Министерство образования и науки Российской Федерации
Московский физико-технический институт (государственный университет)
Факультет управления и прикладной математики
Кафедра «Интеллектуальные системы»
при федеральном исследовательском центре
«Информатика и управление» РАН

Лукошкин Владислав Сергеевич

Обобщённая вычислительная схема конечных элементов в проблемах с несколькими масштабами

010990 — Интеллектуальный анализ данных

Выпускная квалификационная работа магистра

Научный руководитель:
д. ф.-м. н. Воронцов Константин Вячеславович

Москва
2020

Обобщенная вычислительная схема конечных элементов в проблемах с несколькими масштабами

Лукошкин Владислав Сергеевич

Представлено в Сколковский институт науки и технологий
Май 29, 2020

Аннотация

Данная работа посвящена практической реализации обобщённого многомасштабного метода конечных элементов. Последний зарекомендовал себя достойной заменой классического метода конечных элекментов (МКЭ) в решении проблем с резко неоднородной структурой. Помимо распараллеливания и тестирования алгоритма обобщённой вычислительной схемы, в статье проводится анализ его вычислительной сложности в сравнении с традиционным методом. Также изучаются потенциальные трудности, которые могут возникнуть в применении обобщённого МКЭ, и возможные пути к их решению.

Научный руководитель:
Имя: Иван Валерьевич Оселедец
Ученое звание, степень: Ph.D.
Должность: Профессор

Acknowledgments

I am grateful to my supervisor, Ivan Oseledets, for his guidance and mathematical tips that have led to improved performance of the final algorithm. I also acknowledge the use of the HPC cluster, Zhores, for obtaining the results presented here.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With rapidly growing computational resources, the number of ambitious problems that require processing capacities exceeding available ones is growing. So, there is always the need to develop new tools that can partially lift the computational burden. Applied physics is one of many domains sensible to the amount of accessible computing power. Often, a numerical solution is the only possible way to address contemporary scientific or engineering problems. Among them, one should highlight the class of problems described with partial differential equations (PDEs). Modeling processes in nature boils down to solving PDEs. Just for instance, such processes can be:

- flow in heterogeneous porous media
- seismic wave propagation
- turbulent transport in high Reynolds number flows
- tensile failure in fiber-reinforced composites.

Although, it is a well-studied issue: there exist many methods and libraries that are capable of casting PDEs; there are also multiscale problems which are difficult to deal with using a conventional approach (precisely due to computational cost). To this end, multiscale methods emerge. They are able to capture the multiscale structure and fastly build an approximation of the solution without a great sacrifice in the accuracy. In this work, we are focusing on the popular representative of multiscale methods - the Generalized Multiscale Finite Element Method (GMsFEM)[1]. To solve the underlying PDE of the problem, its authors propose to precalculate offline space by solving many local problems on a fine mesh. At the online stage, one can reuse the functions from offline calculations to construct multiscale functions and solve the PDE in the weak formulation as it goes in the regular finite element setting. A particular interest in GMsFEM is connected with its effectiveness in the modeling of two-phase flow and other problems that require a large number of forward simulations:

$$\partial_t u + \text{div}(\kappa(x,u)\nabla u) = f$$

$$\text{div}(\kappa(x,u)\nabla u) = f$$

Annotation: *Simulation of time-dependent problems (1st eq.) and solving with Picard iteration (2nd eq.)*

At the time this paper is written, there are few GitHub -repositories demonstrating method's applications on a specific problem and no usable libraries. What about commercial solvers - they exist but are limited in their functionality. It appears that they are not suitable for the problems we are interested in. In this way, we get to the main target of the current work: the development of an open-source hydrodynamic simulator on the base of GMsFEM. To this end, the objectives are the following:

- Study available open-source platforms for solving PDEs with FEM
- Implement the baseline method for solving multiscale problems
- Parallelize the algorithm with existing frameworks

Chapter 2

Background

2.1 Finite Element Method

The finite element method (FEM) is the most widely used method for solving PDEs from applied physics. Its popularity is due to the deep theoretical background and can be applied to a wide range of problems:

- heat transfer
- fluid flow
- structural analysis
- mass transport
- electromagnetic field distribution

The main concept of FEM is that any continuous variable (temperature, pressure, displacement) in some domain can be approximated with a discrete model built on the set of piecewise continuous functions, locally supported on the corresponding elements from a finite collection of subdomains. According to this idea, we can get to the following algorithm:

1. Generate mesh with a finite number of nodes
2. Values of the target variable are unknown and are to be determined
3. The domain under examination is split into a finite number of subdomains (elements) that have common nodes.
4. The continuous target variable is approximated on each element with polynomial determined by the nodal values (degrees of freedom - dofs). Coefficients of polynomials are chosen in the way that the target variable is continuous on the boundary of elements.

For a better understanding, let us refer to the mathematical notation. Consider Poisson's equation

$$-\Delta u = f \tag{2.1}$$

on a domain $\Omega \subset \mathbb{R}^d$ with essential conditions (please, refer to table 2.1 for naming conventions of condition types at the end of the subsection)

For any test function v integrable in the domain, with square-integrable derivatives Ω, and is zero on the boundary $\partial\Omega$ (easy to write $v \in H_0^1(\Omega)$, H_0^1 - Sobolev space), the following holds:

$$\int_\Omega -\Delta u \; v dx = \int_\Omega \nabla u \nabla v dx = a(u, v)$$

Here, integration by parts has been applied. Form a is called the stiffness form. The integration of a test function with the right-hand side gives the load vector $l(v) = \int_\Omega f v dx$

In defined components, we can argue that the weak solution to equation 2.1 is characterized by:

$$u \in V \qquad \text{such that} \quad a(u, v) = l(v) \qquad \forall v \in V$$

where $V = \{v \in L^2(0, 1) : \quad a(v, v) < \infty \;\; \& \;\; v(\partial\Omega) = 0\}$

An interested reader can find the proof of existence and uniqueness of the solution in the literature devoted to FEM's overview (e.g., Hoang et al. (2019) [2])

