Naive Implementations of Overlapping Domain Decomposition Finite Element Method (FE-DDM) of Electrostatic Problem

Junda Feng (jundaf2)

***Abstract*—Domain Decomposition Methods have been widely used in the Computational Electromagnetics community in the recent a few decades to tackle large or multi-scale problems with Finite Element Methods. In this course project, three numerical methods with respect to DDM and FEM are carried out to solve one two-dimensional electrostatic problem. The first one FEM-CG is using the classic FEM to solve the total solution domain of the boundary value problem with Conjugated Gradient method for solving the sparse linear system. In the second one FE-DDM-ASM, the total solution domain is divided into two-by-two overlapped subdomains and solved using Additive Schwarz Method. In the third one FE-DDM-PCG, the ASM is used as a preconditioner of the Preconditioned Conjugated Gradient method. Because the nowadays DDM is highly related to and cannot be separated from parallel programming, the contents in this report involve some discussion related to the parallel programming.**

***Index Terms*—Finite element method (FEM), Domain decomposition method (DDM), Boundary value problem (BVP), Partial differential equations (PDE), Electromagnetics, High performance computing (HPC), Matlab, Additive Schwarz Method (ASM), Conjugated Gradient Method (CGM), Preconditioned Conjugated Gradient (PCG), ILU factorization.**

I. Introduction

BEFORE we begin, we need to introduce the composition of the files attached to this project report in the project folder because this FEM project is much more sophisticated than the previous FDTD project. This is easy to understand in that FEM is more useful than FDTD in most of the real world applications so that it have more branches during years of development by scientists and engineers in the field of computation electromagnetic (CEM).

There are basically five .m files used in this project, three of them are user-defined functions, one of them is the executable script, another one of them is a helper function.

- 'C1_FEM_CGM.m'
  This contains the function of the traditional FEM with CG solver described in Section II.
- 'C2_FEM_DDM_ASM.m'
  This contains the function of the FE-DDM based on ASM described in Section II.
- 'C3_FEM_DDM_PCG.m'
  This contains the function of the FE-DDM based on Preconditioned Conjugated Gradient using Additive Schwarz DDM as preconditioner.
- 'Main_Pre_Post_Processing.m'
  This is the main executable script for pre-processing (geometry and mesh generation), benchmarking the three functions C1-C3 and post-processing the results returned by the three function. Because all the functions are used to deal with the toy electrostatic problem proposed in Section II, the geometry and mesh information will only be generated at this file. So be sure to run this script rather than the other three .m files to get the desirable results.
- 'stlwrite.m'
  This helper file is downloaded from the Matlab community to convert connectivity information from array to .stl file. The file 'naive_fe_ddm.stl' is generated using this helper function on the fly when you run the benchmarking script.

Other .png picture files are generated and saved from the Matlab results after running the benchmarking script.

In the first section of the executable script, you can specify the following parameters:

- N
  N is the resolution of the solution Φ, you can think of it as the number of points along each direction of the two-dimensional domain to be interpolated using the coefficients associated with the linear basis functions defined at the nodes of the triangular elements or the number of pixels along each direction of the picture generated as final result.
- Hmax
  Hmax is the approximate maximum of the mesh edge lengths. The smaller this value, the finer the mesh is, the more number of nodes (and thus unknowns) exists in the entire problem domain.
- use_parallel_for
  use_parallel_for is a Boolean parameter for you to decide whether to use the parallel processing technique provided by Matlab (parfar) to perform Additive Schwarz DDM iterations in a parallel manner when running function C2.
- plot_logistics
  plot_logistics is a Boolean parameter to decide whether to plot the less important results such as the mesh information and the finite element matrices.

In the following part of this section, we will introduce the finite element method (FEM), the domain decomposition method (DDM) and the iterative linear system solvers. In par-

Junda Feng is with the Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign e-mail: jundaf2@illinois.edu

ticular, we will pay some attention to what can be parallelized in each of these three steps.

A. FEM

Both the finite-difference method (FDM) and FEM [1] are commonly used in numerical electromagnetics and are common techniques for solving the BVP of PDE numerically. The FDM is relatively simple to implement (stencil operations), but it is not easy to deal with underlying complex boundaries. The finite element method is a general technique for constructing approximate solutions to differential equations[2]. FEM was first developed to solve potential problem in 1940s and then applied to electromagnetics in the late 1960s. Although the FEM is more complicated to implement, it can deal with more complex boundary and has good computational reliability. FEM is different from FDM mainly in its approximation. In FDM, we approximate both first and second derivatives in $\hat{x}$, $\hat{y}$ and $\hat{z}$ directions, which means we can only implement it on rectangular grids. In FEM, whose derivation can be carried out from either weighted residual method or variational formulation, derivatives are carried out directly but simple approximate solutions are used. We can carry out derivatives on the approximate solution. Since approximate solution can always be constructed on arbitrary shape, FEM method can be constructed to model arbitrary geometry very accurately, which makes FEM powerful and versatile.

From the FEM perspective, the procedure for FE-DDM can be abstracted as follows, these are routines in numerical method society.

1) Dividing and Meshing: Decompose the total solution domain and separately discretize the the subdomains into small elements associated with basis functions and coefficients.
2) Assembly: Use Galerkin's method to formulate linear algebraic equations and evaluate them either using analytical integration or numerical integration. And then assemble these components into finite element matrix. the calculation of local finite element matrix ($\mathbb{R}^{3\times3}$ for first order triangular element) can be done in parallel once we have the element-to-node connectivity list and the node list.
3) Solving: Solve the assembled sparse linear system and obtain the unknowns. Many serial linear sparse solver has their parallel counterparts.

B. DDM

High performance computing (HPC) technologies provide faster and larger electromagnetic simulations with modern ever-growing HPC hardware. Two of the major goals commercial CEM or multi-physics softwares [3][4] pursue are speed-up and scale-up. The former can be achieved by using multiprocessing, distributed memory matrix (DMM) solving, spectral decomposition method, s-parameter only matrix solve, while the latter is by using DDM.

DDM is a substructuring technique that involves the division of a finite element mesh into a number of sub-domains which only interact at their common interfaces and shared crosspoints. While the clock speed of CPUs stops increasing due to the heat dissipation issue, FE-DDM can explore the nowadays parallel multiprocessing computing power due to the significant amounts of independent operations inherited in both FEM and DDM.

DDM was proposed by the German mathematician H.A.Schwarz in 1870 [5] and was first used for solving elliptic partial differential equations in the 1950s. In essence, DDM itself as a mathematical idea, it must be combined with other existing numerical methods in specific application field. As is well known, DDM is quite successful within the FEM and FDM for reducing both computation time and the memory requirements on processors of modern computers.

There are two major paradigms used to solve the BVP of numerical PDE by domain decomposition. Schwarz methods [6], also called overlapping domain decomposition methods as opposed to non-overlapping Schur methods, treat the unknown coefficients on the boundaries of each subdomain that reside in other subdomains as inhomogeneous Dirichlet boundary conditions in each iteration. This alternating pattern of iterations continues from a starting guess to the convergence of solution on the total domain.

Alternating Schwarz method, a.k.a. the multiplicative Schwarz method, whose iterating process is in a serial manner, cannot be fully parallelized when the number of subdomains is less than the number of processing element (PE, a thread or a process) used in the parallel machine [1]. This is due to the fact that those overlapping subdomains must be solved sequentially and only the subdomains that do not have any overlap can be executed in parallel. Additive Schwarz method, on the other hand, updates the Dirichlet boundaries simply based on the results of the previous iteration at the price of sacrificing performance. Hybrid version that combines these two methods can be used in practical applications.

DDM is often carried out on large multi-core and multi-node parallel machines for the scaling-up purpose. Because currently we only use Matlab to implement the DDM algorithms on personal computer in this course project, we here briefly describe how to use the MPI-OpenMP multi-level parallelism for inter-node and intra-node parallelization of the FE-DDM in the real application from programming perspective. MPI is used to solve inter-node multi-processing parallelism, and OpenMP is responsible for intra-node multi-threading parallelism. MPI is at the top level (coarse-grained); the solution domain is divided into small subdomains, which are assigned to each node. The nodes communicate and synchronize with each other through message passing. OpenMP is at the bottom level (fine-grained); thread-level parallelism deals with the computational tasks assigned to each node and assigns them to each processor, and processors within the node communicate and synchronize through shared memory and cache coherence. Here by coarse-grained, we mean that a significant portion of computations can be performed in parallel during the interval of communication, and vice-versa.

