Least-Squares Monte Carlo Simulation and High Performance Computing for Solvency II Regulatory Capital Estimation

A thesis submitted to The University of Manchester for the degree of Master of Philosophy in the Faculty of Humanities.

2013

Georgios Dimitrakopoulos

Manchester Business School

Contents

Word count: 21,958

List of Tables

List of Figures

The University of Manchester

Georgios Dimitrakopoulos

Master of Philosophy

Least-Squares Monte Carlo and High Performance Computing for Solvency II Regulatory Capital Estimation

September 2013

ABSTRACT

Financial institutions stand at the edge of what is called a dynamic environment. Acting fast is a vital requirement of this environment. On the top of that, changes at the regulations [31, 32] make institutions' obligations more complex and compute intensive than ever.

Financial researchers try constantly to create new models and improve the existing ones to reduce the complexity and make them simpler and more efficient. In addition, computer scientists have made a significant progress in the acceleration of complex algorithms using parallel and distributed systems. High Performance Computing techniques are used more often to meet the speed requirements that have been set by the modern economic situations.

In this thesis we will present a state-of-the-art financial method, called Least-Squares Monte Carlo. This method is presented to reduce the complexity of a more complex method called Stochastic-on-Stochastic valuation. New regulations have introduced by the European Commission for the insurance companies and this algorithm can fulfill the requirements of these regulations faster and accurately.

Furthermore, the utilization of the High Performance Computing techniques will reveal the power of the modern hardware architectures and programming languages as they can be able to accelerate the performance of a complex financial application by multiple times.

Declaration

Copyright

Dedication

I dedicate this thesis to my parents, Nikolaos Dimitrakopoulos and Antonia Nikolopoulou, and my sister, Dimitra Dimitrakopoulou, for their continuous support, their love and their encouragement since the beginning of my academic life.

Acknowledgment

The completion of this thesis gives me the pleasure to thank people for whom I am deeply grateful. Firstly, I would like to thank Prof. Ser-Huang Poon, first supervisor of this thesis. Under her guidance I had the opportunity to wade into cutting-edge issues of modern research in the fields of IT and finance and particularly Monte Carlo Simulation, Options Pricing and High performance Computing techniques and parallel implementations. Her observations, couched in lengthy and constructive discussions, governing the underlying research approach outlined in this thesis. In addition, I need to thank Prof. Ser-Huang Poon for the full encouragement and support that provides to the graduate students working with her, one of whom I had the good fortune to be too. The spirit of equality and solidarity that characterizes her in her relations with her graduate students and her overall attitude as a university teacher are for me constant standards and motivations and yet the strongest reinforcement of my faith in the academic institutions.

In addition, I warmly thank Prof. John Keane, second supervisor of this thesis and my advisor Mr. Erik Vynckier for their constructive comments and for their interest and expectations with which they approached this research effort. Additionally, I would like to thank Mr. Erik Vynckier for the opportunity to work at SWIP's offices in Edinburgh and his support since the first moment of my participation as a research fellow in the Marie Curie Initial Training Network.

Finally, I would like to thank NAG for their advices and technical support, and particularly Chris Armstrong and Jacques Du Toit (both from NAG) for providing codes, many helpful comments and suggestions.

Chapter 1

Introduction

1.1 Introduction

The burgeoning growth of the international banking and insurance sector in conjunction with the fast and ever-changing economic system has increased the need for both quick and accurate decisions. These decisions require the processing and analyzing of large volumes of data combined with the execution of economic models and algorithms. This data processing and algorithm execution are data - and compute - intensive processes respectively which require a reasonable amount of time for their completion.

Due to the instability of the financial markets in recent years, the banking and insurance institutions are trying to optimize and accelerate these processes to remain competitive. The achievement of this objective requires knowledge derived from two different scientific fields, finance and computer science. Regarding the financial part great emphasis has been given to finding new algorithms and economic models or optimizing the existing ones in order to reduce their complexity while maintaining the same level of accuracy. An emerging particular compute-intensive problem that insurance companies face recently is the calculation of the so called Solvency Capital Requirement.

As the insurance industry is moving from a direct supervisory control system in a more liberalized environment, it requires new control systems and risk management. Moreover, the monitoring authorities need to use improved and more sophisticated techniques for the supervision of insurance companies. Since these institutions are major investors, their credibility has a great impact on the financial stability. The main reference point for an insurance company is its solvency or its financial strength (Sandstrom, 2006).

The basic obligations of an insurance company are the expected underwriting requirements and its associated expenses which are often calculated by actuarial methods. However, these calculations are nothing more than estimates that incorporate probability of errors. For the protection of the policyholders of an insurance product or contract and to ensure the stability of the financial market, insurance companies are required to have in reserve a certain amount of additional assets, called “Solvency Capital Requirement” introduced by the European Union (EU) in the Solvency II regulatory framework.

The calculation of the SCR may require, in principle, stochastic-on-stochastic or nested simulation. This process demands a huge amount of calculation and simulation in order to produce an accurate estimation. This problem has created the need for a more sophisticated algorithm to reduce the computational effort. The Least-Squares Monte Carlo Simulation (LSMC) is proposed as an efficient solution to this problem as it could reduce the number of simulations combined with a least-squares regression. As the financial products have become more and more complex and the volume of the transactions increases, the acceleration achieved from the improvement of the financial algorithms was proven insufficient. This continuously growing need for acceleration has led insurance companies to utilize state-of-the-art computational platforms. High Performance Computing (HPC) platforms combined with parallel programming languages are able to exploit the characteristics of the parallel architectures in order to maximize the acceleration of the financial algorithms.

Parallel programming is defined as the practice of using a large number of co-operating processors, which communicate with each other in order to solve, quickly, compute-intensive problems. Parallel programming is evolving rapidly into a major area of computer science. It can be already seen that the prediction made by Mousa in 2005, that it is possible that in the future years this will become so large and powerful that most of the research conducted in the fields of design and analysis of algorithms, biomedical applications, computer architectures and financial applications will be in the context of parallel calculation, proved true.

The potential of HPC has already been proved in multiple scientific fields but it is still under-utilized for financial applications. One reason for this is that almost all financial institutions hold legacy programs that were developed some decades ago. This means that they must re-write all these codes in order to take advantage of the characteristics of parallel architectures. In addition, the evolution of HPC platforms have resulted in multiple hardware and software solutions. Many-core and multi-core architectures are available and multiple programming models created for each of these platforms. Financial institutions should be able to find the efficient combination of software and hardware that suits their needs, their investment policy and will maximize the acceleration considering the specific characteristics of different financial applications. This will result in an improvement of their risk-management, their trading policy and other internal processes. This thesis presents a comparison of multiple programming models using different parallel architectures in order to evaluate the performance of the LSMC Simulation that can be used for the calculation of the Solvency Capital Requirement under the Solvency II Regulatory Framework.

1.2 Thesis Outline and Contributions

The structure of the thesis is as follows:

In chapter 2 we present the financial problem for insurance companies following introduction of new EU regulations. In addition, a brief description of the Monte Carlo Simulation will be given as well as a variance reduction technique that can improve the accuracy of the simulation.

In chapter 3 we will present the Least-Squares Monte Carlo Simulation (LSMC) for American Options and show how it can solve the problem described at chapter 2. The main decisions regarding the implementation will be discussed and in order to reveal their importance and their impact on estimation accuracy.

In chapter 4 the application of the algorithm will be presented for the valuation of a European Put Option, an Asian Option and a Barrier Option in order to benchmark the accuracy of the LSMC estimates. This is one of the contributions of this thesis where the valuation of complex products, like Asian options, is done under the context of LSMC. The results of this application will be discussed and analyzed to show the success and the drawbacks of the algorithm for different products.

The second part of this thesis consists of the HPC discussion and analysis. In chapter 5 the parallel CPU architectures and their programming model OpenMP will be described. In chapter 6 an analysis and description of the characteristics of the parallel GPU architectures will be discussed. Chapter 6 also contains a description of the CUDA programming model and the OpenACC programming model that can be used to exploit at the parallel GPU attributes.

In chapter 7 we present the results of the multiple implementations, the optimization techniques used for the acceleration of the LSMC algorithm and the pros and cons, of each programming model and parallel architectures will be analyzed. This is the main contribution of this thesis as the only discussion in the literature regarding the implementation of LSMC using high performance techniques has been done from Abbas-Turki et al. [34] where they price European and American Options using CPU and GPU clusters. They proved that they can accelerate their CPU implementation up to 10 times depending on the number of the simulation paths and and the dimension of the problem. The main difference between the implementation of this thesis and their implementation is that in this thesis we describe the application of LSMC in the context of Solvency II and various programming models for GPUs are compared by discussing optimization techniques that can be used to accelerate serial implementations. Our results extend their findings that GPUs can accelerate Monte Carlo simulation.

In chapter 8 the finance results and the HPC implementations will be summarized and future work description for the optimization of the algorithm in both scientific fields.

Chapter 2

Literature Review

In the first part of this thesis LSMC method is presented. This method was first presented by Longstaff and Schwartz [21] in the context of pricing American options. This approach can be interpreted as a way to estimate the value of a function of dynamic programming with linear regression, with the help of basis functions. The potential of this method has already been proved [21] and many attempts had tried to optimize this method in order to improve the estimate accuracy. However, nowadays, insurance companies are turning their interest to this method as it can be used to calculate the Solvency Capital Requirement.

2.1 The Solvency II Regulatory Framework

Solvency II is the new EU directive to regulate capital adequacy requirements of insurance companies and it is expected to be applied at the end of 2016 [24]. Solvency II introduces a new system to calculate the capital requirements for every EU member, which will replace the existing system and the insurance laws of the member states, and will adopt risk management techniques, corporate governance and transparency, which are necessary for the proper functioning of the market and the protection of the policyholders. It will also carry out more effective comparisons between companies within a state as well as between companies located in different states.

The first pillar, the quantitative solvency requirements, includes all relevant rules that an insurance company must follow to form its technical reserves. It will also regulate the investments, determine the assets and the quality of capital that will provide the required solvency for insurance companies.

The key terms included in the first pillar are the Solvency Capital Requirement (SCR) which is the capital an insurance company will hold in order to avoid bankruptcy, with a confidence level of 99.5% for a one year time horizon, and the Minimum Capital Requirement (MCR), which is the lower limit of capital that should be held by an insurance company in order to avoid supervisory intervention and possible revocation of operation.

The second pillar establishes quality standards for solvency, i.e. the principles of internal control activities on which risk evaluation will be based (corporate governance and risk management system) and the framework for conducting the Own Risk and Solvency Assessment (ORSA), which refers to the set of procedures used to identify, evaluate, monitor, manage and report the risks that an insurance company would face now and in the future and to ensure the necessary and ongoing solvency of the firm.

Especially in the field of risk management, reference is being made to all the main areas of risk faced by an insurance company. These include market risk, insurance risk, operational risk and counterparty risk. These risks are quantified in pillar one. In addition, reference is being made to other risks such as liquidity risk and reputational risk which are mostly dealt with qualitatively.

The second pillar also defines the general framework of the internal audit operation, the actuarial operation and outsourcing. According to the directive, supervisory authorities may impose additional capital requirements if they believe that a firm's governance system is inadequate.

The third pillar defines the requirements of transparency and publication of the data of an insurance company. Firms should create two reports to different audiences. The first will only be sent to supervisors and will include confidential information regarding the operation of the firm; the second will be made publicly available and will contain information on solvency and financial conditions.

2.2 Variable Annuities

In recent years, insurance companies offer flexible products that combine multiple investment options with guarantees in order to attract more customers and also to benefit from the long-term positive trends that some indexes might have.

A characteristic example of this kind of products is the Variable Annuity insurance contracts which combine investment options with guarantees that are available to the policyholder at certain policy anniversaries. The level of the guarantees depends on the age of the policyholder, the lapse rate, the profit/loss of the investments etc.. The guarantees can be grouped in five main categories:

- Guaranteed Minimum Death Benefits (GMDB).
- Guaranteed Minimum Living Benefits (GMLB).
- Guaranteed Minimum Accumulation Benefits (GMAB).
- Guaranteed Minimum Income Benefits (GMIB).
- Guaranteed Minimum Withdrawal Benefits (GMWB).

In order to give a more detailed description of these guarantees we will discuss the GMWB which is the most used. The following description will be based in the variable annuity type of contract that is presented and discussed by Ledlie M., C., in Variable Annuities [35].

We assume that the policyholder is 65 years old and he has chosen to activate the GMWB option since the beginning of the contract. The activation of this option costs to the policyholder 1% of the total account value. This cost will not be applied if at some point in time the policyholder decide to deactivate the GMWB option.

GMWB option offers to the policyholder a guaranteed payment of 5% of the total account value per year when the policyholder is older than 65 years old. This guaranteed payment is called 'guarantee base' and it is set according to the aggregated amount of money given by the policyholder. This guarantee base can change subject to the performance of the investments associated with the contract for the next ten years after his 65 birthday. Then, he will receive the constant pre-specified percentage of this fund value until he dies. For the following description we will assume that the total aggregated amount is 100,000 pounds and every year the fund value is invested by 40% at zero coupon swaps with ten years duration and 60% in equity index.

In order to calculate the liabilities of this product, policyholder's life expectancy is the first issue that needs to be addressed. Although a stochastic model should be used for the calculation of the uncertainty of the mortality rates, it can be assumed that these are deterministic and the policyholder cannot be older than 120 years old. This means that Monte Carlo simulations should simulate 55 years into the future considering that the policyholder is 65 years old. Of course this is only an assumption and this area is an open research subject.

For each of the 55 years we need to calculate the fund value, the income level and the guarantee base. These values are interlinked and stochastic, making the calculation of the liabilities complex and compute intensive.

The simulation of the equities and bonds price should be performed in order to calculate the level of the fund at each year after policyholder's 65th birthday (annuitisation). The following equation describes the behaviour of the find value FV_i in year i after annuitisation:

$$FV_i = \max\left(\left(FV_{i-1} - I_{i-i}\right)\left(1 + R_i\right), 0\right)$$

where

$$R_i = xE_i + yB_i - \nu - \lambda,$$

where x and y are the bond and equity investment percentages which in our case are 40% and 60%. E_i and B_i shows the simulated prices of the equities and bonds. I_{i-1} shows the income level at year $i-1$, ν represents administration fees and λ is the 1% additional fee we discussed above and applies when the GMWB option is activated.

When the fund value is defined the guarantee base should be calculated. Each year after annuitisation the guarantee base can be changed depending on the performance of the investments. The following example will make this concept clear. The fund value is 100,000 pounds when the policyholder is 65 years old. If at the end of the next year the fund value is 110,000 pounds then the guarantee base will also be higher, 110,000. This is applied only for ten years after annuitisation. Any change at the fund value after ten years will not have any impact to the level of the guarantee base. Also the change of the guarantee base cannot be higher than 15% each year. This means that if the investments increase the fund value by 20% only 15% will be applied to the guarantee base. In addition, the guarantee base can be the same or higher compared to the previous year. This is in accordance with the 5% minimum income of the initial value of the fund. This can be expressed in the following equation:

$$GB_i = I\left(i \leq \gamma\right) \min\left(1.15xGB_{i-1}, \max\left(GB_{i-1}, F_i\right)\right) + I\left(i > \gamma\right) GB_{i-1}$$

When the guarantee base is defined, the income level of the policyholder after annuitisation will be 5% of the guarantee base. There have been some discussions [36] regarding the ability of the policyholder to withdraw the whole amount or not in order to achieve the maximum performance of GMWB option but under the Solvency II capital estimation a sensible assumption is that the policyholder always withdraws the whole amount.

When the fund value, the guarantee base and the income value are defined we can calculate the cost to the insurer for each of the 55 future years. If the return of the investments is lower than expected and the fund value is lower than the initial value then the insurer should cover the difference from his capital. Also, there is a possibility that the policyholder might switch-off the GMWB option. The following equation gives the cost of guarantees V_i at year i.

$$V_i = -P_i \min\left(F_i - I_i, 0\right).$$

where P_i gives the policy in force as this will be active if the policyholder is alive and he hasn't deactivate the GMWB option (lapse rate). Because this valuation is being done under the risk-neutral context we should discount this value with the simulated discount factor of the time of the valuation.

It is obvious that the structure of the above equation is very similar with the payoff of an option and this will be discussed further in Chapter 4 where various types of options will be used for the evaluation of the Least-Squares Monte Carlo algorithm.

We can see that these contracts are very attractive to customers as they offer protection and at the same time investment options that can increase the profit at a relatively low level risk. Furthermore, the expiration of these contracts lasts many years into the future as it depends on the death of the contract owner which might happen many years after acquisition of the contract.

The risk-management of this kind of products is a real challenge for insurance companies because of their complexity. The path-dependence and the number of risk-factors affecting this product can be large as it might include the stock price, the interest rate, the volatility, the lapse rate, the life expectancy of the policyholder, etc.. The calculation of the risk for these products as the number of contracts increases can lead to a compute intensive process with an increasing need for an efficient method to accelerate existing approaches.

2.3 Monte Carlo Simulation

The Monte Carlo method has proven to be a valuable and versatile computational tool in modern financial theory. The complexity of numerical calculations has grown rapidly, requiring greater speed and efficiency in the calculations. Numerical methods are used in various purposes in finance such as the valuation of securities, assessing their sensitivities to certain factors, and assessing their risks, as well as in various portfolios stress tests.

Presented for the first time in finance by Hertz [10] in 1964 in 1977, Boyle [9] introduced the Monte Carlo simulation for valuation of derivative securities.

In finance, the prices of basic securities and the state variables of the underlying assets are often modeled as stochastic processes in continuous time. The decision of buying or selling a derivative security depends on the value of securities. Under no arbitrage condition, it has been proved that a derivative's price can be calculated from the expected value of the discounted profit. The expected value is taken based on the transformation of the original probability measure (risk-neutral measure).

In this dissertation, we use Monte Carlo simulation in the two stages of the LSMC simulation. In the first stage, by using Monte Carlo we create multiple realizations of the risk-factors that affect the valuation of our product and in the second stage we value our product conditional on each of these realizations produced earlier.

2.4 Euler Discretization Scheme

The starting point of the implementation of the Monte Carlo methods for valuation of options is to create sampling paths of the underlying assets. Simple options do not require the creation of sampling paths as their value depends only on the price of the underlying asset at maturity, but for other type of options there are cases where their values depend on the whole path or at least on one sequence of values at given times. There are two different sources of error in the path creation process:

1. The sampling error.

2. The discretization error.

The first error is the result of the Monte Carlo randomness and can be minimized with the use of variance reduction techniques [26]. In this thesis we will use the Euler discretization scheme for the Monte Carlo simulations. In order to explain the discretization error resulting from this scheme we will discretize a continuous time model.

$$dS_t = a\left(S_t, t\right) dt + b\left(S_t, t\right) dW_t$$

If we apply the Euler discretization scheme to this process we obtain the following discretized model:

$$\delta S = S_{t+\delta t} - S_t = a\left(S_t, t\right) \delta t + b\left(S_t, t\right) \sqrt{\delta \mathrm{t}}\varepsilon,$$

where δt is the discretization step and ε ˜ N(0,1).

One of the most important issues in stochastic differential equations is the convergence. If we use the standard normal distribution to sample the random variable ε, we could simulate one discrete time stochastic process associated with the solution of the continuous time equation. The error can be reduced even if we increase the number of paths or the numbers of repetitions.