To finish the transition from PDEs to algebraic equations (in stationary case) or ordinary differential equations (ODEs - in case of time-dependent problems), we decompose the solution (in fact, the approximation to it) into a sum of basis functions of scarce function space $\hat{V}$. It is scarce in the sense that it is finite: contains a finite number of simple functions - $\phi_i, \quad i = \overline{1, n}$

$$\tilde{u} = u^i \phi_i$$

If, as test function, we take ϕ_i from the same basis we used for representing the solution approximation (or φ_i from similar but different finite space), we will get the linear system $Au = b$.

$$A = [a(\phi_i, \phi_j)]_{i,j}, \;\; i, j = \overline{1, n}; \qquad b = (l(\phi_1), \ldots, l(\phi_n))^\top$$
$$u = (u_1, \ldots, u_n)^\top$$

The choice of basis functions affects the computational cost of the algorithm and the accuracy of the approximation. Usually, ϕ_i-s are locally supported functions, defined on the mesh elements - this makes stiffness matrix A sparse, directly reducing the number of entries to be calculated and accelerating the work of a direct solver or iterative solver, depending on the linear system. On how to successfully choose basis functions and make discretization of a computing domain, one may

have a look at the materials [3, 4].

The tables below present the naming conventions used throughout the paper

Boundary Condition	Variational Name	Proper Name
$u(x) = 0$	essential	Dirichlet
$u'(x) = 0$	natural	Neumann

Table 2.1: Naming conventions
for two types of boundary conditions (Brenner, 1994[5])

K	coarse element
ω	neighborhood
n_K	# of K-s along one of dimensions
$n_\omega = n_K - 1$	# of ω-s along one of dimensions
N_ω	total # of ω-s
N_κ or N_μ	# of hyperparameter values μ_j
N_ℓ^D	# of fine elements (FE) in $D: \ \omega, \ \Omega, \ K$
n_ℓ^D	# of FE in D along one of dim-s; $D: \ \omega, \ \Omega, \ K$
V	some function space (FS)
V^ω or $V(\omega)$	FS over a mesh defined on ω
V_h	FE FS over a mesh with fineness h
V_{tag}	FS of tag: snap, off, on
N_{tag}	$\dim(V_{\text{tag}})$; tag: off, on

Table 2.2: List of symbols
used throughout the paper

2.2 Baseline method - GMsFEM

To deal with multiscale problems, one needs to make a tiny mesh to solve them with FEM. A large number of cells can be a reason to think of other methods which account for multiple scales and are less demanding in resources. GMsFEM [1] is an example of such. The method starts with two-level discretization: along with a coarse mesh $\mathcal{T}_H$, one also constructs a fine mesh $\mathcal{T}_h$. They both cover just a subregion of the domain of interest. After finishing computations on one subdomain, the solver proceeds with the next if the pipeline is serial. Otherwise, it can be viewed as many multiscale solvers working almost independently. only its subregions. Therefore, the instant computational workload in GMsFEM is determined by the partition size of the coarse mesh. To understand the method's concept, let us refer to a concrete example:

$$\text{div}(\kappa(x, \mu)\nabla u(x)) = f(x) \quad \text{in } \Omega \tag{2.2}$$

Here, u is known on $\partial\Omega$, and μ is a parameter (in the permeability coefficient in case of porous media applications) of multiscale nature, i.e., has high variations in the domain. Assume, we know the limits that confine the range of values of μ, so that it can be discretized $\mu_j,\ j = \overline{1, N_\mu}$. We define a neighborhood ω_i as a union of (four - depends on the discretization) adjacent coarse elements that share the same vertex on the coarse mesh (see fig. 2.1).

$$\omega_i = \bigcup \{K_j \in \mathcal{T}^H \mid x_i \in \overline{K}_j\}$$

Having done these steps, we can start building a snapshot space $V_{\text{snap}}^{\omega_i}$ - a space of functions representing a response of the simulated environment to different input parameters.

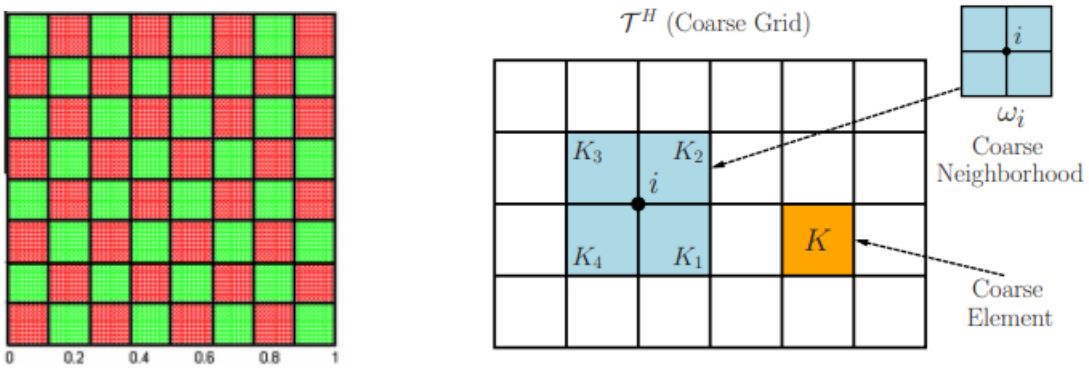

Figure 2.1: Two-level mesh representation
(left): coarse mesh - a checkerboard pattern, fine mesh - ripple which can be seen on a separate coarse element (the image is cropped from [6]); **(right):** coarse mesh structure (cropped from [7])

2.2.1 Snapshot Space

Option 1.

For each neighborhood w_i and parameter value μ_j, we consider the homogeneous version of the original problem 2.2:

$$\begin{aligned} -\text{div}(\kappa(x, \mu_j)\nabla\psi_{k,j}^{(i),\,\text{snap}}) &= 0 \quad \text{in } \omega_i, \\ \psi_{k,j}^{(i),\,\text{snap}} &= \delta_{k,j} \quad \text{on } \partial\omega_i \end{aligned} \tag{2.3}$$

To find the kth snapshot function $\psi_{k,j}^{(i),\,\text{snap}} \in V_{\text{snap}}^{\omega_i}$, we put discrete delta function at each point of the fine mesh on the boundary of a neighborhood and solve the problem 2.3.

$$\delta_{k,j}(x_l^{(i)}) = \begin{cases} 1, & l = k, \\ 0, & l \neq k \end{cases} \qquad x_l^{(i)} \text{ on } \partial\omega_i \bigcap \mathcal{T}_h$$

Option 2.

