Porous Medium Flow Simulations using Massively Parallel MLMC algorithm

Nikolay Shegunov

From the faculty of *Mathematics and Informatics*, Sofia University, a thesis presented for the degree of Doctor of Philosophy in Informatics

Supervised by: assoc. prof. dr. Petar Armyanov, Sofia University.

Professional field: 4.6 Informatics and Computer Sciences, doctoral program "Information Systems"

2021

Acknowledgements

Firstly, I would like to thank my supervisor assoc. prof. dr. Petar Armyanov, Sofia University, and my scientific advisor prof. dr. Oleg Iliev from Kaiserslautern university, for their support during the entire time of my Ph.D. Furthermore, I would like to thank Fraunhofer ITWM and the university of Kaiserslautern for their financial support during the first years of my work. I am grateful for the provided computational resources on the HPC Beehive cluster hosted at ITWM Fraunhofer and SuperMuc supercomputer located at Technical University Munich. Without their support, my work would be impossible. Finally, I would like to thank my colleges from ITWM and Sofia University for the pleasant working environment.

Contents

Chapter 1

Introduction

1.1 Motivation

Uncertainty is a part of many contemporary scientific models. With a growing demand for more accurate and predictive models, stochastic modeling is a rapidly growing area in applied mathematics and scientific computing. To name just a few applications, consider stochastic gradient descent methods - an approach well adopted in neural networks and machine learning. Another area where uncertainty quantification is adopted, is computational finance - in the pricing of financial derivatives and quantitative risk management. Uncertainty Quantification (UQ) is also well established in many engineering and science models. It has led to new SIAM journals and associated annual conferences [1]. There are even areas of research, that would be impossible without stochastic modeling. Porous media flow modeling is great example of such. Those models are of great importance for many societal, environmental, and industrial problems, such as drug delivery, metal composite materials, radioactive waste modeling, filtration process and many more. As stochastic models, simulations of flow in porous medium requires an extreme computational power to obtain reliable simulations. Hopes are, that with the arrival

of *exa*-capable computers the research of that particular area will be boosted. New numerical simulation algorithms, capable to utilize the computational power of the coming High-Performance Computing (HPC) systems will be needed. To answer this demand, several different numerical algorithms have been developed, each of which has pros and cons. A well-established algorithm that can utilize the parallel capabilities of the modern HPC systems, is the well known Monte Carlo (MC) class of algorithms. Those computational methods rely on repeated random sampling to obtain numerical results. This simple idea is used in many models, where it is difficult to impose some kind of physical limitations. It is even applicable to deterministic models, where one can introduce randomness into the model to solve it. Monte Carlo algorithms are important in computational physics, physical chemistry and related applied fields. Employed in industry, those algorithms are extremely useful for problems involving simulations of complex systems - such as studies of fluids, composite materials, strongly coupled solids cellular structures and others. Unfortunately, applying a Monte Carlo algorithm is not very practical for models involving computational fluid dynamics. In that field, the problems have extreme dimensions. For such problems faster methods are needed. To overcome this, a generalization of the MC method has attracted great scientific interest in the last two decades. The generalized method, called Multilevel Monte Carlo (MLMC), can achieve much faster simulation times compared to the classical MC method. Its first use was for parametric integration for the expected value of $f(x, \lambda)$ - $E[f(x, \lambda)]$ where x is a finite-dimensional random variable and λ is a parameter, by Heinrich, in the beginning of the 21-st century **[2, 3, 4]**. Nowadays it can be found in various applications **[1]**.

The idea of the algorithm is to divide the problem into different sub-problems called levels. Each level is characterized with computational cost. Those levels act as an approximations to the problem, that are much faster to compute than computing the original problem. Combined appropriately it leads to

significant computational cost reduction. That novel algorithm comes with a variety of interesting problems. Construction of the algorithm requires proper arrangement and definition of the levels. Another important aspect is the parallelization of the algorithm. It turns out that in its most general form the optimal scheduling of the MLMC is an NP-complete problem **[5]**.

In this work the Multilevel Monte Carlo algorithm is considered for porous medium flow simulation. For such problems, both proper level arrangement and the optimal scheduling strategy are considered. The aim is to provide an efficient parallel variant of that algorithm. As such the algorithm can be used for realistic simulation with the help of High-Performance Computing (HPC) systems.

Simulation of a flow in a porous medium requires solving Stochastic Partial Differential Equations (SPDEs). Those differential equations extend the idea of partial differential equations, by incorporating uncertainty in the model, for example as an input parameter or as a coefficient parameter. Those uncertainties are modeled as stochastic processes with a probability distribution, from witch samples can be drawn. The expected value of some random variable can be approximated by taking the empirical mean (the sample mean) of independent samples of the variable. Quantifying the uncertainty leads to remarkably large dimensional computational problems.

In principle, MC methods can be used to solve any problem having a probabilistic interpretation, not just SPDEs. The expected value of some random variable can be approximated by taking the empirical mean (the sample mean) of independent samples of the variable.

1.2 Outline of this work

In chapter **2** the necessary mathematical foundations needed for the simulations are presented. The general form of MLMC is given, including analysis

of its computational cost. In chapter **3** a simple stochastic Laplace equation model is presented. This stochastic partial differential equation is well established as a model equation in the field of uncertainty quantification in porous medium flows and shows well the computational and modeling challenges of applying Multilevel Monte Carlo algorithms. Chapter **4** considers a practical example of transport of mass within a porous media. The formulated equation has much higher computational complexity than the Laplace equation. It is widely used and has many application areas as building block in more complex models. The final chapter **5** considers the programming aspect of the problem. In it the algorithm procedure for computation is formulated and different scheduling strategies are considered capable of utilizing a large number of processor cores. The chapter ends with experiments and analysis of the proposed algorithms. Each chapter finishes with a conclusion section, consisting of a summary with the key points in the chapter.

1.3 Stochastic computations

Many real world problems are subject to uncertainty due to some kind of limitation such as, but not limited to, physical phenomena, expensive measurements, or inability to obtain data. Uncertainty can be included into mathematical models and experimental measurements in various contexts: **parameter uncertainty** - which comes from the model parameters that are inputs to the computer model; **structural uncertainty**; **model inadequacy** - which comes from the lack of knowledge of the underlying physics describing the problem. Extensive efforts have been devoted to characterization and reduction of uncertainties in the models. A small change in the data can lead to significant changes in the solution. An illustrative example is a simple viscous Burger' equation, where a small change in the uncertainty leads to a dramatic change of the solution **[6]**. In the context of differential equation models more formally, one can consider a physical domain with $D \subset R^d, d \in \{1, 2, 3\}$, with boundary ∂D and $x = (x_1, \dots x_d)$.

Consider a PDE operator L with boundary condition operator B:

$$L(x, u; y) = 0, \text{ in } D.$$
$$B(x, u; y) = 0, \text{ in } \partial D.$$

Here $y = (y_1, \dots y_N)$ are parameters of interest. Assume they are mutually independent. In practice one may be interested in a set of quantities $g = (g_1, \dots, g_k) \in R^k$, called *observables*, that are functions of the solution u. To model the uncertainty a probabilistic framework is adopted and $y = (y_1, \dots y_N)$ is modeled as a $N - variate$ random vector with independent components in a properly defined probability space (Ω, F, P). Ω is the sample space, $F \subseteq 2^\Omega$ is the σ-algebra and $P : F \to [0, 1]$ probability measure function. Defining $\rho_i : \Gamma_i \to R^+$ to be probability density function of a random variable $y_i(\omega)$ where $\omega \in \Omega$ has image $\Gamma_i := y_i(\Omega), \subset R$ for $i = \{1 \dots N\}$. Then joint probability density function of the random vector y is:

$$\rho(y) = \prod_{i=1}^{N} \rho_i(y_i), \text{ where } \Gamma := \prod_{i=1}^{N} \Gamma_i.$$

If all random variables have the same support then the finite-dimensional probability space Γ is a hypercube: $(-1, 1)^N$, $(0, \infty)^N$, R^N **[6]**.

Before moving to simulations of such stochastic systems, despite of the numerical methods used to solve the problem, it is necessary to properly identify the random variables y, such that data uncertainty is accurately modeled. An essential matter is to parameterize the input uncertainty by a set of a finite number of random variables. A popular choice for parameterization is Karhunen-Loeve expansion. That method seeks to represent the Gaussian process by a Gaussian random variable in a series, very similar to the Taylor series. Despite this, each turn includes an independent random variable

and hence an additional dimension [7]. Besides, Karhunen-Loeve expansion requires solving a Fredholm integral of a second kind, which leads to a dense matrix linear system. This is very expensive to compute. Possible other means of representation is by employing forward and inverse fast Fourier transforms over a circulant matrix [8], or to use Cholesky decomposition [9]. Another scheme, that attracts great attention in recent years, is to model the stochastic field by generated solutions to stochastic PDEs [10] under certain assumptions about the field. In the past years the attention to this topic is increasing. Complex systems, where models can serve only as a reduced representation of true physics, revealing quantitative connections between predictions and observations, are constrained by the ability to accurately assign values to various parameters in the governing equations. Subsequently, after the uncertainty is parameterized, several main classes of numerical methods exist to solve the SPDE model:

- **Perturbation methods.**
  This is the most popular non-sampling method, where the random fields are expanded via the Taylor series. The random fields are expanded around their means and are truncated at a given order. The order of magnitude of the uncertainties cannot be too large - typically at most second order is employed, because resulting system becomes extremely complicated [6, 11, 12].

- **Moment equations.**
  In this approach the effort is to find the moments of the stochastic solution directly. The unknowns are the stochastic moments of the solution and their equations. The challenge lies in the fact that the derivation of a moment almost always excepts knowledge of higher stochastic moments. This brings out the so-called "closure" problem, which is often dealt with by utilizing some "ad-hoc" arguments on the properties of the higher moments [6].

- **Generalized polynomial chaos methods.**
  In this approaches stochastic solutions are expressed as orthogonal polynomials of the input random parameters - a generalization of the classical polynomial chaos methods **[13]**. Here different polynomials can be chosen, for a particular problem, in order to achieve better convergence.

- **Sampling based methods.**
  Commonly used methods in this category are Monte Carlo sampling or one of its variants. Here the approach is to generate an independent realization of random inputs, based on the prescribed probability distribution function. For the generated random realization, data becomes fixed and the problem becomes deterministic. Upon solving the deterministic realizations one collects an ensemble of solutions and statistical information is extracted. Simple brute force MC algorithm convergences very slow. To overcome this limitation a number of standard approaches have been developed e.g. Latin hypercube sampling or Quasi-Monte Carlo **[14, 15, 16, 17, 18, 19]**

1.4 Existing software for high-performance systems

Concurrent programming is a form of computation in which a set of tasks are computed at the same time. In contrast to sequential computing, where each task is computed before the next starts, in the concurrent world two tasks may be computed at overlapping time frames. The main idea of concurrent programming is to divide the problem into small sub-tasks that can be computed concurrently. Concurrent programming is strongly coupled with parallel programming, where the different sub-tasks are computed in the same time frame. Concurrent models are a well-established way for computation. Different concurrent logical models have been developed. Remarkable ones are the actor model in which the main unit is the actor. Each

actor can receive messages and act accordingly to them. The communication between the different actors is done indirectly through messages. Another popular model is the client-server model, successful in network applications. In the transaction model, the operations consist of reads and write. They are atomic and at each point in time the state validity of the system must be ensured. Those designs are typically applied in the context of parallel machines. By definition parallel machine is a machine that combines a set of processors that can work cooperatively to solve a computational problem. This definition is broad enough to include parallel supercomputers, that have hundreds or thousands of processors in networks of workstations; multiple-processor workstations, and embedded systems. Nowadays the development of parallel algorithms has been motivated not only for scientific purposes but also for industrial, thus the demand for more efficient and scalable algorithms is even greater. This is an area where computer science is behind the hardware capabilities. Mote in-depth discussion of the different concurrent models can be found in: **[20, 21, 22, 23]**. There are many ways one can utilize the hardware and thus different parallel concepts: **vector computing** (vectorized instructions), **multi-core** (several CPU cores), cache-coherent Non-Uniform Memory Access **(ccNUMA)** , **GPU** (fast parallel vector processing). In general there are three main memory models, depending on how memory is accessed by the processors: **multiprocessors** (shared memory), **distributed memory** (multi-computers), **hybrid architectures** (multi-computers + multiprocessors).

Architecture overview

The main difference between the architectures is how the memory and the CPUs interact with each other. Figure **1.1**, represents a schematic overview of the shared memory model. Each CPU has equal access to the whole available memory and each process can read and write to any given memory location. In programs, utilizing this model, a data race for a given resource can

occur at any time, therefore the main difficulty in the shared memory model is how to synchronize the access of the shared objects. In this model, a Uniform Memory Access (**UMA**) is possible with Symmetric Multi-Processing (**SMP**)

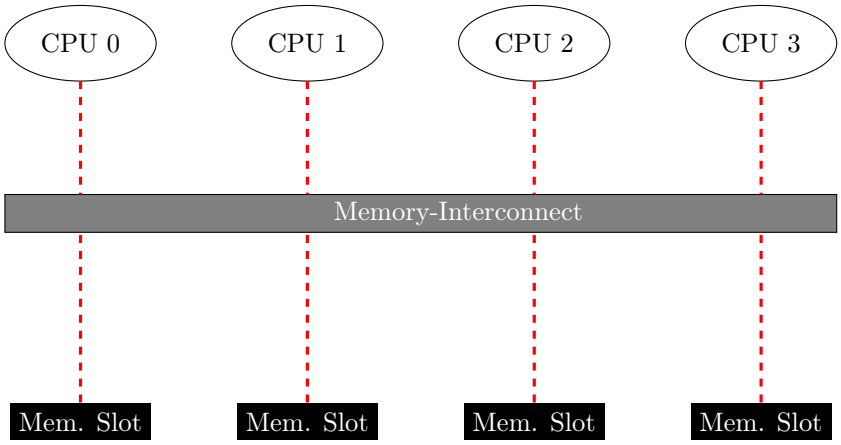


Figure 1.1: Shared memory model

On figure **1.2** a multi-computer model (distributed memory) is presented. In this model, nodes are coupled by a node interconnect and each CPU has fast access to its local memory and slow access to the remote memory. This architecture has Non-Uniform Memory Access (**NUMA**). The speed at which a CPU can access a remote memory is strongly dependent on the network topology used. There are many different network topologies. As an example of such topologies, consider islands of nodes with relatively fast memory access among them and relatively slow access to the memory between the nodes (the different islands are physically far one from another).

Figure **1.3** shows a hybrid model - nodes with distributed and shared memory architecture. This model is also called the cache-coherent Non-Uniform Memory Access (ccNUMA) model. This architecture allows fast access to the data in local memory and slower access to the data in remote memory. Shared memory programming is possible. Each node has a specialized memory controller, and the memory controllers in all the nodes cooperate using directory techniques to maintain cache-coherence across the system.

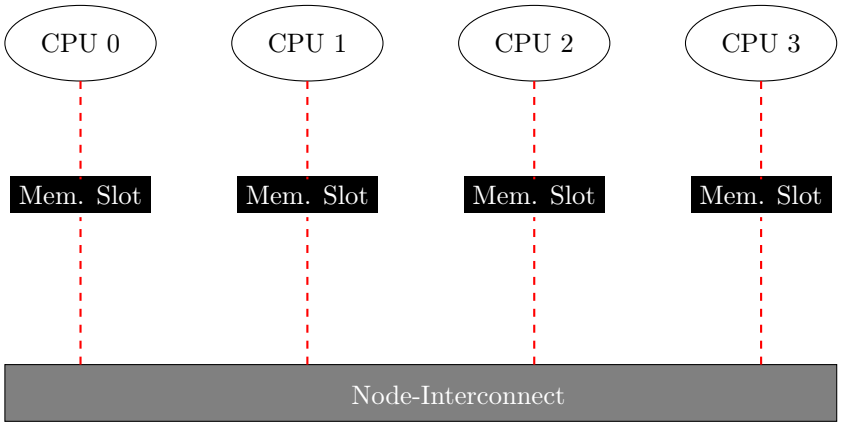


Figure 1.2: Distributed memory model

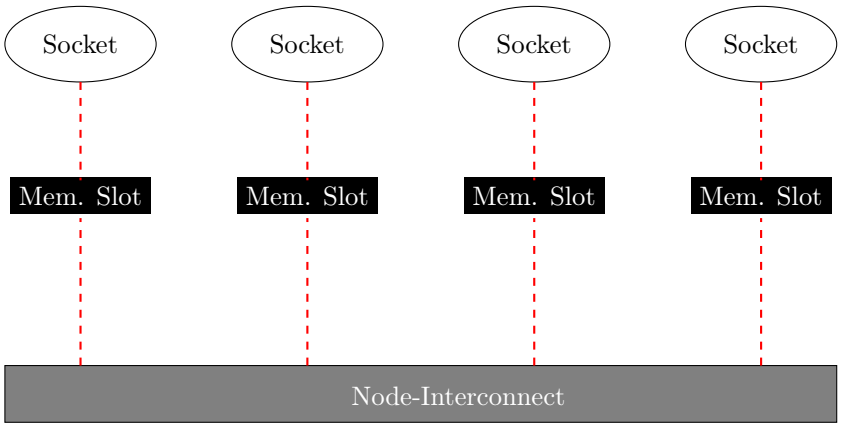


Figure 1.3: Distributed Memory with shared memory

Finally on figure **1.4** a scheme of a hybrid memory model is shown. Most modern-day **HPC** systems utilize this model. It consists of a symmetric multi-processing inside each node and distributed memory parallelization on the node interconnect. Such architectures can be viewed as a collection of standard modern multi-core computer systems, physically close to eachother and connected by high-performance *Ethernet* type network, typically *InfiniBand*, that allows very high throughput and very low latency.

Logical model of HPC systems

The main hardware resources, that programs have to utilize, are the processor, memory and potentially graphical accelerators, that become popular

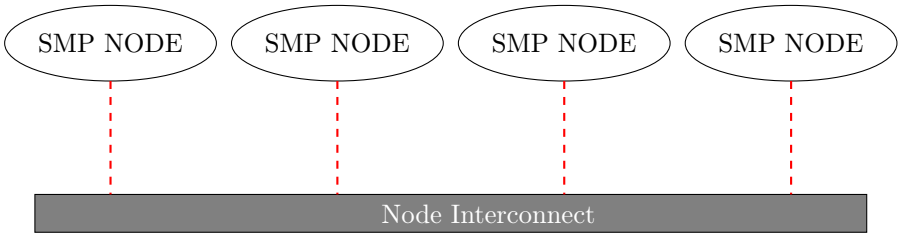


Figure 1.4: Hybrid Architectures

recently. This means a parallel program can distribute work across different processors and graphical accelerators and distribute data between different nodes, but also between the host memory (main memory) and device memory (graphical memory) and synchronize and communicate locally within a single symmetric multi-processing (SMP) node or globally between different SMP nodes. To address these capabilities different programming frameworks have been developed. The Message Passing Interface (MPI) is a standardized actor-like model interface, that uses a message-passing standard to organize communication between nodes **[24]**. It is designed to function on a wide variety of parallel computing architectures. The standard defines the syntax and semantics of library routines. Most notable implementations include Open MPI and Intel implementation of the standard. As a communication protocol it offers both point-to-point and collective communication. The first MPI version has no shared memory concept, and MPI-2 offers only a limited distributed shared memory concept. However, Open MPI implementation, can use shared memory for message transfer if it is available **[25]**. Explicit shared memory programming is available in the third reversion of the MPI standard **[26, 27, 28]**. The MPI is meant to provide essential virtual topology, synchronization and communication functionality between a set of processes, that are mapped to nodes. Those processes are grouped in communicators. Each process can communicate with any other process within a communicator. Processes from different communicators can not communicate.

Libraries for scientific computing

There exist a large number of specialized libraries for linear algebra, as it is an important part in many models, not only limited to HPC such as: *LAPACK, Eigen, Armadillo, SuperLU* . . . , each with own strength and weakness. However, few open-source, general purpose libraries exist, that are suitable for massively parallel problems. In the past few years a rapid development have been done towards such libraries aimed for HPC systems, with high scalability factors, such as *Deal.ii*[1] and *Dune*[2] libraries **[29, 30]** , *OpenFoam* (specialized in Computational Fluid dynamics)[3], *Portable, Extensible Toolkit for Scientific Computation* (PETSc)[4]. Dune library is a modular toolbox for solving partial differential equations, using grid base methods. The underlying idea of Dune is the creation of interfaces, allowing the use of legacy and/or new libraries. It is written in *C++*, with heavy use of template meta-programming techniques and feathers from the **C++0x** standard family. It is also capable of supporting shared memory as well as distributed memory computations.

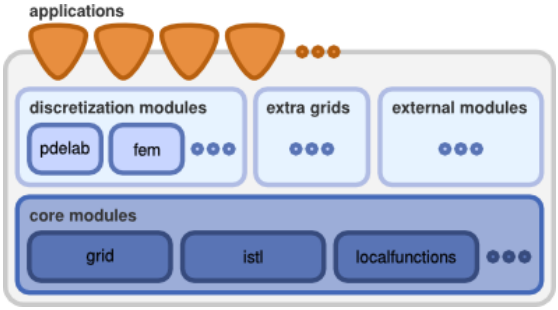


Figure 1.5: Dune design structure **[31]**

The library consists of core modules and modules build on top of them, as illustrated in figure **1.5**. It consists of five core modules, each with a specific purpose. **Dune-common** is the base model for all other modules. It defines internal structures, some interfaces for MPI wrapper, time measure-

[1] http://dealii.org/
[2] https://www.dune-project.org/
[3] https://www.openfoam.com/
[4] https://www.mcs.anl.gov/petsc/

ments, build system, etc. **Dune-geometry** provides interfaces for different geometries, iterators, and so on. **Dune-grid** represents the grid manager. It supports structured and unstructured grids, conforming, non-conforming grids etc. **Dune-localfunctions** provides interface and implementation for shape functions, defined on Dune reference elements. **Dune-istl** or Dune iterative solver template library, represents different linear algebra abstractions as vectors, matrices, sparse vectors and sparse matrices, and different linear solvers: sequential or parallel. On top of the core, modules exist higher-level libraries. One of the commonly used modules is *Dune-PdeLab*. It is aimed to provide easy and fast way of implementing a numerical methods for PDE. It is based on the residual assemble approach. The user provides an implementation on a reference element which is then mapped to a global residual matrix. It uses Dune-istl as a linear algebra backend. In general, the library is based on the idea of static polymorphism design pattern with template specialization and type propagation (type traits as std::numeric_limits). The idea of such representation is better optimization of the code, done by the compiler since there are no virtual tables and the exact type is known at compile-time which enables the compiler to produce more optimized code. However, such an approach is not without drawbacks. The code is hard to read, understand and support.

1.5 Aim and scope of this work

In this work the Multilevel Monte Carlo algorithm is considered to address to problems involving porous medium flows. The aim is to:

- Provide an efficient general purpose parallel variant of the algorithm, capable of much faster simulations compared to the classical Monte Carlo approach setting. The algorithm must consider the sample to sample time fluctuations raised by the randomness of the problem.
- Provide an efficient parallel computation scheme, in order to enable realistic simulations of porous medium flow, with the help of high-performance computing systems.

To achieve that goal the following tasks are considered:

- Explore the existing approaches to the problem.
- Choose effective algorithm for generation of random porous medium permeability fields.
- Provide an efficient coarsening strategies for the MLMC setting.
- Provide an adequate heuristic algorithm for efficient work scheduling.
- Choose appropriate software for effective implementation in HPC setting.

Chapter 2

Mathematical Models

This chapter considers the underlying mathematical tools needed for constructing the computational methods given in chapters **3** and **4**. Section **2.1** elaborates on the random number sampling from random fields. The section describes the idea behind the circulant embedding algorithm. Section **2.2** discusses the method used for discretization of the PDE. Next section **2.4** formulates the Multilevel Monte Carlo algorithm in its general form and compares it to the classical Monte Carlo approach. Section **2.3** discuses two important properties of porous medium, that are later incorporated in to the models. The chapter concludes with short summary section **2.5**.

2.1 Random Sampling Algorithms

An essential problem in uncertainty quantification applications is how to generate a coordinated random sample by a given covariance matrix from a multivariate random distribution. Different approaches have been developed for tackling this problem. Before the sampling algorithms are described, rewind that for two joint distributed random variables X, Y the *covariance* is defined as:

$$cov(X,Y) = E[(X - E[X])(Y - E[Y])] = E[XY] - E[X]E[Y]$$

The covariance is a measure of the joint variability of two random variables. Depending on how the random variables are distributed, the covariance may be positive or negative. The closely related statistical measure *correlation* is defined as:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_x \sigma_y} = E[(X - E[X])(Y - E[Y])]/(\sigma_x \sigma_y)$$

The correlation is a measure of how two random variables are related one to another, i.e. how a change in the value of one of the variables causes a change in the value of the other variable. It is very useful measure of statistical relationship and is used for studying and simulation in many statistical models. Both measures can be generalized to N dimensional multivariate distribution. Let $X = (X_1, X_2, \dots X_n)^T$ is a column vector of random variables, each with finite variance and expected value. Then the covariance matrix is defined as a matrix with entries $\Sigma_{i,j} = cov(X_i, X_j)$ or:

$$\Sigma = \begin{bmatrix} E[(X_1 - E[X_1])(X_1 - E[X_1])] & \cdots & E[(X_1 - E[X_1])(X_n - E[X_n])] \\ E[(X_2 - E[X_2])(X_1 - E[X_1])] & \cdots & E[(X_2 - E[X_2])(X_n - E[X_n])] \\ \cdots & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ E[(X_n - E[X_n])(X_1 - E[X_1])] & \cdots & E[(X_n - E[X_n])(X_n - E[X_n])] \end{bmatrix}$$

The correlation matrix is defined as the matrix with entries $cov(X_i, X_j)/\sigma_{x_i}\sigma_{x_j}$. It is clear that in order to generate correlated random variables, the covariance matrix is needed. The idea is to generate uncorrelated random variables and to find a linear transformation that correlates them, i.e. find a linear transformation, such that the covariance matrix is diagonalized. To achieve that, consider a random N-dimensional vector consisting of independent ran-

dom variables with zero mean and variance of 1. Since $E(X_i X_j) = \delta_{ij}$, hence the covariance matrix:

$$\Sigma = E(XX^T) = I.$$

The random vector then can be correlated with covariance matrix Q, by decomposing the matrix Q in the form $Q = LL^T$. This can be done using the Cholesky decomposition of Q, since by definition the covariance matrix is positive definite and symmetric, setting the new random vector $Z = LX$ and considering the expansion $E(ZZ^T)$ **[32]**:

$$E((LX)(LX)^T) = E(LXX^T L^T) = \underbrace{LE(XX^T)L^T}_{\text{E is linear operator}} = LIL^T = Q$$

Given the $Q = LL^T$ decomposition of the covariance matrix, generating random samples is straightforward. Generate uncorrelated random vector X and obtain the random vector $Z = LX$. More information can be found in **[33, 34, 35, 36]**. From a computational standpoint, this method can be applied to all types of covariance matrices. However, an obscure problem with covariance matrices is the significant memory requirements. The required memory to store the whole covariance matrix grows extremely fast and generation of $N \times N$ correlated random variables requires $N^2 \times N^2$ covariance matrix. Under the assumption, that the points that needs to be generated are equally spaced correlated random variables, the covariance matrix size can be reduced and more efficient algorithm can be used.