A major problem of the Euler discretization scheme is that it cannot guarantee the positive values for a CIR process and this will cause problems to the calculation of the square root. For example, if we apply the Euler discretization scheme in the following CIR process:

$$dv_t = \varkappa(\vartheta - v_t)\mathrm{dt} + \xi\sqrt{v_t}dW_t^v$$

we obtain the following equation

$$v\left(t_{i+1}\right) = v\left(t_i\right) + \varkappa\left(\vartheta - v\left(t_i\right)\right)\left[t_{i+1} - t_i\right] + \xi\sqrt{v\left(t_i\right)}Z_i\sqrt{\left[t_{i+1} - t_i\right]}$$

In order to assure that the $v\left(t_i\right)$ will never take negative values, Deelstra and Delbaen (1998) proposed an extension of this scheme to force this value to 0 whenever a negative value appears.

2.5 Antithetic Variates

The method of antithetic variates is one of the simplest and most commonly used methods for variance reduction. The main idea behind this technique, which seems to be introduced by Hammersley and Morton in 1956 [18] in the context of Monte Carlo Simulation, is that suitable pairs are created on the observations obtained from the simulation. The premise is that if we want to estimate a parameter ϑ we need two unbiased estimators which would have a strong negative correlation [27].

Each pair of these observations shall be such that one observation is greater average than the average price ϑ and the other smaller. So considering as an estimate of their average, the price would be closer to the value of the parameter. If Y_1 and Y_2 are unbiased estimators of ϑ, not independent of each other, then the $\frac{(Y_1+Y_2)}{2}$ is also an unbiased estimator of this parameter and its variance will be:

$$Var\left(\frac{(Y_1+Y_2)}{2}\right) = \frac{1}{4}Var\left(Y_1\right) + \frac{1}{4}Var\left(Y_2\right) + \frac{1}{2}Cov\left(Y_1, Y_2\right)$$

Therefore, from the last equation, if the $Cov\left(Y_1, Y_2\right)$ is negative this method can be effective for the reduction of the variance. Suppose we have random variables $X_1, X_2, ..., X_n$ which we use to calculate their sampling average as an estimator of the parameter ϑ. If these parameters are independent then:

$$V\left(\bar{X}\right) = \frac{1}{n^2}V\left(\sum_{i=1}^{n} X_i\right) = \frac{\sigma^2}{n}$$

If these parameters are not independent:

$$\frac{1}{n^2}V\left(\sum_{i=1}^{n} X_i\right) = \frac{1}{n^2}\left(\sum_{i=1}^{n} V\left(X_i\right) + 2\sum_{i<k} Cov\left(X_i, X_k\right)\right) = \frac{\sigma^2}{n} + \frac{2}{n^2}\sum_{i<k} Cov\left(X_i, X_j\right)$$

If $Cov\left(X_1, X_2\right) < 0$ then $V\left(\bar{X}\right) < \frac{\sigma^2}{n}$ and as a result we will have an estimation with a lower variance using the same number of observations.

This method is termed antithetic variates because of the way we produce negative correlated random numbers. Assume that for the calculation of each of the X_i we need m independent random numbers $U_1^i, U_2^i, ..., U_m^i$ from the uniform distribution $U\left(0,1\right)$ and every X_i is a monotonic function of the vector $U_i = \left(U_1^i, U_2^i, ..., U_m^i\right)$ so that $X_i = g\left(U_i\right)$. In order to produce a negatively correlated X_i we produce pairs of vectors U_i and $1-U_i$, where:

$$1-U_i=\left(1-U_1^i,1-U_2^i,...,1-U_m^i\right)$$

The vector $1-U_i$ is consisted of m independent uniform random numbers so the monotonic functions $g\left(U_i\right)$ and $g\left(1-U_i\right)$ will have the same distributions [28]. Now we simulate our X_i as follows:

$$X_{2k-1}=g\left(U_{2k+1}\right)$$

$$X_{2k}=g\left(1-U_{2k+1}\right)$$

for every $k=1,...,n$. Then

$$Cov\left(U_j^i,1-U_j^i\right)=E\left(U_j^i\left(1-U_j^i\right)\right)-E\left(U_j^i\right)E\left(1-U_j^i\right)=$$

$$=E\left(U_j^i\left(1-U_j^i\right)\right)-\frac{1}{2}\frac{1}{2}=$$

$$=E\left(U_j^i-\left(U_j^i\right)^2\right)-\frac{1}{4}=\int_0^1\left(x-x^2\right)f_U\left(x\right)dx-\frac{1}{4}=$$

$$=\int_0^1\left(x-x^2\right)dx-\frac{1}{4}=\frac{1}{6}-\frac{1}{4}=-\frac{1}{12}<0$$

for every $j=1,2,...,m$.

The benefit is that the new estimator has a lower variance compared to the variance of using m new independent U_i values and we save time from generating new random numbers from the uniform distribution as we just use the negative values of the already created random numbers.

2.6 Stochastic-on-Stochastic Simulation and Alternatives

The Stochastic-on-Stochastic simulation allows the production of the distribution of the total liabilities and the direct estimation of the SCR at any selected confidence level. This method is particularly useful when values of liabilities are driven by multiple risk factors and analytical solutions are not available.

Monte Carlo Simulation is a parametric method, with assumed distributions for asset returns for every risk-factor. The calculation of the SRC contains the following basic steps:

1. **Scenario Generation:** Use estimates for the model parameters of the risk-factors (price of the underlying asset, volatility, interest rate etc.) and correlation or dependence structure of the risk-factors (as appropriate), we produce a large number of future values for each of these factors at the projection date.

2. **Liabilities Valuation:** For each scenario in step 1, use another set of model parameters estimate to calculate the value of the liability at the projection date.

Liabilities that are affected by multiple risk-factors are more complex to value. As a result, there is no analytical solution which means that Monte Carlo simulation has to be performed for their valuation, conditional on each of the generated scenarios.

An important observation regarding this algorithm is the different calibration performed for the two simulation stages. In the first stage the purpose is to realistically capture the behavior of all the risk-factors affecting the value of the liability on the projection date. In order to achieve this, the model used for the simulation should be calibrated according to the physical measure proxy by historical data.

In the second stage the simulation should be performed under the risk-neutral measure for market valuation of the liability. One problem with this second stage simulation arises from the fact that many insurance contracts typically last for thirty to forty years into the future and this makes the risk-neutral calibration difficult because usually options with forty years maturity are not available and if there are, they are not liquid. So some heuristic input estimates are necessary.

The execution of this nested simulation is very costly even when the number of risk-factors is low. When the number of risk-factors is high, the valuation of the liabilities is prohibit especially with standard single processor computers. Figure 2.1 presents the simulation schemes under stochastic-on-stochastic (or nested) simulation and Least-Squares Monte Carlo (LSMC), the later of which will be described in the next chapter. As described in Section 2.1, insurance companies are required to calculate their SCR which means that they should minimize their probability of

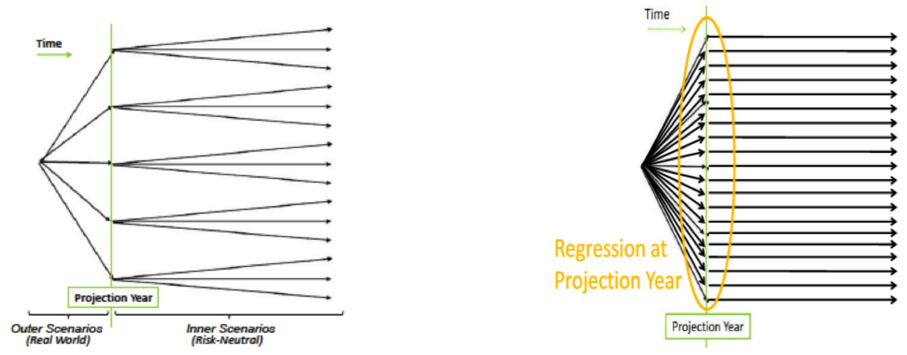

Time
Projection Year
Outer Scenarios (Real World)
Inner Scenarios (Risk-Neutral)
Time
Regression at Projection Year
Projection Year

Figure 2.1: Nested vs LSMC Simulation

default to 0.5% confidence level. For a reasonable accuracy, at least 10000 outer simulation scenarios are needed. This step is what we described above as “Scenario Generation”. In the second step the liabilities should be valued conditional on each of these scenarios. In order to achieve an acceptable accuracy for each valuation we need to perform at least 10.000 simulations in the inner loop. As a result 10.000x10.000 simulation paths are needed to produce the estimation of the total liabilities on the projection date, which is the year from now according to Solvency II. This number is extremely high for the capabilities of a single processor computer especially when considering a portfolio of complex products that an insurance company typically holds.

Multiple solutions have been proposed as alternatives to the stochastic-on-stochastic simulation in order to reduce the computational effort and consequently make the valuation of the liabilities easier. These are the curve fitting approach, the Replicating portfolios and the Least-Squares Monte Carlo.

According to the Curve fitting approach the inner valuation of the liabilities can be performed with the use of few outer scenarios and then we can interpolate the behaviour. The replicating portfolios approach follows the logic of using some assets that areeasy to be valued instead of liabilities. Finally, the Least-Squares Monte Carlo approach reduces the number of the inner valuation scenarios and improve the accuracy of the inaccurate inner valuations with the use of the least-squares method.

In practice, the idea of LSMC is very similar to the stochastic-on-stochastic simulation but instead of using many paths to value the liabilities for each outer scenario we use only one path. Because of the big number of outer scenarios, a regression fit can be built from all these valuation which will be able to minimize the errors of the inaccurate valuations. As will be discussed in the next chapter, the risk-factors act as the explanatory variables of the method and the liability valuations are the responce variables. When the curve is built the method is the same as the curve fitting.

As the number of the risk-factors is getting higher LSMC adds more value to the reduction of the computational effort. The total number of simulations is less compared to the curve fitting for the same level of accuracy and this is what makes the LSMC method efficient compared to other approaches. In addition, the distribution of the outer scenarios that is used for the construction of the regression function can be different than the actual distribution of the real scenarios and this can improve the accuracy of the model at the tails as discussed in the literature.

The disadvantages of the model is that its theory is more complicated compared to the other approaches making it harder to implement and explain. In addition most of the insurance companies have already implemented and used one of the other two approaches and the implementation of the LSMC would be a hard change.

Chapter 3

Least-Squares Monte Carlo Simulation

3.1 Least-Squares Monte Carlo Simulation for American Options

The Least-Squares Monte Carlo (LSMC) algorithm is widely used in finance and it is one of the most important tools for valuing American options. An American option can be exercised any time before and at option maturity. At maturity the option is exercised, if the exercise value is positive. Before maturity, the optimal strategy for the investor is to compare the exercise value at each point in time with the continuation value of the option. If the exercise value is higher then the investor should exercise the option. The key to this valuation strategy is the determination of the continuation value of the option.

The Solvency II Capital Requirement calculation has features similar to the valuation of American options in that the total liability is equivalent to the value of a put option which has to be valued at some future point in time, the projection date at 1-year horizon in this case. Hence, the LSMC for SCR calculation consists of three key steps

- Generation of scenarios for the projection date
- Estimation of the Least-Squares function
- Valuation of liabilities (as a put option)

The LSMC method creates scenarios through Monte Carlo simulation, and then performs iteratively, and at each time step a least-squares approximation of the continuation function. Suppose that ω is a simulated scenario at some projection date. $0 < t_1 \leq t_2 \leq ... \leq t_T = T$ are the exercise dates of this contract and $C(\omega, t)$ is a cash flow scenario calculated assuming that the investor has not exercised the option at or before time t.

At maturity T the investor exercises the option if the exercised value is positive. At time t_k before maturity the investor should decide if he will exercise the option or wait until the next exercise date. The amount of money he will receive for exercising the option is known but the expected value cash flows from holding the option are not known for a particular path in the Monte Carlo simulations.

The continuation value of the option $F(\omega; t_k)$ at time t_k for the scenario ω is calculated as follows:

$$F(\omega; t_k) = E_Q\left[\sum_{j=k+1}^{K} \exp\left(-\int_{t_k}^{t_j} r(\omega,\mathrm{s})\, ds\right) C(\omega; t_j)\,|F_{t_k}\right]$$

where $r(\omega, t)$ is the discount rate to time t_k and Q the risk-neutral measure.

The LSMC algorithm aims at approximating the continuation value, $F(\omega; t_k)$, without iterative simulation at each time step . Given $F(\omega; t_k)$, the investor should be able to follow the optimal strategy either by exercising the option at this time or hold the optional $F(\omega; t_k)$. This is an iterative process until the decisions have been calculated for every scenario at every time step. When the strategy has been fully determined, the valuation of the American options is simple. In summary multiple scenarios are generated through Monte Carlo simulation and for every scenario the optimal stopping point is defined by the LSMC estimate for $F(\omega; t_k)$ and the exercised value of the option.

$F(\omega; t_k)$ is typically approximated as follows:

$$F(\omega; t_j) = \sum_{i=1}^{m} a_i L_i\left(S(t_j, \omega)\right)$$

where $S(t_j, \omega)$ is the price of the underlying asset at t_j for the scenario ω and a_i is the coefficient of the i^{th} basis function L_i.

The coefficients a_i are calculated through the least-squares method as explained in the next section.

3.2 Least-Squares Method

If fitted values y are a linear function of x as shown bellow:

$$y_i = a_0 + a_1 x_i, i = 1 \ldots n$$

the parameters a_0 and a_1, can be estimated from n pairs of $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$ in which the y values deviate from predicted value by a random error of the normal distribution with zero mean. For polynomial of degree m

$$y = a_0 + a_1 x_i + ... + a_m x_i^m$$

$$\sum_{i=1}^{n} [y_i - \hat{y}_i]^2 = \sum_{i=1}^{n} [y_i - (a_0 + a_1 x_i + ... + a_m x_i^m)]^2$$

The unknown coefficients must field zero first derivatives in order to obtain the least-squares error

$$\sum_{i=1}^{n} y_i = a_0 \sum_{i=1}^{n} 1 + a_1 \sum_{i=1}^{n} x_i + a_2 \sum_{i=1}^{n} x_i^2 + ... + a_m \sum_{i=1}^{n} x_i^m$$

$$\sum_{i=1}^{n} x_i y_i = a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 + a_2 \sum_{i=1}^{n} x_i^3 + ... + a_m \sum_{i=1}^{n} x_i^{m+1}$$

$$...$$

$$\sum_{i=1}^{n} x_i^m y_i = a_0 \sum_{i=1}^{n} x_i^m + a_1 \sum_{i=1}^{n} x_i^{m+1} + a_2 \sum_{i=1}^{n} x_i^{m+2} + ... + a_m \sum_{i=1}^{n} x_i^{2m}$$

which can be written as A*b = Y as follows:

$$a = \begin{pmatrix} a_0 \\ a_1 \\ ... \\ a_k \end{pmatrix}, A = \begin{pmatrix} n & \sum_{i=1}^{n} x_i & ... & \sum_{i=1}^{n} x_i^m \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 & ... & \sum_{i=1}^{n} x_i^{m+1} \\ ... & ... & ... & ... \\ \sum_{i=1}^{n} x_i^m & \sum_{i=1}^{n} x_i^{m+1} & ... & \sum_{i=1}^{n} x_i^{2m} \end{pmatrix}, Y = \begin{pmatrix} \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i y_i \\ ... \\ \sum_{i=1}^{n} x_i^m y_i \end{pmatrix}$$

where, n is the number of the outer scenarios, x is the price of the underlying asset, b is the coefficients vector, and y is the value of the liabilities. If the value of the liabilities depends on multiple risk-factors then

$(x_{1,1}, x_{1,2}, ..., x_{1,n}, y_1), (x_{2,1}, x_{2,2}, ..., x_{2,n}, y_2), ..., (x_{n,1}, x_{n,2}, ..., x_{n,n}, y_n)$.

In the case of the SCR calculation, y and x are, respectively, the values of the option (or liability) and the underlying factor (e.g. asset price) on projection date. In particular y is generated by analytical function (typically unavailable) or by inner loop simulations. In the case of multiple factors, x could include stochastic interest rate, asset price, volatility etc.

In our implementation the solution of the above system is done with the use of a function that implements the LU decomposition with partial pivoting and row interchanges to factor A.

The outcome of LU factorization is a lower triagonal matrix L and an upper triagonal matrix U so that:

$$PA = LU.$$

The elements of the diagonal of matrix L are identical and equal to one and the other non-zero positions are the multiplier of Gauss elimination (used to eliminate each element). Matrix U is an upper triagonal matrix resulting from Gauss elimination. Finally, matrix P, is the permutation matrix corresponding to the lines transitions that were performed according to partial pivoting. When L, U and P have been calculated the solution of Ax = B is reduced to:

$$Ly = Pb$$

$$Ux = y$$

3.3 Sampling methods for the outer scenarios

Previous study shows that sampling the outer scenarios from a distribution other than the distribution generated by the model selected for option pricing, can improve the estimation of the LSMC [19]. This is partly because models for option pricing typically leaves very few observations in the tails. There are several choices for the outer scenario sampling and the most obvious one is the full grid sampling where we should fill the grid formed by the risk-drivers. This method can improve the

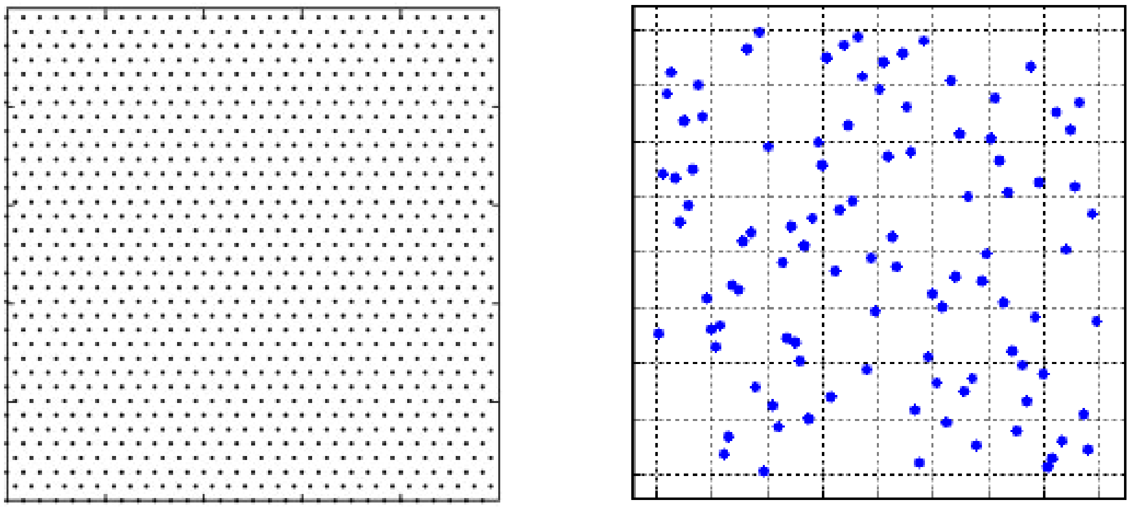


Figure 3.1: Regular grid (Left) and Latin hypercube sampling (Right)

estimation of the LSMC but it suffers from the so-called curse of dimensionality. This means that as the number of the risk-factors increases the dimension of the grid will become prohibitively high.

