# Design and Implementation of Hardware Architectures for Pricing Financial Derivatives on Reconfigurable Logic

A dissertation submitted by

**Konstantina G. Miteloudi**

in partial fulfillment of the requirements for the degree of
M.Sc in Electrical and Computer Engineering

Technical University of Crete

2019

| | | | |
|---|---|---|---|
| Committee: | Professor | Dionisios Pnevmatikatos | T.U.C. |
| | Associate Professor | Ioannis Papaefstathiou | A.U.TH. |
| | Professor | Apostolos Dollas | T.U.C. |

# Acknowledgements

I would like to thank my supervisors, Professor Ioannis Papaefstathiou and Professor Dionisios Pnevmatikatos for co-supervising this dissertation. Also, I would like to thank Professor Apostolos Dollas for participating to the committee. All the three of them have been source of inspiration.

Thanks to my partner and colleague Alkaios Sakellaris for everything. Together, we make a strong team.

Finally, most of all, I owe it to my parents for the unconditional love they give me, because they have taught me to dream and support me with patience and perseverance in making my dreams come true.

# Abstract

Option pricing is a fundamental problem in financial sector. This work presents a hardware accelerator on FPGAs for Option Pricing using Crank-Nicolson Finite Difference scheme for solving the Black-Scholes PDE. A variant of Cyclic Reduction called normalized Cyclic Reduction algorithm is used as Tridiagonal Solver. The thesis contains all the theoretical background of option pricing models and finite difference schemes. A literature review had been carried out covering extensively all the FPGA based option pricing accelerators. The effort of this work was concentrated mainly to hardware low-level optimizations that were implemented to produce a parallel system that scale up efficiently. These optimizations, such as custom precision arithmetic, fused operators, pipelined designs etc., were on level of hardware design and had scope to produce a highly parallel hardware architecture. Three different hardware architectures for the main computation core were presented. First a naïve non-pipelined at 32bit precision, next a pipelined architecture using a custom 3 operand adder, also at 32bit precision and finally the proposed hardware architecture with a Fused Multiply Addition operator (FMA or fmadd). This architecture was implemented with 48bit precision, which was selected after taking into account error analysis with MPRF library and design decisions. The implementation was on a Xilinx FPGA device, Ultrascale xcvu9p, and achieved clock frequency 263MHZ.

# List of figures

# List of tables

# Table of Contents

# Chapter 1: Introduction

## 1.1 The derivatives market

Financial derivative products are not an innovation of modern capital and money markets. Their use is extended back in time, many centuries ago, when the Ancient Phoenicians and the Ancient Greeks were selling full shiploads in advance, namely at a predetermined price and delivery in the future. The renaissance of letters and the arts in Europe brought innovations in the markets of the Netherlands (Belgium and Dutch), which formed the center of European trade. In Amsterdam futures contracts date back to the "tulip mania" in the 1630s. During the 1970s and 1980s, the liberalization of foreign exchange markets, as well as the contribution of academics to the pricing of financial derivatives and especially the Options, managed to radically change the landscape and significantly widen their use.

The Chicago Board Options Exchange (CBOE, www.cboe.com) started trading call option contracts on 16 stocks in 1973. Options had traded prior to 1973, but the CBOE succeeded in creating an orderly market with well-defined contracts. Put option contracts started trading on the exchange in 1977. The CBOE now trades options on over 2,500 stocks and many different stock indices. Like futures, options have proved to be very popular contracts. Many other exchanges throughout the world now trade options. The underlying assets include foreign currencies and futures contracts as well as stocks and stock indices.

Financial derivatives today are a powerful tool in the hands of financial institutions and investors and are becoming more and more interesting. The Figure 1 shows the monthly volume of exchange-traded options in trillion of dollars. The data source is from The Bank for International Settlements (www.bis.org), which is an institution in Basel (Switzerland) that acts as a bank for central banks. This figure depicts the last 20 years of option market. As can be seen from the graph, there are three distinct phases:

