{ such that, for every } g_{ji}' \in H^1(\Sigma_{ij}): \\
\int_{\Sigma_{ij}} g_{ji}^{n+1} g_{ji}' \, d\Sigma_{ij} = - \int_{\Sigma_{ij}} g_{ij}^n g_{ji}' \, d\Sigma_{ij} + 2 \int_{\Sigma_{ij}} S \tilde{u}_{i}^{n+1} g_{ji}' \, d\Sigma_{ij}.
\end{cases}
\]

(17)
On the transmission boundaries. Depending on the choice of the transmission operator $S$, the quantities $\int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} \tilde{u}'_i \, d\Sigma_{ij}$ and $\int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} g'_{ji} \, d\Sigma_{ij}$ expand as follows:

- **IBC($\chi$):**
  \[
  \int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} \tilde{u}'_i \, d\Sigma_{ij} := \int_{\Sigma_{ij}} (-ik + \chi)\tilde{u}_i^{n+1} \tilde{u}'_i \, d\Sigma_{ij}; \tag{18}
  \]
  \[
  \int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} g'_{ji} \, d\Sigma_{ij} := \int_{\Sigma_{ij}} (-ik + \chi)\tilde{u}_i^{n+1} g'_{ji} \, d\Sigma_{ij}. \tag{19}
  \]

- **GIBC($a, b$):**
  \[
  \int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} \tilde{u}'_i \, d\Sigma_{ij} := \int_{\Sigma_{ij}} a\tilde{u}_i^{n+1} \tilde{u}'_i \, d\Sigma_{ij} - \int_{\Sigma_{ij}} b\nabla \tilde{u}_i^{n+1} \cdot \nabla \tilde{u}'_i \, d\Sigma_{ij}; \tag{20}
  \]
  \[
  \int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} g'_{ji} \, d\Sigma_{ij} := \int_{\Sigma_{ij}} a\tilde{u}_i^{n+1} g'_{ji} \, d\Sigma_{ij} - \int_{\Sigma_{ij}} b\nabla \tilde{u}_i^{n+1} \cdot \nabla g'_{ji} \, d\Sigma_{ij}. \tag{21}
  \]

- **GIBC($N_p, \alpha, \varepsilon$):**
  \[
  \int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} \tilde{u}'_i \, d\Sigma_{ij} := -ikC_0 \int_{\Sigma_{ij}} \tilde{u}_i^{n+1} \tilde{u}'_i \, d\Sigma_{ij} + \sum_{\ell=1}^{N_p} A_{\ell} \int_{\Sigma_{ij}} \frac{1}{k_2} \nabla \tilde{u}_i^{n+1} \varphi_\ell \cdot \nabla \tilde{u}_i^{n+1} \tilde{u}'_i \, d\Sigma_{ij}, \tag{22}
  \]
  where, for every $\ell = 1, \ldots, N_p$, the function $\varphi_\ell$ is obtained through the resolution of
  \[
  \begin{cases}
    \text{Find } \varphi_\ell \text{ in } H^1(\Sigma_{ij}) \text{ such that, for every } \varphi_\ell ' \in H^1(\Sigma_{ij}): \\
    - \int_{\Sigma_{ij}} \tilde{u}_i^{n+1} \varphi_\ell ' \, d\Sigma_{ij} - B_\ell \int_{\Sigma_{ij}} \frac{1}{k_2} \nabla \tilde{u}_i^{n+1} \varphi_\ell \cdot \nabla \tilde{u}_i^{n+1} \varphi_\ell ' \, d\Sigma_{ij} + \int_{\Sigma_{ij}} \varphi_\ell \cdot \varphi_\ell ' \, d\Sigma_{ij} = 0;
  \end{cases}
  \]
  \[
  \int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} g'_{ji} \, d\Sigma_{ij} := -ikC_0 \int_{\Sigma_{ij}} \tilde{u}_i^{n+1} g'_{ji} \, d\Sigma_{ij} - ik \sum_{\ell=1}^{N_p} A_{\ell} \int_{\Sigma_{ij}} (\tilde{u}_i^{n+1} - \varphi_\ell)g'_{ji} \, d\Sigma_{ij}. \tag{23}
  \]

