High performance computing for numerical methods and data analysis

—

Project 2: Domain decomposition

Emile Parolin

Notations In the following we use the notations of the lecture notes on domain decomposition methods without defining them again.

1 Model problem and its finite element solution

1.1 Model problem

We consider the Helmholtz equation model problem of the lecture notes in dimension $d = 2$. The domain is a rectangle $\Omega := (0, L_x) \times (0, L_y)$ for $L_x, L_y > 0$.

The source term f consists in $N_s \geq 1$ regularized point sources defined by

$$f(\mathbf{x}) := \sum_{i=1}^{N_s} w_i e^{-\frac{10}{\lambda^2}|\mathbf{x}-\mathbf{s}_i|^2}, \tag{1}$$

where $\{\mathbf{s}_i\}_{i=1}^{N_s}$ denote the source locations and $\{w_i\}_{i=1}^{N_s}$ are associated weights. Besides we let $g = 0$, which is crude model for an outgoing radiation condition.

1.2 Finite element discretization

The triangular mesh $\mathcal{T}$ stems from a uniform discretization in both directions using $N_x, N_y \in \mathbb{N}$ points in the x and y directions respectively. The discretization of the solution uses conformal $\mathbb{P}_1$ scalar Lagrange finite elements, see Figure 1. To resolve the oscillations of the waves, the mesh should be sufficiently fine. A general rule of thumb is that there should be at least 10 points per

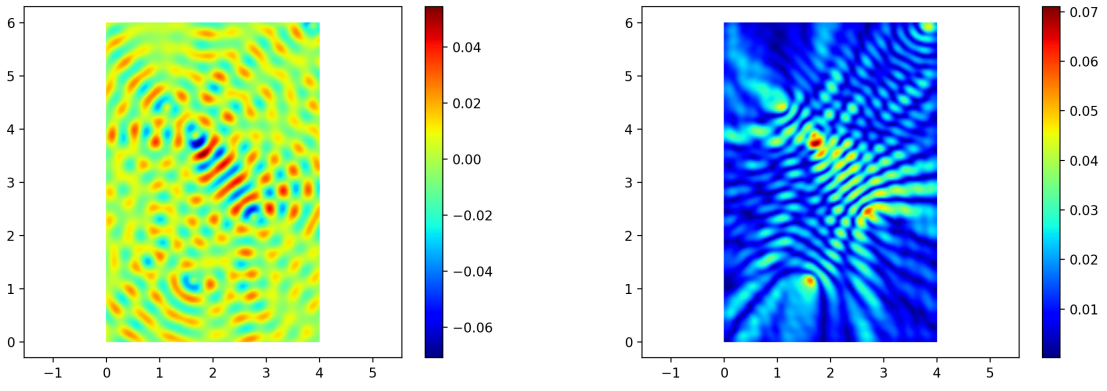

Figure 1: Real part (left) and modulus (right) of a typical numerical simulation ($k = 16$).

wavelength λ, namely the typical mesh size h should be smaller than $\lambda/10$. A possible Python implementation of such a method is attached.

The resolution of the linear system can be obtained using a direct solver (the code attached uses `scipy.sparse.linalg.spsolve`). However direct solvers typically do not scale very well on distributed architectures. One then turns to iterative solvers, in particular to GMRES since this linear system is indefinite (here we use `scipy.sparse.linalg.gmres`).

Try to refine the mesh (increasing N_x and N_y). Note that the meshes should remain uniform (isotropic), in particular the mesh refinement should not generate elongated triangles. How does the number of iterations to reach a given fixed tolerance changes?

2 Sequential domain decomposition algorithm

The purpose of this project is to investigate the domain decomposition method seen in the lectures as a possible way to obtain efficient iterative solvers to solve the previous linear system.

We decompose the domain Ω into J sub-domains. The decomposition will be done in the y direction only, so we get J horizontal slabs of size $(0, L_x) \times (0, L_y/J)$. To avoid unecessary difficulties, we assume that $N_y - 1$ is a multiple of J. More generally we shall make use of the particular way the domain is split to ease the implementation of the method. Below is a step by step guide to implement the method.