Alternatively, we can take $V_{\text{snap}}^{\omega_i}$ as the function space $V_h^{\omega_i}$ we would use to solve 2.3 on the fine mesh within a neighborhood w_i with FEM. Since such snapshot space "contains" no information about the equation parameters and bigger than the one resulting if following the first option, one may want to apply the model reduction procedure used for obtaining online space, while iterating over all μ_j - see [8].

2.2.2 Offline Space

We perform a spectral decomposition of the space of snapshots $V_{\text{snap}}^{\omega_i}$ to obtain an offline space $V_{\text{off}}^{\omega_i}$. More precisely, we consider the generalized eigenvalue problem in $V_{\text{snap}}^{\omega_i}$ (which is the discrete version of the problem of finding eigenfunctions of $\text{div}(\kappa\nabla\bullet)$-operator):

$$M\Psi_k = \lambda_k S\Psi_k \tag{2.4}$$

We re-numerate functions $\psi_{k,j}^{(i),\,\text{snap}}$ using a single index: $k, j \to m$, so the mass and stiffness matrices, M and S, are as follows:

$$M_{mn} = \int_{\omega_i} \bar{\kappa}\ \psi_m^{(i),\,\text{snap}}\ \psi_n^{(i),\,\text{snap}}$$
$$S_{mn} = \int_{\omega_i} \bar{\kappa}\ \nabla\psi_m^{(i),\,\text{snap}} \nabla\psi_n^{(i),\,\text{snap}}$$

Here, $\bar{\kappa}(x) = t_j\kappa(x, \mu_j)$ is averaged over μ (with some prescribed positive weights).

We then choose N_{off} eigenvectors corresponding to the dominant eigenvalues. Their coordinates will determine linear combinations

$$\psi_k = \Psi_{kl}\psi^l \tag{2.5}$$

that constitute an offline space $V_{\text{off}}^{\omega_i}$. Moderate compressions allow us to reuse offline functions effectively during repeating calculations.

It is worthwhile noting that if we solve 2.3 with FEM, then we know the dofs $R^{(i),\,\text{snap}}$ of snapshot functions $\psi_p^{(i),\,\text{snap}}$ in the fine-grid basis $V_h = \{\phi_q^{(i)}\}$ supported within ω_i:

$$\psi_p^{\text{snap}} = R_{pq}^{\text{snap}}\phi^q$$

In this way, we can write:

$$M = R^{\text{snap}}\ \overline{M}(\bar{\kappa})\,(R^{\text{snap}})^\top$$
$$S = R^{\text{snap}}\ \overline{S}(\bar{\kappa})\,(R^{\text{snap}})^\top$$

$\bar{M}(\bar{\kappa})$ and $\bar{S}(\bar{\kappa})$ are mass and stiffness band matrices of $V_h^{\omega_i}$. The argument in parantheses specifies the permeability used for matrix assembling.

$$\overline{M}(\varkappa)_{mn} = \int_{\omega_i} \varkappa \, \phi_m \, \phi_n$$
$$\overline{S}(\varkappa)_{mn} = \int_{\omega_i} \varkappa \, \nabla\phi_m \nabla\phi_n$$

2.2.3 Online Space

As for online space, we repeat the same model reduction procedure with a target permeability coefficient (the one for which we want to solve the equation - it can be fixed or depend on time or iteration step) instead of averaged.

Again, we extract the subspace of the given function space - now it is an offline space $V_{\text{off}}^{\omega_i}$, - by solving 2.4 with the following matrices:

$$M_{mn} = \int_{\omega_i} \kappa \, \psi_m^{\text{off}} \, \psi_n^{\text{off}}$$
$$S_{mn} = \int_{\omega_i} \kappa \, \nabla\psi_m^{\text{off}} \nabla\psi_n^{\text{off}}$$

The resulting N_{on} eigenvectors corresponding to the dominant eigenvalues define the basis of an online space $V_{\text{on}}^{\omega_i}$ by the formula 2.5.

A more efficient way to assemble matrices is to leverage the sparsity of matrices in the space of fine-grid functions within ω_i:

$$M = R^{\text{off}} \, \overline{M}(\kappa) \left(R^{\text{off}}\right)^\top$$
$$S = R^{\text{off}} \, \overline{S}(\kappa) \left(R^{\text{off}}\right)^\top$$

where $R^{\text{off}} = \Psi^{\text{snap}} R^{\text{snap}}$. Thus, for the dofs of online functions, we can write:

$$R^{\text{on}} = \Psi^{\text{off}} R^{\text{off}} = \Psi^{\text{off}} \Psi^{\text{snap}} R^{\text{snap}}$$

2.2.4 Global Coupling

In order to obtain a conforming basis [9], we need to multiply online functions by a partition of unity functions [10, 11]. The latter satisfy to:

$$
\begin{aligned}
-\text{div}(\kappa \nabla \chi^{(i)}) = 0 \qquad & \text{in all } K \subset \omega_i \\
\chi^{(i)} = g_i \qquad & \text{on } \partial K \setminus \partial \omega_i (\forall K \subset \omega_i) \\
\chi^{(i)} = 0 \qquad & \text{on } \partial \omega_i
\end{aligned}
$$

where g_i is a continuous linear function on all edges of ∂K

So, the multiscale basis is

$$
V_{\text{ms}} = \text{span}_{i,j} \left(\chi^{(i)} \, \psi_j{}^{(i),\,\text{on}} \right)
$$

And the corresponding dofs are

$$
R_{pk}^{\text{ms}} = \chi_k R_{pk}^{\text{on}}
$$

We find the approximation in the form:

$$
\tilde{u} = u^i \, \psi_i^{\text{ms}} \qquad \psi_i^{\text{ms}} \in V_{\text{ms}} \tag{2.6}
$$

In a Galerkin formulation (a scheme in the weak formulation [12]), we write the stiffness matrix:

$$
\begin{aligned}
A_{ip,jq} &= \int_\Omega \kappa \nabla \psi_p^{(i),\,\text{ms}} \, \nabla \psi_q^{(j),\,\text{ms}} = \int\limits_{\omega_i \bigcup \omega_j} \kappa \nabla \left(R_{pk}^{(i),\,\text{ms}} \phi^{(i),k} \right) \nabla \left(R_{qk'}^{(j),\,\text{ms}} \phi^{(j),k'} \right) \\
&= R_{pk}^{(i),\,\text{ms}} R_{qk'}^{(j),\,\text{ms}} \int\limits_{\omega_i \bigcup \omega_j} \kappa \nabla \phi_k^{(i)} \nabla \phi_{k'}^{(j)} = R_{pk}^{(i),\,\text{ms}} R_{qk'}^{(j),\,\text{ms}} A_{kk'}^{(i,j)}
\end{aligned}
$$

Define the set of relative distances to the 'top-right' adjacent neighborhoods [1]

$P = \{1,\ n_\omega - 1,\ n_\omega,\ n_\omega + 1\}$ and the set of pairs $I = \left\{ i, j \mid i, j = \overline{1, N_\omega} : |i - j| \in P \right\}$.