- **PML($\sigma$):**
  \[
  \int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} \tilde{u}'_i \, d\Sigma_{ij} := \int_{\Omega_{\text{PML}}} D \nabla \tilde{u}_i^{n+1} \cdot \nabla \tilde{u}'_i \, d\Omega_{\text{PML}} - \int_{\Omega_{\text{PML}}} k^2 E \tilde{u}_i^{n+1} \tilde{u}'_i \, d\Omega_{\text{PML}}; \tag{24}
  \]
  \[
  \int_{\Sigma_{ij}} S\tilde{u}_i^{n+1} g'_{ji} \, d\Sigma_{ij} := \int_{\Omega_{\text{PML}}} D \nabla \tilde{u}_i^{n+1} \cdot \nabla g'_{ji} \, d\Omega_{\text{PML}} - \int_{\Omega_{\text{PML}}} k^2 E \tilde{u}_i^{n+1} g'_{ji} \, d\Omega_{\text{PML}}, \tag{25}
  \]

where $D = \text{diag}(1, \gamma_x, \gamma_x)$ and $E = \gamma_x$, with $\gamma_x(x_{\text{PML}}) = 1 + \frac{2}{3} \sigma_x(x_{\text{PML}})$, that is, we consider a 1D PML with an absorption function that grows only in the direction normal to the interface. In the last expression the domain of definition of the test functions $g_{ji}'$ on $\Sigma_{ij}$ is extended to the neighboring PML layer $\Omega_{\text{PML}}$, 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 $\Omega^{-}$ with a smooth boundary $\Gamma$ and a three dimensional medium. When illuminated by an incident electric field $E^{\text{inc}}$, a perfectly conducting body $\Omega^{-}$ generates a scattered field $E$, solution of the following exterior electromagnetic scattering problem:

\[
\begin{align*}
\text{curl curl } E - k^2 E &= 0, \quad \text{in } \Omega^+, \\
\gamma^{T}(E) &= -\gamma^{T}(E), \quad \text{on } \Gamma, \\
\lim_{||x|| \to \infty} \frac{x}{||x||} \times (\text{curl } E + ikE) &= 0,
\end{align*}
\]  

(26)

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

The $\text{curl}$ operator is defined by $\text{curl } \mathbf{a} := \nabla \times \mathbf{a}$, 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 $\mathbf{b}$. 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 fictitious boundary $\Gamma^\infty$ surrounding $\Omega^{-}$. The problem to be solved is then defined on the bounded domain $\Omega$, with boundaries $\Gamma$ and $\Gamma^\infty$:

\[
\begin{align*}
\text{curl curl } E - k^2 E &= 0, \quad \text{in } \Omega, \\
\gamma^{T}(E) &= -\gamma^{T}(E), \quad \text{on } \Gamma, \\
\gamma^{t}(\text{curl } E) + \mathcal{B}(\gamma^{T}(E)) &= 0, \quad \text{on } \Gamma^\infty,
\end{align*}
\]  

(27)

with $\gamma^{t}$ the tangential trace operator:

\[\gamma^{t}: \mathbf{v} \mapsto \mathbf{n} \times \mathbf{v}.\]

As above, the unit normal $\mathbf{n}$ is outwardly directed to $\Omega$ and, to simplify, the solution of the above problem is still designated by $E$. The operator $\mathcal{B}$ is an approximation of the Magnetic-to-Electric (MtE) operator. Several expressions of $\mathcal{B}$ can be found in the literature [23]; for conciseness we restrict ourselves to the simplest one, the well-known Silver-Müller ABC at finite distance: $\mathcal{B} = ik$, similar to [3] for acoustics modulo the sign (due to the trace operator definitions).