Other sampling approaches that do not suffer from the curse of dimensionality include the Latin hypercube sampling, the uniform sampling and the quasi-random sampling. The Latin hypercube sampling allows a good degree of reliability with a smaller number of samples and thus less computational time. The main idea of the Latin hypercube sampling is the separation of the space of the multidimensional distribution at various intervals and the creation of samples through random sampling from all these intervals.

The other two approaches are based on pseudo-random numbers (uniform distribution) and semi-random numbers (quasi-random) generators. The quasi-random numbers introduce lower discrepancy compared to the uniform random numbers but this does not necessarily lead to a better estimation result.

3.4 Basis functions

The selection of basis functions for the regression model is a crucial choice for the accuracy of the LSMC simulation described in section 3.1. Choices of the basis functions have been widely discussed in the literature. Basis functions can be powers or from the polynomial families such as orthogonal polynomials, Legendre polynomials,

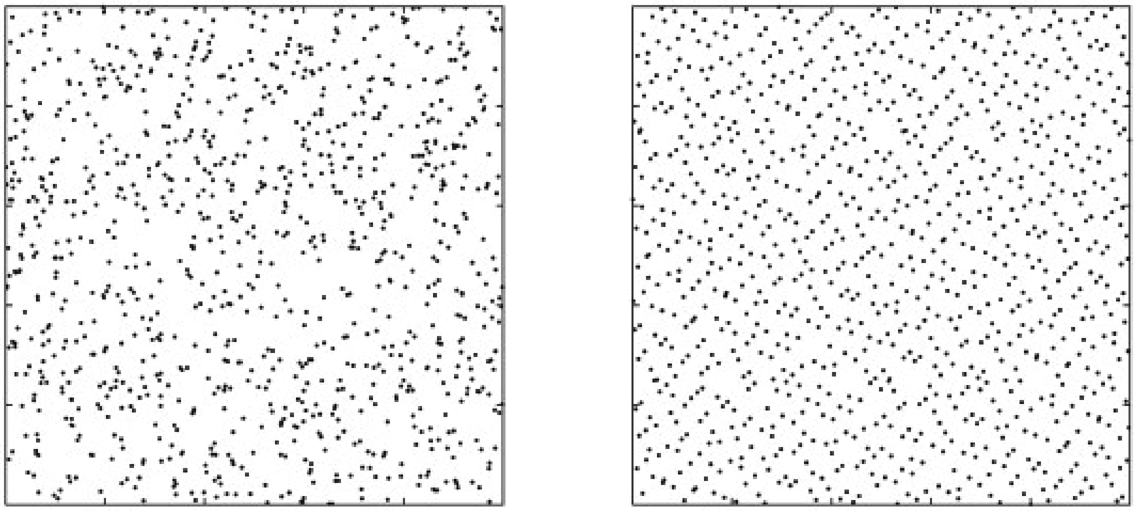


Figure 3.2: Uniform Sampling (Left) and Quasi-random Sampling (Right)

Hermitte polynomials etc.

Moreno and Navas (2003) [13] tested the robustness of the LSMC method against multiple polynomial basis functions and found little difference in their performance. They identified that the method produce similar and accurate results for all of their choices. Moreover, they state that the use of polynomials with degree higher than 20 may cause the least-squares regression to malfunction. Finally, the accuracy and the robustness of the LSMC cannot be guaranteed for complex products because of the weakness of the polynomial basis functions to proxy the value of these products.

Areal et al. (2008) [12] found all the polynomial basis functions they tested provided almost the same accuracy. In addition, the use of powers form of the basis functions is recommended because of the improved computational speed. The choice of the basis functions depends on the type of the option and to keep the same order of accuracy the number of simulation paths must increase with a higher number of basis functions. Finally they find low-discrepancy series combined with Brownian Bridges can improve efficiency as it can be used as a dimension reduction technique.

Glasserman and Yu (2005) [14] examined the convergence of LSMC for American options in terms of the number of basis functions and the number of the simulated paths. They checked their results assuming that the underlying asset follows a Brownian motion and a Geometric Brownian motion and they showed that the number of the simulated paths increases exponentially as the degree of the polynomial basis function increases.

Clement et al. (2002) [15] discussed the Longstaff and Schwartz solution for American options concentrating on the discrete time problem. For their experiments they used in-the-money paths and out-of-money paths in order to calculate the continuation value of the option showing that this method can converge. Their results were verified and extended by Daniel Egloff [33] who proved the convergence of the method if the basis function is fixed.

In this thesis powers were used in the basis functions and logarithmic transformation. Since all the polynomial families produce almost the same accuracy, we choose the fastest form which turns at to be the powers function. In the following chapters we show the accuracy of the LSMC simulation by using these basis functions to price various types of options including exotic options.

3.5 Information Criteria

To choose the best form of the regression function for use in the LSMC one may calculate the Akaike's Information Criterion (AIC) and the Bayesian Information Criterion (BIC). The general form of the AIC is

$$AIC\left(M\right) = 2\log\left(likelihood_{max}\left(M\right)\right) - 2dim\left(M\right)$$

where M is the specific model and dim(M) is the size of the model's parameters.

The AIC criterion evaluates each model according to its goodness of fit by the maximum likelihood and at the same time it penaltizes models complexity with the term -2dim(M). If the penalty term was not used, then the AIC would inevitably lead to the most complex model which is likely to be difficult to calibrate and more prone to errors.

The BIC defined as follows:

$$BIC\left(M\right) = 2\log\left(likelihood_{max}\left(M\right)\right) - \left(\log n\right) dim\left(M\right)$$

It is obvious that the AIC and the BIC information criteria are very similar. They both utilize the maximum log-likelihood of the candidate models and impose an appropriate penalty term, which is $2dim\left(M\right)$ for AIC and $\left(logn\right) dim\left(M\right)$ for BIC. Both penaltize models that have a large number of parameters. For BIC, it also depends on the logarithm of the total number of data points.

The penalty term of BIC is higher and thus stricter regarding the addition of variables for $n > 8$ ($n > e^2$). As a result, BIC discourage the selection of models with many variables compared to AIC. They both go for model with small number of parameters.

On the other hand, AIC performs better in terms of the selection of a model with the least mean squared error [22]. This means that AIC select the same model that is selected by minimizing the squared error.

These two models can both choose the regression function that is closer to the real distribution of data as $n \to \infty$. It is an open research subject how the efficiency of the AIC could be combined with the consistency of BIC.

In this thesis the stepwise AIC method was used [19] to select the best regression functions. A brief description of this process will follow. First, the function with the least elements is tested. Each time a new basis function is added and the calculation of the AIC is carried out. Then a comparison of the AICs is done and the one with the biggest decrease in AIC will be chosen. Then this new model will be tested against all the remaining basis functions and the AIC will be calculated again for all possible choices. If the subsequent AICs are all larger it means that the stepwise algorithm should be terminated and the last model will be the most suitable candidate regression function.

3.6 Error Statistics

Assuming the best choice and number of basis functions are chosen in the previous step, the performance of the LSMC is evaluated based on two error statistics. For this purpose we will utilize two different error metrics. The first error statistic can be used when an analytical formula is available for the valuation of the liabilities, i.e. for the model used in the inner loop simulation. Of course, this is not applicable to many real-world liabilities which are very complex. This error statistic is used in our experiments to study the properties of LSMC.

$$MAE = \frac{1}{n}\sum_{i=0}^{n} |\hat{V}_t^i - V_t^i|$$

Here, the outer scenarios are not sampled from an artificial distribution but they are taken from the real-world distribution instead. This means that the n points used for the valuation of the liabilities are taken directly from the outer Monte Carlo simulation. The MAE (Mean Absolute Error) measures the absolute error for all these points disregarding if the approximated value is higher or lower than the analytical solution.

The second error statistic, PE (Percentile Estimation), describes the behavior of the LSMC approximation compared to the stochastic-on-stochastic valuation at some specific percentiles of the liabilities distribution. More precisely, in the equation below, $\hat{V}_t^i$ is the LSMC valuation of the liability and $V_t^{[j']}$ is the stochastic-on-stochastic valuation, both at percentile j.

$$PE = |\hat{V}_t^i - V_t^{[j']}|$$

In order to determine $V_t^{[j']}$ the following process is performed:

1. n scenarios of the multiple risk-factors are simulated from now to the projection date by using a real-world model.

2. These n points are used as inputs to the LSMC approximation function in order to determine the value of the liability conditional on each scenario generated in step 1.

3. These calculated values are sorted in increasing order. The percentiles of this distribution can be found by multiplying the number of scenarios n with the required percentile. For example for the 50-th percentile, $j' = 0.50 * n$

In order to determine the $\hat{V}_t^i$ the scenario that generated the $V_t^{[j']}$ should be found and then a full simulation will be performed for this scenario in order to value the liability using stochastic-on-stochastic simulation. This will be only an approximation of the 50-th percentile of the distribution which can be improved if we increase the number of the full simulated scenarios around this percentile. Of course, this will consequently increase the required computational effort.

The second error metric can be used in cases where an analytical formula is not available for the inner valuation model and it can be applied to real-world valuation processes.

Chapter 4

Application of Least-Squares Monte Carlo

4.1 European Put Option

As mentioned above, recently, practitioners tend to use LSMC in order to calculate economic capital requirements in accordance to Solvency II regulation. Bauer et al. (2010) published a comparison between nested simulations and the LSMC approach in order to calculate the solvency capital requirement. Their results reveal the higher computational effort required by the nested simulation as well as the acceptable accuracy of the LSMC although this method is heavily dependent on the choice of the basis functions.

Koursaris (2011) [16] also discussed the application of the LSMC in the context of SCR. He introduced a two risk-factors model for variable annuities. In the first case study used the risk-factors consisted of the price of the underlying asset and the volatility whereas in the second case study he discussed the price and the interest rate. The results of these implementations shows that using polynomials as basis functions makes the method more robust as polynomials can be configured to fit multiple function shapes and ultimately multiple different payoffs. He also acknowledges the reduction in computational effort compared to nested simulation and the high fitting speed.

Cathcart et al. (2012) [17] used the LSMC method to calculate variable annuity economic capital. The main contribution of his work is that he treats the distribution of the outer scenarios as a choice parameter in order to improve the behavior of the regression function at the tails of the distribution. This idea improves the accuracy only at the tails and not in the middle of the distribution. In addition, he makes use of the powers of the risk-factors as basis functions and he discusses the form of the regression function by comparing them using the Akaike Information Criterion.

The above research efforts shows that there are some critical decisions that affect the performance of the LSMC method. These are the use of variance reduction techniques (antithetic variates), the sampling method that will be used for the fitting of the outer simulation scenarios (Uniform, Quasi-Random) and the choice of the basis functions that will be used for the regression of the risk-factors. In addition, a common finding of the above research is that even after the compute reduction provided by the LSMC compared to the stochastic-on-stochastic valuation, this method is still compute-intensive as we increase the number of risk-factors of our economy and the simulated paths. In the next part of this thesis we introduce the utilization of the state-of-the-art HPC techniques for the acceleration of this method.

In this chapter we present the application of the LSMC simulation for the valuation of a European Put Option, with 10 years maturity, at year 1. The reason for this choice is that the payoff of a European Put Option is similar to the liabilities that are faced by the insurance companies. As is known, the payoff of a European Put Option is $max\{K - S_T, 0\}$ where K is the strike price and S_T is the price of the underlying asset at maturity. The liabilities of an insurance company at year t can be expressed as the $max\{P - F_t, 0\}$ where P is the amount of money that the insurance company should pay to the contract owner and F_t is the amount of money in an underlying fund.

As mentioned above the multiple investment options that are available to the policyholder of the variable annuities contract, can increase or decrease the amount of money available to the fund. As a result, if at time T, $P > F_T$ the insurance company must cover the difference with its own reserved capital, but if $P < F_T$ then the insurance company will not face any liability. It is obvious that the structure of these liabilities follows exactly the same logic as the payoff of the European Put Option.

The value of this option will be calculated against two different models, the Heston Model and the Black-Scholes CIR model. In the first case an analytical formula is available to evaluate the LSMC approximation but for the second case, in the absence of an analytical solution, the stochastic-on-stochastic valuation will be used.

4.1.1 Heston Model

We introduce a two risk-factors model to our regression function with the use of the Heston model. Heston (1993) developed a closed form solution for pricing European Options. He modeled the stochastic process of variance as a mean reverting process using its square root. He assumed that the stock price follows the stochastic differential equation:

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^S$$

where W_t^S is a Wiener process. Also, the volatility follows the process:

$$dv_t = \varkappa(\vartheta - v_t)\mathrm{dt} + \xi\sqrt{v_t} dW_t^v$$

where $\langle dW_t^S, dW_t^v \rangle = \varrho dt$

Our risk-factors for this model will be the stock price and the volatility of the underlying asset at the projection date. The parameters used for these experiments are as follows: the valuation date is t=1, the maturity T of the European Put Option is 10, the price of the underlying asset is 1 and the volatility 20%. The strike price K is 1.3 and the interest rate for the outer scenarios and the risk-free rate is 0.05. We will perform 50,000 real-world simulation scenarios to capture the behavior of the risk-factors at year one and then we will sample 1,000 outer scenarios uniformly and for each scenario we perform a second Monte Carlo simulation to value our liabilities. In addition, we try to show the success of this algorithm even if different models for the simulation of the risk-factors and the valuation of the liabilities are used. For this application we use the Heston model for the simulation of the risk-factors as we mentioned above and for the valuation of the liabilities we use the Black-Scholes model. For the valuation model there is available an analytical solution in order to measure the accuracy of the LSMC results.

In chapter three we described the stepwise AIC method which is used for the definition of the form of the regression function and the impact of this to the estimate accuracy. Below, the LSMC simulation will be applied using multiple regression function in order to prove the previous argument.

For the first regression function the basis functions were used only two basis functions the S_1 and σ_1. At the second regression function the S_1^2, σ_1^2 and a mixed term $S_1\sigma_1$ were added. A the last regression function used the basis functions were up to third order for S_1 and up to second order for σ_1 with the addition of some mixed terms $S_1\sigma 1$, $S_1^2\sigma 1$ and $S_1^3\sigma_1$. Table 4.1 shows the error metrics for the various regression functions compared to the analytical solution of the valuation model.

It is obvious that as we include more basis functions and mixed terms at the regression function the error metrics are decreasing to lower levels. For the f_3 proxy function we can observe that the error compared to the whole valuation distribution is only 0.41% and the error at the 99.5-th percentile is 0.25%.

$$f_1\left(S_1, \sigma_1\right) = c_0 + c_1 S_1 + c_2 \sigma_1$$

$$f_2\left(S_1, \sigma_1\right) = c_0 + c_1 S_1 + c_2 \sigma_1 + c_3 S_1^2 + \mathrm{c}_4 \sigma_1^2 + c_5 S_1 \sigma_1$$

$$f_3\left(S_1, \sigma_1\right) = c_0 + c_1 S_1 + c_2 \sigma_1 + c_3 S_1^2 + \mathrm{c}_4 \sigma_1^2 + c_5 S_1 \sigma_1 + c_6 \mathrm{S}_1^3 + c_7 S_1^2 \sigma_1 + c_8 S_1 \sigma_1^2 + \mathrm{c}_9 S_1^3 \sigma_1$$

$$f_4\left(S_1, \sigma_1\right) = c_0 + c_1 S_1 + c_2 \sigma_1 + c_3 S_1^2 + \mathrm{c}_4 \sigma_1^2 + c_5 S_1 \sigma_1 + c_6 \sigma_1^3 + c_7 S_1^2 \sigma_1 + c_8 S_1 \sigma_1^3 + \mathrm{c}_9 S_1^2 \sigma_1^2 + c_{10} S^5 + c_{11} S^5 \sigma_1$$

Proxy Function	MAE	PE	Log-MAE	Log-PE
f_1	3.4%	1.2%	0.94%	0.11%
f_2	1.4%	1.7%	0.78%	0.42%
f_3	0.40%	0.35%	0.25%	0.19%
f_4	0.56%	0.43%	0.13%	0.20%

Table 4.1: Regression and Logarithmic regression error metric MAE and PE

In chapter 3 we also stated that, except for the form of the regression function, the form of the basis functions impacts the algorithm's accuracy. In the following experiments we use the same form for the three regression functions presented above but in this case we will use the logarithmic transformation of S_1 instead. The accuracy results of the regression functions are presented in Table 4.1. For every case there is an improvement of the estimates compared to the previous experiment where we did not use the logarithmic transformation of the stock price.

Figure 4.2 below shows the analytical surface of the valuation model and Figure 4.3 shows the LSMC regression formula surface. We can see that the LSMC surface

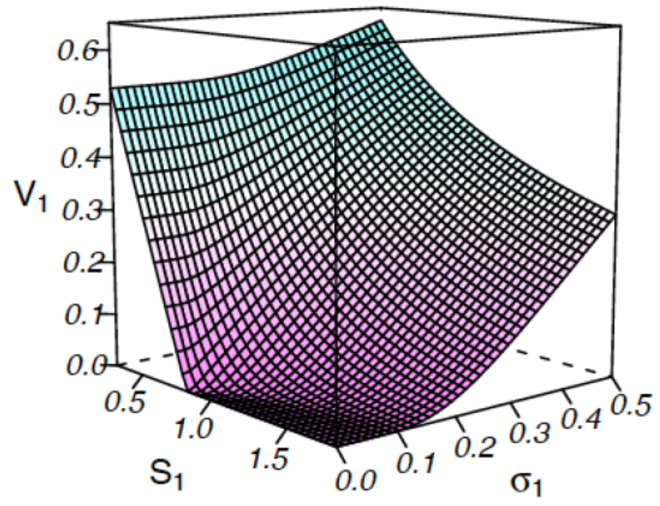


Figure 4.1: Analytical Surface - European Put Option

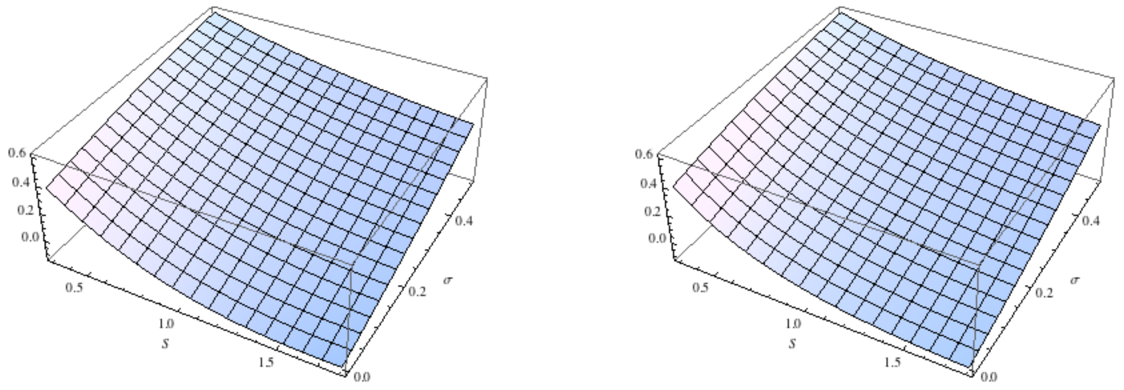
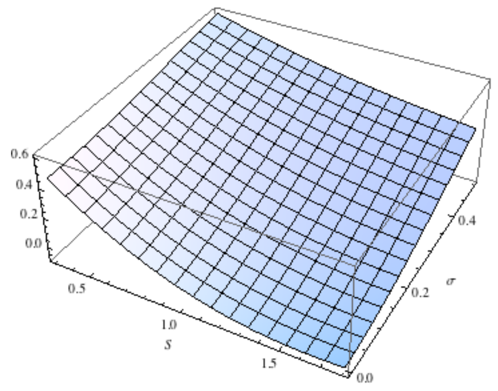


Figure 4.2: Regression surface for f_1 (left) and f_1 with logarithmic stock price (right)

follows the behavior of the analytical surface as when the stock price is getting higher the option value decreases and vice versa and when the volatility of the underlying asset is increasing the option value increases. When $S_1 > 0.8$ the value of the option is very close to zero and the same happens at the analytical surface. When the volatility $\sigma < 0.2$ the value of the option is also close to zero. With the naked eye, it is not easy to capture the real accuracy of the LSMC simulation but the error metrics presented above shows the success of this algorithm.

