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

Tiago Silva
tiagodsilva@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

June 2017

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

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. The work presented in this thesis focuses on parasitic capacitances, which are caused by an unintended accumulation of electric charge.

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) [8]. This is a discretisation of the circuit space into a grid of points, for each of which the distance to the nearest electrode is determined. Solving a distance map on rectilinear or Manhattan geometries is simple as it can be done geometrically. For non-Manhattan geometries though, the process gets complex and fast marching is typically used instead [2].

The classical methods would solve the Laplace's equation for the electrostatic potential and then calculate the gradient to obtain the electric field. However 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.

The chosen alternative by academia and industry is stochastic methods. With stochastic methods [2], the calculations are done point-wise rather than matrix-based. A point-wise calculation is intrinsically parallelizable as the calculations for each point do not depend on other results. 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. Stochastic methods are the preferred way to face this problem, namely the Walk-On-Spheres by M. E. Muller [7] and the Floating Random Walk by Le Coz and Iverson [6]. Much of the research on this subject focuses on techniques to reduce the variance of these methods. Variance reduction techniques are of great importance in Monte Carlo methods because a decrease in vari-

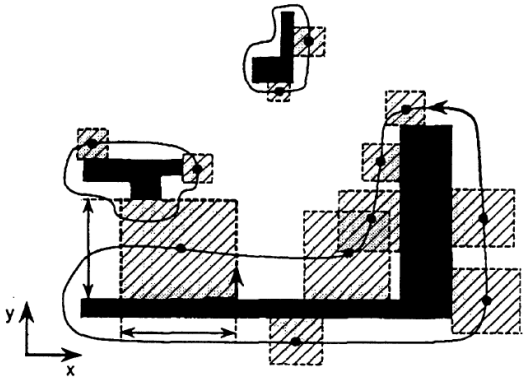

Figure 1: Example of first maximal square boundaries. The line surrounding the black electrodes is the integration surface. The maximal square boundaries are centred on the integration surface [6].

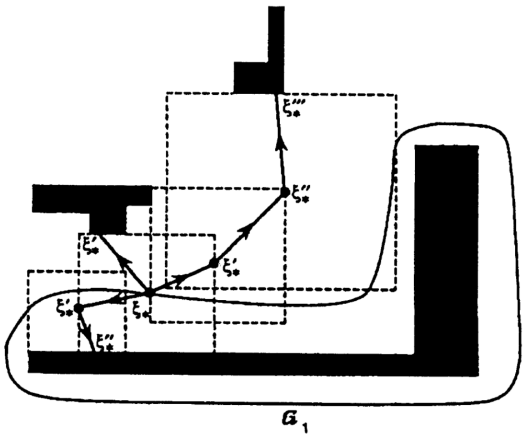

Figure 2: Example of three random paths generated from the same starting point on the surface G1, and their corresponding square boundaries [6].

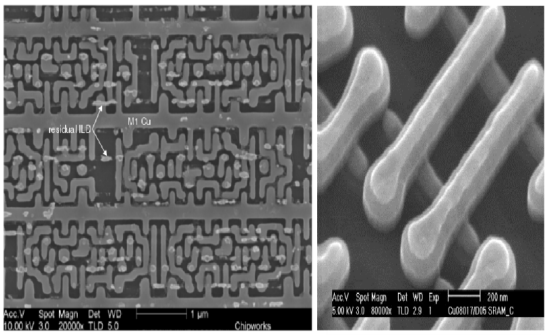

Figure 3: SEM images of commercial SRAM memories. [2].

ance translates an increase in performance [4]. This work 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 [4]. Its application to a capacitance extraction algorithm is a novel idea which promises to further improve the performance of these algorithms. This technique also has the added benefit that it does not exclude the possibility to use other known techniques alongside it [4].