2.2.2 Domain Decomposition and Transmission operators

Substructured formulation. The optimized Schwarz domain decomposition without overlap is now considered for Maxwell’s problem (27). The domain $\Omega$ 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 $(E^{n+1}_i)_{i \in D}$ at iteration $n+1$ involves, first, the solution of the $N_{\text{dom}}$ following problems

\[
\begin{align*}
\text{curl curl } E^{n+1}_i - k^2 E^{n+1}_i &= 0, \quad \text{in } \Omega_i, \\
\gamma^{T}_i(E^{n+1}_i) &= -\gamma^{T}_i(E^{\text{inc}}), \quad \text{on } \Gamma_i, \\
\gamma^{t}_i(\text{curl } E^{n+1}_i) + \mathcal{B}(\gamma^{T}_i(E^{n+1}_i)) &= 0, \quad \text{on } \Gamma^{\infty}_i, \\
\gamma^{t}_i(\text{curl } E^{n+1}_i) + S(\gamma^{T}_i(E^{n+1}_i)) &= g^{n}_{ij}, \quad \text{on } \Sigma_{ij}, \forall j \in D_i,
\end{align*}
\]  

(28)
and then forming the quantities $g_{ji}^{n+1}$ through

$$g_{ji}^{n+1} = \gamma_i^T(\text{curl } E_i^{n+1}) + S(\gamma_i^T(E_i^{n+1})) = -g_{ij}^n + 2S(\gamma_i^T(E_i^{n+1})), \quad \text{on } \Sigma_{ij},$$  

(29)

where, for $i \in D$, $E_i = E_{i\Omega}$, $S$ is an invertible transmission operator through the interfaces $\Sigma_{ij}$ and $\gamma_i^T$ are the local tangential trace and tangential component trace operators:

$$\gamma_i^T : v_i \mapsto n_i \times v_i|_{\partial \Omega_i} \quad \text{and} \quad \gamma_i^T : v_i \mapsto n_i \times (v_i|_{\partial \Omega_i} \times n_i),$$

with $n_i$ the outward-pointing unit normal to $\Omega_i$.

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

1. $E_i^{n+1} = \mathcal{K}_i(E_i^{inc}, g^n) \iff E_i^{n+1}$ is solution of problem (28), where $g^n = (g^n_{ji})_{j \in D, j \in D_i}$ collects all the unknowns at iteration $n$;

2. $g_{ji}^{n+1} = \mathcal{J}_{ji}(g_{ij}^n, E_i^{n+1}) \iff g_{ji}^{n+1}$ is solution of problem (29).

By linearity, we decompose the field $E_i^{n+1}$ as $E_i^{n+1} = F_i^{n+1} + \tilde{E}_i^{n+1}$, where

$$F_i^{n+1} = \mathcal{K}_i(E_{inc}, 0) \quad \text{and} \quad \tilde{E}_i^{n+1} = \mathcal{K}_i(0, g^n).$$  

(30)

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

$$(I - A) g = b,$$  

(31)

that can be solved by a Krylov subspace solver.

As in the acoustic case, for a vector $g^n$, the quantity $A g^n$ is given by, for $i \in D$ and $j \in D_i$, $(A g^n)_{ji} = \mathcal{J}_{ji}(g_{ij}^n, E_i^{n+1})$. The information about the incident wave is contained in the right-hand side: $b_{ji} = \mathcal{J}_{ji}(0, F_i)$. The domain decomposition algorithm for the Maxwell’s equation 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 $F_i, E^{inc}, g$ and $E_i$, 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 $\Sigma$:

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

$$S_{\text{IBC}(0)}(\gamma^T(E)) = i k \gamma^T(E),$$  

(32)

- **Optimized second-order transmission condition** [51]:

$$S_{\text{GIBC}(a,b)}(\gamma^T(E)) = i k \left( I + \frac{a}{k^2} \nabla_\Sigma \text{div}_\Sigma \right)^{-1} \left( I - \frac{b}{k^2} \text{curl}_\Sigma \text{curl}_\Sigma \right) (\gamma^T(E)),$$  

