Porous Medium Flow Simulations using Massively Parallel MLMC algorithm

Nikolay Shegunov

Summary

SOFIA UNIVERSITY
ST. KLIMENT OHRIDSKI

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.

2021

0. Contents

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 praticular 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 has own pros and cons. A well-established algorithmic scheme 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 time compared to the classical MC method. Its first use was for parametric integration for the expected value of $E[f(x, \lambda)]$ 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. The 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 then 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.

1.2 Stochastic computations

Many real World problems are subject to uncertainty due to some kind of limitations 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**, or model inadequacy - which comes from the lack of knowledge of the underlying physics describing the problem. An essential matter is to parameterize the input uncertainty by a set of a finite number of random variables. Extensive efforts have been devoted in order to model the uncertainty accurately.

The literature presents several ways of making the aforementioned. The popular choice for parameterization is Karhunen-Loeve expansion [6]. Other possible means of representation is by employing forward and inverse fast Fourier transforms over a circulant matrix [7], or to use Cholesky decomposition [8]. In the models considered in this work a circulant embedding approach has been chosen. It offers fast effective way for sample generation.

1.3 Modern high-performance computing systems

Modern high-performance computing clusters, utilize a hybrid hardware memory model (see figure 1.1). Each node consists of a SMP unit. Distributed memory parallelization is done through the node interconnection. To achieve very high throughput and very low latency for the node to node communication an *InfiniBand* networks are used.

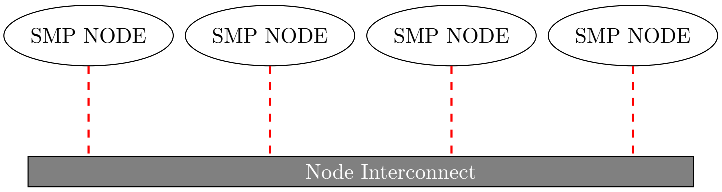


Figure 1.1: Hybrid Architecture

For this hardware model, the main resources, that programs have to utilize, are the processors and memory. This means a parallel program can distribute work across different processors and distribute data among different nodes and also between the host memory (main memory) and device memory (graphical memory). The program can also communicate locally within a single symmetric multi-processing (SMP) node or globally between different SMP nodes. The established programming framework for such systems is called *Message Passing Interface* (MPI). The MPI is a standardized actor-like model, that uses a message-passing protocol to organize communication between nodes

[9]. It is designed to function on a wide variety of parallel computing architectures. The standard defines the syntax and semantics of library routines. 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 processes are grouped in communicators. Each process can communicate with each process within a communicator. Processes from different communicators can not communicate. Along MPI library implementations, for simulations of different industrial processes, an additional libraries are necessary. The most important are the libraries that implements efficient parallel linear algebra solvers. There exist a large number of such specialized libraries, as it is an important part in many models, not only limited to HPC, such as: *LAPACK, Eigen, Armadillo, SuperLU, etc.*. However, few of them are suitable for massively parallel problems. In the past few years a rapid development have been done towards such software. Most notably ones are *Deal.ii*[1] and *DUNE*[2] libraries [10, 11] , *OpenFoam* (specialized in Computational Fluid dynamics), Portable, Extensible Toolkit for Scientific Computation (PETSc)[3].

For the models considered, the DUNE library is used, as it offers an easy way of different PDE discrete scheme implementations and efficient parallel linear solvers. The 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 **C++0x** standard family. It is also capable of supporting shared memory, as well as distributed memory computations. The library consists of core modules and modules build on top of them [12].

1.4 Aims and scope of this work

In this work the Multilevel Monte Carlo algorithm to problems involving porous medium flows is considered. The aim is to provide an efficient parallel version of the algorithm, capable of much faster simulations compared to the classical Monte Carlo approach. As such, the algorithm has to be applicable for realistic simulation with the help of High-Performance Computing (HPC) systems. To achieve that goal the following tasks are considered:

- Explore the existing approaches to the problem.

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

- 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.

2. Mathematical models

2.1 Random sampling algorithms

An essential problem in uncertainty quantification applications is how to generate a coordinated random samples by a given covariance matrix from a multivariate random distribution. A multivariate random distribution is defined completely by the covariance matrix. Most sampling based algorithms, use this matrix as a basis. 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 for one of the variables causes a change in the value of the other.

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 $\Sigma_{i,j} = cov(X_i, X_j)$, and the correlation matrix is defined as the matrix with entries $cov(X_i, X_j)/\sigma_{x_i}\sigma_{x_j}$. 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. An easy way to do this is by Cholesky decomposition [13, 14, 15, 16]. 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. Under certain assumptions about the random process and sampling at equidistant points an efficient algorithm can be used over a reduced covariance matrix. It is both faster and memory efficient then the Cholesky decomposition approach. The idea is to use Fourier transforms to diagonalize the covariance matrix [13, 14].