Assume a random field, with constant mean and covariance $C(x, y) = c(x - y),\ x, y \in D$. The function $c : D \to \mathbb{R}$, is known as a stationary covariance and the field itself is called stationary. Let $x_1, x_2, \ldots, x_M \in D$ be the chosen

sample points. Realisations of discrete random fields associated with the Gaussian random filed $Z(x,\omega)$, can be generated using circulant embedded algorithm if $Z(x,\omega)$ is stationary and $x_1, x_2, \ldots, x_M$ are uniformly spread. The necessary mathematical tools to describe circulant embedded algorithm are complex-valued random variables, Fourier transform, and Toeplitz and Circulant matrices. Before the algorithm is sketched, the necessary results from the literature are given. For more detailed discussion, refer to **[33, 34]**.

Definition *complex multivariate Gaussian distribution*: A $\mathbb{C}^M$ random variable $Z = X + iY$ follows the complex multivariate Gaussian distribution $CN(\mu, C)$ with $\mu \in \mathbb{C}^M$ and $C \in \mathbb{R}^M \times \mathbb{R}^M$, if the real and imaginary part of Z are independent and

$$X \sim N(\mathrm{Re}(\mu), \frac{1}{2}C),$$
$$Y \sim N(\mathrm{Im}(\mu), \frac{1}{2}C)$$

The entries of the covariance matrix are $c_{i,j} = Cov(Z_i, Z_j) = E[(Z_i - E[Z_i])(\overline{Z_j - E[Z_j]})]$. Please note, that since the real and imaginary part of Z are intendant the values of $c_{i,j}$ are real **[33, 34]**.

Definition *Toeplitz matrix:* An $M \times M$ real-valued matrix C is Toeplitz if $c_{i,j} = c_{i-j}$ for some real numbers $c_{1-M}, \ldots, c_{M-1}$.

$$C = \begin{pmatrix} c_0 & c_{-1} & \ldots & c_{2-M} & c_{1-M} \\ c_1 & c_0 & c_{-1} & \ldots & c_{2-M} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ c_{M-2} & \ddots & c_1 & c_0 & c_{-1} \\ c_{M-1} & c_{M-2} & \ldots & c_1 & c_0 \end{pmatrix}$$

A symmetric Toeplitz matrix has entries $c_{i-j} = c_{j-i}$ and is defined by its first column:

$$c_1 = [c_0, c_1, \dots c_{M-1}]^T \in \mathbb{R}^M.$$

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 2 & 3 \\ 3 & 2 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{pmatrix}$$

Figure 2.1: Example of symmetric Toeplitz matrix

Definition *Circulant matrix:* An $M \times M$ real-valued Toeplitz matrix C is circulant if each column is a circular shift of the elements of the previous column. Those matrices are fully defined by their first column.

$$\begin{pmatrix} 3 & 2 & 1 & 2 \\ 2 & 3 & 2 & 1 \\ 1 & 2 & 3 & 2 \\ 2 & 1 & 2 & 3 \end{pmatrix}$$

Figure 2.2: Example of Circulant symmetric matrix

Theorem: Let C be an $M \times M$ real-valued circulant matrix with first column c_1. Then

$$C = W \Lambda W^*$$

where W is the Fourier matrix and Λ is the diagonal matrix of eigenvalues with diagonal:

$$d = \sqrt{M} W^* c_1$$

If C is symmetric, c_1 is Hermitian vector and the eigenvalues are real **[33, 34]**.

Using the above theorem in case the covariance matrix is circulant, generating a pair of samples from $N(0, C)$ is very easy. Let $\xi = [\xi_1, \ldots, \xi_M]$ be a random uncorrelated sample, i.e. $\xi \sim CN(0, 2I)$, and consider $Z = W\Lambda^{1/2}\xi$. Since $E[Z] = E[\xi] = 0$ the covariance matrix becomes [33, 34]:

$$E[ZZ^*] = W\Lambda^{1/2}E[\xi\xi^*]\Lambda^{1/2}W^* = W\Lambda^{1/2}2I\Lambda^{1/2}W^* = 2C$$

In addition, hence $\mathrm{Re}(Z), \mathrm{Im}(Z) \sim N(0, C)$ and are independent

$$E[ZZ^T] = W\Lambda^{1/2}E[\xi\xi^T]\Lambda^{1/2}W^* = 0$$

This means that to draw a pair of independent samples from $N(0, C)$, when C is circulant two independent samples needs to be computed from $N(0, C)$. Then factorize the matrix C using inverse discrete Fourier transform and perform matrix-vector multiplication with W - the discrete Fourier transform matrix. The matrix C is always symmetric and Toeplitz on uniformly spaced grid points [33, 34]. Any symmetric and Toeplitz matrix can be embedded into a larger symmetric circulant matrix, satisfying theorem conditions. As this special structure of the covariance matrix is uniquely defined by the first column of the matrix, the memory that is needed for the simulations is reduced.

2.2 Finite volume numerical method

Finite volume (FV) method is one of the most versatile methods. It was developed with the aim to discretize the equation that describes conservation laws coming from physics, frequently used in computational fluid dynamics [37, 38]. The simplest example of one dimensional conservation law is the following PDE:

$$q_t(x, t) + f(q(x, t))_x = 0 \tag{2.1}$$

The conservation laws arise most naturally in an integral forms, stating that for two points x_1, x_2 for equation **2.1**

$$\frac{d}{dt}\int_{x_1}^{x_2} q(x,t)dx = f(q(x_1,t)) - f(q(x_2,t)) \tag{2.2}$$

Each component of q measures the density of some conserved quantity and the equation **2.2** states that the total mass of this quantity between any two points can only change due to the mass flux past the endpoints x_1, x_2. If the total mass is not conserved then the equation **2.2** has to contain source terms, for example, in some chemical reactions taking place. There are many other such conservative systems and in practice many of them are non-linear, non-smooth and contains discontinuities. In that case equation **2.1** can be derived from **2.2**, provided that q and $f(q)$ are sufficiently smooth. This discontinuities lead to computational difficulties and discretization with the classical approaches, such as finite difference methods in which the derivatives are approximated directly by finite difference tend to break near this discontinuities. The resulting solution can even become physically incorrect, for example the solution can contain negative mass. For more in-depth approaches and methods for solving such systems and the computational challenges that they posses, consider the great books by Randall and LeVeque **[37, 38]**. The most compelling reason to use finite volume method - it is conservative. Instead of approximating the derivatives directly, FV approach considers the solution within a control volumes, called cells. Those volumes are usually polygons. The unknown solution is approximated by cell average over the control volume and the flux is balanced. Consider the multivariate form of equation **2.1** and reformulate the equation:

$$\begin{aligned} &\frac{\partial u}{\partial t} + \bigtriangledown . f(u) = 0, \ \textit{where } u \textit{ is conserved quantity and} \\ &f(u) = [f_1(u), \ldots, f_d(u)] \end{aligned} \tag{2.3}$$

Then integrating over a volume $V \in R^2$, and using the divergence theorem to convert the divergence term into a surface integral over the surface ∂V, and denoting with $\nu(\nu_1, ...\nu_d)$ the unit outer normal vector, equation **2.3** is transformed into:

$$\frac{\partial u}{\partial t} \int_V u dx + \oint_{\partial V} f(u) dx \tag{2.4}$$

The computational domain V is divided over non-overlapping cells of finite volumes: $V_r, r = 1, \dots N$, $V = \bigcup_r V_r$. Let u_r be the cell average of the unknown quantify into V_r. Then

$$u_r = \frac{1}{|V_r|} \int_{V_r} u(x) dx \tag{2.5}$$

Then the integral conservation law **2.4** becomes:

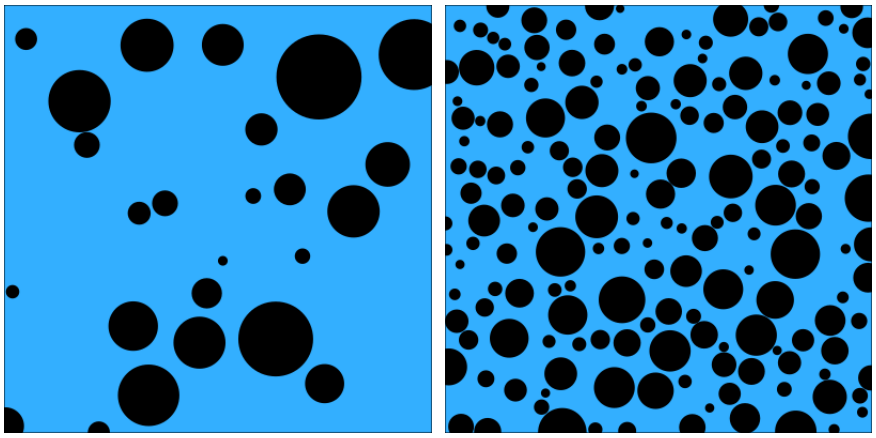
$$\begin{aligned} &|V_r| \frac{du_r}{dt} + \sum_{s \in N(r)} \int_{V_r \cap V_s} f_i n_i ds, where \\ &N(r) = \{Set\ of\ the\ cell\ with\ common\ interface\ with\ V_r\} \end{aligned} \tag{2.6}$$

The above equation is exact. To derive a discretization scheme the flux integral has to be approximated. This can be achieved by using Gaussian quadrature. For detailed analysis of that approach please refer to **[37, 38, 39, 40]**.

2.3 Porosity and Permeability

Two important properties, that characterizes a porous medium, are *porosity* and *permeability*. Those two properties present in many models in various forms and are closely related. Both properties are related to the number, size

and the connected openings in the rock or other porous medium. Porosity measures the medium ability to hold water or other types of fluid within its porous. It is defined as the ratio of open space in a medium divided by the total medium volume (solid and open space). Permeability is a measure of the ease of flow of a fluid to pass through a porous solid. For example, a rock may be extremely porous, but if the pores are not connected it will have no permeability. Likewise, a rock may have a few continuous cracks which allow ease of fluid flow, but when porosity is calculated, the rock doesn't seem very porous. Since porosity is a ratio between volumes it can take values from the range of $[0, 1]$. In contrast, permeability is dimensionless property. It can take values from $[0, \infty)$.



(a) 0.8 porosity (b) 0.6 porosity

Figure 2.3: Porous media consisting of spherical particles

When the flow within the media is laminar the permeability and porosity can be connected as Kozeny–Carman equations **[41]**. This equation is extensively used in practice. Equation contains a constant term that is normally problem-specific. Different constants are used for different models of porous mediums.

2.4 Multilevel Monte Carlo algorithm

In its simple form Monte Carlo (MC) algorithm is quite intuitive. Suppose the expected value of a given quantity needs to be computed. To achieve this, one generates samples $Q^{(i)}$, and computes the empirical expected value $E[Q]$:

$$E[Q_{M,N}] = \frac{1}{N}\sum_{i=1}^{N} Q_M^{(i)}$$

where M is characteristic parameter of the underling problem and N denotes the number of samples. Because the variance is inversely proportional to the number of samples - $N^{-1}V[Q]$ the Root Mean Square error (**RMS**) is $\mathcal{O}\left(\frac{1}{\sqrt{N}}\right)$. This directly means that achieving accuracy of ϵ requires $N \in \mathcal{O}(\epsilon^{-2})$ number of samples to be computed. Here rests the main weakness of the method - **its computational cost**. It may require an extreme number of costly samples to achieve the desired epsilon precision. One way to overcome the slow convergence is to use distinct samples, picked very carefully, to gain a better approximation, namely Quasi-Monte Carlo methods **[1]**, **[14]**. A more general strategy to overcome the slow convergence of MC method is to divide the problem into a combination of cheap fast estimators, and slow and expensive ones in a proper way for the given problem. Doing so will improve the convergence of MC. The main point of this idea is to represent the expected value of interest as a telescopic sum:

$$Q(\omega) = \underbrace{Q_{M_0}(\omega)}_{Y_0(\omega)} + \underbrace{Q_{M_1}(\omega) - Q_{M_0}(\omega)}_{Y_1(\omega)} + \cdots + \underbrace{Q_{M_L}(\omega) - Q_{M_{L-1}}(\omega)}_{Y_L(\omega)}.$$

Here each $Y(\omega)$ can be seen as a standard Monte Carlo estimator, for a given random vector ω in a properly defined random space. This idea is called

Multilevel Monte Carlo (MLMC) and it is particularly useful in the field of fluid dynamics. To describe it formally, assume $\omega_M : \Omega \longmapsto \mathbb{R}^M$ be a random vector over some probability space (Ω, F, P) and consider the quantity of interest Q_M, defined by some functional, depending on ω_M. Assume also that $E[Q_M]$ can be made arbitrary close to $E[Q]$ by choosing M sufficiently large. Our goal is to approximate $E[Q]$ by $E[Q_M]$. This can be achieved by computing an estimator Q_M and quantifying its accuracy using the root mean square error.

$$e(\widehat{Q}_M) = (E[(\widehat{Q}_M - E[Q])^2])^{1/2} \tag{2.7}$$

Then the standard Monte Carlo **(MC)** estimator is defined as:

$$\widehat{Q}^{MC}_{M,N} = \frac{1}{N}\sum_{i=1}^{N} Q_M^{(i)} \tag{2.8}$$

where $Q_M^{(i)}$, $i = 1, \dots N$ are independent samples of the unknown quantity Q_M. Assuming that the cost of computing one sample is $C(Q_M^i) \in \mathcal{O}(M^\gamma)$, where γ is positive constant. Expanding the mean square error will yield **[42]**:

$$\begin{aligned} e(\widehat{Q}^{MC}_{M,N})^2 &= E[(\widehat{Q}^{MC}_{M,N} - E[\widehat{Q}^{MC}_{M,N}] + E[\widehat{Q}^{MC}_{M,N}] - E[Q])^2] \\ &= E[(\widehat{Q}^{MC}_{M,N} - E[\widehat{Q}^{MC}_{M,N}])^2] + (E[\widehat{Q}^{MC}_{M,N}] - E[Q])^2 \\ &= V[\widehat{Q}^{MC}_{M,N}] + (E[\widehat{Q}^{MC}_{M,N}] - E[Q])^2 \end{aligned} \tag{2.9}$$

Since:

$$E[\widehat{Q}^{MC}_{M,N}] = E[Q_M], \text{ and } V[\widehat{Q}^{MC}_{M,N}] = N^{-1}V[Q_M]$$

the error becomes:

$$e(\widehat{Q}^{MC}_{M,N})^2 = N^{-1}V[Q_M] + (E[Q_M] - E[Q])^2 \tag{2.10}$$

In applications involving an elliptic PDE, M denotes spatial discretization parameter and Q_M approximates the inaccessible quantity Q. Q_M is computed by solving a PDE problem and the second term in equation **2.10** represents the error of the numerical method used for discretization. It has a bias effect on the estimator. Under the assumption M is sufficiently large, it can be considered that $(E[Q_M] - E[Q])^2 \leq \varepsilon^2/2$ holds. Then choosing $N^{-1}V[Q_M] \leq \varepsilon^2/2$, gives error estimation $e(\widehat{Q}^{MC}_{M,N}) \leq \varepsilon$. By neglecting the bias term in **2.10**, MC method converges with a rate of $1/\sqrt{N}$. The additional computational cost for a generated sample, governed by the assumption for large M, leads to large computational times for reasonable simulations. To overcome this limitation sampling from different levels is needed. The extension is quite natural. Let $\{M_l : l = 0 \dots L\} \in \mathbb{N}$ be increasing sequence of numbers called levels with corresponding quantities $\{Q_{M_l}\}_{l=0}^{L}$, and $s \geq 2$ be coarsening factor, such that $M_l = sM_{l-1}$, $l = 1 \dots L$. Defining $\widehat{Y}_l = Q_{M_l} - Q_{M_{l-1}}$ and setting $\widehat{Y}_0 = Q_{M_0}$, the following expansion for $E[Q_M]$ can be formulated:

$$E[Q_M] = E[Q_{M_0}] + \sum_{l=1}^{L} E[Q_{M_l} - Q_{M_{l-1}}] = \sum_{l=0}^{L} E[Y_l] \tag{2.11}$$

The expectation on the finest level is equal to the expectation on the coarsest level plus the sum of corrections on a difference in expectation on consecutive levels. The terms in equation **2.11** are approximated using standard MC independent estimators, each with N_l samples:

$$\widehat{Y}_l = N_l^{-1} \sum_{i=1}^{N_l} (Q_{M_l}^{(i)} - Q_{M_{l-1}}^{(i)}) \tag{2.12}$$

Then the Multilevel Monte Carlo estimator is defined as:

$$\widehat{Q}_{M,N}^{ML} = \sum_{l=1}^{L} \widehat{Y}_l \tag{2.13}$$

Analogously to MC method, for the mean square error (MSE) is:

$$e(\widehat{Q}_{M,N}^{ML})^2 = \sum_{l=0}^{L} N_l^{-1} V[Y_l] + (E[Q_M - E[Q])^2 \tag{2.14}$$

Of the formulation in equation **2.12, 2.13** above it can be seen that the definition of levels is quite arbitrary. It is subject to an open question and it is problem dependent. It is hard to answer the question of apriori estimation for the number of samples needed per level to achieve the desired accuracy. In practice a few samples are needed, to obtain an initial guess. To obtain stopping criteria and express the error in terms of samples, a minimization of the total computational time is done, under the given error tolerance ϵ. By defining C_0V_0 to be the cost and variance of one sample of Y_0 and C_lV_l be the cost and variance of one sample for estimator Y_l, then cost and variance of the method becomes:

$$C^{total} = \sum_{l=0}^{L} C_l N_l$$

$$V^{total} = \sum_{l=0}^{L} N_l^{-1} * V_l$$

Then for a fixed variance, the cost is minimized by choosing N_l to minimize:

$$C^{total} + \lambda^2 V^{total}$$

where λ is some Lagrangian multiplier. This gives : $N_l = \lambda\sqrt{V_l/C_l}$. Achieving total variance of ϵ^2 implies that $\lambda = \epsilon^{-2}\sum_{l=0}^{L}\sqrt{V_l/C_l}$. For more detailed analysis please refer to **[1]** and the references therein. Minimizing the total computational time, relation, between the number of samples that needs to be performed on a given level and the variance on that level. Turning N_l to integer gives the

$$N_l = \lceil \lambda\sqrt{(v_l/t_l)} \rceil \; where \; \lambda = \frac{1}{\epsilon^2}\sum_{l=0}^{L}\sqrt{(v_l/t_l)} \tag{2.15}$$

Then the total computational cost takes the form:

$$C^{total} = \frac{1}{\epsilon^2}\left(\sum_{l=0}^{L}\sqrt{V_l/C_l}\right)^2$$

A direct consequence is whether the product $V_l C_l$ increases or decreases with the estimator on level l, i.e. the cost grows with the level faster than the variance decreases. This determines the efficiency of the algorithm. For example, if the product increases with the level, then the dominant contribution to the cost comes from the finest estimator (the term $V_L C_L$). C becomes $C^{total} \approx \epsilon^{-2} V_L C_L$. If the dominant contribution comes from coarsest estimator then $C^{total} \approx \epsilon^{-2} V_0 C_0$. This contrasts with the standard MC cost of approximately $\epsilon^{-2} V_0 C_L$, assuming that the cost of computing Q_{M_L} is similar to the cost of computing $Q_{M_L} - Q_{M_{L-1}}$, and that $V[Q_{M_L}] \approx V[Q_0]$. This shows that in the first case the MLMC cost is reduced by a factor V_L/V_0, corresponding to the ratio of the variances $V[Q_{M_L} - Q_{M_{L-1}}]$ and $V[Q_{M_L}]$,

whereas in the second case it is reduced by a factor C_0/C_L - the ratio of the costs of computing Q_{M_0} and $Q_{M_L} - Q_{M_{L-1}}$ [1].

2.5 Conclusions

The circulant embedding algorithm provides a computationally efficient way of generating correlated random fields given a covariance matrix over a regular equidistant grid. The algorithm suggests that only part of the covariance matrix needs to be stored in the memory. This reduces the memory restrictions on the program. The generation of random variables points on the grid centers coincides well with the finite volume method. The finite volume method itself is treating the solution of the PDE within a control volume (in this case a square cell in 2-d) as constant. Combined with the local preservation of conservative laws and its low computational cost compared to methods such as finite volume makes it ideal for uncertainty quantification method. Finally, the independent sampling property of MC and MLMC makes them perfect, but not without computational challenge, for simulations on high-performance computational clusters.

By the opinion of the author, the main contributions of this chapter are:

- A review and analysis of the existing solutions to the considered problems are made. The advantages and disadvantages of the existing solutions for generating stochastic fields and corresponding sampling algorithms are evaluated;

Chapter 3

Multilevel Monte Carlo method for Laplace Equation

Many problems that incorporate uncertainty usually require solving a Stochastic Partial Differential Equation (SPDE). Those equations have attracted great attention due to their importance in modeling a variety of environmental and industrial processes. Thanks to the increased computational power, such solutions can be facilitated. For example, a simulation of a subsurface water flow in an area of hundreds of square km. In this chapter scalar elliptic SPDE can be used as a model for such a problem. However the approach considered here is not limited to this problem. The problem itself is a well-established model in the area and demonstrates the computational challenges in Uncertainty Quantification (UQ) for porous medium. To name few other applications, consider saturated flow in the subsurface or heat conduction in metal-matrix composites or other composite materials. This chapter starts by introducing the SPDE model in **3.1**. In sections **3.2**, **3.3**, **3.4** the different numerical model components are constructed and the Multilevel Monte Carlo

(MLMC) levels are defined. Section **3.4** develops the different approximations of the underling permeability field for the MLMC levels. The chapter conclude with a section **3.5**, dedicated to the numerical simulations. In this section, a comparison between MLMC and the pure MC algorithm is done. Also, results are presented for the performed number of samples per level, for different permeability generating parameters. The section ends with tables comparing the different approximation techniques. The chapter finishes with conclusions section, summarizing the obtained results.

3.1 Model equation

Consider steady state single phase flow in random porous media in a unit cube with domain $D = (0,1)^2$ and pressure drop from left boundary to the right boundary:

$$-\nabla \cdot [k(x,\omega)\nabla p(x,\omega)] = 0 \text{ for } x(x_1,x_2) \in D = (0,1)^2, \omega \in \Omega. \tag{3.1}$$

Subject to boundary conditions:

$$\begin{aligned} p_{x_1=0} &= 1 \\ p_{x_1=1} &= 0 \\ \partial_n p &= 0 \text{ on other boundaries,} \end{aligned} \tag{3.2}$$

Both the coefficient $k(x,\omega)$ and the solution $p(x,\omega)$ are subject to uncertainty, characterized by the random vector ω from a properly defined random space (Ω, F, P). The coefficient $k(x,\omega)$ describes the permeability field within the domain and the solution $p(x,\omega)$ describes the steady pressure distribution under pressure drop. An object of interest for this model is the mean quantity of the total flux through the unit cube:

$$Q(x,\omega) := \int_{x_1=1} k(x,\omega)\partial_n p(x,\omega)dx. \tag{3.3}$$

In this general form solving equation **3.1** is extremely challenging. Common way to overcome this limitation is to use simple designs for $k(x,\omega)$, that expresses the data as well as possible. One model that has been studied extensively is a log-normal distribution for $k(x,\omega)$. The correlation length scale for k, although small is significant, to fall outside the domain of stochastic homogenization techniques (see **[43]**). A commonly used covariance function that has been used in applications is the following:

$$C(x,y) = \sigma^2 exp(-||x-y||_p/\lambda) \ , \ p = \{1,2\}. \tag{3.4}$$

Where $||\cdot||_p$ denotes the l_p norm in R^2, and satisfies:

$$\begin{aligned} &E[K(x,.)] = 0, \\ &E[K(x,.),K(y,.)] = C(x,y) = C(y,x) \\ &\text{for } x,y \in D \text{ and } K(x,\omega) = log(k(x,\omega)) \end{aligned}$$

Here σ is the variance of the distribution and λ is correlation length - a constant parameter that characterizes typical repeatability range (for example, how likely a stone with given dimensions will appear on some observed underground field).

Solving numerically equations **3.1, 3.2** and acquiring quantity of interest **3.3**, requires first a correlated random permeability field (fixed vector drawn from the random space) to be generated. Then for such fixed field, the formed PDE can be solved. Conclusively quantifying the uncertainty is done by drawing samples and taking the empirical variance. Discussion is made for each separately.

3.2 Random field generation

Generating permeability field requires generation of random correlated variables according to a given covariance matrix. Different approaches have been developed which are applicable for large scale simulations. In [42], a Karhunen–Loève expansion has been considered, in [44] a Circulant Embedding approach is used. For a generating algorithm, the approach presented in **2.4** is followed and circulant embedding algorithm is used. More in depth analysis can be found in [44] and [8].

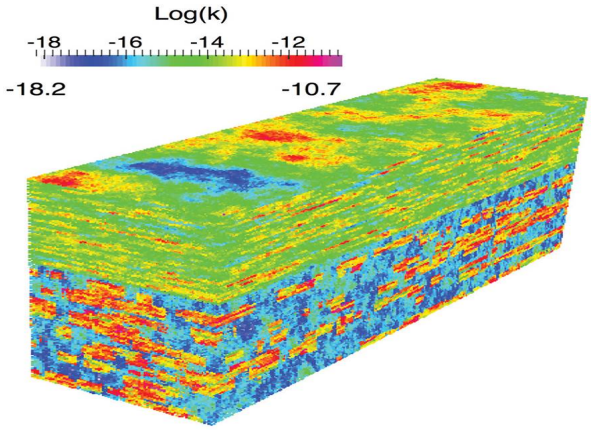


Figure 3.1: Single realization of permeability field in 3D

3.3 Problem discretization

After the random vector has been fixed, i.e permeability has been generated, the SPDE **3.1** with boundary conditions **3.2** for the corresponding realization of the permeability field is transformed to a PDE. To discretize the formed PDE, a Cell centered finite volume method is used. This method preserves the conservative laws in physics, such as mass preservation or energy preservation, and naturally leads to harmonic averaging of the discontinuous coefficients. This property represents the underlying physics more accurately. It is also the usual method of choice for flow simulation discretization. First the domain $D = [0, 1]^2$ is divided uniformly into $m \times m$ square cells denoted by $D_{ij} : (\frac{i-1}{m}, \frac{i}{m}) \times (\frac{j-1}{m}, \frac{j}{m})$, for $i, j = 1 \ldots, m$, and $x_{i,j}$ the center of the cell.