If $\omega_i \bigcap \omega_j = \varnothing$, what is the case when $|i - j| \notin P \bigcup \{0\}$, then $A_{kk'}^{(i,j)} = 0$ and, thus, $A_{ip,jq} = 0$

Consider diagonal blocks of A:

$$
A_{i:,i:} = R^{(i),\,\text{ms}} \, A^{(i)} \, \left(R^{(i),\,\text{ms}} \right)^\top \text{ [2]}
$$

Here, $A_{kk'}^{(i)} = \int\limits_{\omega_i} \kappa \nabla \phi_k \nabla \phi_{k'}, \qquad k, k' = \overline{1, \dim[V_h^{\omega_i}]}$.

All the integrand functions relate to the integration domain [3].

As for off-diagonal blocks, $(i, j) \in I$, we define the form below:

$$
A_{mm'}^{(i,j),\,\text{trunc}} = \int\limits_{\omega_i \bigcap \omega_j} \kappa \nabla \phi_m \nabla \phi_{m'} \qquad m, m' = \overline{1, \dim[V_h(\omega_i \bigcap \omega_j)]}
$$

[1] $\{ j - i \mid j > i : \omega_i \bigcap \omega_j \neq \varnothing \}$

[2] colon in place of tensor's index means a full slice in the latter's dimension

[3] parameters are restricted to the domain, basis functions are from the function space defined on this domain

Again, the same implicit rule applies [3].
$A^{(i,j)}_{kk'}$ and $A^{(i,j),\,\text{trunc}}_{mm'}$ share the same non-zero entries. However, the former contains basis functions from different function spaces, namely $V_h^{\omega_i}$ and $V_h^{\omega_j}$. Typically, existing software can assemble mass or stiffness matrices fast if a single basis is used, and not the other way around. To this end, one may favor coupling via $A^{(i,j),\,\text{trunc}}_{mm'}$:

1. get indices of dofs of $V_h(\omega_i \bigcap \omega_j)$'s basis functions in the spaces $V_h^{\omega_i}$ and $V_h^{\omega_j}$: Λ_i, Λ_j.
2. find necessary permutations σ_i, σ_j to restore the order of the sliced dofs as it should be in the space $V_h(\omega_i \bigcap \omega_j)$
3. $\forall (i,j) \in I, \qquad A_{i:,j:} = R^{(i),\,\text{ms}}_{:\,\sigma_i(\Lambda_i)} A^{(i,j),}_{\text{trunc}} \left(R^{(j),\,\text{ms}}_{:\,\sigma_j(\Lambda_j)}\right)^{\top}$ [2]

After assembling diagonal blocks and blocks from index set I, we get the upper-right 'half'/triangle of the global matrix A filled. Since it is symmetric, we finish coupling by copying and transposing the entries to bottom-left positions.
The rest is easy - we compute the load vector:

$$b_{ip} = \int_{\Omega} f\ \psi_p^{(i),\,\text{ms}} = \int_{\omega_i} f\ R_{pk}^{(i),\,\text{ms}} \phi^{(i),k}$$

$$b_{i:} = R^{(i),\,\text{ms}}\, b^{(i)}, \qquad b_k^{(i)} = \int_{\omega_i} f\ \phi_k^{(i)}$$

Depending on the size, the system $Au = b$ can be solved with a direct or iterative solver. The substitution of the resulting dofs u in 2.6 gives an accurate approximation to the multiscale problem solution 2.3.

Chapter 3

Methodology and Experimental Setup

3.1 Software

In this work, for solving PDEs on a fine mesh, we use an open-source platform, FEniCS [13]. Also, we consider the results it gives for a multiscale problem (wherever it is possible to obtain them with FEM not crashing down because of the overload) as a benchmark. The rest of the algorithm is written with Python: matrix-vector operations with NumPy, solving linear systems and generalized eigenvalue problems - SciPy. Some parts of the code require many calls of the interpreter (basically, for-loops), so they have been re-written with C++ and wrapped with pybind11 to make them callable from Python. Since the calculations in the construction of spaces (snapshot, offline, online) recur for each neighborhood and parameter value, and assembling of different blocks of the global matrix can go independently, it is reasonable to parallel the work. We do it by leveraging OpenMPI. For an arbitrary number of computing nodes available, one needs to write auxiliary algorithms for the work distribution amidst the processes. We confine ourselves to considering only the case when the number of cores is equal to the number of neighborhoods. This choice reduces the number of communications among processes. (Off-topic: the mesh geometry also affects the communication time. Take a look at figure 3.1. Triangular cells have maximum three 'bottom-right' overlaps of equal area, while square cells - four, and they of different size.) The module development and unit tests are conducted in a Docker container. We run the experiments in Singularity containers on the HPC cluster, Zhores.