(33)

where the curl operator is the dual operator of $\text{curl}$ and where $a$ and $b$ are chosen so that an optimal convergence rate is obtained for the (TE) and (TM) modes; see [51] 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].
\[ S_{\text{GIBC}(N_p, a, e)}(\gamma^T(\mathbf{E})) = \imath k \left( C_0 + \sum_{\ell=1}^{N_p} A_\ell X (I + B_\ell X)^{-1} \right)^{-1} \left( I - \text{curl} \frac{1}{k_\varepsilon} \text{curl}_\Sigma \right) (\gamma^T(\mathbf{E})), \]

with \( X := \nabla \sigma \frac{1}{k_\varepsilon} \text{div}_\Sigma - \text{curl} \frac{1}{k_\varepsilon} \text{curl}_\Sigma, \) and where \( k_\varepsilon, C_0, A_\ell \) and \( B_\ell \) 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** [57, 58]: The operator \( S_{\text{PML}(\sigma)} \) is constructed by appending a layer \( \Omega_{\text{PML}} \) to the transmission interface, into which a PML transformation with absorption profile \( \sigma \) 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 \( \Omega_i \), 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 \( \Omega_{\text{PML}} \) are potentially appended to the artificial interfaces \( \Sigma_{ij} \), and define \( \Omega_i^* := \Omega_i \cup \Omega_{\text{PML}} \). The space of complex-valued curl-conforming vector fields on \( \Omega_i^* \) is denoted by \( H(\text{curl}, \Omega_i^*) := \{ \mathbf{W} \in (L^2(\Omega_i^*))^3 \) such that \( \text{curl} (\mathbf{W}) \in (L^2(\Omega_i^*))^3 \}. \) The functional space \( \mathbf{H}_0(\text{curl}, \Omega_i^*) \) is the space of functions \( \mathbf{W}_i \) in \( H(\text{curl}, \Omega_i^*) \) such that \( \gamma_i^T(\mathbf{W}_i) = 0 \) on \( \Gamma_i = 0 \) (the boundary condition is only imposed on the part \( \partial \Omega_i^* \)).

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

  \[ \begin{cases} \text{Find } \tilde{\mathbf{E}}_i^{n+1} \in \mathbf{H}_0(\text{curl}, \Omega_i) \text{ such that, for every } \tilde{\mathbf{E}}'_i \in \mathbf{H}_0(\text{curl}, \Omega_i): \\ \int_{\Omega_i} \text{curl} \tilde{\mathbf{E}}_i \cdot \text{curl} \tilde{\mathbf{E}}'_i \, d\Omega_i - \int_{\Omega_i} k^2 \tilde{\mathbf{E}}_i \cdot \tilde{\mathbf{E}}'_i \, d\Omega_i - \int_{\Gamma_i} \mathbf{B}^T(\tilde{\mathbf{E}}_i) \cdot \tilde{\mathbf{E}}'_i \, d\Gamma_i^\infty \\ \quad - \sum_{j \in D_i} \int_{\Sigma_{ij}} \mathbf{S}(\gamma_i^T(\tilde{\mathbf{E}}_i)) \cdot \tilde{\mathbf{E}}'_i \, d\Sigma_{ij} = - \sum_{j \in D_i} \int_{\Sigma_{ij}} \mathbf{g}_{ij}^n \cdot \tilde{\mathbf{E}}'_i \, d\Sigma_{ij}. \end{cases} \]  

- The surface PDE \( \mathbf{g}_{ji}^{n+1} = \mathcal{B}_{ji}(\mathbf{g}_{ij}^n, \tilde{\mathbf{E}}_i^{n+1}) \) has the following one:

  \[ \begin{cases} \text{Find } \mathbf{g}_{ji}^{n+1} \text{ in } \mathbf{H}(\text{curl}, \Sigma_{ij}) \text{ such that, for every } \mathbf{g}'_{ji} \in \mathbf{H}(\text{curl}, \Sigma_{ij}): \\ \int_{\Sigma_{ij}} \mathbf{g}_{ji}^{n+1} \cdot \mathbf{g}'_{ji} \, d\Sigma_{ij} = - \int_{\Sigma_{ij}} \mathbf{g}_{ij}^n \cdot \mathbf{g}'_{ji} \, d\Sigma_{ij} + 2 \int_{\Sigma_{ij}} \mathbf{S}(\gamma_i^T(\tilde{\mathbf{E}}_i^{n+1})) \cdot \mathbf{g}'_{ji} \, d\Sigma_{ij}. \end{cases} \]