2.2 Finite volume numerical method

Finite volume method is one of the most versatile methods, frequently used in computational fluid dynamics. It was developed with the aim to discretize the equations that describes conservation laws coming from physics. The simplest example of one dimensional conservation law is the PDE **2.1** [17]:

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

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

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

each component of q measures the density of some quantity and the above 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 and x_2. If the total mass is not conserved then the equation **2.2** has to contain source terms. There are many such conservative systems, unfortunately many of them are non-linear and they contain contains discontinuities. This discontinuities lead to computational difficulties. The solution can even become physically incorrect because of that [17, 18]. To overcome this, instead of approximating the derivatives directly, like most of the other methods, finite volume 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 between the cells is balanced.

2.3 Multilevel Monte Carlo algorithm

In its simple form Monte Carlo algorithm is quite intuitive. Suppose the expected value of a given quantity needs to be computed. To achieve this, multiple samples Q^i are generated, and the empirical expected value $E[Q]$ is computed by:

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

where M is characteristic parameter of the underling problem and N denotes the number of samples. The Root Mean Square error (**RMS**) is $\mathcal{O}\left(\frac{V[Q]}{\sqrt{N}}\right)$. Here $V[Q]$ is the variance. This directly means that to achieve accuracy of ϵ, requires $N = \mathcal{O}(\epsilon^{-2})$ number of samples to be computed. Here rests the main weakness of the methods - **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], [19]**. 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 $Y(\omega)$ is a standard Monte Carlo estimator and ω is a random vector. This idea is called Multilevel Monte Carlo and it is particularly useful in the field of fluid dynamics.

Assume that $E[Q_M]$ can be made arbitrary close to $E[Q]$ by choosing M sufficiently large. The goal is to approximate $E[Q]$ by $E[Q_M]$. This can be achieved by computing an estimator $\widehat{Q}_M$, and quantifying its accuracy using the root mean square error.

$$e(\widehat{Q}_M) = (E[(\widehat{Q}_M - E[Q])^2])^{1/2} \tag{2.3}$$

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

$$\widehat{Q}_{M,N}^{MC} = \frac{1}{N}\sum_{i=1}^{N} Q_M^i \tag{2.4}$$

where Q_M^i, $i = 1, \dots N$ are independent samples of the unknown quantity Q_M. Setting the cost of computing one sample is $C(Q_M^i) = \mathcal{O}(M^\gamma)$, where γ is positive constant, and expanding the mean square error will yield:

$$\begin{aligned} e(\widehat{Q}^{MC}_{M,N})^2 &= E[(\widehat{Q}^{MC}_{M,N} - E[\widehat{Q}^{MC}_{M,N}] + E[\widehat{Q}^{MC}_{M,N}] - E[Q])^2] \\ &= E[(\widehat{Q}^{MC}_{M,N} - E[\widehat{Q}^{MC}_{M,N}])^2] + (E[\widehat{Q}^{MC}_{M,N}] - E[Q])^2 \\ &= V[\widehat{Q}^{MC}_{M,N}] + (E[\widehat{Q}^{MC}_{M,N}] - E[Q])^2 \end{aligned} \tag{2.5}$$

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:

$$e(\widehat{Q}^{MC}_{M,N})^2 = N^{-1}V[Q_M] + (E[Q_M] - E[Q])^2 \tag{2.6}$$

In applications involving a 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 **2.6** represents the error of the numerical method used for discretization and has a bias effect on the estimator.

The extension of the MC to MLMC is quite natural. Let $\{M_l : l = 0 \dots L\} \in N$ 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 $M_l = sM_{l-1}$, for $l = 0 \dots L$. Defining $Y_l = Q_{M_l} - Q_{M_{l-1}}$ and setting $Y_0 = Q_{M_0}$, following expansion for $E[Q_M]$ can be formulated

$$E[Q_M] = E[Q_{M_0}] + \sum_{l=1}^{L} E[Q_{M_l} - Q_{M_{l-1}}] = \sum_{l=0}^{L} E[Y_l] \tag{2.7}$$

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 **2.7** are approximated using standard MC independent estimators, each with N_l samples:

$$\widehat{Y}_l = N_l^{-1} \sum_{i=1}^{N_l} (Q^{(i)}_{M_l} - Q^{(i)}_{M_{l-1}}) \tag{2.8}$$

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 [1]. By defining C_0, V_0 to be the cost and variance of one sample of Y_0 and C_l, V_l be the cost and variance of one sample for estimator Y_l. Then cost and variance of the method becomes:

$$C^{total} = \sum_{l=0}^{L} C_l N_l \tag{2.9}$$

$$V^{total} = \sum_{l=0}^{L} N_l^{-1} * V_l \tag{2.10}$$

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

$$C^{total} + \lambda^2 V^{total} \tag{2.11}$$

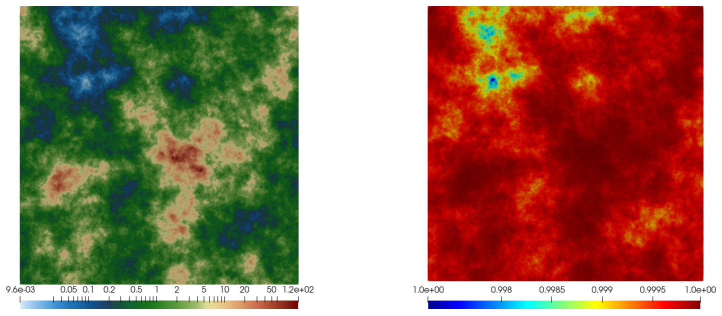
Then the total computational cost takes the form:

$$C^{total} = \frac{1}{\epsilon^2} \left(\sum_{l=0}^{L} \sqrt{V_l / C_l} \right)^2 \tag{2.12}$$

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].