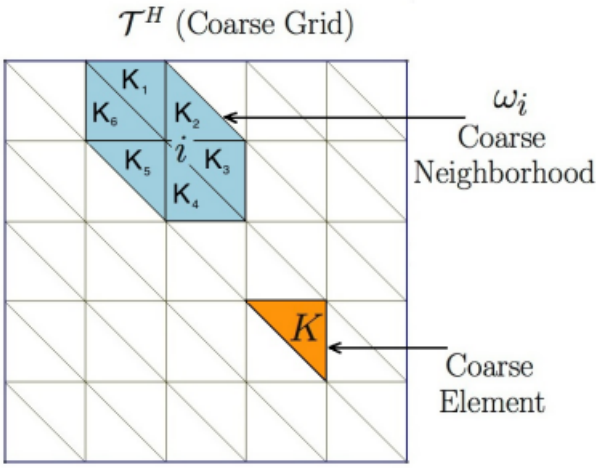

Figure 3.1: Triangular mesh.
The image is taken from [8]

3.2 Metrics

As stated in the previous section, we measure the solver performance by comparing its output with what a standard FEM produces. More specifically, we calculate the L_2 distance between these two approximations. In addition to accuracy criteria, a good multiscale solver should meet timing requirements. Possible, that offline space construction may be time-consuming. However, one run of online calculations must be faster than the execution of a regular FEM approximation procedure. For multiscale problems where the equation is to be solved several times, time spent for both, construction of the offline space and online calculations, should be smaller than the total time it takes FEM to make all runs.

Chapter 4

Results and Discussion

4.1 Synthetic Experiments

The section is called so, because of the way the equation coefficients are created. They do not relate to a specific problem from the physics domain. However, this fact does not make the simulation easy. Randomly generated vertical and horizontal strips of varying length form a mask of the permeability coefficient. Thus, the distribution of high-contrast regions may have an irregular structure, which causes issues during the discretization. If permeability coefficients are identical or similar in some neighborhood ω_i, then the union $V_{\text{snap}}^{\omega_i} = \bigcup_j V_{\text{snap},j}^{\omega_i}$ of the corresponding snapshot functions will give "rank-deficient" basis. Which, in turn, leads to singular matrices in the generalized eigenvalue problem. There are two options one may use to cope with this impediment.

1. Re-discretize the computational domain: if there are many "empty" regions, make coarse elements bigger and fine elements smaller (the latter helps to keep the balance)
2. Make matrices regular
    - by adding a small value to the diagonal elements
      (- degrade approximation accuracy)
    - with random perturbations [14]

With regard to the right-hand side (RHS), we just select the expressions that have an uncomplicated analytical form, though the choice is not limited to this matter.
To test the implemented solver, we start by considering the case with no parameter. Following this scenario, we only need to construct the snapshot and offline spaces. For the second one, the target permeability coefficient is used during the build procedure. All outcomes of this stage are equivalent to online computations. The following two figures present the comparison of approximated solutions obtained with GMsFEM and FEM (2nd row, in the respective order) for two different settings (1st row: permeability - left, source term - right).

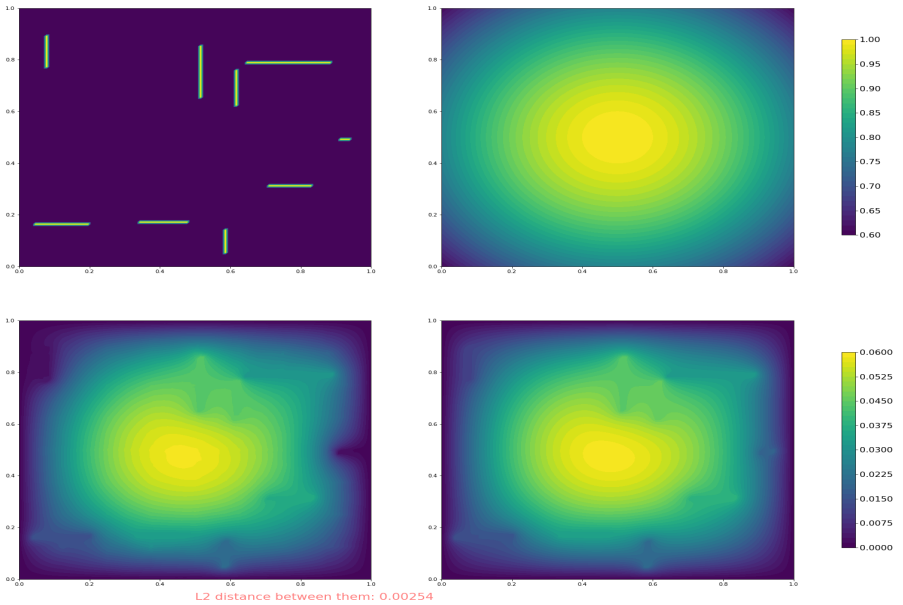

Figure 4.1: Experiment 1-1. No parameter. Exponential RHS $f = \exp(-(x-0.5)^2 - (y-0.5)^2)$

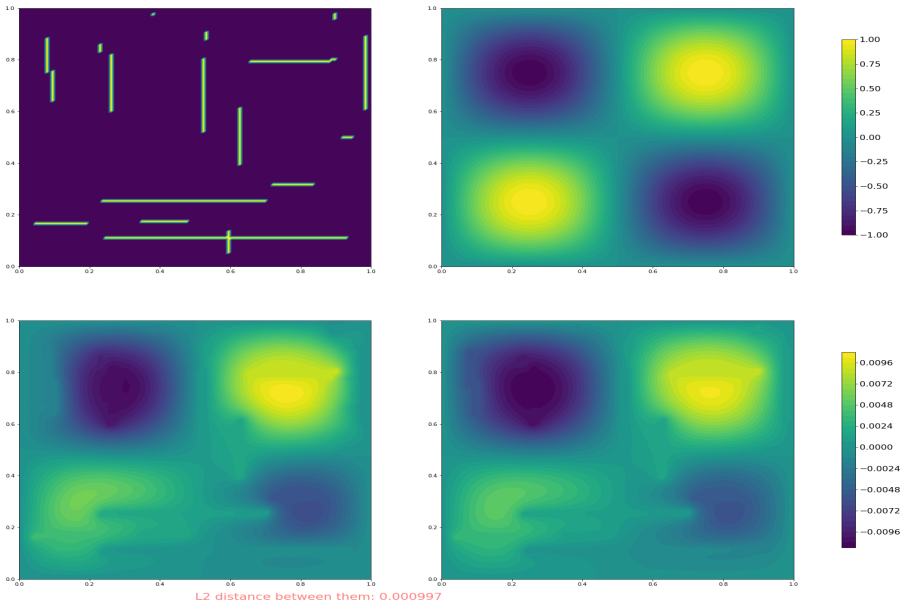

Figure 4.2: Experiment 1-2. No parameter. Sinusoidal RHS $f = \sin 2\pi x \sin 2\pi y$

GMsFEM is not a proper choice for this problem (applying it, we expect a large input parameter space). The experiments with the latter serve more as a validation step. They demonstrate the ability of GMsFESolver to build a decent approximation of the solution comparable to the one obtained with FEM.

