Overlapping Domain Decomposition Finite Element Method (FE-DDM) of Poisson Equation

Junda Feng (jundaf2)

***Abstract*—High performance computing technologies provide faster and larger electromagnetic simulations with modern ever-growing HPC hardware. Two of the major goals commercial CEM or multi-physics softwares [1][2] pursue are speed-up and scale-up. The former can be achieved by using multiprocessing, while the latter is by using domain decomposition. In this course project, we choose to use FE-DDM to solve a toy electrostatic problem in a distributed memory machine - Campus Cluster. The DDM we choose to implement is Additive Schwarz Method based on MPI. The linear system solver we choose is Conjugated Gradient Method using OpenMP. A serial version for the project milestone and a parallel version for this final submission are implemented respectively. Results for both the physical simulation itself and the benchmarking study are shown and analysis in detail.**

***Index Terms*—High performance computing (HPC), Finite element method (FEM), Domain decomposition method (DDM), Additive Schwarz Method (ASM), Conjugated Gradient method.**

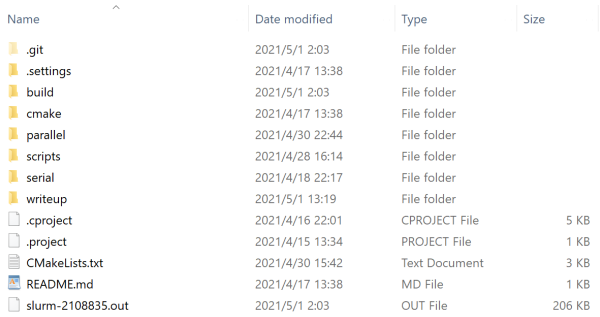
I. Overview of the Artifact and Introduction

To begin with, we briefly introduce what we have done in this course project. This is also an introduction to the C++ artifact. Because this is my first time in writing a HPC program from a very basic level of mathematical/physical equations to an executable and benchmarkable C++ artifact and from literature reading to the final submission, I can assure that all the codes in this course project are built from zero in this semester with no previous accumulation. The course staff can access the artifact using the Gitlab URL.

A. Work Done

In this course project, we did the following work.

- By reading literature, we surveyed which domain decomposition is a good practice for high performance computing using the hybrid parallel technique based on MPI and OpenMP.
- We wrote a Matlab artifact to verify the correctness and effectiveness of ASM and CGM before writing the final version of the project proposal.
- We wrote a serial C++ artifact based on the experience accumulated from the previous step.
- We wrote a parallel C++ artifact using MPI and OpenMP with MPI for DDM and OpenMP for CGM.
- We wrote two .slurm shell scripts for the benchmarking on the Campus Cluster.
- We wrote several .m Matlab scripts for generating the input geometrical data and processing the output data for the visualization of the calculated electric potential and benchmarking result.



Name	Date modified	Type	Size
.git	2021/5/1 2:03	File folder	
.settings	2021/4/17 13:38	File folder	
build	2021/5/1 2:03	File folder	
cmake	2021/4/17 13:38	File folder	
parallel	2021/4/30 22:44	File folder	
scripts	2021/4/28 16:14	File folder	
serial	2021/4/18 22:17	File folder	
writeup	2021/5/1 13:19	File folder	
.cproject	2021/4/16 22:01	CPROJECT File	5 KB
.project	2021/4/15 13:34	PROJECT File	1 KB
CMakeLists.txt	2021/4/30 15:42	Text Document	3 KB
README.md	2021/4/17 13:38	MD File	1 KB
slurm-2108835.out	2021/5/1 2:03	OUT File	206 KB

Fig. 1: The screenshot of the project folder.

B. File Structure

The structure of this self-determined project is very similar to the machine problems and pre-determined project, as can be seen from Fig. 1. The details about the information about the folders are enumerated as follows.

- 'build': CMake results, binary files and the .txt data files.
- 'cmake': the same as the Machine Problems.
- 'parallel': the source code and header files of the parallel part of the artifact.
- 'serial': the source code and header files of the serial part of the artifact.
- 'scripts': .slurm shell scripts for benchmarking on Campus Cluster and .m Matlab scripts for generating input data and post-processing the output data.
- 'writeup': .pdf project reports, .txt output of the benchmarking shell scripts and images.

As for how to play with the artifact, please refer to the Markdown file 'README.md'.

C. FEM

The FEM [3] is commonly used for solving the BVP of PDE numerically. The derivation of FEM can be carried out from either weighted residual method or variational formulation, the derivatives in the differential equation are carried out directly but simple approximate solutions are used. 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. The procedures for FEM can be briefly summarized as follows.

- Discretization: discretize the BVP into small elements associated with basis functions and coefficients.
- Assembly: Use Galerkin's method to formulate linear algebraic equations and evaluate them either using an-

alytical integration or numerical integration. And then assemble these components into finite element matrix.
- Solving: Solve the assembled sparse linear system and obtain the unknowns.

D. DDM

DDM was proposed by the German mathematician H.A.Schwarz in 1870 [4] 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 for reducing both computation time and the memory requirements on processors of modern computers [5].

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 sub-region that reside in other sub-regions 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. As one of the overlapped DDMs, Additive Schwarz method updates the Dirichlet boundaries simply based on the results of the previous iteration at the price of sacrificing accuracy compared to the multiplicative Schwarz method. The decomposition of the overlapping DDM when the total domain is decomposed into four sub-regions is illustrated in Fig. 2. The boundaries of the sub-regions are highlighted and explained in Fig. 3.

E. (Parallel) Conjugated Gradient Method

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. 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 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.

There exists many parallel iterative solvers among which the parallel conjugate gradient (parallel CG) method which is the most widely studied and used.