2.4 Porosity and Permeability

Two important properties, that characterizes a porous medium, are *porosity* and *permeability*. Those two properties presented 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 pours. 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. When the flow within the media is laminar the permeability and porosity can be connected as Kozeny–Carman equations **[20]**. In practice the permeability field is modeled as a log-normal distribution with a given covariance function and two parameters: σ - the standard deviation and λ - constant, called correlation length.



(a) Permeability (b) Porosity

Figure 2.1: Single realization of permeability with $\sigma = 2, \lambda = 0.2$ and the corresponding porosity field computed with Kozeny-Carman formula.

3. Multilevel Monte Carlo for porous medium flows

Stochastic partial differential equations, as mentioned, have attracted great attention due to their importance in modeling a variety industrial processes. Thanks to the increased computational power, such solutions can be facilitated. For example, imagine a simulation of a subsurface water flow in an area of hundreds of square km.

In this work two models are considered, however the approach proposed here is not limited to this problems, and it can be applied to other areas. The first is the stochastic Laplace equation. This equation itself is a well-established model in the area and demonstrates the computational challenges in uncertainty quantification for pours medium problems. The second equation is more practical and it is important in many areas of industry. This equation is used as a model in a chemical reaction, filtration processes as well as building block in many other applications from physics, biology, and chemistry.

3.1 Stochastic Laplace equation

Consider steady state single phase flow in random porous media in a unit cube, with domain $D = (0,1)^2$ and properly defined random space Ω, 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. 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:

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

3.1.1 Coarse grain for Multilevel Monte Carlo

The loosely defined definition of what a level means in the interpretation of equation **2.7**, 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. 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 **2.8**, 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 $2^M \times 2^M$ number of cells, then on coarse level $2^{M-1} \times 2^{M-1}$ number of cells is used. This selection of levels directly leads to a need for permeability approximation across the levels. An intuitive way is to represent the permeability on the coarser levels to be arithmetic averaging over 4 cells in the case of two dimensional problem and 8 in three dimensional case. However this does not lead to good variance preservation. This is crucial for the efficiency of the Multilevel Monte Carlo. Good variance preservation means small variance on the fine estimators and large variance on the coarse levels. For this reason a technique called simplified renormalizaion is considered. This technique has been widely used in the past (and is still intensively used by many groups) for upscaling hydraulic conductivity in heterogeneous media. Details can be found in **[21]**, **[22]**, **[23]** and the references therein. In a nutshell, the simplified renormalization procedure is based on recursive harmonic arithmetic and geometric averaging. If two cells are in parallel to the flux, the harmonic average of those two cells is taken, if they are not an arithmetic average is used. Depending how the procedure starts, two different values are obtained. The approximation is done by taking the geometric average of those two values.

3.1.2 Simulation results

For testing how MLMC performs the Laplace equation, consider table **3.1**. The table contains the results of testing a two and three-level MLMC method with simplified renormalization. As an averaging technique, simplified renormalization has a smoothing effect. Renormalized approximation field are 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

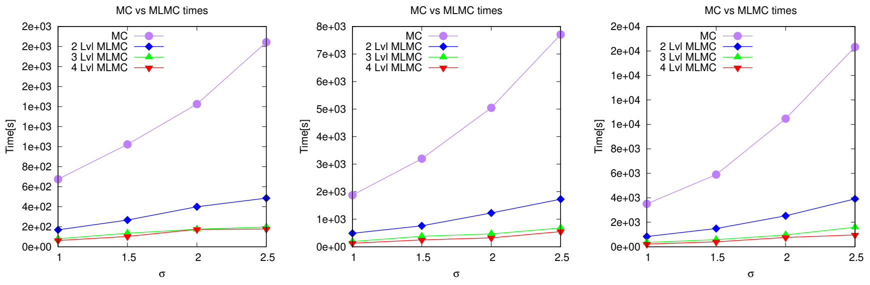
The fourth column shows that while MC needs tens of thousands of sample realizations on the finest grid. For MLMC algorithm almost the same number of realizations are needed on a 16 times coarser grid while only few realizations are needed on the finest grid. The renormalization technique leads to very 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	N_l
MC	-	$V[Y_0]$:	1.10483	$2^{10} \times 2^{10}$	122761
2L MLMC	0.00115	$V[Y_0]$:	1.17	$2^{9} \times 2^{9}$	132229
		$V[Y_1]$:	3.36e−5	$2^{10} \times 2^{10}$	324
3L MLMC	0.00590	$V[Y_0]$:	1.26	$2^{8} \times 2^{8}$	128315
		$V[Y_1]$:	8.89e−6	$2^{9} \times 2^{9}$	205
		$V[Y_2]$:	9.82e−6	$2^{10} \times 2^{10}$	107

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

To further illustrate the superiority of the MLMC algorithm, compared to a standard Monte Carlo sampling, the total computational time for achieving tolerance of $\epsilon = 3e{-}3$ with MC and MLMC approaches for different stochastic generating parameters is considered. Figure **3.1** shows the results.