1.2. Contributions of this Work

We developed and tested a parallel Multilevel Walk-On-Spheres algorithm for capacitance extraction. We achieved an increase in efficiency showed by a smaller running time. We also demonstrate the scalability of the parallel algorithm increases as the number of processors increases.

2. Background

2.1. Floating Random Walk

The Floating Random Walk method was first proposed by Le Coz and Iverson [6] as a more efficient solution to calculate the electric field compared to deterministic methods.It is the basis for some 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 large as can be without overlapping with an electrode, so a maximal square boundary always touches at least one electrode. Fig. 1 shows an example. 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 inside an electrode. The generated points define a "walk". The algorithm generates these walks many times for each electrode. Fig. 2 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 of the starting point. The more paths simulated the better the accuracy. With enough points calculated, we can determine the capacitance of the electrode through integration.

Floating Random Walk works only for Manhattan geometries. As shown in Fig. 3 circuit components do not have a perfectly rectilinear shape. Considering these shapes to be rectilinear is an approximation that induces an error in the results. According to the error analysis done in [2], this approximation gives a very significant error. The example shown in Fig. 4 illustrates this. The capacitance calculated on the circuit with sharp corners has an error of 17%, even though the example has only two components.

2.2. Walk-On-Spheres

Walk-On-Spheres is a Monte Carlo algorithm that calculates approximate solutions of some specific boundary value problem for partial differential

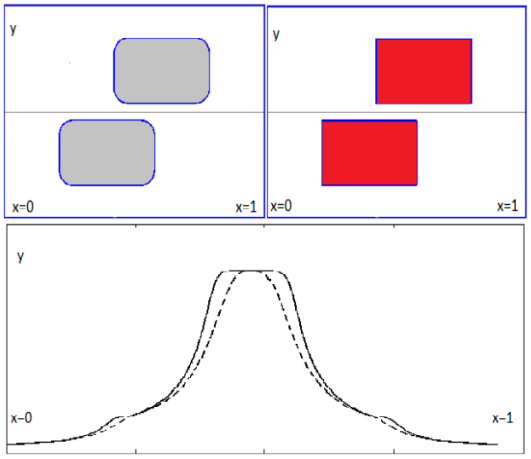

Figure 4: The graph shows the flux of the electric field of the two examples circuits above. The capacitance is proportional to the integral of the areas beneath the lines [2].

equations. It was introduced by M. E. Muller in 1956 to solve Laplace's equation [7]. When applied to capacitance extraction it works on any arbitrary geometry. It works using probabilistic interpretations of partial differential equations by simulating paths of Brownian motion [5]. Based on the starting position of a particle, a spherical boundary is defined, as large 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 ε. With Floating Random Walk, there is no such factor since the square boundaries touch the electrodes in a significant portion, while with Walk-On-Spheres the sphere boundary touches only at an one point. Fig. 5 shows these boundaries and an example of two generated paths. Once again, the average voltage over all electrodes collided with, allows to get an approximation to the voltage of the starting point of the paths. For both of these Monte Carlo algorithms, the end goal is the same. After calculating the voltage of enough points that surround an electrode its capacitance 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 increased with more generated paths. However, there are techniques that can increase the performance of these methods, for example by reducing the variance of the sampling done, less simulations are needed to achieve the same accuracy.

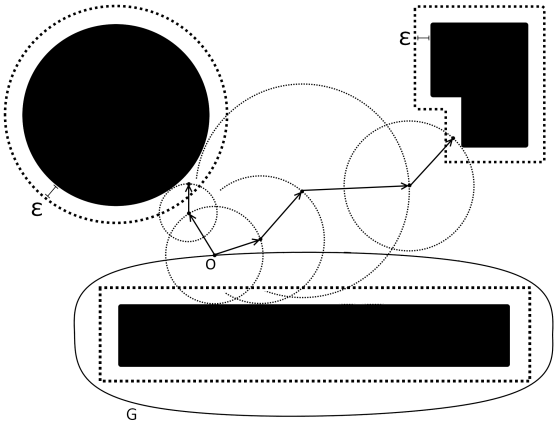

Figure 5: Example of two Brownian trajectories generated by the Walk-On-Spheres algorithm.