The next step is to tackle problems with a set of different input parameters. At this point, it is important to make a compressible snapshot space. One should pay attention to the eigenvalue decay. There are different techniques aimed at improving its rate: oversampling neighborhoods [15], using different mass forms in generalized eigenvalue problem [6] We conduct experiment with a small set of permeability coefficients. All stages of the GMsFEM are involved. The approximation attains an adequate accuracy. The timing results are presented in table 4.1. The speed gain (online computations vs. FEM) is almost negligible, although the parallel pipeline is used. By and large, this is due to dense-dense matrix multiplication used, which strongly affects the algorithm's asymptotic computational complexity.

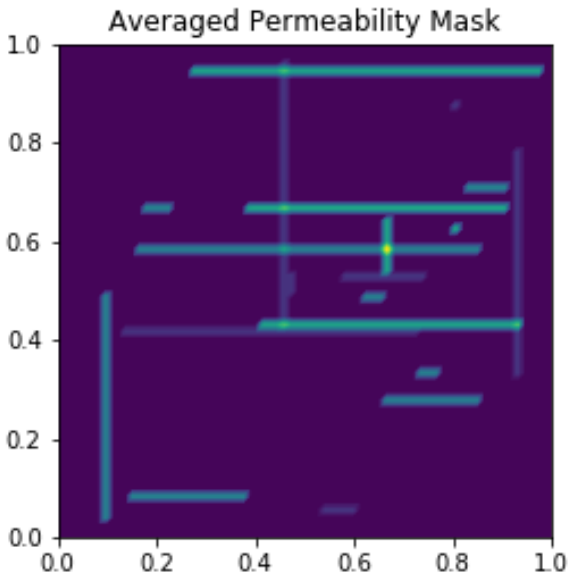

Figure 4.3: Experiment 2. Averaged permeability.
Three colors are in use. Inclusions may overlap forming a new color

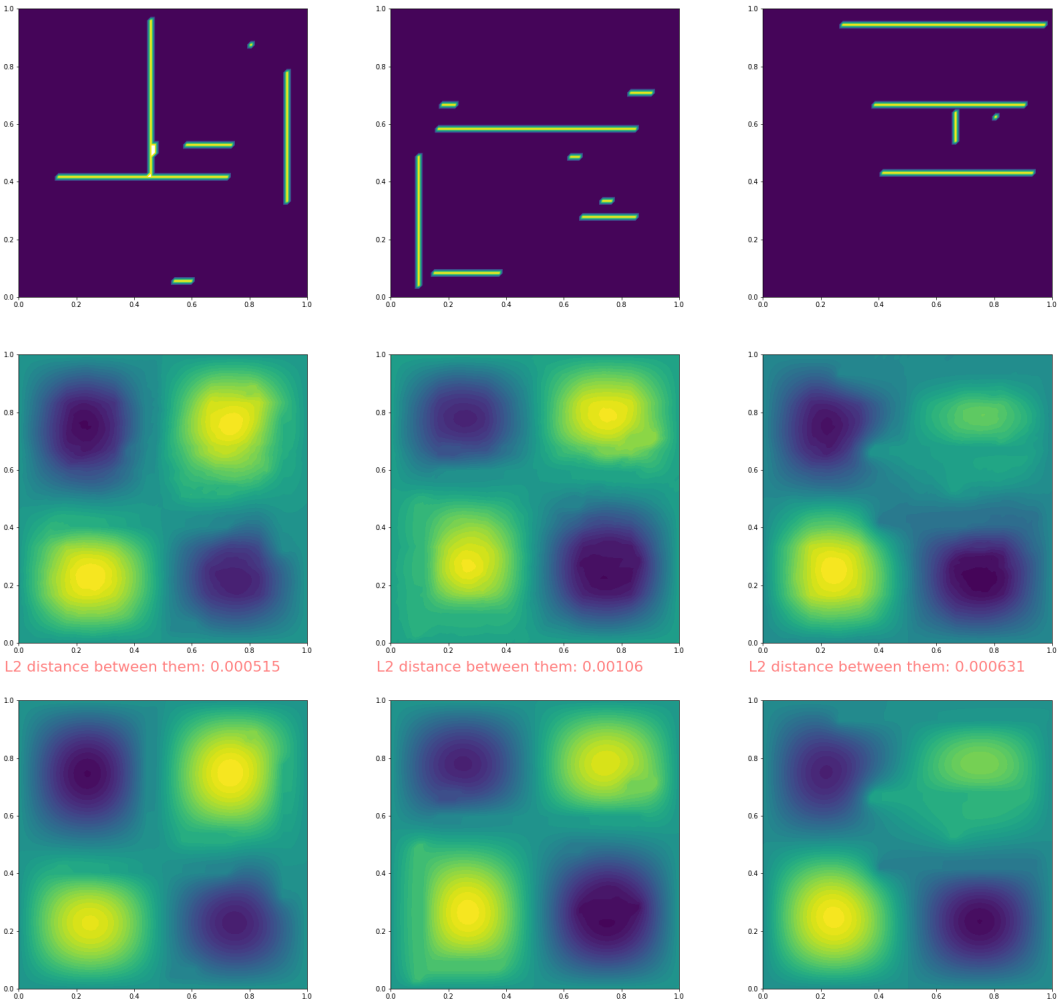

Figure 4.4: Experiment 2. Parametric problem
First row: permeability masks; **second row:** GMsFESolver results; **third row:** FEM solutions

	parallel GMsFEM	FEM
offline	8 s	–
online + GC	340 ms	16 s

Table 4.1: Experiment 2. Timing results
for parameter-dependent problem with $N_\mu = 3$.
The rest settings: $n_\omega = 64,\ n_{\ell_K} = 16,\ N_{\text{off}} = 20,\ N_{\text{on}} = 8$.
Dense-dense matrix multiplication (a.k.a. DDP - see tables 4.3,4.4)

4.2 Complexity Analysis

To understand the potential of GMsFEM, we estimate the complexity of each method's stage. We take into account the band structure of matrices assembled from fine-grid functions of V_h, the cost of matrix multiplication in case of dense-dense and dense-sparse arguments *. *(In the table cells, if the type is not specified, DDP complexities are displayed)* It is supposed to use an iterative solver for large linear systems, and direct methods - for small ones.