II. Fromulation

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 sub-regions for illustration purpose. Finally, we briefly introduce the conjugate gradient algorithm we used for solving the linear system. 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). The total problem is a two-dimensional square region consisting of $R \times R$ sub-regions whose side length is 6 in both vertical($\hat{y}$) and horizontal($\hat{x}$) directions. Thus, with a overlapping length of 1 between the regions that are adjacent to each other, the total regions is of size $(5N+1)\times(5N+1)$. The outer boundary of this total region is connected to the ground, that is to say, the value of electric potential is zero at the boundary. In the example for illustration purpose shown in Fig. 2, the total region is composed of 2×2 subregions. There are four objects of shape circle, ellipse, square and triangle that are also connected to the ground located respectively in the middle of the SW, SE, NE and NW sub-regions. For the more general cases that is composed of $R \times R$ sub-regions, the objects in the middle of each sub-region are all circle because it will be relatively easy to generate such geometry automatically for an arbitrary N using the Matlab script. A brief outline of the FEM developed for this specific toy electrostatic problem is described below with special reference to [7].

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,

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

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.

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

TABLE I: The color-to-sub-region table used for the sub-region colors of the DDM illustration in Fig. 2 and the text colors of the artificial boundary illustration in Fig. 3.

	sub-region 1	sub-region 2	sub-region 3	sub-region 4
Color	light blue	purple	green	red

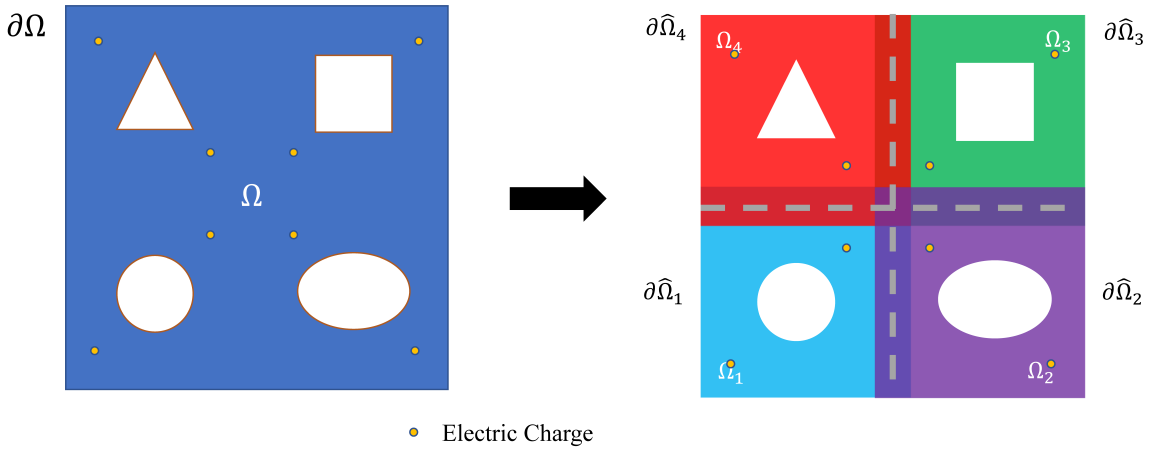


Fig. 2: Schematic of DDM. Ω denotes the total domain on which the final solution $\Phi(x, y)$ will be interpolated. $\partial\Omega$ denotes the boundary that isolates the total solution domain from the infinitely large space. $\partial\Omega$ will be divided to $\partial\hat{\Omega}_1, \ldots, \partial\hat{\Omega}_4$ 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 sub-regions $\Omega_1, \ldots, \Omega_4$ after domain decomposition as shown on the right. Different colors represent different sub-regions, this can be seen both from the figure and the Table I.

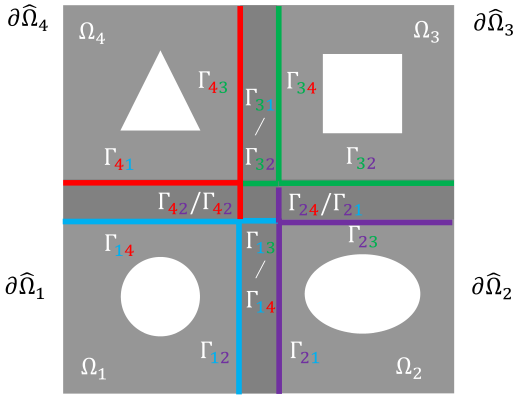


Fig. 3: Illustration of the artificial boundaries that reside in other sub-regions for each of the four overlapped sub-regions shown in Fig. 2. The respective colour indicates which sub-region the artificial boundary resides. The processor to which this sub-region belongs is the sender of the message at each iteration of the ASM if we use MPI in the distributed memory system.

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

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

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

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

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

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).

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

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

$\boldsymbol{K}$ 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 can be expressed as

$$\Phi(x,y) = \sum_{j=1}^{N} N_j(x,y)\phi_j \tag{10}$$

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 sub-regions are two major concerns in the geometry aspect of FE-DDM.

1) Meshing: Any polygon, no matter how irregular, can be represented exactly as a union of triangles [8]. 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, divided and visualized using the Partial Differential Equation Toolbox provided in Matlab 2021a. 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.

2) Domain Decomposition: A parallel program is one in which multiple processors are going to cooperate to solve one problem through either message passing in distributed memory system or cache coherence in shared memory system. The domain decomposition method, as illustrated in Fig. 2, is a method to decompose the solution domain into smaller sub-regions and is very suitable to implement in the distributed memory system in that we can solve the sub-regions separately and then the solution of the total domain is obtained through the boundary continuity conditions. Also, because we can combine shared memory programming with message passing, we can solve the sub-region problem locally with the help of OpenMP multi-threading. In this course project, we choose to implement a overlapping Schwarz DDM with a total sub-region number $M = R \times R$. Different ranks of the processors group, except for the root rank that is responsible for global reductions, execute almost identical instructions, the major difference in different ranks is the input data. Thus the DDM procedure itself is a kind of SPMD style of programming.

As for the communication, we need to know what to send/receive and where to send/receive because the communication between the sub-regions is based on MPI. Thus, besides generating the geometry information, we also need to convert the global indexing into local indexing in the Matlab scripts because each rank in the communication domain only knows the local information and send/receive the artificial boundary values (a.k.a. the ghost cells in our lecture) in the ASM using message passing. This is usually known as the process for figuring out what data do I need from others and who needs my data. For simplicity, although not every two of the sub-regions have overlapping region, we let every rank send to all the ranks and receive from all the ranks including itself. This makes the sending and receiving calls during the running of the additive Schwarz iteration possible. Although the procedure of finding out what and where to send and receive is quite complex in our overlapped DDM, the data structures required in the message passing is not complicated in that we can simply use a data type of float instead of user defined and committed data type.