To obtain a discretization equation **3.1** is integrated over each cell to obtain a set of m^2 algebraic equations.

$$\int_{D_{i,j}} - \bigtriangledown \cdot (k(x,\omega) \bigtriangledown p(x,\omega)) = 0 \text{ for } i,j = \{1 \dots m\} \tag{3.5}$$

Then using divergence theorem for the left hand side of equation **3.5** the integral is transformed to a boundary integral:

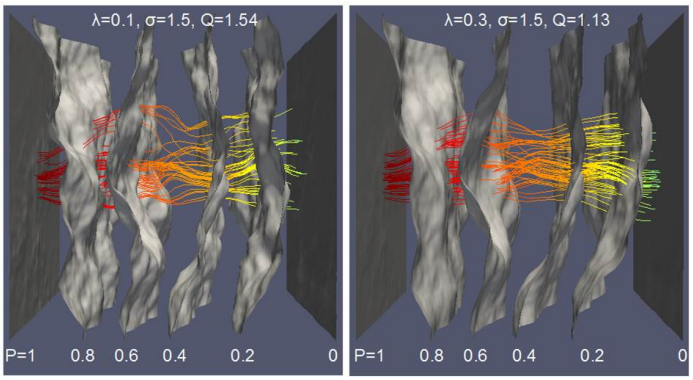
$$\oint_{D_{i,j}} -k(x,\omega) \bigtriangledown p(x,\omega) \cdot n \text{ for } i,j = \{1 \dots m\}$$

Let $k_{i,j}$ denote the value of k at $x_{i,j}$, and $p_{i,j}$ denote the approximation to p at $x_{i,j}$. To approximate the surface integrals, each cell edge is considered individually. For the contribution of the edge between $D_{i,j}$ and $D_{i+1,j}$ midpoint rule is used. To approximate k on the edge, harmonic average is used $\overline{k}_{i+1/2,j}$ of $k_{i,j}$ and $k_{i+1,j}$. To approximate $\bigtriangledown p(x,w) \cdot n$, central difference approximation is used: $(p_{i+1,j} - p_{i,j})/(x_{i+1,j} - x_{i,j})$. The contributions to other edges are approximated similarly, leading to standard five point stencil equation for (i,j):

$$\begin{aligned} &- \overline{k}_{i-1/2,j} p_{i-1,j} - \overline{k}_{i,j-1/2} p_{i,j-1} + \\ &- (\overline{k}_{i-1/2,j} + \overline{k}_{i,j-1/2} + \overline{k}_{i+1/2,j} + \overline{k}_{i,j+1/2}) p_{i,j} + \\ &- \overline{k}_{i+1/2,j} p_{i+1,j} - \overline{k}_{i,j+1/2} p_{i,j+1} \end{aligned}$$

A Neumann boundary condition, i.e. prescribed flux $-k\nabla p \cdot n$, is straightforward to approximate, by simply substituting the corresponding boundary term with 0. For the Dirichlet boundaries, midpoint rule and one-sided difference are used. For ∇p the diffusion coefficient in the boundary cell is taken. To solve the deterministic problem an account must be taken that

the permeability field has a huge magnitude. The governing matrix has bad condition number and it is quite problematic for numerical computations. To overcome this difficulty, Conjugate Gradient method preconditioned with Algebraic Multi Grid (AMG) is used. As linear backend, the solver provided by *DUNE* library is used in the current implementation.



Single realization in 3D of Laplace equation, and computed Q

3.4 Coarse Grain

The loosely defined definition of what a level means in the interpretation of equation **2.11** gives great flexibility when designing an algorithm that uses MLMC. One of the key components of MLMC is the selection of the coarser levels and it is strongly coupled with the simulated problem. In **[42]** authors use the number of the terms in the Karhunen–Loève expansion to define coarser levels. In **[45]** the authors consider coarser grids approximations of the fine grid problem build with AMG. In **[46]** the levels are defined as grid resolutions and arithmetic averaging for the coefficients is used to represent the permeability field on coarser grids. The number of the basis functions in a Reduced Basis mixed Multiscale Finite Element Method algorithm (MsFEM) is used to build the coarser levels in **[47, 48]**. To define a level in our Multilevel MC setting, the natural choice is the problem resolution i.e the grid size for the PDE. For a given estimator in equation **2.12**, the level is defined as the number of cells along an axis direction used in the discretization of the PDE. Assuming that on the fine grid the discrete PDE system has 2^M number

of cells, then on coarse level 2^{M-1} number of cells is used. Using this definition for two dimensions there are exactly 4 times more discretization points on the fine grid compared to the coarse and for the three-dimensional case - 8 times more. The fact that the stochastic coefficients are discontinuous means some kind of approximation must be employed. To represent the corresponding permeability field for the term to the coarse grid, three different approaches are considered.

Random field approximations

Renormalization

Here the coarser levels in MLMC are constructed by renormalization. This technique has been widely used in the past (and is still intensively used by many groups) for upscaling hydraulic conductivity in heterogeneous media. For details please refer to papers **[49], [50], [51]** and the references therein. Note that the effective hydraulic conductivity obtained as a result of the renormalization can be used to calculate an effective flux. In a nutshell, the simplified renormalization procedure is based on recursive harmonic arithmetic and geometric averaging. Dividing the domain $D = [0, 1]^2$ into $M \times M$ square cells, corresponding to level L of our MLMC algorithm. The generation of one realization of the random permeability field means that a permeability value $K_l(x)$ is associated with each cell. The permeability of twice coarser grid, corresponding to level $L - 1$, is calculated recursively by composition of harmonic, arithmetic and geometric means. If two neighboring cells are in series with respect to the flow direction, then the equivalent permeability is estimated as a harmonic mean; if the two cells are in parallel concerning the flow direction, then equivalent permeability is calculated as an arithmetic mean of the two. As illustrated on fig. **3.2**, by starting with harmonic average a combination of cells 1 and 2 to a rectangle $h_{1,2}$ is done. The value for the rectangle is harmonic average of the two cells. Similarly, cells 3 and 4 can be combined to a rectangle $h_{3,4}$. Now both rectangles are

perpendicular to the flow meaning, that they must be combined by arithmetic averaging. This leads to $K^{ha}_{1234} = (h_{1,2} + h_{3,4})/2$. In a similar manner K^{ah}_{1234} can be computed by first starting combining perpendicular to the flow. To obtain an equivalent permeability coefficient on a twice coarser grid, the geometric mean of $K_{1234} = \sqrt{K^{ah}_{1234} K^{ha}_{1234}}$ is taken. In the three-dimensional case, there are 3 different options of renormalization, depending on how you start combining. In our experiments, the sequence *harmonic-arithmetic- harmonic* is used. The procedure can be repeated recursively for more than two levels.

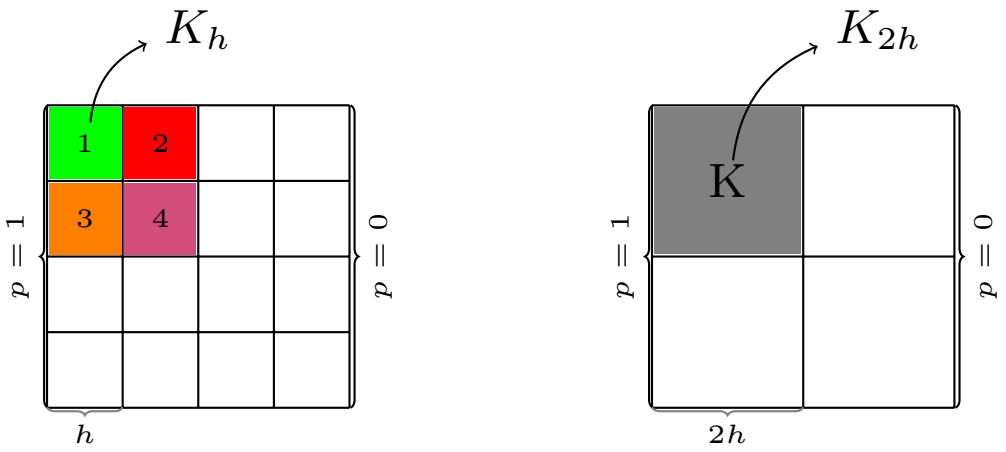
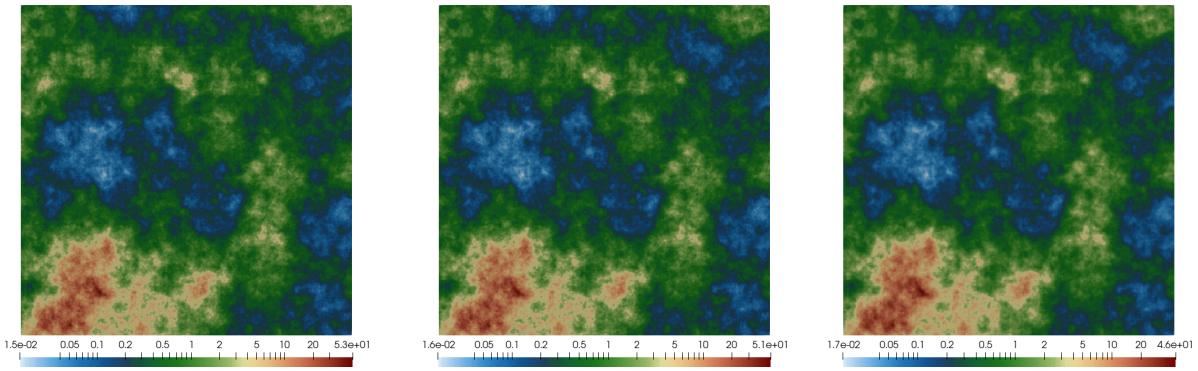


Figure 3.2: Simplified renormalization

On figure **3.3**, a simulation of a permeability field is shown. The permeability field is approximated with the described above renormalization. The generating parameters are: $\sigma = 2.0,\ \lambda = 0.3$; the covariance function is the two norm function (see equation **3.4**). The procedure acts as a smoother over the field and preserves the geometrical structure, which leads to preserved variance operator between two fields. Generation with one norm covariance function leads to similar results.

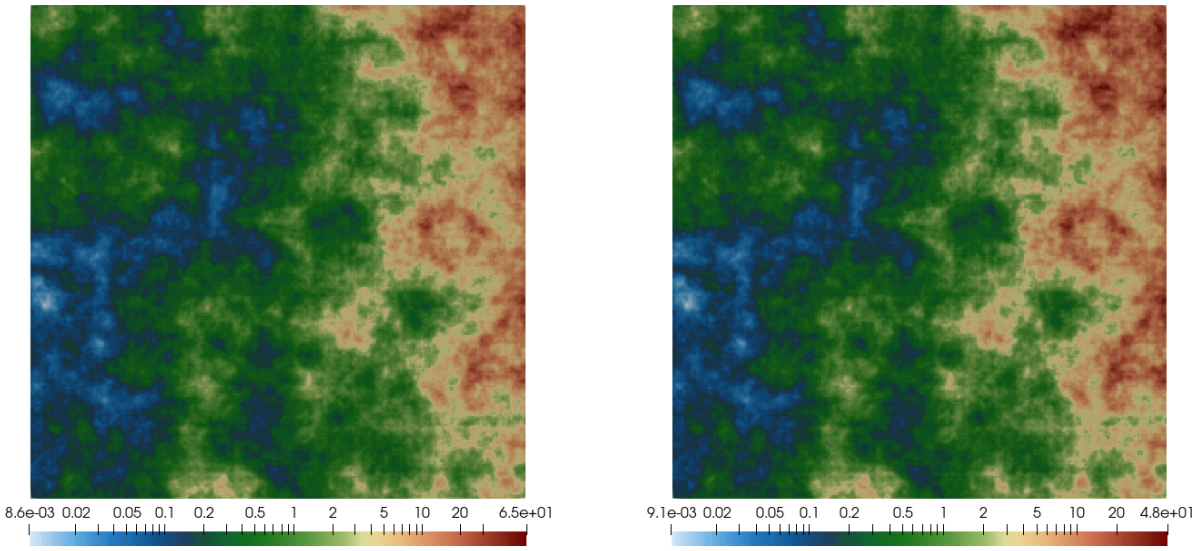


(a) Original field (b) Renormalized ones (c) Renormalized twice

Figure 3.3: Simulations of permeability, $\sigma = 2.0, \lambda = 0.3$

Simple Average

Consider again figure **3.3**. One simple idea to combine the permeability cells is to take the simple average of 4 by 4 for the case of two dimensions and 8 by 8 for three. This approach does not consider any underlying physics.



(a) with AVG (b) with Renormalizaion

Figure 3.4: Random Field Representation, $\sigma = 3, \lambda = 0.4$

Continuous interpolation

Another idea is to use interpolation for calculation of the field at a given point. The permeability field can be approximated by a low order point-wise

polynomial over a cell or a spawn of connected cells. For example a simple way of interpolation is to use Bi-linear approximation over a cell, where the coefficient of the approximation is determined by the corners of the cell. The permeability can even be sampled on a larger grid then the fine grid defined in MLMC setting, and interpolated by methods that use more points. The Bi-linear initial test were quite inaccurate, with relative error 10 times larger than the other discussed methods, and it is not considered and investigated any further, as it is out of the scope of this work.

Figure **3.4**, shows a single generation of the field obtained by the two renormalizaion methods. The field is approximated twice. It is clear, that structurally they seem to represent the field similarly, this however changes when the uncertainty is quantified.

3.5 Numerical Experiments

For testing how MLMC performs, the Laplace equation **3.1** with boundary conditions **3.2** is considered. The quantity of interest is given by **3.3**.

Table **3.1** contains the measured results from experiments with a three-level MLMC method with simplified renormalization. Theory behind this approach is that the true permeability value lies between the two terms K^{ah}_{1234} *and* K^{ha}_{1234} **[51], [50]**. That is why it is expected the variance operator to be preserved well across the different levels. As an averaging technique, simplified renormalization has a smoothing effect. Compared to arithmetic averaging the variance of the renormalized field is not far from the variance of the original field. This can be quantitatively confirmed by the presented data. The variances presented in the third column confirms both:

- after renormalization the variance at the coarsest level is close to the variance on the original fine grid.

- the variances for the corrections in MLMC are decaying very fast furthermore, the second column shows that the difference of the mean flux computed with MC and MLMC is close, and in the range of ϵ

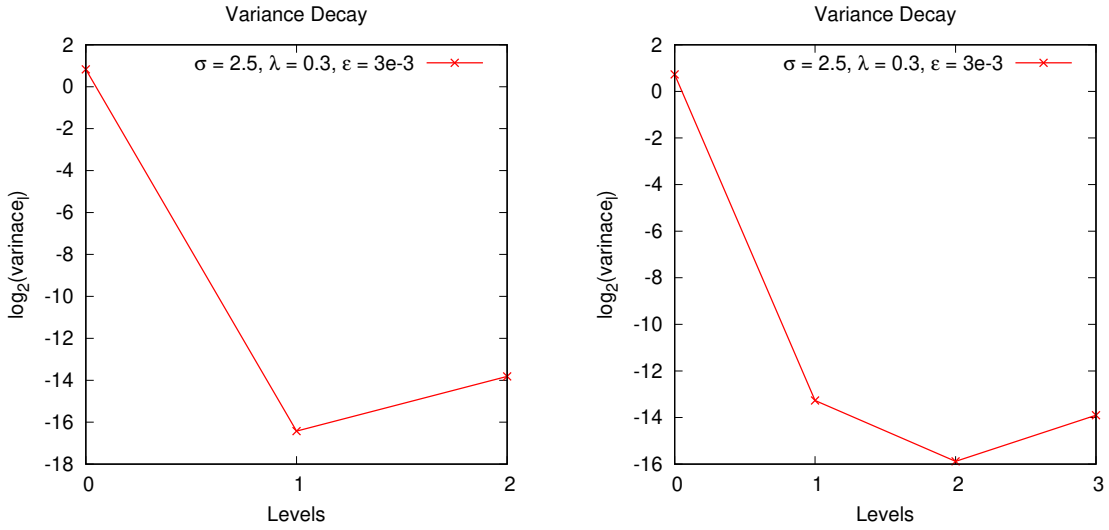
The fourth column shows that while MC needs tens of thousands of sample realizations on the finest grid. For the MLMC algorithm almost the same number of realizations are needed on a 16 times coarser grid, while only few computationally expensive realizations are needed on the finest grid. The renormalization technique leads to very effective MLMC and significant speedup can be achieved in comparison to the standard MC algorithm.

	$\lvert E[Q_{MLMC}] - E[Q_{MC}] \rvert$	$V[Y_l]$		Grid size and	N_l
MC	-	$V[Y_0]$:	1.10483	$2^{10} \times 2^{10}$	122761
Two level MLMC	0.00115	$V[Y_1]$:	1.17	$2^9 \times 2^9$	132229
		$V[Y_2]$:	$3.36e-05$	$2^{10} \times 2^{10}$	324
Three level MLMC	0.00590	$V[Y_0]$:	1.26	$2^8 \times 2^8$	128315
		$V[Y_1]$:	$8.89e-06$	$2^9 \times 2^9$	205
		$V[Y_2]$:	$9.82e-06$	$2^{10} \times 2^{10}$	107

Table 3.1: Simulation with permeability generating parameters $\sigma = 2$, $\lambda = 0.3$ and with Monte Carlo method tolerance $\epsilon = 3e-3$

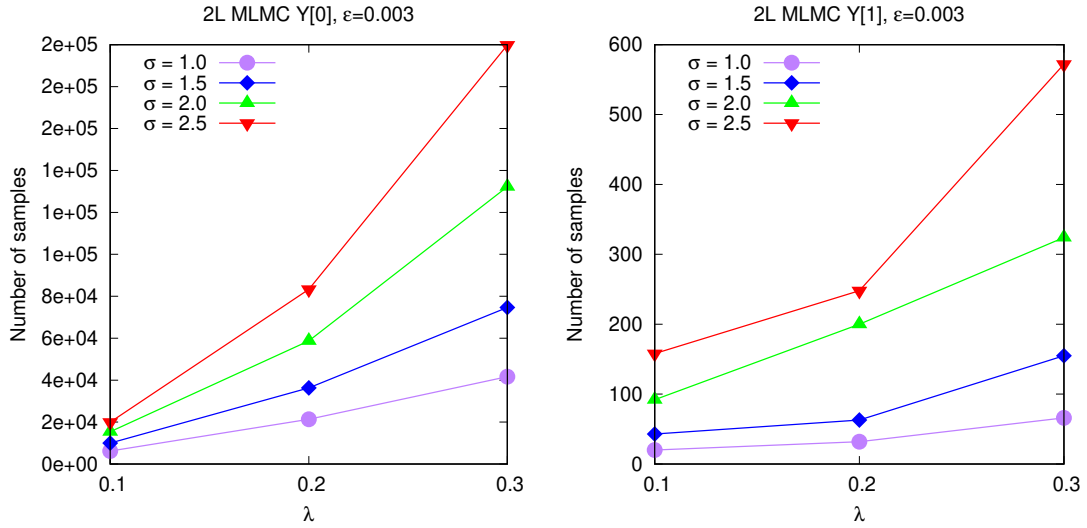
On figure **3.5** the decay of the empirical variance over the different levels of MLMC is plotted for a harder problem: $\sigma = 2.5, \lambda = 0.3$, compared to the problem considered in table **3.1**. Again, the renormalizaion technique accurately represents the permeability field on the coarser grids and very fast decay of variance is achieved, leading to a small number of expensive realizations on the finest level.

On figure **3.6**, the change of the number of samples, needed to solve different problems using MLMC, is shown. The simulation is done with renormalization. The figure shows the samples per problem across different stochastic permeability parameters. In all of the cases the dominant contribution to the cost of the algorithm comes from the coarsest level and $C^{total} \approx \epsilon^{-2} V_0 C_0$. In contrast, to achieve the same tolerance with the MC method, thousands of samples are required on the finest level.



(a) 3 Level MLMC (b) 4 Level MLMC

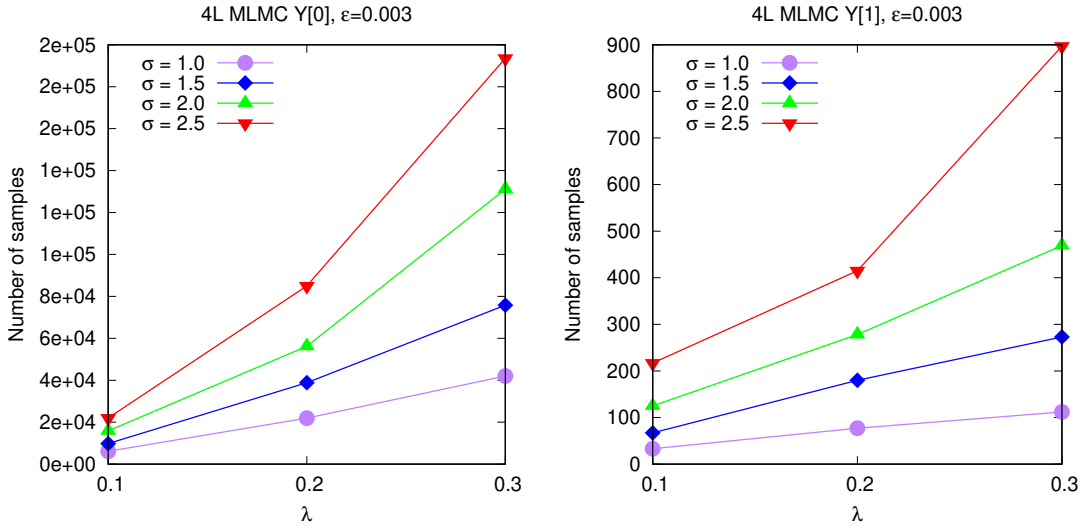
Figure 3.5: Decay of empirical variance for $\sigma = 2.5, \lambda = 0.3, \epsilon = 3e - 3$



(a) Coarse level (b) Fine level

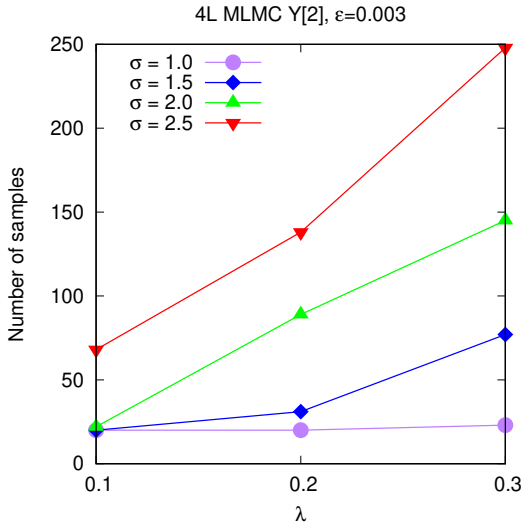
Figure 3.6: Estimated number of samples for 2 level MLMC estimator.

In figure **3.7** the same test as figure **3.6** is performed on a four level MLMC. Again the dominant contribution to the cost of the algorithm comes from the coarsest level and significant reduction of time is achieved, compared to MC.

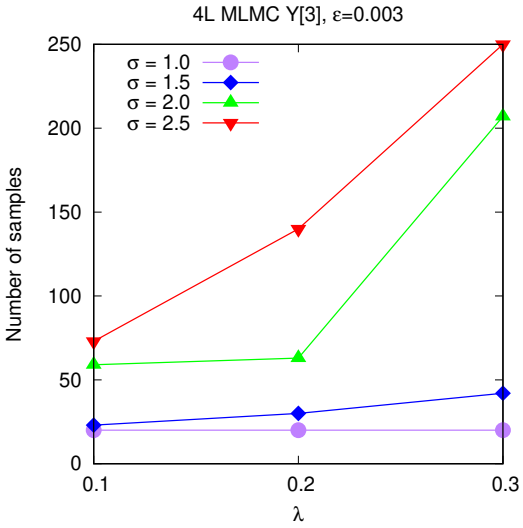


(a) Coarse level

(b) Middle levels



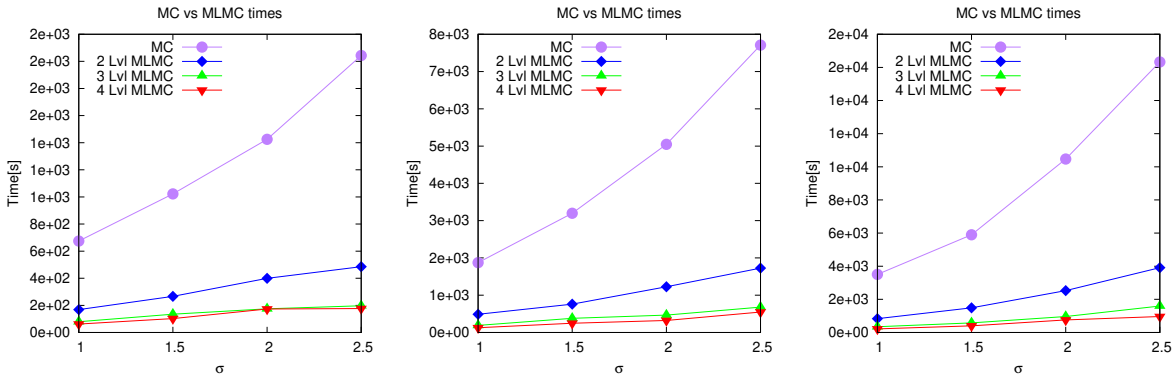
(c) Middle levels



(d) Fine level

Figure 3.7: Estimated number of samples for 4 level MLMC estimator.

To further illustrate the superiority of the MLMC algorithm, compared to a standard Monte Carlo sampling, the total computational time for achieving tolerance of $\epsilon = 0.003$ with MC and MLMC approaches for different stochastic generating parameters is considered. Figure **3.8** shows the results.



(a) $\lambda = 0.1$ (b) $\lambda = 0.2$ (c) $\lambda = 0.3$

Figure 3.8: Speedup of MLMC with compared MC on 196 cores

Experiments show that if the magnitude of the permeability is quite large this will yield the need for fine grids to achieve reasonable expected values for the total flux. The described MLMC method gives substantial speedup, compared to the MC method. The usage of the simplified renormalization provides a cheap way to build coarse levels in the MLMC. The variance at the coarser levels is very close to the variance at the fine level, which makes the presented particular MLMC method a very efficient variance reduction method.

Permeability approximation

In the next set of tests, the main interest is how the choice of approximation of the random field changes the MLMC sampling. The setting is the same: equation **3.1** with boundary conditions **3.2** and quantity of interest **3.3**, using finite volume is considered. A simulation of MLMC with four levels has been done with stochastic parameters $\sigma = 2.75, \lambda = 0.25$. The results are shown in table **3.2**. The prescribed tolerance is $\epsilon = 1e - 3$. The test have been done on 960 cores. A similar setting can be observed on table **3.3**, however slightly harder, with stochastic parameters: $\sigma = 2.75, \lambda = 0.3$. This test have been performed on 3840 cores. The speedup achieved in comparison to MC is between 7 and 10 times. The experiments are done using averaged

results over 10 consecutive runs. It is also worth pointing out that for the simulation of the Monte Carlo and using finite volume as a numeric method does not yield any kind of approximation of the permeability.

#Method	E[Q]	Time[s]	l_0	l_1	l_2	l_3
AVG	1.3411	3696	1.5e6	9035	3760	2211
RENORM	1.3412	3262	1.4e6	2674	1025	144

Table 3.2: MLMC simulation, on 960 cores, $\sigma = 2.75, \lambda = 0.25$, $\epsilon = 1e-3$, $E_{MC}[Q] = 1.3403$

From the data in table **3.2** it is directly observable that using simplified renormalization is a better way to represent the field than just simple arithmetic average. It is faster and preserves the variance better than the simple average. As for the fine level, only 144 samples have to be done as opposed to the 2211. This is also true for all other levels. Both approaches give value close to the expected value computed with Monte Carlo as the relative error for the simplified renormalization is $\approx 7e-4$, and for the simple average, it is $\approx 7e-4$.