2.2.1 Multilevel

Multilevel is a generic technique reduces the variance and therefore the complexity of Monte Carlo methods [3]. Because of the popularity of Monte Carlo, the Multilevel technique sees several applications, many on computational finance, but using it for capacitance extraction is novel. The idea behind it is simple. Instead of doing all Monte Carlo simulations with the same accuracy and performance cost, we can achieve a result that is equally accurate faster if we run a higher number of less accurate but very fast simulations and only a few accurate and slow simulations. Effectively Multilevel is looking at different levels of accuracy as an optimisation problem: "How many simulations should we run for each accuracy level, such that the least amount of time is used to reach an accuracy goal?". The problem is that this optimisation has to be solved very quickly. It is not useful to invest too much time in getting the exact number of simulations that are optimal when that time could be better used running the actual simulations, so an approximation to it is better. The algorithm starts by doing a fixed number of simulations for each level, then determining the average variance and average cost of a simulation at each level. Only then, knowing the cost and benefit of a simulation at each level, an approximate solution can be calculated. 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 [4]. 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 and the performance increase becomes more significant. Monte Carlo path simulations gives us the expected value of a variable, solution to a stochastic differ-

ential equation. It was shown in [9] [1], that the mean-square-error of such estimate has an asymptote of

$$e^2 = \frac{c_1}{N} + c_2\varepsilon^2, \quad (1)$$

where c_1 and c_2 are positive constants, N the number of simulations and ε 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. If we use standard Monte Carlo to determine an expected value 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 $O(e^{-3})$. While using Multilevel the same accuracy can be achieved with a complexity cost of $O(e^{-2}(log(e)^{-2}))$, provided that we have a good approximation to the constants c_1, c_2 [4]. The Multilevel technique considers a sequence, $l = 0, 1, ..., L$, where for each l the discretisation ε and the number of simulations decreases. The simulations on the last levels, with very small discretisation, are very computationally expensive yet yield a very small error. For the simulations on the first levels the accuracy gain is considerably less, but the cost is also much smaller. The simplified idea of Multilevel is to achieve the same accuracy of N simulations with discretisation L, running instead many simulations on the faster levels and a few simulations on the most costly levels, for a smaller total computational cost. Each simulation gives a result, called the payoff P. P_l denotes the approximation to P and $E[P_l]$ is the estimator of P. Then, the Multilevel technique can be summed up as:

$$E[P_L] = E[P_0] + \sum_{l=1}^{L}(E[P_l - P_{l-1}]) \quad (2)$$

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. The left term $E[P_0]$ is calculated using standard Monte Carlo simulations just like $E[P_L]$ is, except at a rougher discretisation. The estimator used is the average pay-off of N_0. The right term of the sum, the estimator for $P_l - P_{l-1}$, is determined differently. For two simulations with same conditions and same randomly generated values, two pay-offs are obtained, one with discretisation ε_l and the other with ε_2. The difference between those two pay-offs is the contribution to the estimator. There are $L - 1$ estimators total, each calculating the average of the differences between two simulations with different discretizations. The final result is determined from the sum of the values from these estimators. The optimal number of simulations at each level can be known by solving the optimisation problem:

$$minimise \sum_{i=0}^{L} N_l C_L \text{ such that } e^2/2 = \sum_{i=0}^{L} \frac{V_l}{N_L} \quad (3)$$

Which gives the formula [4]:

$$N_l = \left\lceil \frac{1}{2e^2}\sqrt{\frac{V_l}{C_l}} \sum_{i=0}^{L} \sqrt{V_i C_i} \right\rceil \quad (4)$$