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.



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

Figure 3.1: Speedup of MLMC with respect to MC on 196 cores

3.2 Stochastic Convection-Reaction-Diffusion equation

Consider a domain $\bar{A} = (0, \bar{L})^2$, and in that domain steady dimensional convection-diffusion-reaction equation describing reactive transport in random porous media. 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, $\bar{C}_{in}$ to be the characteristic concentration (this is the prescribed concentration at the inlet), $D = \bar{D}$ to be the characteristic value of the diffusion, Pe to be the Peclet number and Da to be Damkohler number, and let $\bar{\kappa}$ be characteristic reaction rate, the dimensionless form of the convection-reaction-diffusion equation states as follows:

$$-\nabla \cdot (\nabla C) + Pe(\omega)\nabla \cdot (v(x,\omega)C) + Da(\omega)C = 0, \ \ x \in (0,1)^2, \omega \in \Omega. \quad (3.4)$$

Subject to prescribed boundary conditions. The dimensionless *Peclet* and *Damkohler* numbers in equation **3.4** are defined as follows:

$$Pe(\omega) = \frac{\tilde{Pe}}{\phi(\omega)}, \quad Da(\omega) = \frac{\tilde{Da}}{\phi(\omega)} \quad (3.5)$$

Where $\tilde{Pe}, \tilde{Da}$ are predefined, based on intrisic diffusion at pore scale constants, and ϕ is the porosity. The considered quantities of interest, computed via Multilevel Monte Carlo algorithm are the concentration and the flow across the outflow boundary of the domain.

3.2.1 Coarse grain for Multilevel Monte Carlo

In the model given by equation **3.4** the three sources of uncertainty that need 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 level are defined as grid resolution. Two ways of coarse graning 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.

3.2.2 Simulation results

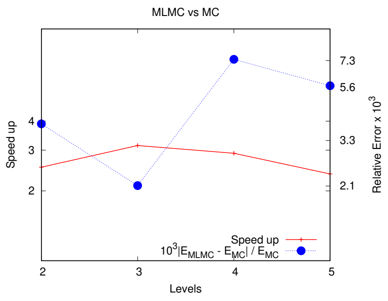
In table **3.2**, an investigation of the effectiveness of the MLMC with *Solve then Renormalize* coarse-grain approach, compared to the classical MC algorithm, is performed.

$\sigma : 2$			$\lambda : 0.2$		
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

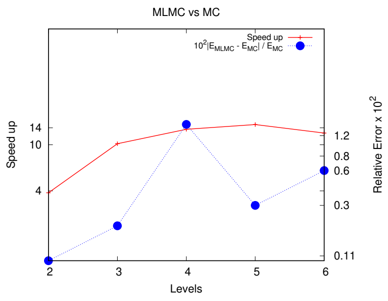
Table 3.2: Samples per level, quantity of interest - concentration

The quantity of interest computed is the concentration. For each problem on each level, the pressure linear system has to be solved 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 at a low number of levels. This is exactly the case. Figure **3.2** a), shows that most gain is achieved at a 3 level MLMC. On figure **3.2** b) an examination for the other type of coarsening is shown. 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 lesser work a processor must do to compute a single sample and, more importantly, the variance is preserved better across the different estimators.



(a) SR, Concentration



(b) RS, Flow

Figure 3.2: MLMC vs MC speedup

4. Parallel scheduling strategies for MLMC algorithm

4.1 MLMC algorithmic procedure

Both MC and MLMC, as a sampling base class of algorithms, rely on repeated 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 **4.1** ilustrates the idea. Each *estimation* phase is followed by a *solve* phase, and those *estimate-solve* blocks are repeated, until the stopping criteria is met.

$$\begin{tikzcd} \boxed{\text{Estimate}} & \bullet \arrow[r] & \bullet & \text{Solve} & \bullet \arrow[r] & \bullet & \boxed{\text{Estimate}} & \bullet \arrow[r] & \bullet & \text{Solve} & \cdots \end{tikzcd}$$