From the DDM perspective, the parallelism in FE-DDM can be explored as follows.

1) the finite element discretization (meshing) can be generated simultaneously for each subdomain when using

non-conformal domain decomposition techniques [7][8]. This is a geometrical problem related to the definition of geometry and material properties and discretization level (h-refinement).
2) the finite element sub-matrices for each of the subdomains can be solved in parallel and the solution for the total domain will be generated after we get the solution for each subdomain.
3) the interpolating procedure (using linear basis functions defined at the nodes of triangular elements and the corresponding coefficients) for visualization (a.k.a. post-processing stage). The interpolation for the overlapped region can be parallelized either by assigning to different processors that are responsible for different subdomains or using atomic operations provided by C++ 11.

C. Iterative linear system solvers

Iterative solvers for linear system begin with initial guess for solution and successively improve it until the accuracy of the solution meets the requirement or the number of iteration exceed some prescribed limitations. Such methods differ from direct methods in that an exact solution is not being found. Although direct solvers are more robust than and thus preferred to the iterative solvers in some scenarios, the memory and computational requirements for the high-dimensional electromagnetic problems with many degrees of freedom per node and element may cause serious challenges to the direct solvers available these days. In addition, iterative solvers are especially useful compared with direct solvers when system matrix is sparse. In real world applications, the square matrix $\boldsymbol{K}$ formed by the FEM for electromagnetic problems is extremely sparse because only the basis functions that are defined at the same element will give non-zero entries in the finite element matrix.

Classical iterative solvers includes Jacobi method, Gauss-Seidel method, Successive over-relaxation (SOR) method and Conjugated Gradient (CG) method. If the coefficients used in the updating equation do not depend on the current iteration, then the method is called stationary methods. Otherwise, they are called non-stationary method or Krylov sub-space method. If we write the updating equations explicitly, then we can easily find that the Jacobi method and the Gauss-Seidel method are stationary methods while the SOR method and the CG method belong to Krylov sub-space method.

A good preconditioner plays an essential role in the success of any iterative methods used to solve the linear system and there are many techniques for generating a preconditioner $\boldsymbol{M}$ that you can choose between depending on the properties of the matrix to be preconditioned. In this course project, we will choose Additive Schwarz Method with incomplete LU (ILU) factorization as preconditioner for the sparse matrix $\boldsymbol{K}$ generated by FEM. The problem with LU factorization is that even though the matrix $\boldsymbol{K}$ is sparse, its LU factorization may result in non-sparse triangular matrices, which means it requires more computations in order to solve the sparse linear system. In this case, we can use ILU factorization to approximate the LU factorization by setting the values that are trivial in the upper and lower triangular matrices to zero and thus forcing the sparsity in the result of factorization.

From the perspective of linear system solver, the parallelism in FE-DDM can be explored by utilizing the parallel iterative solvers. There exists many parallel iterative solvers. In addition to the parallel conjugate gradient (parallel CG) method which is the most widely studied and used, classical ones include parallel Jacobi method, parallel Gauss-Seidel method and parallel successive over-relaxation method. The iterative process can be preconditioned by an operator which typically incorporates the solutions of the subproblems [9]. Thus the combination of DDM and CG provide us with an iterative method for the solution of a linear finite element system, suitable for a parallel implementation.

II. Formulation

In this section, we are going to formulate the problems and methods used in this FEM project. First, we carry out some derivations to demonstrate the FEM formulation of the two-dimensional electrostatic problem. Then, we discretize the entire solution domain into conformal unstructured mesh and divide the total domain into four subdomains. Finally, we briefly introduce the numerical algorithms we use for solving the linear system of the FEM and FE-DDMs. All units of length used in this section is in meter.

A. Problem of Interest and FEM formulation

In this project, we choose to solve an electrostatic problem with finite element method (domain decomposition). As illustrated in Fig. 1, the problem is a two-dimensional square region whose side length is 10 ranging from $[-5, 5]$ in both $\hat{x}$ and $\hat{y}$ directions. The outer boundary of this 10×10 region is connected to the ground, that is to say, the value of electric potential is zero at the boundary. There are four objects of shape circle, ellipse, square and triangle that are also connected to the ground located respectively in the SW, SE, NE and NW sub-region. A brief outline of the FEM developed for this specific toy electrostatic problem is described below with special reference to [10].

According to the Maxwell's equation, the electric field generated by the charge satisfies the Faraday's Law and Gauss' Law:

$$\nabla \times \boldsymbol{E} = 0$$

$$\nabla \cdot \varepsilon \boldsymbol{E} = \rho_e$$

In our electrostatic problem, the relationship between the electric field $\boldsymbol{E}$ and the electric scalar potential Φ derived from the vector identity $\nabla \times \nabla\Phi = 0$ can be expressed as Equ. (1).

$$\boldsymbol{E} = -\nabla\Phi \tag{1}$$

Substituting Equ. (1) into the Gauss' Law yields a general form of Poisson's equation that describes our electrostatic problem expressed as Equ. (2), which allows the material property to vary with position (in the Cartesian coordinate) and permits sources to exist,

$$-\nabla \cdot (\varepsilon(x, y)\nabla\Phi) = \rho_e(x, y) \tag{2}$$

TABLE I: The color-to-subdomain table used for the subdomain colors of the DDM illustration in Fig. 1 and the text colors of the artificial boundary illustration in Fig. 2.

	Subdomain 1	Subdomain 2	Subdomain 3	Subdomain 4
Color	light blue	purple	green	red

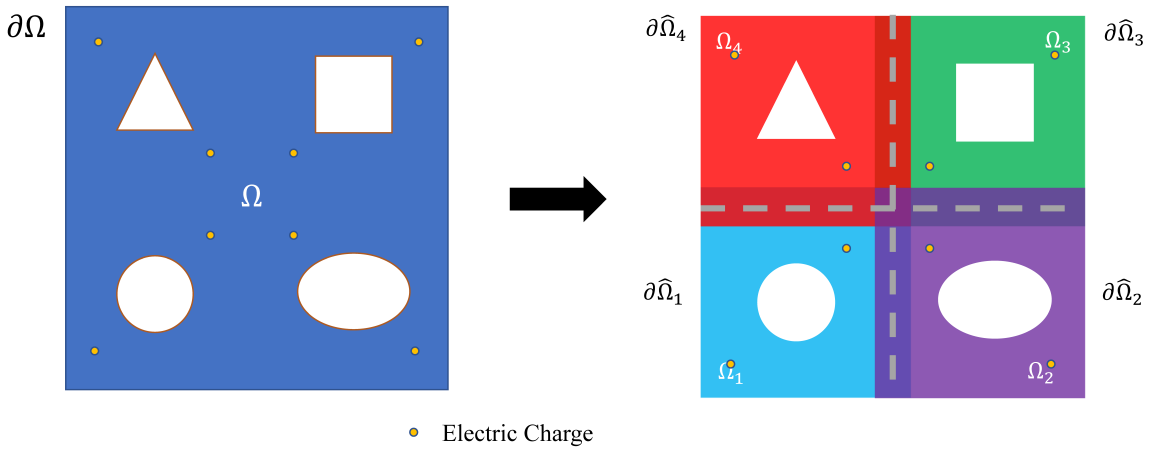


Fig. 1: Schematic of DDM. Ω denotes the total domain on which the final solution $\Phi(x, y)$ will be interpolated. $\partial\Omega$ denotes the boundary that isolates the total solution domain from the infinitely large space. $\partial\Omega$ will be divided to $\partial\hat{\Omega}_1, \ldots, \partial\hat{\Omega}_4$ with overlap after domain decomposition as shown on the right. Yellow dots represents the location where the electric charge with unit charge density is placed in the electrostatic problem. White halo regions with specific shapes represent the different objects connected to the ground (with a fixed electric potential of zero) in different location of the total domain. They will be inside different subdomains $\Omega_1, \ldots, \Omega_4$ after domain decomposition as shown on the right. Different colors represent different subdomains, this can be seen both from the figure and the Table I.