Where N_l is the number of simulations for level l, e is the maximum allowed error, V_l is the variance of a simulation on level l, C_l the cost, and L the most accurate discretisation level. Recall that 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 4 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. Recall equation 1 which expresses the two sources of error. They have equal weight on the total error which means that they each contribute half to it on average. Therefore the bias contributes to the error as:

$$\frac{e}{2} = c_2\varepsilon^2 \quad (5)$$

For this reason, if the algorithm is required to give a solution with maximum error e then the discretisation for the level L should determined by:

$$\varepsilon = \frac{e}{2c_2} \quad (6)$$

3. Implementation

3.1. Circuit Representation

We were interested in demonstrating the Multilevel Walk-on-Spheres in a non-Manhattan geometry circuit, since we chose Walk-on-Spheres over Floating Random Walk because it works in non-Manhattan geometries. However, 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. In order to represent rectangles in two-dimensional space, we need only the coordinates of two corners so long as they are opposite to each other. An electrode can

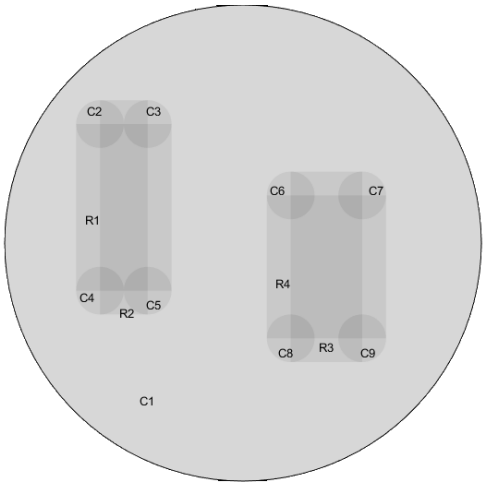

Figure 6: Example of a very simple circuit with rounded corners.

be shaped by multiple rectangles. 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 as well. The Figure 6 shows an example circuit were it is evident how the electrodes are represented.

3.2. Standard Walk On Spheres

We already described generally how Walk-on-Spheres can be used for calculating the voltage of a 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. This movement stops only when the particle collides with an electrode, then the voltage of that electrode is recorded. The output is the average of those voltages. 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 the discretisation. And finally the last input is the circuit, a collection of electrodes. For each simulation, we consider a particle which is represented by its two coordinates. The distance from this particle to the nearest electrode is determined. If this distance is less than ε, the simulation stops and the voltage of the electrode is saved. Otherwise, the particle will move that same distance in a random direction, recheck the distance to the closest electrode and repeat. Finally, the sum of all voltages recorded is divided by the number of simulations done. The function "distance" calculates the distance to the nearest electrode and records what electrode it is. It also detects collisions by checking if the distance is smaller than ε. It also calculates the distance to the end boundary of the circuit and returns that instead if it is closer than any electrode. It works by iterating

Input: $numberofSimulations$, $initialPoint$, ε, $circuit$
for $numberOfSimulations$ **do**
  $point = initialPoint$
  $electrode = NOELECTRODE$
  while $!collision$ **do**
    $d, electrode, collision =$ distance$(point, circuit, \varepsilon)$
    $point =$ 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

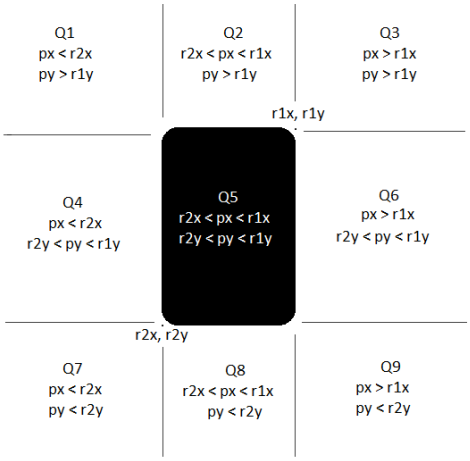

Figure 7: Quadrants of a rectangle and how to calculate in which is the particle.