#Method	E[Q]	Time[s]	l_0	l_1	l_2	l_3	l_4
AVG	1.4317	2273	2.4e6	11082	10651	4744	2586
RENORM	1.4316	1156	2.3e6	9473	3608	1420	252

Table 3.3: MLMC simulation, on 3840 cores, $\sigma = 2.75, \lambda = 0.3$, $\epsilon = 1e-3$, $E_{MC}[Q] = 1.4309$

At table **3.3**, where a harder stochastic test is performed, for a 5 level Multilevel Monte Carlo algorithm, again similar results can be observed:

Considering the comparison of the computational times and the number of samples, one would expect the proportion between the different times to be the same or comparable to **3.2**, but that is not the case. It is clear that using simplified renormalization is now much faster than simple averaging. Comparing the different number of samples that renormalization technique and simple average needs, shows that simplified renormalizaion needs much

fewer samples from the different levels and the computational cost of just averaging is the same. This means, that this technique preserves the variance operator much better, since by the formula **2.15**, for a comparable computational cost between simple average and simplified renormalization, the speedup must come from the reduced variance on the finer levels - the variance decays faster using simplified renormalization.

3.6 Conclusions

The proposed MLMC algorithm achieves a significant speedup compared to simulations with classical MC for a large range of permeability fields. The renormalization permeability approximation is on average 16 times faster than the classical MC (see figure **3.8**). At the same time, the relative error compared to the MC approach is in the range of $1e-4$. This makes the renormalization technique very suitable for large scale simulations. The approach with simple arithmetic averaging leads to similar result as the renormalization technique, but in the price of more samples on the finer levels. This means that renormalization is more productive way of representing the permeability field on the coarser levels. Comparable in cost by floating-point operations both requiring exponential time, the simplified renormalizaion requires less computational resources in terms of number of samples needed to be performed, to achieve the same tolerance.

By the opinion of the author, the main contributions of this chapter are:

- Different approaches for approximation of the stochastic field for the Laplace problem are analyzed and compared;
- An effective method for renormalization of the stochastic field for the purposes of the Multilevel Monte Carlo has been developed;
- An approach for determining the levels for the Multilevel Monte Carlo for the two considered problems is defined;

- Analysis and comparison between the rate of convergence and the time for calculation of the Multilevel Monte Carlo method with simplified renormalization and the classical Monte Carlo are made.

Chapter 4

Multilevel Monte Carlo method for Convection-Reaction-Diffusion equation

In this chapter problem that describes a reactive transport inside a random porous medium is considered. The main driving force of the transport is the processes of convection and diffusion. The equation used to model such processes is the well-known Convection-Reaction-Diffusion model [52]. This equation is used as a model in a chemical reactions, filtration processes as well as many other applications from physics, biology and chemistry. In section **4.1** a steady-state two-dimensional convection-reaction-diffusion equation is considered. It describes reactive transport in random porous media consisting of sand, gravel, and other soils. For practical experiments, the equation is considered in its dimensionless form. The quantity of interest is also addressed in this section. Sections **4.2, 4.3, 4.3** are devoted to the construction of the computational model, how the equation is discretized and how the quantity of interest is computed. Section **4.4** describes how the variance reduction in the model is done using MLMC algorithm. The chapter

concludes with numerical experiments in section **4.5** and summary section **4.6**.

4.1 Model equation

Consider a domain $\bar{A} = (0, \bar{L})^2$ and in that domain steady dimensional convection-reaction-diffusion equation describing reactive transport in random porous media:

$$\begin{aligned} &-\nabla \cdot (\bar{D}\nabla\bar{C}(x,\omega)) + \nabla \cdot (\bar{v}(x,\omega)\bar{C}(x,\omega)) + \bar{\kappa}\bar{C}(x,\omega) = 0 \\ &\text{where } x = (x_1, x_2) \in \bar{A} = (0, \bar{L})^2, \omega \in \Omega. \end{aligned} \tag{4.1}$$

Subject to boundary conditions:

$$\begin{aligned} &C_{x_1=0} = 1 \\ &\partial_n C_{x_1=L} = 0 \\ &\text{Symmetric boundaries on the lateral boundaries } (\partial_n C = 0) \end{aligned} \tag{4.2}$$

In equation **4.1** $\bar{D}[\frac{m^2}{s}]$ is the diffusion coefficient, $\bar{v}[\frac{m}{s}]$ is the Darcy velocity, $\bar{\kappa}[s^{-1}]$ is the upscaled volumetric (homogeneous) reaction rate and $\bar{C}[\frac{mol}{m^3}]$ is the unknown concentration. The volumetric reaction rate $\bar{\kappa}$ in the case of heterogeneous reaction (surface reaction at pore scale) can be expressed as $\bar{\kappa}{=}\bar{\gamma} \times \bar{\kappa_s}$, where $\bar{\kappa_s}[\frac{m}{s}]$ is the reaction rate of the heterogeneous reaction (reaction rate per unit surface area) and $\bar{\gamma}[m^{-1}]$ is the specific surface area (surface per unit volume).

To transform the equation **4.1** in a dimensionless form, consider the following dimensionless variables:

$$x = \frac{\bar{x}}{\bar{L}}, \quad v = \frac{\bar{v}}{\bar{v}_{in}}, \quad t = \frac{\bar{\kappa}\bar{L}^2}{\bar{D}}, \quad C = \frac{\bar{C}}{\bar{C}_{in}}$$

Setting $\bar{v}_{in}$ to be the characteristic velocity, $\bar{C}_{in}$ be the characteristic concentration (this is the prescribed concentration at the inlet), $D_p = \bar{D}$ be the characteristic value of the diffusion and let $\bar{\kappa}$ be characteristic reaction rate, the dimensionless form of the convection-reaction-diffusion equation **4.1** is written as follows:

$$-\nabla\cdot(\nabla C)+Pe(\omega)\nabla\cdot(v(x,\omega)C)+Da(\omega)C=0,\ x\in(0,1)^2,\omega\in\Omega. \quad (4.3)$$

The dimensionless *Peclet* and *Damkohler* numbers in equation **4.3** are defined as follows:

$$Pe(\omega) = \frac{\bar{v}_{in}\bar{L}}{\bar{D}(\omega)}, \quad Da(\omega) = \frac{\bar{\kappa}\bar{L}^2}{\bar{D}(\omega)} \quad (4.4)$$

In this formulation of **4.4**, the only part which is not a constant is the diffusion $\bar{D}$. Please note that both *Peclet* and *Damkohler* numbers depend on the diffusion, which in turn is proportional to the porosity, so one can rewrite the **4.4** as:

$$Pe(\omega) = \frac{\tilde{Pe}}{\phi(\omega)}, \quad Da(\omega) = \frac{\tilde{Da}}{\phi(\omega)} \quad (4.5)$$

Where $\tilde{Pe}, \tilde{Da}$ are predefined, based on intrisic diffusion at pore scale constants, and ϕ is the porosity. There are several formulas, relating the dimensional permeability $\bar{K}[m^2]$ to the porosity $\phi[-]$ and the specific surface area $\bar{\gamma}[m^{-1}]$. An extensively studied relation is given by the Kozeny-Carman

formula **[53]**. Many different interpretations exist of that formula. In this work, the one, given by Panda and Lake **[54]** is adopted:

$$\bar{K}(\omega) = \frac{\phi(\omega)^3}{2\tau(1-\phi(\omega))^2\bar{\gamma}^2} \tag{4.6}$$

In equation **4.6**, τ is the tortuosity value, this is a measure of the curvature at pore scale. For simplicity, in the simulations, the porous media is considered to be built by spheres (circles in 2D) with an average radius $\bar{r}$. For this case, the specific surface area is given by:

$$\bar{\gamma} = \frac{2}{\bar{r}} \text{ in 2D and } \bar{\gamma} = \frac{3}{\bar{r}} \text{ in 3D.} \tag{4.7}$$

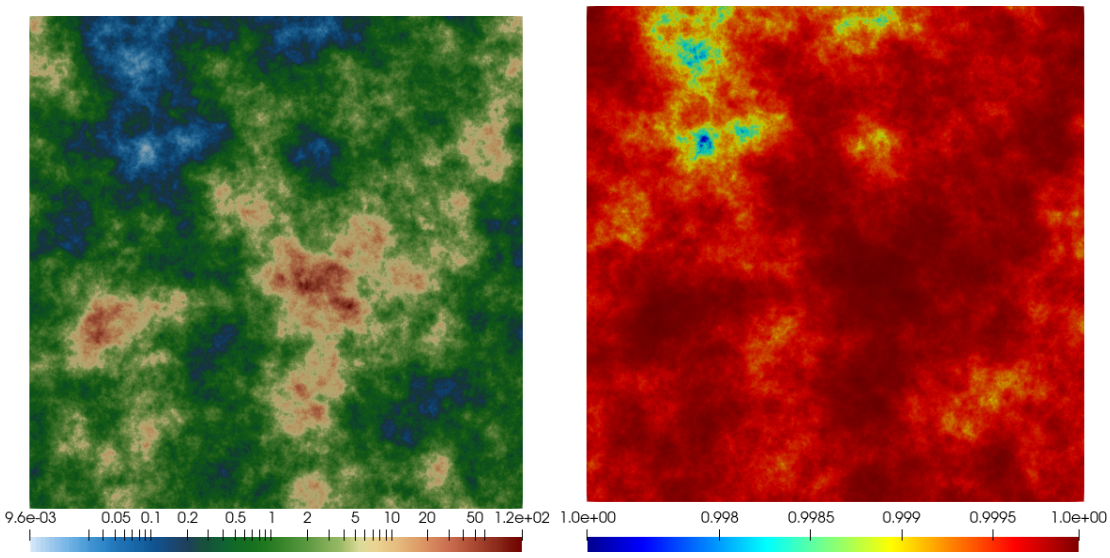
For the simulations, a medium consisting of soil, gravel, sand, silt, and clay is assumed. Without any loss of generality in the simulations the average radius of the particles is given by:

$$\bar{r} = 1 * 10^{-3}\ m.$$

The velocity filed in equation **4.3** is defined by *Darcy's law* **[55]**: $v(x,\omega) = -k(x,\omega)\nabla p(x,\omega)$, where $k(x,\omega)$ is the generated scalar permeability for a given point in the domain and $p(x,\omega)$ is the pressure at that given point. In the simulation, velocity is modeled with equation **3.1** coupled with the boundary conditions **3.2**. To model the underling permeability in the media, consider the same model as in the case of Laplace equation i.e. log-normal distribution, governed by covariance function from equation **3.4**. The two measured quantities are the average concentration and the total flux given by: $vC + D\frac{\partial C}{\partial n}$, across the outflow boundary for equation **4.3**.

4.2 Random field generation

The generation of the permeability field is done in the same way as described in section 3.2 - with circulant embedding algorithm. To model the porosity field it is required to solve the Kozeny-Carman equation **4.6** for porosity. The roots of the non-linear equation are computed by the bisection method.



(a) Permeability (b) Porosity

Figure 4.1: Single realization of permeability with $\sigma = 2, \lambda = 0.2$ and the corresponding porosity field with tortuosity $\tau = 1.3$

4.3 Problem discretization

To solve the PDE **4.3** for a fixed value of ω with boundary conditions **4.2**, a cell centered finite volume scheme on a uniform Cartesian grid applied. Integrating equation **4.3** over a control volume V gives:

$$-\int_V \nabla \cdot [\nabla C(x)] + \int_V Pe \nabla \cdot [v(x,\omega) C(x)] + \int_V Da\ C(x) = 0 \qquad (4.8)$$

Applying the divergence theorem to the first two terms, equation **4.8** takes the form:

$$-\oint_S [\nabla C(x)] \cdot \vec{n} + \oint_S Pe[v(x,\omega)C(x)] \cdot \vec{n} + \int_V Da\ C(x) = 0 \qquad (4.9)$$

The domain $D = [0,1]^2$ is then uniformly divided into $m \times m$ square cells with size $h \times h$ and $h = \frac{1}{m}$. Each cell is denoted by $D_{i,j}$ - the control volume V (see figure **4.2**). The center of the $D_{i,j}$ cell is labeled by P. The left cell along x coordinate of $D_{i,j}$ - $D_{i-1,j}$ is labeled as W (from west). Analogously the center of $D_{i+1,j}$ as E (from east), $D_{i,j+1}$ as N (from north) and $D_{i,j-1}$ as S (from south). The interfaces of the $D_{i,j}$ are labeled accordingly as : f_w, f_e, f_n, f_s and the normal vectors respectfully: $\vec{n}_{f_w} = (-1,0), \vec{n}_{f_e} = (1,0), \vec{n}_{f_n} = (0,1), \vec{n}_{f_s} = (0,-1)$. Using the mid-point rule the equations are transformed:

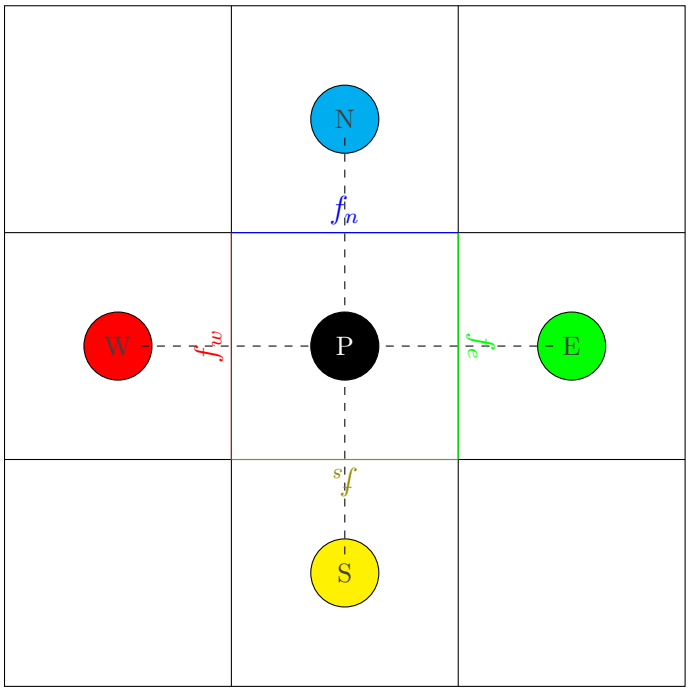


Figure 4.2: Control Volume V

$$-\oint_S [\nabla C(x)] \cdot \vec{n} \approx -\sum_{i=f_e,f_w,f_n,f_s} h(\nabla C_i \cdot \vec{n}) \tag{4.10}$$

$$\oint_S Pe[v(x,\omega)C(x)] \cdot \vec{n} \approx \sum_{i=f_e,f_w,f_n,f_s} hPe\ (v_i C_i \cdot \vec{n}) \tag{4.11}$$

$$\int_V DaC(x) \approx h^2 Da\ C_p \tag{4.12}$$

In the above equations, ∇C_i in **4.10** represents the value of $\nabla C(x)$, C_i in **4.11** the value of $C(x)$ at the middle point of the interface and C_p in **4.12** are the values of the reaction rate and the concentration at point P. To approximate the values at $\nabla C(x)$, C_i for the diffusion term **4.10**, a central difference is employed between the centers of the two cells shearing a common interface. Then equation **4.10** becomes:

$$-\oint_S [\nabla C(x)] \cdot \vec{n} \approx -\sum_{i=f_e,f_w,f_n,f_s} h(\nabla C_i) \cdot \vec{n} =$$

$$-h\left[\frac{C_e - C_p}{h} - \frac{C_p - C_w}{h} + \frac{C_n - C_p}{h} - \frac{C_p - C_s}{h}\right]$$

To obtain a numerical approximation for the advection term **4.11** upwind scheme is used. Equation **3.1** with boundary contribution **3.2** is solved on the same grid and the velocity: $v = -k\nabla p$ is approximated analogues to the **4.10** by central difference For the discontinuous coefficient k a harmonic average is used. To obtain a value for the stochastic $Pe(\omega) = \tilde{Pe} * 1/\phi(\omega)$, first a porosity values for the two neighboring cells are computed using Kozeny-Carman equation **4.6**, then to approximate it on the face the average value

of the two porosity values is taken. For example for the west interface Pe_{wp} becomes $Pe_{wp}(\omega) = \frac{2\tilde{P}e}{\phi_w(\omega)+\phi_p(\omega)}$. The convective term becomes:

$$\begin{aligned}
&\oint_S [v(x,\omega)C(x)] \cdot \vec{n} \approx \sum_{i=f_e,f_w,f_n,f_s} hPe_i v_i C_i \cdot \vec{n} = \\
&h[-P_{wp}(C_w max(0, v_i\vec{n}) - C_p max(0, -v_i\vec{n}))+ \\
&P_{pe}(C_p max(0, v_i\vec{n}) - C_e max(0, -v_i\vec{n}))- \\
&P_{sp}(C_s max(0, v_i\vec{n}) - C_p max(0, -v_i\vec{n}))+ \\
&P_{pn}(C_p max(0, v_i\vec{n}) - C_n max(0, -v_i\vec{n}))]
\end{aligned} \tag{4.13}$$

The *Damkohler* number in equation **4.12** is approximated in the same manner as *Peclet* number. To obtain a formula for the flux at the **inflow boundary** for the **diffusion term** (Dirichlet at $x = 0$), a forward Taylor expansion of first order is used.

$$C(x_{p_0}) = C(x_{w_0}) + \frac{\partial C}{\partial x}(x_{w_0})\frac{\Delta x}{2} \tag{4.14}$$

Here x_{w_0} is the point in the middle of the face coinciding with the Dirichlet boundary. Rearranging the previous equation gives:

$$\frac{\partial C}{\partial x}(x_{w_0}) = \frac{2}{\Delta x}(C(x_{p_0}) - C(x_{w_0})) \tag{4.15}$$

And for the **convection term** one-sided difference and Darcy's law is used:

$$\Delta x(Pe(w_0)\, v_{w_0} C(x_{w_0})) = \Delta x Pe(x_{p_0})(-K(x_{p_0}) * (p(x_{p_0}) - 1))C(x_{w_0}) \quad (4.16)$$

Putting the last two equations into equation **4.10**, the discretization scheme for the full flux on the Dirichlet boundary f_w takes the form:

$$-\Delta x(\frac{2}{\Delta x}(C(x_{p_0}) - C(x_{w_0}))) + \Delta x(-K(x_{p_0}) * (p(x_{p_0}) - 1))C(x_{w_0}) \quad (4.17)$$

The **outflow boundary**, where for equation **4.3**, has a Neumann (x = 1) boundary condition, the discretization for the **diffusion term** is 0. For the **convection term**, analogues to the inflow boundary, single sided difference approximation is used:

$$\Delta x(Pe(e_n) v_{e_n} C(x_{e_n})) = \Delta x Pe(x_{p_n})(-K(x_{p_n}) * (0 - p(x_{p_n})))C(x_{e_0}) \quad (4.18)$$

where $C(x_{e_0}) = C(x_{p_0})$ because the condition is Neumann and the solution is constant on the boundary.

Then the total flux on the **outflow boundary** takes the form:

$$\Delta x(-K(x_{p_n}) * (0 - p(x_{p_n})))C(x_{e_0}) \quad (4.19)$$

The boundary condition on the **sides** ($y = 0, y = 1$) is 0 - the diffusive term is zero from equation **4.3** and the velocity is zero, and thus the convective term.

Quantity of interest

To obtain a value for the average concentration simple average across the boundary cells is used:

$$Q(x,\omega) = \frac{\sum_{i=0}^{m} C(x_n)}{m} \tag{4.20}$$

To obtain a value for the total flux given by $vC + D\frac{\partial C}{\partial n}$ across the outflow boundary for equation **4.3** and take into account that there is not diffusive flux, reduce the formula to:

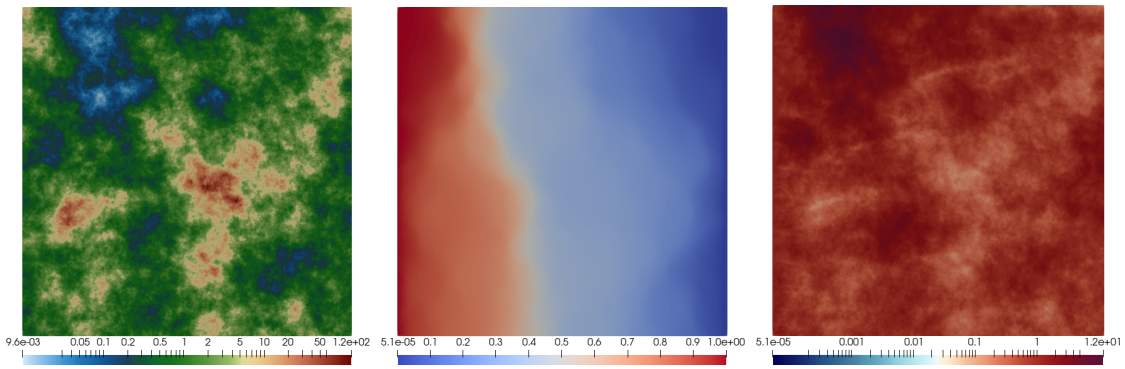
$$Q(x,\omega) = \frac{\sum_{i=0}^{m} v(x_n)C(x_n)}{m} \tag{4.21}$$

4.4 Coarse grain

In the model given by equation **4.3** the three sources of uncertainty that needs to be addressed by MLMC are the uncertainty that comes from the *Peclet* and *Damkohler* numbers and the uncertainty that is incorporated into the velocity field. Similarly to the case of the Laplace equation, MLMC levels are defined as grid resolution. Two ways of coarse graining the velocity field are considered.

- **First Solve then Renormalize**: The velocity equation **3.1** is solved on the fine grid and then it is approximated on the coarse level by simple arithmetic averaging.

- **First Renormalize then solve**: The permeability field is generated on the fine level, and then approximated using simplified renormalization to obtain a representation for the coarse level. Then the pressure equation can be solved and the velocity can be computed.



(a) Permeability (b) Pressure (c) Concentration

Figure 4.3: Single realization for convection-reaction-diffusion equation for given stochastic parameters.

4.5 Numerical experiments

To obtain more reliable results, each experiment is repeated 10 times. On table **4.1** results for 3 level Multilevel Monte Carlo with concentration as quantity of interest are presented. The coarse graning approach considered here is: **First Solve then approximate**. The stochastic parameters are $\sigma = 1.5, \lambda = 0.2$. The given desired tolerance is set to $5e-3$ and the finest grid is $2^{10} \times 2^{10}$. It is clear that raising the *Peklet* number, increases the stochasticity of the problem while increasing the *Damkohler* number leads to easier problems. Similarly on table **4.2**, are shown results from experiments with the flow as quantity of interest. However, for this experiment, the desired tolerance of the algorithm is reduced to $5e-2$. Computing the flow leads to much harder problem than computing the concentration, due to the stochastic velocity field. It seems that *Peklet* number has less effect here, compared to the results for concentration. In both tests, the relative error is low.

In figure **4.4**, an investigation of the effectiveness of the MLMC with *Solve then Renormalize* coarse-grain approach compared to the classical MC algorithm is presented. The quantify of interest computed is the concentration. For each problem on each level, the pressure linear system has to be solved

Pe	Da	E[Q]	$\frac{\lvert E[Q]-E_{mc}[Q]\rvert}{E_{mc}[Q]}$	Time[s]	Y_0	Y_1	Y_2
1	0.5	0.9568	2.472e-3	2539	16467	472	158
1	.15s	0.3955	1.283e-2	914	5199	124	46
1.5	0.5	1.1608	1.029e-3	3473	23121	673	226

$\sigma = 1.5, \lambda = 0.2, \epsilon = 5e-3$

Table 4.1: Concentration simulation on 112 cores, with single core per problem

Pe	Da	E[Q]	$\frac{\lvert E[Q]-E_{mc}[Q]\rvert}{E_{mc}[Q]}$	Time[s]	Y_0	Y_1	Y_2	Y_3	Y_4
1.5	0.5	5.5189	3.906e-3	1398	18213	1164	687	343	102
2	0.5	5.8463	1.365e-3	1452	19541	1297	735	348	101

a) $\sigma = 2, \lambda = 0.2, \epsilon = 5e-2$

Pe	Da	E[Q]	$\frac{\lvert E[Q]-E_{mc}[Q]\rvert}{E_{mc}[Q]}$	Time[s]	Y_0	Y_1	Y_2	Y_3	Y_4
1.5	0.5	6.1716	7.7910e-3	2269	34329	1692	896	571	206
2	0.5	6.4834	1.1876e-3	2450	36687	1932	1005	466	220

b) $\sigma = 2, \lambda = 0.3, \epsilon = 5e-2$

Table 4.2: Flow simulation on 224 cores, with 1 core per problem

on the finest level, which will lead to smaller gains between cheap estimations (coarser levels), and expensive ones (finer levels), thus impacting the effectiveness of the algorithm. Doing this type of coarsening, it is expected that the most effective MLMC will be with a low number of levels. This is exactly the case. Figure **4.4** shows that most gain is achieved at 3 levels MLMC. On the right side of the figure the number of samples is presented for the different MLMC estimators.

On figure **4.5** an examination for the other type of coarsening is considered. Test parameters are $\sigma = 2, \lambda = 0.2, \epsilon = 3e-2, pe = 2.5, da = 0.5$. This time the quantity measured is the flow. Using **Renormalize then Solve** the coarse grain approach gives significantly better overall speed. The two factors contributing are the overall smaller computational cost each processor

$\sigma : 2$			$\lambda : 0.2$		
$Peclet : 1.5$			$Damkohler : 0.5$		
N levels	Y_0	Y_1	Y_2	Y_3	Y_4
2	41502	607	-	-	-
3	43456	1536	536	-	-
4	47063	2809	1630	553	-
5	53093	5039	3084	1728	691

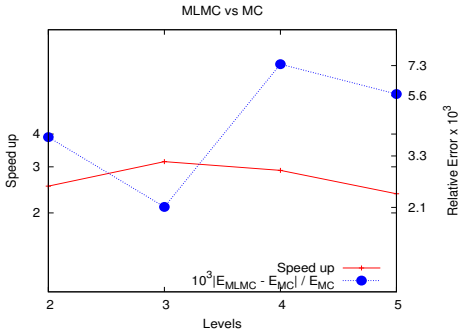
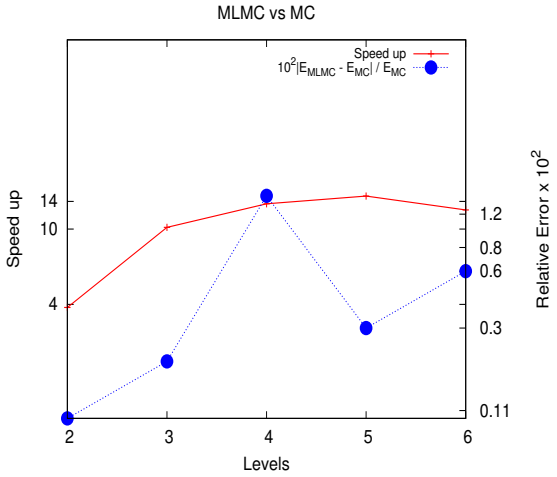


Figure 4.4: Achieved speedup, concentration

must do to compute a single sample and more importantly, the variance is preserved better across the different estimators.