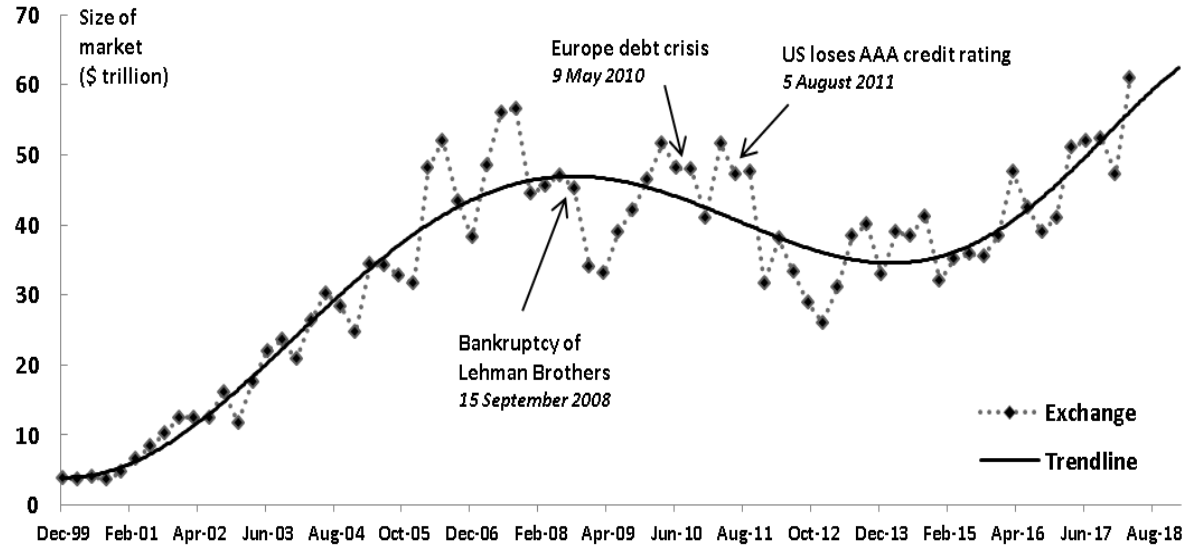
- [phase 1] The period between 2000 and August 2007 saw a dramatic growth in the trade of options. Respectively, there was a steep rise in the volume of transactions from 3.78 to 56.6 trillion US dollars. This is an average increase of 7.5 trillion US dollars per year. Two basic reasons for this escalation were due to new financial instruments and electronic trading. The later has led to a growth in algorithmic trading, which involves the use of computer programs to initiate trades, often without human intervention [1].

- [phase 2] The second period can be defined as the Financial Crisis. Early warning signs of this crisis have been piled up since early 2007. It was caused by the subprime mortgage crisis [2], which itself was caused by the use of derivatives. The most high profile financial incident of this time was the bankruptcy of Lehman Brothers [3] in 15

September 2008. Before the financial markets are able to calm down from the aforementioned events, the Europe debt crisis came with the default of Greece in 9 May 2010 [4],[5],[6],[7]. The United States having not overcome their systemic problems that led them to the 2008 crisis returned to the financial crisis this time in the form of sovereign debt. In 5 August 2011 US lost AAA credit rating for the first time in history [8]. The option market followed these events and its size dramatically shrunk to the lowest level of the last decade, about $ 26 trillion in the last quarter of 2012.

- [phase 3] From 2016 and onwards, as shown in the chart, there is a continuous upward trend in the trading volume of the market. In the first quarter of 2018, it reached the highest historic value of $ 61 trillion in the last 20 years. If this trend continues it will be a new era of growth for the option market.

**Figure 1: Size of exchange-traded derivatives market (Data Source: www.bis.org).**



## 1.2 High Performance Computing in Finance

The finance sector is one of most prominent users of High Performance Computing (HPC). The progress in online applications like news aggregation and analysis and the competition in the field of low-latency and High-Frequency Trading (HFT) required new technologies to keep track with the operational and market demands.

Also, due to the aftermath of the financial crisis in 2008 the computational demands have surged over the last years. New regulations for banks and financial institutes (e.g., Basel III and Solvency II) raised the demand for risk valuation and forced them to deliver valuation and risk simulation results to internal risk management departments and external regulatory authorities frequently.

These changes added up to an important bottleneck in many investment and risk management calculations. The pricing of exotic derivatives in appropriate market models, where no (semi) closed-form pricing formulas exist, and the evaluation is carried out by applying numerical approximations. In most cases, calculating those numbers for a complete portfolio can be very compute intensive and can last hours to days on state-of-the-art compute clusters with thousands of cores.

## 1.3 Motivation and objectives

The main motivation for this work was to produce a hardware solution for option pricing in FPGA. This solution must meet the following criteria:

- ❖ Speed, in terms of how quick can give a value for an option.
- ❖ Accuracy, in the settings that are selected to provide error below the market threshold.
- ❖ Adaptability, the proposed solution can solve different kind of options.

The objectives of this thesis that specialize the above motives can be summarized to the followings:

- ✓ Find and implemented the most suitable FPGA based hardware architecture for Normalized Cyclic Reduction (Norm-CR) algorithm.
- ✓ Incorporate the Norm-CR in the Crank–Nicolson Finite Difference scheme in a highly parallel FPGA design.
- ✓ Provide transparent results in terms of accuracy, timing and utilization for the FPGA implementation.

A secondary objective, that has been created by the volume and interdisciplinary of knowledge needed for the implementation of the above, is this thesis to be a guide for similar scientific research.

## 1.4 Thesis Overview

In chapter 2, the underlying mechanics of financial derivatives are analyzed. The main valuation models are presented, such as the Binomial, the Black & Scholes and the Monte Carlo simulation approach.