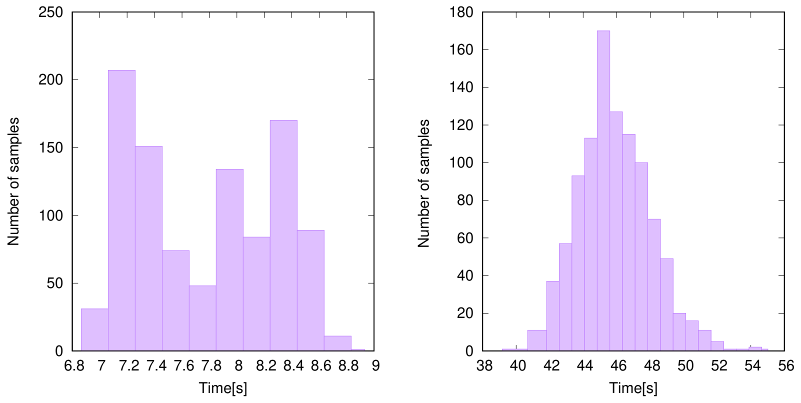
Figure 4.1: Block diagram of MLMC algorithm

At each estimation phase, the number of samples needed is estimated via equation 4.1. Where v_l is the empirical variance at level l, t_l is the expected time for computation of a single sample on level l and ϵ is the desired tolerance.

$$N_l = \lceil \lambda \sqrt{(v_l/t_l)} \rceil \ where \ \lambda = \frac{1}{\epsilon^2} \sum_{l=0}^{L} \sqrt{(v_l/t_l)} \tag{4.1}$$

After the end of estimation phase a computation phase can commence. For each sample on each level the required stochastic field is generated first. Then the generated field is fitted into the model to obtain a deterministic partial differential equation. Then the equation can be solved. For each solved equation the quantity of interest is computed and accumulated to the statistics. Upon completion of the *Solve* phase the equation 4.1 is checked again. This process is repeated until there are no more samples needed (the stopping criteria is met).

MLMC algorithmic approach can provide significant convergence speed-up and better computational cost compared to pure MC, however the simulations are still quite demanding. One simulation may require millions of samples to complete. Running MLMC on a single processor is not feasible and an efficient scheduling strategies for running on multiple CPUs, are a necessity. An efficient strategy have to consider not only the different layers, where parallel computation can be done, but also the sample to sample solution time deviations on a given level. They can be significant, especially for fields that have large variance. Figure **4.2** illustrates the problem.



(a) Eq. **3.1** with $\sigma = 2.0$ $\lambda = 0.2$

(b) Eq. **3.4** with $\sigma = 2.0$ $\lambda = 0.2$ $Pe = 2.5$ $Da = 0.5$

Figure 4.2: Histogram of the solution time for samples on gird $2^{10} \times 2^{10}$, over 1000 samples

For both problems the sample to sample time deviation is significant and can impact the parallel performance of the algorithm. In the case of a sample from convection-reaction-diffusion model, the time between the fastest and slowest sample computation time is more then 10 seconds. 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 most 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 (solve deterministic PDE).
- (ii) Parallelizing the solution of all samples on a given MLMC level.
- (iii) Parallelizing the calculation of all or several MLMC levels simultaneously.

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

4.2 Performance parameters and efficiency measurement

The design of the scheduling strategy requires different parameters to be taken into account. The most prominent is 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:

$$T^{min} = \frac{1}{P^{all}} \sum_{l=0}^{L} N_l E[t_l] \tag{4.2}$$

where P^{all} is the total number of the available CPU cores, N_l is the number of samples on level l and $E[t_l]$ is the expected time to compute one sample on level l. In the case of a single process per sample per level, the minimum time can be computed directly by recording the time to solve one 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:

$$C_l = \theta_l C^l_{min} \tag{4.3}$$

where in equation **4.3** C^l_{min} is the time to compute a single sample on level l on P^l_{min} processors. Rewriting equation **4.2** by substituting t_l with C_l, the equation becomes:

$$T^{min} = \frac{1}{P^{all}} \sum_{l=0}^{L} N_l E[C_l] = \frac{1}{P^{all}} \sum_{l=0}^{L} N_l E[\theta_l C^l_{min}] = \frac{1}{P^{all}} \sum_{l=0}^{L} N_l \theta_l E[C^l_{min}] \tag{4.4}$$

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 value for a given scalability window $\{P^{min}_l, \ldots, P^{Max}_l\}$, where $P^m ax_l$ is the maximum number of processors that achieve the desired threshold efficiency on a level. And $P^m in_l$ is the minimal number of processes that are able to solve the problem by fully available memory. By setting C^p_l to be the time to compute one sample on level l by p processors, θ_l becomes:

$$\theta_l = C^p_l / C^{min}_l \tag{4.5}$$

To define the parallel efficiency of a given level of $MLMC$, the relative inefficiency is computed by substituting the computational time C^{comp}_l with the minimal level time T^{min}_l and the resulting value is divided by T^{min}_l. 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:

$$Eff_l = 1 - (C^{comp}_l - T^{min}_l)/T^{min}_l \tag{4.6}$$

Expressing **4.6** in terms of θ and C^p_l becomes:

$$Eff_l(\theta, p) = 1 - \frac{(C^{comp}_l - N_l \theta_l E[C^l_{min}])}{(N_l \theta_l E[C^l_{min}])} \tag{4.7}$$