4.1.2 Black-Scholes CIR

The second model we use to check the accuracy of the LSMC simulation is the Black-Scholes CIR. For the Black-Scholes CIR model, we assume that the equity



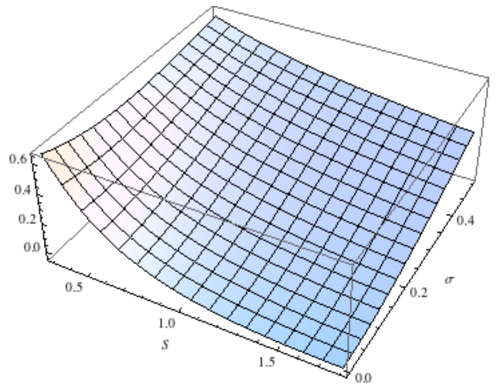


Figure 4.3: Regression surface for f_4 (left) and f_4 with logarithmic stock price (right)

index returns are lognormal as in the Black-Scholes model but the risk-free rate is stochastic and follows a CIR process.

$$dS_t = r_t S_t dt + \sigma S_t dW_t^S$$

$$dr_t = \varkappa(\vartheta - r_t)\mathrm{dt} + \sigma\sqrt{r_t}dW_t^r$$

where $\langle dW_t^S dW_t^r \rangle = \varrho dt$

In this case we use the same model for the real-world simulation and for the valuation of the liabilities. The uniform distribution will be used to sample the outer fitting scenarios. Two different regression functions will be used to show the impact of the regression models to the accuracy and we will also present the impact of the use of powers of the explanatory variables as basis functions and their logarithmic transformation.

The regression functions we will be used for the BS-CIR model are:

$$f_1(S_1, r_1) = c_0 + c_1 S_1 + c_2 r_1 + c_3 S_1^2 + \mathrm{c}_4 r_1^2 + c_5 S_1 r_1$$

$$f_2(S_1, r_1) = c_0 + c_1 S_1 + c_2 r_1 + c_3 S_1^2 + \mathrm{c}_4 r_1^2 + c_5 S_1 r_1 + c_6 S_1^3 + c_7 S_1^2 r_1 + c_8 S_1 r_1^2 + c_9 S_1^4 + c_{10} S_1^3 r_1 +$$

$$+ c_{11} S_1^5 + c_{12} S_1^4 r_1 + c_{13} S_1^6 + c_{14} S_1^5 r_1$$

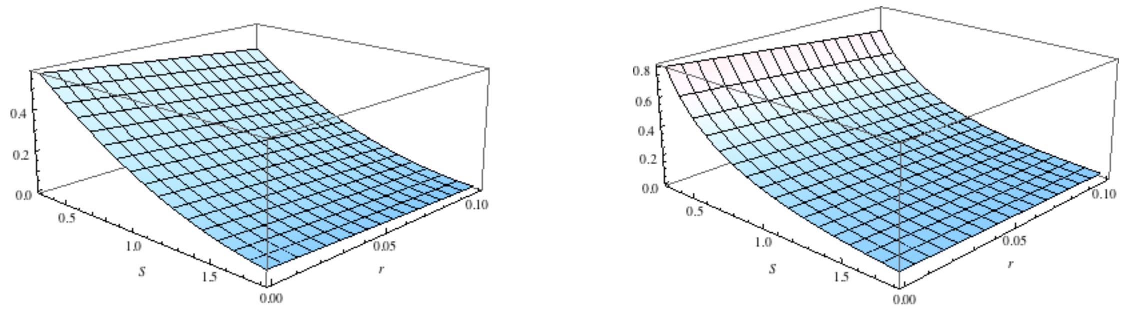


Figure 4.4: Regression surface for f_1 (left) and f_1 with logarithmic stock price (right) - European Put

The first function in simpler than the second as it includes four basis functions instead of thirteen in the second case. In the previous case presented above, there was an analytical formula available for the model used for the valuation of the liabilities. For Black-Scholes CIR such a formula does not exist for the evaluation of the LSMC results. Stochastic-on-stochastic valuation will be used to produce a more accurate result for some percentile of the regression-estimated liabilities distribution. The regression-estimated liabilities distribution is the distribution generated from the regression function calculated before when we use as inputs the scenarios from the real-world simulation model. We perform a stochastic-on-stochastic valuation only for some percentile of this distribution as it would be extremely time-consuming to do it for every scenario as discussed in previous chapters.

The process to calculate multiple percentiles of this distribution is described in Section 3.7. In Table 4.2 the results of the LSMC estimates and the stochastic-on-stochastic valuations are presented for various percentiles from the middle to the upper part of the distribution. Table 4.3 shows the results for the same regression function but now the basis functions are the logarithmic transformation of the stock price instead of the stock price and the interest rate as before. We can see the there is a big improvement in the estimates as now they are closer to the stochastic-on-stochastic valuations. The estimations are improved for all the percentiles checked showing again that the logarithmic basis function can better capture the behavior of the payoff of a European put option even when the valuation model is more complex. Tables 4.4 and 4.5 show the behavior of the LSMC considering a more complex regression function. It is obvious that the results are improved but in this case the incorporation of the logarithmic transformation did not improve the accuracy of the estimates.

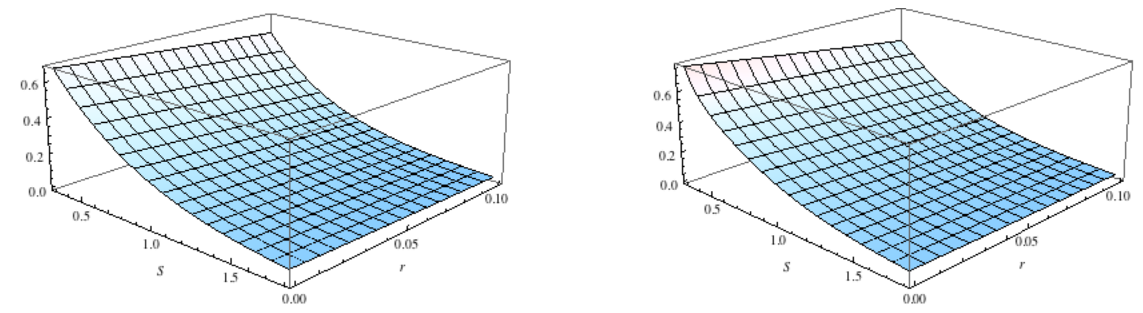


Figure 4.5: Regression surface for f_2 (left) and f_2 with logarithmic stock price (right) - European Put

Perc.	S	r	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0095	4.32%	0.1908	0.1713	1.9%
75-th	0.8846	4.45%	0.2353	0.2149	2.0%
90-th	0.8413	2.16%	0.2736	0.2558	1.7%
95-th	0.7686	3.02%	0.2961	0.2825	1.3%
99-th	0.5734	8.04%	0.3356	0.3478	1.2%
99.5-th	0.6506	2.97%	0.3514	0.3586	0.72%
Average					1.47%

Table 4.2: Percentiles Comparison for f_1 - European Put

Perc.	S	r	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0283	3.56%	0.1698	0.1718	0.18%
75-th	0.8755	4.85%	0.2107	0.2144	0.42%
90-th	0.7922	4.28%	0.2508	0.2569	0.70%
95-th	0.7168	5.50%	0.2770	0.2848	0.76%
99-th	0.6453	4.71%	0.3274	0.3350	0.76%
99.5-th	0.5646	8.13%	0.3483	0.3521	0.48%
Average					0.55%

Table 4.3: Percentiles Comparison for f_1 with logarithmic stock price - European Put

Perc.	S	r	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0766	1.73%	0.1714	0.1720	0.06%
75-th	0.8500	5.98%	0.2156	0.2151	0.04%
90-th	0.7811	4.83%	0.2578	0.2578	0.01%
95-th	0.7441	4.05%	0.2846	0.2848	0.02%
99-th	0.6795	2.79%	0.3345	0.3344	0.009%
99.5-th	0.5861	6.38%	0.3543	0.3565	0.22%
Average					0.06%

Table 4.4: Percentiles Comparison for f_2 - European Put

Perc.	S	r	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0068	4.43	0.1716	0.1713	0.03%
75-th	0.9383	2.16	0.2160	0.2150	0.10%
90-th	0.8014	3.87	0.2582	0.2575	0.06%
95-th	0.7441	4.05	0.2849	0.2848	0.005%
99-th	0.6232	5.95	0.3344	0.3361	0.17%
99.5-th	0.5861	6.38	0.3543	0.3565	0.22%
Average					0.09%

Table 4.5: Percentiles Comparison for f_2 with locarithmic stock price - European Put

4.2 Asian Put Option

The Asian put option is a kind of exotic options where the average price of the underlying asset to maturity or at certain times, shapes the final profit. The average of the stock prices presents a smaller volatility than the final price, these products are cheaper compared to the regular products where their value depends on the price of the underlying asset at maturity.

There are multiple versions of Asian options available based on how the calculation of the average price of the underlying asset is done. Asian options can be based on the arithmetic or the geometric mean of the underlying asset price. In addition, if the mean is calculated for the whole time duration of the simulation then the Asian option is called plain vanilla but if the mean is calculated from some point to maturity then Asian options are called forward-starting options. Another distinction of this kind of products is based on the strike price. If the strike price is constant they are called fix-strike and if the strike is not constant they are called floating-strike

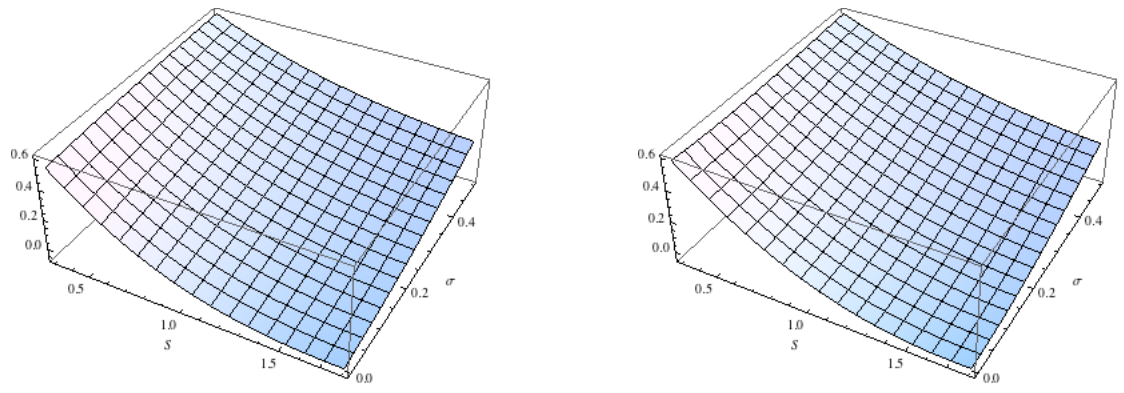


Figure 4.6: Regression surface for f_3 (left) and f_4 (right) - Asian Put

[23].

For the following experiments the geometric mean of the underlying price is calculated taking into consideration the price of the asset at the end of each year until maturity. The strike price is assumed constant.

4.2.1 Heston Model

In order to apply the LSMC simulation for an Asian option we use the Heston model as we did for the European put option. The same parameters will be applied to the models and the same number of outer and inner simulations will take place in order to compare the behavior of the algorithm for different payoffs.

The regression function we use for this experiment will be f_3 and f_4 as they were described in Section 4.1.1 with a logarithmic transformation of the underlying stock price.

Figure 4.6 shows the regression surfaces for f_3 and f_4. We can see that the LSMC simulation can capture the behavior of this product and as the impact of the volatility at this simulation is lower compared to the European put option surface presented above the value of this option is also lower.

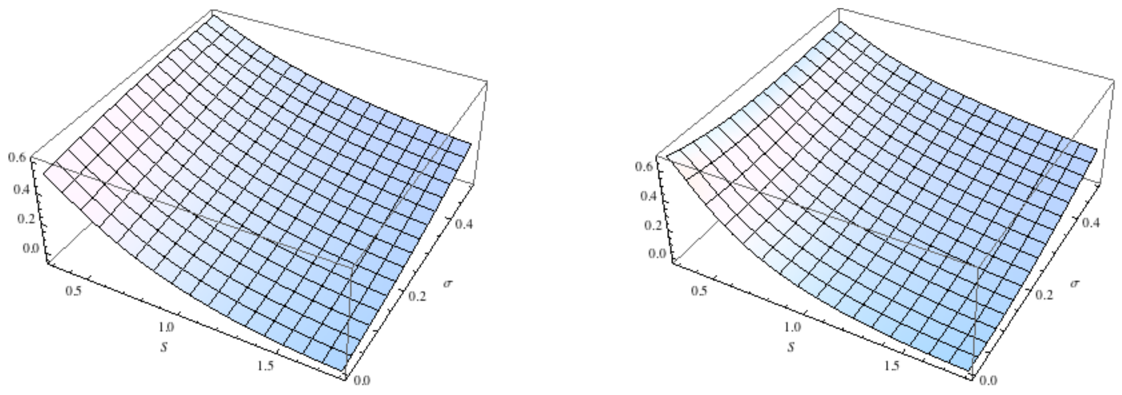


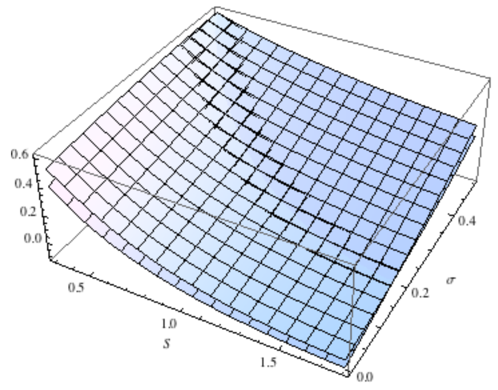
Figure 4.7: Regression surface for f_3 (left) and f_4 (right) with logarithmic stock price - Asian Put

Perc.	S	σ	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0365	14.14%	0.2576	0.2571	0.05%
75-th	0.7474	9.53%	0.3263	0.3235	0.28%
90-th	0.7481	18.52%	0.3888	0.3850	0.38%
95-th	0.6521	18.14%	0.4255	0.4214	0.41%
99-th	0.5002	17.13%	0.4917	0.4886	0.31%
99.5-th	0.4716	20.17%	0.5167	0.5144	0.22%
Average					0.27%

Table 4.6: Percentiles Comparison for f_3 - Asian Put

Perc.	S	σ	Reg.Est.	Full Est.	Abs.Diff.
50-th	0.9448	10.89%	0.2558	0.2561	0.02%
75-th	0.8707	15.91%	0.3213	0.3232	0.19%
90-th	0.7444	18.28%	0.3828	0.3849	0.21%
95-th	0.6052	14.14%	0.4199	0.4219	0.19%
99-th	0.5002	17.13%	0.4893	0.4886	0.07%
99.5-th	0.4880	23.19%	0.5168	0.5168	0.0012%
Average					0.11%

Table 4.7: Percentiles Comparison for f_3 with logarithmic stock price - Asian Put



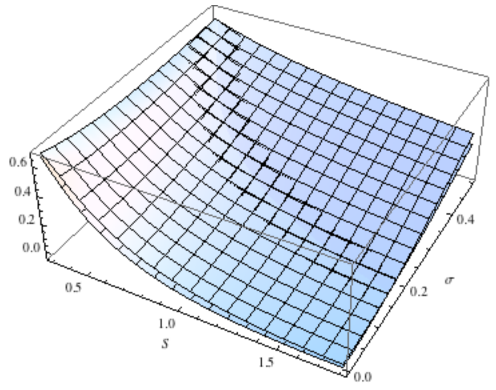


Figure 4.8: Comparison Regression surface for f_3 and f_4 Asian and European Option

Perc.	S	σ	Reg.Est.	Full Est.	Abs.Diff.
50-th	0.9548	11.15%	0.2620	0.2551	0.69%
75-th	0.8098	12.90%	0.3316	0.3232	0.84%
90-th	0.7735	19.96%	0.3933	0.3841	0.91%
95-th	0.6133	14.93%	0.4290	0.4221	0.68%
99-th	0.4858	15.32%	0.4915	0.4892	0.23%
99.5-th	0.5131	25.10%	0.5159	0.5119	0.39%
Average					0.62%

Table 4.8: Percentiles Comparison for f_4 - Asian Put

Perc.	S	σ	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0170	13.36%	0.2558	0.2558	0.003%
75-th	0.8505	14.85%	0.3185	0.3225	0.39%
90-th	0.6724	13.45%	0.3790	0.3859	0.69%
95-th	0.6245	15.85%	0.4169	0.4217	0.48%
99-th	0.5302	21.32%	0.4896	0.4902	0.06%
99.5-th	0.4872	23.16%	0.5180	0.5171	0.09%
Average					0.28%

Table 4.9: Percentiles Comparison for f_4 with logarithmic stock price - Asian Put

4.3 Barrier Put Option

These options are similar to the simple European options with the difference that the price of the underlying asset must exceed or not (depending on the type of the option) a specified price (called the barrier) within a certain period of time, to be considered active; otherwise it is considered dead. These are quite attractive to investors because of their low price, due to the possibility they will expire without being exercised if the price of the underlying asset does not reach the predetermined barrier. Various types of barrier options are traded in the OTC market. The barrier options are categorized as follows:

- The initial price of the underlying asset is higher than the value of the barrier:
  - **down-and-in:** The option is considered active when the price of the underlying asset becomes lower or equal to the value of the barrier until the expiration date date of the contract.
  - **down-and-out:** The option is considered dead when the price of the underlying asset becomes lower or equal to the value of the barrier until the expiration date of the contract.
- The initial price of the underlying asset is lower than the value of the barrier:
  - **up-and-in:** The option is considered active when the price of the underlying asset becomes higher or equal to the barrier value until the expiration date of the contract.
  - **up-and-out:** The option is considered dead when the price of the underlying asset becomes higher or equal to the barrier value until the expiration date of the contract.

In addition, there is a another type of barrier option whereby, when the price of the underlying asset reaches the barrier, the option becomes dead and a part of the premium is returned to the holder of the contract. This type is usually more expensive.

Barrier options can be considered as European (if they can be exercised only at maturity) or American (if they can be exercised any time until maturity) type of options. Investors use these kind of contracts for hedging and to increase their financial leverage (higher profit assuming higher risk).