To be more intuitive, we also take the 2×2 toy problem as an example. Based on the underlying divide-and-conquer philosophy, it divides the total domain Ω with boundary $\partial\Omega = \bigcup_{m=1}^{4} \partial\hat{\Omega}_i$ into 2×2 overlapped sub-regions $\Omega = \bigcup_{m=1}^{4} \Omega_m, \quad \Omega_m \bigcap_{m \neq n} \Omega_n \neq \emptyset$. In this example with only four sub-regions, all of the four sub-regions are at the boundary of the total domain. As can be seen from Fig. 2 and Fig. 3, each sub-region is overlapped with the other three sub-regions and all the four domains 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 sub-regions.

When choosing the scheme for dividing sub-regions, the solving speed of each sub-region 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 this course project, we let each of the sub-regions to have approximately equal number of nodes so that the decomposition yields approximately balanced loads. Also, the size of sub-regions includes the size of overlapping region should be determined by the available memory on each processor.

One of the major issues we need to decide when we parallelize using DDM based on MPI is how are we going to decompose the solution domain and the data each portion need. Unlike the column decomposition used in the literature that proposed Integral Equation based overlapped DDM [9], the sub-regions are evenly divided to square chunks in our problem setting, as illustrated by the dashed grey lines in Fig. 2 for a 2×2 case. Thus we need a square number of processes ($M = R^2 = 3, 9, 16, 25, 36, \dots$). Theoretical analysis [10][11] shows that the rate of convergence increases as the overlap between the sub-regions increases. Numerical experiments have shown that Schwarz methods perform well even if the overlap between neighboring subregions is quite small [11]. Thus we enlarge each of the sub-region by 0.5 meter and thus set the width of overlap region to be 1 in our toy DDM. In this way, the total domain Ω is divided into sub-regions $\Omega_1, \dots, \Omega_M$ where we not only have overlapped region shared by no more than two sub-regions but have overlapped crosspoint shared by the four nearby sub-regions. In sum, we choose to assign the $R \times R$ sub-regions with the same physical size of 6×6 and thus similar number of nodes with unknown coefficients in each sub-region. In this way, efficiency is maximized when load imbalance is minimized.

The ASM was first developed in [12]. To use ASM deal with the problem formulated in this project, we extend the formulation of the ASM described in [3] to the general form of iterative process written as Equ. (11).

$$\boldsymbol{K}_{\Omega_m\Omega_m}\boldsymbol{\phi}_{\Omega_m}^{(k+1)} = \boldsymbol{b}_{\Omega_m} - \sum_{r=1, r\neq m}^{R^2} \boldsymbol{K}_{\Omega_m\Gamma_{rm}}\boldsymbol{R}_{\Gamma_{rm}\Omega_r}\boldsymbol{\phi}_{\Omega_m}^{(k)} \tag{11}$$

To be more specific, for the example 2×2 problem shown in Fig. 2 and Fig. 3, Equ. (11) can be further expanded as Equ. (12), (13), (14) and (15). The subscript of the notation of the artificial boundary (ghost boundary) Γ_{ij} represents the sender (to which sub-region i the nodes belong) and the receiver (of which sub-region j the nodes are used as the artificial 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 by using local indexing to extract the entries in the local sub-region matrices and vectors (where the artificial boundary nodes (ghost nodes) locate in the local node array of each of the M sub-regions).

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

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

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

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

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

Here, in the FE-DDM-ASM iterative process, we need to use the CG solver in every iteration. And there will be MPI_Send/MPI_Recv communication at the end of every iteration.

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.

CG is a kind of Krylov subspace method and is the most popular iterative method for solving large systems of linear equations [13]. 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 [14] 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. (16) and Equ. (17) respectively such that they minimize the norm of the error $\|\boldsymbol{e}_{k+1}\|_{\boldsymbol{K}}^2 = \|\boldsymbol{\phi}_{k+1} - \boldsymbol{\phi}_k\|_{\boldsymbol{K}}^2$ at the k-th iteration. Here $\|\boldsymbol{x}\|_{\boldsymbol{K}}^2$ is defined as the inner product of $\boldsymbol{K}\boldsymbol{x}$ and $\boldsymbol{x}$.

$$\alpha_k = \frac{\|\boldsymbol{r}_k\|_2^2}{\|\boldsymbol{p}_k\|_{\boldsymbol{K}}^2} \tag{16}$$

$$\beta_k = \frac{\|\boldsymbol{r}_{k+1}\|_2^2}{\|\boldsymbol{r}_k\|_2^2} \tag{17}$$

The resulting Conjugated Gradient method [15] is given in Algorithm 1.

Algorithm 1 CG algorithm

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

As we can see, the CG method for linear system is composed of basic operations such as vector updates (saxpy), inner products and matrix-vector multiplication. Thus, the parallelization of the CG solver can be implemented by parallelizing these basic operations using OpenMP pragmas.

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 [16]. 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. Thus by using DDM, the finite element matrix in each sub-region will be smaller and thus the linear system tends to converge faster compared with non-DDM ones.

D. Summary of the two-level parallelism

As we can see from the previous discussion, DDM is often carried out on large multi-core and multi-node parallel machines for the scaling-up purpose. 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 describes parallelism between processes, each one with a separate address space whereas OpenMP provides parallelism within a process. We can take the OpenMP model and scale

it up to as large as a physical node would allow but beyond the physical node, you'd need to use MPI.

MPI is at the top level (coarse-grained); the solution domain is divided into small sub-regions, which are assigned to each node. The nodes communicate and synchronize with each other through non-blocking message passing such as MPI_Isend and MPI_Irecv. By using non-blocking modes, we would like to make communication and computation overlap as much as possible. To be more specific, the $R \times R$ sub-regions will communicate with each other and update coefficients reside at the artificial boundaries (ghost boundaries) of adjacent sub-regions using the ones associated with the local nodes based on MPI.

OpenMP is at the bottom level (fine-grained) for the parallel computation of CGM through OpenMP pragmas; 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. Doing fine-grained parallelism using OpenMP on shared memory system can reduce the copying costs compared to using MPI on distributed memory system.