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

$$Eff(\theta) = 1 - \frac{(\sum_l C_l^{comp} - \sum_l N_l \theta_l E[C_{min}^l])}{\sum_l (N_l \theta_l E[C_{min}^l])} \tag{4.8}$$

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

4.3 Scheduling strategies

An easy, but, non optimal, way of scheduling the samples on the available processors is to treat the MLMC as a collection of pure MC estimators. All of the available processors are put to work on a single MC estimator and the samples is distributed evenly among them. Although this strategy is easy it leads to inefficient use of resources. However, combined with other scheduling strategies it can lead to very efficient parallel algorithms. The focus will be on dynamic approaches. They try to improve processor distribution by a greedy scheme.

Regardless of the design, each estimation step (see figure 4.1) needs information from all levels, as formula **4.1** suggests. The variance from each level is required, to estimate the number of samples that needs to be computed. This is the bottleneck of the algorithm.

4.3.1 Dynamic strategy

This scheme uses greedy approach and adopts during the simulation. It is very flexible as it can be combined with different strategies for pure MC estimation. The design uses all of the described parallel layers to schedule a sample computation.

A prerequisite for the strategy is the availability of solution times and variance statistics across the levels. This information can be obtained in two ways as: a precompute step or as a first compute step by computing first *Estimate-Solve* cycle with non optimal distribution of processors and collecting the real times. To simplify the algorithm, and without loss of generality consider three-level MLMC construction. Let $N_i, i = \{0, 1, 2\}$, 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 $\widehat{Y_i}$, $p_{l_i}^g$ 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 by:

$$T_{CPU}^{total} = N_0 t_0 + N_1 t_1 + N_2 t_2 \tag{4.9}$$

Then the optimal compute time per processor is:

$$T_{CPU}^{p} = \frac{T_{CPU}^{total}}{P^{total}} \tag{4.10}$$

By dividing the CPU time needed for a $\widehat{Y}_i$ with T_{CPU}^{p}, a continuous value for the number of processes on a given MC estimator is obtained:

$$p_i^{ideal} := \frac{N_i t_i}{T_{CPU}^{p}} \text{ for } i = \{0, 1, 2\} \tag{4.11}$$

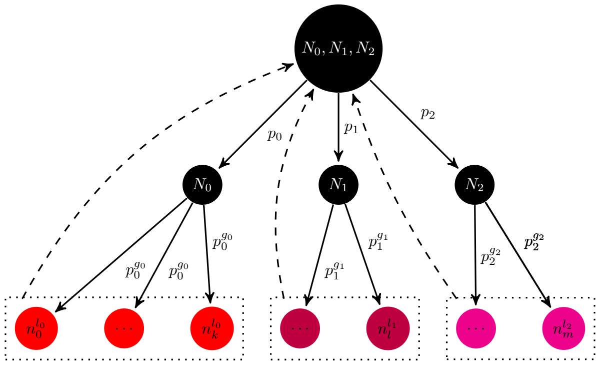


Figure 4.3: Schematic sample distribution of MLMC

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_i^{ideal} value, a processors distribution across the levels can be obtained:

$$p_i := \left\lfloor p_i^{ideal} \right\rfloor, \text{ for } i = \{0, 1, 2\} \tag{4.12}$$

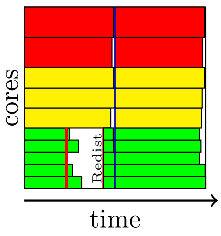
Depending on the scheme that will be used for parallel computing on the estimator $\widehat{Y}_i$, additional, restrictions may be imposed for p_i. Regardless of the estimator scheme, the unallocated processors due to rounding can be left unused for this cycle. To improve the estimation and search for a better approximation, the set of all upper and lower bounds for each of the estimators is constructed, and an integer solution is searched between $\sum_{i=0}^{2} p_i$ and p^{total}.

At this point 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 different combinations of levels computed in parallel have to be considered. This ensures that cases, when all of the processors are allocated on the coarsest level $\{\widehat{Y_0}\}$ first and then all of the processors are distributed across $\{\widehat{Y_1}, \widehat{Y_2}\}$ levels are considered. For the parallel strategy on the estimator, depending on the scenario, allows different schemes to be used.

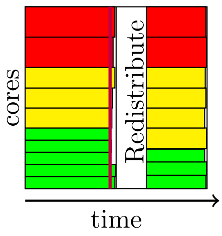
4.3.2 Interrupted Dynamic strategy

This approach is very similar to the processor interruptions. The algorithm starts as a standard dynamic strategy. A heuristic assumption is made that there is no sample to sample computational differences, or if present, they will balanced out. During the parallel computation, due to load imbalances, and sample to sample fluctuations, a group of MPI process, among all of the MPI processes completes, computation before the others. Upon completion this group sends a signal to a part or all of the other groups, that are still computing, informing them that it is in an idle state. Upon receiving the signal, the computing groups interrupts 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*, done within an estimator and the second type of interruption is the *global* one - computation on all MPI processes on all the levels stops. Then a rescheduling is performed. Figure **4.4** illustrates the idea.

