

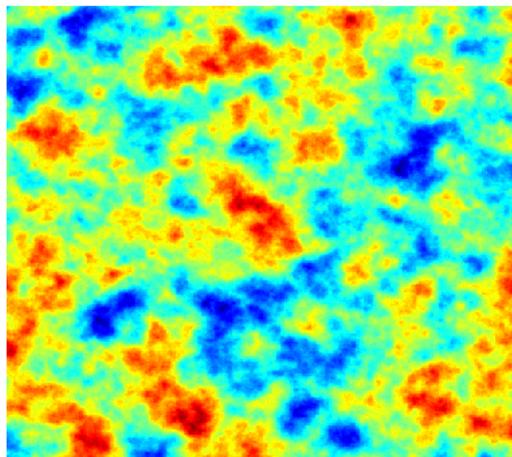
FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT • DEPARTMENT INFORMATIK

Lehrstuhl für Informatik 10 (Systemsimulation)



**Solving Stochastic PDEs with Approximate Gaussian Markov Random
Fields using Different Programming Environments**

Kelvin Kwong Lam Loh



Master Thesis

Solving Stochastic PDEs with Approximate Gaussian Markov Random Fields using Different Programming Environments

Kelvin Kwong Lam Loh

Master Thesis

Aufgabensteller: Prof. Dr. U. Rüde
Betreuer: Dr.-Ing. H. Köstler
Dr.-Ing. B. Gmeiner
S. Kuckuk, M.Sc.
Bearbeitungszeitraum: 15.03.2014 – 15.09.2014

Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Systemsimulation (Informatik 10), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Master Thesis einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 18. August 2014


.....

Abstract

This thesis is a study on the implementation of the Gaussian Markov Random Field (GMRF) for random sample generation and also the Multilevel Monte Carlo (MLMC) method to reduce the computational costs involved with doing uncertainty quantification studies. The GMRF method is implemented in different programming environments in order to evaluate the potential performance enhancements given varying levels of language abstraction. It is seen that the GMRF method can be used to generate Gaussian Fields with a Matérn type covariance function and reduces the computational requirements for large scale problems. Speedups of as much as 1000 can be observed when compared to the standard Cholesky Decomposition sample generation method, even for a relatively small problem size. The MLMC method was shown to be at least 6 times faster than the standard Monte Carlo method and the speedup increases with grid size. It is also seen that in any Monte Carlo type methods, a Krylov subspace type solver is almost always recommended together with a suitable preconditioner for robust sampling.

This thesis also studies the ease of implementation of these methods in varying levels of programming abstraction. The methods are implemented in different languages ranging from the most common language used by mathematicians (MATLAB), to the more performance oriented language (C++-PETSc/MPI), and ends with one of the newest programming concept (ExaStencils). The GMRF method featured in this thesis also is one of the earliest application to be implemented in ExaStencils.

Contents

1	Introduction	1
2	Theory	2
2.1	Gaussian Field	2
2.1.1	Stationary Processes	3
2.1.2	Covariance Function	3
2.2	Gaussian Markov Random Field	3
2.3	Generation of Gaussian Fields	4
2.3.1	Cholesky and Singular Value Decomposition	5
2.3.2	GMRF Approximation	6
2.4	Gaussian White Noise	6
2.5	Standard Monte Carlo	7
2.6	Multilevel Monte Carlo	8
3	Implementation	10
3.1	Finite Volume Discretization	10
3.2	Standard Monte Carlo Parallelization	11
3.2.1	Domain Decomposition	11
3.2.2	Static Master-Slave	11
3.2.3	Dynamic Master-Slave	12
3.3	Multilevel Monte Carlo (MATLAB)	13
3.4	PETSc Details	14
3.4.1	Standard Cholesky decomposition	14
3.4.2	GMRF approximation	14
3.5	ExaStencils	14
3.6	Hardware and Software Specifications	15
4	Results and Discussion	18
4.1	Grid Convergence	18
4.2	Standard Monte Carlo Sampling Convergence	20
4.3	Validation of GMRF approximation	20
4.3.1	1D case	21
4.3.2	2D case	22
4.4	MATLAB	22
4.4.1	MATLAB - Standard Monte Carlo	22
4.4.2	MATLAB - Multilevel Monte Carlo	25
4.4.3	Performance Analysis	26
4.5	PETSc - Standard Monte Carlo	26
4.5.1	Solvers	28
4.5.2	Preconditioners and KSP Solvers	29
4.6	PETSc - Performance Analysis	32
4.6.1	Performance of different parallelization strategies	32
4.6.2	Performance of GMRF compared to Cholesky decomposition	36
4.6.3	Effect of CPU socket utilization	38
4.6.4	Miscellaneous analysis of the GMRF and Variational Poisson program	41

4.7	ExaStencils - Standard Monte Carlo	41
4.7.1	Results	46
4.7.2	Experiences	47
5	Conclusion	50
5.1	Future work	50
5.2	Acknowledgments	50
A	PETSc Solver Settings Report	54
B	Jumpshot Visualizations	55

List of Figures

2.1	Problem domain	2
3.1	FVM discretization of interior cell, Ω_i , and faces, $\Gamma_j^{(i)}$ associated with the cell .	11
3.2	Example DD of matrix for PETSc using the mpiaij type. (Image taken from [3])	12
3.3	The Static Master-Slave (MSS) strategy [$N_s =$ Total number of samples, $P =$ Total number of worker groups]	13
3.4	The Dynamic Master-Slave (MSD) strategy [$N_s =$ Total number of samples, $P =$ Total number of worker groups]	14
3.5	The workflow for the ExaStencils programming paradigm [Image taken from [13]]	16
3.6	The Domain Specific Language (DSL) hierarchy of ExaStencils [Image taken from [13]]	16
4.1	Grid convergence test	20
4.2	SLMC convergence test [Solid line = Curve fit]	20
4.3	Covariance for the point, $x = 0.5$ for the 10000 samples realized, ($\lambda = 0.1$, $\sigma = 0.5$)	21
4.4	Expectation and standard deviation as a function of x for the 10000 samples realized, ($\lambda = 0.1$, $\sigma = 0.5$)	22
4.5	Covariance field for $\mathbf{x} = (0.51, 0.49)$ for the 10000 samples realized, ($\lambda = 0.1$, $\sigma = 0.3$)	23
4.6	GMRF, $a(\mathbf{x})$, random coefficient, $e^{a(\mathbf{x})}$ and solution, $U(\mathbf{x})$ fields for a single realization	23
4.7	Expectation and variance of $a(\mathbf{x})$ as a function of \mathbf{x} for the 10000 samples realized	24
4.8	Expectation and variance of $e^{a(\mathbf{x})}$ as a function of \mathbf{x} for the 10000 samples realized	24
4.9	Expectation and variance of $U(\mathbf{x})$ as a function of \mathbf{x} for the 10000 samples realized	24
4.10	Statistics of Q_M as a function of grid size for the SLMC using $\lambda = 0.1$, $\sigma = 0.3$, sampling error, $\epsilon = 5E - 3$	25
4.11	Statistics of Q_{30} as a function of sampling error for the SLMC using $\lambda = 0.1$, $\sigma = 0.3$, grid size $[30 \times 30]$	26
4.12	MLMC sample case plots for $\lambda = 0.1$, $\sigma = 0.3$, Finest grid size is at $[240 \times 240]$, sampling error, $\epsilon = 5E - 3$	27
4.13	Speedups for the MLMC using $\lambda = 0.1$, $\sigma = 0.3$, finest grid size $[240 \times 240]$, sampling error, $\epsilon = 5E - 3$	27
4.14	GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.01$, $\sigma = 1$	30
4.15	GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.1$, $\sigma = 1$	30
4.16	GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.25$, $\sigma = 1$	30
4.17	GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.1$, $\sigma = 0.3$	31
4.18	GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.1$, $\sigma = 2$	31
4.19	GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.1$, $\sigma = 10$	31
4.20	Speedups as a function of number of processors for different parallelization strategies, 1000 samples, $[500 \times 500]$, 1 processor per sample	33
4.21	Efficiency as a function of number of processors for different parallelization strategies, 1000 samples, $[500 \times 500]$, 1 processor per sample	34
4.22	Efficiency as a function of number of samples for different parallelization strategies, 32 processors, $[500 \times 500]$, 1 processor per sample	34

4.23	Speedups as a function of number of processors for different parallelization strategies, 1000 samples, [500×500], 2 processors per sample	35
4.24	Efficiency as a function of number of processors for different parallelization strategies, 1000 samples, [500×500], 2 processors per sample	35
4.25	Efficiency as a function of number of samples for different parallelization strategies, 32 processors, [500×500], 2 processor per sample	36
4.26	Efficiency as a function of number of processors for different grid sizes, 1000 samples, Dynamic MS, 1 processor per sample	37
4.27	Efficiency as a function of number of samples for different grid sizes, 32 processors, Dynamic MS, 1 processor per sample	37
4.28	Speedup of the GMRF approximation over the Cholesky decomposition approach as a function of number of samples for different number of processors .	38
4.29	Percentage of total wall clock time for Cholesky Decomposition (4 processors, 100x100 grid size, 100 samples)	39
4.30	Percentage of total wall clock time for Cholesky Decomposition (4 processors, 100x100 grid size, 1000 samples)	39
4.31	Percentage of total wall clock time for Cholesky Decomposition (32 processors, 100x100 grid size, 1000 samples)	40
4.32	Efficiency as a function of number of processors for different nsocket values using 2500 samples, [1000×1000] grid and Dynamic master-slave	41
4.33	Efficiency as a function of number of samples for different nsocket values using 32 processors, [1000×1000] grid and Dynamic master-slave	42
4.34	MPE Jumpshot color legend	42
4.35	Processor bindings report from OpenMPI (balanced)	43
4.36	Processor bindings report from OpenMPI (unbalanced)	43
4.37	Jumpshot analysis for 10 processors (balanced)	43
4.38	Jumpshot analysis for 10 processors (unbalanced)	44
4.39	Program output for 10 processors	44
4.40	Percentage of total wall clock time for Dynamic Master-Slave (4 processors, 500x500 grid size, 1000 samples)	45
4.41	Percentage of total wall clock time for Dynamic Master-Slave (64 processors, 500x500 grid size, 1000 samples)	45
4.42	Jumpshot visualization of PETSc program for Dynamic Master-Slave - 4 processors per sample (32 processors, 1000x1000 grid size, 500 samples)	46
4.43	Expectation of the solution field as calculated from the code generated by ExaStencils ([511×511] points), 100 samples	47
4.44	Speedups for ExaStencils	48
B.1	Jumpshot visualization of PETSc program for Dynamic Master-Slave - 1 processor per sample (32 processors, 1000x1000 grid size, 500 samples)	55
B.2	Jumpshot visualization of PETSc program for Dynamic Master-Slave - 2 processors per sample (32 processors, 1000x1000 grid size, 500 samples)	55

List of Tables

2.1	Advantages and disadvantages of the two GF generating methods	5
3.1	Hardware and software specifications	19
4.1	Solver settings for robustness study	28
4.2	Maximum number of iterations for solving Eq. (1) with fgmres+boomeramg preconditioner - [NA = stats problem]	29
4.3	Maximum number of iterations for solving Eq. (1) with boomeramg (symmetric SOR relaxation solver) - [DNC = did not converge, NA = stats problem] . . .	29
4.4	Preconditioner and Solvers	32
4.5	Wall clock times (seconds) to compute 150 samples, with different KSP solvers and preconditioners using 32 processors	32

1 Introduction

The field of uncertainty quantification is important to science and engineering as the variables involved in real-life systems are rarely deterministic. Whenever we solve the Navier-Stokes equations in Computational Fluid Dynamics (CFD), there will always be uncertainties involved with the system coming from various sources. For example the viscosity field, or the boundary conditions since these variables are given as inputs to the simulation. Usually, these variables are obtained from experiments or other sources from which there are errors involved. As such, in order to know the effects of these uncertainties on the solution field, we need to perform an asymptotically unbiased uncertainty quantification experiment. Examples of such simulations are the Monte Carlo and randomized quasi-Monte Carlo methods. The idea is to run multiple independent simulations by considering that the variables involved with the original deterministic system are in fact random variables. By having a large sample set of the resulting solution field, we can then perform statistical analysis on that set to determine the effects of the input uncertainties.

The problem with such methods is that they are computationally very expensive for problem sizes which are large. Suppose one can solve a single M large deterministic system with the computational cost in the order of $\mathcal{O}(M)$, which one can achieve using efficient implementations of multigrid methods. However, when one needs to do Monte Carlo experiments, the computational cost then increases to at least $\mathcal{O}(NM)$, where N is the minimum number of trials to reduce the statistical sampling error. There exist a need to find ways to reduce the cost involved by reducing the number of required samples.

Also, for spatially varying coefficient fields, such as the viscosity, or thermal conductivity, we also need to take into account when there are any spatial correlations between the coefficients in the domain. This is rather common in the field of geophysics and oil reservoir exploration. We then need to find an efficient method to generate the coefficient fields for the simulation in such a way that the statistical properties of the input coefficient fields are still preserved. The problem is due to the covariance function, which when the spatial domain is discretized, result in a dense matrix. There is then a need to efficiently generate these fields by ideally not operating on the dense covariance matrix.

The first problem is addressed by the Multilevel Monte Carlo method, where the primary idea is taken from the Multigrid counterpart in linear algebra solvers. The second problem is addressed by an approximation of the Gaussian field (GF) using a Gaussian Markov Random Field (GMRF).

There are two primary interests for this thesis. The first is in the interest of applied mathematics, and the other in computational engineering. In the applied mathematics interest, this thesis attempts to verify the use of the two recent methods. This thesis also explores the practical implementation and solving methods for the GMRF method. In the same interest as well, the potential performance enhancements are discussed. As for the computational engineering interest, three types of parallelization strategies for the Monte Carlo method are discussed. Also, the methods are implemented in different languages with varying abstraction levels. The idea is to determine the ease of practical implementation of these methods and the experiences of implementation are mentioned in the results. The last objective is also to create an application using the GMRF and Standard Monte Carlo method for a new programming environment called ExaStencils.

This thesis starts of with the description of the deterministic toy problem from which we can easily obtain the solution to it, and then introduce the two recent methods along with

current standard methods in Section 2. We then discuss the implementation of the methods in Section 3. The results of the study are described in Section 4. The section starts with the verification of the newer methods against standard methods, and then we examine the performance gains on the toy problem that can be obtained by using such newer methods.

2 Theory

The variable Poisson equation, Eq. (1) is to be solved in the square domain, $\Omega = [0, 1] \times [0, 1]$, with the exponential Gaussian field, $e^{a(\mathbf{x})}$, as the coefficients. This problem can be thought of physically as a variable heat conduction problem with Dirichlet boundary conditions. The problem can be visualized by Figure 2.1.

$$\begin{aligned}
 \nabla \cdot (e^{a(\mathbf{x})} \nabla U(\mathbf{x})) &= \mathbf{0} & \mathbf{x} \in \Omega &= [0, 1] \times [0, 1] & (1) \\
 U(0, x_2) &= 3 & x_2 \in \partial\Omega_W &= [0, 1] \\
 U(1, x_2) &= 5 & x_2 \in \partial\Omega_E &= [0, 1] \\
 U(x_1, 0) &= 10 & x_1 \in \partial\Omega_S &= [0, 1] \\
 U(x_1, 1) &= 1 & x_1 \in \partial\Omega_N &= [0, 1]
 \end{aligned}$$

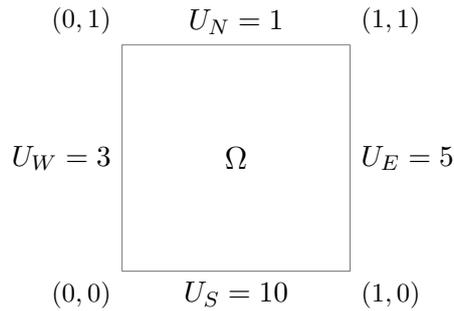


Figure 2.1: Problem domain

2.1 Gaussian Field

In order to generate a Gaussian field (GF), $a(\mathbf{x})$, for Eq. (1), we need to know what is a Gaussian field, and its properties. The following assumes that the reader is already familiar with basic stochastic processes concepts such as random variables, expectation, and variance. For more theoretical background, the reader is invited to refer to a standard text such as [19].

Definition 2.1.1. *A stochastic process, $\{a(\mathbf{x}), \mathbf{x} \in D\}$ where $D \subset \mathbb{R}^d$, and $\mathbf{x} \in D$ represents the location is called a Gaussian process if $(a(\mathbf{x}_1), \dots, a(\mathbf{x}_n))^T$ has a multivariate normal distribution for all $\mathbf{x}_1, \dots, \mathbf{x}_n$*

Since a multivariate normal distribution (MVN), is completely determined by the marginal expectation vector, and covariance matrix, it then follows that a Gaussian process is also

having such a property as well. Hence, a Gaussian random field will have this property

$$a(\mathbf{x}) \sim \text{MVN}(\boldsymbol{\mu}, \boldsymbol{\Sigma}), \mathbf{x} \in \mathbb{R}^d$$

where $\boldsymbol{\mu}(\mathbf{x}) = \text{E}[a(\mathbf{x})]$ is the expectation (vector) of the field, and $C(\mathbf{x}_i, \mathbf{x}_j) = \text{Cov}(a(\mathbf{x}_i), a(\mathbf{x}_j))$ forms the elements in the covariance matrix, $\boldsymbol{\Sigma}$ of the field in \mathbb{R}^d .

2.1.1 Stationary Processes

In this thesis, the random processes are assumed to be stationary.

Definition 2.1.2. *A stochastic process, $\{a(\mathbf{x}), \mathbf{x} \in D\}$, is said to be a stationary process if for all $\mathbf{x} \in D$, $\boldsymbol{\mu}(\mathbf{x}) = \boldsymbol{\mu}$, and if the covariance function only depends on $\mathbf{x}_i - \mathbf{x}_j$, $i \neq j$. A stationary Gaussian Field is also isotropic if the covariance function only depends on the Euclidean distance between \mathbf{x}_i and \mathbf{x}_j , i.e., $C(\mathbf{x}_i, \mathbf{x}_j) = C(\delta x)$, where $\delta x = \sqrt{\|\mathbf{x}_i - \mathbf{x}_j\|_2}$.*

In other words, a process is stationary if choosing any fixed point as the origin, the process has the same probability law [19]. Hence, for a stationary Gaussian random field, such as that in this thesis, the expectation, and covariance matrix is independent of the location of the sampling point origin.

2.1.2 Covariance Function

For any finite set of locations, it can be seen then, that a covariance function must induce a covariance matrix, $\boldsymbol{\Sigma}$ which is positive definite. For isotropic stationary GFs, the matrix is also symmetric. In most applications for geostatistics, one of the following isotropic covariance functions is used [20]:

Exponential	$C(\delta x) = \exp(-3 \delta x)$
Gaussian	$C(\delta x) = \exp(-3 \delta x^2)$
Powered Exponential	$C(\delta x) = \exp(-3 \delta x^\alpha)$, $0 < \alpha \leq 2$
Matérn	See Equation (2)

For this thesis, the Matérn covariance function, Eq. (2) is used to describe the Gaussian random field. Although this seems rather restrictive, the function covers the most important and most used covariance model in spatial statistics. Stein [23] in 1999, even concluded a detailed theoretical analysis with 'Use the Matérn model'. The Matérn covariance function is shown as follows:

$$C(\delta x) = \frac{1}{2^{\nu-1} \Gamma(\nu)} (\kappa \delta x)^\nu K_\nu(\kappa \delta x) \quad (2)$$

where δx is the Euclidean distance between $\mathbf{x}_i, \mathbf{x}_j \in \Omega^d$, K_ν is the modified Bessel function of the second kind and order $\nu > 0$, and κ if a function of ν such that the covariance function is scaled to $C(1) = 0.05$, and $C(0) = 1$. The variance of the distribution becomes σ^2 if we multiply the covariance function, $C(\delta x)$ by σ^2 .

2.2 Gaussian Markov Random Field

From section 2.1.2, we see that the covariance matrix $\boldsymbol{\Sigma}$ is dense. However, for certain types of GFs, the precision matrix, $\mathbf{Q} = \boldsymbol{\Sigma}^{-1}$ can be sparse. For Gaussian Markov Random Fields

(GMRFs), this precision matrix is a more useful matrix than the covariance matrix. The reason is that the off-diagonal elements of the precision matrix characterizes the conditional dependence properties of the distribution.

Very briefly, a GMRF is a multivariate GF that satisfy a certain conditional independence structure. In order to be more rigorous, the concept of a graph is required. For this thesis, the assumptions are that all graphs are finite and unordered unless otherwise specified. Since the graph is finite, the vertices can be numbered from $1, \dots, n$, where n is the number of vertices.

Definition 2.2.1. *A graph is an ordered pair $\mathcal{G} = (\nu, \epsilon)$ comprising a finite set ν of vertices and a set ϵ of edges, where ϵ is a set of unordered pairs of elements of ν . If $\nu = \{1, 2, \dots, n\}$, the graph is a labelled graph.*

The formal definition of a GMRF is then provided by Rue and Held [20].

Definition 2.2.2. *A random vector $\mathbf{x} = (x_1, \dots, x_d)^T \in \mathbb{R}^d$ is called a GMRF wrt a labelled graph $\mathcal{G} = (\nu, \epsilon)$ with expectation $\boldsymbol{\mu}$ and precision matrix $\mathbf{Q} > 0$ iff its density has the form*

$$pdf(\mathbf{x}) = (2\pi)^{-n/2} |\mathbf{Q}|^{1/2} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \mathbf{Q} (\mathbf{x} - \boldsymbol{\mu})\right) \quad (3)$$

and

$$Q_{i,j} \neq 0 \iff \{i, j\} \in \epsilon \quad \forall i \neq j$$

Theorem 2.2.1. (Conditional independence and the precision matrix). *Let $\mathbf{x} \sim N(\boldsymbol{\mu}, \mathbf{Q}^{-1})$, with $\mathbf{Q} > 0$. Then, for $i \neq j$,*

$$x_i \perp x_j | \{x_k : k \in \nu - \{i, j\}\} \iff Q_{i,j} = 0$$

Proof. See Theorem 2.2 from Rue and Held [20] □

Theorem 2.2.1 is a nice and useful result. Simply put, it states that the nonzero pattern of the precision matrix, \mathbf{Q} determines \mathcal{G} , and we can see from \mathbf{Q} whether x_i and x_j are conditionally independent. The converse is also true, which, for a given graph, \mathcal{G} , we can know the nonzero terms in \mathbf{Q} .

2.3 Generation of Gaussian Fields

A Gaussian Field must be generated efficiently for the use in Monte Carlo methods as a field needs to be generated for each independent sample. In this thesis, two types of methods are investigated. The first is the standard method based on Cholesky Decomposition of the Covariance matrix describing the Gaussian field. The second is a recent method which approximates a Gaussian field with a Matérn covariance function. A third method which has not been investigated due to time constraints is the circulant embedding method proposed by Dietrich and Newsam [8]. A summary of the advantages and disadvantages of the two investigated methods is given by Table 2.1.

Method	Advantage	Disadvantage
Direct factorization	<ul style="list-style-type: none"> Arbitrary isotropic stationary covariance functions 	<ul style="list-style-type: none"> Dense covariance matrix, Σ Computationally expensive (memory and operations)
GMRF approximation	<ul style="list-style-type: none"> Sparse precision matrix, \mathbf{Q} Computational cost same as solving another PDE 	<ul style="list-style-type: none"> Restricted to only Matérn covariance functions

Table 2.1: Advantages and disadvantages of the two GF generating methods

2.3.1 Cholesky and Singular Value Decomposition

The Cholesky decomposition method of the covariance matrix is a direct method which is computationally expensive considering the covariance matrix is dense. It is related to the Karhunen-Loeve (KL) expansion (see Lemma 2.3.1) which is used to generate samples with the exponential covariance function in Cliffe et al [6].

Lemma 2.3.1. *The KL expansion is related to the Singular Value Decomposition (SVD) for the covariance matrix. This by extension then is related to the Cholesky decomposition method to generate isotropic stationary Gaussian fields.*

In the 1981 paper by Gerbrands [10], the relationship between KL expansion and SVD is established. The next part of the proof is then to establish a relationship between SVD and the Cholesky decomposition method.

Proof. Any symmetric positive definite (spd) matrix, \mathbf{A} can be decomposed using Cholesky decomposition into:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T$$

The SVD of a spd matrix is given by:

$$\mathbf{A} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$$

where $\mathbf{\Lambda}$ is a diagonal matrix with entries equal to the singular values (or eigenvalues for spd matrices) of \mathbf{A} , and \mathbf{U} are the singular vector matrix in which the columns are the eigenvectors that correspond to the singular values. From there, we can see that:

$$\begin{aligned} \mathbf{A} &= \mathbf{L}\mathbf{L}^T = (\mathbf{U}\sqrt{\mathbf{\Lambda}})(\mathbf{U}\sqrt{\mathbf{\Lambda}})^T \\ \therefore \mathbf{L} &= \mathbf{U}\sqrt{\mathbf{\Lambda}} \end{aligned}$$

□

From Lemma 2.3.1, we can use either KL expansion, SVD, or Cholesky decomposition to generate arbitrary stationary isotropic GFs depending on convenience. This also shows the computational drawback of these methods as they still rely on dense matrix factorizations for the covariance matrix.

The GF, \mathbf{y} with a spd covariance matrix Σ can then be generated by the following:
Suppose $\mathbf{y} \sim N(\mathbf{0}, \Sigma)$ is desired, and SVD is used, then,

$$\begin{aligned} \mathbf{Z} &\sim N(\mathbf{0}, \mathbf{I}) \\ \mathbf{y} &= \mathbf{U}\sqrt{\mathbf{\Lambda}}\mathbf{U}^T\mathbf{Z} \end{aligned}$$

In words, we just need to find the decomposition of the covariance matrix once and then reuse the factors of that matrix to generate GFs with the desired distribution by multiplying with a random variable that has a standard normal distribution.

2.3.2 GMRF Approximation

A Gaussian Field with a Matérn covariance (2) can be approximated by the solution to the SPDE, Eq. (4) as mentioned by Lindgren and Rue in 2011 [15]. Instead of having to do a Karhunen-Loeve expansion (identical to Singular Value Decomposition for the case of zero mean fields [10]) on the dense covariance matrix for the generation of a sample, one can just solve the SPDE, Eq. (4), taking advantage of sparsity in the discretization matrix to generate a GMRF which approximates the Gaussian field instead.

In 2001, Rue and Tjelmeland [21] demonstrated empirically that GMRFs could closely approximate most of the commonly used covariance functions in geostatistics, and they proposed to use them as computational replacements for GFs for computational reasons.

$$(\kappa^2 - \Delta)^{\alpha/2}(\tau a(\mathbf{x})) = W(\mathbf{x}) \quad (4)$$

where $\mathbf{x} \in \Omega^d$, $W(\mathbf{x}) \sim \text{MVN}(\mathbf{0}, \mathbf{1})$ and τ scales the variance of the approximate GF solution to σ^2 .

The solution, $a(x)$, of the SPDE, Eq. (4) has a covariance which approximates Eq. (2) by the following relations, and zero neumann boundary condition, Eq. (5).

$$\nabla a(\mathbf{x}) \cdot \hat{n} = 0, \quad x \in \partial\Omega \quad (5)$$

$$\tau^2 = \frac{\Gamma(\nu)}{\Gamma(\nu)(4\pi)^{d/2}\kappa^{2\nu}\sigma^2} \quad (6)$$

$$\kappa = \frac{\sqrt{8\nu}}{\lambda} \quad (7)$$

$$\nu = \alpha - \frac{d}{2} \quad (8)$$

The boundary condition has to be treated carefully during implementation to ensure that the approximate solution is relatively free from boundary effects. As a rule of thumb proposed by Lindgren and Rue [14], the boundary effect is negligible at a distance, λ , from the boundary. In practice, therefore, we can avoid the boundary effect if we extend the computational domain to at least a distance of λ .

The Gaussian field, $a(\mathbf{x})$ for the coefficient in Eq. (1) is to be generated with a Matern covariance function with constants, $\alpha = 2$, $\sigma = 0.3$, and $\lambda = 0.1$. This is the default case used in this thesis unless otherwise stated.

2.4 Gaussian White Noise

Since the source term of Equation (4) features a Gaussian white noise, $W(\mathbf{x})$, we must know the definition and statistical properties of the Gaussian white noise.

Adler and Taylor [1] defines the following as a Gaussian noise.

Definition 2.4.1. *Let (T, \mathcal{T}, ν) be a σ -finite measure space and denote by \mathcal{T}_ν the collection of sets of \mathcal{T} of finite ν measure. A Gaussian noise based on ν , or 'Gaussian ν -noise' is a random field $W : \mathcal{T}_\nu \rightarrow \mathbb{R}$ s.t., $\forall A, B \in \mathcal{T}_\nu$:*

1. $W(A) \sim N(0, \nu(A))$
2. $A \cap B = \Phi \implies W(A \cup B) = W(A) + W(B)$ a.s.
3. $A \cap B = \Phi \implies W(A)$ and $W(B)$ are independent.

Now that we have a definition of a Gaussian noise, we now proceed to look at its properties. A theorem (2.4.1) introduced by Fuglstad [9] provides the properties that we can use to modify the form of Equation (4).

Theorem 2.4.1. *Let W be a standard Gaussian white noise process on \mathbb{R}^n , for some $n > 0$, and let $L^2(\mathbb{R}^n)$ be the set of Lebesgue square-integrable functions from \mathbb{R}^n to \mathbb{R} . Then the following holds $\forall f, g \in L^2(\mathbb{R}^n)$:*

- i. $\int_{\mathbb{R}^n} f(\mathbf{x})W(\mathbf{x})d\mathbf{x}$ has a Gaussian distribution.
- ii. $\mathbb{E} \left[\int_{\mathbb{R}^n} f(\mathbf{x})W(\mathbf{x})d\mathbf{x} \right] = 0$.
- iii. $\mathbb{E} \left[\left(\int_{\mathbb{R}^n} f(\mathbf{x})W(\mathbf{x})d\mathbf{x} \right) \cdot \left(\int_{\mathbb{R}^n} g(\mathbf{x})W(\mathbf{x})d\mathbf{x} \right) \right] = \int_{\mathbb{R}^n} f(\mathbf{x})g(\mathbf{x})d\mathbf{x}$.

The proof of the theorem can be found in Walsh [26].

2.5 Standard Monte Carlo

Solving just one instance of a SPDE is rather meaningless since the statistics of the solution is what interests us. Hence, the most basic method to obtain such information can be from the standard Monte Carlo (SLMC). The simplest idea is to run multiple independent trials of the deterministic PDE, Eq. (1), using random GFs as input to the system, and then obtain the statistical information of the solution.

A review of the standard Monte Carlo method is provided by this section as this method is used primarily in this thesis. We will then extend this knowledge to the Multilevel Monte Carlo method introduced by Giles in 2008 [11].

Suppose the continuous solution to Eq. (1) is given by \mathbf{U} . The discretized solution can then be given by \mathbf{U}_M , where M is the number of cells contained within the domain, Ω . Suppose now that $Q_M = \mathcal{F}(\mathbf{U}_M)$ be the linear or nonlinear functional of \mathbf{U}_M . In this thesis, the functional is defined by the norm of the solution, $Q_M = \frac{\|\mathbf{U}_M\|_2}{\sqrt{M}}$. The assumption then is that $\mathbb{E}[Q_M] \rightarrow \mathbb{E}[Q]$, as $M \rightarrow \infty$.

The interest is in estimating $\mathbb{E}[Q]$, thus, having a sufficiently large $M \in \mathbb{N}$, we compute the discretization error between the estimator \hat{Q}_M of $\mathbb{E}[Q_M]$ and $\mathbb{E}[Q]$, which can then be quantified via the root mean square error (RMSE)

$$e(\hat{Q}_M) := \left(\mathbb{E} \left[(\hat{Q}_M - \mathbb{E}[Q])^2 \right] \right)^{1/2}$$

In other words, for the PDE application, it just corresponds to choosing a fine enough mesh for the approximation. Hence, a grid convergence study is needed.

The standard Monte Carlo estimator for $\mathbb{E}[Q_M]$ is

$$\hat{Q}_{M,N}^{\text{MC}} := \frac{1}{N} \sum_{i=1}^N Q_M^{(i)} \quad (9)$$

where $Q_M^{(i)}$ is the i -th sample of Q_M and N independent samples are computed in total. Then the mean square error (MSE) of the standard Monte Carlo estimator, $e(\hat{Q}_{M,N}^{\text{MC}})^2$ is given by [11]:

$$e(\hat{Q}_{M,N}^{\text{MC}})^2 = \frac{\mathbb{V}(Q_M)}{N} + (\mathbb{E}[Q_M - Q])^2 \quad (10)$$

The first term of the MSE in Eq. (10) is the variance of the estimator and represents the sampling error (i.e. the error due to the finite-ness of the samples). The second term represents the discretization error.

2.6 Multilevel Monte Carlo

The idea of the Multilevel Monte Carlo (MLMC) method recently developed by Giles [11] is simple. We sample not just from one approximation Q_M of Q , but from multiple levels. This thesis will only outline the main idea of the method. More details can be obtained from the main paper [11].

Let $M_l : l = 0, \dots, L$ be an increasing sequence in \mathbb{N} called *levels*, i.e. $M_0 < M_1 < \dots < M_L =: M$, and assume for simplicity that $\exists s \in \mathbb{N} \setminus \{1\}$ s.t.

$$M_l = sM_{l-1}, \quad \forall l = 1, \dots, L \quad (11)$$

Similar to the multigrid methods applied to linear algebraic systems, the key is to avoid estimating $\mathbb{E}[Q_{M_l}]$ directly on level l , but instead we estimate the correction with respect to the next lower level, i.e. $\mathbb{E}[Y_l]$, where $Y_l := Q_{M_l} - Q_{M_{l-1}}$. The linearity of the expectation operator then implies that

$$\mathbb{E}[Q_M] = \mathbb{E}[Q_{M_0}] + \sum_{l=1}^L \mathbb{E}[Q_{M_l} - Q_{M_{l-1}}] = \sum_{l=0}^L \mathbb{E}[Y_l] \quad (12)$$

where $Y_0 := Q_{M_0}$.

Hence, the expectation on the finest level is equal to the expectation on the coarsest level, $\mathbb{E}(Q_{M_0})$, plus a sum of corrections, $\sum_{l=1}^L \mathbb{E}[Y_l]$ of the difference in expectation between simulations on consecutive levels. The multilevel idea is then to independently estimate each of these expectations such that the overall variance is minimized for a fixed computational cost.

Let \hat{Y}_l be an unbiased estimator for $\mathbb{E}[Y_l]$, e.g. the SLMC estimator

$$\hat{Y}_{l,N_l}^{\text{SL}} := \frac{1}{N_l} \sum_{i=1}^{N_l} \left(Q_{M_l}^{(i)} - Q_{M_{l-1}}^{(i)} \right) \quad (13)$$

with N_l samples. The multilevel estimator is then defined as in Equation 14.

$$\hat{Q}_M^{\text{ML}} := \sum_{l=0}^L \hat{Y}_l \quad (14)$$

If the individual terms are estimated using standard Monte Carlo (SLMC), i.e. Eq. 13 with N_l samples on level l , then the estimator, Eq. (14) is called the multilevel Monte Carlo. We denote it by $\hat{Q}_{M,\{N_l\}}^{\text{MLMC}}$. The quantity $Q_{M_l}^{(i)} - Q_{M_{l-1}}^{(i)}$ in Eq. (13) has to come from using the same random sample $\omega^{(i)} \in \Omega$ on both levels M_l and M_{l-1} . This thesis uses the SLMC and MLMC methods.

We already know that the expectations $\mathbb{E}[\hat{Y}_l]$ are estimated independently, the variance of the MLMC estimator is then $\mathbb{V}(\hat{Q}_M^{\text{MLMC}}) = \sum_{l=0}^L N_l^{-1} \mathbb{V}(Y_l)$. The mean square error (MSE) as defined by Giles [11] is in the form as shown in Eq. (15).

$$e(\hat{Q}_M^{\text{MLMC}})^2 := \mathbb{E} \left[(\hat{Q}_M^{\text{MLMC}} - \mathbb{E}[Q])^2 \right] = \sum_{l=0}^L N_l^{-1} \mathbb{V}(Y_l) + (\mathbb{E}[Q_M - Q])^2 \quad (15)$$

Similar to the SLMC, we notice that the MSE consists of two terms, sampling error, and discretization error. Again, they are assumed to be independent of each other. It is then claimed that MLMC is cheaper than the SLMC to reduce the sampling error term to less than $\epsilon^2/2$ for the following two reasons:

- If $\mathbb{E}[(Q_M - Q)^2] \rightarrow 0$ as $M \rightarrow \infty$, then, $\mathbb{V}(Y_l) = \mathbb{V}(Q_{M_l} - Q_{M_{l-1}}) \rightarrow 0$ as $l \rightarrow \infty$. It is then possible to choose $N_l \rightarrow 1$ as $l \rightarrow \infty$.
- The coarsest level $l = 0$ and thus M_0 can be kept fixed for all ϵ , and so the cost per sample on level $l = 0$ does not grow as $\epsilon \rightarrow 0$.

Practically, M_0 must be chosen sufficiently large to provide a minimum resolution to the problem. For the problem described in the thesis, this depends on the regularity of the covariance function of the coefficient field, $e^{a(x)}$ and on the correlation length.

The computational cost of the multilevel Monte Carlo estimator is

$$\mathcal{C}(\hat{Q}_M^{\text{MLMC}}) = \sum_{l=0}^L N_l \mathcal{C}_l \quad (16)$$

where $\mathcal{C}_l := \mathcal{C}(Y_l^{(i)})$ represents the cost of a single sample of Y_l . Treating the N_l as continuous variables, the variance of the MLMC estimator is minimized for a fixed computational cost by choosing

$$N_l \propto \sqrt{\mathbb{V}(Y_l)/\mathcal{C}_l} \quad (17)$$

with the constant of proportionality chosen so that the overall variance is $\epsilon^2/2$. The total cost on level l is proportional to $\sqrt{\mathbb{V}(Y_l)\mathcal{C}_l}$ and hence

$$\mathcal{C}(\hat{Q}_M^{\text{MLMC}}) \lesssim \sum_{l=0}^L \sqrt{\mathbb{V}(Y_l)\mathcal{C}_l} \quad (18)$$

Putting all the ideas together, the original MLMC algorithm as outlined by Cliffe et al [6] are as follows:

1. Start with $L = 0$.
2. Estimate $\mathbb{V}(Y_L)$ by the sample variance of an initial number of samples.
3. Calculate the optimal N_l , $l = 0, 1, \dots, L$ using Eq. (17).
4. Evaluate extra samples at each level as needed for the new N_l .
5. If $L \geq 1$, test for convergence using $\hat{Y}_L \lesssim M^{-\alpha}$.
6. If not converged, set $L = L + 1$ and go back to 2.

Note that in the above algorithm, step 3 aims to make the variance of the MLMC estimator less than $\frac{1}{2}\epsilon^2$, while step 5 tries to ensure that the remaining bias is less than $\frac{1}{\sqrt{2}}\epsilon$.

3 Implementation

This section will discuss the numerical discretization and algorithms used to solve the stochastic PDE system.

3.1 Finite Volume Discretization

The problem is to be discretized with the Finite Volume Method (FVM) for both the SPDE (4), and Eq. (1). The stencil is shown in Figure 3.1. The FVM discretization for Eq. (1) follows the usual manner [24] for variable elliptic problems since it deals with deterministic variables for each independent sample set. However, special care must be treated with respect to the SPDE described by Eq. (4). To discretize the SPDE, we follow the same steps as averaging over a control volume, and then applying Gauss' theorem to the diffusion term. The difference lie in the treatment of the Gaussian white noise term, $W(\mathbf{x})$, where the equality sign, "=", in the SPDE, is to be understood as having equal statistical properties, " $\stackrel{d}{=}$ " over the finite dimensional distributions (i.e. same expectation, variances, etc...).

Starting from Eq. (4) and using Theorem 2.4.1, assuming $\alpha = 2$, and suppose we are discretizing over the i -th control volume, Ω_i with volume $d\Omega_i$, and surface $\partial\Omega_i$:

$$(\kappa^2 - \Delta)a(\mathbf{x}) = W(\mathbf{x}) \tag{19}$$

$$\int_{\Omega_i} (\kappa^2 a(\mathbf{x}) - \nabla \cdot \nabla a(\mathbf{x})) d\Omega \stackrel{d}{=} \int_{\Omega_i} W(\mathbf{x}) d\Omega \tag{20}$$

$$\int_{\Omega_i} \kappa^2 a(\mathbf{x}) d\Omega - \oint_{\partial\Omega_i} \nabla a(\mathbf{x}) \cdot \hat{n} d\Gamma \stackrel{d}{=} Z \sqrt{\int_{\Omega_i} d\Omega} \tag{21}$$

$$\kappa^2 a(\mathbf{x}_i) d\Omega_i - \sum_{j=1}^{N_f} (\nabla a(\mathbf{x}_j) \cdot \hat{n})|_{d\Gamma_j} d\Gamma_j \stackrel{d}{=} Z \sqrt{d\Omega_i} \tag{22}$$

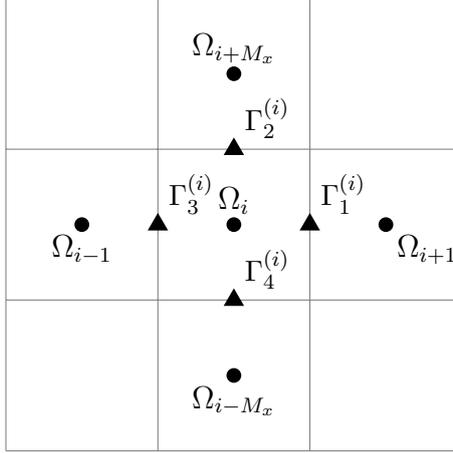


Figure 3.1: FVM discretization of interior cell, Ω_i , and faces, $\Gamma_j^{(i)}$ associated with the cell

where N_f = number of discretized faces over the surface $\partial\Omega_i$, and $Z \sim \mathcal{N}(0, 1)$.

As an example, the FVM implementation done in this thesis referring to Figure 3.1, Eq. (22) is shown as:

$$\kappa^2 a_i d\Omega_i - \left(\Gamma_1^{(i)} + \Gamma_3^{(i)} \right) \Delta y - \left(\Gamma_2^{(i)} + \Gamma_4^{(i)} \right) \Delta x = Z_i \sqrt{d\Omega_i} \quad (23)$$

where $d\Omega_i = \Delta x \Delta y$, and we use linear interpolation in between the cell values for the faces, e.g. $\Gamma_1^{(i)} = \frac{1}{2} (\Omega_i + \Omega_{i+1})$.

3.2 Standard Monte Carlo Parallelization

Monte Carlo methods are examples of embarrassingly parallel problems since the samples are independent of each other. Therefore, the Work Pool/Processor Farms will be a good strategy [27]. Nevertheless, 3 types of parallelization strategy for the SLMC, namely, Domain Decomposition, Static Master-Slave, and Dynamic Master-Slave were implemented.

3.2.1 Domain Decomposition

Within the Monte Carlo method (either SLMC or MLMC), we can parallelize the computation of a sample. Algorithm 1 shows the DD main pseudocode as implemented for this thesis. The "Unit Work" step is inherently made parallel by default using the Domain Decomposition (DD) strategy while the MATLAB code is not utilizing this strategy. The DD strategy deals with the decomposition of the matrices and vectors involved in solving the problem to the multiple processors. In PETSc, the `mpiaij` parallel sparse matrix type is set up such that each process locally owns a submatrix of contiguous global rows. Then, each submatrix consists of diagonal and off-diagonal parts of the matrix. Figure 3.2 shows the decomposition of a matrix using the PETSc `mpiaij` type.

3.2.2 Static Master-Slave

The Static Master-Slave (MSS) is also known as Static Task Assignment [27]. In this strategy the total number of samples are assumed to be known before the simulation and the samples

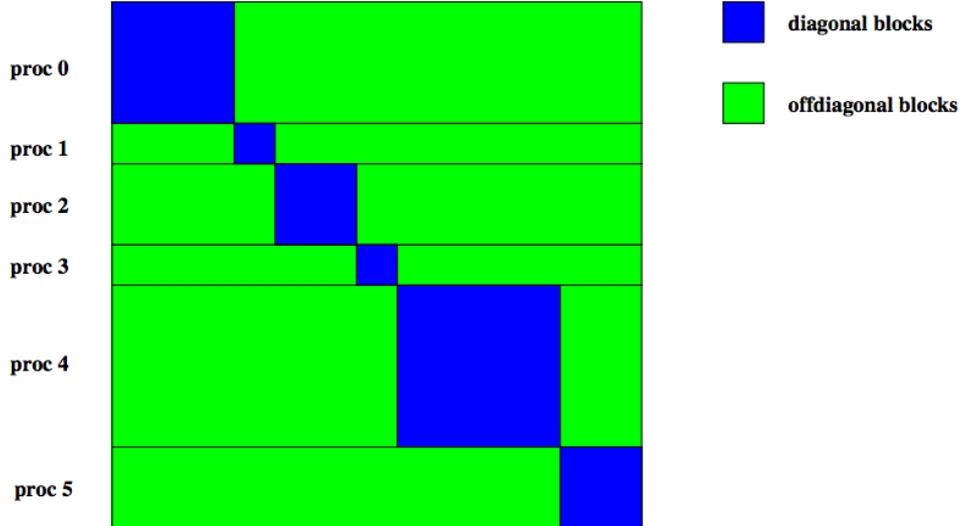


Figure 3.2: Example DD of matrix for PETSc using the mpiadj type. (Image taken from [3])

Algorithm 1: DD main pseudocode as applied to SLMC

```

Initialize random generator with processor rank as seed;
Setup GMRF operator matrix;
while ( $N < N_s$ ) and ( $tol > \epsilon$ ) do
  NormU  $\leftarrow$  Solve "Unit Work";
  tol  $\leftarrow$  Update stats;
  ++N;

```

are distributed evenly amongst the processors. After the computations are done in each of the samples, a reduction to the root processor is performed to obtain the result. Figure 3.3 shows the main idea of the MSS implementation. The main MSS implementation is represented by Algorithm 2. It has to be noted that the tol termination criteria is local to the Slave Processor group. Also, this implementation is the most basic implementation of concurrent sample computations. The advantage of having a static task assignment is that even the root processor can take part in the computation step, and thus increases the efficiency of the parallelization. However, the disadvantage is that the implementation cannot load balance itself automatically as well as being unable to sense a global sampling error termination criterion. Another disadvantage is also that the number of samples must be divisible by the total number of Slave Groups.

3.2.3 Dynamic Master-Slave

The Dynamic Master-Slave (MSD) is based on a centralized dynamic load balancing method which was implemented for the SLMC through PETSc. The master (root) process holds the collection of tasks to be performed. The tasks are then sent to the slave processes. When a slave process completes one task, it requests another task from the master process [27]. Figure 3.4 shows the main idea of the MSD implementation. The main MSD implementation is represented by Algorithm 3. The disadvantage is that the root process takes no part in

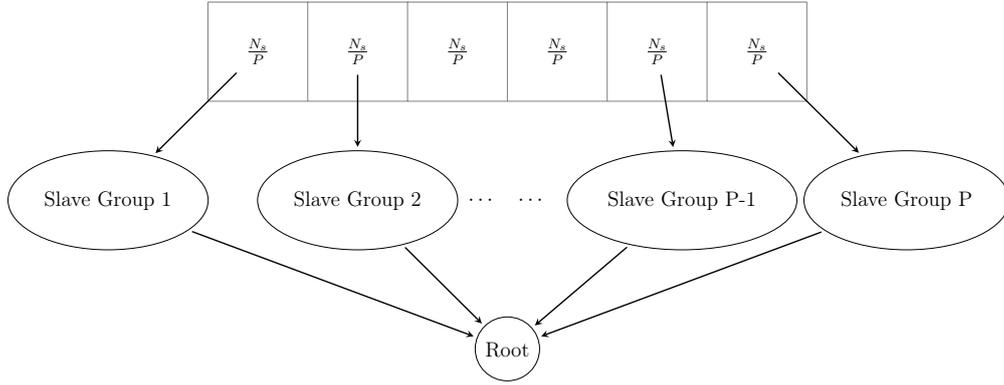


Figure 3.3: The Static Master-Slave (MSS) strategy [N_s = Total number of samples, P = Total number of worker groups]

Algorithm 2: MSS main pseudocode as applied to SLMC

```

Split communicator;
Initialize random generator with processor rank as seed;
if  $\text{mod}(N_s, P) == 0$  then
  |  $N_{\text{spc}} \leftarrow N_s/P$ ;
else
  |  $N_{\text{spc}} \leftarrow N_s/P + 1$ ;
Setup GMRF operator matrix;
for ( $N_s \leftarrow 1$  to  $N_{\text{spc}}$ ) and ( $\text{tol} < \epsilon$ ) do
  | NormU  $\leftarrow$  Solve "Unit Work";
  |  $\text{tol} \leftarrow$  Update stats;
Reduce sum to root processor  $\rightarrow$  NormU;

```

the computation of the samples, hence, efficiency can be lower if generally the number of samples are less or equal to the total number of processes involved. Another disadvantage would be that the master process can only issue one task at a time. Thus, there is a potential for a bottleneck to occur when there are many slave processes making simultaneous requests. The primary advantage of using the MSD concept is that the system is automatically load balanced (i.e. the faster processors will compute more samples). Another advantage of MSD is that the termination condition for the program is simply a matter of the master process to recognize whenever it is met.

3.3 Multilevel Monte Carlo (MATLAB)

As mentioned in Section 2.6, Cliffe et al [6] outlined an algorithm for the Multilevel Monte Carlo (MLMC). As we see later that the error terms are independent of each other, we can devise another algorithm so that the error reductions are also independent, and that this algorithm (4) is purely only for sampling error reduction. It is important to also note that the SLMC is used as an estimator for calculating the samples, and hence, we can parallelize the algorithm by parallelizing the SLMC steps (Algorithm 5). The MATLAB Parallel Toolbox

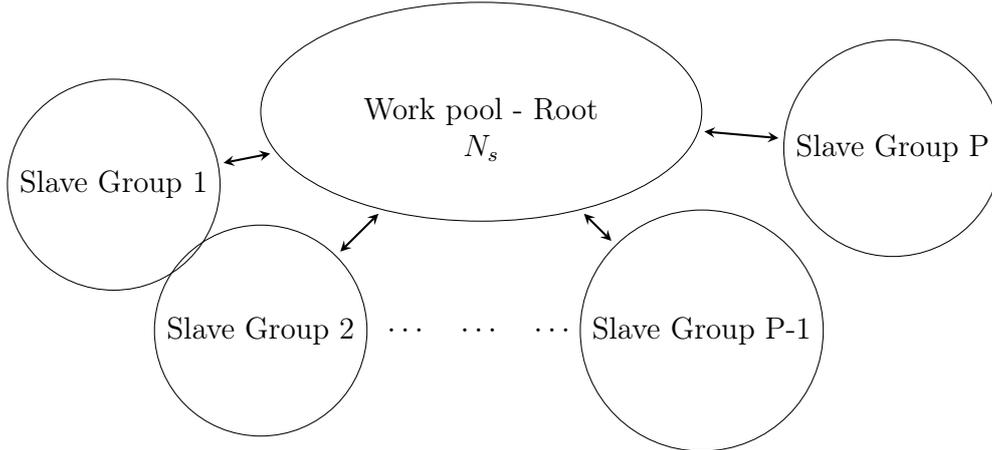


Figure 3.4: The Dynamic Master-Slave (MSD) strategy [N_s = Total number of samples, P = Total number of worker groups]

was used to parallelize the MATLAB implementation.

3.4 PETSc Details

The SLMC method is implemented in PETSc with all 3 parallelization strategies. The "Unit Work" routine for PETSc as mentioned in the previous Algorithms (Alg. 1 to 3) are shown in Algorithm 6. We can see then that all of the steps in the "Unit Work" routine can be parallelized using the DD strategy.

3.4.1 Standard Cholesky decomposition

As we have seen in the "Unit Work" routine (Alg. 6) as well as Section 2.3, there are many ways to generate the Gaussian Field. Algorithm 7 describes the implementation of the Cholesky Decomposition method to generate the Gaussian field.

3.4.2 GMRF approximation

The newest method to generate an approximate GF with the Matérn covariance function is described by Algorithm 8. The effects of the boundary is minimized by following the rule-of-thumb as described by Section 2.3.2. This is implemented by padding more ghost cells until the $a(\mathbf{x})$ domain is larger than the $U(\mathbf{x})$ domain by the correlation length, λ . $a(\mathbf{x})$ is then solved in the larger domain and the corresponding field is transferred directly to Ω . For example, suppose $\lambda = 0.1$, then, $a(\mathbf{x})$ is solved in the domain $[-0.1, 1.1] \times [-0.1, 1.1]$, and the field within $\Omega \in [0, 1] \times [0, 1]$ is used.

3.5 ExaStencils

This thesis also explores the use of a new software technology for applications with exascale performance called ExaStencils [13]. The idea is that at different levels of abstraction to solve a stencil-based problem, the choices made at every refinement step leads to the goal of exascale performance. Each step also benefits from the knowledge of experts at different levels.

Algorithm 3: MSD main pseudocode as applied to SLMC

```
Split communicator excluding root process;
Initialize random generator with processor rank as seed;
Setup GMRF operator matrix;
if (root process) then
    Send initial tasks;
    while ( $N < N_s$ ) and ( $tol > \epsilon$ ) do
        NormU  $\leftarrow$  Receive from slave group;
        tol  $\leftarrow$  Update stats;
        ++N;
        if ( $N_a < N_s$ ) or ( $tol > \epsilon$ ) then
            Send work signal;
            ++Na;
    Send termination signal;
else
    Receive signal;
    while (No terminate signal) do
        NormU  $\leftarrow$  Solve "Unit Work";
        Send NormU;
        Receive signal;
```

Figure 3.5 shows the workflow of ExaStencils to produce a final Exascale code. An end user (e.g. in DSL level 4, fig. 3.6) will provide a Domain Specific Language (DSL) program. The ExaStencils compiler will harness the domain knowledge of the different experts at different levels to finally generate a tuned code for a target machine. The problem described by this thesis is one of the first applications of this programming paradigm. We implemented a GMRF approximation method for the generation of the GF to solve Eq. (1) in DSL level 4. The target code generated was in C++ and is tuned for the cluster of the Computer Science Department (LSS) at the University of Erlangen-Nürnberg [7].

A code snippet from the DSL program to define the boundary conditions of the problem is shown in Listing 1.

Listing 1: DSL level 4 snippet for BC definition

```
def bcSol (xPos : Real, yPos : Real) : Real {
    if ( yPos >= 1.0 ) { return ( UN ) }
    if ( xPos >= 1.0 ) { return ( UE ) }
    if ( yPos <= 0.0 ) { return ( US ) }
    if ( xPos <= 0.0 ) { return ( UW ) }
    return ( 0.0 )
}
```

3.6 Hardware and Software Specifications

Table 3.1 are the specifications of the hardwares and software libraries used to obtain the results in this thesis.

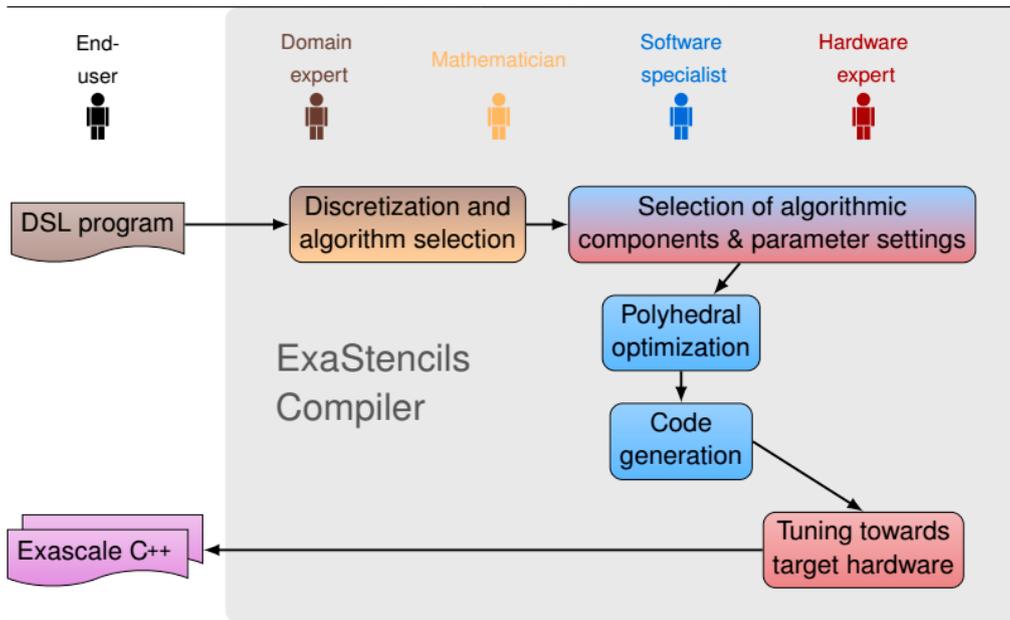


Figure 3.5: The workflow for the ExaStencils programming paradigm [Image taken from [13]]

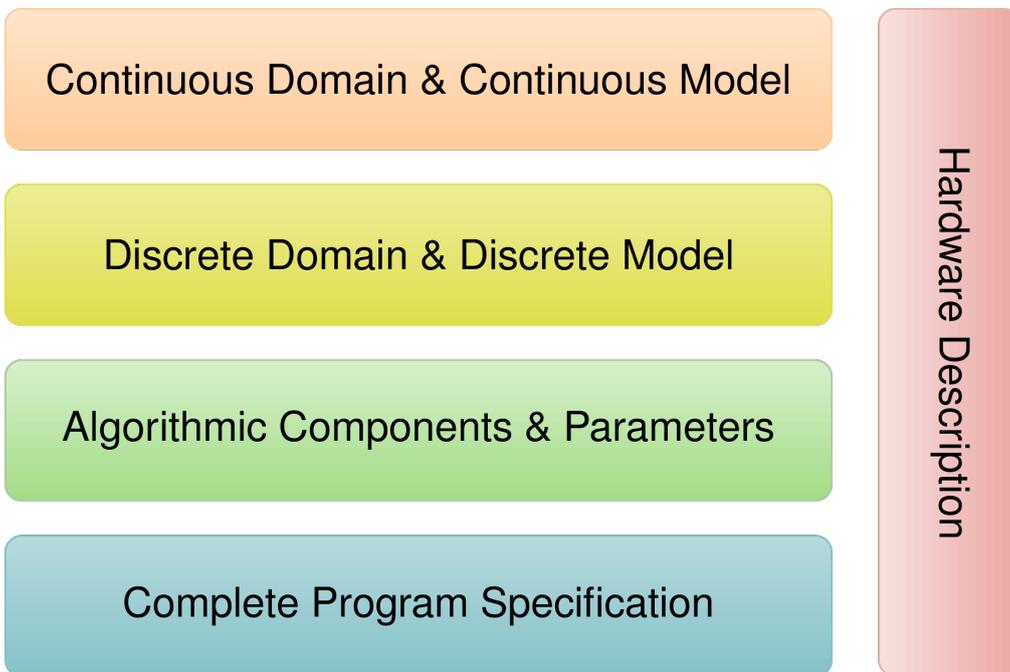


Figure 3.6: The Domain Specific Language (DSL) hierarchy of ExaStencils [Image taken from [13]]

Algorithm 4: MLMC algorithm as implemented in MATLAB

```
for ( $l < L_{max}$ ) do
  Estimate  $\mathbb{V}(Y_l)$  by the initial number of samples;
 $V_{Q_{ml}} \leftarrow \sum_{l=0}^L N_l^{-1} \mathbb{V}(Y_l)$ ;
 $tol \leftarrow \sqrt{V_{Q_{ml}}}$ ;
while ( $tol > \epsilon$ ) and ( $iter < iter_{max}$ ) do
  Calculate  $N_l$  based on Eq. (17);
  for ( $l < L_{max}$ ) do
    Calculate extra samples as needed for the new  $N_l$ ;
   $V_{Q_{ml}} \leftarrow \sum_{l=0}^L N_l^{-1} \mathbb{V}(Y_l)$ ;
   $tol \leftarrow \sqrt{V_{Q_{ml}}}$ ;
```

Algorithm 5: The SLMC estimator in the MLMC method as implemented in MATLAB

```
if  $l \neq 0$  then
  for  $N < N_s$  [Parallel MSD] do
     $a(\mathbf{x}) \leftarrow$  Solve GMRF( $M_x, M_y$ );
    NormU1  $\leftarrow$  Solve PDE( $M_x, M_y, e^{a(\mathbf{x})}$ );
     $a(\mathbf{x}) \leftarrow$  Bilinear interpolation downsample( $M_x, M_y, M_x/2, M_y/2$ );
    NormU2  $\leftarrow$  Solve PDE( $M_x/2, M_y/2, e^{a(\mathbf{x})}$ );
     $Y_l \leftarrow$  NormU1 - NormU2;
else
  for  $N < N_s$  [Parallel MSD] do
     $a(\mathbf{x}) \leftarrow$  Solve GMRF( $M_x, M_y$ );
    NormU  $\leftarrow$  Solve PDE( $M_x, M_y, e^{a(\mathbf{x})}$ );
     $Y_0 \leftarrow$  NormU;
```

The PETSc library for performance benchmarking was compiled using the following configuration command and options:

```
./configure PETSC_ARCH=linux_gnu_all_cluster
--download-f-blas-lapack --download-hypre
--download-superlu_dist --download-parmetis
--download-metis --download-suitesparse
--download-fftw --download-scalapack
--download-mumps --with-clanguage=cxx
--with-mpi-dir=/software/sles/openmpi/1.6.5-ib/
--with-debugging=no --with-shared-libraries=0
```

Algorithm 6: The "Unit Work" routine as implemented in PETSc

$W(\mathbf{x}) \leftarrow$ Set Gaussian white noise field $\sim \mathcal{N}(0, 1)$;
 $a(\mathbf{x}) \leftarrow$ Generate Gaussian Field($W(\mathbf{x})$);
 $a(\mathbf{x}) \leftarrow$ Scale($\frac{a(\mathbf{x})}{\tau^2}$);
 $\mathbf{A} \leftarrow$ Set PDE operator($e^{a(\mathbf{x})}$);
 $\mathbf{b} \leftarrow$ Set PDE source and boundary conditions($e^{a(\mathbf{x})}$);
NormU \leftarrow Solve(\mathbf{A}, \mathbf{b});
return NormU

Algorithm 7: Cholesky Decomposition routine to generate GF as implemented in PETSc

Outside "Unit Work" performed once and residing in all Slave Groups:
 $\mathbf{Cov} \leftarrow$ Generate Covariance matrix;
 $\mathbf{L} \leftarrow$ Get Cholesky Factor(\mathbf{Cov});
Inside "Unit Work(\mathbf{L})":
 $a(\mathbf{x}) \leftarrow \mathbf{L}W(\mathbf{x})$;
return $a(\mathbf{x})$;

A slightly different PETSc library with MPE profiling was used for the Jumpshot visualizations of the PETSc code. The following are the configuration command and options used to build this version of the PETSc library:

```
./configure PETSC_ARCH=linux_gnu_all_cluster_log_C
--download-f-blas-lapack --download-hypre
--download-superlu_dist --download-parmetis
--download-metis --download-suitesparse
--download-fftw --download-scalapack
--download-mumps --with-mpe-dir=~ /mpe/
--with-mpi-dir=/software/sles/openmpi/1.6.5-ib/
--with-debugging=no --with-shared-libraries=0
```

4 Results and Discussion

As previously discussed in Section 2.5, there are two independent error terms involved in the estimator for any Monte Carlo type simulations, namely discretization and sampling errors. A grid convergence study must be performed in order to have a sufficient sample solution accuracy due to the discretization error. A test for the sampling convergence rate over different grid sizes must be performed to validate the claim that the two error terms are independent of each other.

4.1 Grid Convergence

Since we already know the expectation of the GF, i.e. $\mathbb{E}[e^{a(\mathbf{x})}] = \mathbf{1}, \forall \mathbf{x} \in \Omega$, the test is nothing more than solving a Laplace equation subject to the same Dirichlet boundary conditions as Eq. (1). Figure 4.1a shows the grid convergence study result performed using

Algorithm 8: GMRF routine to generate approximate GF as implemented in PETSc

Outside "Unit Work" performed once and residing in all Slave Groups:

$\mathbf{G} \leftarrow$ Generate GMRF sparse matrix operator;

Inside "Unit Work(\mathbf{G})":

$a(\mathbf{x}) \leftarrow$ Solve($\mathbf{G}, W(\mathbf{x})$);

return $a(\mathbf{x})$;

Machine	Hardware	Software
Laptop	<ul style="list-style-type: none">• Intel Core-i7 2630QM• 8 GB RAM	<ul style="list-style-type: none">• CentOS 6.3• MATLAB R2013a (8.1.0.604) 64-bit (glnxa64)• MATLAB Parallel Computing Toolbox V6.2
LSS Cluster	<ul style="list-style-type: none">• 8 compute nodes• Each node has 4 x Intel(R) Xeon(R) CPU E7-4830• Each CPU: 2.13 GHz - 2.4GHz (max. turbo) (8 cores + SMT)• SSE 4.1/4.2• 24 MB shared cache• 256 GB RAM• QDR Infiniband	<ul style="list-style-type: none">• OpenMPI V1.6.5 (InfiniBand)• GCC 4.9.0• CMAKE 2.8.11.1• PETSc 3.4.4• MPE 2.4.6• Jumpshot 4

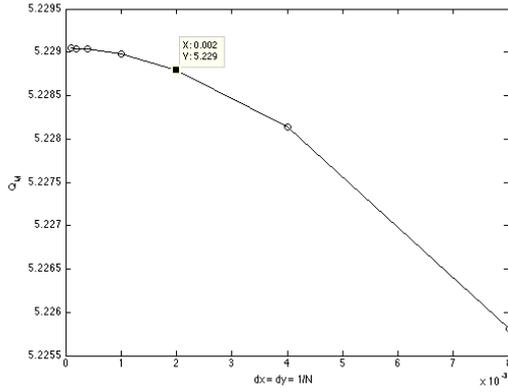
Table 3.1: Hardware and software specifications

the PETSc code. As one can see, the PETSc implementation does behave correctly with the same order of convergence as that of a 'second' order implementation, which is true for the finite volume discretization method used in this thesis. The grid convergence test by Richardson extrapolation gives an average order of convergence, $p = 1.84$ which is lower than 2, but this is expected due to boundary condition treatment, numerical models, and grid implementations [18]. The order of convergence, p , is calculated by using Eq. (24). As we can see, Eq. (24) only needs 3 values of the functional at different grid sizes, while we have many, hence, the p reported is an average of all the different grid sizes. The exact Richardson extrapolation as calculated from the results of the PETSc implementation is $Q = 5.229$.

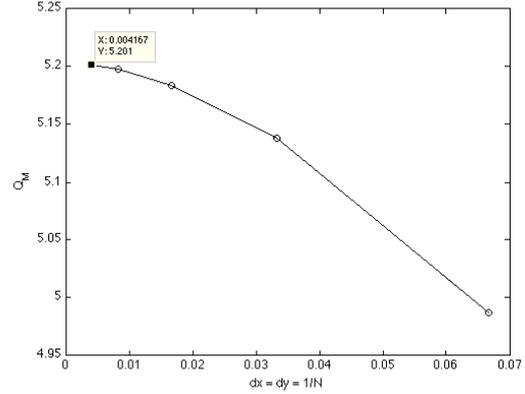
$$p = \ln \left(\frac{Q_3 - Q_2}{Q_2 - Q_1} \right) / \ln(r) \quad (24)$$

where Q_3 is the functional value obtained by the finest grid, and Q_1 by the coarsest, with the number of cells in each direction related by, $M_{Q_3} = rM_{Q_1}$. We are able to use that relation for the 2D case since we discretize the domain with equal number of cells in each direction. For this thesis, the constant grid refinement ratio, $r = 2$.

The result of the implementation from MATLAB (Fig. 4.1b) using much coarser grids show that the order of convergence, $p = 1.75$, with an exact Richardson extrapolation of $Q = 5.203$. This differs from the PETSc implementation, and could be due to the use of a coarse grid in MATLAB to derive the order of convergence and exact solution. The use of the coarse grids are due to hardware limitations as the MATLAB code is run in a laptop. As the results are close to each other, the implementations for both PETSc and MATLAB seem to be correct.

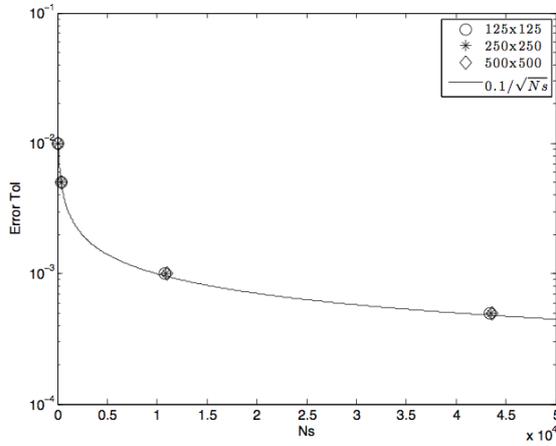


(a) PETSc

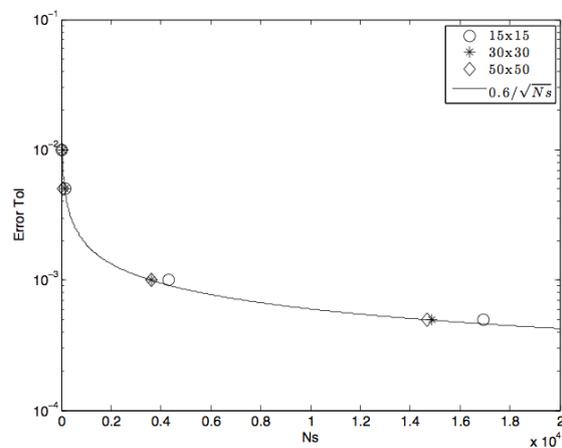


(b) MATLAB

Figure 4.1: Grid convergence test



(a) PETSc



(b) MATLAB

Figure 4.2: SLMC convergence test [Solid line = Curve fit]

4.2 Standard Monte Carlo Sampling Convergence

The sampling convergence rate of the standard Monte Carlo method (SLMC) is known to be $O(\frac{1}{\sqrt{N}})$ [4], and is independent of the grid size. Figures 4.2a and 4.2b show the reduction in the sampling error term as a function of number of samples for the PETSc and MATLAB implementations, respectively. They both show that the error term is independent of the problem grid size. The results are validated against the statement made by Caffisch in 1998 [4]. The bad news then is that for larger grid sizes, the computational cost will combine to be at best $O(NM)$, where M is the number of cells in the domain, and N is the number of samples in the standard Monte Carlo method.

4.3 Validation of GMRF approximation

In order to validate the use of the GMRF approximation, two smaller cases were performed in MATLAB using the standard Monte Carlo. The first case is a 1D simulation of just generating the GMRF and then comparing the results with the actual analytical covariance

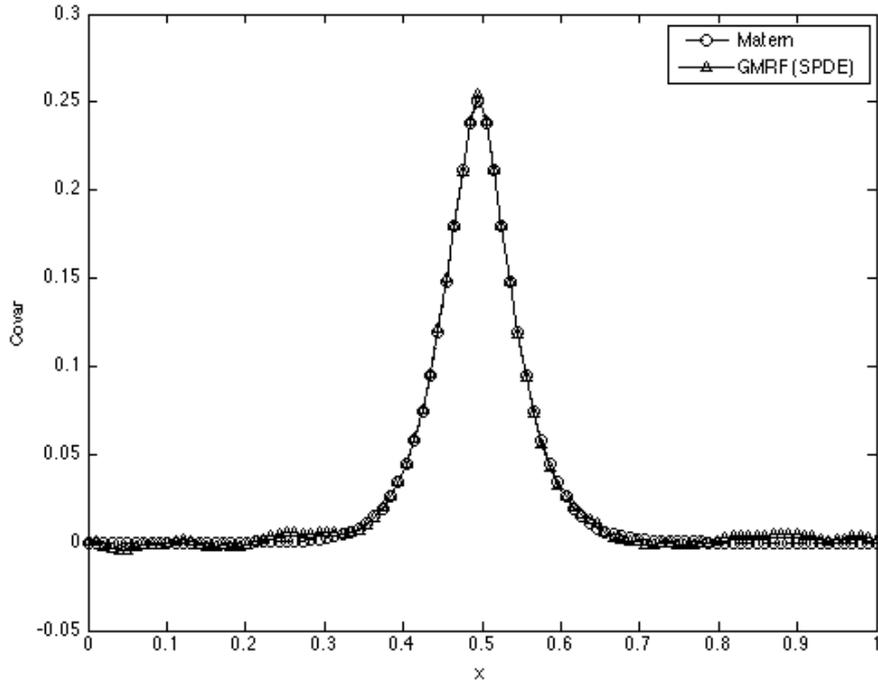


Figure 4.3: Covariance for the point, $x = 0.5$ for the 10000 samples realized, ($\lambda = 0.1$, $\sigma = 0.5$)

function using the SVD. The second case is done in 2D and follows the same procedure as the first case. The difference is that in the 2D case, Equation (1) is also solved, hence, some of the results will also serve to be discussed later in Section 4.4.1. Results from both the cases show that the GMRF approximation to the GF with a Matérn covariance can be used instead, and that the approximation can be close to the desired GF for computational purposes.

4.3.1 1D case

The 1D case is used to test the approximation of the GMRF to a Gaussian field using the exact Matérn covariance with KL expansion (implemented as SVD). As can be seen from Figures 4.3, and 4.4, the approximation of the GMRF is sufficient to reproduce the exact Matérn field.

Figure 4.3 shows the result of the covariance at the point $x = 0.5$ for the exact Matérn covariance (2), and the covariance generated by solving the 1D form of the SPDE (4) using 10000 realizations and 100 discrete spatial points. Figure 4.4 show the spatial statistical variables associated with the realizations between the solutions of the SPDE and the exact Matérn covariance field with KL expansion. The fluctuations in the expectation are due to the sampling error and are reduced as the number of samples are increased. The shift seen in the standard deviation (fig. 4.4b), is conjectured to be due to the GMRF approximation approach to the actual GF covariance function.

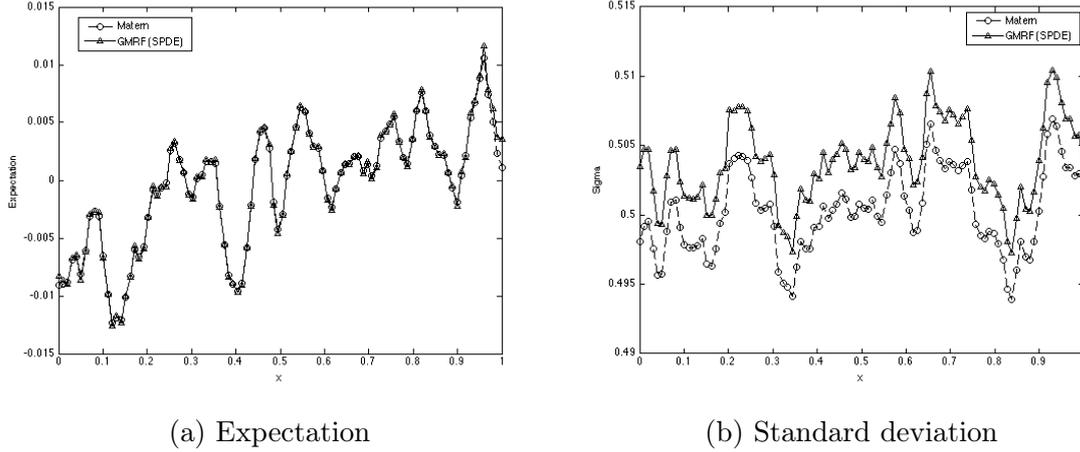


Figure 4.4: Expectation and standard deviation as a function of x for the 10000 samples realized, ($\lambda = 0.1$, $\sigma = 0.5$)

4.3.2 2D case

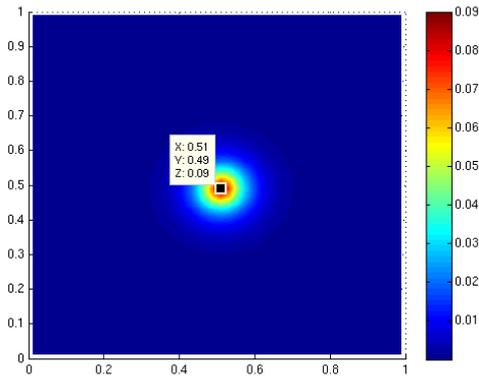
The 2D case is the actual solution to the problem as described in Section 2. The domain is discretized with $[50 \times 50]$ cells of equal sizes. As can be seen from Figure 4.5, the solution to the SPDE manages to also approximate the exact Matérn covariance. Figure 4.6 show a single realization of the GMRF, random coefficient, and solution fields. Figures 4.7 to 4.9 show the spatial statistical variables associated with the GMRF field, $a(\mathbf{x})$, random coefficient field, $e^{a(\mathbf{x})}$ and the solution field, $U(\mathbf{x})$, respectively, using 10000 realizations. We see from the variance of the solution field (fig. 4.9b) that the highest region of uncertainty corresponds to the largest spatial gradient in the solution field. So, we already see a potential in the use of uncertainty quantification in practical engineering design. Suppose this field represents a temperature field of an object, and we want to design an insulation system for the object, then, we would expect to be able to vary the safety factor in the spatial sense (i.e. increase in regions of high variance, and reduce in regions of low variance) to save insulation material cost.

4.4 MATLAB

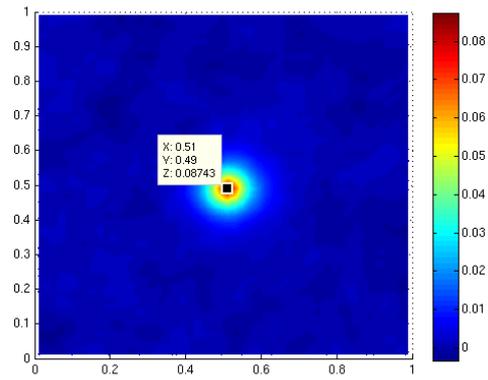
MATLAB codes were written as proof-of-concepts for the mathematical validation exercise and also for potential performance enhancement for the different algorithms such as comparisons between the Standard Monte Carlo (SLMC) and Multilevel Monte Carlo (MLMC) methods.

4.4.1 MATLAB - Standard Monte Carlo

The Standard Monte Carlo method results are shown in this section. Figure 4.10 show the statistical variables associated with the solution, Q_M as computed by the MATLAB implementation for varying grid sizes. The statistical parameters of the input GMRF field were kept constant with $\lambda = 0.1$, $\sigma = 0.3$. The sampling error, $\epsilon = 5E - 3$ is kept constant, i.e. the SLMC simulation is terminated once the sampling error is less or equal to $5E - 3$. As expected, the expectation, $\mathbb{E}[Q_M]$ converges as the grid size is increased. It corresponds

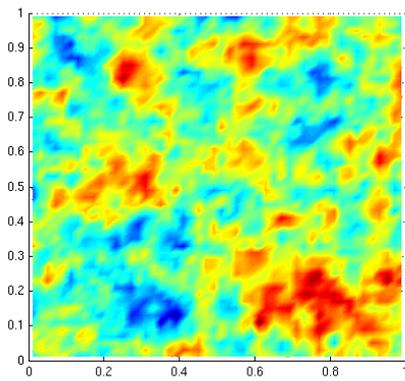


(a) Exact Matérn

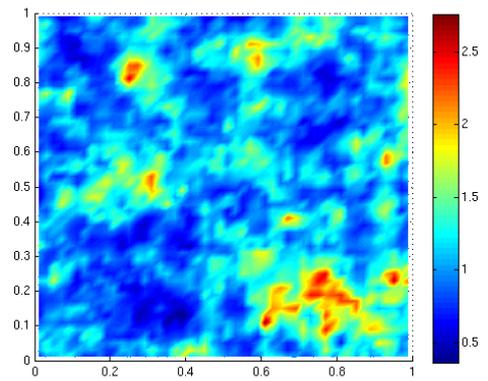


(b) GMRF

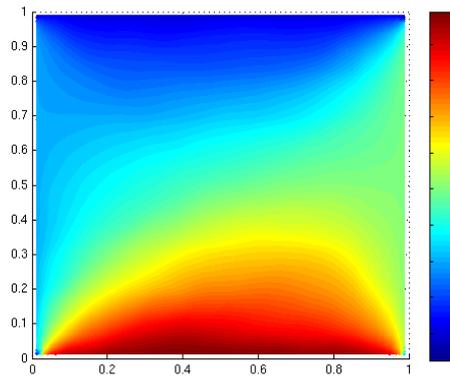
Figure 4.5: Covariance field for $\boldsymbol{x} = (0.51, 0.49)$ for the 10000 samples realized, ($\lambda = 0.1$, $\sigma = 0.3$)



(a) GMRF

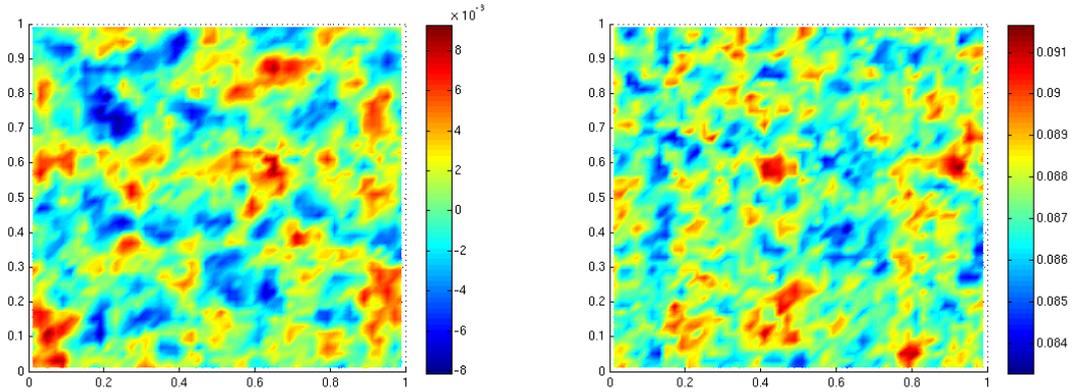


(b) Random coefficient



(c) Solution

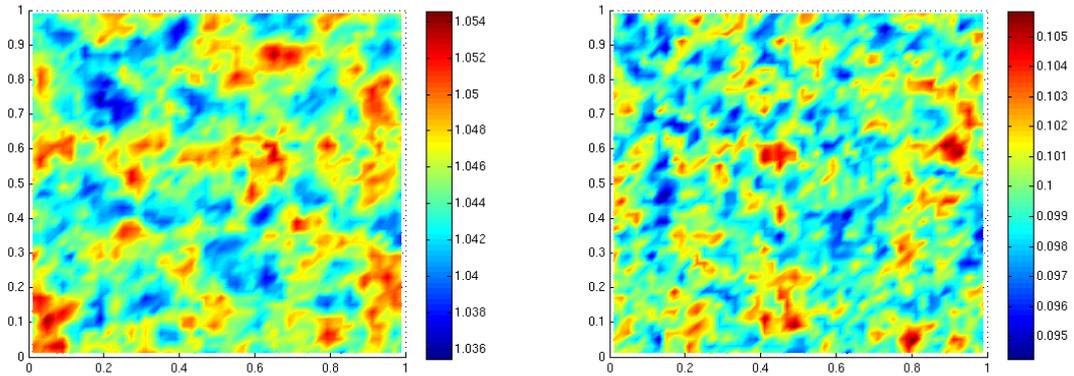
Figure 4.6: GMRF, $a(\boldsymbol{x})$, random coefficient, $e^{a(\boldsymbol{x})}$ and solution, $U(\boldsymbol{x})$ fields for a single realization



(a) Expectation

(b) Variance

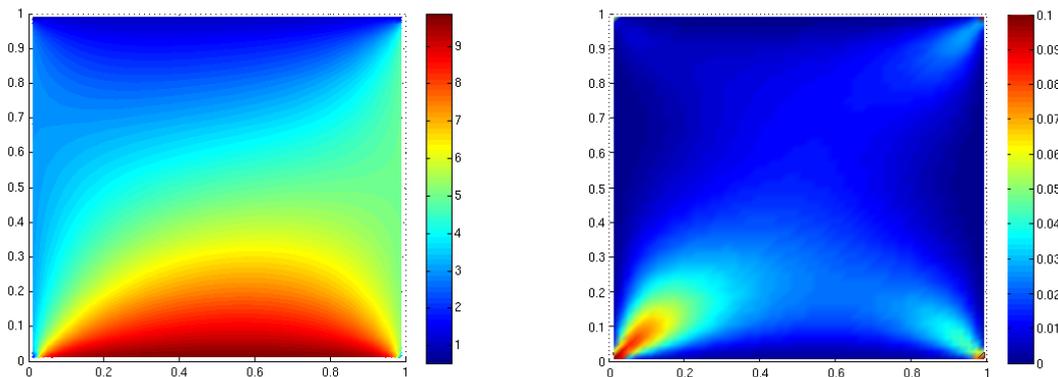
Figure 4.7: Expectation and variance of $a(\mathbf{x})$ as a function of \mathbf{x} for the 10000 samples realized



(a) Expectation

(b) Variance

Figure 4.8: Expectation and variance of $e^{a(\mathbf{x})}$ as a function of \mathbf{x} for the 10000 samples realized



(a) Expectation

(b) Variance

Figure 4.9: Expectation and variance of $U(\mathbf{x})$ as a function of \mathbf{x} for the 10000 samples realized

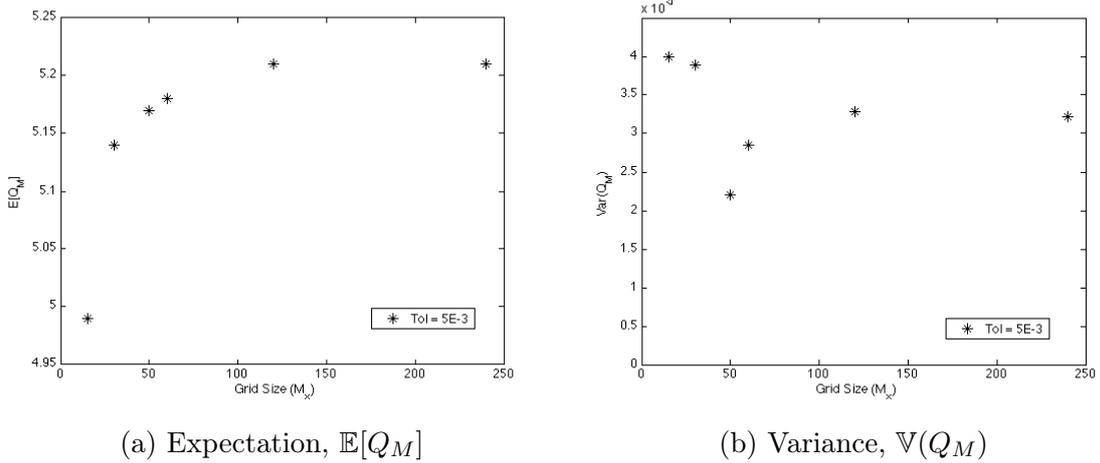


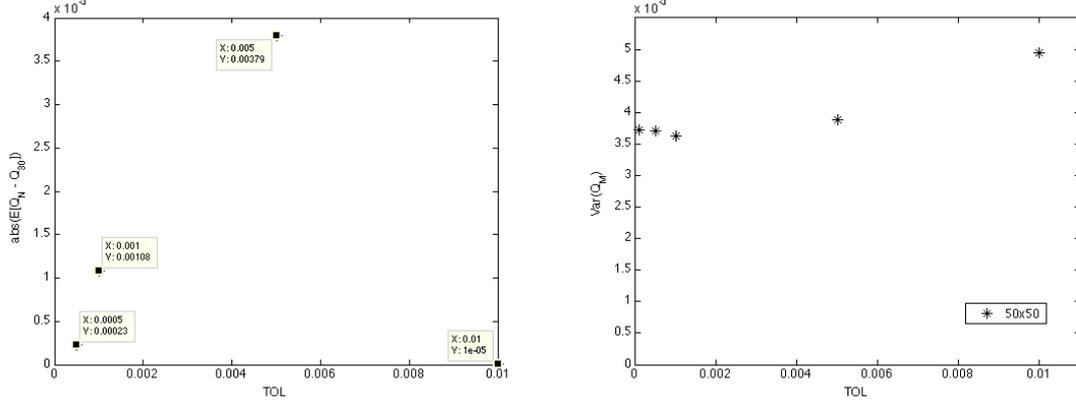
Figure 4.10: Statistics of Q_M as a function of grid size for the SLMC using $\lambda = 0.1$, $\sigma = 0.3$, sampling error, $\epsilon = 5E - 3$.

to one of the assumptions mentioned in Section 2.5. The variance, $\mathbb{V}(Q_M)$ does not show any noticeable trend when the grid size is varied, except that they are within the range, $[0.002, 0.004]$. This could be due to the definition of the sampling error term and subsequently the simulation termination condition. We know that the sampling error for any random variable in the Monte Carlo method is dependent on the variance of that random variable, hence, if we wanted to determine the actual uncertainty involved with the solution, then, we would need to determine the variance of the variance of Q_M , i.e. $\mathbb{V}(\mathbb{V}(Q_M))$. For this thesis, we will focus only on the expectation errors as implementations for the variance errors will be similar with more number of samples required.

Figure 4.11 show the statistical variables associated with the solution as computed by the MATLAB implementation for varying sampling errors. The grid size $[30 \times 30]$ used is coarser to obtain the solutions faster in the laptop. We also know from the grid convergence study that $\mathbb{E}[Q_{30}] = 5.138050$. Figure 4.11a show the variable, $|\mathbb{E}[Q_N - Q_{30}]|$ as a function of sampling error. Notice that there does not seem to be a trend. However, the objective is to show that the absolute difference between the expectation of Q_N at the end of the SLMC simulation is indeed close to the sampling error allowed. The variance is shown for completeness sake.

4.4.2 MATLAB - Multilevel Monte Carlo

As we have seen from previous sections that the sampling and discretization errors are independent of each other, we can use that knowledge to test whether the MLMC implementation is capable of reproducing the statistical information of Q_M in a similar manner as the SLMC. Figure 4.12 show the case plots for a test case using $\lambda = 0.1$, $\sigma = 0.3$, with finest grid size at $[240 \times 240]$, and sampling error, $\epsilon = 5E - 3$. They show that the implementation is indeed consistent with the theory as mentioned in Section 2.6. For the grid size of $[240 \times 240]$, $\mathbb{E}[Q_{240}] = 5.201268$. The expectation as calculated from the MLMC implementation is calculated to be, $\mathbb{E}[Q_{240}^{\text{MLMC}}] = 5.209588$. This gives an absolute error of $8.3E - 3$ which is close to the required sampling error of $5E - 3$. The variance is also calculated at 0.0034 which is within the range as calculated by the SLMC implementation. The result shows that the MLMC method does produce statistical data which are consistent with that of the SLMC



(a) Expectation, $|\mathbb{E}[Q_N - Q_{30}]|$

(b) Variance, $\mathbb{V}(Q_{30})$

Figure 4.11: Statistics of Q_{30} as a function of sampling error for the SLMC using $\lambda = 0.1$, $\sigma = 0.3$, grid size $[30 \times 30]$

method.

4.4.3 Performance Analysis

We have seen previously that the MLMC method is indeed consistent with the SLMC method. This section now discusses the performance enhancements that can be obtained by the MLMC method. The speedup, S , is defined by Equation (25).

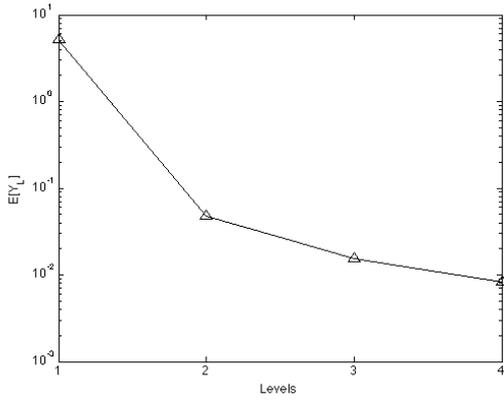
$$S = \frac{T_{\text{SLMC}}}{T_{\text{MLMC}}} \quad (25)$$

where T_{SLMC} and T_{MLMC} are the time-to-solution for the SLMC and MLMC methods respectively.

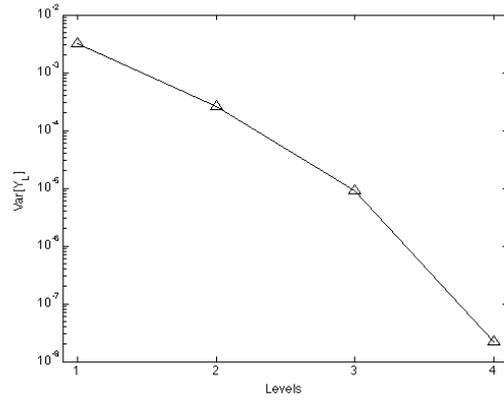
For all the cases, the statistical parameters of the GMRF field are $\lambda = 0.1$, and $\sigma = 0.3$. Figure 4.13a show the speedup obtained when we vary the grid size while keeping the number of grid levels constant at 3. It shows that the speedup increases with grid size and this shows the advantage of MLMC over SLMC for real-world applications. Figure 4.13b show the speedup obtained when we vary the number of levels and keep the grid size constant at $[240 \times 240]$. It shows that there is indeed an optimum number of levels which gives a maximum speedup for a particular grid size. For this case, it gives a maximum speedup of approximately 24 using 4 grid levels. With these results, it shows the potential of MLMC to improve the time-to-solution for Monte Carlo methods.

4.5 PETSc - Standard Monte Carlo

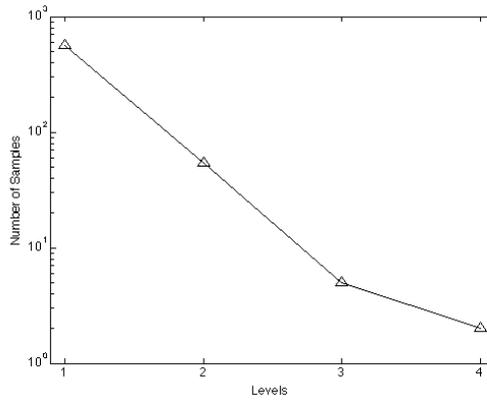
Due to time constraints, only the standard Monte Carlo method was implemented in PETSc. The robustness of the solvers in PETSc are explored with respect to the different statistical parameters. Then, studies were done to determine the speed of the time to solution with different preconditioners. The PETSc cases are run entirely in the cluster of the Informatics Department (LSS) at the University of Erlangen-Nürnberg [7].



(a) Expectation, $\mathbb{E}[Y_l] = \mathbb{E}[Q_l - Q_{l-1}]$

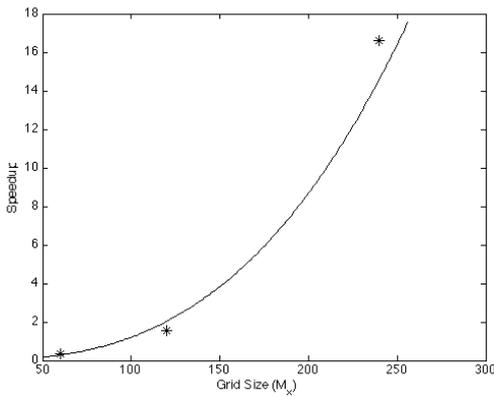


(b) Variance, $\mathbb{V}(Y_l) = \mathbb{V}(Q_l - Q_{l-1})$

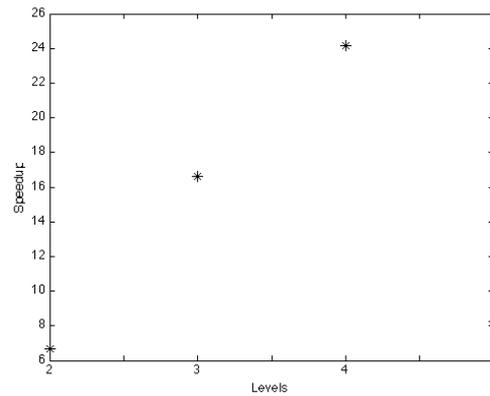


(c) Number of samples

Figure 4.12: MLMC sample case plots for $\lambda = 0.1$, $\sigma = 0.3$, Finest grid size is at $[240 \times 240]$, sampling error, $\epsilon = 5E - 3$.



(a) Speedup as a function of grid size



(b) Speedup as a function of levels

Figure 4.13: Speedups for the MLMC using $\lambda = 0.1$, $\sigma = 0.3$, finest grid size $[240 \times 240]$, sampling error, $\epsilon = 5E - 3$.

4.5.1 Solvers

In order to perform any Monte Carlo simulations with SPDEs, the solver used must be robust enough so that a solution can be found in each sample. This PETSc implementation investigates the robustness of the BoomerAMG solver from the hypre package (which is conveniently interfaced with PETSc) [12] without any preconditioner, as well as the flexible GMRES (fgmres) solver native to PETSc [22, 2] with the BoomerAMG as preconditioner. The solver robustness study was done by varying the statistical parameters of the covariance function (λ and σ) while keeping the grid size at $[500 \times 500]$ and solver relative tolerance at $1.0\text{E}-6$.

The settings of the solvers are given in Table 4.1.

Table 4.1: Solver settings for robustness study

Solver settings	fgmres	BoomerAMG
Preconditioner	BoomerAMG (1 V-Cycle)	None
AMG relaxation solver	Symmetric SOR (up-down) Gaussian Elimination (coarsest)	Symmetric SOR (up-down) Gaussian Elimination (coarsest)
Maximum iterations	1000	1000 (V-Cycles)

For the solvers study, 2000 samples were realized for each parameter and for each sample, the number of iterations were recorded. The average, maximum, and minimum iterations across all the samples were calculated. For the purpose of brevity, only the maximum number of iterations are shown in this thesis as the average and minimum iterations also follow the same trends.

Table 4.2 shows that as we increase the correlation length, λ , the number of iterations decreases, and as we increase the standard deviation, σ , of the GMRF, then, the number of iterations increases. There is a region where NA is denoted and this means that the solver does converge, however, the statistical results obtained does not seem to be converging within 2000 samples. This means that $\mathbb{E}[Q_{500}]$ still has not converged to within $tol = 0.1$. Hence, further study must be done to increase the number of samples to determine if the statistics are reasonable. However, it is determined that the fgmres solver with the BoomerAMG preconditioner is a robust enough solver for the applications in the thesis.

Table 4.3 shows the same trend as the fgmres solver with the difference that it takes a bit more number of iterations for each parameter. Also, DNC in the table denotes that the solver did not converge within the maximum iterations specified in Table 4.1. This actually shows that the solver does have trouble in terms of robustness to obtain solutions which have GMRFs with $\sigma \geq (2.6, 2.8]$.

In order to further understand why the solvers fail to converge, and to also help explain the behaviour of the number of iterations as a function of the statistical parameters λ and σ , we take a look at one sample of the solution and the corresponding GMRF. Figures 4.14 to 4.19 show the realizations for different parameters. They show that as one increases the correlation length, λ , the GMRF field input is less noisy. Therefore the number of iterations to obtain the solution would decrease as one increases λ . As for the increase in iteration with standard deviation, σ , this is easily explained since the variations between the neighbouring cells in the GMRF is higher, so, this behaves closer to a highly irregular exponential coefficient

field for Eq. (1). This is proven when we view the sample (fig. 4.19), from which the solver has problems in the statistical convergence.

Table 4.2: Maximum number of iterations for solving Eq. (1) with fgmres+boomeramg preconditioner - [NA = stats problem]

λ	σ											
	0.1	0.4	0.8	1.1	1.8	2.8	3.8	4.8	5.8	6.8	7.8	9.8
0.1	8	9	9	10	12	13	14	14	14	NA	NA	NA
0.15	8	8	9	9	11	13	13	13	14	14	NA	NA
0.2	8	8	9	9	10	12	13	13	13	13	14	NA
0.25	8	8	9	9	10	11	12	13	13	13	13	13
0.3	8	8	9	9	10	11	11	12	12	13	13	13
0.35	8	8	9	9	10	10	11	12	12	12	12	13
0.4	8	8	9	9	9	10	11	12	12	12	12	13

Table 4.3: Maximum number of iterations for solving Eq. (1) with boomeramg (symmetric SOR relaxation solver) - [DNC = did not converge, NA = stats problem]

λ	σ											
	0.1	0.4	0.8	1.1	1.8	2.8	3.8	4.8	5.8	6.8	7.8	9.8
0.1	11	12	15	19	25	DNC	DNC	DNC	DNC	DNC	DNC	NA
0.15	10	12	15	17	22	DNC	DNC	DNC	DNC	DNC	DNC	NA
0.2	10	12	14	16	20	DNC	DNC	DNC	DNC	DNC	DNC	NA
0.25	10	12	13	15	18	DNC	DNC	DNC	DNC	DNC	DNC	NA
0.3	10	11	13	14	17	DNC						
0.35	10	11	13	15	17	22	DNC	DNC	DNC	DNC	DNC	DNC
0.4	10	11	13	13	17	22	DNC	DNC	DNC	DNC	DNC	DNC

4.5.2 Preconditioners and KSP Solvers

From Section 4.5.1, it can be seen that the Krylov subspace (KSP) type solvers are more robust for the application. Hence, we now test for the performance in using different preconditioners as well as different KSP solvers implemented in PETSc [2]. There are so many variants of preconditioners and KSP solvers and the speed depends on the problem being solved. Hence, again, only a representative few are being tested. The list of tested variants are shown in Table 4.4. The reason these solvers were chosen is that they are each representative of the different KSP type solvers, namely, BiCG, GMRES, and CGNR [25]. The solver SYMMLQ is a hybrid between MINRES and LSQR [16]. The default statistical variables are used, with [500×500] grid size, and 150 samples. The SOR preconditioner is set with $\omega = 1.8$ and 200 iterations.

As can be seen from Table 4.5, for the problem in the thesis, the BoomerAMG yields the fastest results across all solvers except Conjugate Gradient Normal Residual (CGNR). The CGNR solver does show a converging trend, however, the total time-to-solution to converge exceeded the maximum compute time (8 hours) allowed in the LSS cluster.

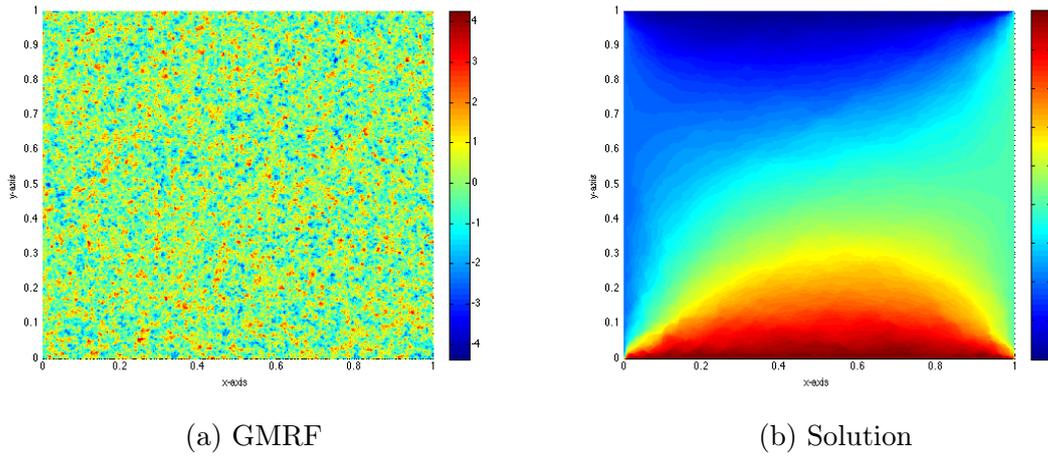


Figure 4.14: GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.01$, $\sigma = 1$

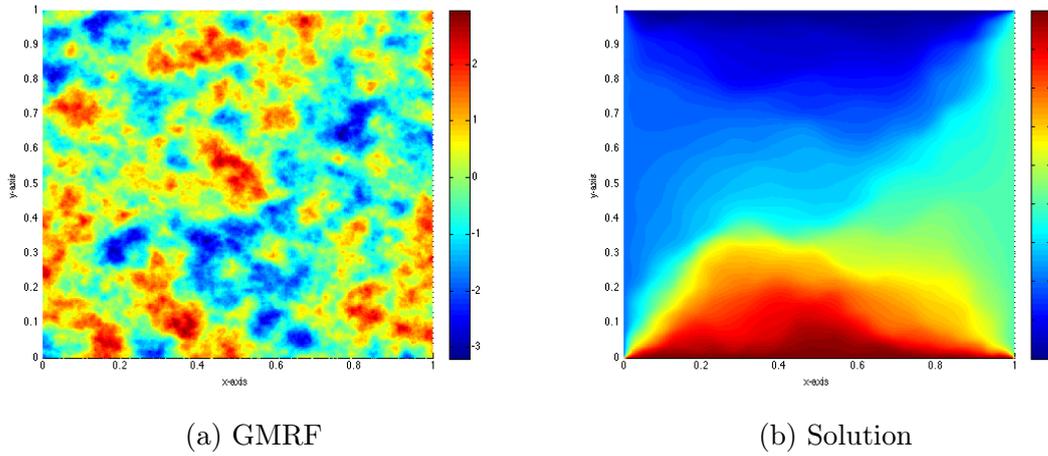


Figure 4.15: GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.1$, $\sigma = 1$

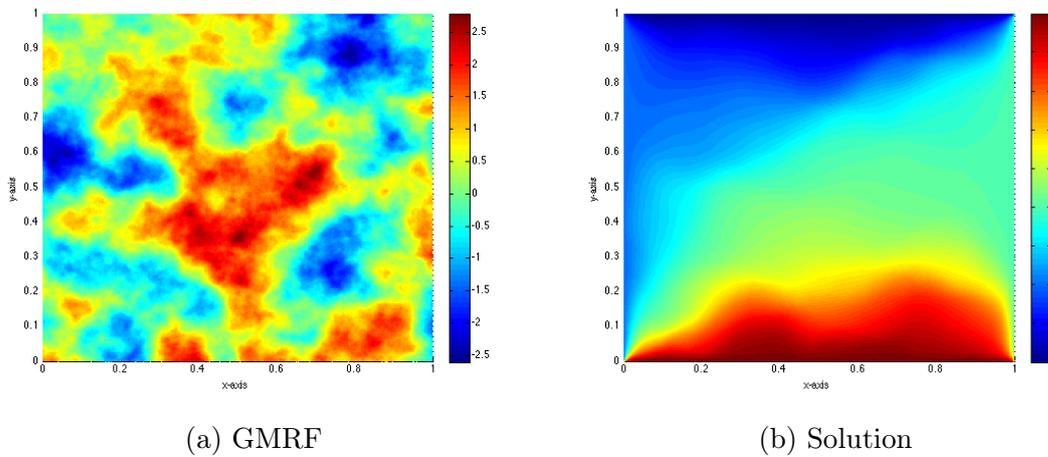
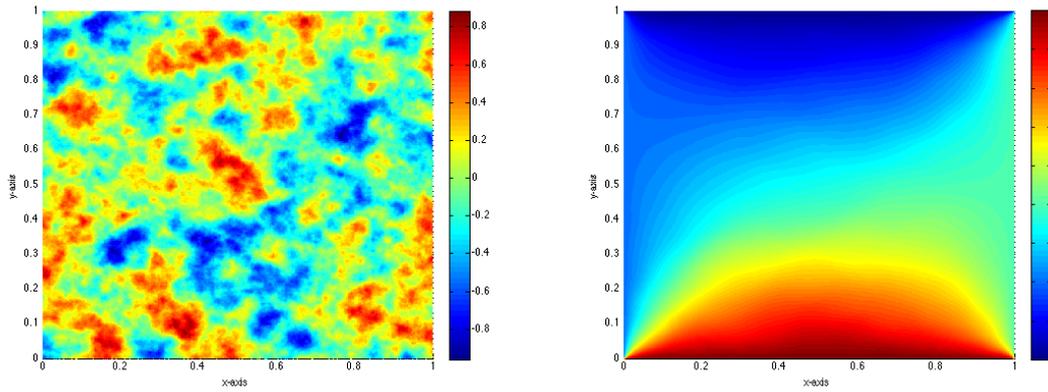


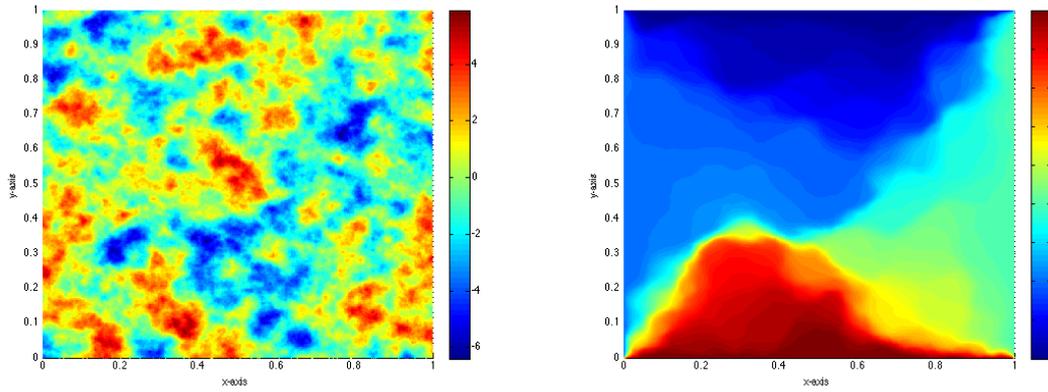
Figure 4.16: GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.25$, $\sigma = 1$



(a) GMRF

(b) Solution

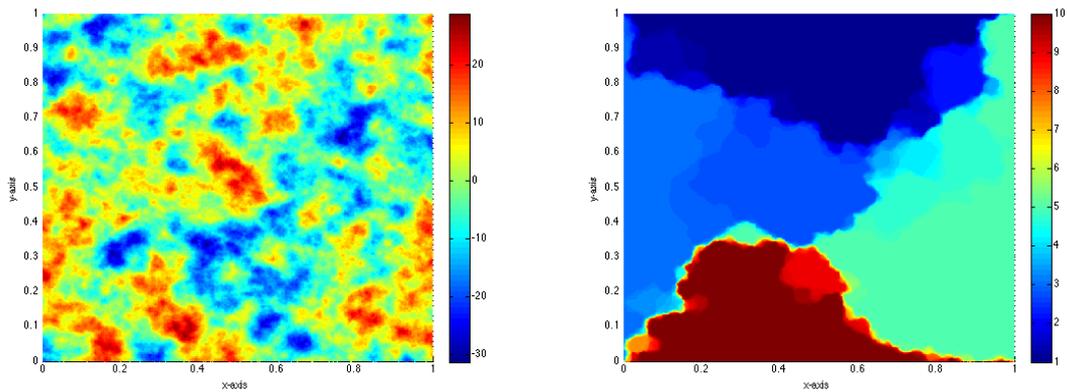
Figure 4.17: GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.1$, $\sigma = 0.3$



(a) GMRF

(b) Solution

Figure 4.18: GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.1$, $\sigma = 2$



(a) GMRF

(b) Solution

Figure 4.19: GMRF, $a(\mathbf{x})$ and solution, $U(\mathbf{x})$ fields for a single realization, $\lambda = 0.1$, $\sigma = 10$

Table 4.4: Preconditioner and Solvers

Solver	Preconditioner
fgmres	BoomerAMG
bi-cg stab	icc(0)
symmlq	ilu(0)
cgnr	sor

Table 4.5: Wall clock times (seconds) to compute 150 samples, with different KSP solvers and preconditioners using 32 processors

Solver	Preconditioner			
	BoomerAMG	icc(0)	ilu(0)	sor
fgmres	25.42	950.61	944.43	461.41
bi-cg stab	31.70	204.05	188.87	710.23
symmlq	34.21	1271.4	1632.6	604.59
cgnr	> 8 h	3617	3654	1702.0

With the results from this study, the remaining studies were performed using the fgmres solver with BoomerAMG preconditioner.

4.6 PETSc - Performance Analysis

The performance analysis of the PETSc code starts with the comparison between the parallelization strategies for the SLMC method. We then determine the performance enhancements that can be potentially obtained from using the GMRF approximation method over the standard Cholesky decomposition to sample the GF field. After that, the compositions of the total wall clock time that is spent to solve, setup, and communicate for the GMRF approximation, Eq. (4) and for the variable Poisson problem, Eq. (1) between processors are shown and discussed. All of the speedups and efficiency variables are measured against the serial code.

4.6.1 Performance of different parallelization strategies

We start the performance analysis by studying the various standard Monte Carlo parallelization strategies that are implemented for this thesis. We define the speedup, and efficiency as Equations (26) and (27), respectively [17].

$$\text{Speedup} = \frac{T_{\text{ser}}}{T_{\text{par}}} \quad (26)$$

$$\text{Efficiency} = \frac{\text{Speedup}}{N_p} \quad (27)$$

where T_{ser} and T_{par} are the total time-to-solution of the serial and parallel implementation, respectively. N_p denotes the number of processors for that parallel run.

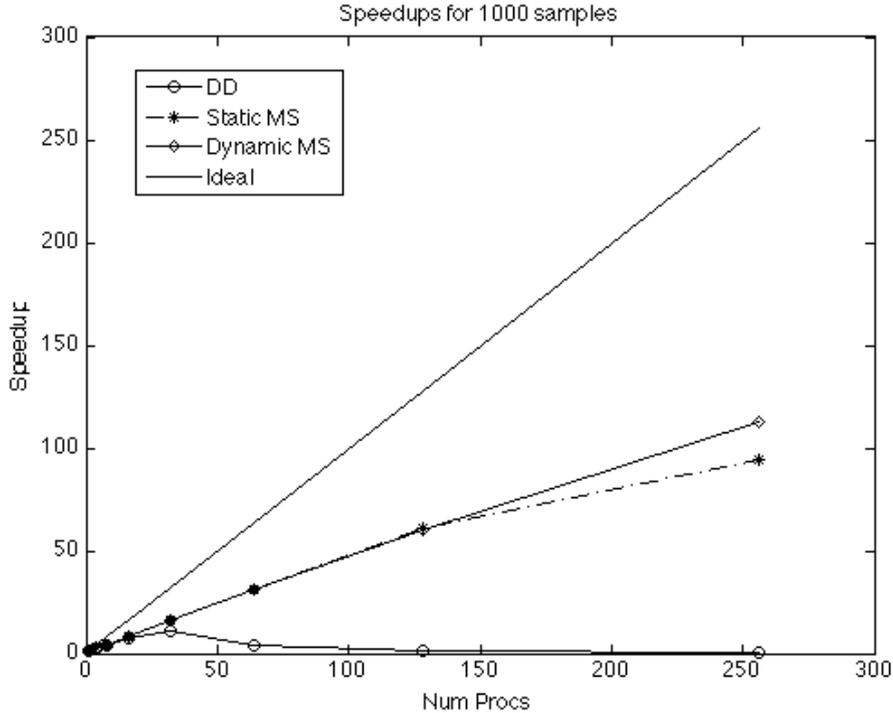


Figure 4.20: Speedups as a function of number of processors for different parallelization strategies, 1000 samples, $[500 \times 500]$, 1 processor per sample

The first case we study is to determine the speedups and efficiencies that we obtain by varying the number of processors with 1000 samples, $[500 \times 500]$ grid size, and 1 processor per sample. Figure 4.20 shows the speedup while Figure 4.21 shows the efficiency.

They both show that applying the Domain Decomposition method alone is insufficient for large number of processors and this is not unexpected since communication overheads will tend to dominate for a small problem size. However, both the Static and Dynamic master-slave strategies does indicate that they are potentially scalable up to at least 256 processors. This is also not unexpected since the Monte Carlo methods are an example of embarrassingly parallel applications. What is most unexpected is that in all of the efficiency curves (fig. 4.21 to fig. 4.25), the efficiency barely reaches 60%. This seems rather discouraging at first but the reason for that is discussed elaborately in Section 4.6.3.

We now turn to changing the number of samples while keeping the other variables constant. Figure 4.22 show the efficiency when we vary the number of samples for the different parallelization strategies keeping a grid size of $[500 \times 500]$. 32 processors were used with 1 processor per sample. It shows that for both strategies, the efficiency fluctuates within a range of 40-60%. It also show that the both the master-slave strategies do not differ from each other that much in terms of parallel efficiency.

The same trend also happens (Figure 4.23 to 4.25) when we mix domain decomposition strategy with the two master-slave implementations. We change to a 2 processor per sample configuration and run the above cases. The efficiencies again show a fluctuation of approximately 40-60% as seen from the 1 processor per sample previously.

Therefore, we fix the parallelization strategy to that of the Dynamic MS due to the advantages of having on-the-fly sampling tolerance termination as well as dynamic load balancing

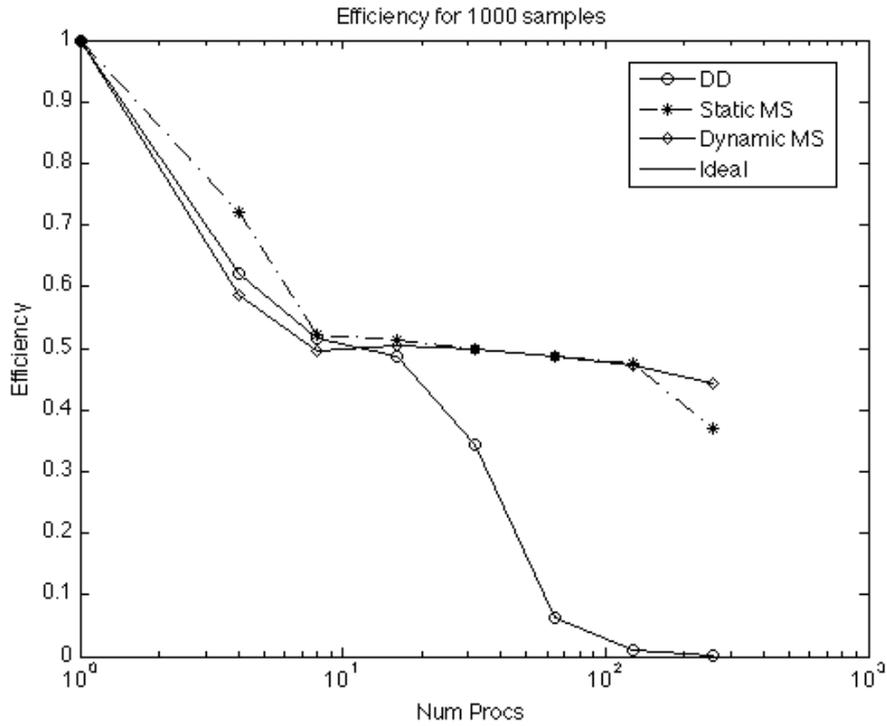


Figure 4.21: Efficiency as a function of number of processors for different parallelization strategies, 1000 samples, $[500 \times 500]$, 1 processor per sample

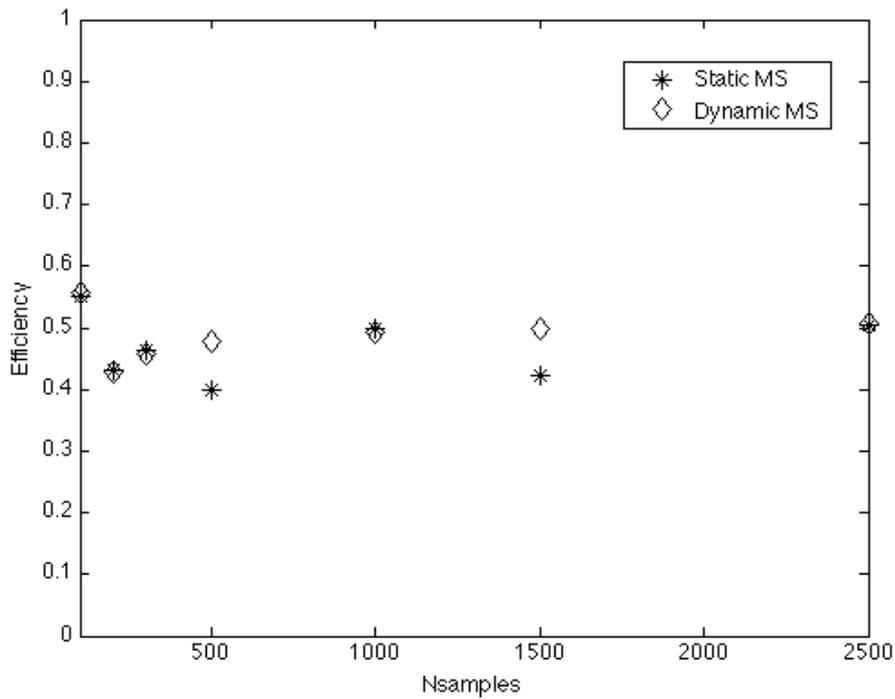


Figure 4.22: Efficiency as a function of number of samples for different parallelization strategies, 32 processors, $[500 \times 500]$, 1 processor per sample

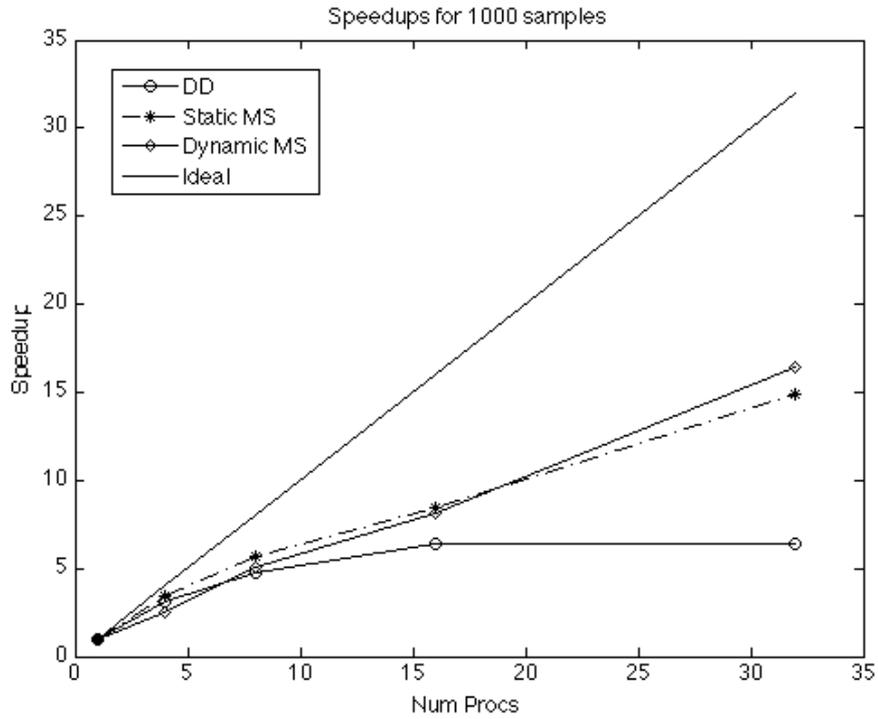


Figure 4.23: Speedups as a function of number of processors for different parallelization strategies, 1000 samples, $[500 \times 500]$, 2 processors per sample

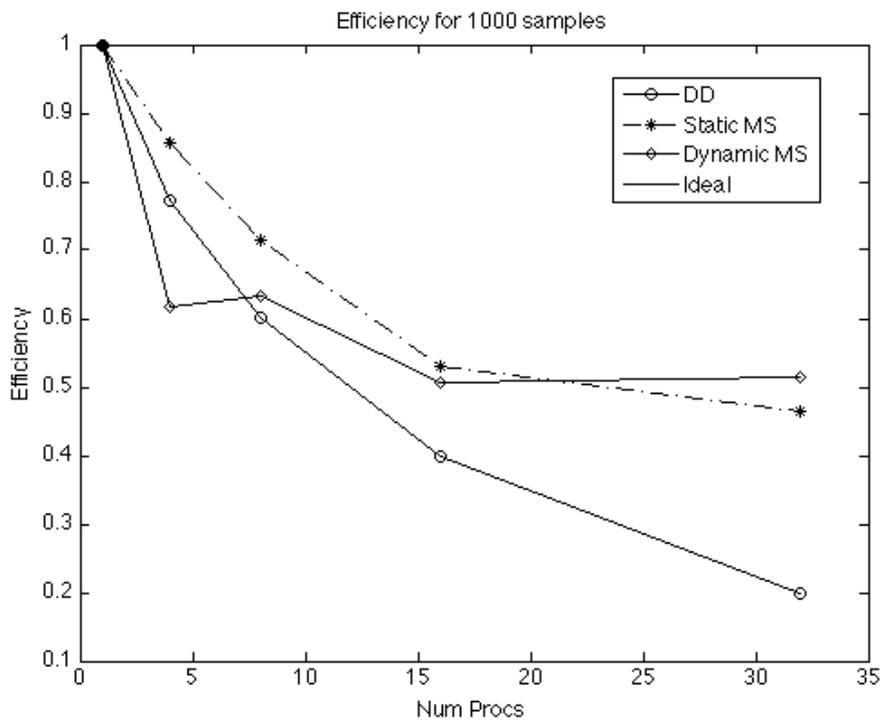


Figure 4.24: Efficiency as a function of number of processors for different parallelization strategies, 1000 samples, $[500 \times 500]$, 2 processors per sample

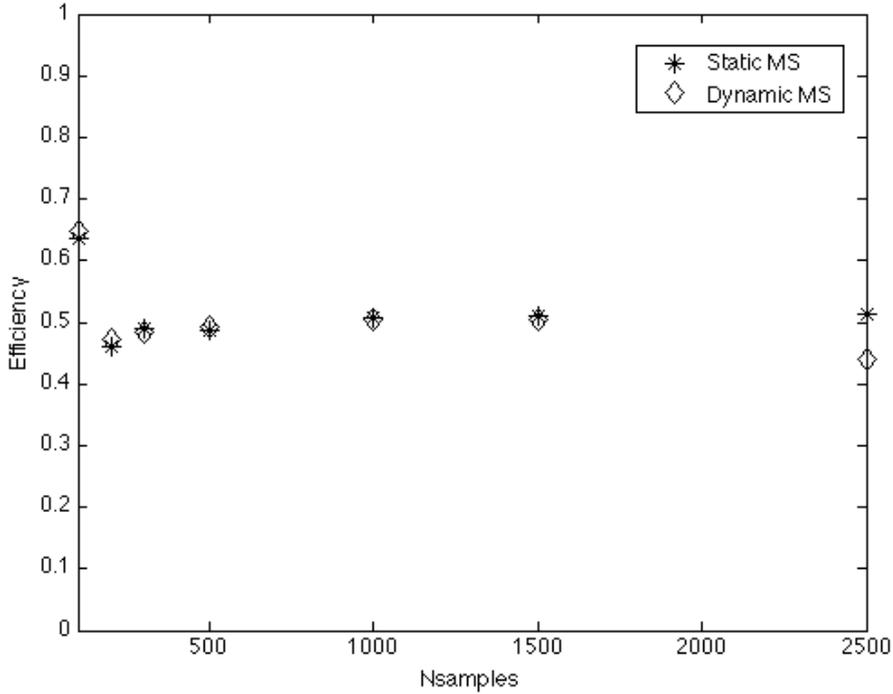


Figure 4.25: Efficiency as a function of number of samples for different parallelization strategies, 32 processors, $[500 \times 500]$, 2 processor per sample

for larger number of processes.

Figures 4.26 and 4.27 show the efficiency curve as a function of number of processors, and number of samples, respectively for different grid sizes. It does show that efficiency is consistently higher for the larger grid size and can be explained by the possibility that the larger grid size makes the serial code to be slower as the socket memory cache becomes full for the larger grid size.

4.6.2 Performance of GMRF compared to Cholesky decomposition

We have already validated that GMRF approximation can be used to generate the GF with the desired Matérn covariance field. Now we would determine the performance enhancement of using a GMRF over the standard Cholesky decomposition in solving Equation (1). In terms of domain size, the GMRF approximation already is advantageous since the Cholesky decomposition program fails to run for grid sizes which are larger than $[100 \times 100]$. The speedup of the GMRF approximation approach is shown in Figure 4.28. The speedup, defined in Equation (28), is measured with respect to the dynamic master-slave implementation using 1 processor per sample.

$$\text{Speedup} = \frac{T_{\text{Chol}}}{T_{\text{GMRF}}} \quad (28)$$

where, T_{Chol} and T_{GMRF} are the wall clock times for the Cholesky and GMRF implementation, respectively. Figure 4.28 shows that the GMRF approximation is consistently faster than using Cholesky decomposition even for large samples. However, the speedup will decrease

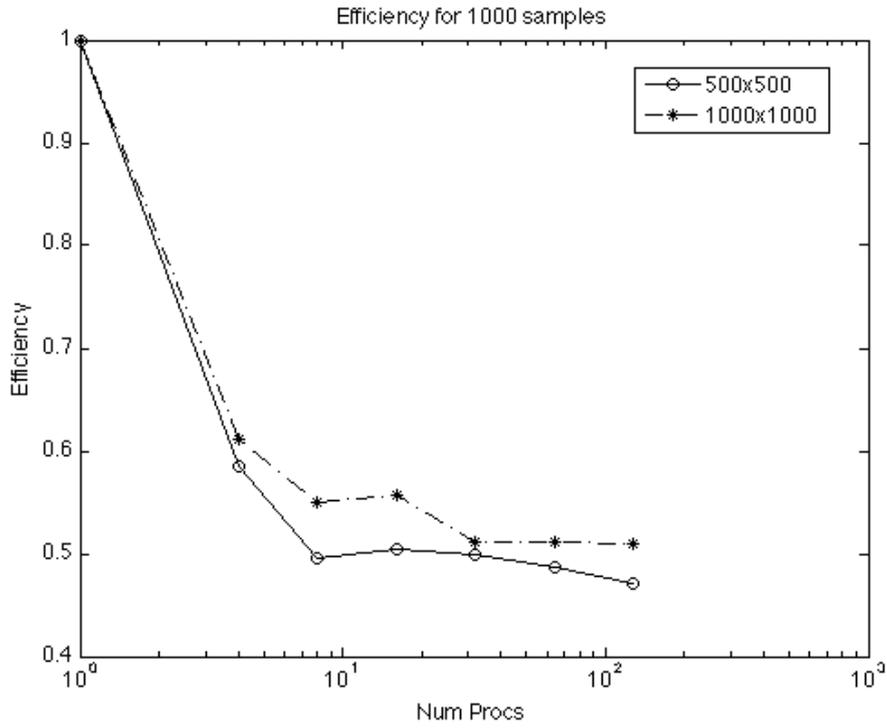


Figure 4.26: Efficiency as a function of number of processors for different grid sizes, 1000 samples, Dynamic MS, 1 processor per sample

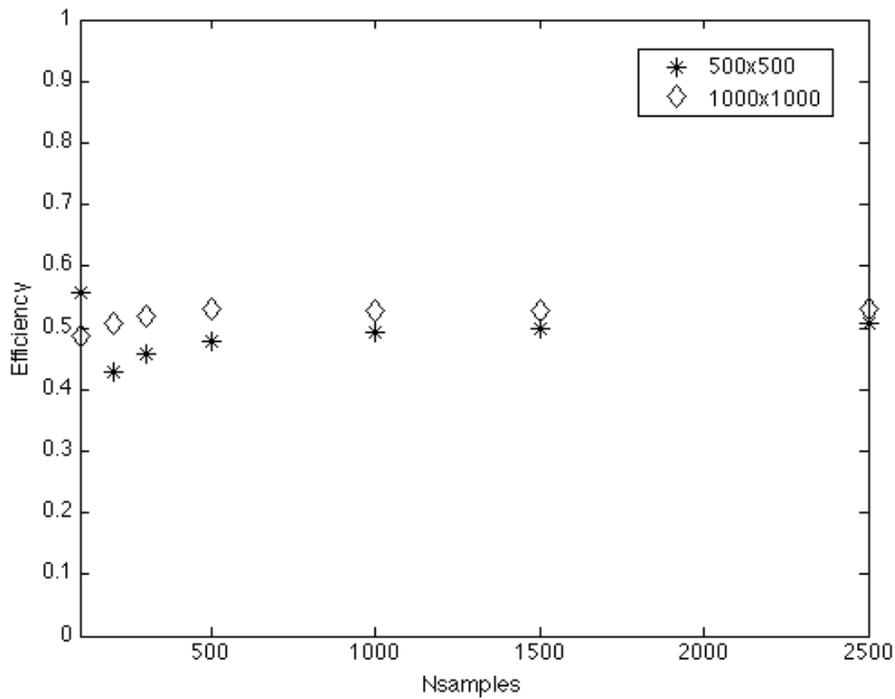


Figure 4.27: Efficiency as a function of number of samples for different grid sizes, 32 processors, Dynamic MS, 1 processor per sample

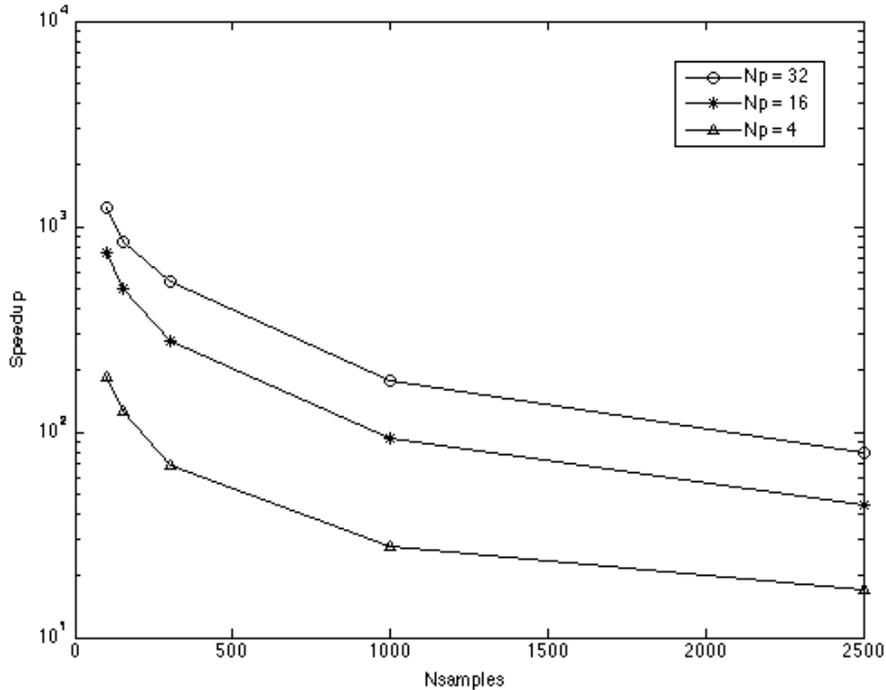


Figure 4.28: Speedup of the GMRF approximation over the Cholesky decomposition approach as a function of number of samples for different number of processors

asymptotically. This can be explained by the initial cost of setting up the covariance matrix and performing the Cholesky decomposition of the matrix.

To further prove the claim, Figures 4.29 to 4.31 show the composition of the wall clock time for the Cholesky decomposition implementation. They show clearly that the largest time is taken by the decomposition step, and as the number of samples increases, the Solve step will dominate but this will only be true for a very large number of samples. We also see an increase in speedup for higher processor numbers. This could be due to the parallelization of only the covariance matrix setup and solution steps in the Cholesky decomposition implementation. This is because the PETSc library does not have a parallel Cholesky decomposition implementation.

4.6.3 Effect of CPU socket utilization

As we have also seen from Section 4.6.1, the efficiency seems to range from 40-60% for the Master-Slave parallelization strategies. This can be explained by the effects of the CPU socket utilization. Since the performance analysis is based on the speed of the serial code, we are utilizing only 1 core in an 8-core socket with a small problem size. However, when we run using the parallel codes, the socket cache usage becomes full, and thus the computational speed for each core is reduced by half. This could also be due to the snoop filter in the Intel Xeon processor as explained by the roofline model proposed by Williams et al [28].

The study to determine the reason for the efficiency drop uses a [1000×1000] grid and the Dynamic master-slave implementation. We bind the processes to each physical core using the OpenMPI's command line switch "--bind-to-core" with two different processors per socket

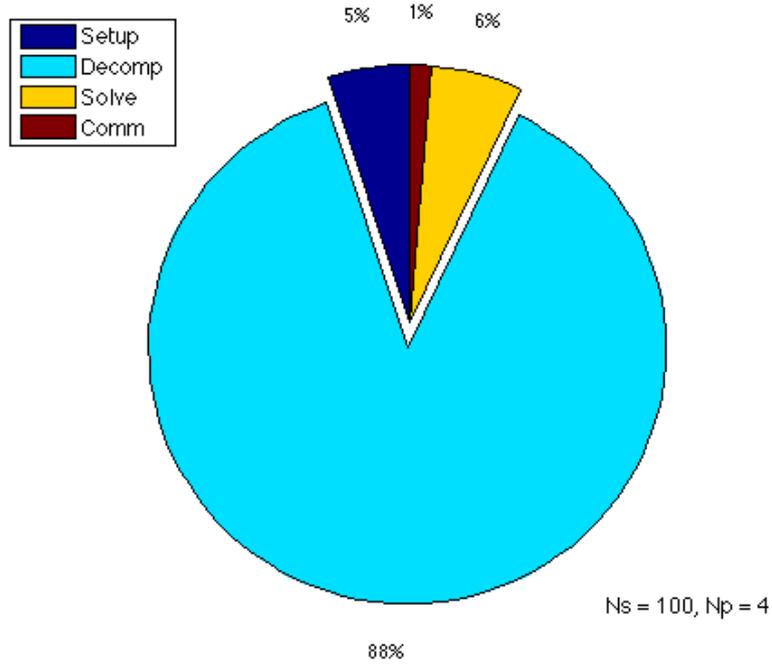


Figure 4.29: Percentage of total wall clock time for Cholesky Decomposition (4 processors, 100x100 grid size, 100 samples)

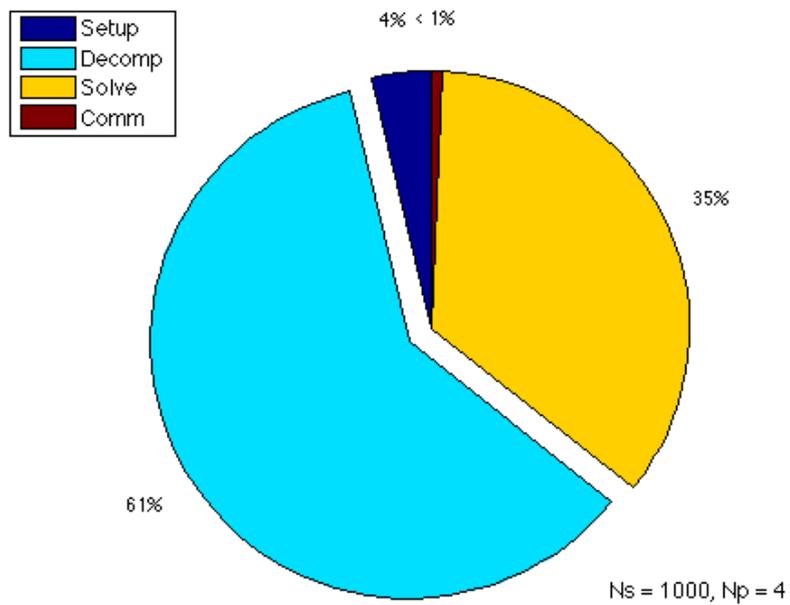


Figure 4.30: Percentage of total wall clock time for Cholesky Decomposition (4 processors, 100x100 grid size, 1000 samples)

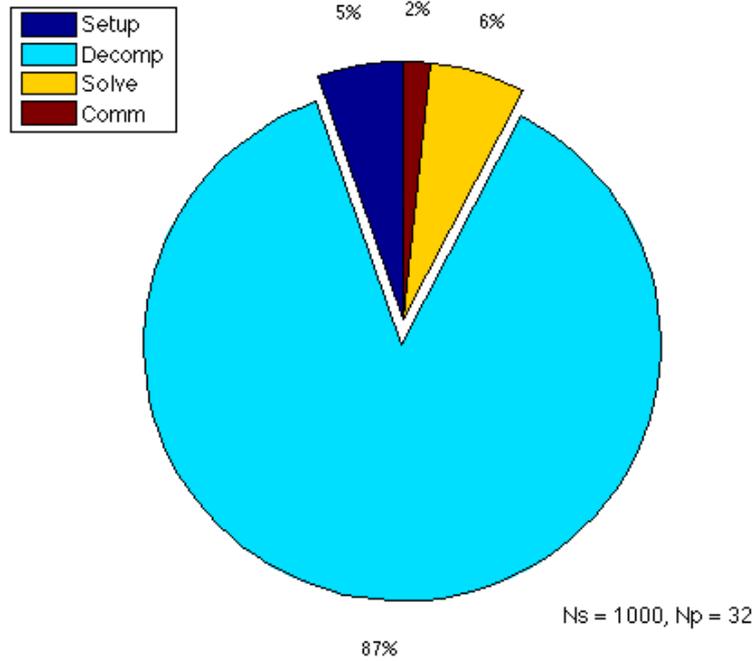


Figure 4.31: Percentage of total wall clock time for Cholesky Decomposition (32 processors, 100x100 grid size, 1000 samples)

values of "-npsocket". Figure 4.32 show the efficiency as we vary the number of processors used in the cluster for different npsocket values. We kept the number of samples constant at 2500. The result show that the Dynamic master-slave implementation does indeed achieve efficiencies which are close to 90% as expected when we do not fully utilize the CPU sockets in each node. Figure 4.33 show the efficiency when we vary the number of samples for different npsocket values. Again, it shows that for the definition of efficiency in this thesis, the cases in which we do not fully utilize the CPU sockets are shown to be more efficient. However, this cannot be the only metric of efficiency as we are not fully utilizing the entire CPU socket. Perhaps a better measure of efficiency for scalability studies would be to compare the timings against that obtained from an entire node.

This argument can in fact be clarified even more when we have a look at logged runs with the MPE library [5]. Two simple cases ($[500 \times 500]$ grid size, 100 samples) are run with balanced and unbalanced processor assigns. Balanced means the processors are evenly distributed across the available sockets, with the processor bindings shown in Figure 4.35, and Figure 4.36, respectively. We visualize the behaviour of the programs by linking the MPE library during compilation and using Jumpshot to visualize the log files. The color legend for the Jumpshot tool is shown in Figure 4.34. The figures which show any Jumpshot analysis will always refer to this color legend. Figures 4.37 and 4.38 show the entire behaviour of the program in Jumpshot for the balanced and unbalanced cases, respectively. It clearly shows that process ranks 8 and 9 for the unbalanced case is calculating approximately twice the number of samples as that of the other ranks. They were bound to an unfilled socket (fig 4.36). Figures 4.39a and 4.39b show the number of samples calculated for each rank. As expected they show that ranks 8 and 9 complete almost twice as many samples as other ranks

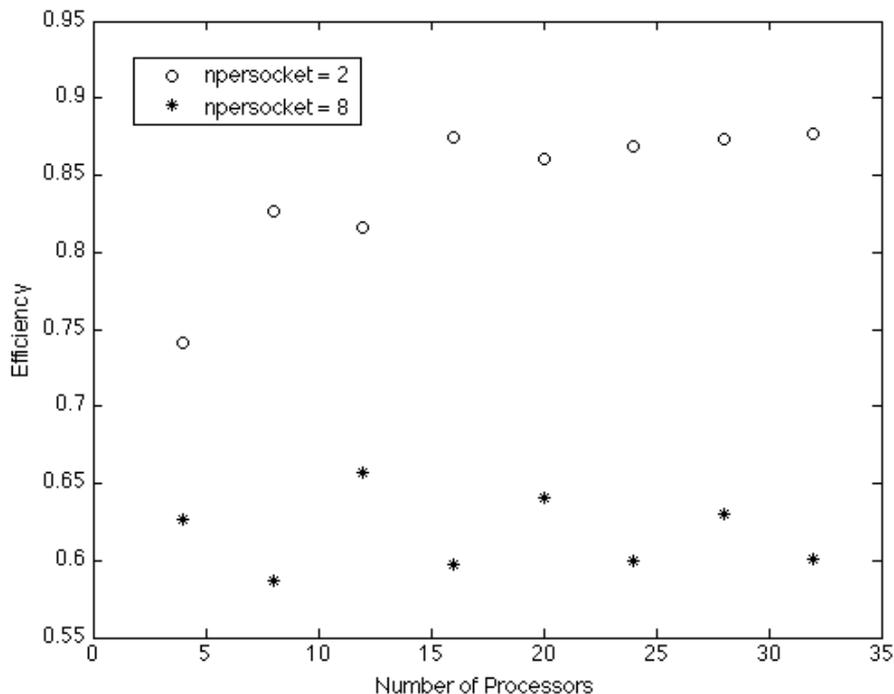


Figure 4.32: Efficiency as a function of number of processors for different npersocket values using 2500 samples, $[1000 \times 1000]$ grid and Dynamic master-slave

for the unbalanced case.

4.6.4 Miscellaneous analysis of the GMRF and Variational Poisson program

We have seen previously the composition of time for the Cholesky decomposition implementation in Section 4.6.2. We now examine the details of the time-to-solution for the GMRF approach. Figures 4.40 and 4.41 show the composition of the total time-to-solution for the GMRF implementation for 4 processors and 64 processors, respectively. The clock times are obtained for solving the system using $[500 \times 500]$ grid size, 1000 samples, and the Dynamic master-slave strategy. It confirms the findings obtained from the previous sections (4.6.1 & 4.6.3) that at least 90% of the wall clock time is being spent to solve the two equations.

The figures also show the potential scalability of the PETSc code since the percentage time taken by the solve steps do not change very significantly across the number of processors.

We next show the entire workings of the program. Figure 4.42 shows the Jumpshot visualization of the program for 4 processors per sample. It shows the working implementation of the Domain decomposition mixed with the Dynamic master-slave. The lines show the message sending and receiving between the global root process and the designated root processes for each sample.

4.7 ExaStencils - Standard Monte Carlo

Some preliminary results and performance data are discussed in this section. As the ExaStencils concept is very new, some experiences as an end user will also be mentioned.

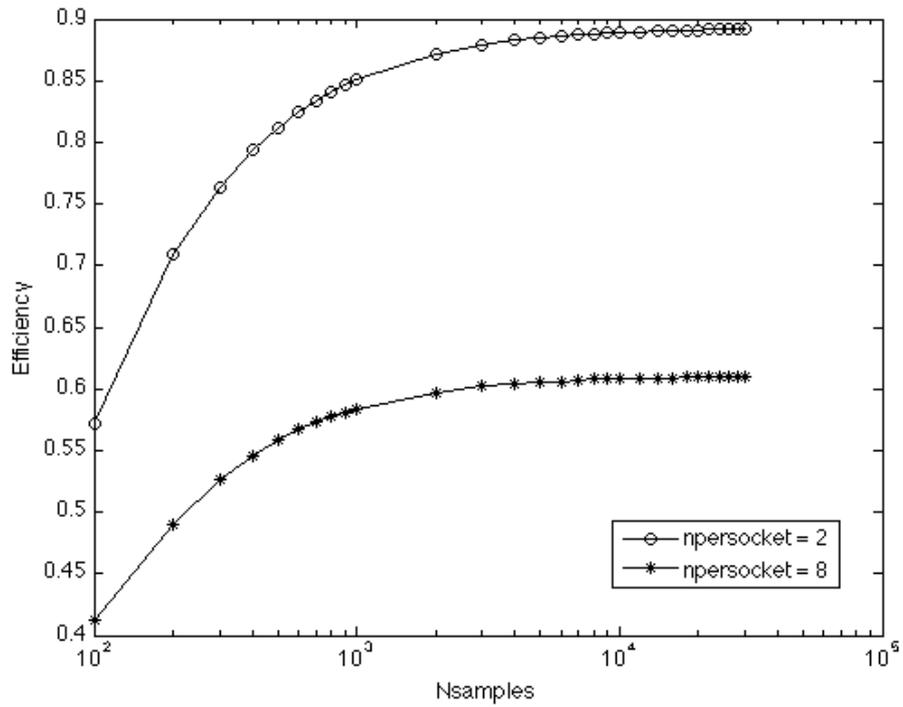


Figure 4.33: Efficiency as a function of number of samples for different npsocket values using 32 processors, [1000×1000] grid and Dynamic master-slave

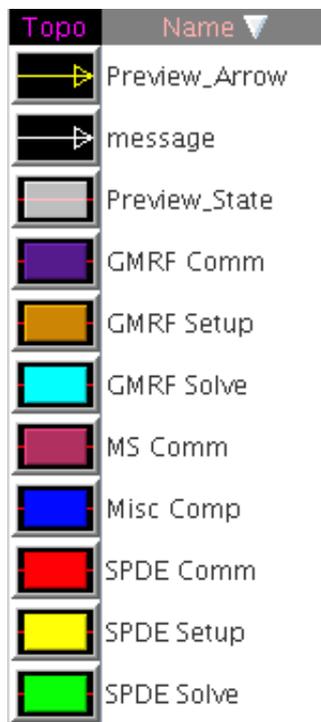


Figure 4.34: MPE Jumpshot color legend

```

[i10hpc2:26596] MCW rank 9 bound to socket 1[core 4]: [ . . . . . ] [ . . . . . B . . . ]
[i10hpc2:26596] MCW rank 0 bound to socket 0[core 0]: [ B . . . . . ] [ . . . . . ]
[i10hpc2:26596] MCW rank 1 bound to socket 0[core 1]: [ . B . . . . . ] [ . . . . . ]
[i10hpc2:26596] MCW rank 2 bound to socket 0[core 2]: [ . . B . . . . ] [ . . . . . ]
[i10hpc2:26596] MCW rank 3 bound to socket 0[core 3]: [ . . . B . . . ] [ . . . . . ]
[i10hpc2:26596] MCW rank 4 bound to socket 0[core 4]: [ . . . . B . . ] [ . . . . . ]
[i10hpc2:26596] MCW rank 5 bound to socket 1[core 0]: [ . . . . . ] [ B . . . . . ]
[i10hpc2:26596] MCW rank 6 bound to socket 1[core 1]: [ . . . . . ] [ . B . . . . . ]
[i10hpc2:26596] MCW rank 7 bound to socket 1[core 2]: [ . . . . . ] [ . . B . . . . ]
[i10hpc2:26596] MCW rank 8 bound to socket 1[core 3]: [ . . . . . ] [ . . . B . . . ]

```

Figure 4.35: Processor bindings report from OpenMPI (balanced)

```

[i10hpc2:25547] MCW rank 9 bound to socket 1[core 1]: [ . . . . . ] [ . B . . . . . ]
[i10hpc2:25547] MCW rank 0 bound to socket 0[core 0]: [ B . . . . . ] [ . . . . . ]
[i10hpc2:25547] MCW rank 1 bound to socket 0[core 1]: [ . B . . . . . ] [ . . . . . ]
[i10hpc2:25547] MCW rank 2 bound to socket 0[core 2]: [ . . B . . . . ] [ . . . . . ]
[i10hpc2:25547] MCW rank 3 bound to socket 0[core 3]: [ . . . B . . . ] [ . . . . . ]
[i10hpc2:25547] MCW rank 4 bound to socket 0[core 4]: [ . . . . B . . ] [ . . . . . ]
[i10hpc2:25547] MCW rank 5 bound to socket 0[core 5]: [ . . . . . B . ] [ . . . . . ]
[i10hpc2:25547] MCW rank 6 bound to socket 0[core 6]: [ . . . . . B . ] [ . . . . . ]
[i10hpc2:25547] MCW rank 7 bound to socket 0[core 7]: [ . . . . . B ] [ . . . . . ]
[i10hpc2:25547] MCW rank 8 bound to socket 1[core 0]: [ . . . . . ] [ B . . . . . ]

```

Figure 4.36: Processor bindings report from OpenMPI (unbalanced)

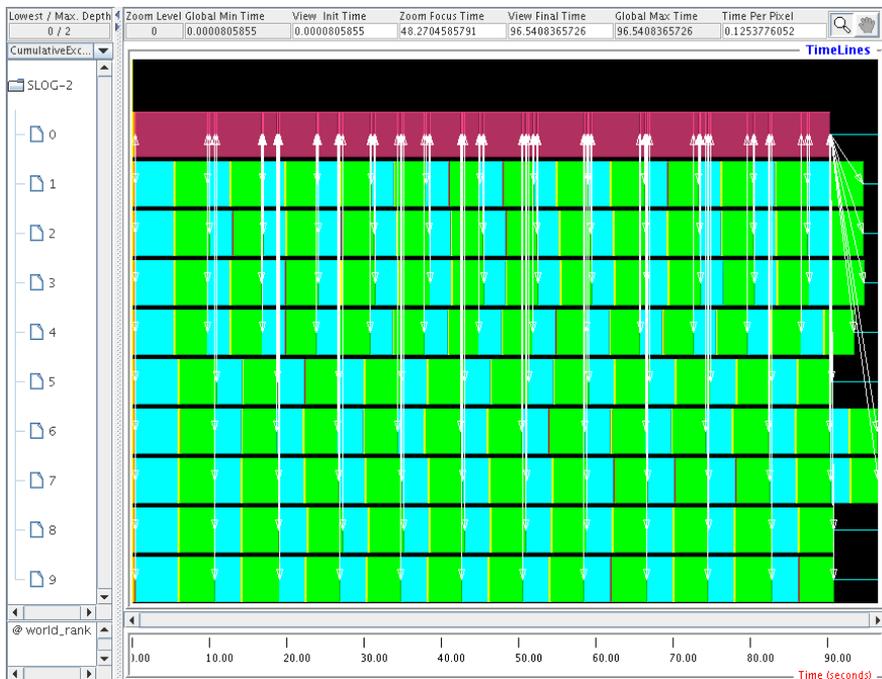


Figure 4.37: Jumpshot analysis for 10 processors (balanced)

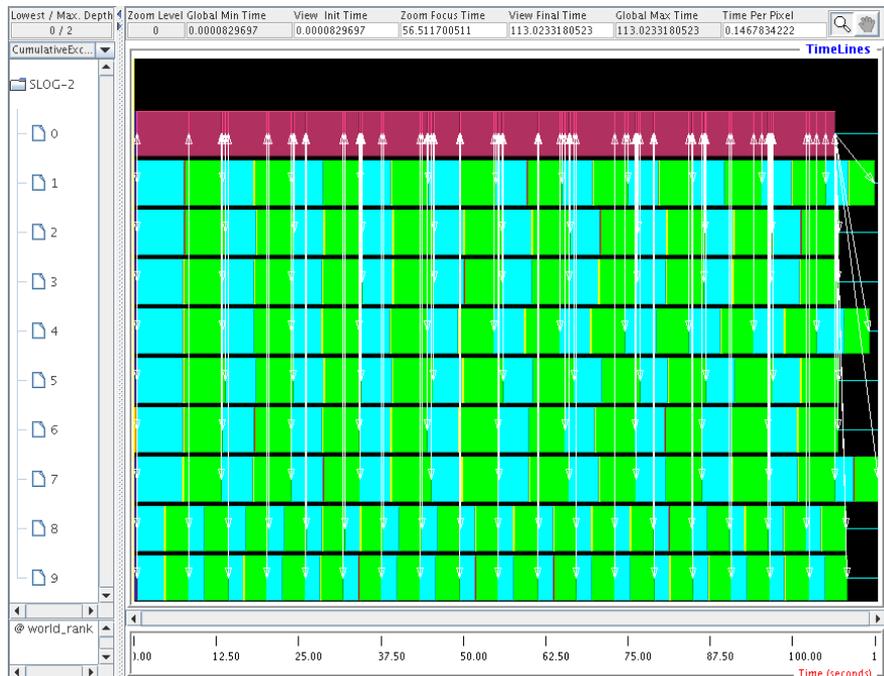


Figure 4.38: Jumpshot analysis for 10 processors (unbalanced)

Proc[5]: We did 10 samples	Proc[5]: We did 9 samples
Proc[7]: We did 11 samples	Proc[7]: We did 10 samples
Proc[9]: We did 10 samples	Proc[9]: We did 17 samples
Proc[6]: We did 11 samples	Proc[6]: We did 9 samples
Proc[8]: We did 10 samples	Proc[8]: We did 17 samples
Proc[4]: We did 12 samples	Proc[3]: We did 9 samples
Proc[3]: We did 12 samples	Proc[4]: We did 10 samples
Proc[1]: We did 12 samples	Proc[1]: We did 10 samples
Proc[2]: We did 12 samples	Proc[2]: We did 9 samples
Proc[0]: All done!	Proc[0]: All done!

(a) Balanced

(b) Unbalanced

Figure 4.39: Program output for 10 processors

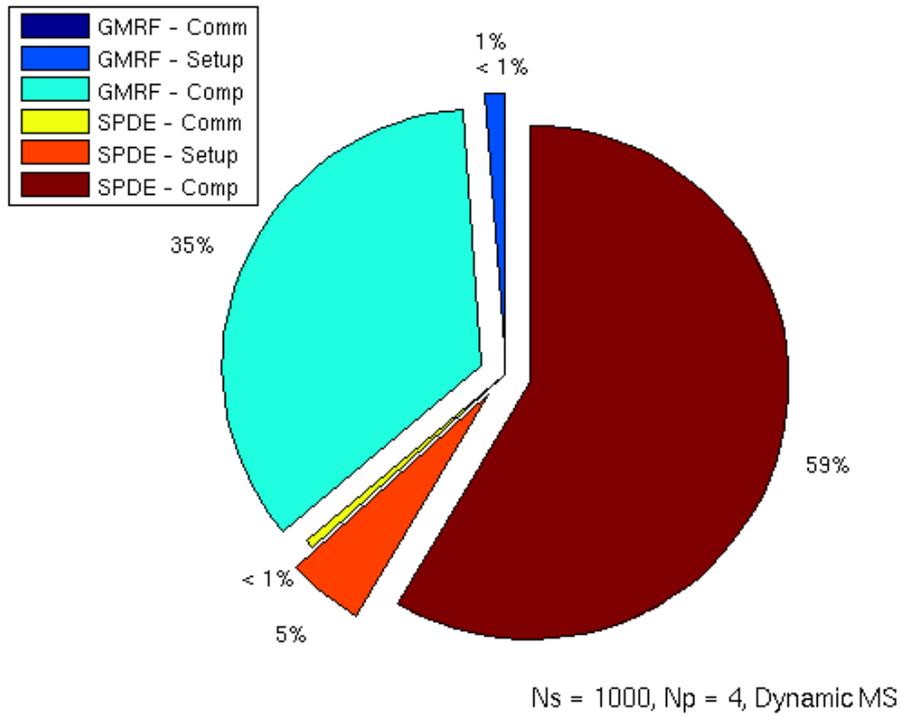


Figure 4.40: Percentage of total wall clock time for Dynamic Master-Slave (4 processors, 500x500 grid size, 1000 samples)

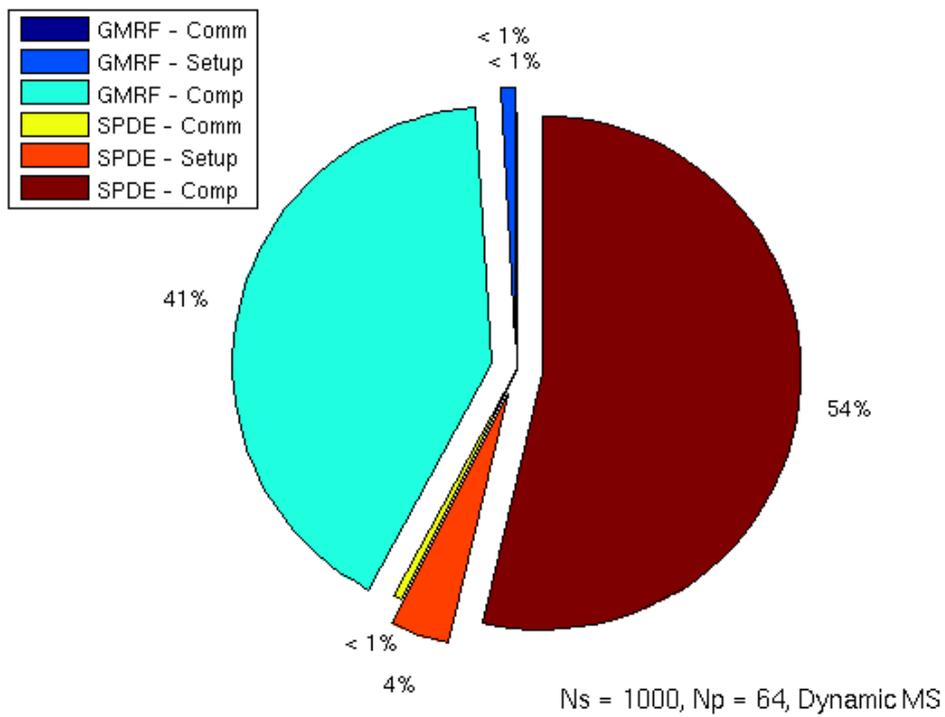


Figure 4.41: Percentage of total wall clock time for Dynamic Master-Slave (64 processors, 500x500 grid size, 1000 samples)

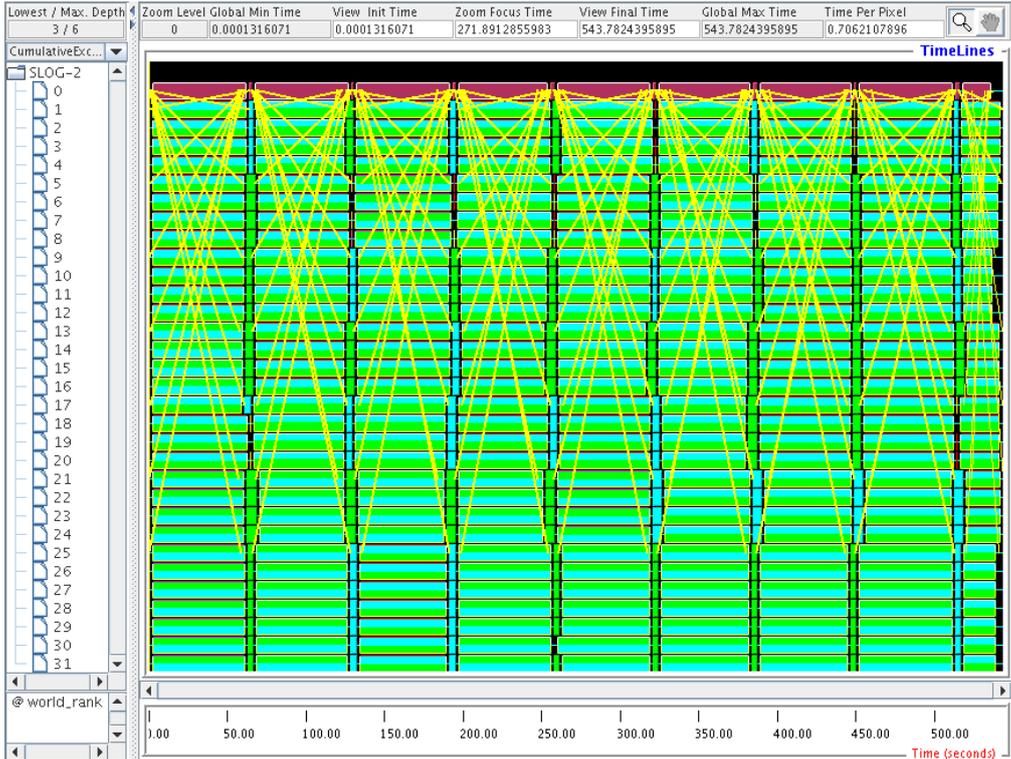


Figure 4.42: Jumpshot visualization of PETSc program for Dynamic Master-Slave - 4 processors per sample (32 processors, 1000x1000 grid size, 500 samples)

4.7.1 Results

Figure 4.43 shows the expectation of the solution field as calculated from the target code generated by ExaStencils. We use a $[511 \times 511]$ grid size since the domain decomposition method employed by ExaStencils does not allow for arbitrarily large grid shapes or sizes.