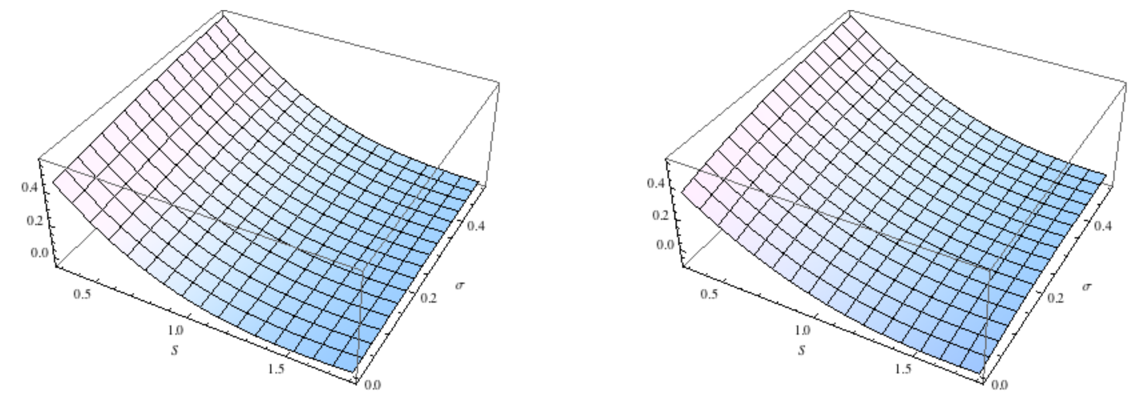


Figure 4.9: Regression surface for f_3 (left) and f_4 (right) - Barrier Put

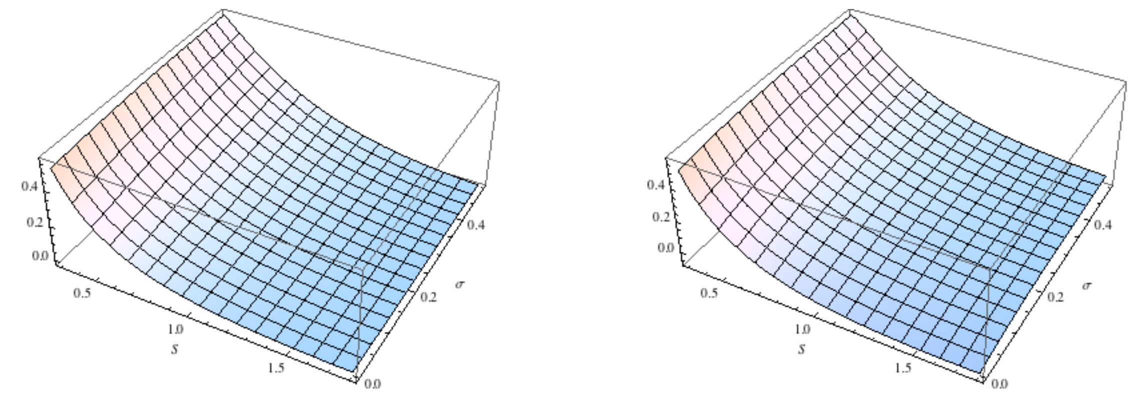


Figure 4.10: Regression surface for f_3 (left) and f_4 (right) with logarithmic stock price - Barrier Put

4.3.1 Heston Model

For this measurement of the accuracy under this product we used the same set-up for the LSMC simulation as we did for the European put option. In the following graphs and tables we present the results for functions f_3 and f_4. For these simulations the barrier option is an up-and-out option where the initial stock price is 1 the strike price is 1.3 and the barrier is 1.1.

Perc.	S	σ	Reg.Est.	Full Est.	Abs.Diff.
50-th	0.9893	10.57%	0.0894	0.0786	1.08%
75-th	0.8206	12.13%	0.1645	0.1714	0.68%
90-th	0.6852	12.84%	0.2412	0.2555	1.42%
95-th	0.6397	22.24%	0.2885	0.3134	2.48%
99-th	0.4883	13.24%	0.3821	0.3912	0.90%
99.5-th	0.4517	15.71%	0.4188	0.4256	0.68%
Average					1.20%

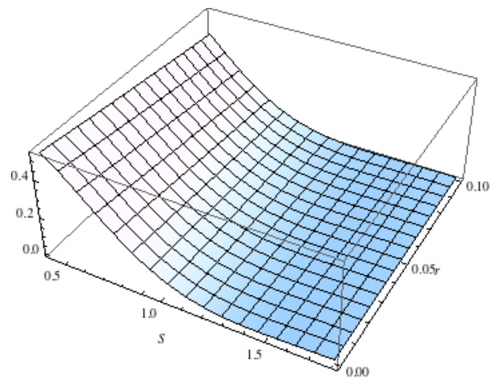
Table 4.10: Percentiles Comparison for f_3 -Barrier Put

Perc.	S	σ	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0070	13.56%	0.0982	0.0792	1.89%
75-th	0.8210	12.42%	0.1634	0.1724	0.90%
90-th	0.6745	11.42%	0.2334	0.2562	2.28%
95-th	0.6055	12.29%	0.2793	0.3060	2.67%
99-th	0.4901	14.21%	0.3730	0.3928	1.98%
99.5-th	0.4558	17.29%	0.4119	0.4268	1.48%
Average					1.86%

Table 4.11: Percentiles Comparison for f_3 with logarithmic stock price - Barrier Put

Perc.	S	σ	Reg.Est.	Full Est.	Abs.Diff.
50-th	0.9990	12.07%	0.0993	0.0787	2.05%
75-th	0.8263	13.27%	0.1770	0.1725	0.44%
90-th	0.6865	13.12%	0.2526	0.2557	0.31%
95-th	0.6119	12.79%	0.2978	0.3038	0.59%
99-th	0.5002	17.13%	0.3845	0.3941	0.95%
99.5-th	0.4600	18.71%	0.4178	0.4273	0.94%
Average					0.88%

Table 4.12: Percentiles Comparison for f_4 - Barrier Put



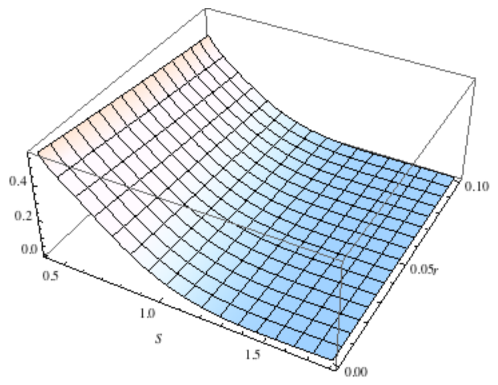


Figure 4.11: Regression surface for f_2 (left) and f_2 with logarithmic stock price (right) - Barrier Put

Perc.	S	σ	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0016	12.51%	0.0969	0.0787	1.81%
75-th	0.8148	11.04%	0.1577	0.1702	1.24%
90-th	0.6833	12.46%	0.2254	0.2551	2.96%
95-th	0.6230	15.97%	0.2714	0.3072	3.57%
99-th	0.4920	14.59%	0.3720	0.3926	2.06%
99.5-th	0.4517	15.71%	0.4137	0.4256	1.19%
Average					2.13%

Table 4.13: Percentiles Comparison for f_4 with logarithmic stock price - Barrier Put

4.3.2 Black-Scholes CIR

For this model we also used the European put option set-up for the LSMC simulation and the same parameters for the valuation model. In the following experiments we compare the behavior of the algorithm under the f_2 proxy function and the f_2 proxy function with logarithmic transformation of the stock price.

Perc.	S	r	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0212	3.52%	0.0517	0.0487	0.29%
75-th	0.9035	3.22%	0.1076	0.1111	0.34%
90-th	0.8069	3.27%	0.1679	0.1711	0.31%
95-th	0.7657	2.49%	0.2067	0.2090	0.22%
99-th	0.6524	4.31%	0.2777	0.2742	0.34%
99.5-th	0.6048	5.39%	0.3047	0.3006	0.40%
Average					0.31%

Table 4.14: Percentiles Comparison for f_2 - Barrier Put

Perc.	S	σ	Reg.Est.	Full Est.	Abs.Diff.
50-th	1.0144	4.07%	0.0517	0.0491	0.26%
75-th	0.8301	8.29%	0.1076	0.1079	0.03%
90-th	0.7863	4.68%	0.1687	0.1693	0.05%
95-th	0.7461	3.81%	0.2081	0.2076	0.05%
99-th	0.6580	3.96%	0.2786	0.2740	0.45%
99.5-th	0.6330	3.60%	0.3046	0.2989	0.56%
Average					0.23%

Table 4.15: Percentiles Comparison for f_2 with logarithmic stock price - Barrier Put

4.4 Summary of Options pricing

In this chapter we discussed the valuation of three different options, a European put option, an Asian put option and a Barrier put option. In every case the LSMC simulation could capture the behavior of the products and present an acceptable accuracy for the various basis functions. There are though some interesting outcomes that show that this algorithm needs to be adjusted each time to the specific product valuation.

For the valuation of the European put option under the Heston model the best regression model is function f_4 with the use of the logarithmic transformation of the stock price. This combination provided the best accuracy compared to whole distribution and almost the same accuracy as function f_3 with the use of the logarithmic transformation of the stock price. In addition, the introduction of the logarithm as a basis function improved the accuracy of the LSMC simulation for every proxy function.

For the valuation of the European put option under the Black-Scholes CIR model function f_2 presented the best accuracy but in this case the introduction of the logarithmic stock price as a basis function did not improve the accuracy. On the other hand, the accuracy was improved for functionf_1 but these estimates were still poor compared to the f_2 proxy function.

When the payoff of an Asian option was used function f_3, using the logarithmic transformation of the stock price was the best choice as it presented better results for the various percentiles of the regression-estimated valuation distribution as well as for the 99-5th percentile specifically.

In the case of the barrier option LSMC presented the poorest estimates compared to the other products. Barrier options are strongly path dependent products and LSMC was unable to capture the behavior as accurately as before. The best estimation results were produced with the use of the f_3 proxy function. The introduction of logarithmic transformation as a basis function worsens the accuracy. The valuation of this product under the Black-Scholes CIR model presented better estimations as for the 99-5th percentile the absolute difference was 0.40%. The use of logarithmic transformation improved the estimates at the middle of the distribution but it did not improve the accuracy at the upper percentiles.

In conclusion, LSMC can capture the behavior of multiple products but the performance of the algorithm may be based on the underlying product. Different products might require different sampling methods, proxy functions, basis functions and different outer and inner simulation paths in order to improve the accuracy of the estimates. It can be a very useful method for practitioners not only because of the accurate estimations but primarily because of the reduction of the computational effort provided compared to the Stochastic-on-Stochastic simulation.

	European	Asian	Barrier
Heston	Log - f_4	Log - f_3	f_3
Black Scholes CIR	f_2	f_2	f_2

Table 4.16: Functions that produce the best results for each model and each Option

Chapter 5

Multicore CPUs

In 1965, Gordon Moore, co-founder of Intel, published an article entitled "Cramming more components into integrated circuits"[6]. In this article, Moore predicted that the number of transistors per chip and the processing power will double every year. This period increased later to 18 months, giving the final shape to what was later referred to as the Moore's Law.

5.1 The collapse of Moore's Law

In the previous decades, Moore's Law was totally respected from the integrated circuits industry, being a motivation which introduced the technology in areas such as economy, entertainment and communication.

Moore made a second prediction, which did not receive much special attention until recently. He predicted that the power consumption of each CPU will decreases as the number of transistors increases. The reduction of the power per processing unit is necessary in order to be able to increase the density of the transistors and at the same time keeping the consumption at relatively stable and acceptable levels, in order to achieve increase in processing power.

The power consumption is proportional to the clock frequency, thus imposing a natural limit to this. The power consumption of an integrated circuit (IC) does not reveal the whole truth as an important parameter is also the power density (Watts / cm2). The miniaturization of ICs resulted in a reduced transistors consumption but also increased their number per unit. Moreover, the static power, which until recently was negligible, is now a major consumption factor, which increases as technology miniaturizes. Accordingly, the density power consumption has risen to the physical limits that can be supported by conventional cooling systems. All the above constructed a power wall which prevented further increase of density and hence processor performance.

Even using exotic materials and sophisticated cooling systems, the increase of the frequency will not lead to the desired benefits. The required increase of the pipeline's level is not easily exploited, while there is a significant burden on the hardware complexity. The frequency wall and the power wall make the increase of the processor's clock unaffordable for the amount of the increase it provides.

Another major limiting factor in the evolution of the processing units is memory. The speed difference between DRAMs and CPUs is continually growing. The relatively increasing access time to the main memory undermines the increase at the speed of the processors, creating this way a memory wall. Multiple cache hierarchies are used in order to conceal the delay of the main memory. These types of memory are usually expensive, they require a lot of space and they are an important factor of static power consumption.

The memory wall, beyond the obvious obstacles created, limits the potential parallelism of sequential programs. The execution speed of these programs is greatly increased when the hardware can exploit parallelism at the instruction level (Instruction Level Parallelism - ILP). The potential parallelism resulting from the code segments that do not have dependencies. The slow memory, along with the inability to improve the branch prediction methods, restricts the theoretical limit of the ILP. The ILP wall is also one of the reasons hardware developers had to seek for an alternative model than the execution of the programs in single processors.

Asanovic et all., mention that: "The power wall + the memory wall + the ILP wall = a brick wall for serial performance" [7]. In April 2010, the vice president of Nvidia, Bill Dally, said that "Moore's law is dead" [8], referring to the inability of the serial models to double their performance every 18 months.

Given that the density of the transistors is able to follow Moore's Law for at least one to two decades yet, the incorporation of many cores in one IC and the dissemination of the parallel programming model provided the most logical solution. The ILP is replaced by Thread Level Parallelism (TLP), so the frequency and the consumption are kept within acceptable limits while the processing power increases.

5.2 Evolution of Multicore CPUs

During the last decade it became clear that the single processor performance was not increasing despite Moore's Law. Hardware companies tried to deal with this through the implementation of products based on multicore architectures. These products contained multiple CPU cores which shared a cache level memory, communicated through simple interface and were located on the same IC (Chip Multiprocessors or CMPs). The architecture of the cores was similar to that of the single processor implementations. The main characteristics of the topology of the bus were design simplicity, low delay, low consumption and small occupation on the IC.

The inevitable increase of the number of the CPU cores however did not allow use of existing, proven, core architectures and interconnection buses. The inability to use similar techniques in IC with multiple cores, led to a new kind of architectures called many-core architectures. The main difference between multicore processors and many-core processors in term of architectural approaches is the architecture of the cores, the caches hierarchy and the interconnection topology of the bus.

5.3 Architecture of Multicore CPUs

For the execution of parallelized applications, it is required the existence of appropriate hardware must exist. There are several classes of parallel multiprocessors that are distinguished based on the type of the parallel architecture used and the type of communication between the processors.

A classification of parallel architectures according to Flynn, which utilizes the relationship between the executed instructions and the processed data is as follows:

- SISD (Single Instruction - Single Data), where in each cycle a single instruction is performed using a single data stream as an input. Computers belonging to this category are serial and cannot be parallelized.

- SIMD (Single Instruction - Multiple Data), where in each cycle a single instruction is performed using multiple data streams as an input. Computer belonging in this category are distinguished in two categories:

  - Processor Arrays where at each clock cycle multiple processors execute the same instruction on different data.

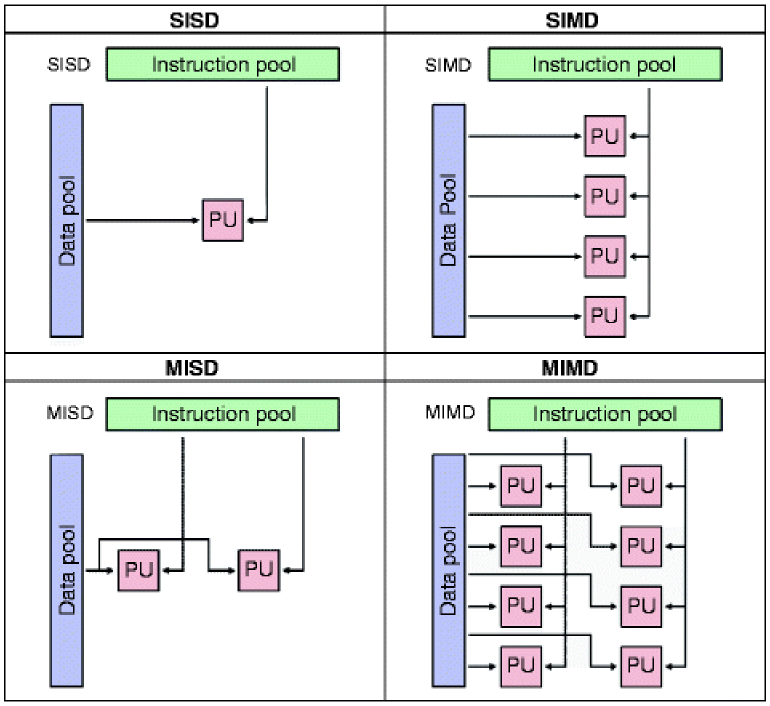


Figure 5.1: Classification of parallel architectures [29]

  – Vector Pipelines where in each clock cycle a whole data vector in one clock cycle is processed instead of one data element.

- MISD (Multiple Instructions - Single Data), where in each clock cycle multiple instructions are executed in parallel using a single data stream.

- MIMD (Multiple Instructions - Multiple Data), where in each clock cycle a different instruction is executed using different data streams. This is the most common category of parallel computers.

The memory architecture is the way in which the processors of a parallel system communicate with each other and affect the implementation of a parallel application. The main memory implementations are shared memory and distributed memory.

With shared memory architectures, processors operate independently but share the same memory through which they communicate. At any time, only one processor can access the shared memory and as a result the synchronization of the read and write instructions is required. The advantage of this architecture is the easy implementation of the applications and the speed of access and the sharing of data via shared memory. The disadvantage is that the bandwidth of the memory is limited so an increase in the number of the processors should require a proportional increase in the memory bandwidth.

With distributed shared memory architectures, processors operate independently and each has its own private memory. Processors communicate via message passing implementations. The advantage of this architecture is the scalability of the size of the memory in relation to the number of processors and the rapid access of each processor to its private memory without contention. The disadvantage of this architecture is the difficulty of matching the existing data structures to this memory organization and the need of the developer to define the messages exchanged between the processors.

5.4 Programming Models for Multicore CPUs

5.4.1 OpenMP Programming Model

The OpenMP standard defines a set of compiler directives to allow parallel execution that can be included in the source code of a program which has been written for a sequential execution. These directives indicate to the compiler how to parallelize the code in a shared memory multicore environment. OpenMP includes also library routines to modify the execution parameters, such as the number of threads.

Before OpenMP, there was not a standard way to parallelize programs in a shared memory environment. Although there was a message passing programming model, developers had to use non-standard and non-transferable API for shared memory. OpenMP changed this situation because:

- OpenMP is a commonly accepted standard. Programs developed under this model are transferable to a wide variety of shared memory multicore environments. The manufacturers of these systems need only implement a well-defined API.

- OpenMP does not require from the developer to change the logic of their programs. Programs can be written using the normal sequential logic and then OpenMP directives can be added without affecting the performance of the serial version. As a result, the work to solve a problem is separated from the parallelization effort.

- The parallelization problem is guided by the user. This means that the compiler does not need to do a thorough analysis of the code, but it is sufficient provided by the user. In this way, the user has complete control over what and how will be parallelized, while rendering the translator simpler in its implementation.