Next, in chapter 3, the finite difference (FD) methods are presented and their application in the solution of Black-Scholes PDE. Three different FD schemes are examined Explicit, Implicit and Crank-Nicolson. Also, algorithms for solving tridiagonal linear systems are examined such as tridiagonal matrix algorithm (TDMA or Thomas), Cyclic Reduction (CR) and a variant with normalization of main diagonal of the tridiagonal matrix called Norm-CR. The latter algorithm is used in the hardware implementation.

In Chapter 4, an extensive literature review is contacted covering the field of hardware based accelerators for Option Pricing since 2005 until papers published on August of 2018. All relative literate is presented and statistical analysis is made to see the trends in the field. Two main classes are examined based on the hardware and algorithms used:

- FPGA based option pricing accelerators covering all the financial models such as the Binomial, the Black & Scholes and the Monte Carlo simulation.

- FPGA and GPU implementations of Tridiagonal Solvers.

In our knowledge is the first literature review in the field.

Chapter 5 is dedicated to accuracy analysis of the option pricing procedure. Many experiments where designed and implemented to evaluate the performance of Crank-Nicolson Finite Difference method for solving Black-Scholes PDE. Different metrics of errors are used to see the behavior of arithmetic solution over the analytical. A special experiment is contacted with the Multiple Precision Floating-Point Reliably (MPFR) to determine the floating precision that the algorithm can efficiently operate.

In Chapter 6 the design decisions and the hardware architectures are presented. There are three basic parts in the analysis of this chapter:

- The first part includes the modeling of the Crank-Nicolson scheme.

- In the second part, two hardware architectures are presented, a first naïve which non-pipelined and a pipelined design that exploit a custom 3 operand adder as a fused floating point operator.

- Finally, the last part is dedicated to the proposed design with a fused multiply adder.

Each design had been implemented in different periods of time with different technology of FPGAs. Implementation results are presented for all designs, but only the proposed architecture is evaluated with performance metrics. Also, analysis for operational limits of the

system are discussed combining the accuracy analysis of the previous chapter with the timings results.

Finally, in Chapter 7 the conclusions and future work are discussed.

# Chapter 2:
# Financial Derivatives

## 2.1 Introduction

In general, a financial derivative is a contract that is based on other financial products. In particular, derivatives are contracts between two counterparties that effect a transaction (purchase or sale) of a particular underlying product. The transaction will take place at some point in the future with pre-agreed terms regarding the price and timing of the transaction. The main reason for introducing and using derivatives is the need to develop safeguards against the risk of an investment.

Derivatives are risk reduction tools but if are used for profit-making purposes the financial risk may increase. Typical paradigm of this is the subprime mortgage securitization, which played a big part in the beginning of the Crisis of 2007. The underlying asset(s) can vary from a commodity to a stock market index or an interest rate, an exchange rate etc. Thus, derivatives facilitate the transfer of financial risks.

Common derivatives are options and futures. The simplest call and put options are now so standard that are called vanilla options. Meanwhile, there exist many kinds of options, including the so-called exotic options. These include for example Asian options, which depend on the average price over a period, look-back options, which depend on the maximum or minimum price over a period, and barrier options, which depend on some price level being attained or not. Many of these complicated payoff patterns can be created from trading in the corresponding underlying product and different standard options [1].

The next sections are devoted to option market Mechanics. So that the reader can have the necessary knowledge required for the next chapters of this thesis.

## 2.2 Vanilla Options

An option is a contract that gives its holder the right to buy or sell a particular underlying product at a predetermined price within a given period of time. The value of the right is determined on the basis of the price of the underlying product on which it is based. There are two types of options, call options and put options.

### 2.2.1 Call Option

As its name suggests, a call option gives its holder the right to buy a certain quantity of the underlying asset at a certain price in the future. There are two types of calls, European calls and American calls. The European call can be exercised only at the end of the predetermined period. In contrast, American call option holder can exercise the right to buy at any time within the specified time period.

Essentially, a call is a "bet" between two investors regarding the price of the underlying asset(s). The buyer of the call believes that there will be a rise in the price in the future, while the seller believes that the price of the underlying product will decrease.

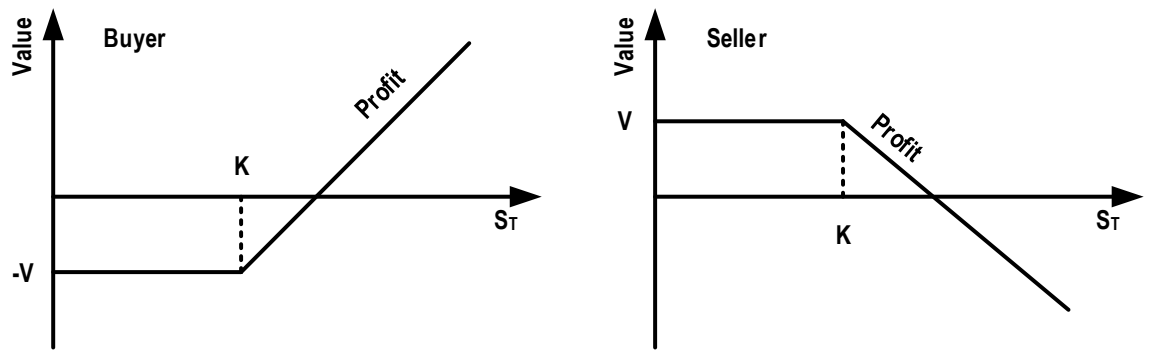
Considering the case of a European call, its buyer has the right to purchase the underlying product at a given price. This value is called strike price or exercise price and henceforth will be denoted as K. On the other hand, the publisher of the call is obliged to sell the underlying asset to the buyer if the latter decides to exercise his right.