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 and enters in idle state. The master process takes responsibility to inform the other groups and synchronize the data between them. The messages carry a small amount of meta-information, and the dominating part of the time needed for the process to complete will be the latency of the system.



(a) *Local*(*Estimator*)



(b) *Global*(*MLMC*)

Figure 4.4: Schematic overview of the interruption process. Different colours represent different levels.

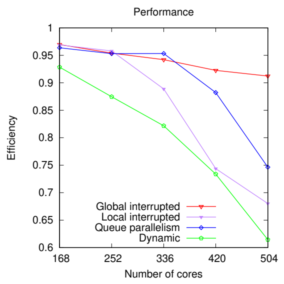
4.3.3 Job queue Dynamic strategy

This procedure 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 **4.11**. For each estimator, using the master-slave programming paradigm, one of the available MPI processes set to be master, and the others are set to be slaves. The master process acts as a dispatcher that assigns work to the slaves. Each of the them performs the given tasks and reports back to the master for more work. This way each process is busy working regardless of the time of a sample. This is at the expense of a large number of small message exchanges. In comparison to the interrupt strategy, where the message exchange is performed at a single point in time, the messages for this strategy are exchanged at different times.

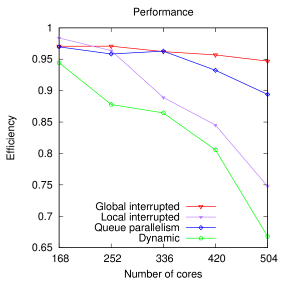
4.3.4 Experiments

Laplace equation parallel experiments

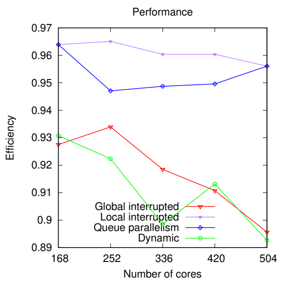
On figure **4.5**, an investigation of the efficiency is done for the different scheduler types. The parallel efficiency is measured under equation **4.8**. For this tests the finest grid of the MLMC algorithm has $2^{10} \times 2^{10}$ number of cells. This is approximately 10^6 unknowns. The number of levels of MLMC is set to 4. On **4.5 a)** parallel efficiency for problem with parameters $\sigma = 2$, $\lambda = 0.3$ and $\epsilon = 1e-3$ is considered. On **4.5 b)** a more computational intensive problem is plotted, with generating parameters $\sigma = 2.25$, $\lambda = 0.4$, and $\epsilon = 1e-3$. For both experiments each sample is computed by a single processor. The figure **4.5 c)** considers the same problem as **4.5 b)**, but with different processor distribution. Each problem on a given level is solved by: $p_0 = 1, p_1 = 5, p_2 = 9, p_3 = 11$ processors respectively. For this processor distribution the efficiency function is $Eff(1, 3.6, 7.35, 9.00)$.



(a) $\sigma = 2.0$ $\lambda = 0.3$
single core per problem



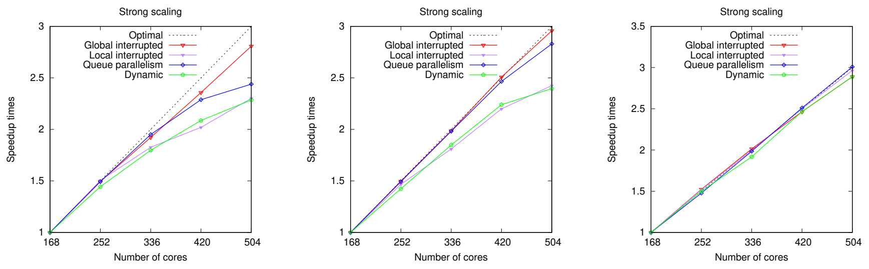
(b) $\sigma = 2.25$ $\lambda = 0.4$
single core per problem



(c) $\sigma = 2.25$ $\lambda = 0.4$
p_0=1, p_1=5, p_2=9, p_3=11

Figure 4.5: Efficiency of different schedulers for Laplace equation

Figures **4.5 a) b)** shows 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 to about 0.08 units better than the pure dynamic approach. There are two main reasons for that. First the increased number of processors leads to larger load imbalances between the different levels when the number of processors per level is estimated. Even if the algorithm finds a good processor distribution and that leads to small waiting time for 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 the large number of messages exchanged at a single point in time. The messages must be sent through the network, and this leads to message flooding of the communication channel. 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. This 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 interrupted technique. It shows small performance degradation when the number of processors is increased. Figure **4.5 c)** shows that utilizing the third layer of parallelism and thus counting all layers of parallel execution, leads to very efficient algorithm. 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 solving of a single sample. Figure **4.6** shows the relative scalability for the considered algorithms