over all electrodes and calculating the distance between the particle and their sides. This function is not efficient for a large number of electrodes, 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. Fig. 7 helps explain how the function works. Depending on which quadrant the particle is in, the distance is calculated to the side or rounded corner of the rectangle. The "particleStep" function is responsible for performing the random steps. 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 $< 2\pi$. Unless the particle is due to make a reflection jump, in which case the direction is given by the a reflecting

Input: $point, d, collision$
$rndAngle = rand(0, 2\pi)$
if $collision$ **then**
  return
end
if $electrode == OUTERLAYER$ **then**
  $p.x- = 2 \times d \times normal.x$
  $p.y- = 2 \times d \times normal.y$
else
  $p.x+ = d \times cos(rndAngle)$
  $p.y+ = d \times sin(rndAngle)$
end
Output: $point$

Algorithm 2: Pseudo-code for the Walk-On-Spheres particle steps

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. This implementation was only a stepping stone for this work and was not used to any of the presented results. It serves to explain and introduce the implementation with gradually increasing complexity.

3.3. Multilevel Technique

The algorithm has to run a certain number of simulations at different levels of discretisation. It also has to perform a different estimation for all levels after the first. Having different levels of discretisation is achieved by using different ε, where ε is the distance at which a particle is considered to be in collision with an electrode. However, it is not simply a matter of running the previous algorithm 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 function. Recall the Equation $E[P_L] = E[P_0] + \sum_{l=1}^{L}(E[P_l - P_{l-1}])$. $E[P_l]$ is equal to the expected value of performing a standard Walk-On-Spheres run with discretisation l. While $E[P_l - P_{l-1}]$ is the expected value of the difference between the pay-offs of two standard Walk-On-Spheres simulations, one with a smaller ε than the other. The simplest way to implement this new estimator would be to run the standard Walk-On-Spheres twice. Both particles start from the same position and are given the same angles for their steps. They go along the same path, but one simulation has a smaller ε. There are three possible cases that can happen. Let's call the discretisations ε and ε_2, where ε_2 is the finest. In the first case, the particles take the exact same path and they collide with an electrode. The distance to the electrode is smaller than ε_2, and therefore also smaller than ε. Both particles have collided. Because they hit the same electrode, the difference between the voltages is zero. This case is shown in Fig. 8. In the second case, both particles take the same path at first. At some point, they are closer than ε to an electrode. However, the distance to the electrode is still higher than ε_2. Therefore the second particle continues until it hits an electrode. It could still hit the same electrode by chance, as shown in Fig. 9, 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. All these contributions are averaged by the number of simulations ran at each level. Fig. 10 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 ε_2 but records the voltage of the first electrode that is ε away from it. Since the two theoretical particles make the exact same path until the moment the distance requirement ε is met, computational effort can be saved by not actually simulating them both. The challenge here was to have a general code that can execute Multilevel simulations or Standard simulations, yet still be fast and efficient. The average variance and average cost of each simulation also needs to be determined. The variance of the pay-off is given by the averaged squared sum of the voltages and the cost of a simulation refers to its running time. It is cal-

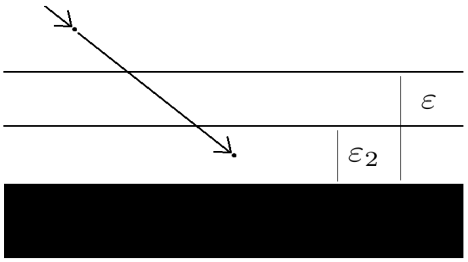

Figure 8: The particle is within ε_2 distance to an electrode. In this case the contribution is zero.

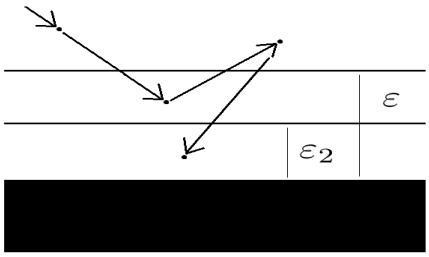

Figure 9: The particle is within ε distance to an electrode so its voltage is recorded. However, soon after the particle gets within ε_2 distance to the same electrode. The contribution is zero.