If we use Φ_D to denote the specified value of the potential on the Dirichlet boundary Γ_D we have the following boundary condition in our problem setting.

$$\Phi_D = 0$$

To transform the above PDE of BVP from continuous domain to the discrete domain based on the piecewise basis functions defined at the nodes, we can use both weighted residual method and weak formulation method. Weighted residual methods are directly applied to the governing PDE of the system whereas the weak form of the original PDE is developed by the weighted integral statement in the weak formulation method. For simplicity, we will only use the weighted residual method to obtain the system equations in this course project.

First, we use the testing function to multiply with the original PDE and integrate the weighted PDE over the entire solution to get the weighted average expression. Then we use the vector identity to transfer one derivative of the second-order derivative on Φ to the testing function ω_i and use Gauss' theorem to transform the area integral on Ω to boundary integral on Γ_D. Among the weighted residual methods, the Galerkin's method uses the same set of functions as both the basis functions and the testing functions. If we expand Φ as

$$\Phi = \sum_{j=1}^{N} \phi_j N_j$$

in terms of linear basis function N_j defined at nodes $j = 1, \ldots, N$, the testing function ω_i is also N_i with $i = 1, \ldots, N$ associated to each node. Especially, to enforce the Dirichlet boundary condition with prescribed coefficients ϕ_j^D, we need to know the global node index on which the boundary value resides. Thus we mark the basis functions defined at those node as N_j^D with $j = 1, \ldots, N_D$ where N_D is the total number of nodes with prescribed coefficient. Here we use coefficient ϕ as expression in the digital world to distinguish from Φ in the analog world. The procedures described above can be abstracted as Equ. (3), (4), (5), (6), (7) and (8), which yields the linear system in Equ. (9).

As for the weak formulation method which is more abstract in terms of understanding, we will only briefly introduce that as follows. The original PDE (differential equation) can be stated as the weak formulation (integral equation), minimizing which yields the best solution to the original PDE of BVP. In particular, when dealing with our elliptic PDE and the associated boundary value, we are looking for a solution in the space of all twice-differentiable functions which satisfy the BVP, i.e. both the PDE and the Dirichlet and Neumann boundary condition. However, when dealing with the weak formulation, we are seeking a function in the space of all once-differentiable functions which minimizes variational statement and satisfies the Dirichlet boundary condition. Since minimizing the latter is equivalent to solving the former, the order of derivatives

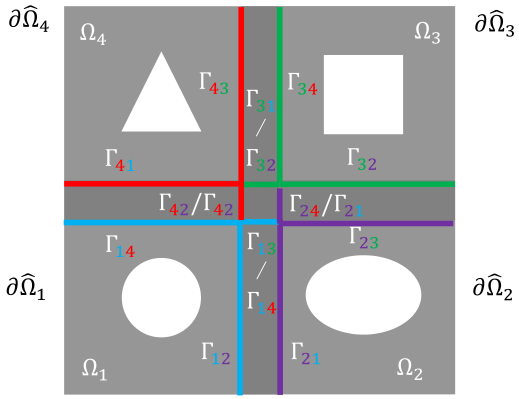


Fig. 2: Illustration of the artificial boundaries that reside in other subdomains for each of the four overlapped subdomains shown in Fig. 1. The respective colour indicates which subdomain the artificial boundary resides. The processor to which this subdomain belongs is the sender of the message at each iteration of the ASM if we use MPI in the distributed memory system.

of the related functions are reduced. Thus we can lessen the burden on the numerical algorithm in evaluating derivatives as well as reducing the smoothness requirement of the input data for higher-order differentiation.

$$-\int_{\Omega}\omega_i\left[\nabla\cdot(\varepsilon(x,y)\nabla\Phi)\right]d\Omega=\int_{\Omega}\omega_i\rho_e(x,y)d\Omega \tag{3}$$

$$\int_{\Omega}\varepsilon(x,y)\nabla\omega_i\cdot\nabla\Phi d\Omega=\int_{\Omega}\omega_i\rho_e(x,y)d\Omega+\oint_{\Gamma_D}\hat{n}\cdot(\varepsilon(x,y)\nabla\Phi)\omega_i d\Gamma_D \tag{4}$$

$$\sum_{j=1}^{N}\phi_j\int_{\Omega}\varepsilon(x,y)\nabla N_i\cdot\nabla N_j d\Omega=\int_{\Omega}N_i\rho_e(x,y)d\Omega-\sum_{j=1}^{N_D}\phi_j^D\int_{\Omega}\varepsilon(x,y)\nabla N_i\cdot\nabla N_j^D d\Omega \tag{5}$$

$$\sum_{j=1}^{N}K_{ij}\phi_j=b_i,\qquad i=1,\ldots,N \tag{6}$$

$$K_{ij}=\begin{cases}1 & \text{if}\quad i=j\quad\text{and node}\quad j\text{ is on }\Gamma_D\\ \int_{\Omega}\varepsilon(x,y)\nabla N_i\cdot\nabla N_j d\Omega & \text{if}\quad \phi_j\text{ is unknown}\end{cases} \tag{7}$$

$$b_i=\begin{cases}\int_{\Omega}N_i\rho_e(x,y)d\Omega-\sum_{j=1}^{N_D}\phi_j^D\int_{\Omega}\varepsilon(x,y)\nabla N_i\cdot\nabla N_j^D d\Omega & \text{if }\phi_i\text{is unknown}\\ \phi_i^D-\sum_{j=1,j\neq i}^{N_D}\phi_j^D\int_{\Omega}\varepsilon(x,y)\nabla N_i\cdot\nabla N_j^D d\Omega & \text{if }\phi_i\text{is prescribed (the potential of ground is zero)}\end{cases} \tag{8}$$

After assembly procedure, we get our linear system as shown in Equ. (9) to solve for the coefficient vector $\boldsymbol{\phi}=[\phi_1,\ldots,\phi_N]^T$, where N is the total number of nodes in the entire domain.

$$\boldsymbol{K}\boldsymbol{\phi}=\boldsymbol{b} \tag{9}$$

$\boldsymbol{K}$ is in $\mathbb{R}^{N,N}$ where N is the total number of basis functions, i.e. nodes in the discretized computational domain Ω.

When using the FEM to solve BVP of PDEs, the LHS matrix is called finite element matrix or stiffness matrix, which contains the information about geometry and basis functions. The RHS vector contains information about the imposed source and enforced Dirichlet boundary conditions. Homogeneous Neumann boundary condition is called natural boundary condition for functional because it is already satisfied in our derivation. If we also have natural boundary condition in our problem to be solved (which is not the case in this course project), we don't need to do anything and just

ignore them, because it will be automatically satisfied at the boundary whereas inhomogenous Neumann BC requires us to calculate its contribution to the right hand vector.

The solution on the entire domain of the original problem can be further calculated by interpolating the potential $\Phi(x, y)$ in each element using the coefficients at the nodes resulting from solving the system equations. The final unknown solution $\Phi(x, y)$ in the entire domain ($x \in [-5, 5]$ and $y \in [-5, 5]$) can be expressed as

$$\Phi(x, y) = \sum_{j=1}^{N} N_j(x, y)\phi_j$$

where the basis functions $N_j(x, y)$ defined at nodes are used as interpolating function in each element, i.e. if we loop through all the element to do the final interpolation of the result, the coefficients ϕ_j will be used for multiple times in all the elements that are directly connected to the associated node.

What needs to be mentioned is that some of the nodal coefficients $\phi_j, \quad j = 1, \ldots, N$ have prescribed values derived from the Dirichlet Boundary conditions while some of them are obtained by solving Equ. (9). The nodes with a prescribed value of zero in this project are on the boundary of the total domain $\partial\Omega$ and the boundary of the objects inside.