When buying a call option, the buyer pays the issuer the value of the option, which henceforth will be denoted as V. Upon expiry of the predetermined period (expiration or maturity of the Call), the buyer may exercise his right to purchase the underlying product or not. The decision it will be taken depends on the relationship between the $S_T$ price of the underlying product at the moment the call option expires and the call exercise price K (**Figure 2**).

In particular, if the price of the underlying product is higher than the exercise price $S_T > K$, then the buyer must exercise the right and buy the underlying product at the value K. After $S_T > K$, obviously the buyer has an inflow of capital equal to $S_T - K$. The net profit is $S_T - K - V$. In the same case the call seller has a net loss equal to $S_T - K - V$.

Let's now examine the case where the price of the underlying asset at the expiration date of call option is less than the strike price $S_T < K$. In this case, the buyer will not exercise its right and the damage will be limited to the amount paid for the purchase of the Call, that is the amount of V. If the right is exercised, this damage will increase by the difference $K - S_T$. On the contrary, the call seller gets a win in this case and his profit is equal to the amount that received from the buyer, that is, V.

**Figure 2: Profit/Loss for a European call option with strike price K.**




## 2.2.2 Put Option

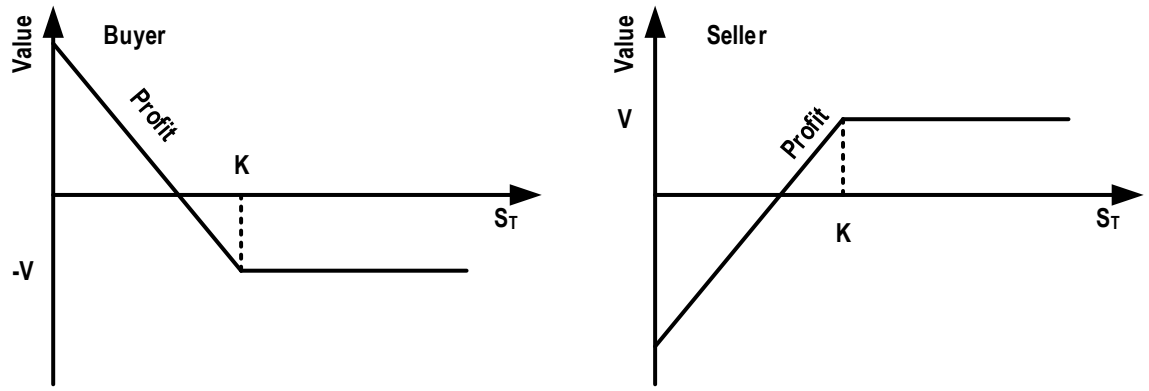
Unlike Calls, the put option gives the holder the right to sell the underlying asset(s) within a predetermined time and at a pre-agreed price. The seller of the Put in this case is obliged to purchase the underlying product from the purchaser of the Put, if the right is exercised.
As in the case of Calls, there are put options of a European type in which the exercise of the right can only be exercised at the expiration of the contract and put options of an American type, where the exercise of the right can be exercised at any time until its expiry. The earnings are determined by the relationship between the $S_T$ value of the underlying asset at the

exercise time and the exercise price K of the underlying (**Figure 3**). The following two cases are more clearly distinguishable:

- $S_T > K$ : In this case, the buyer of the Put will not exercise the right, as he can sell the underlying at $S_T$ , which is higher than the pre-agreed strike price K of the contract. The damage in this case will be equal to the amount paid for the purchase of the right, ie V. On the other hand, the seller of the Put will have an equivalent profit of V.

- $S_T < K$ : In this case, the buyer of the Put will exercise the right, as he may sell the underlying product to the seller of the Put, at a price higher than his actual value. The buyer's net profit will be $K - S_T$ reduced by the option price of V. Instead, the seller has a transaction loss that is equal to the put's owner's profit.