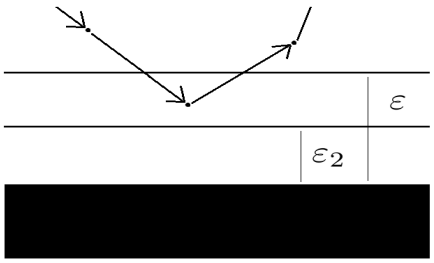

Figure 10: The particle is within ε distance to an electrode so its voltage is recorded. The particle continues and eventually gets within ε_2 distance to a different electrode. The contribution is the difference between the voltages of those two electrodes.

culated by dividing the running time of all simulations of a level with how many were performed. 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 $NfirstRun$. 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. After each cycle the algorithm adjusts the estimates for variance and cost and recalculates the number of simulations needed. If the new number of simulations has not been reached, Walk-On-Spheres is called again.

3.4. Gaussian Surface

The final objective of this method is the extraction of the capacitance. For that an integration surface is need from which several points are chosen. For each of those, the algorithm runs to get the voltages and ultimately calculate the capacitance. The integration surface is calculated automatically by our implementation. There are two input parameters it depends on, the Gaussian δ and the number of integration points. The δ 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 δ 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

Input: $numberofSimulations, \varepsilon, \varepsilon_2, level,$
$point, circuit$
$v1 = v2 = sum = 0$
$t1 = time()$
for $numberofSimulations$ **do**
  $p = initialPoint$
  $electrode = NOELECTRODE$
  $first = NOCOLLISION$
  $second = NOCOLLISION$
  while $second! = COLLIDED$ **do**
    $d, first, second, electrode =$
     distance$(point, circuit, \varepsilon, \varepsilon_2)$
    $point =$
     particleStep$(point, d, first, second, electrode)$
    if $first == JUSTCOLLIDED$ **then**
      if $level == 0$ **then**
        $second = COLLIDED$
        $v1 = 0;$
      else
        $v1 =$ voltage$(electrode, circuit)$
        $first = COLLIDED$
      end
    end
  end
  $v2 =$ voltage$(electrode, circuit)$
  $v = v2 - v1$
  $sum = sum + v$
  $sumSq = sumSq + pow(v, 2)$
end
$t2 = time()$
$avgCost = (t2 - t1)/numberOfSimulations$
$avgVariance =$
 $sumSq/numberOfSimulations$
$avgVoltage = sum/numberOfSimulations$
Output: $avgVoltage, avgVariance, avgCost$

Algorithm 3: Pseudo-code for the Modified Walk-On-Spheres algorithm

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. We did not implement the integration of all the calculated points. The main focus stayed on the performance analysis of calculating the voltage of a single point.

3.5. Parallelization

Each simulation is independent from all others so the workload of running them can be distributed. 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 equally between them.

3.6. Verification and Validation

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 PDEtool, but it is unlikely to be identical because the tool has significantly less accuracy. 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 this 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 $-0.0008 < error < 0.0008$. 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 showed the performance increase of the parallelization of the Multilevel algorithm as the number of processes are increased. 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. 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.

4. Results

Fig. 11 shows the circuit on which the tests were ran. The figure shows also the solution calculated by the PDETool. 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 Fig. 12 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 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

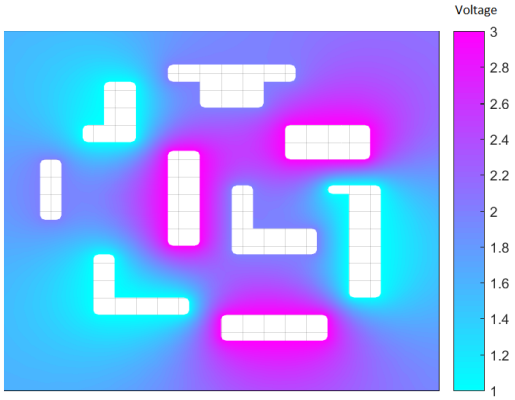

Figure 11: PDETool: Partial view of the circuit where the tests were performed and the solved voltage. The circuit's end is not included in the view.