B. Discretization and Domain Decomposition

Discretizing the continuous solution domain into small elements and decomposing the total domain into smaller subdomains are two major concerns in the geometry aspect of FE-DDM.

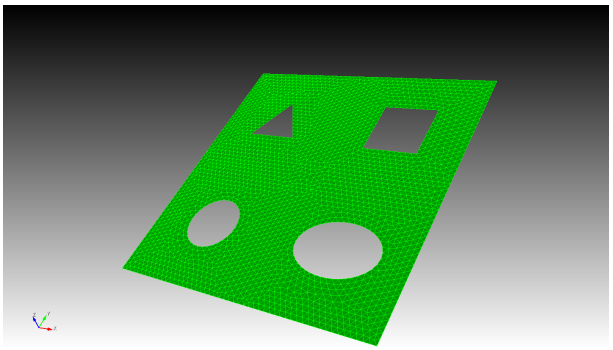


Fig. 3: Discretization of the solution domain using triangular elements visualized using software Cubit.

1) Meshing: Any polygon, no matter how irregular, can be represented exactly as a union of triangles [11]. Also, because we are dealing with scalar potential field generated by electric charge, it is reasonable to employ triangle element with linear basis function defined at the three nodes in our two-dimensional finite element electrostatic problem.

We choose to use unstructured mesh with general connectivity (GCON) in this project to discretize our solution domain, therefore the connectivity of elements must be defined and stored. The geometry is generated and divided using the Partial Differential Equation Toolbox provided in Matlab 2020b and visualized using software Cubit[12] as shown in Fig. 3. This procedure is usually called pre-processing in the finite element analysis and is aimed for constructing the geometric model in the discrete domain and defining attributes such as the material properties.

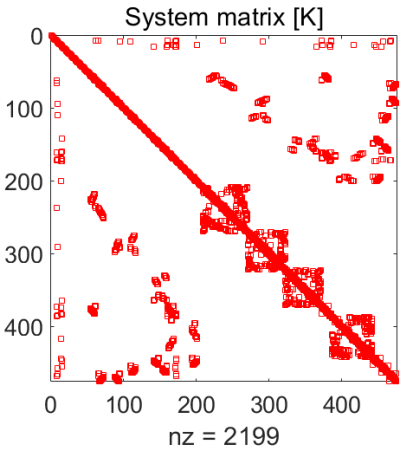


Fig. 4: The finite element matrix for the electrostatic problem in this project.

2) Domain Decomposition: The domain decomposition method, as illustrated in Fig. 1, is a method to decompose the solution domain into smaller subdomains. We can solve the subdomains separately and then the solution of the total domain is obtained through the boundary continuity condition such as first order transmission condition (FOTC) and second order transmission condition (SOTC) of adjacent subdomains when using non-overlapping DDMs. In this course project, we choose to implement a overlapping Schwarz DDM with a total subdomain number $M = 4$. As can be seen from Fig. 1 and Fig. 2, each subdomain is overlapped with the other three subdomains and all the four domain are overlapped in the square region at the center of the total domain besides the four rectangular overlapping bands formed by the overlapping of two adjacent subdomains. Based on the underlying divide-and-conquer philosophy, it divides the total domain Ω with boundary $\partial\Omega = \bigcup_{m=1}^{M} \partial\hat{\Omega}_i$ into 4 overlapped subdomains $\Omega = \bigcup_{m=1}^{4} \Omega_m, \quad \Omega_m \bigcap_{m \neq n} \Omega_n \neq \emptyset$. All of the four subdomains are at the boundary of the total domain.

The various factors that may influence the choice of an overlapping decomposition of the domain are the geometry of the domain, regularity of the solution, availability of fast solvers for subdomain problems and heterogeneity in the coefficients. When choosing the scheme for dividing subdomains, the solving speed of each subdomain should be equal in order to improve the parallel efficiency on the parallel machine. Thus, it is necessary to consider the balance of load, that is to say, each processor should deal with the problem as synchronously as possible. In real world applications, we can use the graph partitioning algorithms so that the decomposition yields approximately balanced loads. Also, in real projects written by C++, the size of subdomains includes the size of overlapping region should be determined by the available memory on each processor.

Unlike the column decomposition used in the literature

that proposed Integral Equation based overlapped DDM [13], the subdomains are evenly divided to square chunks in our problem setting, as illustrated by the dashed grey lines in Fig. 1. Theoretical analysis [14][15] shows that the rate of convergence increases as the overlap between the subdomains increases. Numerical experiments have shown that Schwarz methods perform well even if the overlap between neighboring subregions is quite small [15]. Thus we enlarge each of the subdomain by 0.5 and set the width of overlap region to be 1 in our toy DDM. In this way, the total domain Ω is divided into subdomains $\Omega_1, \ldots, \Omega_4$ where we not only have overlapped region shared by no more than two subdomains but have overlapped crosspoint shared by all the four subdomains. In sum, we choose to assign the four subdomains with the same physical size of 5.5×5.5 and thus similar number of nodes with unknown coefficients in each subdomain because efficiency is maximized when load imbalance is minimized.

The original linear system, as visualized in Fig. 4, is replaced by as many smaller systems (visualized in Fig. 5) as the number of subdomains in our overlapping DDM. If the unknowns are properly sequenced, the resulting finite element matrix for the total domain have a block structure consisting of several independent blocks each corresponding to a subdomain together with some interface blocks corresponding to the connection between subdomains.

C. Iterative solver

One key issue of FE-DDM is still solving a fully coupled FEM matrix. Because the DDM produces no compromised solution, FE-DDM is a true fully EM coupled, full-wave solving technique. Because our finite element stiffness matrix $\boldsymbol{K}$ is a symmetric positive definite (SPD) matrix, we choose the famous Conjugated Gradient method as the iterative solver for our large linear sparse system.

1) CG used in FEM-CG: CG is a kind of Krylov subspace method and is the most popular iterative method for solving large systems of linear equations [16]. CG requires the matrix in the linear system to be SPD. However, because our electrostatic problem is based on the Poisson's equation, which is an elliptic PDE, as well as the mesh we use to discretize the BVP is conforming, the finite element matrix $\boldsymbol{K}$ will be inheritly SPD [17] in this course project.

As can be seen from its name, CG is based on gradient descent (Netwon's method). CG iterative process starts from an initial guess $\boldsymbol{\phi}_0$ of the solution $\boldsymbol{\phi}$ and an initial descent direction

$$\boldsymbol{p}_0 = \boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{K}\boldsymbol{\phi}_0$$

, and then iterate in a following manner

$$\boldsymbol{\phi}_{k+1} = \boldsymbol{\phi}_k + \alpha_k \boldsymbol{p}_k$$

$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{K}\boldsymbol{p}_k$$

$$\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$$

where α_k and β_k are chosen as Equ. (10) and Equ. (11) respectively such that they minimize the norm of the error $\|e_{k+1}\|_{\boldsymbol{K}}^2 = \|\boldsymbol{\phi}_{k+1} - \boldsymbol{\phi}_k\|_{\boldsymbol{K}}^2$ at the k-th iteration. Here $\|\boldsymbol{x}\|_{\boldsymbol{K}}^2$ is defined as the inner product of $\boldsymbol{K}\boldsymbol{x}$ and $\boldsymbol{x}$.

$$\alpha_k = \frac{\|\boldsymbol{r}_k\|_2^2}{\|\boldsymbol{p}_k\|_{\boldsymbol{K}}^2} \tag{10}$$

$$\beta_k = \frac{\|\boldsymbol{r}_{k+1}\|_2^2}{\|\boldsymbol{r}_k\|_2^2} \tag{11}$$

CG will converge in at most N iterations with N being the size of the finite element matrix $\boldsymbol{K}$ according to the minimum error property [18]. Error is reduced at each iteration by a factor in terms of the condition number of $\boldsymbol{K}$, thus the convergence tends to be fast if the finite element matrix is well-conditioned and can be arbitrarily slow if the finite element matrix is ill-conditioned. So based on this property of CG, we can use preconditioning techniques such as incomplete LU factorization (ILU)[19] and symmetric successive over-relaxation (SSOR)[20] to accelerate the convergence rate.