#Samples MLMC 5L	
LO	49055
$L1$	2063
$L2$	916
$L3$	360
$L4$	78
Speedup times	14.95
$\lvert E_{MLMC} - E_{MC}\rvert / E_{MC}$	0.0039

Figure 4.5: Achieved speedup, flow

Table **4.3** is a direct comparison between the two coarsening approaches: **Solve then Renormalize (SR) and Renormalize then Solve (RS)**. The test parameters are the same as for figure **4.5**. This test positively shows the superiority of the **Renormalize then Solve** approach. The variance preservation is much better thus leading to much smaller number of samples generated.

#Coarseing	$L0$	$L1$	$L2$	$L3$	$L4$	Faster than MC	$\frac{\lvert E_{MLMC}-E_{MC}\rvert}{E_{MC}}$
SR	57370	3742	2223	1135	374	$\approx \times 2.5$	$\approx 3.4e-3$
RS	49055	2063	916	360	78	$\approx \times 14.9$	$\approx 3.2e-3$

Table 4.3: Solve then Renormalize (SR) and Renormalize then Solve (RS)

4.6 Conclusions

Compared to the Laplace equation, computing a sample for Convection-Reaction-Diffusion equation is much more expensive, hence not only the pressure field but also the steady-state CRD equation has to be computed. The experiments show that the constants in the *Damkohler* and *Peklet* numbers have a significant impact on the uncertainty of the system. *Peklet* number can be viewed as a contributor to the overall uncertainty, resulting in a need of a larger number of samples. As a system with three main sources of uncertainty, quantifying it is a much computationally harder problem compared to the Laplace equation. This makes MLMC even more appealing as it provides a way to speedup the simulations. Both of the proposed methods for coarse-grain - *solve then renormalize* and *renormalize then solve* achieve speedup. However, the MLMC speedup is not as significant as the Laplace equation in the case of *solve then renormalize*, because the reduction of the time that MLMC provides comes only from the CRD part of the problem and the pressure part has to be solved in both cases, thus limiting the overall gain. Nevertheless *renormalize then solve* gives better speedup, since the overall time needed to approximate the permeability field and compute the pressure field is smaller than the time to first compute it and then renormalize it.

By the opinion of the author, the main contributions of this chapter are:

- The Multilevel Monte Carlo method is applied successfully for the first time to solve the convection-reaction-diffusion equation;
- An approach for determining the levels for the Multilevel Monte Carlo for the two considered problems is defined;

- Analysis and comparison of the two considered approaches for coarse grain, for the Multilevel Monte Carlo versus the classical Monte Carlo for the convection-reaction-diffusion problem are performed.

Chapter 5

Parallel Algorithms

This chapter addresses the parallel challenges of MLMC simulations. As a parallel algorithm, MLMC concurrently computes different MC estimators and properly combines them to obtain the final results. When applied as a variance reduction method, it can be divided into different components each with its specific problems and algorithms. By following the mathematical construction, the distinct parts of a simulation are as follows:

- (i) Generation of the correlated random field that represents the uncertainty in the SPDEs.
- (ii) Solving deterministic PDE, for each realization of the random coefficients.
- (iii) Quantifying the uncertainty, using Multilevel Monte Carlo.

The first two sub-problems are essentially identical as if the simulation is done with the standard Monte Carlo algorithm. The two algorithms diverge in the third part sub-problem - where the UQ is done. To achieve faster convergence MLMC constructs different MC estimators with different sample cost, as

formula **2.13** suggests. This makes the problem of efficient parallelization much harder to answer.

In both cases, the simulations start by acquiring statistical data with initial or preparation step. In this step, several samples are generated to obtain empirical data. From the gathered data, the variance can be computed, and using formula **2.15** the number of samples that need to be computed to obtain the desired tolerance can be estimated. This calculation is repeated until the desired tolerance is obtained. Section **5.1** describes in detail the computational procedure. A discussion of the common problems of oversampling, and underestimation is made alongside some practical approaches to overcome them. Despite its severe superiority to MC (see **[42, 46, 56]**), using MLMC can still be computationally very expensive, particularly for realistic industrial models, where large scale simulations are needed. To overcome this huge computational cost, effective parallel strategies must be put in place. The parallelization can be done at different stages of the algorithm. Analogous to the different stages of the algorithm, main parallel layers can be defined:

- (i) Parallelizing the solution of the PDE for each sample.
- (ii) Parallelizing the solution of all samples at one MLMC level.
- (iii) Parallelizing at all or several MLMC levels simultaneously.

In the work of **[57]**, static parallelization, done at each level separately, is studied. The idling processors are filled with samples, which achieves good scalability but it is less computationally efficient than other methods. In **[5]** the parallelization is done mainly on the first and second layer of parallelism - within a given level and a sample. Section **5.2** extends that mentioned approaches. In the section analysis of different strategies is performed. A scheduling scheme, that performs parallelization on all of the levels simultaneously, is proposed. The section finishes with specific modifications of the

MLMC algorithm, that provide better overall efficiency, namely in section **5.3.5** and **5.3.6**. The chapter concludes with sections **5.5** and **5.6** where parallel performance experiments are presented and section **5.7**, with conclusions from the performed experiments.

5.1 MLMC computation scheme

Both MC and MLMC, as a sampling base class of algorithms, rely on repeated random sampling. A generation of samples from a properly defined probability space is done. After a sample is generated, the underlying equation becomes deterministic and standard methods can be employed to solve it. Upon solving it, the observed quantity of interest can be extracted and accumulated to the statistics. After a given number of samples are computed, the statistical moments are calculated and checked against defined stopping criteria, such as root mean square error or some other type of error measurement. If the condition is satisfied, the procedure ends and if the condition is not satisfied, additional samples must be generated. This process is repeated until the stopping criteria is met. Figure **5.1** lustrates the idea. Each *estimation* phase is followed by a *solve* phase, and those *estimate-solve* blocks are repeated, until the stopping criteria is met.

Figure 5.1: Block diagram of MLMC algorithm

When MC and MLMC algorithms are used for uncertainty quantification of SPDE, for each *estimate-solve* block, in the estimate phase, the required number of PDE samples that needs to be solved is estimated by equation **2.15**. Each *solve* phase consists of three main sub-tasks:

- **(i)** Generation of predefined number of correlated random field data that represents the uncertainty in the SPDEs.

- **(ii)** Solving a deterministic PDE for each realization of the random coefficients.
- **(iii)** Quantifying the uncertainty, using MLMC or MC.

In this phase lies the main computational challenge of the MLMC. By construction, MLMC consists of different MC estimators and each estimator itself has the described solve cycle with its own computational cost. Each *solve* phase of the cycle for the Multilevel Monte Carlo, can be further fragmented to a hierarchy of solves phases. While for a sequential program that is not a problem, designing a concurrent version requires balancing those different phases across the computational resources. Now let us focus on the *solve* phase of the problem.

(i) Generation of correlated random field that represents the uncertainty in the SPDEs.

Generating correlated random fields is an essential problem in many stochastic simulations. For both of the considered SPDEs models, the permeability field is represented as a correlated random field (for details see chapter **2**). The correlation matrix is constructed by one parameter stationary covariance function **3.4**. After the matrix is constructed, the typical approach is to find a linear transform that diagonalizes the covariance matrix, hence such a diagonal matrix consists of uncorrelated random variables. Finding such transformation, however, is a computationally expensive task. The matrix size grows extremely fast as new points are added to the grid. To overcome this, special generation algorithms must be considered. If the points are uniformly spaced, the associated covariance matrix C with a stationary Gaussian random field has a Toeplitz matrix. This special structure allows drawing of samples to be done from a reduced covariance matrix, a necessity for simulations of large scale. The circulant embedding approach is discussed in chapter **2.1**, exploiting Fourier transform of a Block Circulant with Circulant Blocks (BCCB) matrix, offers better computational

complexity, compared to the popular Cholesky decomposition approach and approaches including Karhunen–Loève Transform and the discrete version of Karhunen–Loève expansion. In the Circulant Embedding strategy, a permeability sample requires one forward and one inverse Fourier transform. In practice the transform is done as Fast Fourier transform (FFT), which have complexity of $\mathcal{O}(nlog(n))$, where n is the number of elements in the transform [58, 59, 60]. In this case, n is the number of the reduced covariance matrix rows plus possibly the number of the elements from the embedded extensions. Setting two large embedding extensions, in order to avoid negative eigenvalues, can lead to a performance hit [33, 34]. Our implementation follows the implementation described in [44], with minimal extension of n.

For implementation the open-source *C++ FFTW library* [61] is used. It supports MPI with its own domain decomposition scheme and it is one of the most efficient implementations available. The procedure for generating samples from a given properly defined probability space, using Circulant Embedding consist of preparation step and generation step. In the preparation step, the reduced covariance matrix is embedded into $BCCB$ matrix. Afterwords a forward Fourier transform is applied to the resulting matrix, finishing the preparation step. Then drawing samples is straightforward. Each element is multiplied by the transformed matrix by a random variable with the desired distribution, and by applying inverse Fourier transform the results are obtained. A simple step by step algorithm can be expressed as:

1. Reduce and embed the covariance matrix into $BCCB$ matrix and apply forward FFT.

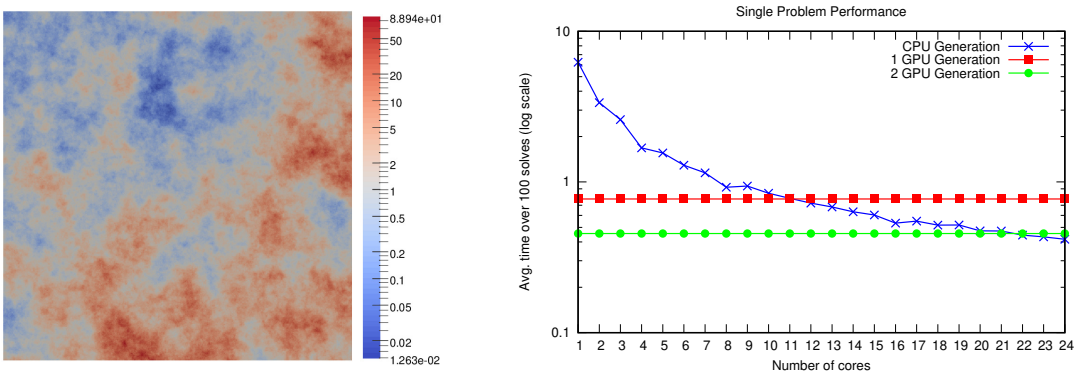
2. Generate samples:

   (a) For each element of the transformed matrix, generate uncorrelated random variable from the desired distribution, and multiply it with the element.

(b) Apply $IFFT$ to the modified matrix to obtain a permeability field.

Note that the transform is done in the complex plane, and each element of the resulting matrix has a real and imaginary part. This means that for a single $IFFT$ two correlated random fields are obtained. In the parallel version of the algorithm, the random number generation can be done without any exchange of information between the *MPI processes*. This means that the scalability of the algorithm will depend mostly on the efficiency of IFFT implementation. The standard data decomposition of the *FFTW library* is performed in one dimension. The data is stored in a standard row-major C order, this means that the matrix is stored row by row, consecutively in memory, and each *MPI process* receives an equal portion of data rows. This type of representation is done for efficiency reasons.

To generate a sample, each MPI process is doing the same type of matrix multiplication and random variable generation many times. Those types of identical work can benefit significantly from co-processor accelerators. To explore this idea an implementation of the algorithm on a NVIDIA GPU, using *cuFFT C++ library* - An NVIDIA implementation of FFT on CUDA cores **[62]** is done. Figure **5.2**, compares the two parallel approaches. The test is performed on a single *GPU Node* consisting of *2 × NVidia Tesla K80 GPUs and 2 × Intel Xeon E5-2695 v2 processors* on the education and testing polygon part of *HybriLIT* supercomputer, Dubna Russia. The averaged times for generation of 100 permeability fields on a grid of size $2^{10} \times 2^{10}$, with $\sigma = 2.0$ and $\lambda = 0.2$ is shown. The speedup that is achieved using only CPU cores for computation at all of the steps is significant up to around 8 cores. At this point, it is approximately up to 6.755 times faster than calculation on a single core. When utilizing more than 8 cores the efficiency gradually decreases. On the other hand, the GPU generation is much faster. Using single GPU generation time is comparable to *11 CPU cores*, and on two $GPUs$ - approximately *22 CPU cores*.



(a) One realization

(b) Gen. Time. CPU vs GPU

Figure 5.2: Generation time and scalability for: $\sigma = 2, \lambda = 0.2$, with grid size: $2^{10} \times 2^{10}$

(ii) Solving deterministic PDE for each realization of the random coefficients.

For a fixed random field (permeability field), the Stochastic partial differential equation becomes deterministic and a standard numerical scheme can be employed to solve it. Using the well-established and standard cell-centered finite volume method leads to sparse linear systems that must be solved. *Dune library* provides a large number of linear solvers. For the elliptic problem *Conjugate Gradient* linear solver is used as the matrix is symmetric and positive-definite. For the convection-reaction-diffusion equation, *Bi Conjugate gradient* (BiCG) is used with generalized minimum residual (GMRES) algorithm after each BiCG step. The partitioning of the data in both cases is done by the same algorithm: the data is partitioned in either in one dimension, similar to FFTW library decomposition, or in two dimension as a blocks. In the case when the number of processors is an exact power of two the data is partitioned as equal squares and cubes. This distribution diverges from the partition used in FFTW library - one-dimensional domain decomposition. This indicates an additional work has to be performed to distribute the data accordingly. The embedded matrix is at least twice as large than the covariance matrix and from the construction of the algorithm, the resulting

permeability will be located on the top left quarter of the embedded matrix, if minimal embedding is used. The permeability will be distributed, on half of the *MPI processes* used for the generation of that sample. The other half of the processes will have to fetch all of their associated data, from processes that contains the data. To check the impact of the overall performance for the averaged times over 100 samples for the Laplace equation, for a different number of processors are presented on figure **5.3**. Test is over a grid size $2^{12} \times 2^{12}$ and permeability generating parameters $\sigma = 2, \lambda = 0.3$.

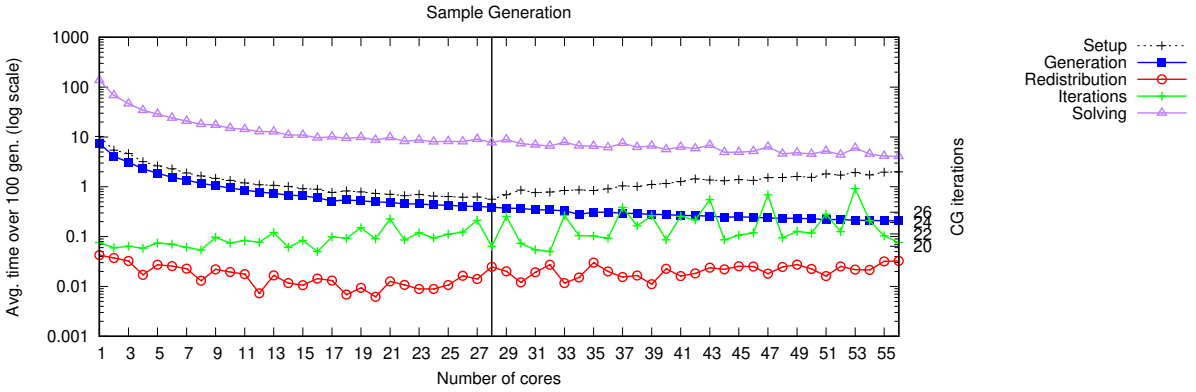


Figure 5.3: Parallel sample generation

The test shows that the time to construct and solve the linear equation is much more than the time for generation of permeability field. Both algorithms show similar scalability efficiency. After the 28 processor (the vertical line in the graph) an inter-node communication begins. This additional communication does not impact significantly the time needed for the permeability redistribution - the part of the generation where the largest quantity of data is exchanged. The time needed for redistribution is orders of magnitude smaller than the time needed for generation of the permeability. And the time to find a solution to the linear system is orders of magnitude smaller than the time for generation. The parallel efficiency of the sample generation will be determined by the effectiveness of the linear solver.

(iii) Quantifying the uncertainty, using Multilevel Monte Carlo.

The last part of the simulation is the uncertainty quantification. The main algorithmic procedure steps, that MLMC consists of, are as follows:

1. Initialize input parameters which includes: size of the finest grid; The number of levels of which MLMC estimator will consist; the permeability governing parameters σ, λ. And eventually additional parameters specific to the SPDE that is solved.

2. Predict the number of samples needed for each MC estimator in order MLMC to converge using formula **2.15**. Generate the required number of samples and compute the statistical moments of interest. Repeat 2 and 3 until converge.

3. For each MC estimator generate a permeability field and for each level of the estimator, solve the formed PDE and accumulate the quantity of interest.

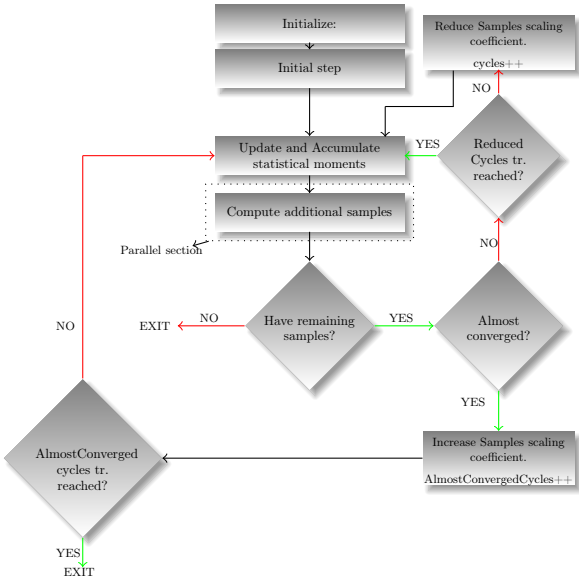
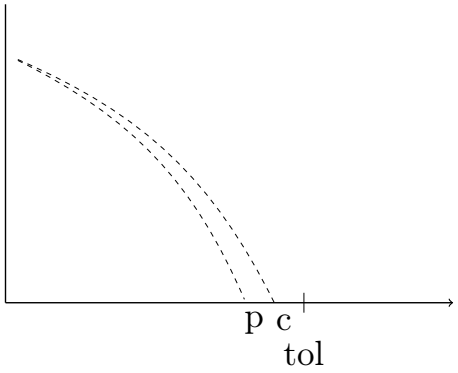


Figure 5.4: Block diagram of MLMC algorithm

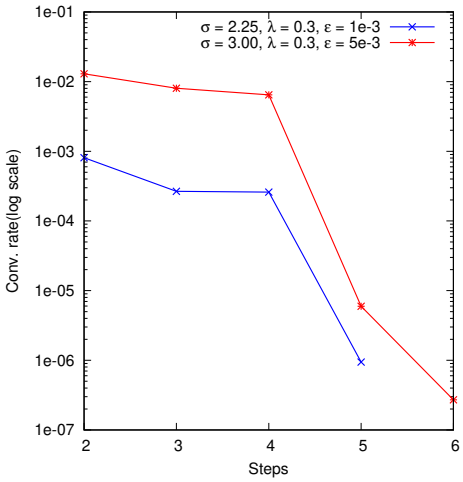
At the start of the algorithm, there is no statistical information available to the estimators, and formula **2.15** can not be used. This implies that MLMC

begins with a selected number of samples that have to be performed on each sub MC estimator, as shown in figure **5.4**. The decision of how many samples to be performed is problem-dependent. For example, setting too many samples on the finest level may lead to oversampling, as the number of samples required for the given tolerance may be much smaller. Analogously, setting to a small number of samples may lead to insufficient statistical data, and inaccurate prediction of the number of additional samples as well as the computation time per sample per estimator. To get a sense of how the samples are distributed and how many samples have to be done at each level, a preparation phase can be used. In this phase, a similar simpler problem will be solved (e.g same statistical parameters, much smaller gird), to obtain a rough estimate of the time distribution of the sample. After the initial step, the algorithm enters its main loop, statistical moments are computed and formula **2.15** is used to determine if additional samples are needed to achieve convergence. After the estimation, parallel computation is performed. At the end of the computation formula **2.15** is used to check again for convergence. If it converges, the algorithm stops; if not - the cycle is repeated. For each estimate-solve cycle the reduction of variance will be smaller and smaller, because each sample contributes to the reduction of the variance. This means, smaller and smaller steps are made towards the desired tolerance, as shown on figure **5.5**. When the algorithm gets very close to convergence, the reduced variance or the steps can get so small that formula **2.15** may estimate that just a few additional samples are needed. While mathematically, this is not a problem, those small steps can lead to significant performance degradation. A few samples will have a tiny impact on the accumulated statistics and combined with numerical rounding, may lead to very slow progress or even no progress towards the desired tolerance.

To improve upon that problem, a measure for how the algorithm converges is introduced. The convergence "rate" between two "estimate-solve" cycles is



(a) Step



(b) MLMC Typical convergence rate

Figure 5.5: MLMC convergence

defined by the following formula:

$$\delta = c - p \tag{5.1}$$

where c denotes the $RMSE$ for the current cycle, and p denotes the $RMSE$ for the previous one. Since formula **2.15**, connects the empirical variance with the error ϵ, when the current RMSE of the algorithm is close to the desired tolerance, the difference will be very small. This measure gives information for the progress of the algorithm and it is allowing more fine-tuned control. When the algorithm is close to converging, some prescribed threshold for convergence speed can be set. If the speed is too low and progress is slow, an artificial increase of the required samples can be made. In the implementation of the algorithm this threshold is set to $10^3\epsilon$. Upon reaching the threshold, an increase in the number of estimated samples by 200% is done. This process repeats maximum three times. The fourth time, when the speed is smaller than desired, the program terminates. This reduces the risk of doing many *estimate-solve* cycles and improves the performance of the algorithm. A very similar situation arises when the algorithm starts. The main issue here is the lack of sufficient statistical data. In the first few *cycles*

estimation using formula **2.15** may lead to sample times and variables that are significantly away from the means. This will lead to inaccurate sample estimations, where much more samples must be computed than necessary for the given tolerance. This oversampling effect can also notably slow the algorithm. These inaccurate predictions also affect how the resources are distributed among the processors in the concurrent implementation. Much alike when the algorithm is very close to converges, where an increase in sampling is needed, here instead decrease is needed. For this reason, in the implementation of the first three $estimate - solve$ cycles, the performed samples are set to 40% of the original estimation. Those two optimizations make current version of MLMC more stable and predictable algorithm.

5.2 Parallel Algorithms

MLMC algorithmic approach can provide significant convergence speedup and better computational cost compared to pure MC, but the simulations are still quite demanding. They may require millions of samples to complete. Running MLMC on a single processor is not feasible. Efficient scheduling strategies for parallel computation are a necessity. In the case of sufficiently large domains, where a single problem has to be solved on multiple processes due to memory restrictions, the strategies also depend strongly on the performance characteristics of a single sample solved in parallel. The ultimate goal of the execution strategy is to schedule as many samples as possible on a given number of processors, for a minimal time. The problem can be formulated as a constrained optimization problem. In the case of MLMC in its general form the problem is $NP - complete$ **[5]**. An efficient solution requires simplifications and assumptions. MLMC approach by design defines three distinct parallel layers:

- (i) Parallelizing the solution for each sample (solving a deterministic PDE in parallel).

- (ii) Parallelizing the solution of all samples on a given MLMC level. (parallel on different levels)

- (iii) Parallelizing at all or several MLMC estimators simultaneously.

The most efficient and flexible strategies will be those, taking advantage of all of the three layers of parallelism.

Performance parameters

The design of the scheduling strategy requires different parameters to be taken into account. The most prominent ones are the number of the different estimators, the samples per estimator for a current *estimate-solve* cycle and the time to compute one sample for a given estimator. In the case of a sample solved by multiple processes the parallel solver inefficiencies contribute to the overall time lost for communication and synchronization. The efficiency and the predictability of the underlying solver is crucial for the performance of the scheduler. Modern multi-grid solvers scales very well under reasonable assumptions **[5]**. Assuming there is no parallel computation overhead and no inefficiency lost due to load imbalances, the theoretical minimum computational time is given by:

$$T^{min} = \frac{1}{P^{all}} \sum_{l=0}^{L} N_l E[t_l] \tag{5.2}$$

where P^{all} is the number of the available processes for parallel computation. N_l is the number of samples that have to be performed in level l and $E[t_l]$ is the expected time for solution of a single sample on a single process.

In the case of a single process per sample per level, the minimum time can be computed directly by recording the time to solve the sample. In the case of more than one processor per problem, the minimal time can only be estimated. Assume θ is a measure of how effective the underlying parallel

sample solver algorithm is. Then the time to compute a sample in parallel can be expressed as:

$$C_l = \theta_l C_l^{min} \tag{5.3}$$

where in equation **5.3** C_l^{min} is the time to compute a single sample on level l on P_{min}^l processors. Rewriting equation **5.2** by substituting t_l with C_l, the equation becomes:

$$T^{min} = \frac{1}{P^{all}} \sum_{l=0}^{L} N_l E[C_l] = \frac{1}{P^{all}} \sum_{l=0}^{L} N_l E[\theta_l C_l^{min}] = \frac{1}{P^{all}} \sum_{l=0}^{L} N_l \theta_l E[C_l^{min}] \tag{5.4}$$

In the case when $\theta_l = 1$ for all levels the two formulations are identical. In the case when more than one process is assigned to a sample on level l, to compute the minimal time the θ_l needs to be determined. This can be done in a pre-processing phase, by computing the θ_l function for a given scalability window $\{P_{min}^l, \ldots, P_{max}^l\}$, where P_{max} is the maximum number of processors that achieve the desired threshold efficiency, and P_{min}^l is the minimal number of processes that are able to solve the problem by fully utilizing the memory capacity. By setting C_l^p to be the time to compute one sample on level l by p processors, θ_l becomes:

$$\theta_l = C_l^p / C_l^{min} \tag{5.5}$$

To define the parallel efficiency of a given level of $MLMC$, the relative inefficiency is computed by substituting the computational time C_l^{comp} with the minimal level time T_l^{min} and the resulting value is divided by T_l^{min}. By this way, it can be estimated what is the percentage of the time lost to

synchronization relative to the minimal computational time. If the computed time is close to the minimum time, the value of this fraction will be close to 1. The lost time for distribution and synchronization will lead to values grater then 1. To get a decreasing function, the fraction is subtracted from 1:

$$Eff_l = 1 - (C_l^{comp} - T_l^{min})/T_l^{min} \tag{5.6}$$

Expressing **5.6** in terms of θ and C_l^p becomes:

$$Eff_l(\theta, p) = 1 - \frac{(C_l^{comp} - N_l\theta_l E[C_l^{min}])}{(N_l\theta_l E[C_l^{min}])} \tag{5.7}$$

Finally the MLMC efficiency is defined as a sum over the levels:

$$Eff(\theta) = 1 - \frac{(\sum_l C_l^{comp} - \sum_l N_l\theta_l E[C_l^{min}])}{\sum_l (N_l\theta_l E[C_l^{min}])} \tag{5.8}$$

where $l \in \{0, \dots L\}$.