OpenMP makes use of the fork-join parallel model. The utilization of this model programs can generate multiple threads. The thread that begins the execution of the program is called the master thread. The master thread executes the program sequentially until the first parallel region which is indicated by the word parallel. At the beginning of a parallel region the master thread creates a number of threads and each thread executes the instructions written in the parallel region. Apart from the parallel command there are commands that define work sharing regions. Inside there regions each thread is responsible for a part of the job. The master thread continues its execution after the completion of the parallel region while all the other threads are killed.

Chapter 6

Graphics Processing Units (GPUs)

The graphic processing unit (GPU) is a specialized microprocessor that was developed to handle the preparation and processing of, initially, two-dimensional (2D) and, later, three-dimensional (3D) computer graphics, freeing the central processing unit (CPU) from these tasks. The GPU runs in parallel to the CPU. The GPU has specialized parallel processing capabilities. In recent years the increasing need for computational power has extended the use of GPUs, beyond graphics editing, into numerically demanding and computationally intensive algorithms. GPU processors quickly replace specialized parallel architectures with its exponential cost reduction, due to their strong demand in the game industry [2].

6.1 Evolution of Graphics Processing Units

The first GPUs appeared in 1981 by IBM and consisted of 16KB of memory and one RAMDAC [3]. At that time, GPUs was served as an intermediate buffer between the CPU and the monitor. The processing capabilities of GPUs that appeared in 1984, incorporated integrated processing and visualization routines, but they were limited only to 2-D graphics. The increasing demand for 3D games and other applications led to the development of the first modern GPUs in 1990, were the acceleration of the demanding 3-D graphics was achieved using separate graphics cards. By the end of the decade, the 2-D and 3-D processing capabilities of graphics acceleration were merged, and processing pipelines were added to the same integrated circuit, forerunners of modern multicore designs. The first cards with a parallel architecture that allowed programming of vertex and pixel shaders appeared in 2001 (NVIDIA

GeForce3) where each vertex and pixel was processed independently, thus paving the way for General Purposes GPUs (GPGPUs) [3].

6.2 General Purpose GPUs

GPGPUs were suitable for solving problems beyond the graphics processing, undertaking the role of a parallel processor for highly parallelized algorithms and numerical applications (computationally intensive). In these designs the GPGPUs can be used in parallel with the CPU, undertaking the computational load of algorithms that are not suitable for the serial (sequential) logic of long pipelines, being used by CPUs. A specially written subroutine, called the kernel, is run simultaneously and independently on every GPU core and each kernel processes its own set of data. The data are first transferred from the computer's main memory to the graphics card's memory, after which they are processed by the GPU and finally returned back to the main memory. These copies/commands dictate the transfer and processing of chunks of memory between the GPU and the CPU. With the increasing processing power of the GPU, the limiting factor in performance is the requirement for large memory transfers of the algorithms. Programs should anticipate this limitation by minimizing and hiding the communication costs in the design of the algorithms through overlapping processes. Often, it is preferable to repeat calculations on the GPU instead of storing and retrieving data from the memory. To achieve maximum parallelisation, attention should be given to the data storage so that each core can process one independent piece of data, stored in successive memory locations, for faster access. The performance achieved by GPGPUs is much better than CPUs, for algorithms suitable for parallel processing, providing acceleration that previously was achieved only by using expensive architectural systems (supercomputers).

6.3 Architecture of GPUs

GPUs are fundamentally different from CPUs as they are designed to meet different computation requirements: GPUs can execute a large number of arithmetic operations with potential parallelism, with emphasis on the increased throughput, i.e. the calculations capacity in a unit of time, rather than latency, i.e. the processing time from data input to data output of the processed data. The GPU development is

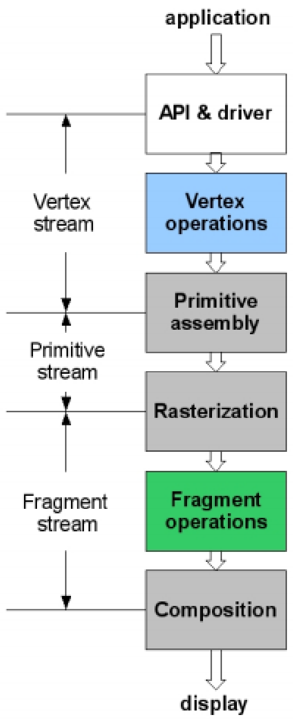


Figure 6.1: Graphics' Pipeline [1]

driven by the way in which the eye process changing images and the number of pixels needed to synthesize a digital image. A typical graphics application implements a series of consecutive calculations called the graphics pipeline, such as vertex and fragment operations, on a large number of input data. To perform this task, a CPU would take a group of the input data and perform the calculations for the vertices, following the calculations for the fragments etc.. All calculations are performed sequentially for each set of data within a short time (low latency for each data set). The graphical pipeline is divided in time, and each step is performed serially.

On the other hand, GPUs sacrifice the low latency and the functionality of a general purpose processor to take advantage of two other features: the ability for parallel processing of multiple sets of data (data parallelism) and the parallelisation of the graphical pipeline, with the simultaneous execution of different stages using different data sets (task parallelism). There are two different kinds of parallelism although the second one has been eliminated in recent years with the advent of Unified Shader Models in modern GPGPUs.

The input of the GPU is a set of geometric objects, typically triangles in a 3-D space, which after a series of processing algorithms mapped to a 2-D image. The graphical pipeline is divided in space and the GPU processes simultaneously different stages, and each part of the processor that performs each step, extract its data in the part of the processor that executes the next segment of the pipeline. In this way, the parts of the processor that execute a specific part of the pipeline can be implemented appropriately in the hardware level in order to be more efficient for the type of calculation they undertake. In addition, apart from the simultaneous execution of the different stages, each stage can accept multiple input and apply the same calculation simultaneously, exploiting the fact that the calculations are the same. This feature means that GPUs are SIMD (Single Input - Multiple Data) implementations where the parallelisation is achieved by the simultaneous execution of the same operations on multiple items. With the introduction of the programmable stages of pixels and vertex shaders described above, the specialized material of the stages was replaced by programmable units without changing the organization and the logic of the pipeline.

The result is a long feed-forward graphical pipeline with many specialized stages, where each calculation may require thousands machine cycles to execute (high latency), but the task and data parallelism are performed on a large number of items achieving high throughput. In contrast to the CPU every action needs a few clock cycles, but only one item or group of items will be processed.

6.4 The Tesla Architecture

Tesla is Nvidia's first dedicated GPGPU. Nvidia calls the multiprocessor's processing units Streaming Multiprocessors (SMs). When a program is executed on the GPU, it generates multiple threads, each of which belongs to a block on a grid. The blocks are identified by a number, each of which is executed entirely in one SM in parallel and independently of the other blocks. Each SM has multiple blocks ready for execution so that once a block completes its calculations to be immediately replaced by another. It is obvious that care must be taken so that the number of blocks overlaps the number of SMs in each card in order to achieve optimum performance.

The overhead for the creation, planning and execution of the threads is practically zero, allowing the creation of blocks with a large number of threads. In addition, the command to synchronize the threads of a block, __synchthreads(), requires only one machine instruction. These two features allow a high discretization of the parallel

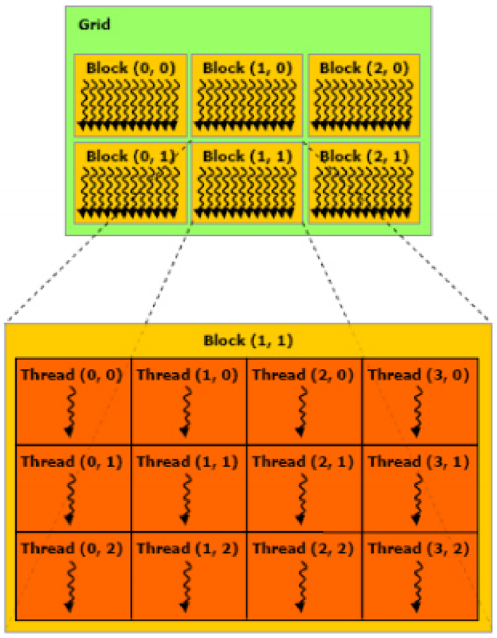


Figure 6.2: Grids and Blocks [1]

process, with the creation of multiple independent threads. Nvidia calls this way of processing SIMT (Single Instruction - Multiple Threads) [1]. Each kernel creates and manages the threads in consecutive groups of 32, called warps. Threads in the same warp start their execution from the same memory address but each also has its own instruction address and registers and continues its execution independently.

If a thread follows a different path of a branching instruction, its execution will be performed independently and in addition (not in parallel) from the others meaning that these branches should be avoided by the clever split of the problem into blocks (or warps as each warp executes independently). Unlike the conventional SIMD architectures where the size of the parallel processing vector might be known at compilation time, the size of the SIMT warp is not needed in order to execute a CUDA program on the GPU. SMs are responsible for the separation and scheduling of the threads. The programmer defines the vector length and the size of the block. Moreover, in SIMT, threads in each warp follow independent command sequences in contrast to vector architectures (vector machines) where the programmer should anticipate and handle exceptions to the common sequence of commands.

For every runtime command, the SM chooses a warp and executes the command simultaneously for all the threads in the warp. The corollary of this is that the blocks

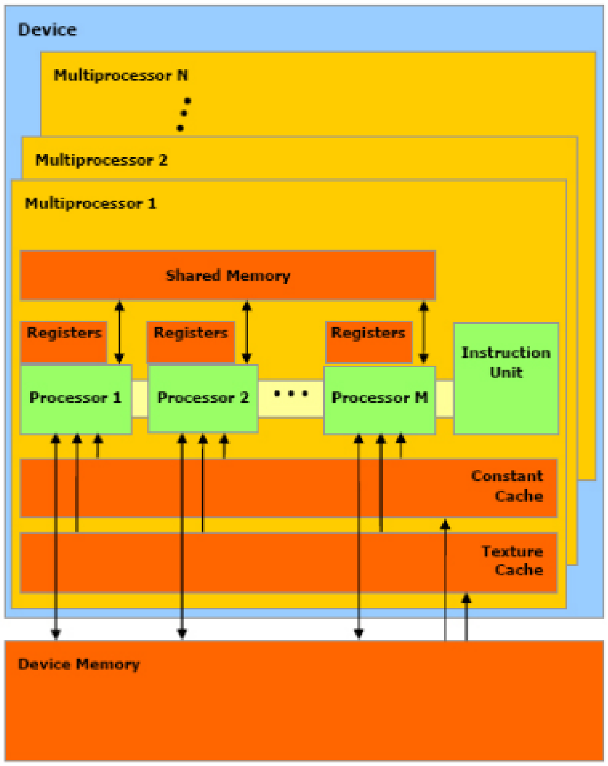


Figure 6.3: Tesla Architecture[1]

should always contain 32 or more threads. Previous research results indicate that 64 threads per block are enough to avoid warps queueing for execution and, as a result, running out of the on-chip memory[5].

The memory available at each SM includes:

- A set of registers
- A shared memory for all cores Single Processors (SPs)
- A constant read-only memory cache for all the SPs that accelerates the readings from the constant memory of the graphic card's main memory
- A read-only texture cache for all the SPs that accelerates the readings from the texture memory of the graphic card's main memory

The main memory of the graphic card is classified as local or global memory and it is not possible to access these memories space through the cache memory.

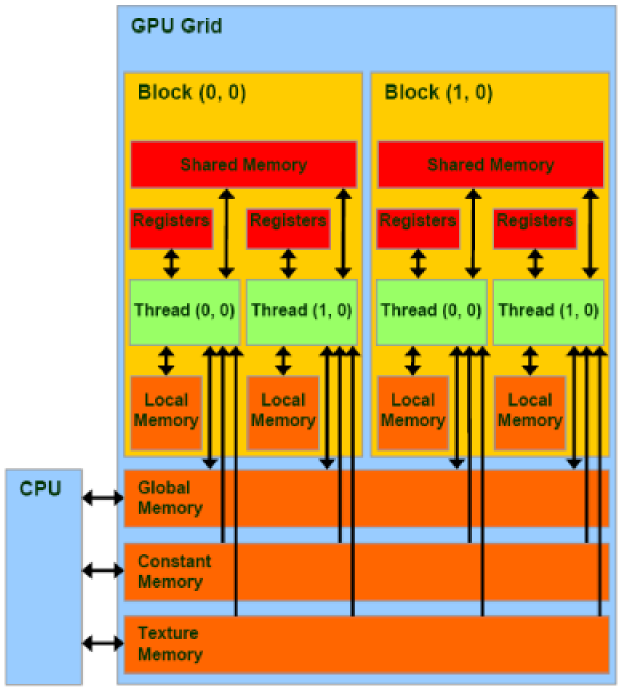


Figure 6.4: Memory Hierarchy [1]

6.5 Programming Models for GPUs

The development of programming tools to facilitate and improve the experience of writing parallel code was the natural next step after the creation of GPUs. In the past, writing a program for GPUs was done using APIs for GPUs with the direct handling of concepts such as texture filtering, vertices, blending, etc. The increasing demand for APIs beyond graphics usage, such as DirectX and OpenCL, led to the extension of high level languages (especially C), for managing the GPU parallel architectures. Early efforts, such as BrookGPU and Sh, let developers handle parallelism using abstract concepts such as streams and kernels without special knowledge of hardware architecture[4]. Today, OpenCL and CUDA are the most popular APIs that have gained widespread. Although CUDA can be used only for Nvidia's graphic cards at present, its wide dissemination and the collection of libraries for mathematical and financial algorithms led to its popularity. A detailed description of CUDA is presented.

6.5.1 CUDA Programming Model

The thread hierarchy, the memory hierarchy and the synchronization of the kernels are three key elements in the CUDA kernel execution. These are handled through appropriate calls to the CUDA API functions regardless of the type of processor being used. Accordingly, the program can be executed on any number of cores and only the graphic card driver needs to know their quantity and the type of the card.

6.5.1.1 Threads Hierarchy

CUDA programmers control the GPU cores by writing specialized C functions, called kernels. Each kernel determines the code that each thread runs (in the core) at the GPU. Accordingly, every thread runs the same piece of code. Kernel requires two arguments and is called with the following command:

```
kernel <<< dimGrid, dimBlock>>> (... parameter list ...);
```

where dimGrid defines the number of threads and dimBlock defines the number of blocks. Each block contains dimGrid number of threads and thus, the total number of threads created is dimGrid x dimBlock. The call of a kernel introduces a relatively small overhead (3-7 ms)[5], while the creation and execution of the threads does not introduce any more costs[1]. Threads within the same block can be synchronized with the command __synchthreads() and they can exchange data through their shared memory, available to each core. On the other hand, threads in different blocks can be synchronized only through atomic memory operations using global memory, which add much delays, and each application should treat them as separate, parallel computational units. The execution time sequence of the blocks is determined exclusively by the control units of the processor and cannot be changed by the programmer. This characteristic enables the parallel implementation of low granularity code, using several parallel blocks, and higher granularity code through the threads of each block.

6.5.1.2 Memory Hierarchy

In CUDA, the CPU-associated memory (RAM) is called host memory, while the memory on the graphics card is called device memory. The kernel code can only process data stored in the device memory. For this reason there are functions to allocate, deallocate, copy and transfer data between host and device memory.

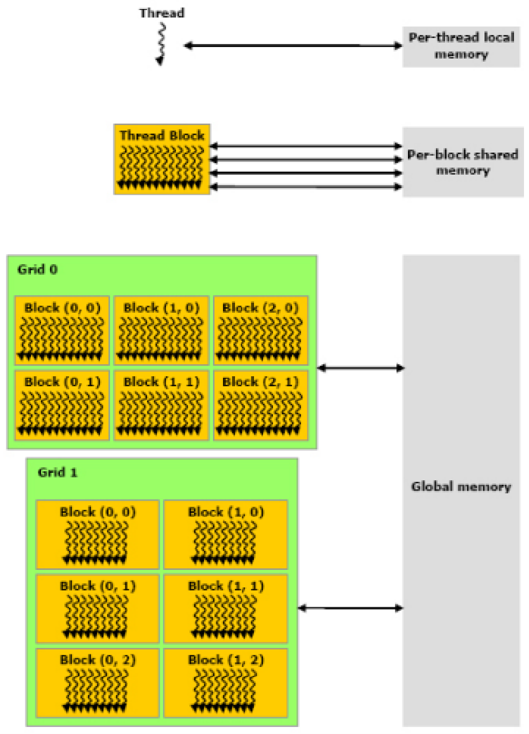


Figure 6.5: Shared Memory Spaces[1]

Threads can access data from various memory spaces of the graphic card. Each thread is assigned a set of registers (See FIgure 6.5). The total size of the register file for each block is 32-64 KB and it is the fastest memory that the thread can access. In addition, every thread in a block has access to a common memory space of 16 KB per core. These two memory spaces are on-chip and thus achieve very high throughput. If there is no need for communication between threads the register memory should always be used because it is faster than the shared memory[5]. Finally, the global or device memory of the graphic card is used to store data and transfer data to and from host memory, while the two separate memory areas, constant and texture memory, are optimized for some specific uses (data filtering etc).[1].

6.5.2 OpenACC Programming Model

OpenACC is a new interface [20], proposed in order to exploit the advantages of GPUs. This language is developed along the lines of OpenMP to combine the simplicity of the OpenMP with the performance of GPUs. OpenACC is considered a high-level language as it has directives that can be added to the C/C++ or Fortran code without the need to re-write or re-structure the code.

In addition, this API does not require the knowledge of the detailed characteristics of the GPU architectures. Regions of code can be easily offloaded to the GPU as the OpenACC directives take care of the memory allocation and transfers. Similar to OpenMP, the developer should include regions of code that are candidates for parallelization inside pragmas and copy clauses and the compiler will automatically port this part of the code to the GPU for execution. Furthermore, an important aspect of OpenACC is its portability. This interface is independent of the hardware and can be executed on every available GPU. It is continuously being improved and, at the last update the utilization of multiple GPUs is available through the creation of multiple CPU threads being done by OpenMP whereby each CPU thread is responsible for one co-processor.

The simplicity of this interface essentially removes from the programmer much of the implementation responsibility. This also means that the developer cannot control most aspects of the architecture as the compiler hides low-level implementation details. Sometime this might have adverse impact on the performance.

Chapter 7

Parallelism Analysis and Performance

7.1 Experimental Environment

The experimental environment consists of 2 (x8 cores) CPUs Intel Xeon E5-2670, with 64 gigabytes DDR3 RAM memory and one GPU card Nvidia Tesla M2050 computing module. The technical specifications of the two environments are presented in tables 7.1 and 7.2. The Intel Xeon E5-2670 is based on the Sandy-Bridge architecture which at the time of writing it is one of the fastest CPUs in the market.

The operating system was Ubuntu Linux 64 bit and the CUDA toolkit version was the 4.2.9. We used the Intel compiler version 12.0.5. In order to better capture the performance of the multiple environments all implementations ran 10 times and the aggregated times were divided by 10. Every measurement was repeated 5 times and the median of these measurements was selected.