For practical consideration of the limited node resources provided, the MPI can actually run on a multi-core machine by putting an MPI rank on hardware thread or every core rather than using the same number of physical nodes in the computing cluster network. Also, instead of simply using MPI_Init(), MPI_Init_thread() is required in the multi-threaded program.

In sum, from the DDM perspective, the finite element matrices for each of the sub-regions can be solved in parallel and the solution for the total domain will be generated after we get the solution for each sub-region when the ASM iterative process comes to an end (reaches convergence or reaches a fixed number of iteration), the solved parameters on each of the rank are all gathered at the root rank. From the perspective of linear system solver, the parallelism in FE-DDM can be explored by utilizing the parallelized iterative solvers.

III. Results

In this section, first we briefly describe the work flow of the main function of the parallel part and then the results produced by the artifact are shown and analyzed in detail. The time counting in this section has already taken load imbalance into account since we are doing tic-toc only at the root rank.

A. Description of the code of parallel part

1) We first use conventional routines to initialize MPI and OpenMP at the beginning of the main() function.
2) Each rank loads its own sub-region information to the C++ container 'std::vector' from the corresponding .txt file using function ReadSubdomainInfo(), which is also defined at the main() function.
3) Each rank assembles the local FEM matrix that includes both the interior nodes (those not on the artificial boundaries) and the interface nodes (those on the artificial boundaries) using Assemble() based on the information read from the previous step.
4) Then, by using function SubArray() and SubMatrix(), we extract the information required by the general iterative equation of the ASM as expressed in Equ. (11).
5) Next, we use function MPI_Gather() to let the root rank know how many interior nodes are in each of the sub-region and function MPI_Reduce() to do the summation of all the number of interior nodes in each of the sub-region and store the summation result into the root rank.
6) Now, we follow Equ. (11) to do the additive Schwarz iteration for a fixed number of steps. At the end of each iteration, every rank communicates the coefficient values with all the other ranks through function MPI_Isend(), MPI_Irecv() and MPI_Waitall(). And then each rank assigns its received data back to the coefficient array using function AssignSubArray(). In this step, it is critical to know what to send/receive and where to send/receive. The efforts to accomplish this goal is contained in the Matlab script '/scripts/autogeogen.m' which produces all the data required by the parallel part of this course project.
7) In every additive Schwarz iteration, each rank need to solve a linear system corresponding to the interior nodes of that sub-region. This solver is the CG solver accelerated by OpenMP.
8) At the rook rank, after we get the number of nodes in each sub-region, we do an exclusive sequential scan to calculate the offsets which will be used in the MPI_Gatherv() for gathering by the root rank all the solved coefficients and their corresponding global indices.
9) Finally we can use the gathered data and indices together to assemble the global coefficient array, which will be written back to a .txt file by the root rank.

B. Geometry of meshes

To have a better understanding of the varied problem size, we are going to visualize the mesh by varying two parameters. The first one is to vary the number of nodes in each of the sub-region while keeping the number of sub-regions unchanged. As shown in Fig. 4, from (a) to (l), there are respectively 60 ,117, 294, 477, 1087, 2234, 4311, 6895, 10681, 15201, 23708 and 41905 unknowns in one sub-region of the total domain which is composed of 2×2 such sub-regions.

The second one is not only to change the number of sub-regions $R \times R$ from $R = 2$ to $R = 6$ but also change the number of unknowns in each sub-region. In other words, we are going to vary the number of ranks in the communicator from 4 to 36 while increasing the computation load of each processing rank. As shown in Fig. 5, the number of unknowns per sub-region in each row is the same while the number of sub-regions $R \times R$ varies from $R = 3$ to $R = 6$. There are 60, 117, 478, 1544, 3898 unknowns in each sub-region respectively in the first, second, ..., fifth row. There are 3×3, 4×4, 5×5, 6×6 sub-regions in the first, second, ..., forth column.