Layers of execution and parallel models

To classify the parallel scheduling strategies, three main layers of parallelism are defined as in **[5]**:

- *Level parallelism:* Some or all of the estimators on levels $l = 0 \dots L$ can be computed in parallel
- *Sample parallelism:* Some or all of the samples $\{S_l^i\}_{i=0}^{N_l}$ can be computed in parallel.
- *Solver parallelism:* The PDE solver, that computes a single sample, can be done in parallel.

The main loop of the program (see figure 5.4), that loops over the samples and levels is inherently parallel. Only a small post-processing step is required in which the results are accumulated and quantity of interest is computed. The challenge is to balance the workload across all of the parallel layers and to design a concurrent scheduler for the solvers, particularly in the case when no a priori information is available. Concurrent execution can be classified as the number of layers that the model uses. The single-layer approach offers no flexibility and is prone to significant load imbalances. The two-layer approach gives more flexibility. The two layers can be either Level-Sample or Sample-Solver. Typically, for simulation with sufficiently large samples, that have large solver scalability windows and in a total small number of samples, the Sample-Solver model can have significant parallelization potential. The most flexible approach will have to take into account all of the layers of parallelism.

5.3 Scheduling strategies

To design the parallel models, the bulk-synchronous model is adopted. The focus will be on dynamic approaches. They try to improve processor distribution by a greedy scheme. All the designs have to account that each estimation step (see figure 5.1) needs information from all levels, as formula **2.15** suggests. The variance from each level is required, to estimate the number of samples that needs to be computed. This means for each parallel region of the algorithm (see figure 5.4), a synchronization must be done across all levels and all of the available processors. In this synchronous part the statistical moment exchange is performed and the number of required additional samples is calculated. The overall time of this synchronous part is negligible compared to the total run time of the simulation. In the performed experiments this type of estimate occurs on average $5-6$ times. Before considering any scheduling strategies, two important practical aspects of the implementation have to be considered. The first one is the *random number generator*. To generate random permeability fields a random uncorrelated

vector has to be generated. In the implementation this is done by generating a random seed using $C++$ standard library class *random_device*. This is a uniformly-distributed integer random number generator. Each process gets a seed generated at the beginning of the simulation. As a pseudo-random number generator, a standard Mersenne Twister engine with a word length of 64 bits, and a period of $2^{19937} - 1$, is used. Please note that when more than two processes are solving a single sample, the seeds they use are still different. The second aspect is communicator creation. The split of the MPI processes between the different layers of parallel execution is done by API call to *MPI_Comm_split*. The accumulation of the results is done with collective communication. The redistribution of the data within one sample is done using MPI one-sided communication with no locks. This enables multiple messages to be exchanged within a single synchronization procedure.

5.3.1 Level-Solver synchronous (LvlSolSyn)

In this scheduling strategy, the parallelism is done only at the sample layer of parallelism. For this strategy, each PDE is solved by a single process with the assumption that the compute time for each solution is constant. For this strategy, the different estimators are computed sequentially per *estimate-solve* cycle. Each estimator is treated as a pure Monte Carlo and the samples are computed in parallel. The work is divided equally across all the MPI processes. Each processor gets the same number of samples rounded down to integer: $N = N_l/P_{max}$. If the number of samples is not exactly divisible to available processors or if the number

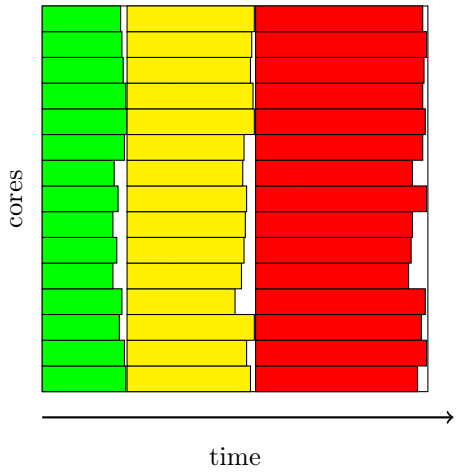


Figure 5.6: LvlSolSyn Time-Processor diagram

of samples needed is smaller than the available MPI processes, then part of the processors may be idle for the current *estimate-solve* execution. Because formula **2.15** gives the minimum number of samples needed per level to achieve the desired tolerance, filling the idling MPI processes with samples is also possible. This can improve the overall speed. Consider the case where the number of the estimated samples is not sufficient to achieve the desired tolerance, then the scheduled samples on the idling MPI processes, will reduce the total number of samples that needs to be computed on the next *estimate-solve* cycle. In the other case, when the estimated number of samples is sufficient for achieving the desired tolerance, the additional samples will lead to reduced RMS error with no additional time cost. For this scheduler strategy samples are scheduled by formula $N = \lceil N_l / P_{max} \rceil$.

5.3.2 Level synchronous homogeneous (LvlSynHom)

In this parallel strategy, the parallelism is done at two levels: sample and solver. Here again, no time variations per PDE solve is assumed. In this parallel scheme, each PDE solution on level l is solved by a p_l^g number of MPI processes within certain scalability window $\{P_l^{min} \dots P_l^{max}\}$. As in the *Level-Solver synchronous scheme*, estimators are computed sequentially. The idea here is to group a certain number of MPI processes (if it is possible, as they are physically close) to work together on a single PDE problem.

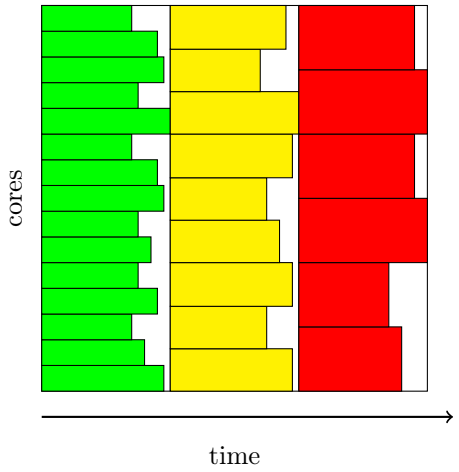
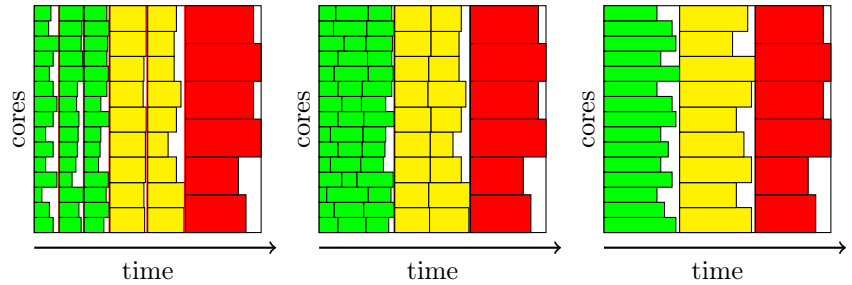


Figure 5.7: LvlSynHom Time-Processor diagram

The samples, that have to be computed on a given level, are divided equally among the groups, that can be formed by the P^{total} number of MPI processes. Each MPI group on level l gets $N = N_l / \lfloor (P_{max}/p_l) \rfloor$. Ideally each p_l^g must divide P^{total} exactly, in order to avoid idling processes. This restricts the

scalability window to a subset of the original. Here again as with the *Level-Solver synchronous* model, to avoid idling processor groups, the idling groups are filled with samples. In this scheduling strategy, the intention is that the imbalances of the workload will be smaller because of the grouping. In the ideal case, where each PDE solution scales perfectly and there is no time loss to synchronization, the *Level synchronous homogeneous* approach, will complete the MLMC simulation for the same amount of time, as *Level-Solver synchronous* approach. However, this approach is more flexible than the single-layer approach. When the available memory on a given node is insufficient to store the PDE problem, to avoid resource wastage, a parallel sample computation must be performed. Besides, in a more realistic situation where the time between two samples on the same level variates, *Level synchronous homogeneous* is more robust than *Level-Solver synchronous* scheduler. Each MPI group has more samples to compute in total but the time to compute one sample is less. This will lead to a smaller sample to sample computational time fluctuations.

The update of the accumulated quantity of interest between two *estimate-solve* cycles for both strategies can be done in different ways. As illustrated on figure **5.8** a simple idea is to schedule a chunk of K samples out of N, to be computed for each group (the group size of *Level-Solver synchronous* schedule is 1) before an update is done.



(a) $K = 1$ (b) $1 < K < N$ (c) $K = N$

Figure 5.8: Different update strategies for *Level-Solver synchronous* and *Level synchronous homogeneous* schedulers

In the extreme case of $K = 1$, $\lfloor (P_{max}/p_l) \rfloor$ samples are computed in parallel and then accumulated to the statistics. This case is not practical as it will require too much synchronization in the form of message exchange. Besides each step will be determined by the slowest sample. A single delay in the computation of one of the samples will directly impact the overall computational time. The other extreme case where $K = N$ requires only one synchronization, but it is more prone to inaccurate sample estimations by formula **2.15**.

5.3.3 Level synchronous heterogeneous (LvlSynHet)

In this approach, again the two layers of parallelism - sample and solution are considered. Again for simplicity, assume no run-time solution deviations are present. In this scheme the idea is to further try to optimize the processor distribution over a given estimator, by constructing groups of different sizes, instead of constructing groups of the same size, like in the case of *Level synchronous homogeneous* scheme. The algorithm consists of pre-processing step and optimization step. In the pre-processing step, a computation is performed and a table is constructed with averaged compute times of a single sample for a different number of MPI processes. This is performed by considering each of the available processor configurations within the estimator scalability window. For each configuration, an apriori number of samples is solved and the averaged times are logged in to the table:

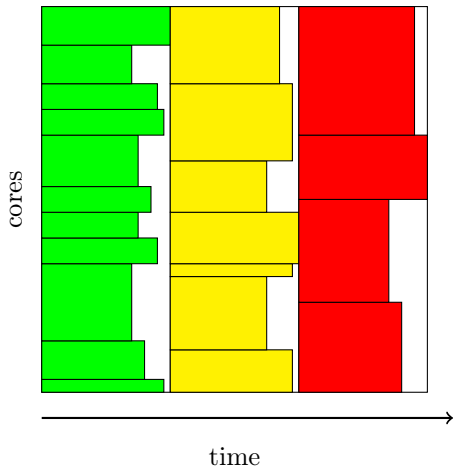


Figure 5.9: LvlSynHet Time-Processor diagram

P_l^{min}	$P_l^{min}+1$	$P_l^{min}+2$	$\ldots$	P_l^{max}
t_l^{min}	t_l^{min+1}	t_l^{min+2}	$\ldots$	t_l^{max}

In the optimization step, first the estimated minimal run time by formula **5.8** is computed. Knowing the optimal time, the maximum number of samples that each group of MPI process can compute can be determined. This is done by dividing the optimal time by the pre-computed time, rounded down to integer (see table **5.1**).

P_l^{min}	$P_l^{min}+1$	$P_l^{min}+2$	$\dots$	P_l^{max}
$\lfloor t^{opt}/t_l^{min} \rfloor$	$\lfloor t^{opt}/t_l^{min+1} \rfloor$	$\lfloor t^{opt}/t_l^{min+2} \rfloor$	$\dots$	$\lfloor t^{opt}/t_l^{max} \rfloor$

Table 5.1: Maximum number of samples for given optimal time

Finally, to obtain, the group distribution an optimization problem (equation **5.9**) is solved:

$$\begin{aligned} &\max\{\sum_{i=0}^{M-1} \lfloor t^{opt}/t_l^{min+i} y_i \rfloor t_l^{min+i}\},\ where\ M = P_l^{max} - P_l^{min} + 1 \\ &\text{with constraints:} \quad \sum_{i=0}^{M-1} (P_l^{min}+i) y_i \leq P^{total} \end{aligned} \tag{5.9}$$

The idea behind equation **5.9** is to select a linear combination of groups within the scalability window, such that each group work time is as close as possible to the optimal time. A significant disadvantage of *Level synchronous heterogeneous* strategy is the computationally intensive pre-process step. Acquiring reliable run times configuration requires a lot of samples to be performed by each group within the scalability window. To overcome this limitation, a fit of cost-function that approximates the times can be considered, or table construction can be performed a on smaller grids, where PDEs can be computed faster and then the values are extrapolated to fit the simulation.

5.3.4 Dynamic strategy

In this section, a greedy scheduling strategy is considered. The scheme adopts during the simulation. It is very flexible as it can be combined with any of the previous strategies. All of the parallel layers are taken into account. This strategy assumes that the first *Estimate-Solve* cycle has been completed and empirical variance and sample computational time are available. Let L be the number of the levels, and $N_i, i = \{0, 1, 2, \ldots, L\}$, be the number of required realizations per Monte Carlo estimator - $\widehat{Y}_l$, where N_0 is the number on the coarsest estimator $\widehat{Y}_0$. Let p_i be the number of processes allocated per $\widehat{Y}_i$, $p^g_{l_i}$ the respective group size of processes working on a single realization, with t_i be respective time constants, for solving a single problem once on a single process and finally with P^{total} the total number of available processes. Then the total CPU compute time for the current *Estimate-Solve* cycle, can be estimated:

$$T^{total}_{CPU} = N_0 t_0 + N_1 t_1 + N_2 t_2 + \cdots + N_L t_L \tag{5.10}$$

Then the optimal compute time per processor is:

$$T^{p}_{CPU} = \frac{T^{total}_{CPU}}{P^{total}} \tag{5.11}$$

By dividing the CPU time needed for estimator $\widehat{Y}_i$ by T^p_{CPU}, a continuous value for the number of processes on a given MC estimator is obtained.

$$p^{ideal}_i := \frac{N_i t_i}{T^p_{CPU}} \text{ for } i = \{0, 1, 2, \ldots, L\} \tag{5.12}$$

Lets further assume that all of the available processors must be distributed on all of the estimators $\widehat{Y}_i$. Then by, rounding down to integer the p^{ideal}_i

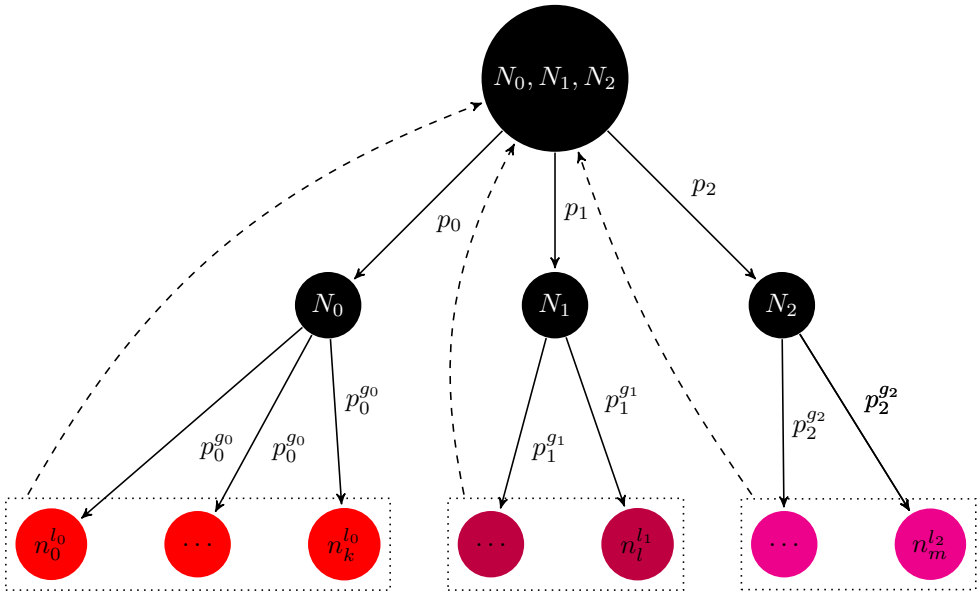


Figure 5.10: Schematic sample distribution of MLMC on three levels

value, a processors distribution for across the levels can be obtained

$$p_i := \left\lfloor p_i^{ideal} \right\rfloor, \text{ for } i = \{0, 1, 2, \ldots, L\} \tag{5.13}$$

Depending on the scheme that will be used for parallel computing on the estimator $\widehat{Y}_i$, additional restrictions may be imposed for p_i. For example if *LvlSynHom* scheme is used, p_i has to be divisible to the fixed group size: $p_i \equiv 0 \ (mod \ p_l^{g_l})$. Regardless of the estimator scheme, the unallocated processors, due to rounding, can be left unused for this cycle. To improve the estimation and search for a better approximation, the set of all upper and lower bounds for each of the estimators is constructed. In other words, an integer solution is searched in the range between $\sum_{i=0}^{2} p_i$ and p^{total}.

Till now the only considered case is the distribution of all of the available processors to work simultaneously on all of the estimators. This may not be the optimal strategy. It is rarely the case, because of the strong imbalance of work between the estimators. To find a reasonable strategy of all possible combinations of unions of elements from the estimator power set is considered. This ensures that cases when all of the processors are allo-

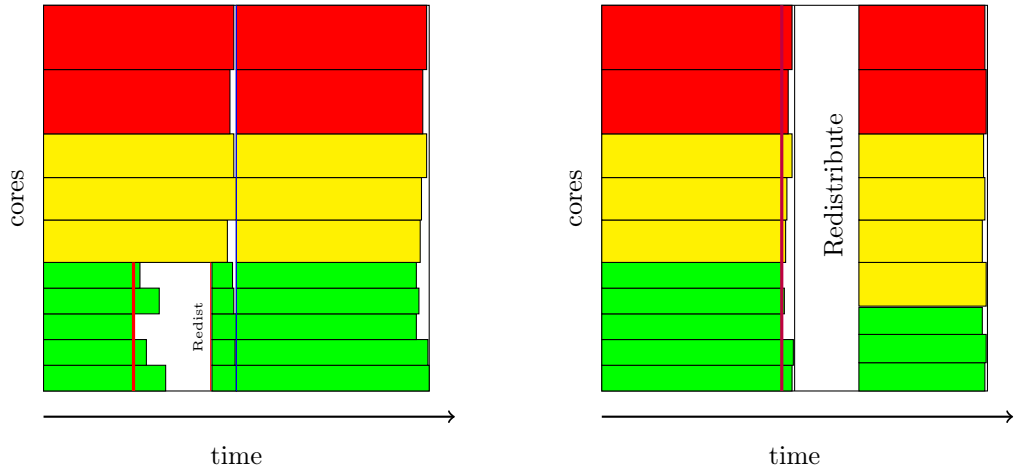
cated on the coarsest level $\{\widehat{Y}_0\}$ and then levels $\{\widehat{Y}_1, \widehat{Y}_2\}$ are concurrently computed, then just trying to balance the processors on all the estimators together $\{\widehat{Y}_0, \widehat{Y}_1, \widehat{Y}_2\}$ for MLMC on three levels. For the parallel strategy for the estimator, depending on the scenario, allows different schemes to be used. *LvlSynHom* can be used when there are no large time variations between the samples. If the times for computing samples for the scalability windows of the samples on all of the estimators are known, *LvlSynHet* can be considered.

At each *Estimate-Solve*, the samples average solving times are updated across the estimators and processor distribution on the level layer of parallelism is adopted according to the number of samples and the averaged times. To incorporate the time fluctuations across different samples on a given estimator, different strategies are considered. The scheduler behavior is modified to achieve faster computation, at the price of additional communication.

5.3.5 Interrupted Dynamic strategy

Here a strategy, very similar to the idea of processor interruptions is considered. The algorithm starts as a standard dynamic strategy, as described in the previous section **5.3.4**. Again heuristical assumption is made, that there is no sample to sample computational differences, or if present, they will balance out, during the parallel computation. During the parallel computation, due to load imbalances and sample to sample fluctuations, a group (or groups) of MPI process among all of the MPI processes can complete computation before the others. Upon completion, this group sends a signal to a part of the other groups or all of the groups, that are still computing, informing them that it is in an idle state. Upon receiving the signal, the computing groups interrupt the current computation. When all those message exchanges completes and all of the groups are in an idle state, rescheduling is done. For this type of optimization two strategies are considered. The first type is the *local interruption* strategy, done within an estimator and the second type is the *global* one - computation on all MPI processes on all the

levels stops and then rescheduling is performed. Figure **5.11** illustrates the idea.



(a) *Local*(*Estimator*) (b) *Global*(*MLMC*)

Figure 5.11: Schematic overview of the interruption process.

In practice, the signals are modeled as a message exchange between the groups, by a master-slave approach. A group sends a message to the designated master processor to notify it is in an idle state. The master process takes responsibility to inform the other groups and synchronize the data between them. The messages carry only a small amount of meta-information and the dominating part of the time, needed for this process to complete, will be the latency of the messaging system. This means, that a local interruption will have not only a smaller amount of messages, but smaller exchange times due to the physical proximity of the processors. This approach can be viewed as a way to guarantee that there will be no idle processors.

5.3.6 Job queue Dynamic strategy

This strategy simulates the idea of job dispatching, or task-based parallelism in the multi-thread environment, adopted to MPI message system. First, an optimal distribution of processors per estimator is obtained by equation **5.12**. For each estimator, using the "master-slave" programming paradigm,

one of the available MPI processes is set to be master, and the others are set to be slaves. The master process acts as a dispatcher, that assigns work to the other processes. Each of them performs a given tasks and then reports back to the master for more work. This way each process is busy working regardless of the time for a sample. This is at the expense of a large number of small message exchanges. The slaves can be organized in groups of equal size, to work together on a single sample or groups of different sizes. The number and the sizes of groups can be determined by solving the optimization problem given by equation **5.9** from *LvlSynHet* model.

5.4 Review of the parallel strategies

The table **5.2** gives a short summary of the considered methods. For each method, the parallel layers used are marked in columns *Level, Sample and Solver*.

# Method	Level	Sample	Solver	Comment
# LvlSolSyn	-	*	-	Most basic scheduler, not very efficient
# LvlSynHom	-	*	*	More robust, may be inefficient
# LvlSynHet	-	*	*	Very hard to compute, a lot of communication
# Dyn + 2 layer sched.	*	*	*	Does not incorporate sample to sample time deviations
# LocDyn + 2 layer sched.	*	*	*	Local(Within level) interruption
# GlobalDyn + 2 layer sched.	*	*	*	Global(Across levels) interruption
# QueueDyn	*	*	*	Dynamic job dispatching, expensive communication

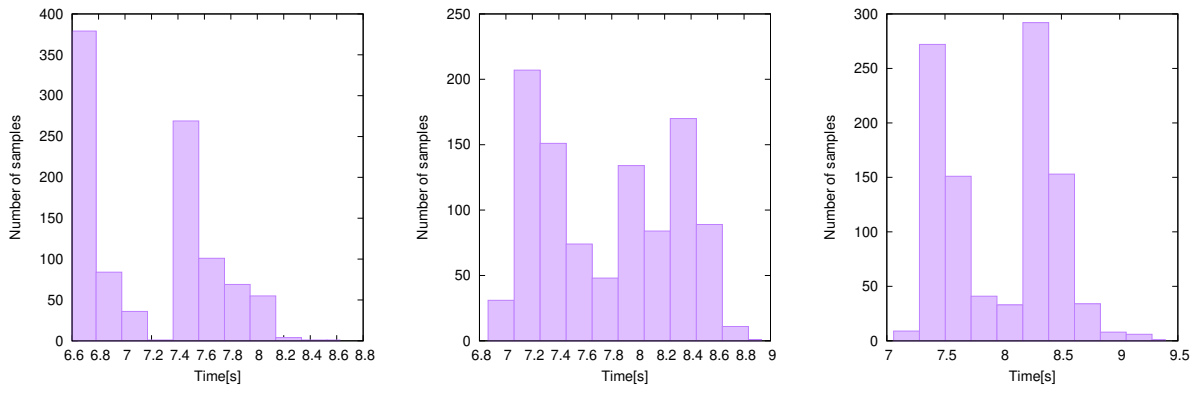
Table 5.2: Summary of the different parallel strategies.

5.5 Parallel experiments for Laplace equation

The parallel experiments for the Laplace equation are divided in three main groups. The first group of experiments, examinates the time deviations and fluctuations on a sample to sample basis. Table containing computed θ function is also considered for different configurations of parameters. The second group of experiments is devoted to the MLMC parallel efficiency under equation **5.8**. Different configurations of processor distributions is considered. In the last part, strong scalability results for the algorithm are shown.

Time to solution and scalability windows

On figure **5.12**, time to solution histograms are presented. The stochastic parameters considered are with increasing variance from left to right. Each test contains 1000 samples. The more uncertainty in the system leads to more difficult overall computation. The minimum and maximum time to compute a sample slowly increase, but overall distance stays relatively unchanged - at around 2 seconds. In all three cases the data is very dispersed. The maximum time deviations value is at around 25% of the mean time.



(a) $\sigma = 1,\ \lambda = 0.2$ (b) $\sigma = 2,\ \lambda = 0.2$ (c) $\sigma = 3,\ \lambda = 0.2$

Figure 5.12: Histogram of solution sample times

On tables **5.3, 5.4, 5.5**, the θ value is computed for the same stochastic parameters as in figure **5.12**. The experiment shows very good scalability

for the conjugate gradienet linear solver. The overall computational time changes slightly for a fixed stochastic parameters. The main contributing factor to the time needed is the grid resolution.

#N of cores:	1	2	3	4	5	6	7	8
θ, grid: $2^7 \times 2^7$:	1	1.813	2.391	3.055	3.321	3.705	4.009	4.526
θ, grid: $2^8 \times 2^8$:	1	1.900	2.680	3.573	4.139	4.798	5.440	6.341
θ, grid: $2^9 \times 2^9$:	1	1.938	2.804	3.91	4.536	5.397	6.132	7.312
θ, grid: $2^{10} \times 2^{10}$:	1	1.942	2.808	3.816	4.534	5.333	6.183	7.365

Table 5.3: Scalability for $\sigma = 1,\ \lambda = 0.2$

#N of cores:	1	2	3	4	5	6	7	8
θ, grid: $2^7 \times 2^7$	1	1.778	2.388	2.994	3.352	3.727	4.015	4.466
θ, grid: $2^8 \times 2^8$	1	1.911	2.744	3.594	4.258	4.966	5.608	6.347
θ, grid: $2^9 \times 2^9$	1	1.971	2.914	3.940	4.756	5.675	6.453	7.510
θ, grid: $2^{10} \times 2^{10}$	1	1.980	2.947	3.779	4.809	5.713	6.569	7.584

Table 5.4: Scalability for $\sigma = 2,\ \lambda = 0.2$

#N of cores:	1	2	3	4	5	6	7	8
θ, grid: $2^7 \times 2^7$	1	1.791	2.394	2.988	3.324	3.707	4.031	4.476
θ, grid: $2^8 \times 2^8$	1	1.924	2.772	3.598	4.311	5.015	5.586	6.395
θ, grid: $2^9 \times 2^9$	1	1.970	2.987	3.946	4.758	5.746	6.540	7.511
θ, grid: $2^{10} \times 2^{10}$	1	1.969	2.925	3.897	4.837	5.759	6.612	7.587

Table 5.5: Scalability for $\sigma = 3,\ \lambda = 0.2$

Parallel efficiency metrics

In this subsection, an investigation of the efficiency is done for different scheduler types. The parallel efficiency is measured under equation **5.8**. For this tests, the finest grid of the MLMC algorithm is $2^{10} \times 2^{10}$ number of cells. This is approximately 10^6 unknowns. The number of levels of MLMC is set to 4.