7.2 Profiling of the application and description of the parallel algorithm

The identification of compute intensive parts of a computer program is a prerequisite to performance acceleration. For this purpose two stress tests were performed on the

Peak double precision floating point performance peak	515 Gigaflops
Peak single precision floating point performance peak	1030 Gigaflops
Memory clock speed	1.55GHz
Core clock speed	1.15GHz
CUDA cores	448
Memory size (GDDR5)	3 GigaBytes
Memory bandwidth (ECC off)	148 GBytes/sec
Power consumption	225w TDP
CUDA SDK	CUDA Driver API 4.2.9

Table 7.1: Technical Specifications GPU

Memory Size/Type	64Gb
Memory clock speed	1.3ghz/DDR3
Core clock speed (max turbo frequency)	3.0 GHz (3.3Ghz)
Number of cores (in each chip)	8
Number of threads (in each chip)	16
Cache memory size	20 Mb
Memory bandwidth	51.2 GBytes/sec
Power consumption	115w TDP
OpenMP version	2.0

Table 7.2: Technical Specifications CPU

	100,000 paths	500,000 paths
Defining Risk-factors' Intervals	2.55	12.79
Sampling Outer Scenarios	0.0075	0.035
Outer/Inner Simulation	2.92	14.67
Calculation of coefficients	0.060	0.197
Total	5.53	27.69

Table 7.3: Performance Results of Serial Implementation

LSMC algorithm. In the first experiment; 100,000 simulation paths were used for the definition of the risk-factors intervals, 100,000 outer scenarios and 1 inner valuation conditional on each of these outer scenarios. For the second experiment we increased the number of paths to 500,000 to observe the impact on the total time performance at every level. For both experiments the number of steps for each simulation path is 100.

Table 7.3 shows very clearly that the most time consuming parts of the LSMC algorithm the definition of the risk-factors' intervals and the outer/inner loop simulations. Furthermore, the increase of the simulation paths increases the execution time linearly. On the other hand, sampling of the outer scenarios and the calculation of the regression coefficients does not require a lot of time and it is not sensitive to the number of the simulation paths.

Based on the profiling results above the simulation of the risk factors intervals should be parallelized. The number of simulation paths is divided by the number of the available threads. If the number of threads is higher than the number of paths then each thread is responsible for the simulation of one path.

Although the sampling of the outer scenarios takes little time, is done in parallel in Section 7.4.3. As the outer simulations takes place the Monte Carlo paths of multiple outer scenarios are executed in parallel. Due to memory constraints that are explained in Section 7.4 the inner simulation paths for each outer scenario are executed serially.

The last part of the algorithm which is consisted of the calculation of the coefficients is done at the CPU as it does not require a substantial computational power as mentioned above.

7.3 Parallel CPU Implementation

During the experiments, a steady behavior was observed for this implementation's performance. Figure 7.4 shows the speedup achieved for this specific parallel CPU implementation. The presented results were obtained by running 500,000 simulation paths for the definition of the risk-factors intervals at the projection date, 50,000 outer scenarios and 1 inner antithetic valuation. 1, 2, 4 and 8 threads were running at one processor while the 16 threads are distributed in two processors with 8 threads each.

The speedup obtained by this implementation in the first case is very close to linear. More precisely, when up to 4 cores are used the speedup is almost linear while when 8 cores are used the acceleration falls to x7.6. When the number of simulated paths increases this acceleration falls even more to x7. In the case of 12 and 16 threads, where all the available processors are used, the acceleration is x10.15 and x12.51 respectively in the first case and x10.11 and x12.48 in the second case which is very good considering the NUMA (Non Uniform Memory Access) architecture.

It is obvious that as the number of threads increases the relative acceleration falls. This is mostly because of the critical sections used in the first part of the algorithm. Critical sections have been used for the definition of the risk-factor intervals. Inside these sections the execution of the program is sequential and only one thread can access the specified memory address each time. This competition for access between the threads cause delays and as a result impacts the performance of the implementation.

7.4 GPU Implementation and Optimization Techniques

The Monte Carlo simulation that is performed for the definition of the risk-factors intervals is done in the GPU. The random numbers that are required for each path are generated directly in the GPU memory and then a kernel is called for the correlation of these random numbers. This is followed by another kernel for the execution of the paths; calculating the values of the stochastic variables at the projection date.

In the second stage of the algorithm, for each sampled outer scenarios, random numbers for the inner loop simulations are generated in the GPU memory. Again,

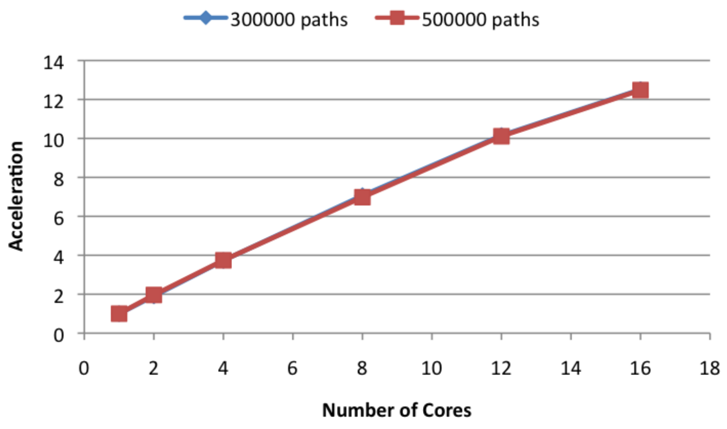


Figure 7.1: Parallel CPU acceleration

a kernel is called for the correlation of the random numbers and another for the simulation of the paths.

It is obvious that many numbers are generated directly to the GPU memory and this creates a limitation on the number of the simulation paths that can be executed in one go. As mentioned in Section 7.1 the inner simulation paths for each outer scenario is done sequentially. An implementation dilemma was whether it is better to lower the limit of the outer scenarios to get the maximum parallelization or execute the inner simulation paths sequentially in order to achieve a higher number of paths.

Due to the complex nature of the liabilities faced by the insurance companies the number of the outer scenarios is crucial for the accuracy of the algorithm. In addition, the goal of the LSMC algorithm is to maintain the accuracy as well as to minimize the number of the inner simulation paths as explained above. These two arguments were the reason of choosing the second implementation method. At the following experiments each inner loop simulation is done sequentially for each outer scenario. In order to make this more clear we will give an example using the Heston model as an outer simulation model and Black-Scholes as the inner valuation model. Let's assume that the outer scenarios are 10,000 and we simulate 2 inner valuations for each of these. The 10,000 outer scenarios are created in parallel. Continuously, in the first run 10,000 inner simulation paths are simulated (one for each outer scenario) and at the second run another 10,000 paths are simulated (again one for each outer scenario). At the end of this simulation we have simulated 20,000 paths in total which was the expected outcome.

7.4.1 Coalesced Memory Access

	Storage efficiency	Load efficiency
CUDA	67.9%	37.00%
ACC	67.9%	53.70%
C-OPT	99.9%	76.70%

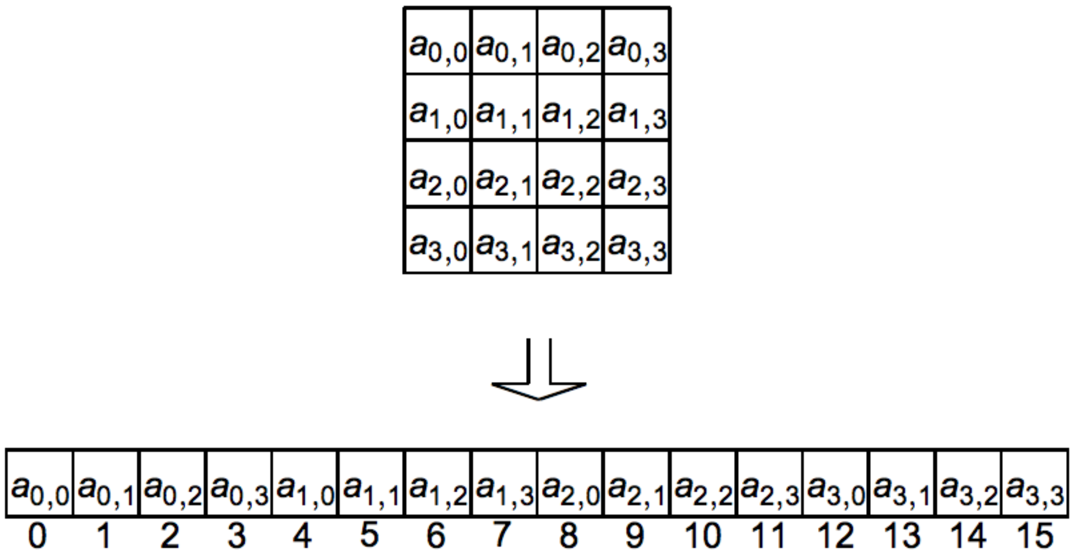
Table 7.4: Global Memory Efficiency

As already mentioned, accesses to the main memory are costly due to the high latency; a global memory access introduces a latency of 400-600 clock cycles. This cost can be significantly reduced if the developer fully exploits the parallel architecture of GPU.

Each access to main memory can transfer data with size 32, 64 and 128 bytes. When a warp executes an instruction that requires data from main memory, the data retrievals are merged into one or more accesses, depending on the size of the requested data and their locations in the main memory. As a result, when the data are not efficiently stored in the memory, the data retrievals cannot be merged and consequently many useless data transfers take place and each of these transfers adds to the latency cost. The effective capability of the main memory is greatly reduced, and this has a significant impact on the overall program performance. For example if each thread needs 4 bytes of data from the main memory, which are not efficiently stored, this generates 32 bytes transfer for each thread and the memory performance is reduced by 8 times.

To achieve the most efficient main memory accesses (memory coalescing) consecutive threads of a warp should access consecutive memory addresses. In this way, all data retrievals from the main memory are merged and the total memory latency is minimized. In order to achieve this, the developer should organize the data, so that the memory address of the data is a function of the thread ID. This storage implementation is not fixed and varies according to the algorithm. In our LSMC GPU implementation this method can be applied to the way we read and store the simulation results in the global memory. Figure 7.1 presents how the C programming language - and CUDA - stores the values of a matrix in the memory. It is obvious that this matrix is stored in the memory in rows.

Let us assume that every row contains the evolution of the simulated price for each thread. This means that the memory addresses $a_{0,0}, a_{1,0}, a_{2,0}, a_{3,0}$ hold the



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Figure 7.2: Matrix storage in memory

initial price of the underlying asset S_0. Because, as we mentioned above, the matrix is stored in rows, these memory addresses are not consecutive and as a result memory accesses cannot be merged and the data is not efficiently stored.

On the other hand, if we store the evolution of the simulated price by columns this means that the addresses $a_{0,0}, a_{0,1}, a_{0,2}, a_{0,3}$ will hold the initial price of the underlying asset. This will improve the storage efficiency in the global memory as consequent memory addresses will be accessed by consequent threads by one merged access. The results at the end of this chapter will reveal the importance of this concept.

7.4.2 Parallel Reduction and Shared Memory usage

An important issue for parallel GPU processing is the calculation of the summary; e.g. the minimum and the maximum of the elements of an array. In order to find the summary of all the elements of an array we could assign the whole task to a single thread and this thread would sum all the elements and store the result in a memory address.

This method would be inefficient as it does not utilize the characteristics of the parallel architecture as only one thread is executing the whole task giving sequential performance. In order to exploit the GPU we use the parallel reduction method. According to this method, in order to add N elements of an array we use N/2 threads, and each will add two array elements. More specifically, the tread with id i

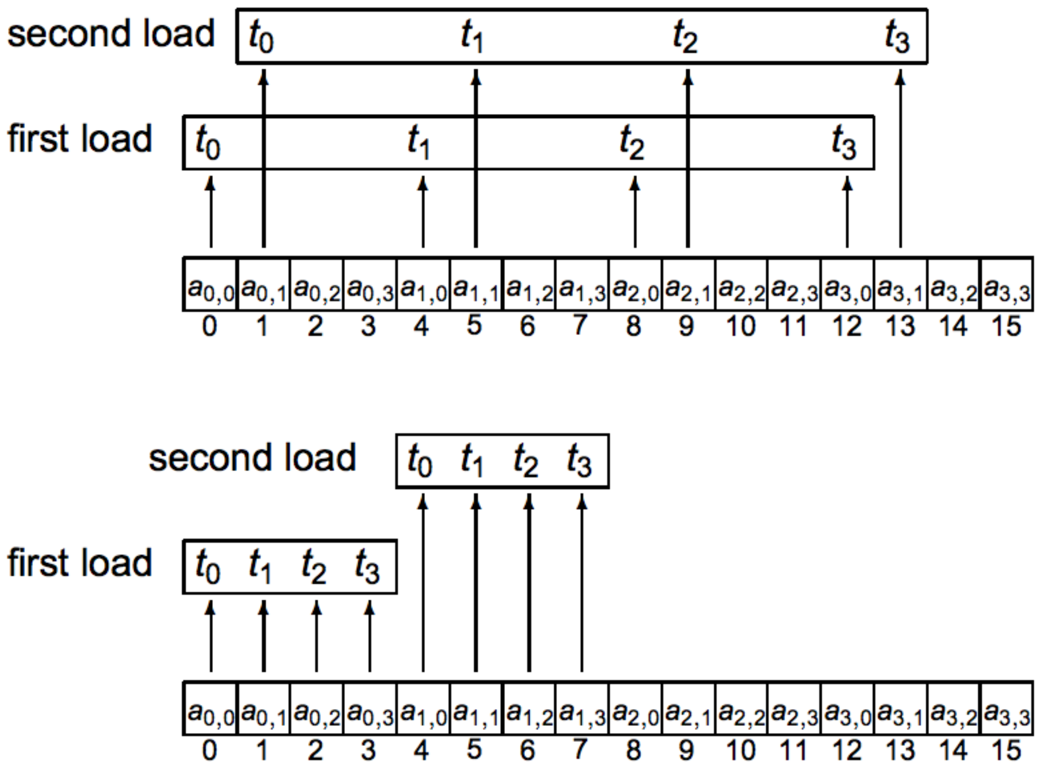

second load
first load
t_0 t_1 t_2 t_3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$ $a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Figure 7.3: Uncoalesced (Up) vs Coalesced Memory(Down)

will add the element located in the address i with the element located in the address $\frac{N}{2} + i$.

The partial sums of the elements of the array will be stored in the first half of the array. Then, the same process will take place but using half the threads used in the first stage, $(N/2)/2 = N/4$. This process is repeated until the number of threads is equal to 1, this will be the last iteration. The result of the parallel additions will be stored in the first memory address of the array.

A very useful and important characteristic of the parallel architecture that be used in the parallel reduction method is the shared memory. The shared memory is on-chip memory, which means it's access is very fast; much faster than other memory types. When it is used properly it provides performance comparable to register.

In order to achieve the maximum performance, shared memory is divided into equally sized parts that can be accessed in parallel - these parts are called banks and they are organized such that 32-bit consecutive words are assigned to consecutive banks. Access to these banks can be done simultaneously with a maximum range of 32 bits per two clock cycles for each bank. So when the request for write or read addresses are to different banks, the performance of the shared memory is maximized and equal to 32*32 bits per two clock cycles. On the other hand, when the memory addresses are in the same bank, accesses are sequential and the performance decreases.

Due to the high performance of the shared memory, it should be used for frequently used data. Its usage starts by copying data from the global memory, carrying out calculations using the shared memory, taking advantage of the multiple banks, and when the calculations are completed the data are transferred back to the global memory. When the size of the shared memory is insufficient, the developer can make use of the read-only constant or texture memory, which are located in the main memory, but they are cached and occasionally offer higher performance compared to the global memory.

Below, we describe how we can apply this concept to improve the performance of the LSMC simulation and some technical optimizations that make use of the shared memory. As described in the previous chapters, in the first phase of the LSMC simulation multiple risk factors scenarios must be created at the projection date. For this purpose, a Monte Carlo simulation should be performed from today to the projection date. This simulation is being done by the GPU and the results are stored in the GPU global memory.

	Uncoalesced	Coalesced	Parallel Reduction
seconds	567.30	91.35	80.34

Table 7.5: Coalesced memory

When this simulation is finished, the developer has the option to transfer these results to the CPU memory and then define the interval of each of these risk factors using the CPU or to use the GPU parallel reduction technique in order to minimize the transfers between the CPU and the GPU. Minimizing the transfers is a general heuristic for every GPU implementation and it should be applied here. When the parallel reduction method finishes, calling the NAG functions we can generate the required scenarios directly in the GPU memory.

There are many ways of implementing the parallel reduction method. The most efficient version, which is applied in this implementation, avoids the divergent warps problem that slows the execution and makes use of the shared memory avoiding bank conflicts for maximum performance. In fact, the cost of the access to the shared memory can be the same as the access to a register if there are not any bank conflicts between the threads. Consequently, if n accesses to the shared memory are stored at n different banks then these memory addresses can be accessed simultaneously. On the other hand, if two or more threads try to read from memory addresses stored at the same bank this will be done serially thus reducing the performance of the shared memory.

7.4.3 CUDA Streams

Data transfer is a critical subject in GPU implementations (and indeed in all computer performance). In the LSMC implementation data transfers can be managed in the following ways. First, the values of the risk-factors scenarios are generated and stored in the CPU memory and then transferred to the GPU or in the second method they are generated and stored directly on the GPU memory.

In order to minimize the duration of the data transfers the use of CUDA streams is required. Stream is an instruction set that is executed sequentially at the device. That means that every device instruction is executed by a stream. If the programmer does not define any streams, memory transfers and kernels executions are executed sequentially by the default stream. If the device is able to perform memory copies and kernel execution at the same time the developer should consider creating more

	First Method	Second Method
ms	155.78	114.42

Table 7.6: CPU-GPU sampling

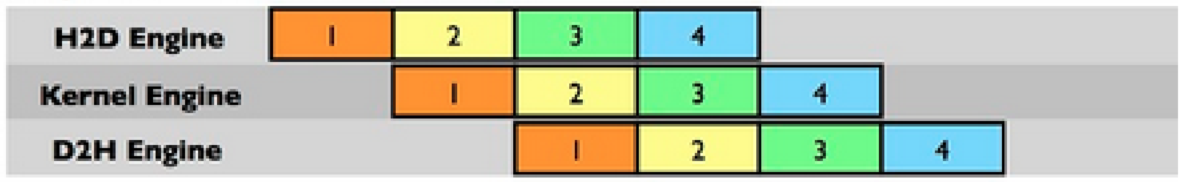


Figure 7.4: Asynchronous Memory Transfer [30]

than one streams to overlap these two processes. Tesla C2050 provides this possibility with one copy engine from host to device, one copy from device to host engine and a kernel engine responsible for kernels execution.

With the first method described above the CPU-generated numbers will be divided into pieces and transferred asynchronously to the GPU device using streams. After the first batch of data is transferred the kernel engine will begin its execution and at the same time the second batch of data will be transferred from host to device. When the kernel engine finishes the first simulation of the first batch of data, the three engines will work in parallel using the device to host copy engine to transfer the simulated data back to the CPU, the kernel engine will simulate the second batch of data and the host to device engine will transfer the next batch of data. This scheme is presented in Figure 7.4. Of course the disadvantage of this method is that the initial generation of scenarios is being done sequentially in the CPU and as a result it is slower than the parallel generation being done by the GPU.

If in contrast we apply the second method, by simulating the risk factors scenarios directly at the GPU we do not need the host to device copy engine as the risk-factors numbers are already in the GPU memory and their transfer from device memory to host will be required at the end of the simulations using the device to host copy engine. On the other hand, the generation of the scenarios is faster because it is executed in the GPU using parallel algorithm.

	No Streams	Streams
ms	114.42	75.65

Table 7.7: Use of streams

7.5 GPU Implementations Comparison

In the previous sections, we described various ways to optimize GPU implementation taking advantage of the parallel architecture. In this section, we present the results of the Heston and Black-Scholes CIR models for the GPU implementations in order to assess and compare these acceleration techniques.