2.1 Mesh

The global mesh consists of the global coordinate array `vtx` (`numpy` matrix with $|V(\Omega)|$ lines and 2 columns for the x and y coordinates of the $|V(\Omega)|$ vertices) and the global connectivity array `elt` (`numpy` matrix with as many lines as triangles and 3 columns corresponding to the indices in `vtx` of the 3 vertices of one triangle). The implementation should construct only local meshes and should not use the global mesh but construct J local versions, for each sub-domain.

Notice that the function `mesh` that is provided has a particularly convenient numbering of the vertices (given by the order in `vtx`). In particular the N_x first lines in `vtx` correspond to the bottom $y = 0$ and the N_x last lines correspond to the top $y = L_y$ of the $(0, L_x) \times (0, L_y)$ domain.

1. Implement a function `local_mesh` to obtain the coordinate array `vtxj` and the triangle connectivity array `eltj` of the local j-th subdomain. This function should call the function `mesh` that is already implemented and take L_x, L_y, N_x, N_y, j and J as arguments.

2. Implement a function `local_boundary` to obtain *two* edge arrays of the local j-th subdomain: the first one `beltj_phys` should correspond to the physical boundary $\partial\Omega_j \cap \partial\Omega$ and the second one `beltj_artf` should correspond to the interfaces with its neighbours. This body of this function can be adapted from the one of the function `boundary` that is already implemented. This function should take N_x, N_y, j and J as arguments.

2.2 Local restriction matrices

Recall that degrees of freddom and coefficients in the unknown vector of the linear system correspond to the mesh vertices in $\mathbb{P}_1$ Lagrange finite elements. In particular, the numbering of the vertices gives a numbering of the degrees of freedom and coefficients of the vectors.

1. Implement a function `Rj_matrix` that constructs the local restriction matrix $\mathbf{R}_j$. This function should take N_x, N_y, j and J as arguments.

2. Implement a function `Bj_matrix` that constructs the local restriction matrix $\mathbf{B}_j$. This function should take N_x, N_y, j, J and `beltj_artf` as arguments.

3. In this project, we take Σ_j to be made of the artificial interfaces, excluding the physical boundary, namely we assume that

$$\Sigma_j := \partial\Omega_j \setminus \partial\Omega. \tag{2}$$

Implement a function `Cj_matrix` that constructs the local restriction matrix $\mathbf{C}_j$ defined by $\mathbf{C}_j \in \{0,1\}^{|V(\Sigma_j)|\times|\mathbb{V}(\mathcal{S})|}$

$$\mathbf{C}_j : \mathbb{V}(\mathcal{S}) \to V(\Sigma_j), \tag{3}$$

$$\mathbf{C}_j \mathbb{x} := \mathrm{x}_j, \ \forall \mathbb{x} = (\mathrm{x}_j)_{j=1}^{J}. \tag{4}$$

This boolean matrix selects the local part x_j of a global vector $\mathbb{x}$ (representing e.g. the unknown of the linear system of the interface problem). This function should take N_x, N_y, j and J as arguments.

2.3 Local problems

1. Implement a function `Aj_matrix` that constructs the local problem matrix $\mathbf{A}_j$. This function should take `vtxj`, `eltj`, `beltj_phys` and the wavenumber κ as arguments and reuse the functions `mass` and `stiffness` that are already implemented, following what is done for the global problem.

2. Implement a function `Tj_matrix` that constructs the local transmission matrix $\mathbf{T}_j$ defined as κ times the mass matrix associated to the interfaces Σ_j. This function should take `vtxj`, `beltj_artf`, $\mathbf{B}_j$ and κ as arguments.

3. Implement a function `Sj_factorization` that constructs the local problem matrix $\mathbf{A}_j - \imath\mathbf{B}_j^*\mathbf{T}_j\mathbf{B}_j$ and performs its LU factorization (using `scipy.sparse.linalg.splu`). This function should take $\mathbf{A}_j$, $\mathbf{T}_j$ and $\mathbf{B}_j$ as arguments.