2) PCG used in FE-DDM-PCG: Preconditioned conjugate gradient method is well established for solving linear systems of equations that arise from the discretization of partial differential equations using FEM. If CG is employed to solve the linear system, a SPD preconditioner $\boldsymbol{M}$ would be necessary in order to have a faster convergence in most cases.

For our discretized linear system $\boldsymbol{K}\boldsymbol{\phi} = \boldsymbol{b}$, $\boldsymbol{\phi}$ can be regarded as the limit as $k \to \infty$ of the iterative recursion

$$\boldsymbol{\phi}^{k+1} = \boldsymbol{\phi}^k + \left[\sum_{m=1}^{4} \boldsymbol{R}_m^T \boldsymbol{K}_m^{-1} \boldsymbol{R}_m\right](\boldsymbol{b} - \boldsymbol{K}\boldsymbol{\phi}_k)$$

, where $\boldsymbol{R}_m, \quad m = 1, \ldots, 4$ is the restriction matrix in $\mathbb{R}^{N_m, N}$ from Ω to Ω_m and $\boldsymbol{R}_m^T$ is the extension matrix in $\mathbb{R}^{N, N_m}$ from Ω_m to Ω such that the principle submatrices of $\boldsymbol{K}$ of order N_m corresponding to the 4 subdomains can be given by

$$\boldsymbol{K}_m = \boldsymbol{R}_m \boldsymbol{K} \boldsymbol{R}_m^T$$

as visualized in Fig. 5. To be more specific, $\boldsymbol{R}_m$ is the identity sub-matrix (ISM) of size $N_m \times N$ whose diagonal elements are set to one if the corresponding node belongs to Ω_m and to zero otherwise.

And the term $\boldsymbol{M}_{as} = \sum_{m=1}^{4} \boldsymbol{R}_m^T \boldsymbol{K}_m^{-1} \boldsymbol{R}_m$ results from the ASM can work as a preconditioner known as additive Schwarz preconditioner for some type of iterative method whose convergence property is analyzed in [21]. Obviously, in the particular case where there is no overlapping, i.e. when the $\Omega_m, \quad m = 1, \ldots, 4$ form an actual partition of Ω, then the additive Schwarz preconditioner is nothing but a block Jacobi preconditioner [22]. Thus this preconditioner can be applied generally to other polynomial-based method (PBM) such as Richardson's method and GMRES.

The incomplete LU factorization [23] of the finite element matrix and its sub-matrices are a widely used class of preconditioners for sparse matrix [24]. However, for large sparse matrices, the best preconditioner is unlikely to be an ILU factorization of the entire matrix. Instead, a domain decomposition is preferred, using ILU as preconditioning method in the subdomain solver.

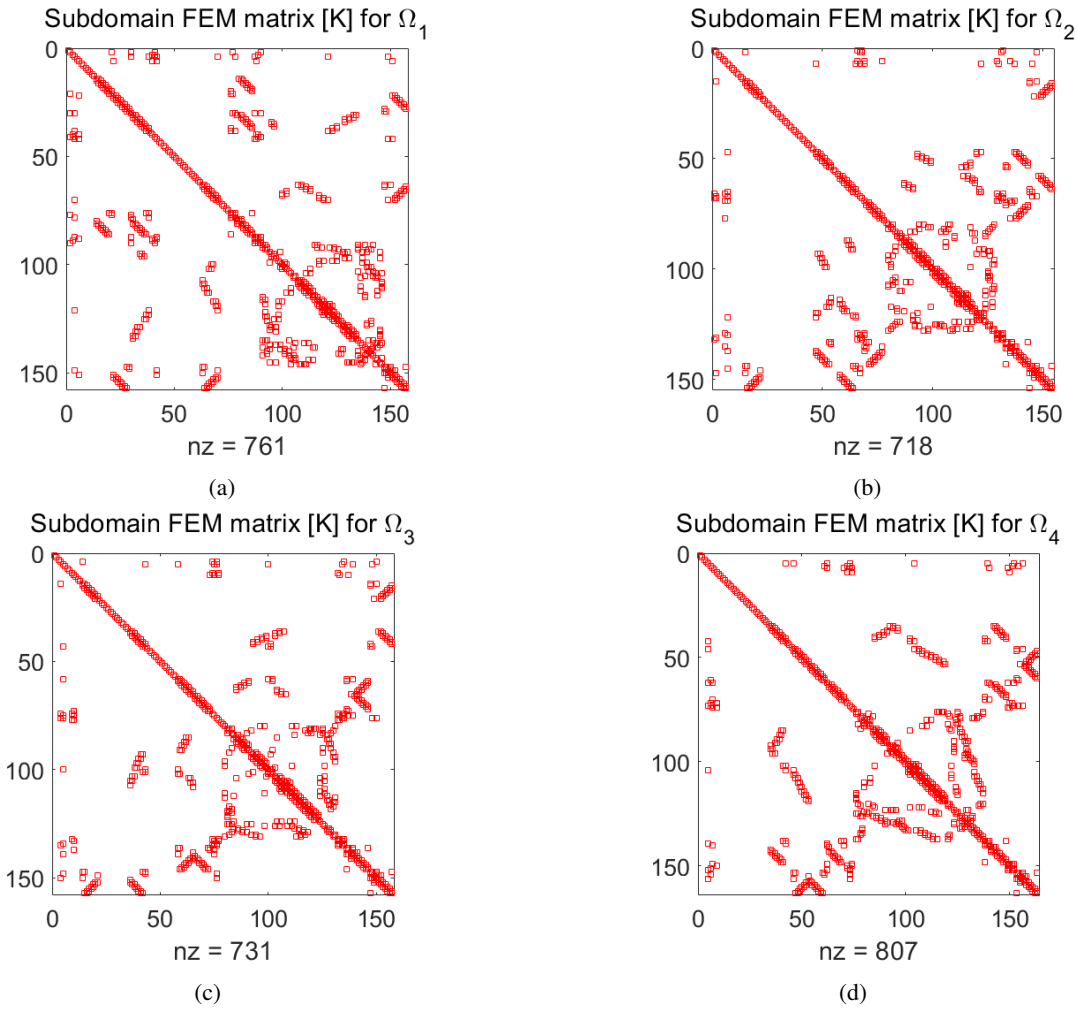


Fig. 5: The finite element sub-matrices of the four subdomains. (a) Ω_1. (b) Ω_2. (c) Ω_3.(d) Ω_4.

To get the ILU factorization of the original matrix, we can simply perform the standard Gaussian Elimination first to get the lower and upper triangular matrices $\boldsymbol{L}$ and $\boldsymbol{U}$ and set the values in $\boldsymbol{L}$ and $\boldsymbol{U}$ below a threshold of ϵ to be zero. Or we can decide which entries to be zero according to the sparsity pattern of $\boldsymbol{K}$. If $\boldsymbol{L}$ and $\boldsymbol{U}$ have the same sparsity pattern as the original matrix $\boldsymbol{K}$, i.e. $\boldsymbol{L}+\boldsymbol{U}$ has the same sparsity pattern as $\boldsymbol{K}$ and $(\boldsymbol{LU})_{ij} = \boldsymbol{K}_{ij}$ for each pair of (i, j) that belongs to the nonzero entry of $\boldsymbol{K}$, the resulting preconditioner is called ILU(0).

Applying the ILU factorization with pivoting (for stability reasons) on the finite element matrix, we have

$$\boldsymbol{PK} \approx \boldsymbol{LU}, \quad \boldsymbol{K}\phi = \boldsymbol{b} \Longrightarrow \phi = \boldsymbol{K}^{-1}\boldsymbol{b} = \boldsymbol{U}^{-1}\boldsymbol{L}^{-1}\boldsymbol{Pb}$$

Thus we can find a preconditioner $\boldsymbol{M} = \boldsymbol{M}_{as}$ as the approximate inverse of $\boldsymbol{K}$ by ILU factorization to make the condition number of $\boldsymbol{MK}$ small and create an altered system

$$\boldsymbol{MK}\phi = \boldsymbol{Mb}$$

where the preconditioner $\boldsymbol{M}$ in the ILU factorization case is

$$\boldsymbol{M} = \sum_{m=1}^{4} \boldsymbol{R}_m^T \boldsymbol{U}_m^{-1} \boldsymbol{L}_m^{-1} \boldsymbol{P}_m \boldsymbol{R}_m$$

with $\boldsymbol{P}_m \boldsymbol{K}_m \approx \boldsymbol{L}_m \boldsymbol{U}_m$ being the ILU factorization of the m-th sub-matrix $\boldsymbol{K}_m$. The resulting Preconditioned Conjugated Gradient method [25] is given in Algorithm 1.