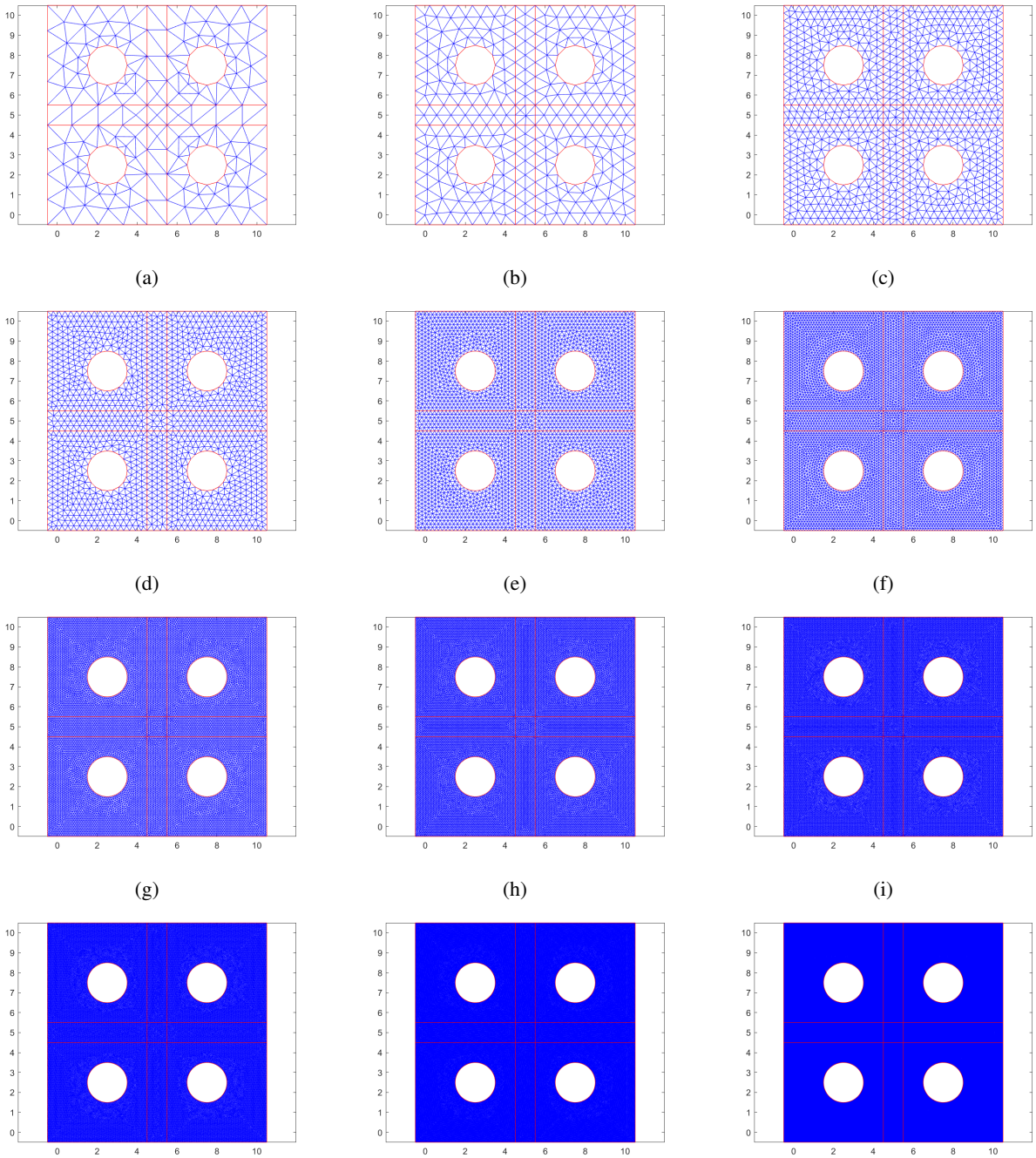


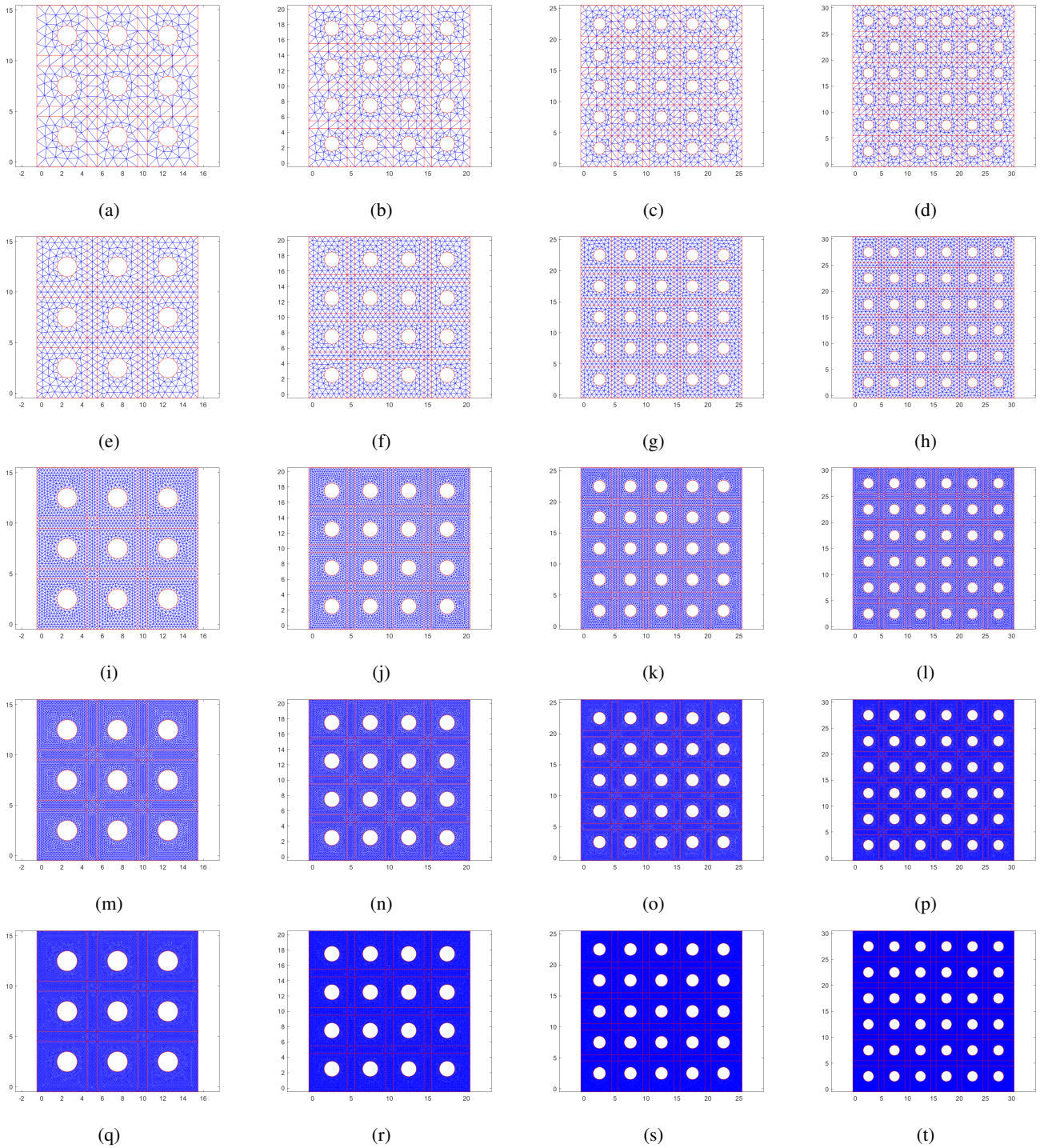
Fig. 4: The mesh of $R \times R$ sub-regions with $R = 2$ from coarse to dense.

C. Electric Potential Visualization

In Fig. 6 and Fig. 7, the final result of the electric potential are calculated by interpolation using both the coefficients in the whole domain calculated by the FE-DDM-ASM and the linear basis functions defined locally at each triangular element, as indicated by Equ. 10. There are 5 rows in total and the number of sub-regions $R \times R$ ranges from 2×2 to 6×6 is fixed in each row. Different columns in Fig. 6 and Fig. 7 represent different numbers of unknowns in each sub-region. Due to the limitation of page size, when all the columns are combined together, the 1st-5th column respectively represent 47, 60, 91, 135 and 477 unknowns in every single sub-region. As we can see, the result will be more and more accurate when we increase the number of unknowns in each sub-region and thus increase the quality of the FEM mesh.