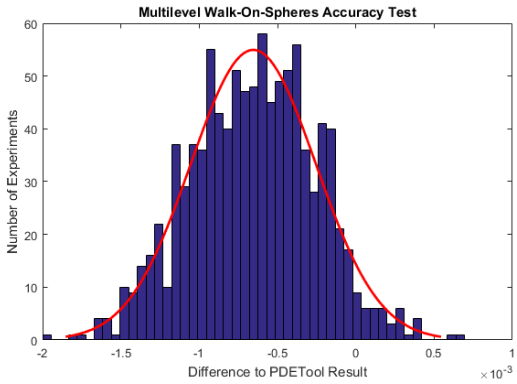

Figure 12: Histogram of Multilevel simulations showing the average difference between the results and the PDETool solution. In red is a fitting Gaussian distribution.

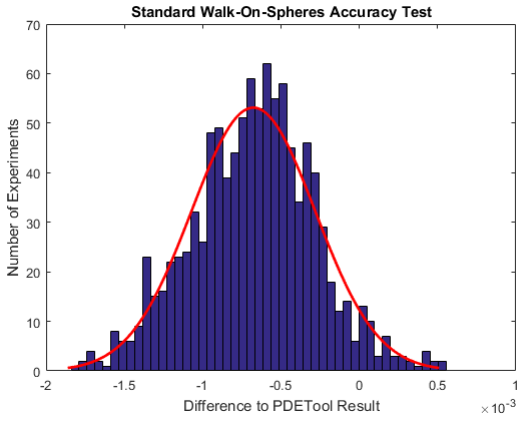

Figure 13: Histogram of Standard simulations showing the average difference between the results and the PDETool solution. In red is a fitting Gaussian distribution.

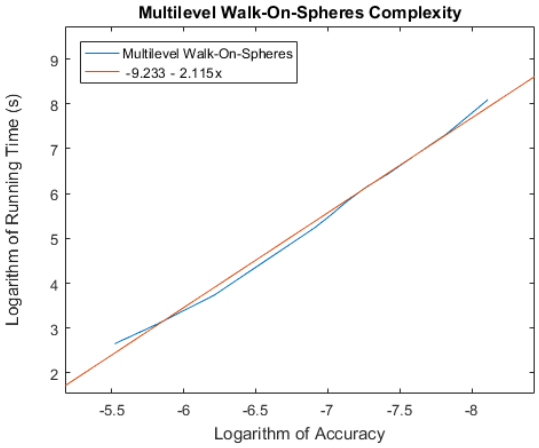

Figure 15: Comparison between the logarithm of running time of Multilevel and logarithm of the accuracy, with a linear function.

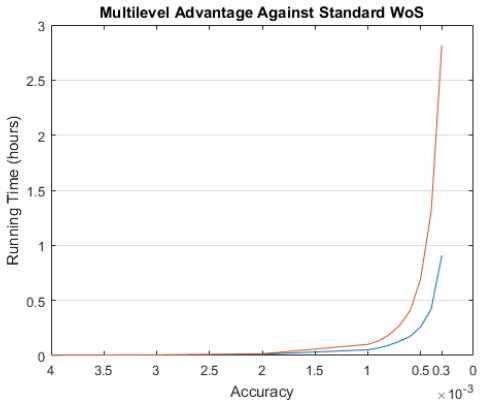

Figure 14: Comparison between the running time of Multilevel and Standard Walk-On-Spheres. Linear scale.