3) ASM used in FE-DDM-ASM: The ASM was first developed in [26]. Here, we take the FE-DDM-ASM iterative process as a kind of iterative solver, the true solver behind this method should be simply the matrix-inverse and matrix-multiplication provided Matlab in this toy project.

To use ASM deal with the problem formulated in this project, we extend the formulation of the ASM described in [1] to the iterative process written as Equ. (12), (13) , (14) and (15) with schematic illustration shown in Fig. 1 and Fig. 2. The subscript of the notation of the artificial boundary Γ_{ij} represents the sender (to which subdomain i the nodes belong) and the receiver (of which subdomain j the nodes are used as the artificial Dirichlet boundary condition). Here, the Boolean vector $\boldsymbol{R}_{\Gamma_{ij}\Omega_j}$ extracts $\boldsymbol{\phi}_{\Gamma_i}$ from $\boldsymbol{\phi}_{\Omega_j}$, which can be implemented either by local indexing (where the artificial

boundary nodes locate in the local node array of each of the four subdomains) (set the use_parallel_for to be 'true' in the Matlab benchmarking script) or simply by using an indexing array (where the artificial boundary nodes locate in the global node set) to extract the $\boldsymbol{\phi}_{\Gamma_i}$ from the global array $\boldsymbol{\phi}$ (set the use_parallel_for to be 'false' in the Matlab benchmarking script).

$$\boldsymbol{K}_{\Omega_1\Omega_1}\boldsymbol{\phi}_{\Omega_1}^{(k+1)} = \boldsymbol{b}_{\Omega_1} - \left[\boldsymbol{K}_{\Omega_1\Gamma_{41}}\boldsymbol{R}_{\Gamma_{41}\Omega_4}\boldsymbol{\phi}_{\Omega_4}^{(k)} + \boldsymbol{K}_{\Omega_1\Gamma_{31}}\boldsymbol{R}_{\Gamma_{31}\Omega_3}\boldsymbol{\phi}_{\Omega_3}^{(k)} + \boldsymbol{K}_{\Omega_1\Gamma_{21}}\boldsymbol{R}_{\Gamma_{21}\Omega_2}\boldsymbol{\phi}_{\Omega_2}^{(k)}\right] \tag{12}$$

$$\boldsymbol{K}_{\Omega_2\Omega_2}\boldsymbol{\phi}_{\Omega_2}^{(k+1)} = \boldsymbol{b}_{\Omega_2} - \left[\boldsymbol{K}_{\Omega_2\Gamma_{42}}\boldsymbol{R}_{\Gamma_{42}\Omega_4}\boldsymbol{\phi}_{\Omega_4}^{(k)} + \boldsymbol{K}_{\Omega_2\Gamma_{32}}\boldsymbol{R}_{\Gamma_{32}\Omega_3}\boldsymbol{\phi}_{\Omega_3}^{(k)} + \boldsymbol{K}_{\Omega_2\Gamma_{12}}\boldsymbol{R}_{\Gamma_{12}\Omega_1}\boldsymbol{\phi}_{\Omega_1}^{(k)}\right] \tag{13}$$

$$\boldsymbol{K}_{\Omega_3\Omega_3}\boldsymbol{\phi}_{\Omega_3}^{(k+1)} = \boldsymbol{b}_{\Omega_3} - \left[\boldsymbol{K}_{\Omega_3\Gamma_{43}}\boldsymbol{R}_{\Gamma_{43}\Omega_4}\boldsymbol{\phi}_{\Omega_4}^{(k)} + \boldsymbol{K}_{\Omega_3\Gamma_{23}}\boldsymbol{R}_{\Gamma_{23}\Omega_2}\boldsymbol{\phi}_{\Omega_2}^{(k)} + \boldsymbol{K}_{\Omega_3\Gamma_{13}}\boldsymbol{R}_{\Gamma_{13}\Omega_1}\boldsymbol{\phi}_{\Omega_1}^{(k)}\right] \tag{14}$$

$$\boldsymbol{K}_{\Omega_4\Omega_4}\boldsymbol{\phi}_{\Omega_4}^{(k+1)} = \boldsymbol{b}_{\Omega_4} - \left[\boldsymbol{K}_{\Omega_4\Gamma_{34}}\boldsymbol{R}_{\Gamma_{34}\Omega_3}\boldsymbol{\phi}_{\Omega_3}^{(k)} + \boldsymbol{K}_{\Omega_4\Gamma_{24}}\boldsymbol{R}_{\Gamma_{24}\Omega_2}\boldsymbol{\phi}_{\Omega_2}^{(k)} + \boldsymbol{K}_{\Omega_4\Gamma_{14}}\boldsymbol{R}_{\Gamma_{13}\Omega_1}\boldsymbol{\phi}_{\Omega_1}^{(k)}\right] \tag{15}$$

Algorithm 1 Preconditioned CG algorithm

Initialize $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{K}\boldsymbol{\phi}_0$,$\boldsymbol{z}_0 = \boldsymbol{M}^{-1}\boldsymbol{r}_0$ and $\boldsymbol{p}_0 = \boldsymbol{z}_0$
for $k = 1, 2, \ldots$ **do**
  $\rho_k = \|\boldsymbol{r}_k, \boldsymbol{z}_k\|_2$
  $\boldsymbol{q}_k = \boldsymbol{K}\boldsymbol{p}_k$
  $\alpha_k = \frac{\rho_k}{\|\boldsymbol{p}_k, \boldsymbol{q}_k\|_2^2}$
  $\boldsymbol{\phi}_{k+1} = \boldsymbol{\phi}_k + \alpha_i \boldsymbol{p}_i$
  $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_i \boldsymbol{q}_i$
  $\boldsymbol{z}_{k+1} = \boldsymbol{M}^{-1}\boldsymbol{r}_{k+1}$
  $\rho_{k+1} = \|\boldsymbol{r}_{k+1}, \boldsymbol{z}_{k+1}\|_2^2$
  $\beta_{k+1} = \frac{\rho_{k+1}}{\rho_k}$
  $\boldsymbol{p}_{k+1} = \boldsymbol{z}_{k+1} + \beta_{k+1}\boldsymbol{p}_k$
  Check convergence status
end for

Although this FE-DDM-ASM algorithm that has both sequential version and parallel version is also implemented on Matlab for now, there are still some advantages over non-DDM version. The most obvious one is the saving in memory cost when the problem size is extremely large in that the computation for each subdomain can be done separately, i.e. only a single sub-problem needs to be stored in memory at any given time.

D. Parallel Preconditioned Conjugate Gradients

If time allows, this part will be implemented in the final C++ artifact.

III. Result

In this section, results generated by the Matlab codes attached are used to verify the theoretical analysis of classical FEM with CG solver, the FE-DDM using overlapped ASM and the FE-DDM using ASM as the preconditioner of the PCG that are formulated in Section II.

A. Interpolated Electric Potential

We first examine the correctness of the functions of these three methods by visualizing the interpolated results. By interpolation, we mean that we interpolate the $\boldsymbol{\Phi}$ with linear basis functions N_i and the associated coefficient ϕ_i by telling if a pixel falls in the elements connect to the node or not. In FEM-CG, the result of which is denoted as Φ_1 shown in Fig. 6 (a), the interpolation is performed on the entire domain. In FE-DDM-ASM and FE-DDM-PCG, the result of which is denoted as Φ_2 and Φ_3 shown respectively in Fig. 6 (b) and (c), we do the interpolation also in a DDM way, i.e. we search and interpolate separately on each subdomain rather than on the entire domain.