(a) Experiment 4.5 a) (b) Experiment 4.5 b) (c) Experiment 4.5 c)

Figure 4.6: Strong scalability for experiments on figure **4.5**

Convection-reaction-diffusion parallel experiments

On figure **4.7** the efficiency is measured by equation **4.8** for the three different dynamic optimizations: *Local interrupted, Global interrupted, Queue parallelism*. Each sample is solved by a single processes. 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 for that strategy significantly drops at higher number of cores, meanwhile both global and queue approaches retain lower efficiency drop when the number of cores increases. The average number of samples per level for this particular test are shown in table **4.1**

SuperMuc experiments for Laplace equation

The tests in this section are performed on the SuperMuc cluster hosted at Technical University Munich (TUM). Each node is consisting of 48 cores. The total node memory is $96GB$.[1]

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

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

Table 4.1: Performed samples

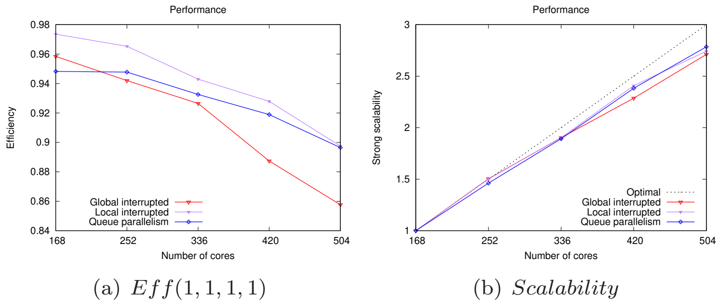


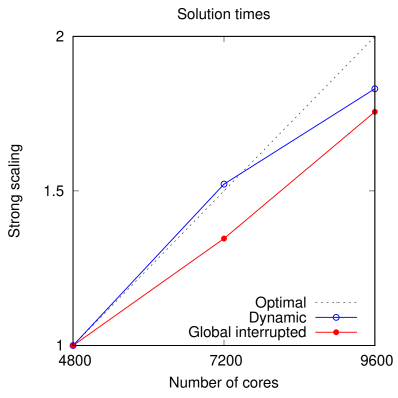
Figure 4.7: Efficiency and scaling

The test case shown on figure **4.8** is designed to prove the effect of the global interruption optimization for the dynamic scheduler for a large number of cores on a relatively small problem, compared to the number of cores used. 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 **4.2**. As a coarsening technique an simplified

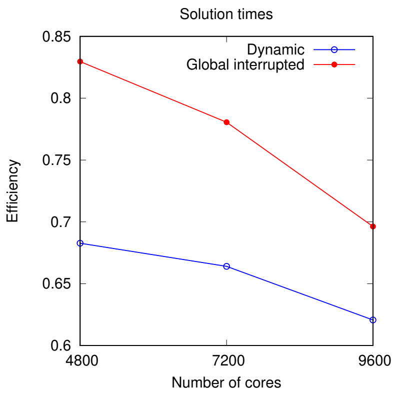
Max. grid size: $2^{10} \times 2^{10}$	$\sigma = 3$	λ=0.3		ϵ= 1e-3	
Cores per problem		Lvl. 0	Lvl. 1	Lvl. 2	Lvl. 3
		1	1	1	1

Table 4.2: Simulation parameters for figure **4.8**

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 the global interrupt optimization leads to significant improvement in the efficiency - close to 85%. In the case of 7200 cores, the optimization gain is around 12%. In the case of 9600 the gain is around 7%. This steady decline of the gains is due to the increased communication time, and the overall smaller computational time. In the case of 9600 cores the total simulation time is only 621 seconds using the global interrupt strategy. More computationally expensive simulations lead to even better effectiveness.



(a) Strong scaling



(b) Efficiency

Figure 4.8: Large number of cores, single core per problem

5. 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] 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.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

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

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

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

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

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

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

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

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

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

[21] 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.

[22] 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.

[23] 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.

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;
  - 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 adaptive algorithm for resource allocation between the different levels of the Multilevel Monte Carlo algorithm has been developed;
  - The Multilevel Monte Carlo method is applied successfully for the first time to solve the convection-reaction-diffusion equation.
- Scientific and applied contributions
  - An approach for determining the levels for the Multilevel Monte Carlo for the two considered problems is defined;
  - Analysis and comparison of two considered approaches for coarse grain, for the Multilevel Monte Carlo versus the classical Monte Carlo for the convection-reaction-diffusion problem are performed;
  - 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;
  - An overview, analysis and comparison of six parallelization strategies are made.
- Applied contributions
  - 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.

List of 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

Acknowledgements

I would like to thank my supervisor assoc. prof Petar Armyanov and my scientific advisor profesor Oleg Iliev for their support during the entire time of my Ph.D. Furthermore, I would like to thank Fraunhofer ITWM and Technical University Munich for the access to their high performance computational systems - Beehive and SuperMuc.

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: Nikolay Shegunov

Digitally signed by Nikolay Shegunov
Date: 2021.06.07 10:39:57 +03'00'