**On the transmission boundaries.**

- **IBC(0):**

  \[ \int_{\Sigma_{ij}} \mathbf{S}(\gamma_i^T(\tilde{\mathbf{E}}_i)) \cdot \tilde{\mathbf{E}}'_i \, d\Sigma_{ij} := \int_{\Sigma_{ij}} \imath k (\gamma_i^T(\tilde{\mathbf{E}}_i)) \cdot \tilde{\mathbf{E}}'_i \, d\Sigma_{ij}; \]  

- **GIBC(0, b):**

  \[ \int_{\Sigma_{ij}} \mathbf{S}(\gamma_i^T(\tilde{\mathbf{E}}_i)) \cdot \tilde{\mathbf{E}}'_i \, d\Sigma_{ij} := \int_{\Sigma_{ij}} \imath k r \cdot \tilde{\mathbf{E}}'_i \, d\Sigma_{ij}. \]
where the function \( \mathbf{r} \in \mathbf{H}(\text{curl}, \Sigma_{ij}) \) is obtained through the solution of

\[
\begin{aligned}
\int_{\Omega} \mathbf{S}(\gamma^T_i(\mathbf{E}_i^{n+1})) \cdot \mathbf{g}_{ji} \, d\Sigma_{ij} := \int_{\Sigma_{ij}} i\mathbf{k} \cdot \mathbf{g}_{ji} \, d\Sigma_{ij}.
\end{aligned}
\]

- \text{GIBC}(N_p, \alpha, \varepsilon):
  \[
  \int_{\Sigma_{ij}} \mathbf{S}(\gamma^T_i(\mathbf{E}_i^i)) \cdot \mathbf{E}_i^i \, d\Sigma_{ij} := \int_{\Sigma_{ij}} i\mathbf{k} \cdot \mathbf{E}_i^i \, d\Sigma_{ij},
  \]

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

\[
\begin{aligned}
\int_{\Sigma_{ij}} \mathbf{S}(\gamma^T_i(\mathbf{E}_i^{n+1})) \cdot \mathbf{g}_{ji} \, d\Sigma_{ij} := \int_{\Sigma_{ij}} i\mathbf{k} \cdot \mathbf{g}_{ji} \, d\Sigma_{ij}.
\end{aligned}
\]

- \text{PML}(\sigma):
  \[
  \int_{\Sigma_{ij}} \mathbf{S}(\gamma^T_i(\mathbf{E}_i^{n+1})) \cdot \mathbf{g}_{ji} \, d\Sigma_{ij} := \int_{\Sigma_{ij}} i\mathbf{k} \cdot \mathbf{g}_{ji} \, d\Sigma_{ij}.
  \]

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 \((\text{http://getdp.info})\) and the open source mesh generator Gmsh \((\text{http://gmsh.info})\). 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 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
### 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 with a Dirichlet boundary condition, we naturally consider the classical linear finite element (P1) space, using the built-in `BF_Node` basis functions. 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 $\varphi_\ell$ 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 (see Listing 3), for both the volume and surface unknowns. 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 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 \( \text{idom} \) 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 \([\ldots]\) 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) [57]. 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\((N_p, \alpha, \varepsilon)\) condition [22], the auxiliary quantities, such as \(r, \varphi_\ell\) and \(\rho_\ell\) 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) or the construction of integration rules (Jacobian and Integration objects).
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(\chi) 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(Np, 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)
Listing 9: Equation part of the surface formulation (17) for the Helmholtz problem. (Code extracted from the file Helmholz.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 $\Omega_i$ and is then in charge of solving the volume PDE on $\Omega_i$ 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.

---

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 $\Omega_i$ and is then in charge of solving the volume PDE on $\Omega_i$ 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.