Table **5.6** contains the efficiency results for $Eff(1,1,1,1)$ and single processor per problem. Table **5.7** is shows the data for the test where larger stochastic parameters are used. This leads to larger uncertainty in the system and more computationally challenging problem. The processor distribution is the

same. On table **5.8** same problem as table **5.7**, but with different processor distribution, is considered with $Eff(1, 3.6, 7.35, 9.00)$. For this parameters the required processors per problem is: $p_0 = 1, p_1 = 5, p_2 = 9, p_3 = 11$. Under this processor distribution, each sample is solved with expected efficiency at least: $1, 0.78, 0.79, 0.80$ respectively. The efficiency is computed via equation **5.6**.

# n Cores	168	252	336	420	504
# Dyn + LvlSolSyn	0.928	0.875	0.822	0.734	0.615
# LocDyn + LvlSolSyn	0.969	0.958	0.889	0.749	0.912
# GlobalDyn + LvlSolSyn	0.970	0.954	0.942	0.922	0.680
# QueueDyn	0.963	0.953	0.953	0.882	0.746

Table 5.6: Parallel efficiency for $\sigma = 2,\ \lambda = 0.3,\ \epsilon = 1e-3$

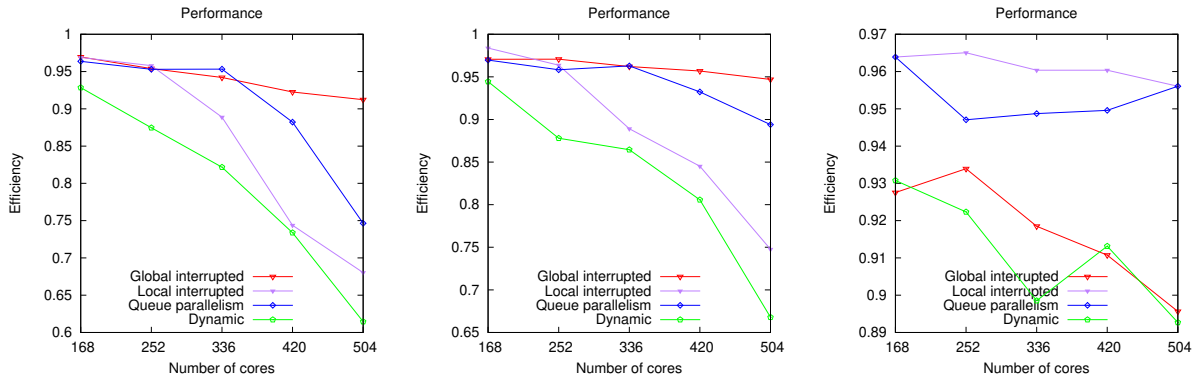
# n Cores	168	252	336	420	504
# Dyn + LvlSolSyn	0.944	0.878	0.864	0.805	0.667
# LocDyn + LvlSolSyn	0.984	0.963	0.889	0.956	0.947
# GlobalDyn + LvlSolSyn	0.970	0.970	0.962	0.845	0.747
# QueueDyn	0.960	0.958	0.962	0.932	0.893

Table 5.7: Parallel efficiency for $\sigma = 2.25,\ \lambda = 0.3,\ \epsilon = 1e-3$

# n Cores	168	252	336	420	504
# Dyn + LvlSolSyn	0.931	0.922	0.899	0.893	0.892
# LocDyn + LvlSolSyn	0.964	0.965	0.912	0.956	0.895
# GlobalDyn + LvlSolSyn	0.928	0.933	0.960	0.896	0.956
# QueueDyn	0.939	0.947	0.948	0.951	0.951

Table 5.8: Parallel efficiency for $\sigma = 2.25,\ \lambda = 0.3,\ \epsilon = 1e-3$

Tables **5.6 and 5.7** show that local interrupted approach achieves best efficiency for a relatively small number of processors. These efficiency values, nonetheless, are not kept when the number of processors increases. In the case of 504 cores, the effectiveness drops from 7 to 8% better than simple dynamic approach. There are two main reasons for that: the increased number of processors leads to larger load imbalances between the different levels when



(a) Efficiency from table 5.6 (b) Efficiency from table 5.7 (c) Efficiency from table 5.8

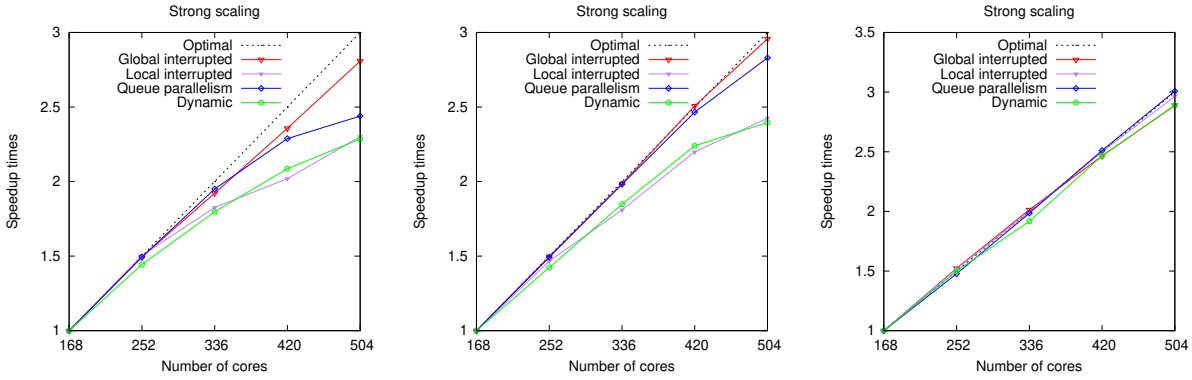
Figure 5.13: Graphical representation of tables **5.6, 5.7 and 5.8**

the number of processors per level is estimated. Even if the algorithm finds a good processor distribution that leads to small waiting time per processor level groups, this time has much more impact on the effectiveness, compared to the same waiting time on a smaller number of processors. Simply, there are more idling processors. The other reason is a large number of messages exchanges at a single point in time. These messages must be sent through the network that can lead to flooding the communication canal. This problem becomes apparent when the queue method is compared to the local one. Although there are much more messages, that are exchanged between the processors, all of them have small byte sizes and are spread in time, leads to better overall efficiency for the queue scheduler. The best performing algorithm that is considered for the case of a single processor per problem is the global technique. It shows small performance degradation when the number of processors is increased. Table **5.8** shows that utilizing the third layer of parallelism and thus counting all layers of parallel execution. This leads to very efficient algorithms. All of the considered methods achieve more than 0.89 units of efficiency. In this case, the local interrupted technique outperforms the others. The increased effectiveness comes again from the fact that groups of processors are considered by the dynamic scheduler, rather than all

the processors. Part of the imbalances is offloaded to the underlying parallel algorithm used for the generation of a single sample.

Strong scalability experiments

In this section main focus is the strong scalability of the algorithm. On figure **5.14** the strong scalability performance of the algorithm is plotted for the tests considered in tables **5.6, 5.7 and 5.8**.



(a) Scalability from table 5.6 (b) Scalability from table 5.7 (c) Scalability from table 5.8

Figure 5.14: Graphical representation of tables **5.6, 5.7 and 5.8**

The experiments show good scalability for all of the considered methods if all of the layer of parallelism are used. Although for the cases of local interuped and dynamic schedulers lack behind the other two schedulers in the case of 2 layer parallelism (**5.14** a), b)), the speedup is close to 2.3 times of maximum 3, which is still good. Figures **5.13** and **5.14**, shows that the efficiency and scalability metrics are correlated, however it seems that they are not linearly correlated.

Experiments with large number of cores for Laplace equation

The tests in this section are performed on the SuperMuc cluster hosted at TUM Munich. Each node is consisting of 48 cores. The total node memory is $96GB$. The processor model is *Intel Skylake Xeon Platinum 8174* [1].

[1] https://doku.lrz.de/display/PUBLIC/Hardware+of+SuperMUC-NG

Figure **5.15** shows the results for the simulation time, with different coarse grain methods and figure **5.16** shows the efficiency and the scalability of the considered methods. The simulation parameters are summarized in table **5.9**.

Max. grid size	$2^{10} \times 2^{10}$	σ	2.75	λ	0.3	ϵ	1e-3
Cores per problem			Lvl. 0	Lvl. 1	Lvl. 2	Lvl. 3	Lvl. 4
			1	2	3	4	5

Table 5.9: Simulation parameters for figures **5.15** and **5.16**

The experiment shows optimal scalability for both of the considered methods. The efficiency of the averaging approach is better in all of the cases. This can be explained by the overall larger simulation times, compared to the simplified renormalizaion idea. Considering the small simulation time, appropriately 1000 seconds in the case of 3840 cores, efficiency remains very good. Since the dynamic scheduler does not consider any handling of the sample to sample inefficiencies, scheduling which considers them, will bring even better effectiveness. For sufficiently large problems, the dynamic approach is good enough on its own. This is evident by the effectiveness at 960 cores.

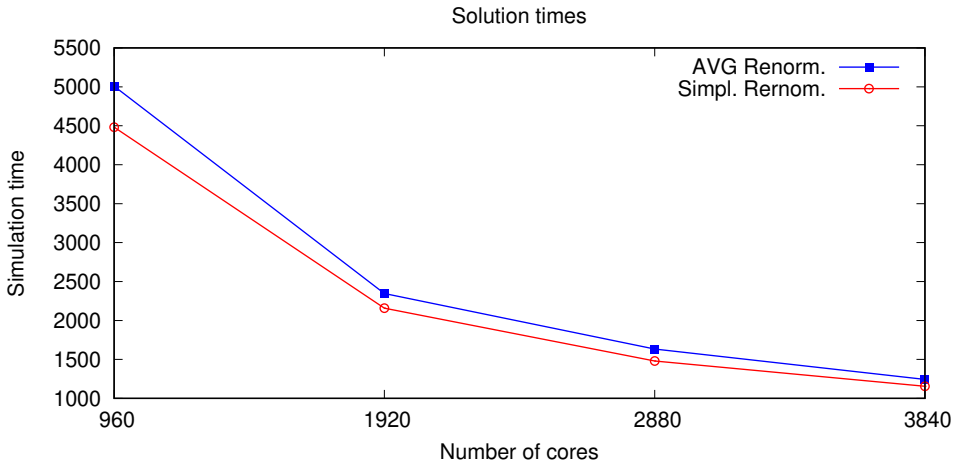
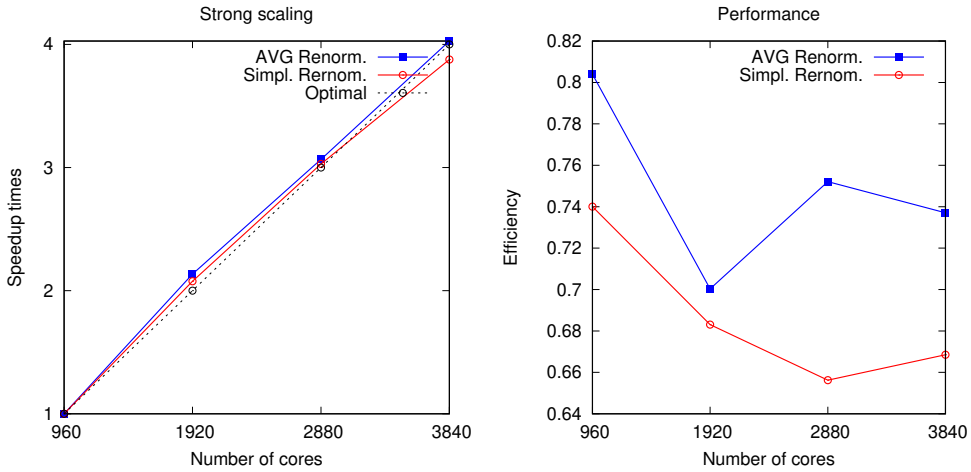


Figure 5.15: Simulation time



(a) Strong scale (b) Efficiency

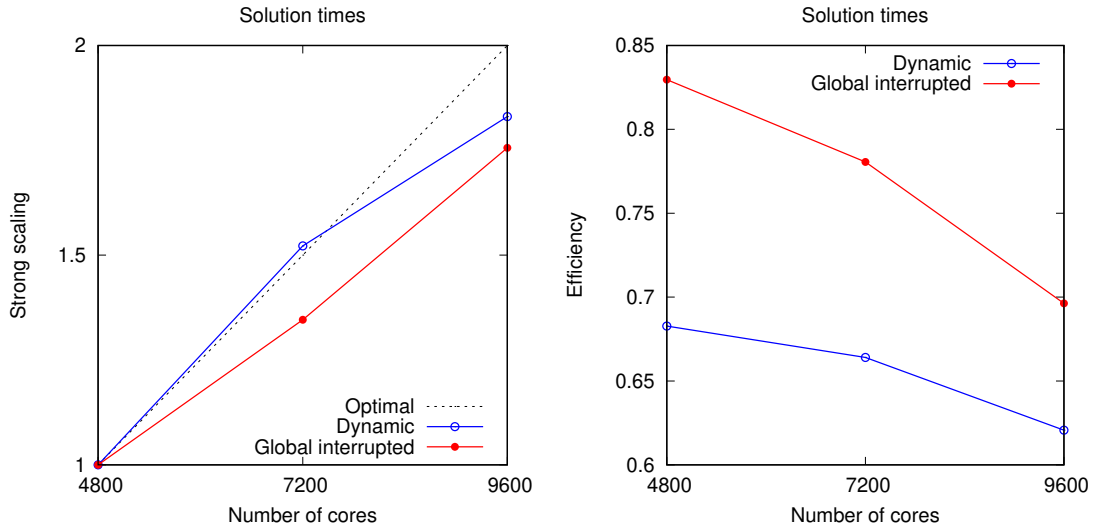
Figure 5.16: Dynamic scheduler for different coarse grain MLMC techniques

The test shown on figure **5.17** is designed to test the effect of the global interruption optimization of the dynamic scheduler for a large number of cores on a relatively small problem. In such case the time lost to synchronization will have more significant impact on the overall computational time and effectiveness respectively. The test parameters are summarized in table **5.10**. As a coarsening technique a simplified renormalization is used. The experiment shows good processor distribution among the different levels for the case of 4800 cores and thus efficiency for the case of dynamic scheduler. The achieved efficiency is close to 70%. Using global interrupt optimization leads to significant improvement in the efficiency - close to 15%. In the case of 7200 cores, the optimization gains around 12% improvement. In the case of 9600 the gain is around 7%. This steady decline of the gains is due to the increased communication time, and compared to the overall smaller computational time. In the case of 9600 cores the total simulation time is only 621 seconds with the global interrupt. More computationally expensive simulations will lead to even better effectiveness.

The test on figure **5.18** is very similar to the test shown on figure **5.17**. The key difference is the processor distribution of the cores per problem. In this

Max. grid size	$2^{10} \times 2^{10}$	σ	3	λ	0.3	ϵ	1e-3
Cores per problem				Lvl. 0	Lvl. 1	Lvl. 2	Lvl. 3
				1	1	1	1

Table 5.10: Simulation parameters for figure **5.17**



(a) Strong scaling

(b) Efficiency

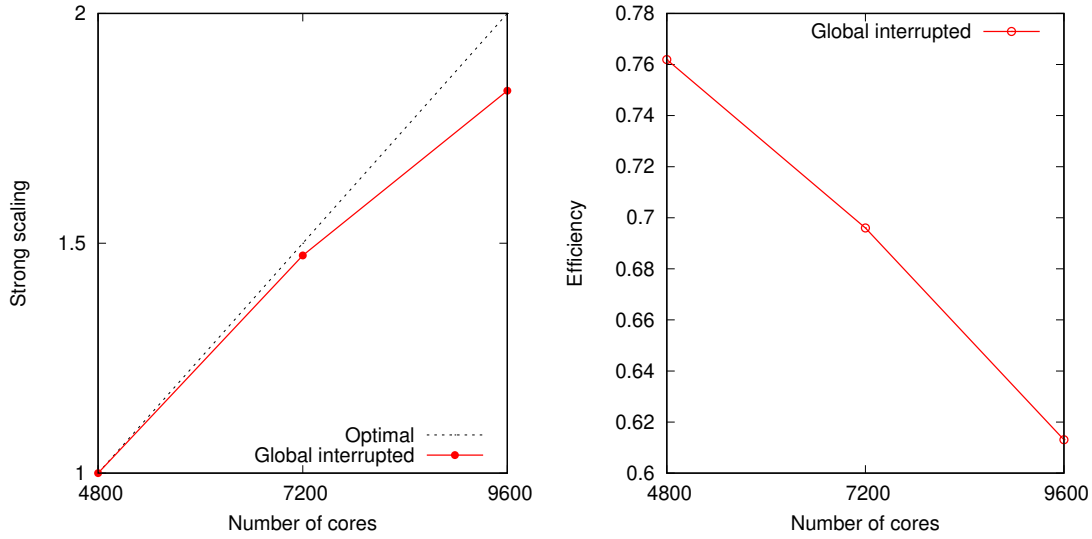
Figure 5.17: Large number of cores, single processor per problem

case a single problem on all levels, except the coarsest level is solved by more than one core. Again the simulation takes less than one hour on 9600 cores. The simulation parameters are summarized in table **5.11**. Compared to the previous case the total communication is much more, due to the addition of the communication within a PDE solution.

Max. grid size	$2^{10} \times 2^{10}$	σ	3.25	λ	0.3	ϵ	1e-3
Cores per problem			Lvl. 0	Lvl. 1	Lvl. 2	Lvl. 3	Lvl. 4
			1	2	3	4	5

Table 5.11: Simulation parameters for figure **5.18**

The test shows degraded parallel efficiency compared to the case of single core per problem, but better scalability.



(a) Strong scaling

(b) Efficiency

Figure 5.18: Large number of cores with single core per problem

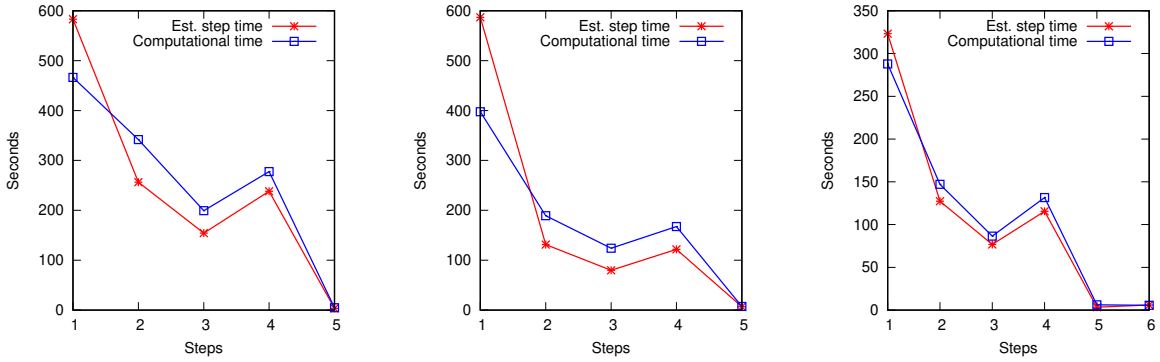
Figures **5.19 and 5.20** illustrates how the algorithm adopts during the different *Estimate-Solve* cycles. The simulation parameters are summarized in table **5.12**.

Max. grid size	$2^{10} \times 2^{10}$	σ	3	λ	0.3	ϵ	1e-3
Cores per problem				Lvl. 0	Lvl. 1	Lvl. 2	Lvl. 3
				1	1	1	1

Table 5.12: Simulation parameters for figures **5.19** and **5.20**

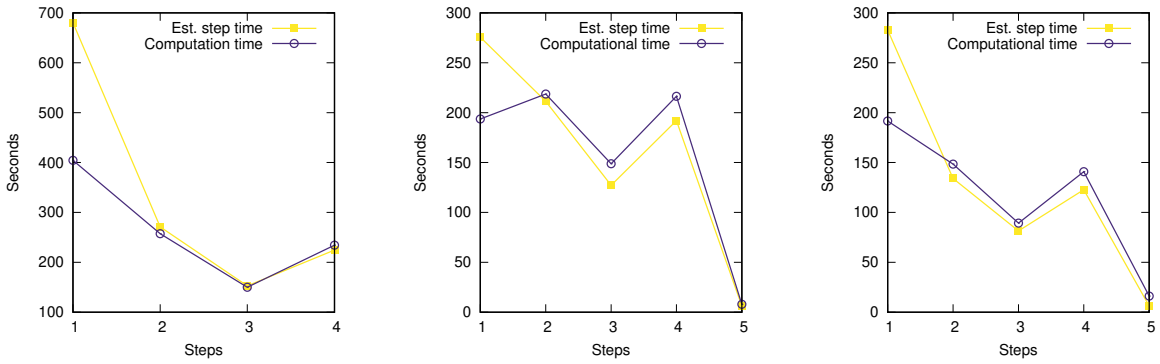
The predicted time from the algorithm is plotted against the actual compute time for that cycle. At the zero step, where the initial number of samples is computed to obtain empirical variance, there are no statistics and no prediction can be made. After this step a prediction is made, for the time that the next *Estimate-Solve* cycle will need and the processors are distributed accordingly. At the first step prediction time is quite inaccurate due to the very small sample size available. At later stages of the algorithm the gap between the predicted and actual time closes significantly. The test also shows that the relative distance between two graphs is smaller in the case of global interrupted approach. This is because in the case of interruption on

some level, the remaining work is rebalanced. This type of interruption can happen in two cases: either a group on a given level has finished its work or the total work on a given level has been finished. In either cases, since the work is rebalanced globally, the parallel overhead imbalances, that comes from the dynamic process distribution, are also accounted. This happens at the expense of global communication. In the case of 9600 cores this means that at given time point there will be at least 9600 messages circulating in the network. Furthermore, this messages are not evenly distributed among the processors which slows down even more the synchronization process.



(a) 4800 cores (b) 7200 cores (c) 9600 cores

Figure 5.19: Estimated and actual step time, no interruption



(a) 4800 cores (b) 7200 cores (c) 9600 cores

Figure 5.20: Estimated and actual step time, global interruption

5.6 Parallel experiments for convection-reaction-diffusion problem

To test the magnitude of the time fluctuations in a sample to sample solutions, a histogram of number of samples for given time interval is plotted on figure **5.21**. The simulation consists of 1000 samples on a grid of size $2^{10} \times 2^{10}$ and stochastic parameters: $\sigma = 2, \lambda = 0.2, Pe = 2.5, Da = 0.5$. The histogram shows large fluctuations in time. The minimal time to compute a sample is 39.5544 seconds and the maximum time is 54.6439 seconds. The difference between the two times is more then 10 seconds. Using parallel scheduler that does not take into account those fluctuations would be very ineffective.

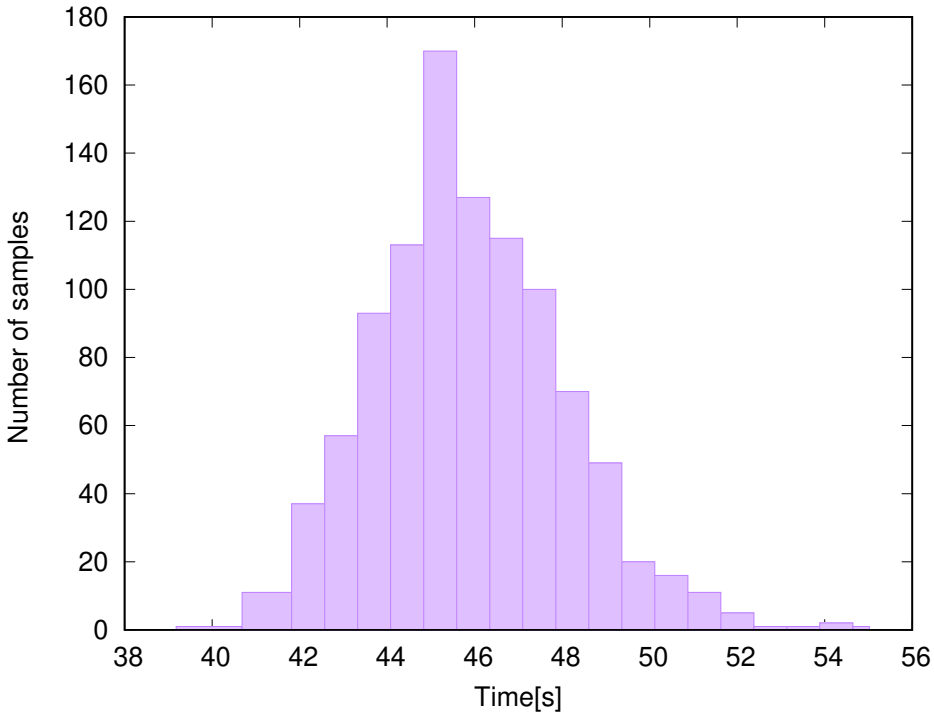


Figure 5.21: Time to solution histogram

On figure **5.22** a parallel MLMC performance with *Solve first then Renormalize* approach is shown. The estimator is a 3 level MLMC with simulation parameters $\sigma = 2, \lambda = 0.2, \epsilon = 5e-3, Pe = 1.5, Da = 0.5$ for the concentration. For the flow - 5 level MLMC with parameters $\sigma = 2, \lambda = 0.2, \epsilon =$

$5e-2, Pe = 2, Da = 0.5$. In both cases the fine grid size is $2^{10} \times 2^{10}$. An impacting factor for the parallel implementations is the fluctuations in the times of the solutions, as show on figure **5.21**, of the problems on a given level. However the performance of the scheduler is similar regardless, of quantity of interest.