D. Scaling

The coefficients for interpolating the electric potential in the previous subsection is generated through benchmarking the parallel code by varying the number of unknowns in each



(a) (b) (c) (d)

(e) (f) (g) (h)

(i) (j) (k) (l)

(m) (n) (o) (p)

(q) (r) (s) (t)

Fig. 5: Visualization of mesh: different rows have different numbers of unknowns in the sub-region while different columns have different numbers of sub-regions.

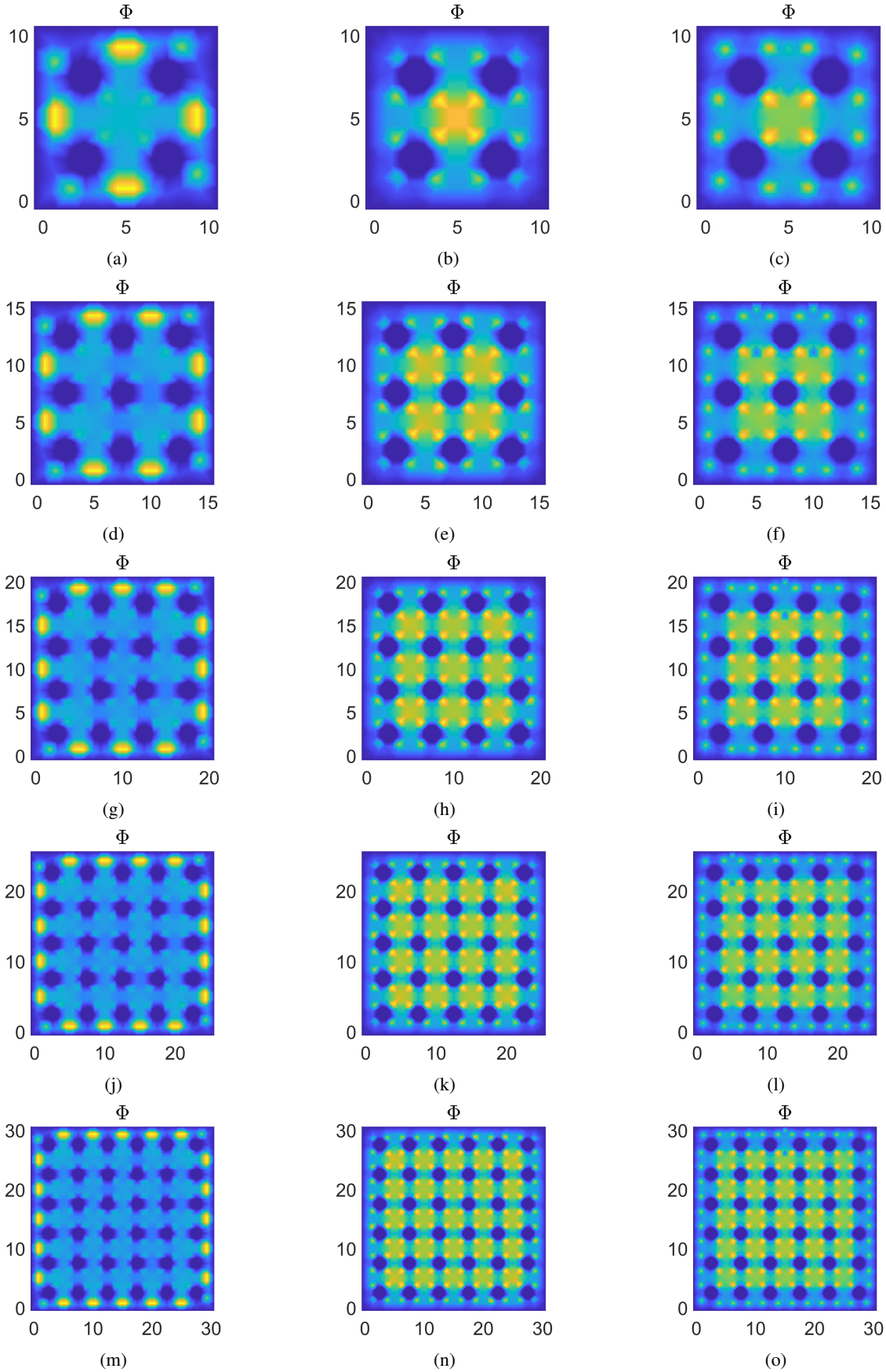


Fig. 6: Visualization of electric potential: different rows have different numbers of sub-regions while different columns have different numbers of unknowns in the sub-region.