---

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 $\Omega_i$ and is then in charge of solving the volume PDE on $\Omega_i$ 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.
summarize, 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 briefly 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: $v_i = \mathcal{V}_i(u_{\text{inc}}, 0)$.
2. Compute the local contribution to the right hand side: $b_{ji} = \mathcal{T}_{ji}(0, v_i)$.
3. Exchange information about the (geometric) topology between processes.
4. Enter the parallel iterative Krylov subspace solver for the system $(I - A)g = b$; at each iteration, compute the local contribution $(Ag^n)_{ji}$ 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: $\tilde{u}_{i}^{n+1} = \mathcal{V}_i(0, g^n)$.
   - Solve the local surface problems: $(Ag^n)_{ji} = \mathcal{T}_{ji}(g^n_{ij}, \tilde{u}_{i}^{n+1}), \forall j \in D_i$.
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: $u_i = \mathcal{V}_i(u_{\text{inc}}, g)$.

### 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 $N_{\text{dom}}$ subdomains, from “left” (0) to “right” ($N_{\text{dom}} - 1$) as depicted in the table below (each cell being a subdomain):

<table>
<thead>
<tr>
<th>Subdomain number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>$N_{\text{dom}} - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indices $(j, i)$ (of $g_{ji}$)</td>
<td>(1, 0)</td>
<td>(0, 1)</td>
<td>(2, 1)</td>
<td>(1, 2)</td>
<td>(2, 3)</td>
</tr>
<tr>
<td>Global index of $(j, i)$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

To simplify, the number of MPI processes is set to $N_{\text{dom}}$ and the domain $\Omega_i$ 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 ($\Omega_0$ 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:

<table>
<thead>
<tr>
<th>MPI Process</th>
<th>ListOfSubdomains</th>
<th>ListOfFields</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0]</td>
<td>[0]</td>
</tr>
<tr>
<td>1</td>
<td>[1]</td>
<td>[1, 2]</td>
</tr>
<tr>
<td>2</td>
<td>[2]</td>
<td>[3, 4]</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$N_{\text{dom}} - 1$</td>
<td>[$N_{\text{dom}} - 1$]</td>
<td>[$2N_{\text{dom}} - 3$]</td>
</tr>
</tbody>
</table>

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

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 $N_{\text{dom}}$ 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 $j, k \in D_i, j \neq k$). 21
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:

<table>
<thead>
<tr>
<th>Subdomain number</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>$N_{\text{dom}} - 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index of Field storing $g_{ji}$</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>...</td>
</tr>
<tr>
<td>Index of connected Field storing $g_{ij}$</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

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
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.)

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:

Coarse grids and 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

---

For the complete list of available solvers, see http://www.mcs.anl.gov/petsc/
Listing 12: Creation of the list `ListOfConnectedFields` for a layered decomposition. (Code extracted from the file `Decomposition.pro`.)

```plaintext
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
EndFor
```
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 SolveSurfacePDE;
  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)
Listing 16: Macro to update the surface fields. (Code extracted from the file Schwarz Macros.pro)

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, 54, 57]. 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 [58].

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.

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 presented in this paper for the Helmholtz 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 pre-compiled 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

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
Figure 2: Sample models available online at [http://onelab.info/wiki/GetDDM](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 [57]. g.: guided acoustic or electromagnetic waves in the COBRA benchmark defined by the JINA98 workgroup [58]. h.: acoustic waves in the underground Marmousi model [54].

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 $\frac{\omega}{\pi} = 700$ (uniformly discretized by a mesh density of 20 points per the smallest wavelength), with 358 subdomains for a total of 4296 CPUs. 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, each with 2.6Gb of RAM, with 302,715,000 complex unknowns.
All the specific problem files include the same generic implementation of the Schwarz domain decomposition algorithm, contained in the following 5 files:

- 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.
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 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 nº2.5020.11, and the Tier-1 supercomputer of the Fédération Wallonie-Bruxelles, funded by the Walloon Region under grant nº1117545.

References


32