**Figure 3: Profit/Loss for a European put option with strike price K.**




## 2.3 Option Pricing Mechanics

In the following sections, basic options pricing methodologies are presented. The analysis of these models is made for the case where the option under consideration is a European Call. As will be shown, typically American-style Calls are not exercised before their expiration. Consequently, a similar analysis to that presented may be used in the case of the American Type Option. Given a relationship that correlates the value of a European-style Call and the value of a European-style Put, the models can also be used to measure Put. Therefore, before analyzing the two options described above, some basic principles of valuation of options will be presented.

### 2.3.1 Minimum Price of Call Option

Assuming that at some point $t$ an investor has the following two alternatives available:

- **Portfolio 1:** Purchase a call option at $V$ and simultaneously invest $K \ / \ (1 + r)$ with $r$ stands for interest rate until the expiration date.

- **Portfolio 2:** Direct purchase of the underlying asset of the Call option of the alternative 1 in the current value of $S$.

**Table 1** shows the inflows and outflows for both options, both in time t and at maturity. As can be seen, in any case income from portfolio 1 at the maturity of the option exceeds the revenues of portfolio 2. Therefore, the relationship between the values of the two options today should be respectively the same. Such as:

$$V + \frac{K}{(1+r)} > S \Rightarrow V > S - \frac{K}{(1+r)}$$

This relationship determines the minimum price of a call option.

**Table 1: Capital inflows and outflows of investment with call option.**

| Time to maturity | $t = 0$ | $t = T$ | |
|---|---|---|---|
| | | $S_T < K$ | $S_T > K$ |
| Portfolio 1 | | | |
| Buy Call Option | $-V$ | $0$ | $S_T - K$ |
| Investment | $-K\ /\ (1+r)$ | $K$ | $K$ |
| Total | $-V - K\ /\ (1+r)$ | $K$ | $S_T$ |
| Portfolio 2 | | | |
| Total | $-S$ | $S_T$ | $S_T$ |

### 2.3.2 Exercise of American Options

As has already been mentioned, the main feature of American options is that they can be exercised at any time until their expiration. Assume that at some point in time, the value of the underlying asset is $S$ before expiry of the option, and that the underlying product will not yield revenue in the form of coupons (for bonds) or dividends (for shares / indicators). Then with the exercise of the option, the investor will have earnings of $\max\{0,\ S - K\}$. If, on the contrary, proceeds with the sale of the right, it will have revenue:

$$V > S - \frac{K}{(1+r)}$$

Exercising the right makes sense only if $S > K$. But in this case $V > S - K$, so the sale of the right will bring higher returns than its exercise. Therefore, the exercise of the right before maturity should not be preferred by the investor. Alternatively, he may proceed with the sale of the right.

### 2.3.3 Put-Call parity relationship

The value of a European call option is associated with the value of a corresponding European put option through a relationship known as Put-Call parity. Let's assume that at some point in time $t$ two alternative investments are considered. Portfolio 1 concerns the purchase of the underlying asset and a put option with price $V_{put}$ . Alongside, lending capital of value $K/(1+r)$ with interest rate $r$ until the maturity of the put option, where K is the strike price. Portfolio 2 involves buying a call option of value $V_{call}$. The results (capital inflows / outflows) of these two portfolios at maturity are summarized in the **Table 2** below.

**Table 2: Capital inflows and outflows of investment with put option.**

| Time to maturity | $t = 0$ | $t = T$ | |
|---|---|---|---|
| | | $S_T < K$ | $S_T > K$ |
| Portfolio 1 | | | |
| Buy Stock | $-S$ | $S_T$ | $S_T$ |
| Buy Put Option | $-V_{put}$ | $K - S_T$ | $0$ |
| Lending | $K\ /\ (1+r)$ | $-K$ | $-K$ |
| Total | $-S-V_{put} + K\ /\ (1+r)$ | $0$ | $S_T - K$ |
| Portfolio 2 | | | |
| Total | $V_{call}$ | $0$ | $S_T - K$ |

As can be seen, the results of the two alternatives at the end of the time period are absolutely identical and therefore have the same value. Such as:

$$V_{call} = S + V_{put} - \frac{K}{(1+r)} \qquad \textbf{(2.1)}$$

This relationship is particularly useful as it allows the valuation of the price $V_{put}$ of the Put option based on the given value of the Call option and vice versa. This equation (2.1) is known as call-put parity.

## 2.4 Option Pricing Models

The option pricing is one of the most important areas of financial science research, over the last 45 years. This subject is not only of considerable research interest, but also of increased practical interest, as valuation techniques for options and other forms of derivatives are used on a daily basis by portfolio managers. The dissemination of options coincides with the publication of the works of Fischer Black and Myron Scholes in 1973 on the pricing of stock options [9] and Robert Merton [10].

There are four basic families of models in the taxonomy of option pricing methods (Figure 4):

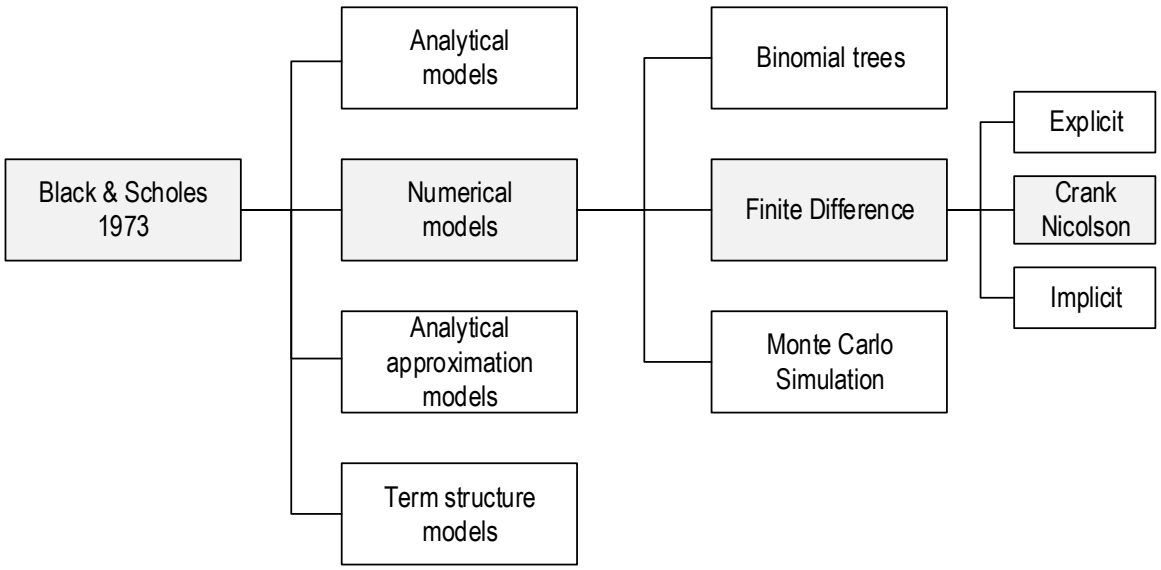
- The analytical models, where a closed form solution exists for valuating, usually a European type option [9], [10].

- The numerical models, where the most known methods are the binomial model [11],[12], finite difference methodology [13],[14],[15] and Monte Carlo simulation [16]. These methods can be used for any kind of option vanilla or exotic.

- The analytical approximation is a technique that combines numerical and analytical models. The value for early exercise option is assessed using a numerical technique and then the premium is added to the price of a European option, which has been obtained from an analytical model. This approach has been used to value American calls and puts on stocks, stock indices, currencies and futures contracts [17].

- Finally, term structure models are mainly used for options on bonds and interest rates. Its solution methods are from analytical and numerical methods [18].

**Figure 4: A taxonomy of option valuation models and methods.**




The highlighted boxes in Figure 4 depict the path this thesis is going to take. Before deepen into the theory of finite difference, it is essential to analyze in brief the basic models that are more commonly implemented in hardware. In following sections the binomial, the Monte Carlo simulation and the Black & Scholes models are presented.

### 2.4.1 The Binomial model

The binomial model [11],[12] is the simplest approach to the valuation of stock options. It is a discrete-time model, under which the analysis is based on changes of the underlying asset at discrete points until the time to maturity of the option.

Suppose that at some time $t$ a portfolio combines a call option and purchase $a$ amount of the underlying asset. The value of the option is $V$ and it has a strike price $K$ and expires after a period of time. The value of the underlying product at the given time $t$ is $S$. During the time period until the expiration of the call option, the value of the underlying product may be increased to $uS$ or reduced to $dS$ , where $u$ and $d$ coefficients are determined based on the corresponding logarithmic returns.

The cash flows of the portfolio at the maturity of the call option are presented in the following Table 3. In this table, $V_u$ and $V_d$ denotes the value of the call in case of increasing and decreasing respectfully, of the value of the underlying. On this base of analysis, the value of a call option can be calculated as follows: $V_u = \max\{0, uS - K\}$ and $V_d = \max\{0, dS - K\}$.

**Table 3: Portfolio cash flows at the expiration of the call option.**

| Time to maturity | $t = 0$ | $t = T$ | |
|---|---|---|---|
| | | Increase | Decrease |
| Write Call option | $V$ | $-V_u$ | $-V_d$ |
| Buy $a$ quantity of asset | $-aS$ | $auS$ | $adS$ |

Again, suppose that the investor is interested in compiling a portfolio so that the result of the investment at maturity is not affected by changes in the price of the underlying product. Such a portfolio is said to offset the risk (hedged portfolio). For such portfolio, it must:

$$-V_u + auS = -V_d + adS \Rightarrow a = \frac{V_u - V_d}{S(u-d)} \quad \textbf{(2.2)}$$

This ratio determines the number of items of the underlying product to be purchased so that portfolio results are not affected by changes in the price of the underlying product and is reported as a hedging factor.

The certain result of this portfolio can thus be achieved by investing some capital for the period until the maturity date of the call with a fixed interest rate of $r$. The $X$ capital to be invested is:

$$X = \frac{-V_u + auS}{1+r} = \frac{-V_d + adS}{1+r} \quad \textbf{(2.3)}$$

If there are two ways (portfolio or investment) to achieve the same result, then their present value should be equal, otherwise it will be possible to make a direct profit, and this is an imbalance that cannot last long. Therefore, it should:

$$V - aS = X \Rightarrow V - aS = -\frac{-V_d + adS}{1+r} \Rightarrow V = \frac{a(1+i)S + V_d - adS}{1+r} \quad \textbf{(2.4)}$$

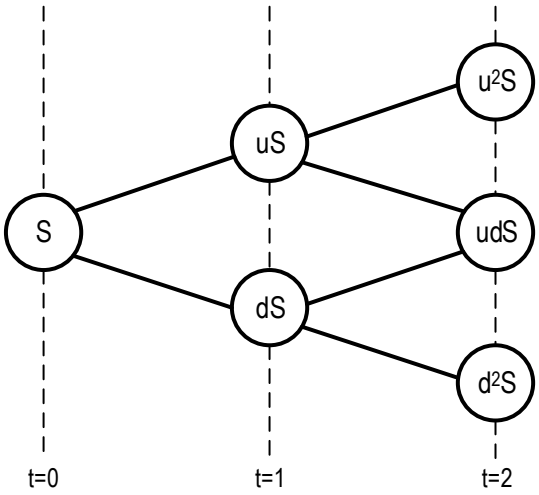
By substituting in this relation the amount of shares that included in the portfolio from the underlying asset, as determined by the relation (2.2), it follows that:

$$V = -\frac{pV_u + (1-p)V_d}{r} \quad \textbf{(2.5)}$$

where $p = (r-d)/(u-d)$.

This relationship determines the value of the call option when one time period remains to expire. On the basis of this relationship and following the same reasoning, it is possible to evaluate the option at any time before its expiry and of course at time 0. In the case where two time periods remain until the expiration of the option, the course of the share until the expiration of the option is given in the Figure 5 below.

**Figure 5: The course of a share stock, two periods for the expiry of the call.**




### 2.4.2 Monte Carlo Simulation

The Monte-Carlo simulation [16] is used to valuate options for the neutral risk. Initially, many yield paths are sampled in order to find the expected return in a neutral risk environment, and then, for this return, the present value is estimated.

Let's consider a financial derivative that depends on a single market variable S and provides some yield in time T. Assuming the interest rate remains stable, the derivative can be valued as follows:

1. Sampling a random path for variable S in a risk-free environment.
2. Performance calculation of the derivative.
3. Repeat steps 1 and 2 in order to estimate as many performance values possible.

4. Calculate the average of the sample of returns to estimate the expected return in a no-risk environment.
5. Redemption of the expected return with the risk-free interest rate at present value.

Suppose that the procedure followed by the variable S in a risk-free environment is:

$$dS = \hat{\mu}Sdt + \sigma Sdz \tag{2.6}$$

Where $dz$ is a Wiener process, $\hat{\mu}$ is the expected return in a risk-free environment and $\sigma$ is the volatility. To simulate the path followed by the variable S, the life of the derivative is divided into N time intervals of size $dt$ and the equation (2.6) is approximated as follows:

$$S(t+dt) - S(t) = \hat{\mu}S(t)dt + \sigma S(t)\epsilon\sqrt{dt} \tag{2.7}$$

Where S(t) denotes the value of S at time t, $\epsilon$ is the random sample of a normal distribution with an average of zero and a standard deviation of one. This process makes it easier to calculate the value of S at time dt from the initial value of S. A simulation involves creating a complete path for S using N random samples from a normal distribution.
In practice, it is usually more accurate to simulate $\ln(S)$ instead of S. From the lemma Itô for $\ln(S)$ it follows:

$$dlnS = \left(\mu - \frac{\sigma^2}{2}\right)dt + \sigma dz \tag{2.8}$$

In order that,

$$lnS(t+dt) - lnS(t) = \left(\mu - \frac{\sigma^2}{2}\right)dt + \sigma\epsilon\sqrt{dt} \tag{2.9}$$

The above equation is used to construct the path for S. The main advantage of the Monte-Carlo simulation is that it can also be used for returns dependent on both the path followed by the variable S and its final value. Also returns can take place at various moments in the time to maturity of the derivative.

### 2.4.3 Black & Scholes model

In a previous section, the binomial valuation model has been analyzed, based on the assumption of discrete time. When the number of periods until the expiration of the option is too high, the discrete time hypothesis can be eliminated and a model valuation in a continuous time can be developed. This model is known as the Black and Scholes model [9], from the names of the two researchers who developed it in 1973 and is the main tool currently used for option pricing.