B. Difference between the generated result

If we do subtraction and take the absolute value between each two of the three results calculated by different methods, we find that they do not have much difference with each other from Fig 7. If we take the result generated by the classical FEM with CG solver as the ground truth, we can conclude that both of the two FE-DDMs can achieve good accuracy for this toy problem.

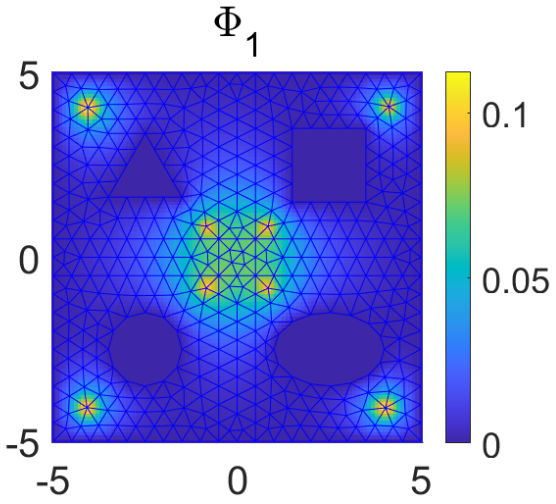
C. Convergence

The accuracy of the solutions can be measure by their residuals. As we can see from the convergence curves shown in Fig. 8, convergence rate of CG can be substantially accelerated by preconditioning. To be more specific,

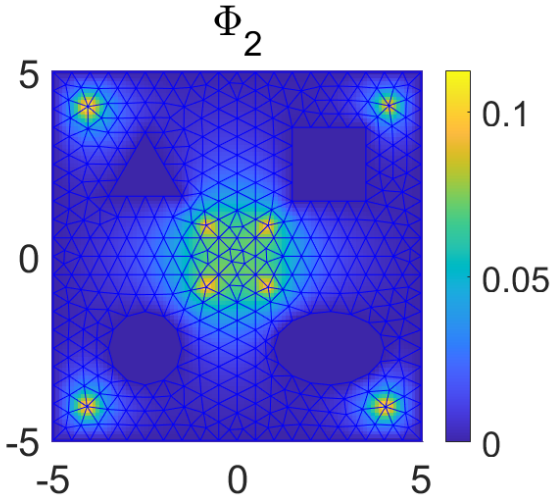
1) using ASM as the preconditioner for FE-DDM-PCG has the fastest convergence rate and highest accuracy for solving the linear system in this problem.
2) the convergence rate of the FE-DDM-ASM is faster than the traditional FEM-CG without DDM
3) FE-DDM-ASM experiences low accuracy due to obvious reasons.

IV. Conclusions and future work.

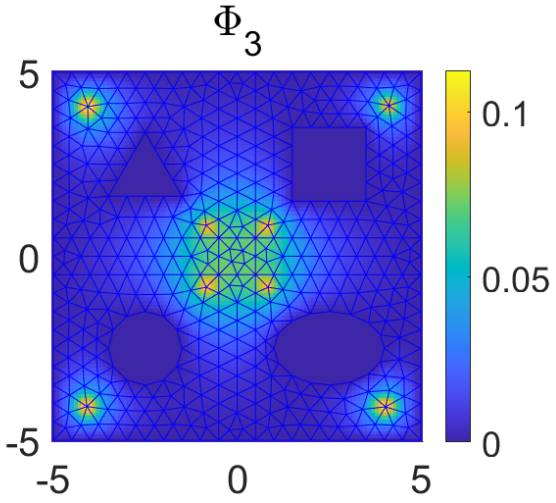
In this course project, three algorithms about FEM and DDM are developed to solve a two-dimensional electrostatic problem.



(a)

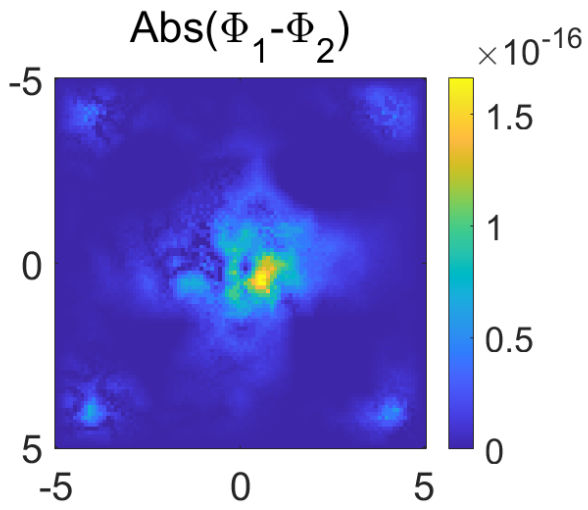


(b)

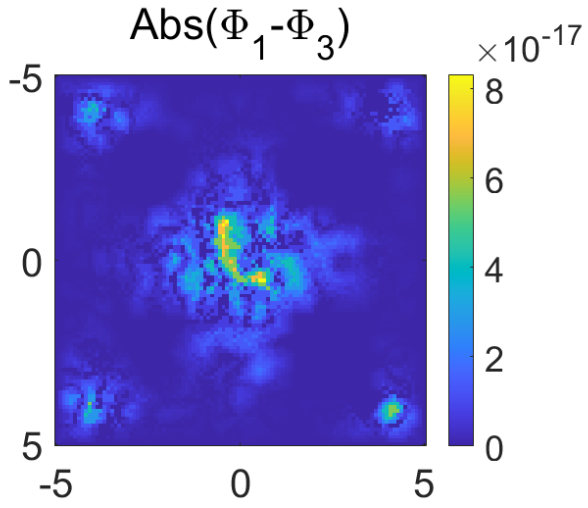


(c)

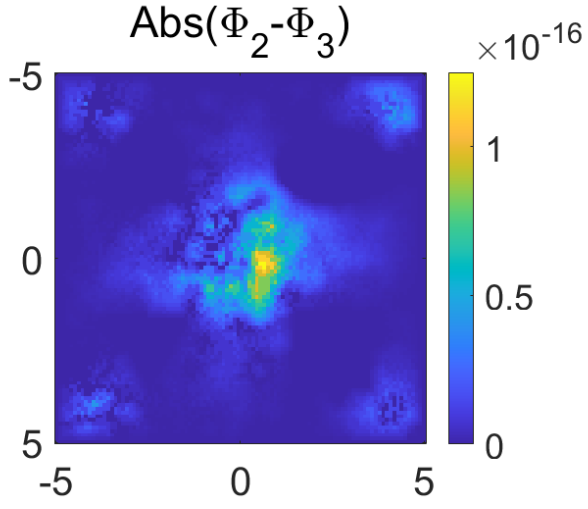
Fig. 6: The results of the same electric static problem produced by different methods. (a) traditional FEM using CG as linear system solver. (b) FE-DDM using ASM. (c) FE-DDM using PCG as linear system solver with Additive Schwarz preconditioner.



(a)



(b)



(c)

Fig. 7: The difference of the results of the same electric static problem produced by different methods. (a) the difference between traditional FEM using CG and FE-DDM using ASM. (b) the difference between traditional FEM using CG and FE-DDM using PCG with Additive Schwarz preconditioner. (c) the difference between FE-DDM using ASM and FE-DDM using PCG with Additive Schwarz preconditioner.