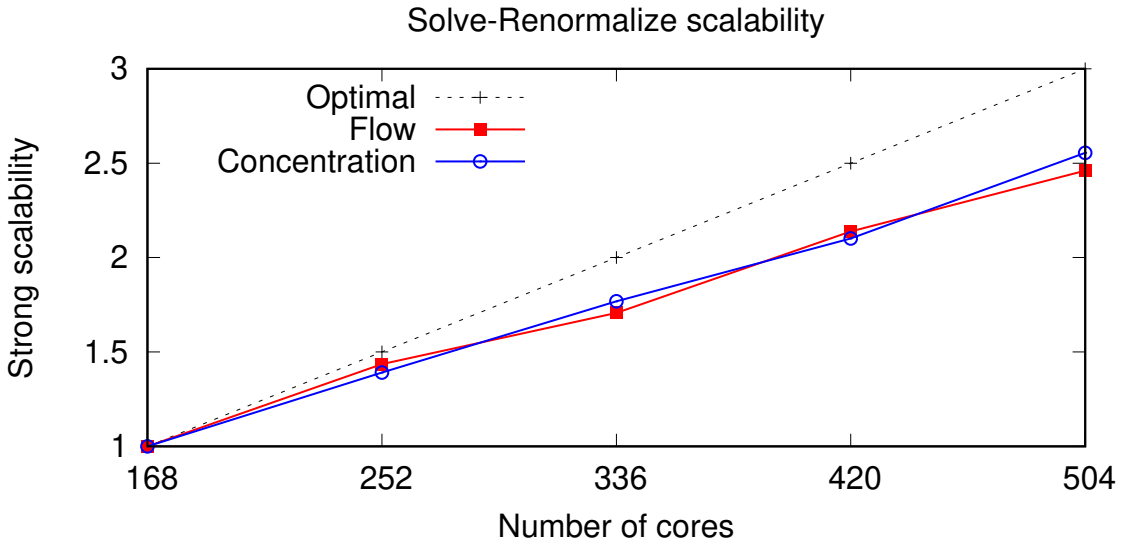


Figure 5.22: Performance for SR coarse grain approach

Table **5.13** contains the efficiency measured by equation **5.8** for the three different dynamic optimizations: *Local interrupted, Global interrupted, Queue parallelism*. Each sample is solved by a single processor. The testing parameters are: $\sigma = 2, \lambda = 0.2, \epsilon = 1e-2, Pe = 2.5, Da = 0.5$. The fine grid is $2^{10} \times 2^{10}$, and the estimator is a 4 *level* $MLMC$ with *renormalize first then solve approach*. The quantity of interest computed is the flow. The results shows very good efficiency for *Local interrupted* approach for 168 and 252 cores. It outperforms the other two methods. The efficiency significantly drops at higher number of cores, as both global and queue approaches retain lower efficiency drop when the number of cores increases. The average number of samples for this particular test are shown in table **5.14**

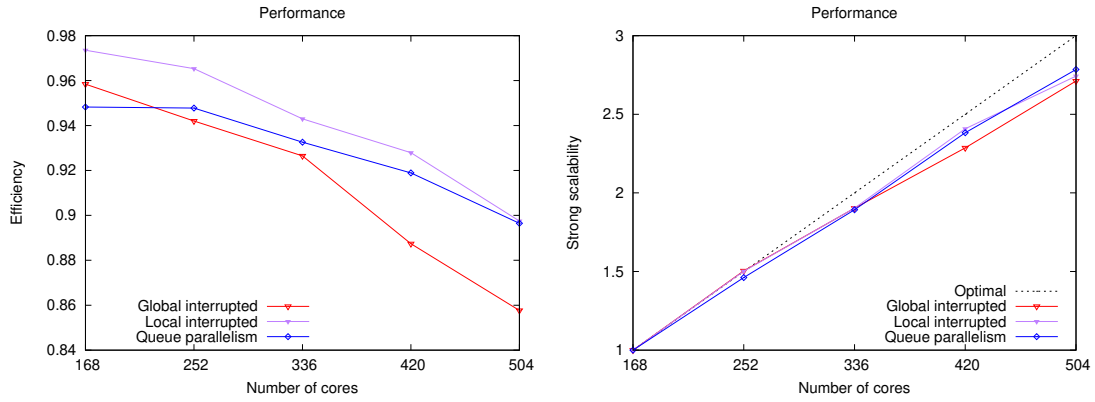
Figure **5.23** shows the graphical representation of the test data from table **5.13**.

# n Cores	168	252	336	420	504
# LocDyn + LvlSolSyn	0.974	0.965	0.898	0.845	0.709
# GlobalDyn + LvlSolSyn	0.958	0.942	0.926	0.887	0.858
# QueueDyn	0.948	0.948	0.933	0.919	0.896

Table 5.13: Parallel efficiency

# Level	0	1	2	3
# LocDyn + LvlSolSyn	444975	9466	3382	991
# GlobalDyn + LvlSolSyn	447649	9571	3359	962
# QueueDyn	449734	9398	3289	999

Table 5.14: Performed samples



(a) $Eff(1,1,1,1)$ (b) *Scalability*

Figure 5.23: Efficiency and scaling for table **5.13**

5.7 Conclusions

From algorithmic standpoint the MLMC algorithm is very similar to the pure MC approach. MLMC can be seen as collection of intendant MC estimators that work together to compute given quantity. However those MC estimators have different computational cost and are working on different levels. The computational cost is defined by the underling coarse grain method in MLMC. The main difficulty of implementing MLMC is how to efficiently parallelize it. Compared to the classical MC method, the problem of parallelizaion for MLMC is much harder. The method has many different levels

that can be parallelized in different ways. Considering some or all of the parallel levels leads to a different parallel scheme. The schemes that exploits all of the levels are most flexible. The scheduling algorithm has also to consider the sample to sample time deviations, that comes from the underling problem calculation. The proposed dynamic load balancer provides a way to exploit the different levels of MC as levels of parallelism that leads to very flexible schemes. It can be combined with different MC scheduler methods - schedulers that exploits only two level parallelism. Both *LvlSmlSyn* and *LvlSynHom* can be seen as edge cases of the more general dynamic load balancer. Incorporating the interruption mechanism with the scheduler leads to strategies close to the theoretical optimal time. However the choice of the type of interpretation technique is problem depended. For the case of Laplace equation, where the computation of a single sample is faster compared to the convection-reaction-diffusion, the global interruption gives better results compared to strategies that tries to balance the work locally. The opposite is true for the case of convection-reaction-diffusion local interruption or queue approaches are better. The results show that the achieved efficiency and scalability of the considered algorithms is high and the most impacting factor for the synchronization time is the number of processors used.

By the opinion of the author, the main contributions of this chapter are:

- An adaptive algorithm for resource allocation between the different levels of the Multilevel Monte Carlo algorithm has been developed;
- An overview, analysis and comparison of six parallelization strategies are made;
- A strategy for generating random fields on graphic accelerators has been developed and implemented;
- Four advanced parallel algorithms were proposed, implemented and compared;

- The applicability of the considered approaches for large scale simulations of realistic problems was confirmed with tests on a large number of cores.

Chapter 6

Final thoughts and future work

6.1 Final thoughts

The main aim of this work is to provide efficient implementation of Multilevel Monte Carlo algorithm to address problems involving porous medium flow simulations. More concretely a MLMC algorithm, that is capable of much faster simulations, compared to the classical Monte Carlo algorithm. The developed parallel algorithm must use effective resource allocation and consider the sample to sample computational time fluctuations, in order to be applicable for large simulations.

In chapter **1** an overview of existing approaches for stochastic modeling and the computational challenges involving such simulations is done. In chapter **2** a detailed explanation of necessary components for porous medium flow simulations is done. Also, a brief analysis of random sampling approaches is done and as a result an efficient algorithm for such random data generation is chosen. Detailed analysis of the Multilevel Monte Carlo algorithm is

done and it is compared to the classical Monte Carlo. In chapter **3** a well-established model equation is considered as a model problem, that shows well the computational and modelling challenges, involving MLMC usage. For this problem an efficient coarse graining technique is considered, that leads to MLMC algorithm that is much faster than the classical MC algorithm. In chapter **4** a solution of stochastic convection-reaction-diffusion problem is presented. The equation is used as a building block in many complex simulation models and it is more computationally expensive than the Laplace equation, considered in the previous chapter. Different approaches for coarsening are considered for this equation and the results are compared to the classical MC algorithm. Chapter **5** considers the computational challenges that MLMC introduce. A different approaches for parallelizaion are developed and tested. Different computational algorithms, that use all of the MLMC parallel layers, are formulated. For described algorithms computational tests with different number of cores and different MLMC settings are performed and in depth analysis of all collected results is present.

The solution described in this work achieves all of formulated aims. The proposed scheduling algorithms use all three defined layers of parallelism. The dynamic load balancer can be combined with different two layer scheduling algorithms, that are more suitable for the simulation in question. A scheduling approach, that incorporate the sample to sample solution time fluctuations are proposed. Combined with the proposed renormalizaion technique, fitted for MLMC, leads to much more superior algorithm, compared to the classical MC. It is capable of much faster simulations for porous medium flow problems and it is suitable for realistic simulations with very effective parallel scheduling, shown by the performed tests. An experiment with large number of cores over a computationally easy problem, for that number of cores, is performed to test the limits of the algorithm.

6.2 Author contributions

By the opinion of the author, the main contributions of this work are:

- Scientific contributions:
  - A review and analysis of the existing solutions to the considered problems are made. The advantages and disadvantages of the existing solutions for generating stochastic fields and corresponding sampling algorithms are evaluated (Chapter 2);
  - Different approaches for approximation of the stochastic field for the Laplace problem are analyzed and compared (Chapter 3);
  - An effective method for renormalization of the stochastic field for the purposes of the Multilevel Monte Carlo has been developed (Chapter 3);
  - An adaptive algorithm for resource allocation between the different levels of the Multilevel Monte Carlo algorithm has been developed (Chapter 5);
  - The Multilevel Monte Carlo method is applied successfully for the first time to solve the convection-reaction-diffusion equation (Chapter 4).
- Scientific and applied contributions
  - An approach for determining the levels for the Multilevel Monte Carlo for the two considered problems is defined (Chapter 3 and Chapter 4);
  - Analysis and comparison of the two considered approaches for coarse grain, for the Multilevel Monte Carlo versus the classi-

cal Monte Carlo for the convection-reaction-diffusion problem are performed (Chapter 4);

  - Analysis and comparison between the rate of convergence and the time for calculation of the Multilevel Monte Carlo method with simplified renormalization and the classical Monte Carlo are made (Chapter 3);

  - An overview, analysis and comparison of six parallelization strategies are made (Chapter 5).

- Applied contributions

  - A strategy for generating random fields on graphic accelerators has been developed and implemented (Chapter 5);

  - Four advanced parallel algorithms were proposed, implemented and compared (Chapter 5);

  - The applicability of the considered approaches for large scale simulations of realistic problems was confirmed with tests on a large number of cores (Chapter 5).

6.3 Author publications

- **[1]** Iliev, O., Mohring, J., **Shegunov, N.**, *Renormalization Based MLMC Method for Scalar Elliptic SPDE*, International Conference on Large-Scale Scientific Computing, pp.295-303, 2017, Springer, ISSN: 0302-9743, SJR (2017) - 0.295

- **[2] Shegunov, N.**, Armianov, P., Semerdjiev, A., Iliev, O., *GPU accelerated Monte Carlo sampling for SPDEs*, 2019, Conf. Proc. of the 12th ISGT 2018, ISSN:1613-0073, SJR (2019) - 0.177

- **[3]** Zakharov, P., Iliev, O., Mohring, J., **Shegunov, N.**, *Parallel Multilevel Monte Carlo Algorithms for Elliptic PDEs with Random Coefficients*, International Conference on Large-Scale Scientific Computing, pp.463-472, 2019, Springer, ISSN: 0302-9743, SJR (2019) - 0.427

- **[4]** Bastian, P., Altenbernd, M., Dreier, N., Engwer, Ch., Fahlke, J., Fritze, R., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Mohring, J., Müthing, S., Ohlberger, M., Ribbrock, D., **Shegunov, N.**, Turek, S., *Exa-Dune: Flexible PDE Solvers, Numerical Methods and Applications, Software for Exascale Computing-SPPEXA, 2016-2019*, pp. 225-269, 2020, https://doi.org/10.1007/978-3-030-47956-5_9, Springer

- **[5] Shegunov, N.**, Iliev, O., *On Dynamic Parallelization of Multilevel Monte Carlo Algorithm, Cybernetics and Information Technologies*, Volume 20, No 6, pp. 116-125, 2020, Journal Sciendo Print ISSN: 1311-9702, Online ISSN: 1314-4081, SJR (2020) - 0.310

6.4 Future work

Although the proposed algorithms are very effective, there are many other aspects of the problem, that are not considered and are subject to future work. One such aspect is the approximation of the permeability field on the coarser levels. Although the renormalizaion approach represent the variance accurately, it is computationally intensive technique. There may be more suitable approximation techniques, that are faster but less accurate, compared to the renormalizaion. For the convection-reaction-diffusion equation different constants for the Kozeny-Carman equation can be considered, specific to different soil composites. Furthermore, even different relations between permeability and porosity can be considered. An important setting, where the algorithm can be adopted for use, is for the dynamic version of the convection-reaction-diffusion model - important for the chemical industry. From the scheduling prespective, the processor distribution across the levels

of the MLMC algorithm is determined by an algorithm. Improvement of this algorithm, that selects the work-processor distribution automatically within a level can be considered - a level synchronous heterogeneous approach. By constructing and solving the optimization problem within a level, the synchronization time can be further reduced, leading to even more parallel efficient algorithm. From simulation perspective, a test that uses finer grids can be performed - e.g. simulation with high number of unknowns, for example 10^8. For such a simulation the number of samples must be examined. Another perspective of future work is the problem of finding optimal number of levels for the MLMC algorithm, or adaptive MLMC based on the level variance can be researched.

Appendices

Appendix A

Abbreviations

AMG	-	Algebraic Multi-Grid
AVG	-	Average
BCCB	-	Block Circulant with Circulant Blocks
BTTB	-	Block Toeplitz with Toeplitz Blocks
BiCG	-	Bi Conjugate Gradient
ccNUMA	-	cache coherence Non-Uniform Memory Access
CG	-	Conjugate Gradient
FEM	-	Finite Element Method
FFT	-	Fast Fourier Transform
FV	-	Finite Volume
HPC	-	High Performance Computing
IFFT	-	Inverse Fast Fourier Transform
MC	-	Monte Carlo
MLMC	-	Multilevel Monte Carlo
MPI	-	Message passing interface
MsFEM	-	Multi-scale Finite Element Method
NUMA	-	Non-Uniform Memory Access
PDE	-	Partial Different Equation
RMS, RMSE	-	Root Mean Square Error
RS	-	Renormalize then Solve approach
SBT	-	Symmetric Block Toeplitz
SIMD	-	Single instruction, multiple data
SMP	-	Symmetric Multi-Processing

SPDE	-	Stochastic Partial Different Equation
SR	-	Solve then Renormalize approach
UMA	-	Uniform Memory Access
UQ	-	Uncertainty quantification

Appendix B

Hardware resources

B.1 Beehive cluster

Total number of nodes:	48
Sockets per node:	2
Cores per socket:	14
Memory per node:	185GB
Node interconnect:	InfiniBand
CPU model:	Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
Total number of cores:	1344
Host:	Fraunhofer ITWM

B.2 SuperMUC cluster

Total number of nodes:	6336
Sockets per node:	2
Cores per socket:	24
Memory per node:	96GB
Node interconnect:	InfiniBand
CPU model:	Intel Skylake Xeon Platinum 8174
Total number of cores:	304,128
Host:	TUM Munich

Appendix C

List of figures and tables

List of Figures

List of Tables

Bibliography

[1] M.B. Giles. "Multilevel Monte Carlo Methods". In: *Acta Numerica* (2018). DOI: 10.1017/S09624929.

[2] S. Heinrich. "Monte Carlo complexity of global solution of integral equations." In: *Journal of Complexity* 14 (1998), pp. 151–175. DOI: 10.1006/jcom.1998.0471.

[3] S. Heinrich and E. Sindambiwe. "Monte Carlo complexity of parametric integration." In: *Journal of Complexity* 15 (1999), pp. 317–341. DOI: 10.1006/jcom.1999.0508.

[4] S. Heinrich. "The multilevel method of dependent tests." In: *Advances in Stochastic Simulation Methods* (2000), pp. 47–61. DOI: 10.1007/978-1-4612-1318-5_4.

[5] D. Drziga et al. "SCHEDULING MASSIVELY PARALLEL MULTIGRID FOR MULTILEVEL MONTE CARLO METHODS". In: *SIAM J. SCI. COMPUT* 39.5 (2017), S873–S897. DOI: 10.1137/16M1083591.

[6] Dongbin Xiu. "Fast Numerical Methods for Stochastic Computations: A Review". In: *Communications in computational physics* (2009).

[7] Wolfgang Betz, Iason Papaioannou, and Daniel Straub. "Numerical methods for the discretization of random fields by means of the Karhunen-Loève expansion". In: *Computer Methods in Applied Mechanics and Engineering* 271 (2014), pp. 109–129. DOI: doi:10.1016/j.cma.2013.12.010.

[8] I. G. Graham et al. "Analysis of circulant embedding methods for sampling stationary random fields". In: *SIAM Journal on Numerical Analysis* 56 (2018). DOI: https://doi.org/10.1137/17M1149730.

[9] Francisco Cuevas, Emilio Porcu, and Denis Allard. *Fast and exact simulation of isotropic Gaussian random fields on* $\mathbb{S}^2$ *and* $\mathbb{S}^2 \times \mathbb{R}$. 2018. eprint: arXiv:1807.04145.

[10] Sarah Osborn, Panayot Vassilevski, and Umberto Villa. "A MULTILEVEL HIERARCHICAL SAMPLING TECHNIQUE FOR SPATIALLY CORRELATED RANDOM FIELDS". In: *SIAM J. SCI. COMPUT* (2017). DOI: 10.1137/16M1082688.

[11] W.K. Liu, T. Belytschko, and A. Mani. "Probabilistic finite elements for nonlinear structural dynamics". In: *Comput. Methods Appl. Mech. Engrg* 56 (1986), pp. 61–81. DOI: 10.1016/0045-7825(86)90136-2.

[12] W.K. Liu, T. Belytschko, and A. Mani. "Random field finite elements." In: *Int. J. Num. Meth.Engng.* 23 (1986), pp. 1831–1845. DOI: 0.1002/nme.1620231004.

[13] R.G. Ghanem and P. Spanos. *Stochastic Finite Elements: a Spectral Approach*. Springer-Verlag, 1991. ISBN: 978-1-4612-7795-8.

[14] J. Dick, F. Kuo, and I. Sloan. "High-dimensional integration: The quasi-Monte Carlo way." In: *Acta Numerica* (2013). DOI: 10.1017/s0962492913000044.

[15] D. Zhang. *Stochastic Methods for Flow in Porous Media: Coping With Uncertainties*. Academic Press, 2002. ISBN: 0127796215.

[16] W.L. Loh. "Latin hypercube sampling." In: *The Annals of Statistics.* 24 (1996), pp. 2058–2080. DOI: 10.1214/aos/1069362310.

[17] B.L. Fox. *Strategies for Quasi-Monte Carlo*. International Series in Operations Research and Management Science. Kluwer Academic Pub., 1999. ISBN: 978-1-4615-5221-5. DOI: 10.1007/978-1-4615-5221-5.

[18] H. Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, 1992. ISBN: 978-0-89871-295-7. DOI: 10.1137/1.9781611970081.

[19] Kurt Binder. "Monte Carlo Methods: a powerful tool of statistical physics". In: *Monte Carlo and Quasi-Monte Carlo Methods* (1998).

[20] Anthony Williams. *C++ Concurrency in Action*. 2019. ISBN: 1617294691.

[21] Blaise Barney. *Introduction to Parallel Computing*. Lawrence Livermore National Laboratory.

[22] Ian Foster. *Designing and building parallel programs*. Addison Wesley, 1995. ISBN: 978-0-201-57594-1.

[23] Bertil Schmidt et al. *Parallel Programming*. Elsevier, 2017. ISBN: 978-0-128-49890-3.

[24] Walker DW. *Standards for message-passing in a distributed memory environment*. 1992. URL: https://www.osti.gov/biblio/7104668 (visited on 05/01/2021).

[25] *FAQ: Tuning the run-time characteristics of MPI sm communications*. URL: https://www.open-mpi.org/faq/?category=sm.

[26] *The MPI-3 standard introduces another approach to hybrid programming that uses the new MPI Shared Memory (SHM) model*. URL: https://software.intel.com/en-us/articles/an-introduction-to-mpi-3-shared-memory-programming?language=en.

[27] *Shared memory and MPI3.0*. URL: https://insidehpc.com/2016/01/shared-memory-mpi-3-0/.

[28] *Using MPI-3 Shared Memory As a Multicore Programming System*. URL: https://www.caam.rice.edu/~mk51/presentations/SIAMPP2016_4.pdf.

[29] Bungartz Hans-Joachim, Neumann Philipp, and Nagel Wolfgang. "Software for Exascale Computing - SPPEXA 2013-2015". In: (2016). DOI: 10.1007/978-3-319-40528-5.

[30] Bastian P et al. "A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part I: Abstract Framework." In: *Computing* 103–119 (2008). DOI: 10.1007/s00607-008-0003-x.

[31] *Dune Numerics*. URL: https://dune-project.org (visited on 05/01/2021).

[32] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002.

[33] E.Powell. *Numerical Methods for Generating Realisations of Gaussian Random Fields*. URL: www.maths.manchester.ac.uk/~cp (visited on 05/01/2021).

[34] Gabriel Lord, E.Powell, and Tony Shardow. *An introduction to Computational Stochastic PDEs*. Cambridge University Press, 2014. ISBN: 978-0521728522. DOI: 10.1017/CBO9781139017329.

[35] Robert Gould and Colleen Ryan. *Introductory Statistics*. Pearson, 2016. ISBN: 978-0321978271.

[36] Joseph F. Hair. *Multivariate Data Analysis – A Global Perspective, 7th Edition*. Pearson Education, 2010. ISBN: 9780135153093.

[37] Randall LeVeque. *Numerical Methods for Conservation Laws*. Birkhauser-Verlag, 1990. ISBN: 978-3-0348-8629-1.

[38] Randall LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511791253.

[39] Wendt and John (Ed.) *Computational Fluid Dynamics*. Springer Verlag, 2009. ISBN: 978-3-540-85056-4. DOI: 10.1007/978-3-540-85056-4.

[40] Germund Dahlquist and Åke Björck. *Numerical Methods*. Courier Corporation, 2003. ISBN: 978-0486428079.

[41] Jack Dvorkin. *Kozeny-Carman equation revisited*. 2009. URL: https://pangea.stanford.edu/~jack/KC_2009_JD.pdf (visited on 05/01/2021).

[42] K.A. Cliffe et al. "Multilevel Monte Carlo Methods and Applications to Elliptic PDEs with Random Coefficients." In: *Computing and Visualization in Science* 14 (2010). DOI: https://doi.org/10.1007/s00791-011-0160-x.

[43] Hoeksema R.J. and Kitanidis P.K. "Analysis of the spatial structure of properties of selected aquifers." In: *Water Resour. Res* 21 (1985), pp. 563–572. DOI: 10.1029/WR021i004p00563.

[44] G. Graham et al. "Quasi-Monte Carlo methods for elliptic PDEs with random coefficients and applications." In: *Journal of Computational*

Physics 230 (2011), pp. 3668–3694. DOI: https://doi.org/10.1016/j.jcp.2011.01.023.

[45] A. Cliffe et al. "Parallel computation of flow in heterogeneous media using mixed finite elements". In: *J. Comput. Phys.* 164 (2000). DOI: https://doi.org/10.1006/jcph.2000.6593.

[46] Blaheta R., Béreš M., and Domesová S. "A study of stochastic FEM method for porous media flow problem". In: *Applied Mathematics in Engineering and Reliability* (2016). DOI: 10.1201/b21348-47.

[47] Mohring J. et al. "Uncertainty Quantification for Porous Media Flow Using Multilevel Monte Carlo". In: *International Conference on Large-Scale Scientific Computing* 9374 (2015), pp. 145–152. DOI: 10.1007/978-3-319-26520-9_15.

[48] Efendiev Y., Iliev O., and Kronsbein C. "Multilevel Monte Carlo methods using ensemble level mixed MsFEM for two-phase flow and transport simulations." In: *Comput. Geosci* 17 (2013), pp. 833–850. DOI: 10.1007/s10596-013-9358-y.

[49] Renard P. and De Marsily G. "Calculating equivalent permeability: a review." In: *Adv. Water Resour.* 20 (1997), pp. 253–278. DOI: 10.1016/S0309-1708(96)00050-4.

[50] Wen X.H. and Gomez-Hern ández J.J. "Upscaling hydraulic conductivities in heterogeneous media: an overview." In: *Journal of Hydrology* 183 (1996), pp. ix–xxxii. DOI: 10.1016/S0022-1694(96)80030-8.

[51] Ivan Lunati et al. "A numerical comparison between two upscaling techniques: non-local inverse based scaling and simplified renormalization." In: *Advances in Watter Resources* 24 (2001), pp. 913–929. DOI: 10.1016/S0309-1708(01)00008-2.

[52] Oleg Iliev et al. "On the Pore-Scale Modeling and Simulation of Reactive Transport in 3D Geometries". In: *Mathematical Modelling and Analysis* 22.5 (2017), pp. 671–694. DOI: 10.3846/13926292.2017.1356759. eprint: https://doi.org/10.3846/13926292.2017.1356759. URL: https://doi.org/10.3846/13926292.2017.1356759.

[53] Turkuler Ozgumus, Moghtada Mobedi, and Unver Ozkol. "Determination of Kozeny Constant Based on Porosity and Pore to Throat Size Ratio in Porous Medium with Rectangular Rods". In: *Engineering Applications of Computational Fluid Mechanics* 8.2 (2014), pp. 308–318. DOI: 10.1080/19942060.2014.11015516. eprint: https://doi.org/10.1080/19942060.2014.11015516. URL: https://doi.org/10.1080/19942060.2014.11015516.

[54] M.N. Panda an L.W. Lake. "Estimation of single-phase permeability from parameters of particle-size distribution". In: *American Association of Petroleum Geologists Bulletin* (1994).

[55] Abdon Atangana. *Fractional Operators with Constant and Variable Order with Application to Geo-Hydrology*. Elsevier, 2018. ISBN: 978-0-12-809670-3. DOI: https://doi.org/10.1016/C2015-0-05711-2.

[56] Oleg Iliev, Jan Mohring, and Nikolay Shegunov. "Renormalization Based MLMC Method for Scalar Elliptic SPDE". In: *Large-Scale Scientific Computing. LSSC 2017. Lecture Notes in Computer Science* 10665 (2018), pp. 295–303. DOI: 10.1007/978-3-319-73441-5_31.

[57] Petr Zakharov et al. "Parallel Multilevel Monte Carlo Algorithms for Elliptic PDEs with Random Coefficients". In: *International Conference on Large-Scale Scientific Computing* (2019), pp. 463–472. DOI: 10.1007/978-3-030-41032-2_53.

[58] James Cooley, Peter Lewis, and Peter Welch. "The Fast Fourier Transfrom and Its Applications". In: *Transactions on Education* (1969).

[59] James Cooley, Peter Lewis, and Peter Welch. "Application of the Fast Fourier Transfrom to Computation of Fourier integrals, Fourier Series, and Convolution Integrals". In: *Transactions on audio and electroacoustics* (1967).

[60] James Cooley and Jonh Tukey. "An Algorithm for the Machine of Complex Fourier Series". In: *Mathematics of Computation* (1965).

[61] Matteo Frigo and Steven Johnson. *FFTW is a C subroutine library*. URL: http://www.fftw.org/ (visited on 05/01/2021).

[62] Nikolay Shegunov et al. “GPU accelerated Monte Carlo sampling for SPDEs”. In: *Proceedings of the Information Systems and Grid Technologies* 2464 (2018). ISSN: 1613-0073.

Declaration of Originality

I declare that the present dissertation contains original results obtained from my research (with the support of and the assistance of my supervisor and all my co-authors). The results obtained, described and published by other scientists, are duly and in detail cited in the bibliography. This work has not been applied for the acquisition of a scientific degree in another higher school, university or scientific institute.

Signiture: