Parallel Multilevel Monte Carlo Algorithm for Capacitance Extraction in Non-Manhattan Geometries

Tiago João Pereira da Silva

Thesis to obtain the Master of Science Degree in Engineering Systems and Computer Engineering

Supervisors: Prof. José Carlos Alves Pereira Monteiro
Dr. Francisco Manuel Bernal Martinez

Examination Committee
Chairperson: Prof. João António Madeiras Pereira
Supervisor: Prof. José Carlos Alves Pereira Monteiro
Member of the Committee: Prof. Jorge Miguel Ramos Domingues Ferreira Vieira

June 2017
Dedicated to my loved ones...
Acknowledgments

I would like to thank my family for their unflattering love, support, and guidance. My girlfriend for motivating me and standing with me on the harder times. My advisor Prof. José Carlos Monteiro for his advice and friendly support. I also extend my thanks to my co-advisor Prof. Juan A. Acebron and to Dr. Francisco Bernal for their readiness to help and teach me.
Resumo

Quanto mais o número de componentes de circuitos integrados aumenta, maior há uma procura por algoritmos eficientes que analisem e encontrem falhas nos projetos de circuitos. Um dos problemas que se procura são capacitâncias parasíticas. Os métodos de extração de capacitância elétrica mais utilizados são algoritmos Monte Carlo pois estes são facilmente paralelizáveis e portanto capazes de resolver problemas de larga escala em tempo útil. Estes algoritmos têm sido objeto de estudo ao longo dos anos e surgiram algumas técnicas que permitem que tenham um melhor desempenho. Nesta tese, apresentamos as várias soluções que existem e discutimos as vantagens e desvantagens de cada uma. Propomos uma nova solução, usando a técnica Multinível com algoritmo Walk-On-Spheres. Esta técnica melhora o desempenho de métodos Monte Carlo através da redução da sua variância estatística. Esta técnica já foi implementada com sucesso para outros problemas mas é nova nesta área. A complexidade do algoritmo é reduzida de $O(e^{-3})$ para $O(e^{-2 \log^2(e)})$, onde $e$ é o erro máximo erro permitido, usando esta técnica. Os resultados obtidos mostram um decréscimo no tempo de computação na ordem de grandeza que era esperada e mostram também que a paralelização permite a escala eficiente do desempenho do algoritmo.

Palavras-chave: Redução da Variância, Multinível, Monte Carlo, Cálculo de Capacitância, Walk-on-Spheres
Abstract

As the number and density of components in integrated circuits increases, the more there is a need for efficient algorithms to analyse and find flaws in circuit design, especially before production. One of the problems to look for in the designs are parasitic capacitances, and that analysis is done by calculating the electrical capacitance. Parallelization is required for handling large scale problems, and Monte Carlo methods are the exact fit for this as they can be easily parallelized. Some parallel Monte Carlo solutions already exist, and some research has been done into improving these algorithms. We discuss the advantages and disadvantages of the existing solutions. We propose and have developed new solution by applying a Multilevel technique to the Walk-On-Spheres algorithm. This technique does variance reduction, which leads to a smaller sampling size and thus better performance. It has been applied in several problems but is novel in this area. We decrease the running time complexity from $O(e^{-3})$ to $O(e^{-2} \log^2(e))$ where $e$ is the maximum error of the result. The results show this improvement and show that the parallel performance scales almost perfectly.

**Keywords:** Variance Reduction, Multilevel, Monte Carlo, Capacitance Calculation, Walk-On-Spheres
Contents

Acknowledgments .................................................. v
Resumo .......................................................... vii
Abstract ........................................................ ix
List of Figures ..................................................... xiii
Nomenclature ...................................................... xv
Glossary .......................................................... xvii

1 Introduction ..................................................... 1
  1.1 Circuit Analysis ............................................. 1
  1.2 Contributions of this Thesis ............................... 2
  1.3 Thesis Outline ............................................. 3

2 Tools ............................................................... 5
  2.1 Message Passing Interface ................................ 5
  2.2 Matlab PDE Tool ........................................... 6

3 State-of-the-Art ................................................ 7
  3.1 Classical Numerical Methods ............................. 7
  3.2 Euler-Maruyama Method .................................. 8
  3.3 Monte Carlo Methods ..................................... 10
    3.3.1 Floating Random Walk ................................ 10
    3.3.2 Walk-On-Spheres ..................................... 11
    3.3.3 Variance Reduction .................................. 14
    3.3.4 Multilevel ............................................ 14

4 Implementation ................................................ 19
  4.1 Circuit Representation ................................... 19
  4.2 Standard Walk On Spheres ............................... 20
  4.3 Multilevel Technique ..................................... 24
  4.4 Gaussian Surface .......................................... 29
  4.5 Parallelization ........................................... 29
  4.6 Verification and Validation .............................. 30
List of Figures

2.1 The MPI Allreduce sum procedure. The numbers represent different processes. Initially each one has a vector of size two with values they calculated themselves. After the procedure they have the sum of values of all processes. Taken from Ref. [9]............. 6

3.1 Summary of the classical method for capacitance extraction. ............................. 8

3.2 Example of an approximate Euler-Maruyama solution and the original function. The original function is the full line and the dashed line connects the points calculated by Euler-Maruyama [5]. ................................................................. 9

3.3 Example of first maximal square boundaries. Three electrodes are represented in full black. The line surrounding them is the integration surface. The maximal square boundaries are centred on the integration surface [4]. ................................................................. 10

3.4 Example of three random paths generated from the same starting point on the surface G1, and their corresponding square boundaries [4]. ................................................................. 11

3.5 SEM images of commercial SRAM memories. [2]. ................................................ 12

3.6 The graph shows the flux of the electric field of the two examples circuits above. The capacitance is proportional to the integral of the areas beneath the lines. The area below the solid curve (corresponding to the sharp-cornered set) is about 1/6 larger than that under the dashed curve (rounded corners) [2]. ................................................................. 12

3.7 Example of two Brownian trajectories generated by the Walk-On-Spheres algorithm. Notice: the starting point O on the Gaussian surface G, the hitting boundary around the components defined by the distance $\varepsilon$, the spheres generated centred on each point of the walk and touching one of the components. Some sphere boundaries are partially hidden to avoid making the diagram confusing. ................................................................. 13

4.1 Example of a very simple circuit composed of two electrodes with rounded corners. Each is represented by composing two rectangles and four circumferences. ....................... 20

4.2 Quadrants of a rectangle and how to calculate in which is the particle. ....................... 23

4.3 The particle’s distance to an electrode is less than $\varepsilon_2$. The path ends and in this case the contribution is zero. ................................................................. 25
4.4 The particle’s distance to an electrode is less than \( \varepsilon \). The voltage of this electrode is recorded and the particle continues. However, soon after the particle gets within \( \varepsilon_2 \) distance to the same electrode. The path ends and in this case the contribution is zero.

4.5 The particle’s distance to an electrode is less than \( \varepsilon \). The voltage of this electrode is recorded. The path continues and eventually the particle gets within \( \varepsilon_2 \) distance to a different electrode. The contribution is the difference between the voltages of those two electrodes.

5.1 PDETool: Partial view of the circuit where the tests were performed and the solved voltage. The circuit’s boundary is not included in the view.

5.2 Histogram of Multilevel simulations showing the average difference between the voltage calculated by the algorithm and the PDETool solution. In red is a fitting Gaussian distribution.

5.3 Histogram of Standard simulations showing the average difference between the results and the PDETool solution. In red is a fitting Gaussian distribution.

5.4 Comparison between the running time of Multilevel and Standard Walk-On-Spheres. Logarithmic scale.

5.5 Comparison between the running time of Multilevel and Standard Walk-On-Spheres. Linear scale.

5.6 Comparison between the logarithm of running time of Multilevel and logarithm of the accuracy, with a linear function.

5.7 Comparison between the running time of parallel Multilevel and parallel Standard Walk-On-Spheres.

5.8 Comparison between the running time of parallel Multilevel WoS, a perfect speed-up and a 95% speed-up.

5.9 Comparison between the running time of parallel Standard WoS, a perfect speed-up and a 95% speed-up.
Nomenclature

\( u \)  Electric potential.

\( \varepsilon \)  Discretisation: distance from which a particle is considered to be in collision to an electrode.

\( e \)  Maximum allowed error of the output of the algorithm.

\( V \)  Variance of the result of Walk-On-Spheres simulations.

\( C \)  Cost or running time of a Walk-On-Spheres simulation.

\( \delta \)  Gaussian delta refers to the distance between the electrodes and the Gaussian integration surface defined around each of them.
## Glossary

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MC</td>
<td>Monte-Carlo are a class of computational algorithms that rely on repeated random sampling to obtain numerical results.</td>
</tr>
<tr>
<td>WoS</td>
<td>Walk-on-Spheres is a Monte-Carlo method used to solve a wide variety of boundary value problems for partial differential equations.</td>
</tr>
<tr>
<td>mse</td>
<td>Mean Square Error.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The density and number of electronic components in integrated circuits has increased over the years and shows no sign of stopping. They reach the order of billions of components. For manufacturing companies it is of the utmost importance to detect early any flaws in their design that can result in malfunctions. And that detection has to happen before the manufacturing process begins or the entire process has to be stopped and redone. These errors are commonly caused by parasitic capacitances, resistances and impedances. The work presented in this thesis focuses on parasitic capacitances, which are caused by an unintended accumulation of electric charge. There are several algorithms and techniques for this. Due to the sheer size and complexity of integrated circuits faster, parallelizable algorithms are in demand.

1.1 Circuit Analysis

For analysing a circuit we need to know the distance to components from each point we would analyse. With classical methods, a distance map was calculated the Eikonal equation (wave propagation) [1]. This is a discretisation of the circuit space into a grid of points, for each of which the distance to the nearest electrode is stored. Solving a distance map on rectilinear or Manhattan geometries is simple as it can be done geometrically. For non-rectilinear geometries though, the process gets complex and fast marching is typically used instead [2].

Having the distance map, the classical methods would proceed to solve the Laplace’s equation for the electrostatic potential and then calculate the gradient to obtain the electric field. However these calculations cannot be easily done for typical integrated circuits. The scale of these problems makes them infeasible to solve in useful time without parallel computers, and the classical deterministic methods are unsuitable for parallelization. They require a large amount of communication between processors because they calculate a global solution, where each point of the distance map could depend on several other calculations. Additionally, because these are matrix based calculations, the complexity grows with an exponent of at least two for two dimension examples or exponent of three for representations of three dimensional circuits. If we could choose to calculate only the exact points needed, much computational
effort could be avoided.

The chosen alternative by academia and industry is stochastic methods. With stochastic methods [2], the calculations are done point-wise rather than being matrix-based. Doing the calculations only on isolated points, to later use those to integrate around the electrodes, is much more efficient. It has the added benefit that a point-wise calculation is intrinsically parallelizable as the calculations for each point do not depend on the results from other calculations of other points. Of course, stochastic implies some degree of inaccuracy, so choosing stochastic methods is a trade-off between performance and accuracy. However, it benefits largely in our favour, as they are more efficient and are highly parallelizable, and still accurate enough for the intended purpose.

For these reasons, stochastic methods are the preferred way to face this problem. Namely the Walk-On-Spheres by M. E. Muller [3], the Floating Random Walk (FRW) by Le Coz and Iverson [4] and the Euler-Maruyama method [5].

As will be discussed further in the State-of-the-Art chapter, much of the research on this subject focuses on solutions such as Walk-on-Spheres or Floating Random Walk and several techniques to reduce the variance of these methods. Variance reduction techniques are of great importance in Monte Carlo methods because a decrease in variance translates to a decrease in complexity of these algorithms and therefore an increase in performance [6]. This thesis proposes a form of variance reduction based on Multilevel Monte Carlo simulations.

Using Multilevel to reduce variance has had several applications, most in computational finance [6]. Its application to a capacitance extraction algorithm is a novel idea which promises to further improve the performance of these algorithms. The several already existing variance reduction techniques cannot be used simultaneously. This technique, however, has the added benefit that it does not exclude the possibility to use other known techniques alongside it [6].

1.2 Contributions of this Thesis

The objective of this work was to find, develop and test an efficient and novel solution for reducing variance in algorithms for capacitance extraction on integrated circuits. Specifically, to develop and test a parallel Multilevel Monte Carlo algorithm for capacitance extraction based on the Walk-On-Spheres method.

Two important aspects were expected from the results of the performance tests of the algorithm. First, that there would be an increase in efficiency because of the reduction in variance through Multilevel, showed by a smaller running time when comparing to Standard Monte Carlo. Second, that the performance of the parallel Multilevel algorithm increases as the number of processors increases. Both objectives were achieved as the results show.
1.3 Thesis Outline

This thesis is organised as follows: This chapter is the introduction. It explains the motivation behind the problem, how and why it is relevant today.

Chapter 2 describes the main tools and technology used.

Chapter 3 is the State-of-the-Art. We first introduce the classical algorithmic methods of circuit analysis, followed by the discussion of the strengths and weaknesses of three different algorithms: Floating Random Walk, Euler-Maruyama Method, Walk-On-Spheres. We justify our choice for the Walk-On-Spheres algorithm. We discuss speed-up techniques used for Walk-On-Spheres and other algorithms such as Variance Reduction techniques. Finally, we explain the Multilevel technique and how it can result in an increase in performance. We go over the disadvantages and how they were addressed.

Chapter 4 goes over the details of our implementation of the Multilevel technique on the Walk-On-Spheres, and goes over the main algorithm of this work.

Chapter 5 presents the results of this thesis, namely, the speed-up obtained with Multilevel compared to Standard Monte Carlo. We show also the scalability of the parallel implementation.

Finally, in Chapter 6 we draw our conclusions about the results obtained. We then discuss whether the objectives were met and possible future work relating this topic that could be interesting to explore.
Chapter 2

Tools

In this chapter the main technologies that were used in developing and testing the algorithm are presented and their use explained. The algorithm’s implementation was done in the programming language C++. For the parallelization of the algorithm OpenMPI was used. As for the testing phase, the Matlab PDETool was used to make the circuit designs and to compare solutions to verify the correctness of the algorithm.

2.1 Message Passing Interface

In parallel distributed systems there is a need for communication between computation tasks [7]. This communication has to happen through messages. Message Passing Interface is a standardized message-passing protocol that works on many different parallel computers - it is the de facto standard. The specification for MPI is defined by a group of researchers from both academia and from the industry: the MPI Forum [8]. It includes, for example, parallel computer vendors and library writers.

MPI provides several routines that facilitate the management of the distributed tasks and the message exchange between them. It can be used to establish communication between processes in a single multi-core computer or between computers in a cluster, or both simultaneously. It allows for global communication between all tasks or specifying a message from one task to another. MPI launches the same executable on the specified set of machines and keeps track of where each task is running at any given moment by the machine address and the process ID. This means the programmer needs only to consider the tasks he or she defines and not the machines since MPI assigns them to machines and knows where each task is. The programmer has only to define which tasks communicate with which without worry of the fact that they could run on the same or in different machines.

We use only one procedure of MPI, called AllReduce sum. If each process has a certain set of values, calling this procedure will compute the sum of all those values and return that result to all processes. Figure 2.1 gives an example of such a scenario.

OpenMPI is a free and open source implementation that offers great performance [10]. There are many implementations of MPI, but the advantages of using OpenMPI is that it is freely available, is
Figure 2.1: The MPI Allreduce sum procedure. The numbers represent different processes. Initially each one has a vector of size two with values they calculated themselves. After the procedure they have the sum of values of all processes. Taken from Ref. [9].

portable as it runs on a variety of systems and has good performance.

2.2 Matlab PDE Tool

During the development of this work, there was a need to have a robust tool that would allow for solving partial differential equations (PDE). The PDE Toolbox is a MATLAB tool which can do exactly that [11]. It also allows to create and design mesh two-dimensional geometries.

This was the main tool used to design circuits for testing purposes. It was first used to draw representations of integrated circuits and then used to compare the PDE solutions from the tool with our own implementation. The tool uses deterministic methods, which are slow and cannot reach the same levels of accuracy this work deals with. However, it was still useful to verify the correctness of our implementation as we will discuss further in the Methodology section.
Chapter 3

State-of-the-Art

In this chapter we explain how capacitance extraction was been done beginning with classical methods and ending with the methods most used nowadays. We go over each algorithm explaining their functioning and details and discussing their applications, advantages and limitations.

3.1 Classical Numerical Methods

Classical numerical methods for capacitance extraction rely on electromagnetic field solvers. These are based on deterministic methods for solving partial differential equations, for example, finite elements or boundary elements methods [2]. These deterministic methods calculate global solutions. By their very nature, they are slow and cannot be parallelized easily, as almost all calculations depend on the results of other calculations. This means that no matter how the work is divided into tasks, communication between them is always necessary. Amdahl's argument [12] states that the benefit of parallelization is only gained for the portion of the program that can be parallelized. These methods require much communication between tasks, which is not a workload that can be distributed, which makes them scale poorly.

To go into further detail, classical methods first consider the following relation between the electrode charges $Q_1, ..., Q_M$, the eletronic potential $u_1, ..., u_M$ and the coefficients $C_{ij}$ of the capacitance matrix:

$$Q_i = \sum_{j=1}^{M} C_{ij} u_j, \quad i = 1, ..., M$$  \hspace{1cm} (3.1)

where $M$ is the number of electrodes. The voltages are known and to determine the capacitances we first need to calculate the charges. Calculating the electric field

$$E = -\epsilon \nabla u$$  \hspace{1cm} (3.2)

where $u$ is the electric potential and $\epsilon$ the permittivity, there is enough information to use Gauss's Theo-
rem and calculate the charge $Q_i$ on each electrode:

$$Q_i = \oint_\omega (E \cdot N) dx^{d-1}$$  \hspace{1cm} (3.3)

With the integral on a closed surface $\omega$ that contains electrode $i$ and no other, and $N_\omega$ is the normal outward unit vector. In other words, it calculates the charge inside the defined surface from the electric field that “flows” through that surface. Finally, an approximation to the capacitance matrix values can be done by solving the partial differential equations:

$$C_{ij} = \frac{\partial Q_i}{\partial V_j}$$  \hspace{1cm} (3.4)

The graph in Fig. 3.1 summarises the process.

The last step in Fig. 3.1, solving the PDE is very computationally heavy. As mentioned before, better performance is needed than the one offered by deterministic methods. Otherwise, we cannot tackle large-scale cases. The main approaches when it comes to the stochastic methods for evaluating circuits are Euler-Maruyama, Floating Random Walk and Walk-On-Spheres method. We will refer to these as integrators as we go over how they work and their advantages and disadvantages.

### 3.2 Euler-Maruyama Method

The Euler-Maruyama method is a general method used to obtain approximate numerical solutions of stochastic differential equations. It can be derived through a generalisation of the Euler method for ordinary differential equations [5].

Consider a stochastic differential equation in its usual form:

$$dX(t) = f(X(t)) \, dt + g(X(t)) \, dW(t)$$  \hspace{1cm} (3.5)

with a certain initial value $X(0) = x_0$. We want to solve the equation for an interval $[0, T]$. This method first divides this interval into $N$ subintervals with the same size $\Delta t = T/N$:

$$0 = \tau_0 < ... < \tau_N = T$$  \hspace{1cm} (3.6)

and then for each of these small intervals, step by step, calculates an approximation to the derivative.
Figure 3.2: Example of an approximate Euler-Maruyama solution and the original function. The original function is the full line and the dashed line connects the points calculated by Euler-Maruyama [5].

For $\tau_j = j \Delta t$, the numerical approximation to $X(\tau_j)$ is $X_j$:

$$X_j = X_{j-1} + f(X_{j-1}) \Delta t + g(X_{j-1}) (W(\tau_j) - W(\tau_{j-1}))$$

This equation defines $X_j$ using $X_{j-1}$, where $W$ are independent and identically distributed normal random variables with expected value zero and variance $\Delta t$, usually expressed as:

$$W(\tau_j) - W(\tau_{j-1}) \sim \sqrt{\Delta t} \mathcal{N}$$

$\mathcal{N}$ being a standard normal distribution. Since it is a recursive definition, $X_j$ from $X_{j-1}$, a given initial value is necessary. Effectively, this method starts from a known point $X(0) = x_0$, and calculates the consecutive point $x_1$, then $x_2, x_3, ..., x_n$, where the distance between each point is always $\Delta t$. This gives an approximation to the original function. Figure 3.2 shows an example of a result of the Euler-Maruyama method.

The distance between each point and the total number of points defines the discretisation level. The smaller distance between them, the better is the accuracy. However, the trade-off is that the performance is worse. This is a rather generic and slow algorithm. In the case of circuit analysis, it has the benefit that it works for any geometry. It can be used for computing the electrostatic potential in the problem being studied. However, the main focus of academia goes to the two next methods as they are competing for
3.3 Monte Carlo Methods

3.3.1 Floating Random Walk

The Floating Random Walk Monte Carlo method was first proposed by Le Coz and Iverson [4] as a much more efficient solution to calculate the electric field compared to deterministic methods. The algorithm has been implemented with different variants and techniques to reduce its variance and increase its performance [13]. It is the basis for much of the research done for capacitance extraction.

It first generates a random point on the defined Gauss surface over an electrode. Then it calculates a maximal square boundary using that point as the centre of the square. A maximal square is a square that is as big as can be without overlapping with an electrode, so a maximal square boundary always touches at least one electrode. An example of maximal squares can be seen in Fig. 3.3.

The algorithm then generates another random point, this time on the square boundary of the last point. This process is repeated until the point chosen is hitting an electrode. The generated points define a “walk”. The algorithm generates these walks many times for each electrode. Figure 3.4 shows example paths generated from one point in the Gaussian surface of an electrode.

The sum of these walks can be used to approximately calculate the voltage for the starting point. The more paths simulated the better the accuracy. The algorithm stops after it reaches a specified accuracy goal.

Floating Random Walk works only for Manhattan geometries. It cannot be used in arbitrary geome-
Figure 3.4: Example of three random paths generated from the same starting point on the surface G1, and their corresponding square boundaries [4].

tries. As shown in Fig. 3.5 circuit components do not have a perfectly rectilinear shape. Considering these shapes to be rectilinear is an approximation that obviously induces an error in the calculations and the final results. According to the error analysis done in [2], this approximation gives a very significant error. The example shown in Fig. 3.6 illustrates this. The figure shows the same electrodes with rounded and with sharp corners. The capacitance calculated on the circuit with sharp corners has an error of 17%, even though the example has only two components. This is the major reason why this work uses instead the Walk-On-Spheres method, which does not imposing a restriction on the geometry of the circuit.

Floating Random Walk algorithm is fast and important to this area, as such it could not go without mention. However, it is more complex and less flexible than the final method we will cover and has the disadvantage of working only on Manhattan geometries. For these reasons, this work did not include it.

3.3.2 Walk-On-Spheres

Walk-On-Spheres is a Monte Carlo algorithm that calculates approximate solutions of some specific boundary value problem for partial differential equations. It was introduced by M. E. Muller in 1956 to solve Laplace’s equation [3]. When applied to capacitance extraction it works on any arbitrary geometry, even non-Manhattan.

It works using probabilistic interpretations of partial differential equations by simulating paths of Brownian motion [14]. Brownian motion is an erratic, seemingly random movement. We can think of this method as particles jumping. In that, it is similar to Floating Random Walk.
Figure 3.5: SEM images of commercial SRAM memories. [2].

Figure 3.6: The graph shows the flux of the electric field of the two examples circuits above. The capacitance is proportional to the integral of the areas beneath the lines. The area below the solid curve (corresponding to the sharp-cornered set) is about 1/6 larger than that under the dashed curve (rounded corners) [2].
Based on the starting position of a particle, a spherical boundary is defined, as big as can be before overlapping with any component. A random point on that surface is chosen as the next position of the particle. The algorithm repeats until hitting any component with the path formed by the points. A particle counts as hitting a component if it is within a certain distance to it, which is defined by a factor $\varepsilon$. With Floating Random Walk, there is no $\varepsilon$ since the square boundaries touch the electrodes a significant portion, while with Walk-On-Spheres the sphere boundary touches only at an one point. Figure 3.7 shows these boundaries and an example of two generated paths.

Once again, the average voltage of the electrodes to which the random walks collided with, gives the estimate for the electric potential for the starting point of those walks. For both of these Monte Carlo algorithms, the end goal is the same. After calculating the voltage of enough points that surround an electrode, all of them in a defined Gaussian surface, the capacitance of it can be known by means of integration. For this mathematical equivalence to be true, in both these algorithms the particles must be reflected by the circuit boundaries, so that their random path ends only at an electrode.

For these Monte Carlo methods, the accuracy is clearly increased with more generated paths, as it is essentially having a bigger sample for a better approximation. However, there are techniques that can increase the performance of these methods. If there were ways to reduce the variance of the sampling done, less samples or simulations would be needed to achieve the same accuracy in the result.
3.3.3 Variance Reduction

Both Floating-Random-Walk and Walk-On-Spheres have been the subject of studies on how to increase their performance [15] [16]. Most of which coincide with the same techniques that can improve any Monte Carlo method: variance reduction techniques.

This work focuses on reducing the variance of an integrator. Other research done on this includes, for example, Yu et al [17] used a form of predictive sampling to decrease the variance of the Floating Random Walk. Instead of picking random samples uniformly distributed, they used a probability function. This allows to perform fewer random walks without significant impact on accuracy. By sampling with averaged weight values instead of equal probability for all, they could have their method converging faster.

Another example of variance reduction is the use of stratified sampling. A simplified explanation is that it accelerates the algorithm by dividing the domain into sub-domains, which in practice is turning the integral into a sum of integrals. This reduces the variance because due to the stochastic independence, the sum of the variances is less than the total variance [17].

This is the context where Multilevel comes into the picture, as another technique that can accelerate Monte Carlo algorithms.

3.3.4 Multilevel

Multilevel is a generic technique that can be used to improve the accuracy and therefore the complexity of the Monte Carlo method [18]. Because of the popularity of Monte Carlo, the Multilevel technique sees plenty of applications, many on computational finance, but using it for capacitance extraction is novel. Examples of its applications include computing the expected pay-off of several financial options for comparing them and calculating the sensitivities of certain prices to various input parameters (called “Greeks” in mathematical finance) [13]. In [18] it is used to increase the performance of solving elliptic partial differential equations with random coefficients in the case of model uncertainty quantification for groundwater flow.

The idea behind it is simple. Instead of doing all Monte Carlo simulations with the same accuracy and performance cost, the same accuracy can be reached in the final result with less computational time by performing many not so accurate but very fast simulations, and only a few very accurate but performance heavy simulations. Effectively Multilevel is looking at different levels of accuracy (and inversely computational time), as an optimisation problem: “How many simulations should I run in each accuracy level, such that I use the least amount of time to reach an accuracy goal?”.

The disadvantage of Multilevel comes from trying to solve the optimisation problem as best as possible in a very limited time. Different runs of the random simulations will have different costs and contributions, and as such different solutions to the optimisation problem of how many simulations should run at each level. Therefore, it is not useful to invest too much time in getting the exact number of simulations that are optimal. In order for Multilevel to be worth it, it has to approximate a solution to the optimisation problem and actually run that number of simulations faster than simply running all simulations with the
same accuracy.

To solve this problem, what is done is a fast approximation to how many simulations should be ran at each level. Starting by running several thousand simulations at different accuracy levels, then the average variance and running time of a simulation at each level is calculated. Finally, a solution is put forward, and the number of simulations from that solution are performed. The solution is not guaranteed to be optimal and with millions of simulations it is statically impossible that it is. However it does not have to, so long as a good enough approximation can be achieved that would increase performance.

Multilevel thus comes at the expense of more calculations. Although an overhead is introduced with these calculations, it has been shown that Multilevel Monte Carlo can achieve the same goal accuracy with less computational time compared to standard Monte Carlo [6]. It is important to retain that the more accurate we want the result, the more simulations are run. When dealing with significantly high numbers of simulations, the overhead becomes negligible. Therefore, this technique might not promise any improvement if the accuracy asked for is trivial. For real cases where the available computational power is significant and the accuracy needed requires hundreds of millions of simulations, the performance gain can be very attractive.

It is clear now that Multilevel has a very different approach to other techniques, and that it can bring about better results. Before we explain how we implemented this general technique, we will demonstrate mathematically how the multilevel technique reduces the variance of Monte Carlo and thus decreases its running time.

Monte Carlo path simulations can give us the expected value of a variable, solution to a stochastic differential equation. It was shown in [19] [20], that the mean-square-error $e$ of such estimate has an asymptote of

$$e = \frac{c_1}{N} + c_2\varepsilon^2,$$

(3.9)

where $c_1$ and $c_2$ are positive constants, $N$ the number of simulations and $\varepsilon$ the discretisation. The first part of this error has its source on the variance of the Monte Carlo sampling and decreases with the number of simulations. The second source of error is the square bias introduced by the discretisation of the simulations $O(\varepsilon^2)$. The constants $c_1$ and $c_2$ determine the decrease of the error in sampling as the number of simulations increases and the decrease of the bias as the discretisation decreases accordingly. If they were known, then the optimal number of simulations at each discretisation level would be known. Although they cannot be determined, they can be estimated with great accuracy as we will explain later.

If using standard Monte Carlo with a target root-mean-square-error of $O(e)$, then a magnitude order of $O(e^{-2})$ simulations would have to be ran with discretisation $O(e)$ for a total computational complexity cost of $O(e^{-3})$. However using a Multilevel technique a smaller sampling error can be achieved. Which means it can achieve the same accuracy while reducing the complexity cost to $O(e^{-2}(\log(e)^{-2}))$, provided that the constants $c_1, c_2$ are known [6].

The Multilevel technique considers a sequence, $l = 0, 1, ..., L$, where for each $l$ the discretisation $\varepsilon$ is smaller and the number of simulations smaller. Let us consider the implications of these different levels of
discretisation. The simulations on the last levels, with very small discretisation, are very computationally expensive yet yield a very small error allowing greater accuracy. Although for the simulations on the first levels the gain from each is considerably less, the cost of running such simulations is also much smaller. The simplified idea of Multilevel is to achieve the same accuracy of N simulations on the level with these smallest discretisation, running instead many simulations on the cheapest levels and a few simulations on the most costly levels, for a smaller total computational cost. Essentially the technique allows us to have less variance in the simulations by optimising how many simulations are done with different discretisations.

Let’s say that each simulation or Brownian path gives a result, called the payoff \( P \), while \( P_l \) denotes the approximation to \( P \) and \( E[P_l] \) is the estimator of \( P \). Then, the Multilevel technique described in the above paragraphs can be summed up as:

\[
E[P_L] = E[P_0] + \sum_{l=1}^{L} (E[P_l - P_{l-1}])
\]

(3.10)

The \( E[P_L] \) estimator, is equivalent to standard Monte Carlo with discretisation level \( L \). \( E[P_0] + \sum_{l=1}^{L} (E[P_l - P_{l-1}] \) is equivalent to the Multilevel technique. It has the same expected value and thus equal accuracy to standard. However, it reaches that value by calculating the sum of different estimators.

The left term of the sum \( E[P_0] \) is called the initial solution. It is calculated using standard Monte Carlo simulations just like \( E[P_L] \) is, except at a bigger discretisation and thus cheaper in computational cost. The estimator used is the average pay-off of \( N_0 \) independent simulations.

The right term of the sum is calculated using the Multilevel technique. It is important to understand what is the estimator for \( P_l - P_{l-1} \). For two simulations with same conditions and same randomly generated values, two pay-offs are obtained one with discretisation \( \varepsilon_l \) and the other with \( \varepsilon_l - 1 \). The difference between those two pay-offs is the contribution to that estimator. There are \( L - 1 \) estimators total, each calculating the average of the differences between two simulations. These are called the corrections to the initial solution. The final result is thus calculated from the sum of the value from these estimators with the average result of \( N_0 \) low accuracy Monte Carlo simulations.

In order to achieve the best performance possible, the exact average cost and variance of a simulation at each level would have to be known. The number of simulations that should be ran at each level to minimise cost is a solved optimisation problem. However cost and variance change with discretisation and circumstance (in the case of particle simulation, the starting position differs). So the best that can be done is to estimate them, which needs to happen while the algorithm runs.

The optimal number of simulations at each discretisation level can be known by solving the optimisation problem:

\[
\text{minimise } \sum_{i=0}^{L} N_l C_i \text{ such that } \frac{e^2}{2} = \sum_{i=0}^{L} \frac{V_i}{N_l}
\]

(3.11)

Which gives the formula [6]:

\[
N_l = \left\lceil \frac{1}{2e^2} \sqrt{\frac{V_l}{C_l}} \sum_{i=0}^{L} \sqrt{V_i C_i} \right\rceil
\]

(3.12)
Where $N_l$ is the number of simulations for level $l$, $e$ is the maximum allowed mean-square-error, $V_l$ is the variance of a simulation on level $l$, $C_l$ the cost, and $L$ the smallest and most accurate discretisation level. As mentioned, the algorithm starts by running a fixed number of simulations at each level. It is then possible to calculate the average cost at each level by measuring the time elapsed. The variance can be calculated from the average of the squared result of the simulations. Finally the equation 3:12 can be used, and the execution continues until finishing all $N_l$ simulations for each level $l$, $0 < l < L$.

The Multilevel technique reduces the error from variance, but not from the bias. However, there is still a need to consider the error from the bias in order to maximise performance and accuracy. Recall equation 3.9 which expresses the two sources of error. They have equal weight on the total error on average, which means that they each contribute half to it on average. Therefore the bias contributes to the error as:

$$\frac{\varepsilon^2}{2} = c_2 e^2$$

(3.13)

For this reason, if the algorithm is required to give a solution with accuracy, or maximum error, of $e$ then the discretisation $\varepsilon$ should be:

$$\varepsilon = \sqrt{\frac{e}{2c_2}}$$

(3.14)

In the next chapter, I will discuss how I implemented the circuit representation and the Multilevel Walk-on-Spheres algorithm on capacitance extraction.
Chapter 4

Implementation

In this chapter, we will discuss how we implemented the standard Monte Carlo Walk-on-Spheres algorithm and then the Multilevel version of the algorithm for electrical capacitance extraction. We begin by explaining how we designed the circuit representation. Next, we explain an early and simpler version of the algorithm that did only the Standard Walk-On-Spheres. After that, we explain how we improved and modified that into the Multilevel version, the challenges it entailed and how they were overcome.

4.1 Circuit Representation

We were interested in demonstrating the Multilevel Walk-on-Spheres in a non-Manhattan geometry circuit, since the most important factor in choosing Walk-on-Spheres over Floating Random Walk is that it works in non-Manhattan geometries. As was referred in section 3.3.1, two circuits composed of two rectangular components, one circuit with rounded corners on the electrodes and the another with ninety-degree corners, the difference in the calculated capacitance could be over 17%. Considering the time it would take to make circuit representation that allows for arbitrary geometries, we chose to limit this work to a geometry where electrodes are composed of adjacent rectangles with possibly rounded corners. We can confidently say the algorithm would work for arbitrary geometries as it has no more restrictions to circuit geometry than Walk-on-Spheres does because the same mathematical premises are respected. This ensures that the developed algorithm will converge for any kind of geometry and not just for an almost Manhattan geometry. Having a simple geometry like the one we chose allowed us to invest more time in the implementation of the algorithm. The exception to this claim is the module that calculates the distance between the particle and the nearby electrodes. However, the program’s implementation is general enough to allow this module to be replaced.

In order to represent rectangles in two-dimensional space, we need only two corners so long as they are opposite to each other. Each of these points has two coordinates. This is enough to represent the rectangle in the circuit space, and calculate if there is a particle collision, which is fundamental to the algorithm.

An electrode can be shaped by multiple rectangles with at least one side adjacent to another rectan-
Figure 4.1: Example of a very simple circuit composed of two electrodes with rounded corners. Each is represented by composing two rectangles and four circumferences.

gle of the electrode. Each rectangle has the identification of the electrode it belongs to, and the voltage of that electrode. Finally, the corners of the rectangle are defined as one-fourth of a circumference. The radius of that circumference is determined by input.

The Figure 4.1 shows an example circuit were it is evident how the electrodes are represented.

4.2 Standard Walk On Spheres

The Multilevel version of the Walk-on-Spheres algorithm is built on the Standard Walk-on-Spheres algorithm, which is why it is fundamentally important to understand it first. We already described generally in Chapter 3 how Walk-on-Spheres can be used for calculating the voltage of one point in a circuit. The first step of this work was to make an implementation of such an algorithm.

The algorithm consists in repeated simulations of Brownian motion, in other words, a particle moving randomly. This movement stops only when the particle collides with an electrode. At that moment, the voltage of the electrode that the particle collided with is recorded. The end result that is wanted is the average of those voltages. Consider that the particle movement occurs in steps 0, 1, ..., N, where the size of each step is the distance between the particle and the nearest electrode. For now, consider the following pseudo-code that calculates the voltage of a single point. We will later go further in depth on how to do the particle steps and calculate the distance to electrodes.

The input to the algorithm is the number of simulations we want to run, followed by the coordinates of the point we want to know the voltage of. Next is ε, which is the discretisation of the simulations, or the distance that is considered to be a collision with an electrode. And finally the last input is the circuit,
Input: `numberOfSimulations, initialPoint, ε, circuit`

forall `numberOfSimulations` do
  `point = initialPoint`
  `electrode = NOELECTRODE`
  while `collision` do
    `d, electrode, collision = distance(point, circuit, ε)`
    `point = particleStep(point, d, collision, electrode)`
  end
  `v = voltage(electrode, circuit)`
  `sum = sum + v`
end

`result = sum/numberOfSimulations`

Output: `result`

Algorithm 1: Pseudo-code for the first version of Walk-On-Spheres algorithm

For each simulation, we consider a particle which is represented by its two coordinates. The distance from this particle to the nearest electrode is calculated. While this particle is not considered to be in collision to an electrode, it will make a particle jump, recheck the distance to the closest electrode and repeat. When the distance to the closest electrode is less than ε, the simulation stops and the voltage of the electrode is saved. Finally, the sum of all voltages is divided by the number of simulations done and we obtain the average.

The function "distance" calculates the distance to the nearest electrode and what electrode it is. It has the responsibility to detect collisions. The function does so by simply iterating over all electrodes, subtracting the current particle coordinates to their edges, and returns the smallest distance. If that distance is smaller than ε, then there is a collision. It also calculates the distance to the end of the circuit and returns that instead if it is closer than any electrode. In the case that the particle is in collision with the edge of the circuit, a reflection step is performed. In that particular case, this function has the added responsibility to calculate the normal vector to the closest point of the end boundary of the circuit, and then pass the responsibility of making the reflection step to the function "particleStep". This function is not efficient for a large number of electrodes as it makes calculations for each one of them, however it is not trivial to improve it. Considering that it has the same behaviour and performance for both Standard or Multilevel Monte Carlo, it does not affect the results and findings of this work.

The following simplified code and accompanying Figure 4.2 explain how the function calculates the distance between a point and the closest object.

The "particleStep" function is responsible for performing the steps of the random path. It calculates the new position of the particle after a jump of size `d` using vector addition. The direction is given by a random number generator configured to give a random angle between 0 < `rndAngle` < 2π. Unless the particle is due to make a reflection jump, in which case the direction is given by the reflected vector calculated by the function "distance". This vector is defined such that the angle of incidence on the boundary is the same as the angle of reflection. The "particleStep" function knows if it should do
Input: point, circuit, $\varepsilon$

$c = \text{circuit.center}$

distanceToCenter = $\sqrt{(\text{point.x} - c.x)^2 + (\text{point.y} - c.y)^2}$

distanceToEnd = circuit.radius - distanceToCenter

dMax = 0

for circuit.electrodes : e do
    for e.rectangles : r do
        $q = \text{getQuadrant}(\text{point}, r)$
        if $q == Q2$ then
            $\text{dist} = p.y - r.1y$
        end
        if $q == Q8$ then
            $\text{dist} = r.2y - p.y$
        end
        if $q == Q4$ then
            $\text{dist} = r.2x - p.x$
        end
        if $q == Q6$ then
            $\text{dist} = p.x - r.1x$
        else
            $\text{dist} = \sqrt{(\text{point.x} - (r.x - \text{radius}).2 + (\text{point.y} - (r.y - \text{radius}).2)) - \text{radius}}$
            /* Q1, Q3, Q7, Q9 respective combinations of r1x, r1y, r2x, r2y not shown */
        end
        if dMax < dist then
            dMax = dist
            electrode = e
        end
    end
end

if dMax > distanceToEnd then
    electrode = OUTERLAYER
    normal.x = (p.x - c.x)/distanceToCenter
    normal.y = (p.y - c.y)/distanceToCenter
end

if dMax < $\varepsilon$ then
    collision = true
end

Output: dMax, electrode

Algorithm 2: Pseudo-code for computing the distance to the nearest electrode.
Figure 4.2: Quadrants of a rectangle and how to calculate in which is the particle.

A random step or a reflection depending on the variable "electrode" which contains information about the closest component or boundary. A particle step will only occur provided that the particle is not in collision already and that the particle is close to an electrode. If the particle is near the outer boundary instead, it performs a reflection step.

**Algorithm 3:** Pseudo-code for the Walk-On-Spheres particle steps

```plaintext
Input: point, d, collision
rndAngle = rand(0, 2π)
if collision then
    return
end
if electrode == OUTERLAYER then
    p.x+ = 2 × d × normal.x
    p.y+ = 2 × d × normal.y
else
    p.x+ = d × cos(rndAngle)
    p.y+ = d × sin(rndAngle)
end
Output: point
```

The size of the step of the reflection is the same as the one the particle would do if there was no circuit boundary in its path, but with a different direction. As a side note, notice that if that distance was zero, the particle would perform a step with size zero as well and thus it would be stuck. To solve this, a
check had to be made to detect these situations, which is not represented in this code. Essentially if the
distance to the circuit boundary is several orders smaller than $\varepsilon$, it performs the step with size $\varepsilon$ instead.

This implementation was only a stepping stone for this work and was not used to any of the presented
results, as it is missing several changes required to implement the multilevel technique. It serves to
explain and introduce the implementation with gradually increasing complexity.

4.3 Multilevel Technique

As discussed in Chapter 3 the algorithm has to run a certain number of simulations at different levels
of discretisation and perform a different estimation for all levels after the first. Having different levels of
discretisation is achieved by using different $\varepsilon$, where $\varepsilon$ is the distance at which a particle is considered
to be in collision with an electrode. However, it is not simply a matter of using the algorithm shown pre-
viously several times with different input. A new estimator is needed, one that calculates the difference
between the results of two discretisation levels for the exact same particle path, which imposed several
changes to the original program.

Recall Equation 3.10 $E[P_L] = E[P_0] + \sum_{l=1}^{L} (E[P_l - P_{l-1}])$, which expresses that an estimator for a
discretisation level $P_L$ can be substituted by the sum of an estimator $P_0$ with the estimated difference
between $P_1$ and $P_2$, summed with $P_2 - P_3$, repeating until $P_{L-1} - P_L$.

The estimator being used is the Walk-On-Spheres algorithm described in the previous section. Meaning
that $E[P_1]$ is equal to the expected value of performing a standard Monte Carlo with discretisation
$l$. And $E[P_l - P_{l-1}]$ is the expected value of the difference between the pay-offs of two standard Monte
Carlo simulations, one with a smaller $\varepsilon$ than the other.

The equivalence expressed by the equation is what fundamentally makes the Multilevel technique
possible. An estimation with a certain discretisation can be replaced by several estimations with different
discretisations. Then, how much computational effort goes into each of those estimators can be adjusted
for a performance increase.

The main difficulties of this work were implementing the algorithm of the previous section and then
modifying it to do what we described. The program had determine the sum of the different estimators,
dealing with the fact that there are two types of estimators, standard and difference between two levels.
At the same time it is running simulations for the estimators of voltage, it has to analyse the variance
and cost of all those simulations in order to calculate an approximation to the optimal number of runs at
each level.

The simplest way to do the new estimator would be to run the standard algorithm discussed previ-
ously twice. Both particles start from the same position and are given the same angles for their steps.
They will perform the same steps, but one simulation has a smaller $\varepsilon$ so they have different stopping
conditions. There are two possible cases that can happen. Let’s call the discretisations $\varepsilon$ and $\varepsilon_2$, where
$\varepsilon_2$ is the smallest or finest.

In the first case, the particles make the exact same path and they collide with an electrode. The
distance to the electrode is smaller than $\varepsilon_2$, and therefore also smaller than $\varepsilon$. Since that is so, both
Figure 4.3: The particle's distance to an electrode is less than $\varepsilon_2$. The path ends and in this case the contribution is zero.

Figure 4.4: The particle's distance to an electrode is less than $\varepsilon$. The voltage of this electrode is recorded and the particle continues. However, soon after the particle gets within $\varepsilon_2$ distance to the same electrode. The path ends and in this case the contribution is zero.

The particles are considered as having collided and both simulations finish. Because they both hit the same electrode, the difference between the voltages is zero and contribution is zero, but important nonetheless, as we are calculating an average expected value. This is shown in Figure 4.3.

In the second case, both particles make the same path at first. At some point, they are closer than $\varepsilon$ to an electrode and the particle is considered to be in collision. However, the distance to the electrode is still higher than $\varepsilon_2$. Therefore the second particle continues its random path until it hits an electrode. It could still hit the same electrode by chance, as shown in Figure 4.4, and that is equivalent to the first case. The most interesting case is when the particle hits a different electrode. The difference between the voltages is non-zero, and there is a contribution. Like before, all these contributions are averaged, divided by the number of simulations ran at this level. Figure 4.5 illustrates an example of this case.

An optimisation that becomes apparent is to simulate only one of those two particles. This particle only stops when within distance $\varepsilon_2$ but records the voltage of the first electrode that is $\varepsilon$ away from it (again these could be the same electrode). Since the two theoretical particles make the exact same path until the moment the distance requirement $\varepsilon$ is met, computational effort can be saved by not actually
Figure 4.5: The particle's distance to an electrode is less than $\varepsilon$. The voltage of this electrode is recorded. The path continues and eventually the particle gets within $\varepsilon_2$ distance to a different electrode. The contribution is the difference between the voltages of those two electrodes.

simulating them both.

In this case, because two collisions have to be detected, the "electrode" variable cannot be used for that. Another difference to the simple standard algorithm that we first implemented is that now the average variance and average cost of one simulation needs to be calculated. The variance of the pay-off is given by the squared average difference between the average voltage and the sum of the contributions (or voltage difference for Multilevel). The cost of a simulation refers to its running time. It is calculated by dividing the running time of all simulations with how many were performed.

The function "distance" now has the added responsibility to detect collisions on both discretisation levels, which is a trivial change.

The challenge here was to have a general code that can execute Multilevel simulations or Standard simulations, yet still be fast and efficient. The particle can collide for one discretisation level and continue, or can collide for both discretisation levels at once if after a jump it is close enough.

The first time the modified Walk-On-Spheres function is called, the number of simulations ran at each level is determined by a parameter from input $N_{\text{firstRun}}$. This quick first run allows to have estimates for variance and cost per level of discretisation, so that the optimal number of simulations per level can be calculated with a reasonable accuracy. In the following calls, the number of simulations already performed is subtracted from the estimation of needed simulations. Each cycle the algorithm adjusts the estimates for variance and cost, recalculates the number of simulations needed. If that number has not been reached, more calls to Walk-On-Spheres are done.

$M$ is a parameter that defines how much smaller the discretisation gets at each level. The other input parameter $\beta$ is the constant associated with the bias error. Both are experimental results. We expect half of the error to come from the discretisation and as such we divide the accuracy accordingly. $N_{\text{SimsNeed}}$ is the estimation of how many walks have to run to reach the intended accuracy, and $N_{\text{SimsDone}}$ the number of simulations that have already ran (per level).
Input: \( \text{numberOfSimulations}, \varepsilon, \varepsilon_2, \text{level}, \text{point}, \text{circuit} \)

\[ v_1 = v_2 = \text{sum} = 0 \]

\[ t_1 = \text{time()} \]

\textbf{for} numberOfSimulations \textbf{do}

\[ p = \text{initialPoint} \]

\[ \text{electrode} = \text{NOELECTRODE} \]

\[ \text{first} = \text{NOCOLLISION} \]

\[ \text{second} = \text{NOCOLLISION} \]

\textbf{while} second! = COLLIDED \textbf{do}

\[ d, \text{first}, \text{second}, \text{electrode} = \text{distance(point, circuit, } \varepsilon, \varepsilon_2) \]

\[ \text{point} = \text{particleStep(point, } d, \text{first}, \text{second}, \text{electrode}) \]

\textbf{if} first == JUSTCOLLIDED \textbf{then}

\textbf{if} level == 0 \textbf{then}

\[ \text{second} = \text{COLLIDED} \]

\[ v_1 = 0; \]

\textbf{else}

\[ v_1 = \text{voltage(electrode, circuit)} \]

\[ \text{first} = \text{COLLIDED} \]

\textbf{end}

\textbf{end}

\[ v_2 = \text{voltage(electrode, circuit)} \]

\[ v = v_2 - v_1 \]

\[ \text{sum} = \text{sum} + v \]

\[ \text{sumSq} = \text{sumSq} + \text{pow}(v, 2) \]

\textbf{end}

\[ t_2 = \text{time()} \]

\[ \text{avgCost} = (t_2 - t_1)/\text{numberOfSimulations} \]

\[ \text{avgVariance} = \text{sumSq}/\text{numberOfSimulations} \]

\[ \text{avgVoltage} = \text{sum}/\text{numberOfSimulations} \]

\textbf{Output:} avgVoltage, avgVariance, avgCost

\textbf{Algorithm 4:} Pseudo-code for the Modified Walk-On-Spheres algorithm
Input: $N_{firstRun}, targetAccuracy, beta, maxLevel, M, Circuit, point$

$DeltaL[maxLevel] = targetAccuracy/(2 \times beta)$

for $l = maxLevel - 1; i >= 0; i --$ do

$DeltaL[l] = DeltaL[l + 1] \times M$

$NSimsNeed[l] = N_{firstRun}$

$NSimsDone[l] = 0$

end

if $Delta[0] > guassianDelta$ then

err: “Starting position of particle is already in collision”

end

for $l = 0; l < maxLevel; l ++$ do

$N[l] = NSimsNeed[l] - NSimsDone[l]$


$NSimsDone[l]++ = N[l]$

end

MPI_AllReduce(avgVoltage, avgLVariance, avgLCost, SUM)

eqSum = 0

result = 0

for $l = 0; l <= maxLevel; l ++$ do

$eqSum += sqrt(var[l] \times cost[l])$

end

for $l = 0; l <= maxLevel; l ++$ do

$NSimsNeed[l] = ceil(eqSum * sqrt(var[l]/cost[l])/(2 \times pow(targetAccuracy, 2)))$

$NSimsNeed[l] -= NSimsDone[l]$

end

dueSimulations = 0

for $l = 0; l <= maxLevel; l ++$ do

$dueSimulations += NSimsNeed[l]$

end

for $l = 0; l <= maxLevel; l ++$ do

$result += avgVoltage[l]$

end

Output: result

Algorithm 5: Pseudo-code for the main program structure
4.4 Gaussian Surface

Our final objective is the extraction of the capacitance. For that we need an integration surface from which we will choose several points. For each of those, we run the algorithm to get the voltages and ultimately calculate the capacitance. This integration surface is calculated automatically by my implementation. There are two input parameters it depends on, the Gaussian $\delta$ and the number of integration points. The $\delta$ defines the distance between the electrode and the integration surface. So long as the surface contains only the electrode we are evaluating, the result is the same no matter the distance. For that reason, the integration surface should be close enough so it does not contain any part of another electrode. As for the number of integration points per electrode, the more points are evaluated, the higher the accuracy.

Then we calculate the perimeter of the integration surface from the coordinates of the circuit rectangles and the Gaussian $\delta$ and then divides it by the number of points to be evaluated. Let us call that result $r$. The algorithm moves along the integration surface, using linear arithmetic to calculate the coordinates of the points equally spaced by distance $r$ if they are on the same side of the surface. If they are close to a corner, then the distance to the corner is subtracted from $r$, the next point calculated from the remainder, and the process continues. Finally, the Walk-on-Spheres algorithm can be called on all those points. This process occurs for all electrodes in the circuit.

4.5 Parallelization

Monte Carlo methods are known for being easily parallelizable. It is just in the same in this case, each simulation is independent from all others and therefore the workload can be distributed without needing to have communication between processes. Communication is what increases the difficulty of the implementation and decreases the performance of the algorithm. However, the entire algorithm cannot run in parallel. Each time the algorithm needs to estimate the variance and the cost again, it does so with the average variance and average cost of all simulations ran so far. Therefore at that point, all processes must communicate. Only then can the number of simulations needed be estimated and the workload divided between them.

Algorithm 5 gives an overview of the entire program. The division of work between processes occurs at all levels, however, the execution does not stop. All processes know how many processes there are in total, and how many simulations to run in total. Therefore a simple division will give the number of simulations each has to run. Only after all simulations of all levels are done do the processes stop and gather the average voltage, variance and cost of each level. This is done using the MPI_AllReduce sum procedure, as mentioned in the Chapter 2. This quick stop in execution is needed to calculate again the number of simulations that are sufficient to reach the target accuracy.

We first had tried to make the division of work and gathering of the data at each invocation of the Walk-On-Spheres function, however the way described before was more efficient as it performed the MPI_AllReduce procedure less often. Each time the processes stop to communicate, performance is
lost. For this reason we tried to minimize the number of times it is done.

4.6 Verification and Validation

In this section, we go over how we validated and assured the correctness of the algorithm, and the conditions of the tests performed to achieve the results presented in the next chapter.

In order to assert the correctness of the algorithm, we make comparisons to the output of the Matlab PDEtool. For these tests, we ran a thousand simulations of Standard Walk-on-Spheres and equally many with equal accuracy of Multilevel. The average voltage from the simulations should be relatively close to the voltage given by the tool, but it is unlikely to be identical because the solution from the PDEtool has significantly less accuracy. Although the accuracy from the PDETool can be adjusted, it takes an rapidly increasing amount of time, and the solution we use here for comparison is as accurate as we could get from it. The expected result of this test is that the estimated voltages follow a Gaussian distribution such that close to 96% of them do not deviate more than the specified accuracy from the average output voltage. Since running time analysis is not a concern for these tests, we used the parallel version so that these tests would not take as long.

The accuracy chosen was 0.0008. What this value means is that the acceptable error to the real voltage is between $0.0008 < \text{Voltage} < 0.0008$. The accuracy was chosen with this value because it is a small enough error that would allow to analyse the algorithm and compare it to the PDETool solution, while not taking longer than a day to run the algorithm a thousand times. This accuracy ensures an error that is at least one order of magnitude smaller than the error of the PDETool.

The most important result of this work is the graph showing the performance increase of the Multilevel technique compared to the Standard Walk-on-Spheres. For this result, we ran ten tests at each increasing accuracy and plotted the average running time of the program for each of those. We did this for both Multilevel and Standard Walk-On-Spheres. All the runs for this result were done without parallelization as to see only the increase the technique gives.

Finally, we show the performance increase of the parallelization of the Multilevel algorithm as the number of processes are increased. The testing environment was a 12-core machine each with hyper-threading (24 total virtual cores). For this last test, we ran the algorithm for the same initial point and accuracy in the order of $10^{-4}$, but with an increasing number of processes. For each of those, the algorithm runs ten times and takes the average of the running time. We compare the running time obtained with the maximum theoretical speed-up of 100% and with a speed-up of 95%. To do this comparison, we divide the running time of the serial algorithm $S_{time}$ with the number of processes. So a perfect speed-up would mean that the parallel running time is equal to the serial running time divided by the number of processes, $P_{time} = S_{time}/N_{Processes}$. A 95% speed-up would be $0.05 \times S_{time} + 0.95 \times S_{time}/N_{Processes}$ meaning that 95% of the workload can be divided and 5% cannot. If the number of processes increased infinitely the running time of the entire algorithm would be equal to the time it takes to compute those 5%.

The initial point for all tests was $(x, y) = (0.6, 0.225)$. This point was chosen for being a non-trivial
case as it is close to two electrodes of different voltages and close to rounded corners. Therefore it shows that the algorithm can handle a non-Manhantan geometry.
Chapter 5

Results

Figure 5.1 shows the circuit on which the tests were ran. The circuit was designed in the PDETool and contains nine electrodes that are composed by one or more rectangles. The electrodes have rounded corners with equal radius. The figure shows also the solution calculated by the tool. As explained before, it calculates a global solution by first generating a mesh that covers the entire circuit. The solution presented gives the voltage of every point in the circuit by averaging the voltages of the nearest mesh points.

Recall that the implemented algorithm is stochastic in nature and therefore gives a different output each time it is ran, while the PDETool uses a deterministic method and thus gives always the same result. The Figure 5.2 shows the behaviour of the output of algorithm running Multilevel ($maxLevel > 0$) for a specific point in the circuit and the difference between the result of the algorithm and the result from the PDETool. The X axis represents the difference between the PDETool result and the output of the algorithm for a thousand runs. The Y axis represents the number of simulations out of a total of one thousand that had an output whose value is within a subset of the X axis. The expected result would be that the output of the algorithm behaves as a random variable that follows a Gaussian distribution with average equal to the correct voltage of that point, and that 96% of the time the output does not deviate from the correct result more than the input accuracy. Considering that the PDETool is less accurate than the algorithm developed, the best method we can do to demonstrate the correctness of the algorithm is to study the behaviour of the output. This figure shows that the output does in fact approximately behave as a Gaussian distribution and that the average of that distribution is close to the PDETool result. The results also show that the output did not deviate from the average result for 95.7% of the times it was executed (43 simulations out of 1000), which further indicates the correctness of the implementation.

Likewise, Figure 5.3 shows the behaviour of the output of the algorithm running standard Walk-On-Spheres ($maxLevel = 0$) and how it compares to the PDETool solution for the same point. Again the results are as were expected, with the output following a Gaussian distribution with an average close to the PDETool solution, which further indicates the correctness of the algorithm. For this set of tests, the output did not deviate from the average result for 95.2% of the times.

The Figures 5.4 and 5.5 shows how the running time of the algorithm increases as the accuracy
Figure 5.1: PDETool: Partial view of the circuit where the tests were performed and the solved voltage. The circuit’s boundary is not included in the view.

Figure 5.2: Histogram of Multilevel simulations showing the average difference between the voltage calculated by the algorithm and the PDETool solution. In red is a fitting Gaussian distribution.
Figure 5.3: Histogram of Standard simulations showing the average difference between the results and the PDETool solution. In red is a fitting Gaussian distribution.

Figure 5.4: Comparison between the running time of Multilevel and Standard Walk-On-Spheres. Logarithmic scale.
required increases for both the Multilevel and Standard Walk-On-Spheres. The results show that the Multilevel technique resulted in a significant performance increase. The Multilevel can reach in less than an hour an accuracy that takes the Standard Walk-On-Spheres three times longer to reach.

As mentioned in the Multilevel section, the complexity of the algorithm should be $O(e^{-2 \log(e)^{-2}})$ for the Multilevel Walk-On-Spheres. Figure 5.6 shows a comparison between the logarithms of the Multilevel running time and accuracy and a linear function obtained using Matlab’s linear regression. The slope of the logarithm of the function is slightly above the slope $-2$ which seems to correspond correctly to $O(e^{-2 \log(e)^{-2}})$.

The scaling of the parallelization of the algorithm is shown in Figure 5.7. For a fixed accuracy the number of processes was increased resulting in a faster running time. It is interesting to note that no matter how many processes are dividing the work, the Multilevel is almost always around 3.5 times faster than the Standard Walk-On-Spheres, with the exception of two outliers. The results show that both benefit equally from being parallelized, which is the behaviour we expected.

Fig. 5.9 and Fig. 5.8 compare the running time decrease of the parallel algorithms as the number of processes increases. The maximum theoretical speed-up possible is 100%. Up to the 12 physical cores that the machine had, the performance is very close to optimal. For the remaining 12 virtual cores, to a total of 24, we can observe a small slowdown where the algorithms still have a speed-up of around 95%. The implementations are highly scalable.
Figure 5.6: Comparison between the logarithm of running time of Multilevel and logarithm of the accuracy, with a linear function.

Figure 5.7: Comparison between the running time of parallel Multilevel and parallel Standard Walk-On-Spheres.
Figure 5.8: Comparison between the running time of parallel Multilevel WoS, a perfect speed-up and a 95% speed-up.

Figure 5.9: Comparison between the running time of parallel Standard WoS, a perfect speed-up and a 95% speed-up.
Chapter 6

Conclusions

In this work we presented a relevant modern problem, capacitance extraction of integrated circuits. We enumerated the existing solutions, explained their advantages or shortcomings and what research has been done in this field. We proposed a new approach that applies the Multilevel technique on the Walk-On-Spheres algorithm, which could have a significant impact on the field and industry.

The objectives of the work were to implement and test such a program that could show the performance improvement of using the Multilevel technique in capacitance extraction. We developed a program capable of performing both Standard and Multilevel Walk-On-Spheres.

The results showed that the implementation of the algorithm is robust and its results accurate. Regarding the performance the tests showed that there was an increase in the running time in the expected order. If the Multilevel runs for an hour it can give a result with an accuracy that would take the Standard Walk-On-Spheres more than three hours to obtain. If left running from one day to the next, the speed-up is expected to reach more than one order of magnitude.

Based on the results, the objective of the work was sufficiently met. However, future work will have to be done to be able to declare Multilevel Walk-On-Spheres as an important research focus. Considering that the major attraction of the Multilevel technique is the fact that it theoretically can be used alongside other variance reduction techniques, the first step would be to show exactly that in practise. If it was shown that an implementation combining Multilevel with another variance reduction technique has a significant performance gain, Multilevel would become a very attractive research focus. A good example would be combining Multilevel and stratified sampling. Intuitively, since they are both variance reduction techniques and thus are reducing the same source of error, we expect that there would be diminishing returns, yet it could still be worth to use both techniques.

In order to show the effectiveness of the Multilevel technique for real scenarios, it would also be necessary to modify it this program to support three-dimensional integrated circuits with millions of components. This requires several contributions, all of which are not trivial. First to design a representation that would allow for arbitrarily shaped electrodes in the circuit. Second to study and implement an efficient algorithm that could calculate what is the closest electrode to any point in the circuit and what is the distance to that electrode. And finally, to develop a tool capable of generating random geometries.
with an arbitrarily large number of components and shapes.
Bibliography