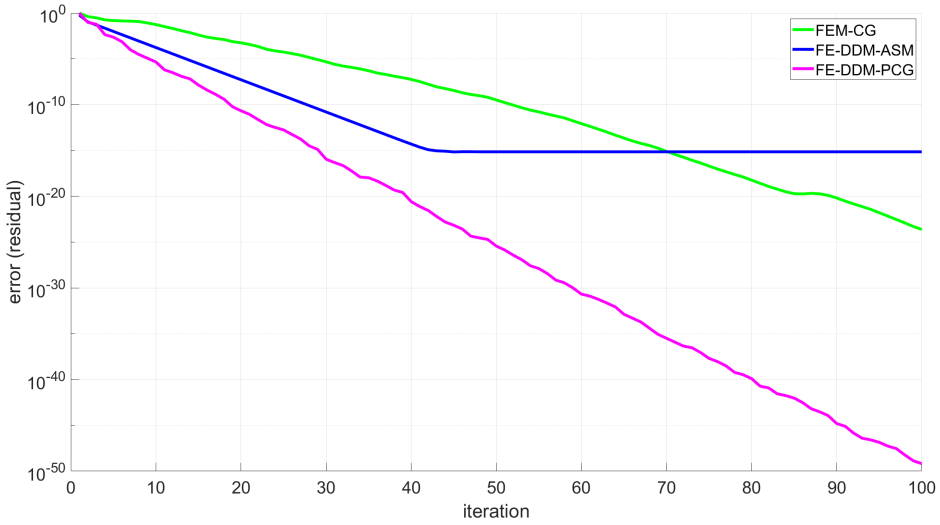


Fig. 8: The convergence of the three methods.

The first one is called FEM-CG, which uses conjugated gradient method to iteratively solve the linear system produced after the assembly procedure of FEM. This one is mainly used as the ground truth to verify the results of latter two algorithms. Also, the CG solver used in this algorithm is further extended to PCG in the third algorithm.

The second one is called FE-DDM-ASM, which uses Additive Schwarz Method with artificial Dirichlet boundary conditions to mimic the parallel DDM implementation with message passing on real distributed memory system.

The third one is called FE-DDM-PCG, which uses Additive Schwarz domain decomposition and ILU method as preconditioner of the PCG. This one gives the fastest convergence and highest accuracy among these three algorithms.

By the end of this semester, this course project will be further extend to a C++ version on Linux system and benchmarking on Campus Cluster, which supports multiple processors per node and cores (threads) per processor. Both the serial implementation of the FEM and parallel implements of the FE-DDM will be presented and delivered as the final artifact. Furthermore, we will write the code using MPI and OpenMP to implement the parallel version of the PCG method instead of using the well-known C++ packages such as MUMPS[27], SuperLU[28] and PETSc[29].

As for the prospective C++ program, the set of input is the discretized solution domain. And we can test the effectiveness of the FE-DDM algorithm by first

1) modifying the number of unknowns (problem size) to be solved in the program.
2) modifying the size of the overlapped region in the DDM problem to test the communication cost used in the distributed memory machine.

and then comparing the time consumption with traditional serial FEM without DDM .

ACKNOWLEDGMENT

THE student would like to thank Prof. Jin and Yanan Liu (the TA) for offering this computational EM course during the global pandemic.

REFERENCES

[1] J. Jin, *The Finite Element Method in Electromagnetics*, ser. Wiley - IEEE. Wiley, 2015. [Online]. Available: https://books.google.com.hk/books?id=DFi-BgAAQBAJ
[2] E. B. Becker, G. F. Carey, and J. T. Oden, "Finite elements, an introduction: Volume i." ., *258*, p. 1981, 1981.
[3] C. Multiphysics, "Introduction to comsol multiphysics®," *COMSOL Multiphysics, Burlington, MA, accessed Feb*, vol. 9, p. 2018, 1998.
[4] "Ansys hfss," ANSYS, Inc. 2021. [Online]. Available: https://www.ansys.com
[5] H. A. Schwarz, "II. Ueber einen Grenzübergang durch alternirendes Verfahren," Wolf J. XV. 272-286. 1870 (1870)., 1870.
[6] H. A. Schwarz, "Über ein die flächen kleinsten flächeninhalts betreffendes problem der variationsrechnung," in *Gesammelte Mathematische Abhandlungen*. Springer, 1890, pp. 223–269.
[7] Z. Peng and J.-F. Lee, "Non-conformal domain decomposition method with second-order transmission conditions for time-harmonic electromagnetics," *Journal of Computational Physics*, vol. 229, no. 16, pp. 5615–5629, 2010.
[8] Z. Peng, X.-c. Wang, and J.-F. Lee, "Integral equation based domain decomposition method for solving electromagnetic wave scattering from non-penetrable objects," *IEEE Transactions on Antennas and Propagation*, vol. 59, no. 9, pp. 3328–3338, 2011.
[9] X.-C. Cai and Y. Saad, "Overlapping domain decomposition algorithms for general sparse matrices," *Numerical linear algebra with applications*, vol. 3, no. 3, pp. 221–237, 1996.
[10] J.-M. Jin, *Theory and computation of electromagnetic fields*. John Wiley & Sons, 2011.
[11] P. P. Silvester and R. L. Ferrari, *Finite elements for electrical engineers*. Cambridge university press, 1996.
[12] T. D. Blacker, S. J. Owen, M. L. Staten, W. R. Quadros, B. Hanks, B. W. Clark, R. J. Meyers, C. Ernst, K. Merkley, R. Morris *et al.*, "Cubit geometry and mesh generation toolkit 15.2 user documentation," Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2016.
[13] W.-D. Li, W. Hong, and H.-X. Zhou, "Integral equation-based overlapped domain decomposition method for the analysis of electromagnetic scattering of 3d conducting objects," *Microwave and Optical Technology Letters*, vol. 49, no. 2, pp. 265–274, 2007.
[14] M. Dryja and O. B. Widlund, "Some domain decomposition algorithms for elliptic problems," in *Iterative methods for large linear systems*. Elsevier, 1990, pp. 273–291.

[15] ——, "Domain decomposition algorithms with small overlap," *SIAM Journal on Scientific Computing*, vol. 15, no. 3, pp. 604–620, 1994.
[16] J. R. Shewchuk *et al.*, "An introduction to the conjugate gradient method without the agonizing pain," 1994.
[17] A. Fumagalli, E. Keilegavlen, and S. Scialò, "Conforming, non-conforming and non-matching discretization couplings in discrete fracture network simulations," *Journal of Computational Physics*, vol. 376, pp. 694–712, 2019.
[18] A. Greenbaum, *Iterative methods for solving linear systems*. SIAM, 1997.
[19] E. Chow and A. Patel, "Fine-grained parallel incomplete lu factorization," *SIAM journal on Scientific Computing*, vol. 37, no. 2, pp. C169–C193, 2015.
[20] R.-S. Chen, E. K.-N. Yung, C. H. Chan, D. X. Wang *et al.*, "Application of the ssor preconditioned cg algorithm to the vector fem for 3d full-wave analysis of electromagnetic-field boundary-value problems," *IEEE Transactions on Microwave Theory and Techniques*, vol. 50, no. 4, pp. 1165–1172, 2002.
[21] T. F. Chan, T. P. Mathew *et al.*, "Domain decomposition algorithms," *Acta numerica*, vol. 3, no. 1, pp. 61–143, 1994.
[22] M. Hegland and P. E. Saylor, "Block jacobi preconditioning of the conjugate gradient method on a vector processor," *International journal of computer mathematics*, vol. 44, no. 1-4, pp. 71–89, 1992.
[23] T. Dupont, R. P. Kendall, and H. Rachford, Jr, "An approximate factorization procedure for solving self-adjoint elliptic difference equations," *SIAM Journal on Numerical Analysis*, vol. 5, no. 3, pp. 559–573, 1968.
[24] Y. Saad, *Iterative methods for sparse linear systems*. SIAM, 2003.
[25] V. Dolean, P. Jolivet, and F. Nataf, *An introduction to domain decomposition methods: algorithms, theory, and parallel implementation*. SIAM, 2015.
[26] S. V. N. A. M. Matsokin, "The schwarz alternation method in a subspace," *Soviet Math. (Iz. VUZ)*, vol. 29, pp. 78–84, 1985.
[27] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster, "Mumps: a general purpose distributed memory sparse solver," in *International Workshop on Applied Parallel Computing*. Springer, 2000, pp. 121–130.
[28] X. S. Li, "An overview of SuperLU: Algorithms, implementation, and user interface," vol. 31, no. 3, pp. 302–325, September 2005.
[29] E. Bueler, *PETSc for Partial Differential Equations: Numerical Solutions in C and Python*. SIAM, 2020.