As described in chapter 4 for the Heston Model implementation, the real-world Heston parameters are used to simulate risk-factors on the projection date and the Black-Scholes model was used for the valuation of the liabilities. The Black-Scholes model has only one stochastic variable; the stock price of the underlying asset. Considering the characteristics of the LSMC simulation and the GPU implementation, the outer fitting scenarios consist of only the stock price. These scenarios are generated directly in the GPU and at the end of the inner loop simulation they are transferred to the CPU memory for the least-squares regression. The various GPU implementations for the Heston model are presented in Figure 7.5. The maximum acceleration achieved is x122.76 for the optimized version of CUDA compared to the sequential version and the performance declined as we increase the number of paths. While the use of the coalesced memory has improved the GPU performance, as we increase the number of paths the impact of the memory transfers dominates. Even with use of the parallel reduction algorithm, the problem exists because of the transfers of the risk factors at the second stage of the algorithm.

On the other hand, the speedup of the CUDA naive implementation and the OpenACC implementation is increasing as the number of paths grows. In this case the memory transfers are hidden from the calculations which are slower without use of the coalesced memory. An important outcome of this comparison is that the OpenACC implementation is slightly faster than the CUDA naive implementation. This happens because, as presented in the previous section, the OpenACC compiler makes better use of global memory achieving better storage and load efficiency percentages than the CUDA naive implementation. Considering the ease of use of the OpenACC language this finding is very important for practitioners who need to balance speed up with implementation's cost. Of course, OpenACC cannot compete with the optimization techniques of CUDA but it is a very good alternative when maximum performance is not the only target.

Figure 7.6 shows the speedup of the Black-Scholes CIR model for the various GPU implementations. For this model the maximum speedup achieved is x117.52

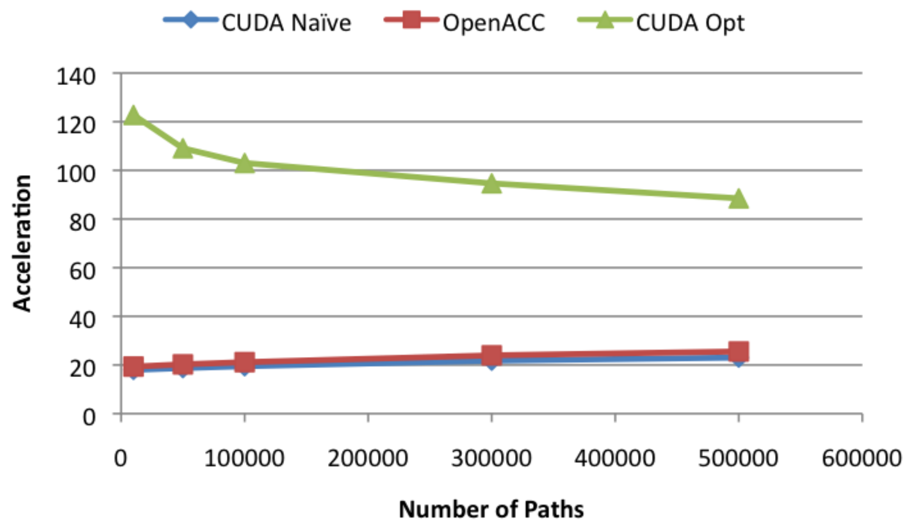


Figure 7.5: GPU Implementations Heston Model

and, as in the Heston model, this number decreases as the number of paths increases. Unlike the Heston model, the same reduction, with a slower rate, occurs for the CUDA naive and OpenACC implementations. As was described in chapter 4, for the implementation of LSMC with the Black-Scholes CIR model we used the same model in both the outer loop and inner loop simulations. In this case, there are two risk-factors (the stock price and the interest rate) and this doubles the numbers of the outer fitting scenarios and the number of the inner simulation paths compared to the previous implementation. When the inner simulation is finished the number of transfers from the GPU to the CPU is higher compared to the Heston model implementation. Consequently, the impact of these transfers is higher and it cannot be hidden even in the CUDA naive and the OpenACC implementations.

Figure 7.7 shows the impact of the addition of one extra risk-factor to the valuation model. The reduction in speedup is higher as we increase the number of paths. The minimum speedup for two risk-factors is x60.33 which is still much higher compared to the OpenMP speedup but these results shows that if the number of risk-factors is higher and the complexity of the valuation model requires many simulation paths to achieve an acceptable accuracy then the choice of the computational platform is not very clear. The OpenMP implementation maintain the same speedup for both models because it does not require any memory transfers.

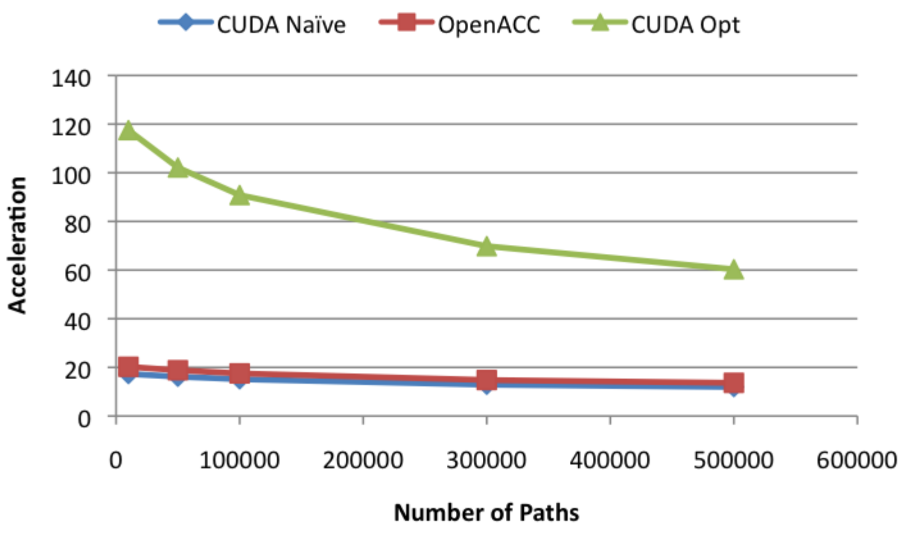


Figure 7.6: GPU Implementations - Black-Scholes CIR

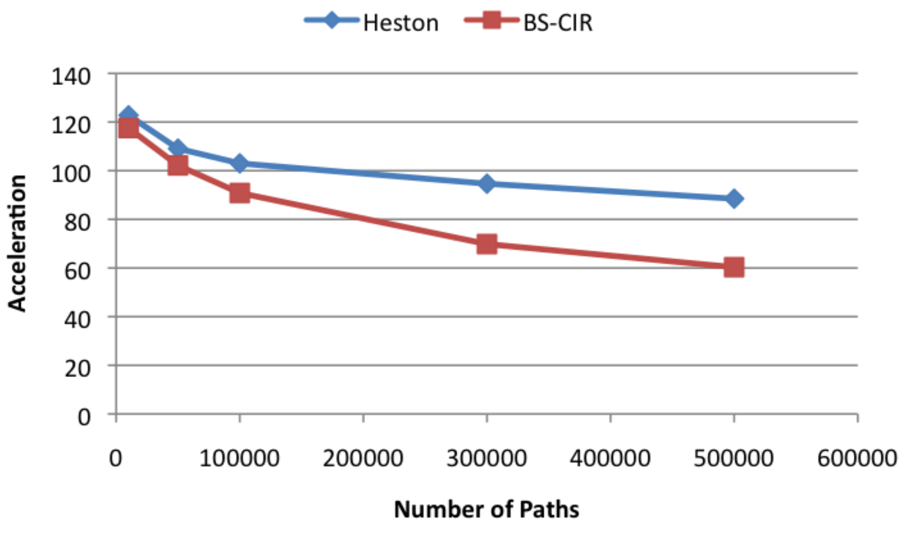


Figure 7.7: GPU Comparison - CUDA Opt version

Chapter 8

Conclusion and Future work

8.1 Conclusion

In this thesis we described the LSMC simulation as a key tool for the calculation of the SCR under the solvency II regulatory framework, by applying this algorithm for the valuation of various options. We explained in detail its function and the various available development choices through the set-up of the algorithm and the impact that these would have in the accuracy of the estimates. The results showed that the LSMC simulation can be an excellent alternative to the stochastic on stochastic valuation as it reduces the required computational effort and at the same time maintains the required accuracy of the estimates in a very good and acceptable level.

Through this analysis, we identified appropriate parallelizable sections of the algorithm. Furthermore, we analyzed the attributes of parallel CPU environments and we described the architecture of GPGPUs. Finally, the LSMC simulation was applied on these parallel architectures using various programming models and optimization techniques to exploit the characteristics of the parallel architectures and thus the acceleration was improved. The results of the implementations are summarized in the following Figure 8.1.

The results showed that the selection of computational platform is not an absolute but can rather depend on the algorithm characteristics. More specifically it depends on the required accuracy, the implementation effort, the available hardware and the complexity and nature of the algorithm. For our implementation the GPU platform provided the greatest speedup. HPC developers introduce new approaches for more efficient use of GPUs for general purpose calculations. In addition, software

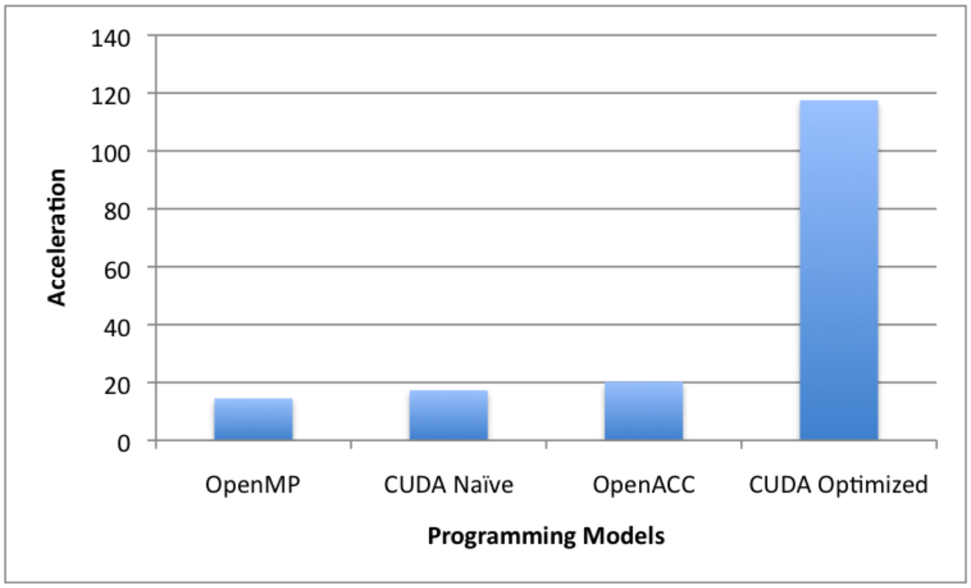


Figure 8.1: Acceleration of the HPC implementations compared to the sequential version

companies are trying to improve the available programming models by improving and simplifying the available APIs. The most difficult implementation problems arise from the legacy codes used by most financial institutions and they are difficult re-write. Software companies are trying to improve the available programming models by improving and simplifying the existing options in order to make it easier to adopt legacy codes and at the same time to not be stacked to a specific hardware. In the same directions, companies that design and implement GPU hardware are trying to increase the performance of their coprocessors by adding features designed only for general purpose calculations.

In summary, nowadays, it is more than obvious that there is a need for integration between finance and HPC implementations. The increasing need for computation arises from the complexity of the financial algorithms makes this integration more vital than ever for the financial institutions if they want to stay competitive under this modern fast-paced environment.

8.2 Future work

In this thesis there were presented results and implementation details from two different scientific fields. There are certain improvements that should be researched in the future in terms of finance and high performance computing.

Regarding the financial part of this thesis the most important research subject regarding the LSMC algorithm is the identification of more efficient basis functions. As already described, basis functions are just polynomials trying to capture the behavior of the payoff of the product. The selection and the efficiency of these basis functions is not clear for complex products. Although these results are promising certain extensions should be done in order to capture the behaviour of more complex products like variables annuities which described in Chapter 2.

In addition, selection of the proxy function should be investigated in order to possibly identify a more efficient algorithm for the definition of the least complex proxy function that will be able to replicate the behavior of different payoffs.

Regarding the HPC part of this thesis the identification of possible vectorizable parts of the OpenMP version would improve the performance of the parallel CPU platform. For the CUDA version the next step is the calculation of the coefficients of the regression function on the GPU. The profiling of the sequential version of LSMC showed that the calculation of the coefficients does not require a great deal of computation and hence its parallelization was unlikely to offer significant speedup. In practice, there was some speedup due to reduction of the required transfers between CPU and GPU memory.

The underlying issue remains that even with the implementation of this part on the GPU there will still be transfers between the GPU and the CPU memory which we cannot be eliminated. Janosek and Nemec [25] discussed the polynomial approximation technique on GPUs. Their implementation calculates elements of the A matrix and the Y vector on the GPU (described in Section 3.2) and these elements are transferred to the CPU memory for completion of the polynomial approximation process. Only a polynomial up to third degree with one variable was considered for these experiments and as a result matrix A did not consist of many elements and consequently the transfer time was small. The drawback of this method is that the implementation of complex regression functions with multiple variables might require substantial implementation effort which will be specific for each different proxy function which makes it useless even for even small changes in the proxy function and further that the number of transfers for this kind of functions will be higher because of the size of matrix A and vector Y. All these hypotheses should be testes in order to investigate if they will improve the performance of the algorithm by making it robust to the increase of the number of the risk-factors or not.

Bibliography

[1] NVIDIA, NVIDIA CUDA Compute Unified Device Architecture Programming Guide, version 2.3, Aug 2009

[2] M. Schmeisser, B. Heisen, M. Luettich, B. Busche, F. Hauer, T. Koske, and K. H. Knauber, Parallel, distributed and GPU computing technologies in single-particle electron microscopy, Acta Cryst. D (65) 659-671 (2009).

[3] R. Bongo, and K. Brodlie State of the Art Report on GPU Visualization, http://www.viznet.ac.uk/reports/gpu/1

[4] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, GPU Computing, Proceedings of the IEEE Vol. 96 No. 5 (May 2008).

[5] V. Volkov, and J. W. Demmel, Benchmarking GPUs to tune dense linear algebra, SC08.

[6] Moore, G. E. Cramming more components onto integrated circuits. Electronics Magazine, v. 38, n. 8, April 1965.

[7] Krste Asanovic, et. all.: The Landscape of Parallel Computing Research: A View from Berkeley, Technical Report No. UCB/EECS-2006-183, December 18, 2006.

[8] B. Dally, "Life After Moore's Law," http://www.forbes.com/2010/04/29/moores-law-computing-processing-opinions-contributors-bill-dally. html

[9] Boyle, P., 1977. Options: a Monte Carlo approach. Journal of Financial Economics 4, 323-338.

[10] Hertz, D., 1964. Risk Analysis in Capital Investments.

[11] Deelstra, G., Delbaen, F., 1998. Convergence of Discretized Stochastic (Interest Rate) Processes with Stochastic Drift Term, Appl. Stochastic Models Data Anal.

[12] Nelson, A., Rodrigues, A., Armada, M., 2008. Improvements to the Least Squares Monte Carlo Option Valuation Method. Review of Derivatives Research 02/2008; 11(1):119-151.

[13] Moreno, M., Navas, J., 2003. On the Robustness of Least-Squares Monte Carlo (LSM) for Pricing American Derivatives. Review of Derivatives Research, Vol. 6, No. 2, 2003.

[14] Glasserman, P., Bin, Y., 2004. Number of paths versus number of basis functions in American option pricing. The Annals of Applied Probability 2004, Vol. 14, No. 4, 2090–2119.

[15] Clement, E., Lamberton, D., Protter, P., 2002. An analysis of a least squares regression method for American option pricing. Finance Stochast. 6, 449–471 (2002).

[16] Koursaris, A., 2011. A Least Squares Monte Carlo Approach to Liability Proxy Modelling and Capital Calculation. barrie+hibbert A Moody's Analytics Company.

[17] Cathcart, M., Morrison, S., 2009. Variable annuity economic capital: the least-squares Monte Carlo approach, Life & pensions, October 2009.

[18] Hammersley, J., Morton, K., 1956. A new Monte Carlo technique: antithetic variates. Journal: Mathematical Proceedings of The Cambridge Philosophical Society - MATH PROC CAMBRIDGE PHIL SOC , vol. 52, no. 03, 1956

[19] Cathcart, M., 2012. Monte Carlo Simulation Approaches to the Valuation and Risk Management of Unit-Linked Insurance Products with Guarantees. School of Mathematical and Computer Sciences, Heriot-Watt University, June 2012.

[20] OpenACC-Standard.org, 2011. The OpenACC Application Programming Interface, Version 1.0, November 2011.

[21] F.A. Longstaff and E.S. Schwartz. Valuing American options by simulation: A simple leastsquares approach, The Review of Financial Studies, 14:113147, 2001.

[22] Yang, Y. 2005. Can the strengths of AIC and BIC be shared?, BIometrika 92: 937-950.

[23] Hull, J., White, A., 1993. Efficient Procedures for Valuing European and American Path-Dependent Options, Journal of Derivatives, 1, pp. 21-31.

[24] European Insurance and Occupational Pensions Authority. http://eiopa.europa.eu.

[25] Janosek, L., Nemec, M., 2012. Fast Polynomial Approximation Acceleration on the GPU, ICDS 2012: The Sixth International Conference on Digital Society.

[26] Kloeden, P., Platen, E., 1992. Numerical Solution of Stochastic Differential Equations.

[27] Higham, D.J 2004. An Introduction to Financial Option valuation: Mathematics, Stochastics and Computation. Cambridge University Press.

[28] Glasserman, P. 2003. Monte Carlo Methods in Financial Engineer- ing. Springer-Verlag, New York.

[29] Calvin, C., Nowak, D., 2010. High Performance Computing in Nuclear Engineering, Handbook of Nuclear Engineering 2010, pp 1449-1517.

[30] Developer's Zone, NVIDIA, 2012. How to Overlap Data Transfers in CUDA C/C++. https://developer.nvidia.com/content/how-overlap-data-transfers-cuda-cc

[31] European Commission, November 2009, Directive 2009/138/EC of the European Parliament and of the Council of 25 November 2009 on the taking-up and pursuit of the business of Insurance and Reinsurance (Solvency II) (Text with EEA relevance)

[32] Basel Committee on Banking Supervision, December 2010 (rev June 2011), Basel III: A global regulatory framework for more resilient banks and banking systems.

[33] Egloff, D., 2005, Monte Carlo algorithms for optimal stopping and statistical learning. Journal: Ann. Appl. Probab. Volume 15, Number 2 (2005), 1396-1432.

[34] Abbas-Turki L., A., Vialle, S., Lapeyre, B., and Mercier, P., 2012. Pricing derivatives on graphics processing units using Monte Carlo simulation, Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cpe.2862

[35] Ledlie, M., C., et al., March 2008. Variable Annuities, Presented to the Faculty of Actuaries, 17 March 2008 and to the Institute of Actuaries, 31 March 2008, British Actuarial Journal, 14 (2, pp. 409-430.

[36] Milevsky, M. and Salisbury, T.S. 2006. Financial valuation of guaranteed minimum withdrawal benets, Insurance: Mathematics and Economics, 38, pp. 21-38.