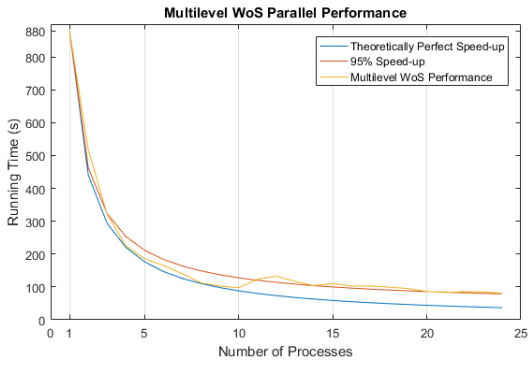

Figure 16: Comparison between the scaling performance of the parallel Multilevel implementation, the maximum theoretical speed-up and a 95% speed-up.

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, Fig. 13 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 Fig. 14 shows how the running time of the algorithm increases as the accuracy required increases for both the Multilevel and Standard Walk-On-Spheres. The results show that the Multilevel technique resulted in a significant performance increase compared to Standard. 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. Fig. 15 shows a comparison between the logarithm of the Multilevel running time and logarithm of the accuracy, and a linear function obtained using Matlab's linear regression. The slope 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 Multilevel algorithm is shown in Fig. 16. The figure compares the running time decrease of the parallel algorithms as the number of processes increases, with the maximum theoretical speed-up possible (100%) and a speed-up of 95%. For this test the algorithm was run multiple times with a fixed accuracy and with an increasing number of processes. Up until the 12 physical cores the machine had, the scalability is near perfect. The Standard Walk-On-Spheres demonstrated the same scalability.

5. Conclusions

In this work we presented a relevant modern problem, capacitance extraction of integrated circuits. We enumerated the existing solutions, explained their advantages or short comings 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 reliable. Regarding the performance the tests showed that there was an significant increase in the running time. 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 speedup is expected to reach at least one order of magnitude.

Based on the results, the objectives of the work were achieved. 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 very attractive. A good example would be combining Multilevel and stratified sampling. It would also be important 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.

Acknowledgements

I would like to thank my family for their unfaltering 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 Juan A. Acebrón and to Francisco Bernal for their readiness to help and teach me.

References

[1] V. Bally and D. Talay. The euler scheme for stochastic differential equations: error analysis with malliavin calculus. *Mathematics and computers in simulation*, 38(1-3):35–41, 1995.

[2] F. Bernal, J. A. Acebrón, and I. Anjam. A Stochastic Algorithm Based on Fast Marching for Automatic Capacitance Extraction in Non-Manhattan Geometries. *SIAM Journal on Imaging Sciences*, 7(4):2657–2674, dec 2014.

[3] K. A. Cliffe, M. B. Giles, R. Scheichl, and A. L. Teckentrup. Multilevel Monte Carlo methods and applications to elliptic PDEs with random coefficients. *Computing and Visualization in Science*, 14(1):3–15, aug 2011.

[4] M. Giles. Multi-level Monte Carlo path simulation. *Operations Research*, 56(3):607–617, 2008.

[5] S. Kakutani. Two-dimensional Brownian motion and harmonic functions. *Proceedings of the Imperial Academy*, 20(10):706–714, 1944.

[6] R. Le Coz, YL and Iverson. A stochastic algorithm for high speed capacitance extraction in integrated circuits. *Solid-State Electronics*, 35(7):1005–1012, 1992.

[7] M. Muller. Some Continuous Monte Carlo Methods for the Dirichlet Problem. *The Annals of Mathematical Statistics*, 27(3):569–589, 1956.

[8] D. T. Paris and F. K. Hurd. *Basic electromagnetic theory*. 1969.

[9] D. Talay and L. Tubaro. Expansion of the global error for numerical schemes solving stochastic differential equations. *Stochastic analysis and applications*, 8(4):483–509, 1990.