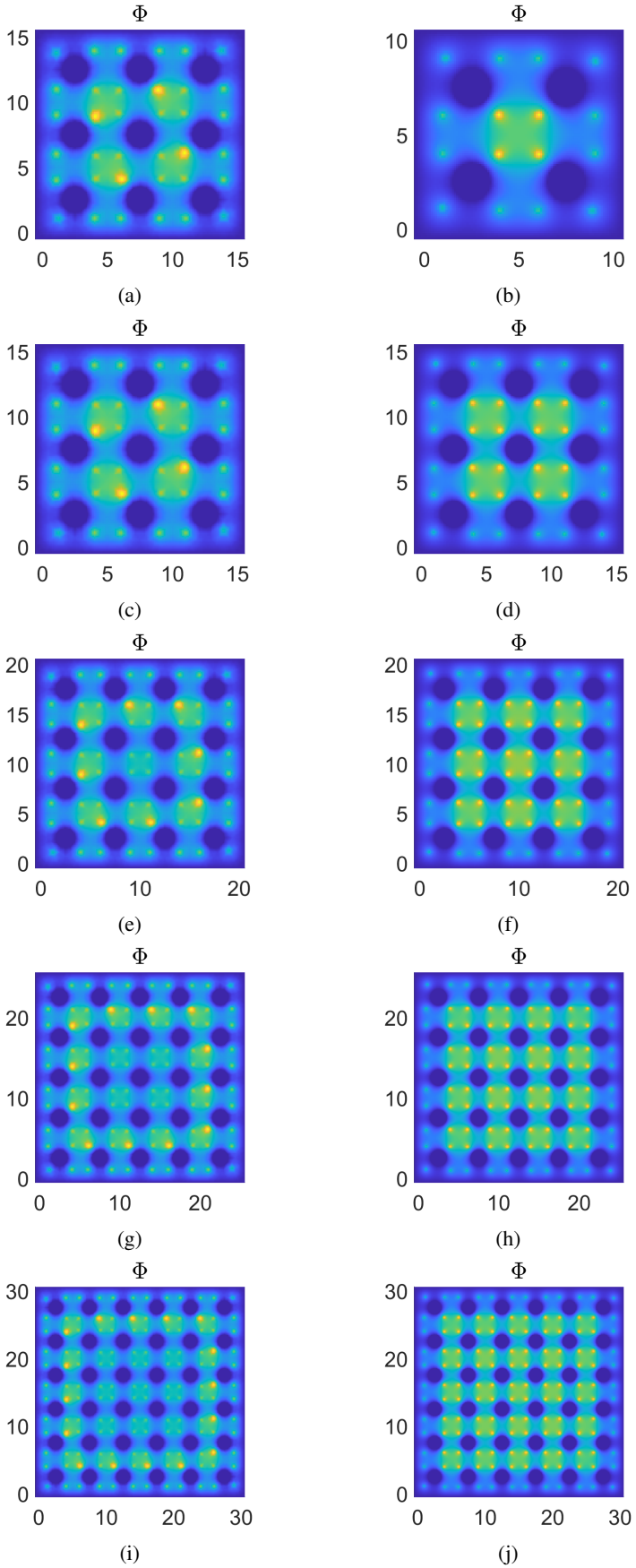


Fig. 7: Visualization of electric potential (continue): different rows have different numbers of sub-regions while different columns have different numbers of unknowns in the sub-region.

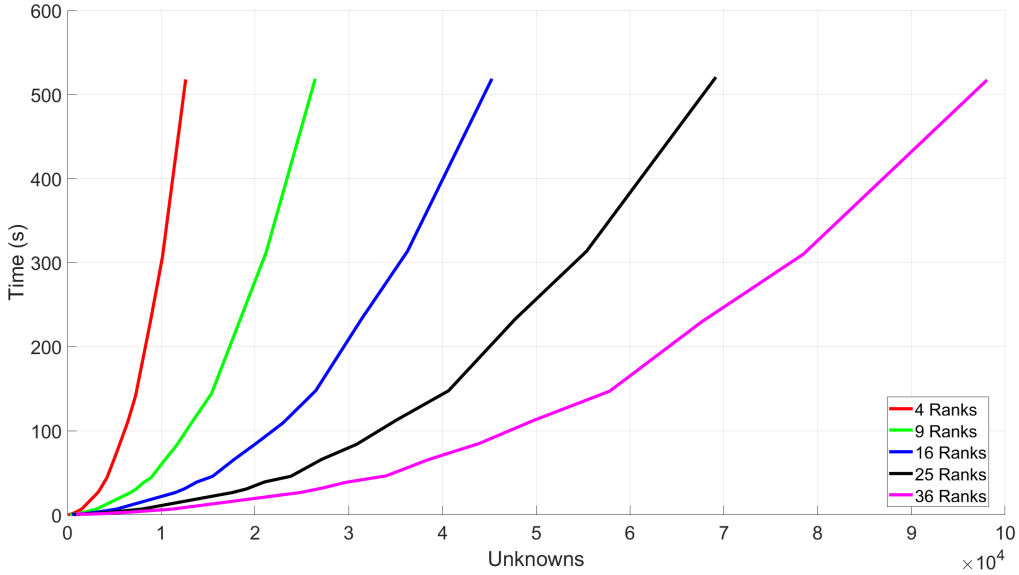


Fig. 8: The benchmarking result to demonstrate the scaling of the parallel algorithm.

sub-region and varying the number of sub-regions (processes). The scaling result is shown in Fig. 8. If we see the problems with the same number of unknowns as the same problem, we are using both weak scaling (a.k.a. scale speed up, different colors of lines in the figure) and strong scaling (fix the horizontal axis and compare the time consumption) in the project benchmarking. Because the size of one sub-region is fixed, the larger the number of sub-regions, the more the total number of unknowns is in the whole solution domain. Thus, although the number of unknowns varies significantly for the increasing number of sub-regions which is indicated by different color of lines, we expect similar amount of time used for generating the solution when the amount of computation work responsible by each rank is similar (actually almost the same). This is exactly what we observe in Fig. 8. In other words, we can retain the benefit of parallel execution by increasing the number of processing element used when the problem size increases. For example, as we can see, the number of unknowns solved by 36 ranks is about 9 times of that of the red line, but their time consumption is similar.

E. Speedup and Parallel Efficiency

This time, we let the serial version and the parallel version of the artifact do the same work. This time we will only do weak scaling (speedup) test by fixing the number of processing elements and increasing the problem size. In reality, we may not keep running the same problem on more and more processors, we solve larger problems on larger machines instead. The number of MPI ranks used in the parallel computation is 4, which indicates there are four sub-regions exist in the total domain. We test the performance improvement of the parallel version over the serial version by varying the number of unknowns in the total domain. That is to say, we use the advanced parallel version and less-efficient serial version to solve the same problem and do almost the same computation. When varying the problem size, the time consumption by the serial version and the parallel version is shown in Fig. 9. In addition, we use

$$Speedup = \frac{T(sequential)}{T(parallel)}$$

to calculate speedup and

$$Parallel \quad Efficiency = \frac{Speedup}{number \quad of \quad processors}$$

to calculate the parallel efficiency, and the result is shown in Fig. 10. As we can see, compared with the serial version, the parallel version achieves a relatively high efficiency (≈ 0.9) and speedup when the problem size (number of unknowns) is large. Although the parallel efficiency is relatively high, it is still not perfect due to several reasons. First, there is some overhead of parallelization. For instance, the communication cost is not negligible even though we already tried to overlap it with the computation time. Second, according to Amdahl's law, the speedup will also be limited by the sequential part in the code that is solely done by the root rank. Because we have already minimized this part by letting each MPI rank read in its own input, this part may not have significant impact on the efficiency.

ACKNOWLEDGMENT

The student would like to thank Prof. Lawrence Rauchwerger, Prof. Laxmikant V Kale and all the TAs for offering this parallel programming course during the global pandemic.