CT	communication time
$\mathcal{S}$	communicator size
$T_{\text{tag}}(N, \mathcal{S})$	CT of tag operation with N parties involved
*tag:*G	gather
*tag:*SR	send-recv redistribution
w	matrix bandwidth
DDP	dense-dense matrix product
DSP	dense-sparse matrix product

Table 4.2: Notation exposed in Complexity Analysis section

Space	Operation	Complexity per Call	# Calls
Snapshot	System assembling	$O(\text{w}(2n_{\ell_K}+1)^2)$	N_μ
	Apply BC and solve the system	$O(\text{w}^2(2n_{\ell_K}+1)^2)$	$8n_{\ell_K}N_\mu N_\omega$
	Step cost	$\mathbf{O(w^2 N_\mu N_\omega N_{\ell_K}^{1.5})}$	
Offline	Matrix assembling	$O(\text{w}(2n_{\ell_K}+1)^2)$	$2N_\omega$
	Matrix multiplication (e.g., $RM, SR^\top$)	$O(8n_{\ell_K}(2n_{\ell_K}+1)^4)$	$2N_\omega$
	Finding N_{off} dominant eigenpairs	$O(N_{\text{off}}(8n_{\ell_K})^2)$	N_ω
	Step cost (DDP)	$\mathbf{O(N_\omega N_{\ell_K}^{2.5})}$	
	Step cost (DSP)	$\mathbf{O(N_\omega N_{\ell_K}^{2})}$	
Online	Matrix assembling	$O(\text{w}(2n_{\ell_K}+1)^2)$	$2N_\omega$
	Matrix multiplication (e.g., $RM, SR^\top$)	$O(N_{\text{off}}(2n_{\ell_K}+1)^4)$	$2N_\omega$
	Finding N_{on} dominant eigenpairs	$O(N_{\text{on}}N_{\text{off}}^2)$	N_ω
	Step cost (DDP)	$\mathbf{O(N_\omega N_{\text{off}} N_{\ell_K}^{2})}$	
	Step cost (DSP)	$\mathbf{O(N_\omega N_{\text{off}}^2 N_{\ell_K})}$	

Table 4.3: Complexities of constructing spaces in serial

Block	Operation	Complexity per Call	# Calls
Diagonal	Matrix and vector assembling	$O(\mathrm{w}(2n_{\ell_K}+1)^2)$	N_ω
	Matrix multiplication	$O(N_{\mathrm{on}}(2n_{\ell_K}+1)^4)$	$2N_\omega$
	Step cost (DDP)	$\mathbf{O(N_\omega N_{on} N^2_{\ell_K})}$	
	Step cost (DSP)	$\mathbf{O(N_\omega N^2_{on} N_{\ell_K})}$	
Off-diagonal	Assembling of top/right	$O(\mathrm{w}(n_{\ell_K}+1)(2n_{\ell_K}+1))$	$2(N_\omega - n_\omega)$
	Multiplication with top/right	$O(N_{\mathrm{on}}[(n_{\ell_K}+1)(2n_{\ell_K}+1)]^2)$	$2(N_\omega - n_\omega)$
	Assembling of top-right/top-left	$O(\mathrm{w}(n_{\ell_K}+1)^2)$	$2(N_\omega - 2n_\omega + 1)$
	Multiplication with top-right/top-left	$O(N_{\mathrm{on}}(n_{\ell_K}+1)^4)$	$2(N_\omega - 2n_\omega + 1)$
	Step cost (DDP)	$\mathbf{O(N_\omega N_{on} N^2_{\ell_K})}$	
	Step cost (DSP)	$\mathbf{O(N_\omega N^2_{on} N_{\ell_K})}$	

Table 4.4: Complexity of the global coupling procedure in serial

Stage	Complexity				
	Serial		Parallel		
	DDP	DSP	DDP	DSP	CT
Snapshot	$O(N_\mu N_\omega N_{\ell_K}^{1.5})$		$O(N_{\ell_K}^{1.5})$		$+\ T_{\mathrm{G}}(N_\mu, \mathcal{S})$
Offline	$O(N_\omega N_{\ell_K}^{2.5})$	$O(N_\omega N_{\ell_K}^2)$	$O(N_{\ell_K}^{2.5})$	$O(N_{\ell_K}^2)$	$+\ T_{\mathrm{SR}}(N_\mu, \mathcal{S})$
Online	$O(N_\omega N_{\mathrm{off}} N_{\ell_K}^2)$	$O(N_\omega N_{\mathrm{off}}^2 N_{\ell_K})$	$O(N_{\mathrm{off}} N_{\ell_K}^2)$	$O(N_{\mathrm{off}}^2 N_{\ell_K})$	$+\ T_{\mathrm{SR}}(4, \mathcal{S})$
Global Coupling	$O(N_\omega N_{\mathrm{on}} N_{\ell_K}^2)$	$O(N_\omega N_{\mathrm{on}}^2 N_{\ell_K})$	$O(N_{\mathrm{on}} N_{\ell_K}^2)$	$O(N_{\mathrm{on}}^2 N_{\ell_K})$	$+\ T_{\mathrm{G}}(N_\omega, \mathcal{S})$
Direct Solver	$O(N_\omega N_{\mathrm{on}}^2)$				

Table 4.5: Complexity analysis summary of GMsFEM

Operation	Complexity per Call	# Calls
System assembling	$O(\mathrm{w}N_\omega^2(2n_{\ell_K}+1)^2)$	1
Iterative solver call	$O(\mathrm{w}N_\omega^2(2n_{\ell_K}+1)^2)$	k
Overall	$O(k\mathrm{w}N_\omega^2 N_{\ell_K}) \rightarrow \mathbf{O(N_\omega^2 N_{\ell_K})}$	

Table 4.6: Complexity of a standard FEM

As a rule, N_{on} is small (experimentally confirmed that the online space is well-compressible), so we can discard in table 4.5 terms containing it. In this way, we can conclude the speedup which GMsFEM gives w.r.t. FEM strongly depends on the compression of the offline space and the number of coarse blocks. In the extreme case: when the implementation is serial, the compression is imperceptible, and DDP is used, GMsFEM is always slower than FEM - $\frac{t_{\mathrm{GMsFEM}}}{t_{\mathrm{FEM}}} = N_{\ell_K} \frac{N_{\mathrm{on}}+N_{\mathrm{off}}}{N_\omega}$ Parallel implementation on N_ω cores and accounting of sparse matrix structure (mass and stiffness) boost this ratio to $\frac{N_{\mathrm{off}}^2+N_{\mathrm{on}}^2}{N_\omega^2}$ (without considering the cost of communication operations) It is worth mentioning that a regular FEM can be parallelized as well. However, there is definitely less freedom in its parallelization.