4. Implement a function `bj_vector` that constructs the local part of the right-hand-side $\mathbf{b}_j$. This function should take `vtxj`, `eltj`, `ps` and κ as arguments and should use the functions `mass` and `point_source` already implemented.

2.4 Global operators

1. Implement a function `S_operator` that computes the action of the operator $\mathbb{S}$ on a global vector of interface unknowns $\mathbb{x}$. Recall that this operator is a purely block diagonal operator, with one sub-domain per block. The implementation of this function requires the factorization of the local problem, the local matrices $\mathbf{B}_j$, $\mathbf{T}_j$ and $\mathbf{C}_j$ for each subdomain.

2. Implement a function `Pi_operator` that computes the action of the operator $\mathbf{\Pi}$ on a global vector of interface unknowns $\mathbb{x}$. Recall that this operator exchanges information by subdomains. Thanks to the particular choice of the local boundary Σ and the transmission matrices $\mathbf{T}_j$, the exchange in this project is purely local. This means that the coefficients associated to a vertex on one interface between two sub-domains are simply swapped. There is no need to solve a linear system.

3. Implement a function `g_vector` that constructs the global right-hand-side $\mathbb{g}$ of the interface problem. The implementation of this function requires the factorization of the local problem, the local matrices $\mathbf{B}_j$, $\mathbf{C}_j$ and the local part $\mathbf{b}_j$ for each subdomain, as well as the operator $\mathbf{\Pi}$.

2.5 Fixed point and GMRES methods

1. Implement the fixed point method, with a relaxation parameter $\omega \in (0, 1)$.
2. Solve the interface problem using GMRES (`scipy.sparse.linalg.gmres`), since the matrix $\mathrm{I} + \mathbf{\Pi}\mathbb{S}$ of the linear system of the interface problem is not explicitly assembled you need to use a `scipy.sparse.linalg.LinearOperator` which can be constructed from a function that performs the matrix-vector product.
3. Plot the convergence curve of the residual for both methods.
4. How does the convergence is affected by mesh refinement ?
5. How does the convergence is affected by increasing the number of sub-domains J:
   - for a fixed domain size ?
   - for a fixed number of degrees of freedom by sub-domain ?
6. Implement a function `uj_solution` that computes the local solution $\mathbf{u}_j$ from a solution of the linear system $\mathbb{x}$. The implementation of this function requires the factorization of the local problem, the local matrices $\mathbf{B}_j$, $\mathbf{T}_j$, $\mathbf{C}_j$ and the local part $\mathbf{b}_j$ for each subdomain.

   Using `uj_solution` and `plot_mesh` (already implemented), plot the local solutions on the same figure. Note that the local solutions $\mathbf{u}_j$ are piecewise continuous (with very small jumps at the interfaces at convergence), it is enough to just plot the local solutions independently.
7. Compare the run time of your algorithm using GMRES with the resolution of the full problem using GMRES.

3 Parallel domain decomposition algorithm

The above algorithm can be easily parallelized, assigning one core to each sub-domain.

1. Implement a parallel fixed point solver using MPI (`mpi4py`). The global solution vector $\mathbb{x}$ should be distributed and sub-domains should communicate data with their neighbours in the definition of the operator $\mathbf{\Pi}$. In addition, a reduction operation to compute the residual at each step to test the convergence needs to be performed. At convergence, the solution $\mathbb{x}$ of the interface problem is collected on the rank process 0 and the solution is plotted.
2. Investigate the parallel efficiency of your algorithm on the run time.
3. **(Optional part)** Solve the skeleton problem using PETSc (through `petsc4py`) and its parallel GMRES.

Note that the optional part is *not* required as part of the assignment.

4 Content of the report (10 pages maximum)

The report should explain your implementations strategies and the results of your numerical investigations playing with the parameters of the code. Notations and definitions/algorithms from the lecture notes should not be introduced again. The Python and MPI code used for the implementation of the algorithms should be included as well, in the .zip file of the submission.