Results show that the expectation of the norm, $\mathbb{E}[Q_{511}] = 5.22667$, calculated by the target code is close to that of the PETSc cases. Also, since, the field qualitatively looks consistent to the solutions produced by PETSc and MATLAB (fig. 4.43), we proceed with the assumption that the generated code works correctly.

As for performance, the target code was run with 36 processors (18 processors per node) in the LSS cluster. Figure 4.44 shows the speedup attained by the ExaStencils target code as a function of number of samples with respect to the PETSc code. The comparison is made between the best timings of the ExaStencils generated code and that of the PETSc code. The ExaStencils code uses a geometric multigrid solver with a Red-Black Gauss-Seidel for the relaxation solver. The CG relaxation is at the coarsest grid only. The PETSc code uses the (fgmres+BoomerAMG) combination as mentioned in previous sections. For the same type of parallelization strategy (DD), we see that the ExaStencils code is consistently 3 times faster than the corresponding PETSc code. However, when we change the parallelization strategy from just DD to Dynamic Master-Slave (MSD), then, approximately after 300 samples, the PETSc code can be as fast as or faster than the ExaStencils generated code. Also from the trend, it can be conjectured that as we increase the number of samples, the PETSc code will be faster than the ExaStencils code. This is not overly surprising since the MSD parallelization strategy is ideally suited for the Monte Carlo method as we have seen from

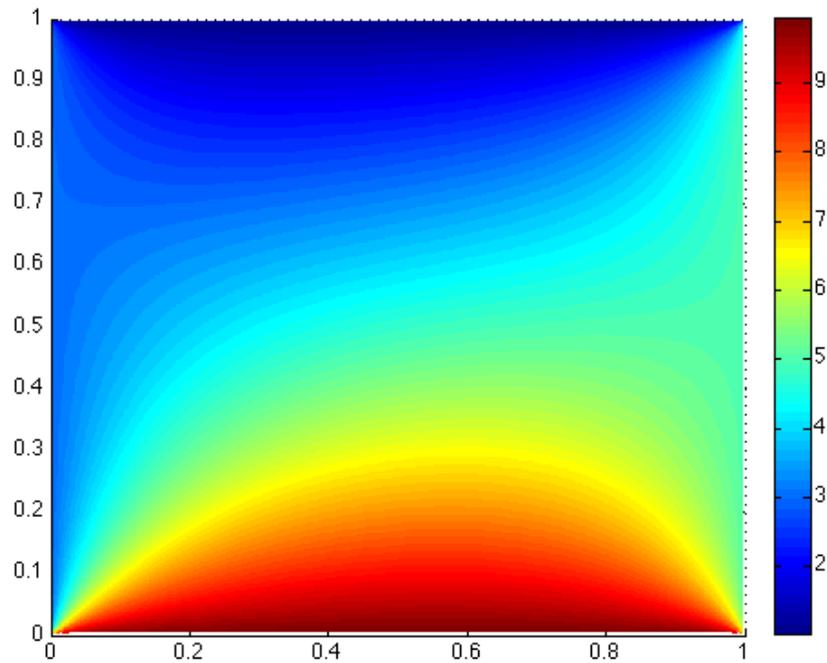


Figure 4.43: Expectation of the solution field as calculated from the code generated by ExaStencils ($[511 \times 511]$ points), 100 samples

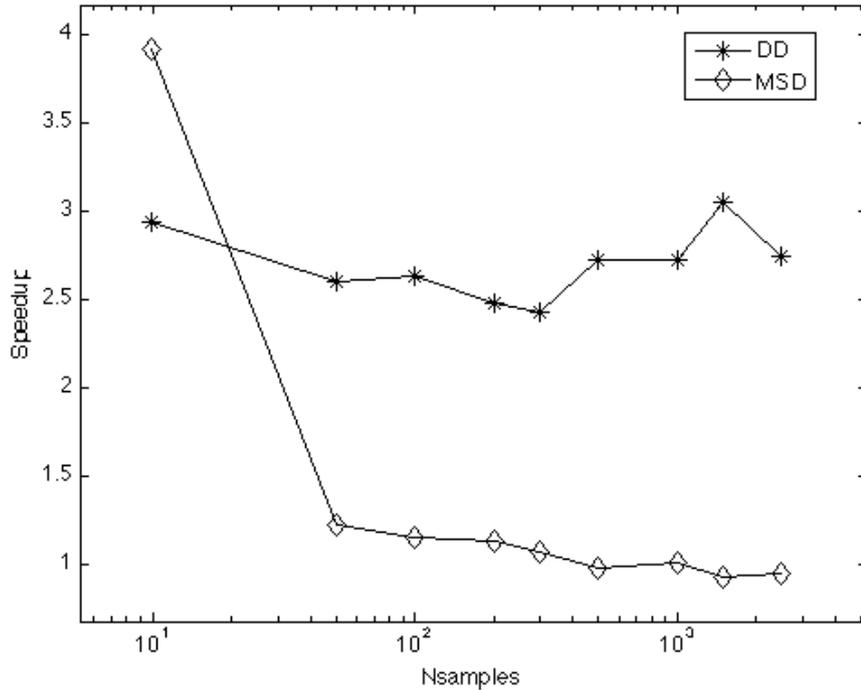
Section 4.6.1. It would be very beneficial if the ExaStencils project can incorporate a MSD type of parallelization strategy in order to support Monte Carlo type applications.

4.7.2 Experiences

ExaStencils is a very new concept. As an early end user, the following are some notes about implementing the thesis problem using ExaStencils:

- The language at layer 4 has a relatively low learning curve provided documentation is available.
- Similar in user friendliness such as MATLAB, but much easier than C/C++ when compared to the PETSc implementation.
- Still limited in application capability (No arbitrary grid size, and no custom parallelization strategy i.e. Master-Slave for concurrent sample calculations).
- Performance of target code is very good and should be even more enhanced if concurrent sample calculations can be performed.

Figure 4.44: Speedups for ExaStencils



- Program code (DSL Layer 4) management is easier than the PETSc implementation (484 lines vs 575 lines) and (1 source file vs 5 source/header files).
- More comfortable with PETSc or MATLAB since the source codes can be easily debugged. Not the same with the generated target code which generated greater than 130 source files.

In general, MATLAB is the easiest to implement but suffers from poor performance and the inability to scale. Implementing the C++ code using the PETSc library made it significantly easier to obtain decent performance gains, but the code development time is also significantly longer compared to that with MATLAB. The DSL Layer 4 program is easy to manage, and the generated code provides a large performance gain but debugging the target code will be a challenge. As a comparison of the different programming languages, below are snippets of code (Listings 2 to 4) from the different languages to calculate $res = b - Ax$, as an example.

Listing 2: MATLAB example

```
|| residual = b - A*x;
```

Listing 3: PETSc example

```
|| ierr = MatMult(A,x,xtemp);CHKERRQ(ierr);
|| ierr = VexWXPY(residual,-1,xtemp,b);CHKERRQ(ierr);
```

Listing 4: ExaStencils Layer 4 example

```
|| loop over inner on Residual@current{  
||   Residual@current = RHS@current - (Laplace@current * Solution[0]  
||     @current)  
|| }
```

5 Conclusion

The results show that the two recent methods, namely, GMRF approximation of GFs with Matérn covariance and Multilevel Monte Carlo (MLMC), are indeed capable of reducing the computational cost in Monte Carlo simulations. The MLMC method implemented in MATLAB is slightly different from the standard MLMC. Results did show that they are consistent. Different solvers and preconditioners were tested for robustness to solve the toy problem with the flexible gmres and BoomerAMG preconditioner performing best. It was found that some solvers failed to converge and this was attributed to the statistical parameters that caused the generation of irregular GFs in the discretized domain. Different parallelization strategies for the SLMC have also been analyzed. From the result, the Dynamic Master-Slave method is more suitable to solve the problem in this thesis. The composition of time-to-solutions for the Cholesky decomposition and GMRF methods were discussed. They show the ability of the GMRF method to generate samples for large computational domains at costs which are close to solving a second PDE. It is also seen that the hardware infrastructure to solve such problems are also important. We see that if the number of processors used are smaller than the total processors available in the computational nodes, then, it is better to distribute the processes evenly amongst the nodes. Some preliminary results was also shown for the implementation of the GMRF approximation method in ExaStencils. Results show a significant speedup of approximately 3 when compared to a corresponding C++/PETSc implementation with just domain decomposition. It actually shows that if ExaStencils can provide a custom parallelization strategy such as Dynamic Master-Slave, the speedup can be enhanced further when dealing with a large number of independent samples. We also see that the ExaStencils concept can produce codes of significant performance gains with a relatively "easy" coding experience requirement.

5.1 Future work

We have seen in the MATLAB implementation that the MLMC method can be used to reduce the computational cost for Monte Carlo experiments. The next step would be to implement it in a lower level language such as C++ with PETSc to determine the actual speed enhancements that can be obtained. Also, the GMRF approximation method needs to be compared with other efficient GF generating methods such as the circulant embedding method. It would be also beneficial to implement the MLMC with the GMRF method in ExaStencils as the performance enhancement shown in the preliminary study indicates the ability of the ExaStencil compiler to generate a tuned code for a particular machine.

5.2 Acknowledgments

This thesis would not have been possible without the help from other individuals both from the academic and personal aspects. I would therefore like to extend the greatest appreciation to my daily thesis supervisor Dr.-Ing. Harald Köstler for his support and guidance throughout my stay in Universität Erlangen. Many thanks also to Dr.-Ing. Björn Gmeiner, and Sebastian Kuckuk for their many advices as well as easing any administrative pressures. I would also like to thank Prof. Dr. Ulrich Rude for accepting me into the LSS Chair 10 to do this thesis.

Besides the people in Universität Erlangen, I would like to specially extend my gratitude to Prof. Kees Vuik, from EWI at TUDelft and the many other professors at the TUDelft for

their guidance during the coursework portion of my Masters program. Your courses helped sharpen my mathematical skills by a lot (qualitatively that is). A special shout out also for my friends in Delft, thanks for the great times! It was a pleasure working in the many coursework projects with all of you. Thanks also go to Mevr. Evelyn Sharabi for her administrative help during my stay in Delft.

I would like to thank the Erasmus Mundus COSSE consortium for giving me this opportunity to pursue this amazing dual Masters program in Europe and for supporting me financially. Thank you Ms. Karin Knutsson for all the COSSE-related admin work to help make transitioning between two universities easier.

Special thanks go to my parents, Loh Siew Weng, and Mah Lee Fung, for their love and support throughout my academic career. And last but not least, thank you *Lee Ping*.

References

- [1] R. J. ADLER AND J. E. TAYLOR, *Random Fields and Geometry*, Springer Verlag, 2007.
- [2] S. BALAY, S. ABHYANKAR, M. F. ADAMS, J. BROWN, P. BRUNE, K. BUSCHELMAN, V. EIJKHOUT, W. D. GROPP, D. KAUSHIK, M. G. KNEPLEY, L. C. MCINNES, K. RUPP, B. F. SMITH, AND H. ZHANG, *PETSc users manual*, Tech. Rep. ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.
- [3] J. BROWN, P. BRUNE, E. CONSTANTINESCU, M. KNEPLEY, AND B. SMITH, *Towards high throughput composable multilevel solvers for implicit multiphysics simulation*, Presented at the National Renewable Energy Laboratory, Golden, CO, 2012.
- [4] R. E. CAFLISCH, *Monte carlo and quasi-monte carlo methods*, Acta Numerica, (1998), pp. 1–49.
- [5] A. CHAN, W. GROPP, AND E. LUSK, *An efficient format for nearly constant-time access to arbitrary time intervals in large trace files*, Scientific Programming, 16 (2008), pp. 155–165.
- [6] K. CLIFFE, M. GILES, R. SCHEICHL, AND A. TECKENTRUP, *Multilevel monte carlo methods and applications to elliptic pdes with random coefficients*, Computing and Visualization in Science, 14 (2011), pp. 3–15.
- [7] F. DESERNO, *Hpc-cluster - access and use (lss)*.
- [8] C. DIETRICH AND G. NEWSAM, *Fast and exact simulation of stationary gaussian processes through circulant embedding of the covariance matrix*, SIAM Journal on Scientific Computing, 18 (1997), pp. 1088–1107.
- [9] G.-A. FUGLSTAD, *Spatial modelling and inference with spde-based gmrf's*, Master's thesis, Norwegian University of Science and Technology, Norway, 2011.
- [10] J. J. GERBRANDS, *On the relationships between svd, klt and pca*, Pattern Recognition, 14 (1981), pp. 375–381.
- [11] M. GILES, *Multi-level monte carlo path simulation*, Operations Research, 56 (2008), pp. 607–617.
- [12] V. E. HENSON AND U. M. YANG, *Boomeramg: a parallel algebraic multigrid solver and preconditioner*, Applied Numerical Mathematics, 41 (2000), pp. 155–177.
- [13] H. KÖSTLER, C. SCHMITT, S. KUCKUK, F. HANNIG, J. TEICH, AND U. RÜDE, *A scala prototype to generate multigrid solver implementations for different problems and target multi-core platforms*, CoRR, abs/1406.5369 (2014).
- [14] F. LINDGREN AND H. RUE, *Bayesian spatial and spatio-temporal modelling with r-inla*, 2013.
- [15] F. LINDGREN, H. RUE, AND J. LINDSTRÖM, *An explicit link between gaussian fields and gaussian markov random fields: the stochastic partial differential equation approach*, Journal of the Royal Statistical Society, 73 (2011), pp. 423–498.

- [16] C. PAIGE AND M. SAUNDERS, *Solution of sparse indefinite systems of linear equations*, SIAM Journal of Numerical Analysis, 14 (1975), pp. 617–629.
- [17] M. QUINN, *Parallel Programming in C with MPI and OpenMP*, McGraw-Hill Science/Engineering/Math Series, USA, 1 ed., 2003.
- [18] P. ROACHE, *Perspective: A method for uniform reporting of grid refinement studies*, ASME Journal of Fluids Engineering, 116 (1994).
- [19] S. ROSS, *Stochastic Processes*, John Wiley & Sons, Inc., United States of America, 2 ed., 1980.
- [20] H. RUE AND L. HELD, *Gaussian Markov Random Fields: Theory and Applications*, Monographs on Statistics and Applied Probability, Chapman & Hall, FL, USA, 2005.
- [21] H. RUE AND H. TJELMELAND, *Fitting gaussian markov random fields to gaussian fields*, Scandinavian Journal of Statistics, 29 (2002), pp. 31–49.
- [22] Y. SAAD, *A flexible inner-outer preconditioned gmres algorithm*, SIAM Journal on Scientific Computing, 14 (1994), pp. 461–469.
- [23] M. L. STEIN, *Interpolation of Spatial Data: Some Theory for Kriging*, no. XVII in Springer Series in Statistics, Springer New York, New York, USA, 1999.
- [24] J. VAN KAN, G. SEGAL, AND F. VERMOLEN, *Numerical Methods in Scientific Computing*, VSSD, Delft, the Netherlands, 2005.
- [25] C. VUIK AND D. LAHAYE, *Scientific computing (wi4201) - notes*, tech. rep., Delft Institute of Applied Mathematics, 2012.
- [26] J. B. WALSH, *École d’Été de Probabilités de Saint Flour XIV - 1984*, Springer Berlin Heidelberg, 1986.
- [27] B. WILKINSON AND C. ALLEN, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*, An Alan R. Apt book, Pearson-Prentice Hall, 2005.
- [28] S. WILLIAMS, A. WATERMAN, AND D. PATTERSON, *Roofline: An insightful visual performance model for multicore architectures*, Communications of the ACM, 52 (2009), pp. 65–76.

A PETSc Solver Settings Report

The following are the details of the default solver (fgmres + BoomerAMG) used to solve the samples in the PETSc program as reported in the PETSc solver log. The settings for the BoomerAMG solver only as described in Section 4.5.1 is the same as that of the preconditioner with the maximum number of V cycles set at 1000.

Solver type: fgmres (flexible gmres)

GMRES: restart=30, using Classical (unmodified) Gram-Schmidt
Orthogonalization with no iterative refinement
GMRES: happy breakdown tolerance 1e-30
maximum iterations=1000, initial guess is zero
tolerances: relative=1e-6, absolute=1e-50, divergence=10000
right preconditioning
using UNPRECONDITIONED norm type for convergence test

Preconditioner

type: hypre

HYPRE BoomerAMG preconditioning
HYPRE BoomerAMG: Cycle type V
HYPRE BoomerAMG: Maximum number of levels 25
HYPRE BoomerAMG: Maximum number of iterations PER hypre call 1
HYPRE BoomerAMG: Convergence tolerance PER hypre call 0
HYPRE BoomerAMG: Threshold for strong coupling 0.25
HYPRE BoomerAMG: Interpolation truncation factor 0
HYPRE BoomerAMG: Interpolation: max elements per row 0
HYPRE BoomerAMG: Number of levels of aggressive coarsening 0
HYPRE BoomerAMG: Number of paths for aggressive coarsening 1
HYPRE BoomerAMG: Maximum row sums 0.9
HYPRE BoomerAMG: Sweeps down 1
HYPRE BoomerAMG: Sweeps up 1
HYPRE BoomerAMG: Sweeps on coarse 1
HYPRE BoomerAMG: Relax down symmetric-SOR/Jacobi
HYPRE BoomerAMG: Relax up symmetric-SOR/Jacobi
HYPRE BoomerAMG: Relax on coarse Gaussian-elimination
HYPRE BoomerAMG: Relax weight (all) 1
HYPRE BoomerAMG: Outer relax weight (all) 1
HYPRE BoomerAMG: Using CF-relaxation
HYPRE BoomerAMG: Measure type local
HYPRE BoomerAMG: Coarsen type Falgout
HYPRE BoomerAMG: Interpolation type classical

B Jumpshot Visualizations

These are the full Jumpshot visualizations to solve a system of $[1000 \times 1000]$, with the GMRF approximation, 500 samples, and the Dynamic master-slave approach using 32 processors.

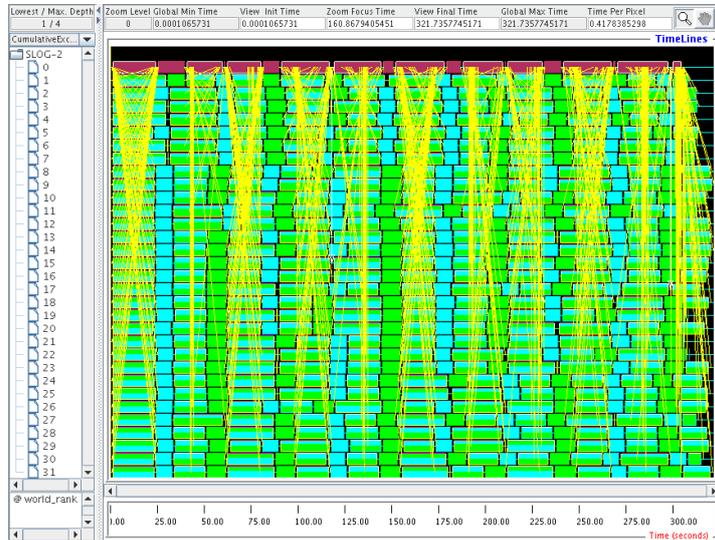


Figure B.1: Jumpshot visualization of PETSc program for Dynamic Master-Slave - 1 processor per sample (32 processors, 1000x1000 grid size, 500 samples)

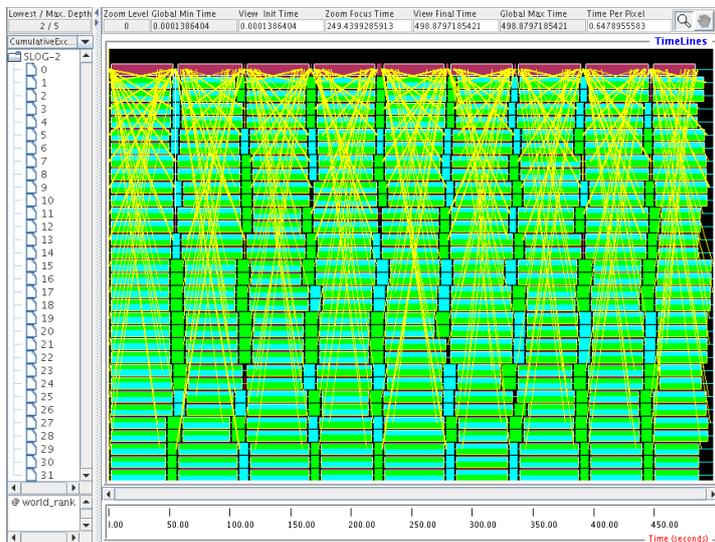


Figure B.2: Jumpshot visualization of PETSc program for Dynamic Master-Slave - 2 processors per sample (32 processors, 1000x1000 grid size, 500 samples)