REFERENCES

[1] C. Multiphysics, "Introduction to comsol multiphysics®," *COMSOL Multiphysics, Burlington, MA, accessed Feb*, vol. 9, p. 2018, 1998.
[2] "Ansys hfss," ANSYS, Inc. 2021. [Online]. Available: https://www.ansys.com

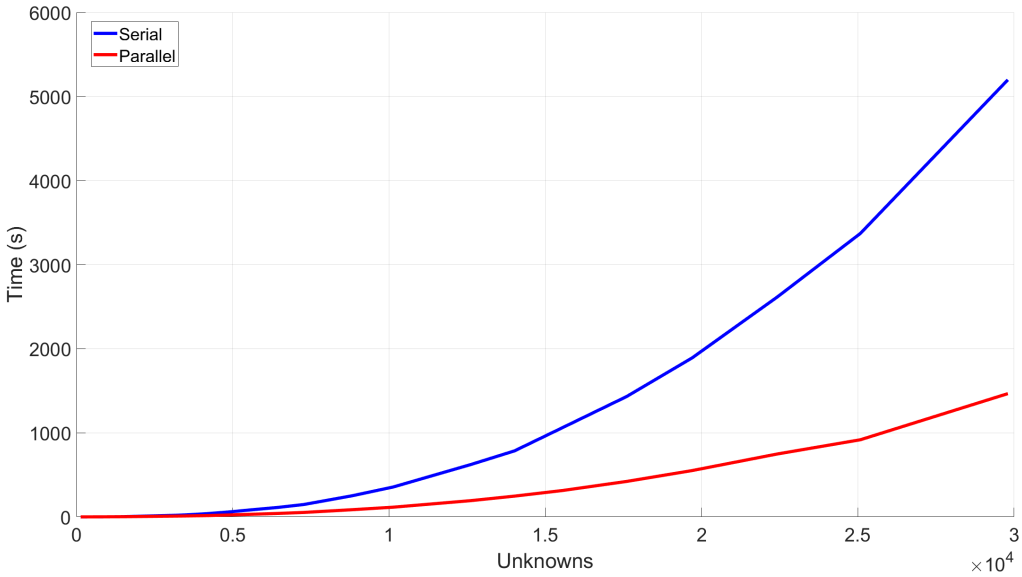


Fig. 9: Comparison of the time consumption by serial and parallel version when computing problem of different size.

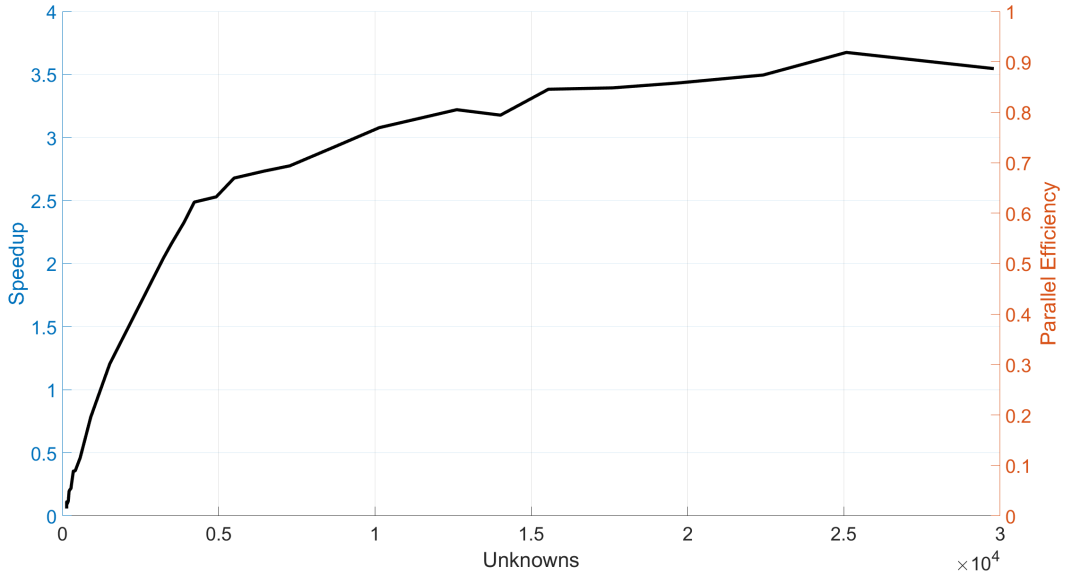


Fig. 10: Parallel efficiency and speedup.

[3] J. Jin, *The Finite Element Method in Electromagnetics*, ser. Wiley - IEEE. Wiley, 2015. [Online]. Available: https://books.google.com.hk/books?id=DFi-BgAAQBAJ

[4] H. A. Schwarz, "II. Ueber einen Grenzübergang durch alternirendes Verfahren," Wolf J. XV. 272-286. 1870 (1870)., 1870.

[5] 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.

[6] H. A. Schwarz, "Über ein die flächen kleinsten flächeninhalts betreffendes problem der variationsrechnung," in *Gesammelte Mathematische Abhandlungen*. Springer, 1890, pp. 223–269.

[7] J.-M. Jin, *Theory and computation of electromagnetic fields*. John Wiley & Sons, 2011.

[8] P. P. Silvester and R. L. Ferrari, *Finite elements for electrical engineers*. Cambridge university press, 1996.

[9] 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.

[10] 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.

[11] ——, "Domain decomposition algorithms with small overlap," *SIAM Journal on Scientific Computing*, vol. 15, no. 3, pp. 604–620, 1994.

[12] S. V. N. A. M. Matsokin, "The schwarz alternation method in a subspace," *Soviet Math. (Iz. VUZ)*, vol. 29, pp. 78–84, 1985.

[13] J. R. Shewchuk *et al.*, "An introduction to the conjugate gradient method without the agonizing pain," 1994.

[14] 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.

[15] V. Dolean, P. Jolivet, and F. Nataf, *An introduction to domain decompo-*

sition methods: algorithms, theory, and parallel implementation. SIAM, 2015.

[16] A. Greenbaum, *Iterative methods for solving linear systems*. SIAM, 1997.