Chapter 5

Conclusion

In this paper, we implement a parallel version of GMsFEM and test its performance in the cluster environment. We study the time complexity of the underlying algorithm and compare it with the conventional approach - FEM. The analysis shows that GMsFESolver has a high degree of parallelism and can benefit greatly from being run on a sufficient number of cores. We indicate that the serial implementation can beat a regular FEM in tasks with repetitive calculations only if one considers all optimization steps in the former's development. For the problems used in the experiments, the parallel scheme simulates a solution with good accuracy and builds the online space within a short time. Global coupling proceeds with an adequate speed in P1 space. Care needs to be exercised when constructing the snapshot space. It may come that some functions thereof are dependent, so the space is difficult to compress. One can mend this issue with a proper discretization; or by adding regularization terms to the stiffness and mass matrices.

Bibliography

[1] Thomas Y. Hou Yalchin Efendiev, Juan Galvis. Generalized multiscale finite element methods (gmsfem). *Journal of Computational Physcis*, 2013.

[2] Viet Ha Hoang Jun Sur Richard Park, Siu Wun Cheung. Multiscale simulations for upscaled multi-continuum flows. *Journal of Natural Gas Science and Engineering*, 2019.

[3] Robert C. KirbyAnders LoggMarie E. Rognes. Common and unusual finite elements. In *Automated solution of differential equations by the finite element method. The FEniCS book*. Springer, 2012.

[4] Л.С. Кондратьева (составитель). *Основы метода конечных элементов (конспект лекций)*. Владимирский гос. институт, 2007.

[5] L. Ridgway Scott Susanne C. Brenner. *The Mathematical Theory of Finite Element Methods*. Springer, 1994.

[6] Xiao-Hui Wu Yalchin Efendiev, Juan Galvis. Multiscale finite element methods for high-constrast problems, using local spectral basis functions. *Journal of Computational Physcis*, 2011.

[7] Wing Tat Leung Eric T. Chung, Yalchin Efendiev. Residual-driven online generalized multiscale finite element methods. *Journal of Computational Physcis*, 2015.

[8] Maria Vasilyeva Donald L. Brown. A generalized multiscale finite element method for poroelasticity problems ii: Nonlinear coupling. *Journal of Computational and Applied Mathematics*, 2015.

[9] Dongwoo Sheen. *Introduction to the conforming and nonconforming finite element methods*. Kyoto University, 2015.

[10] Won-Tak Hong Hae-Soo Oh, June G. Kim. The piecewise polynomial partition of unity functions for the generalized finite element methods (ii). In *Computer Methods in Applied Mechanics and Engineering*. ELSEVIER, 2007.

[11] Sai-Mang Pun Eric Chung. Computational multiscale methods for first-order wave equation using mixed cem-gmsfem. *Journal of Computational Physics*, 2019.

[12] Michael Spevak. Finite element schemes. In *Dissertation on the Specification and the Assembly of Discretized Differential Equations*. Technical University of Vienna, 2008.

[13] In G. N. Wells A. Logg, K.-A. Mardal, editor, *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

[14] Bor Plestenjak Michiel E. Hochstenbach, Christian Mehl. Solving singular generalized eigenvalue problems by a rank-completing perturbation. *SIAM Journal on Matrix Analysis and Applications*, 2019.

[15] Guanglian Li Yalchin Efendiev, Juan Galvis. Generalized multiscale finite element methods oversampling strategies. *International Journal for Multiscale Computational Engineering*, 2013.

[16] Juan Galvis Eduardo Abreu, Ciro Diaz. A convergence analysis of generalized multiscale finite element methods. *Journal of Computational Physics*, 2019.

[17] Lijian Jiang Lingling Ma. Convergence analysis for gmsfem approximation of elliptic eigenvalue problems. *Journal of Computational and Applied Mathematics*, 2017.

[18] Stephen Roberts. Finite element method for numerically solving pde's. In *Instructional Workshop on Analysis and Geometry, Part 1*. School of Mathematical Sciences, A.N.U., 1996.

Appendices

.1 Convergence

Although convergence analysis is not the purpose of this study, it is a crucial point in all numerical methods. So, some of the materials on this matter are presented below. We do not reproduce here any results, since each of them is built on a bulky list of assumptions, lemmas, and hypotheses. Otherwise, it would be just a rote copying of the articles.

- GMsFEM
  - the first error estimates and mention of a decrease in the approximation error with coarse mesh refinement - [1].
  - the explicit expression for the error bounds showing dependence on the coarse mesh size and magnitude of the largest left-out eigenvalue[1] [16, 17]
- FEM: dependence on the mesh size and polynomial degree [18]

.2 DSP

Dense-sparse matrix product (for illustrative purposes only)

Algorithm * Outline of DSP

A is a dense matrix, (n, n)
B is a band $n \times n$-matrix, re-written in the form (w, n)
with padding where needed
C is the result of their product

```
for i := 1,n do
  for j := 1,n do
    for k := 1,w do
      C[i,j] = A[i,j+k]B[k,j]
    end for
  end for
end for
```

Easy to see, the complexity is $O(\mathrm{w}n^2)$

[1] the largest or the smallest eigenpairs depends on how the GHEP is defined:
* $Mx = \lambda Sx$ - the dominant ones
* $Sx = \lambda Mx$ - the smallest

- always check the adopted notation