The development of Black and Scholes model is based on the same logic as the binomial model. It therefore assumes that it is possible to build a portfolio consisting of an option and an underlying asset, so the risk is hedged, by achieving a definite result on the expiry of the

option. Such a portfolio should have the same present value as a capital that moves with a certain interest rate r, until the expiration of the option will have a value equal to the value of the portfolio.

Assume a portfolio that consists of buying $q_s$ shares of the underlying asset and issuing $q_V$ number of option contracts. Then if the value of the underlying product at the time is S and for the option is C, the value P of the portfolio is $P = q_S S + q_V V$. Thus the change in portfolio value for changes in the value of the underlying product and the option (changes denoted as dS and dV respectively) is $P = q_S S + q_V V$. Since the portfolio is supposed to be constructed to offset the risk, the change in its value over any dt period should correspond to the return of a collateral. In particular, if the value P of the portfolio was invested for a dt period with a certain r interest rate, then at the end of the dt period would earn a profit of rPdt. Therefore, in order for the portfolio to be deemed to offset the risk, it should:

$$rPdt = q_S dS + q_V dV \Rightarrow r(q_S dS + q_V dV)dt = q_S dS + q_V dV \quad \textbf{(2.10)}$$

If the investor buys a piece of the underlying product ($q_S = 1$), and at the same time issues $q_V = 1$ pieces of the option, in order for the portfolio to be free of risk, the value of the portfolio must remain stable and independent of changes in the price of the underlying product. Thus it should be:

$$\frac{\partial P}{\partial S} = 0 \Leftrightarrow 1 + q_V \frac{\partial V}{\partial S} = 0 \Leftrightarrow q_V = -\frac{1}{\partial V / \partial S} \quad \textbf{(2.11)}$$

This result is replaced in the relation (2.10), while taking into account that ($q_S = 1$), it appears that:

$$r\left(S - \frac{V}{\partial V/\partial S}\right)dt = dS - \frac{V}{\partial V/\partial S}dV \Rightarrow$$
$$dV = \frac{\partial V}{\partial S}dS - rS\frac{\partial V}{\partial S}dt + rVdt \quad \textbf{(2.12)}$$

Given this relationship, it is now required to define a process that can adequately model the change in the value of the underlying product dS. This is done through the Wiener process. A stochastic process (W = $W_t$: t> 0) is called Wiener, if:

- every $W_t$ of the process at time t is continuous with $W_0 = 0$, and
- the change follows the normal distribution and is independent of the progress of the process until the time t.

Based on this, the change in the price of the underlying product can be adequately attributed by the following relationship:

$$\frac{dS}{S} = \mu dt + \sigma dW \Rightarrow dS = \mu S dt + \sigma S dW \quad \textbf{(2.13)}$$

where μ is the instantaneous expected return of the underlying product, σ the corresponding standard deviation, and dW is a random factor that has the properties of a Wiener process and therefore follows the normal distribution $(0, \sqrt{dt})$ with mean value of zero and a standard

deviation $\sqrt{dt}$. Note that if σ = 0, then the solution of the differential equation (2.13) is $S = S_0 e^{\mu t}$ where $S_0$ is the value of the underlying product at the time t = 0.

Since the value of the option is a function of two variables, the value of the underlying product S and time t, a theorem in the field of stochastic processes can be used. This is the theorem of Ito. This assumes that if a continuous and double differentiable function $f(x,t)$, where x is a random variable that follows a stochastic process of the format $dx = \alpha dt + \beta dW$. Then, according to the theorem of Ito, it is true that:

$$df = \beta \frac{\partial f}{\partial x} dW + \left( \alpha \frac{\partial f}{\partial x} + \frac{\partial f}{\partial t} + \frac{1}{2} \beta^2 \frac{\partial^2 f}{\partial x^2} \right) dt \qquad \textbf{(2.14)}$$

Using this result and replacing $f, x, \alpha, \beta$ with $V, S, \mu, \sigma$ respectively, it follows that:

$$\begin{aligned} dV = & \quad \sigma S \frac{\partial V}{\partial S} dW + \left( \mu S \frac{\partial V}{\partial S} + \frac{\partial V}{\partial t} + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} \right) dt \\ = & \quad \frac{\partial V}{\partial S} (\mu S dt + \sigma S dW) + \frac{\partial V}{\partial t} dt + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} dt \\ = & \quad \frac{\partial V}{\partial S} dS + \frac{\partial V}{\partial t} dt + \frac{1}{2} \sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} dt \end{aligned} \qquad (2.15)$$

By substituting the relation (2.12) it follows that:

$$\frac{\partial V}{\partial t} + rS \frac{\partial V}{\partial S} + \frac{1}{2} \frac{\partial^2 V}{\partial^2 S} \sigma^2 S^2 = rV \qquad \textbf{(2.16)}$$

This is a stochastic differential equation known as the stochastic differential equation of the Black & Scholes model. It is noted that the development of this equation was not based on any assumption regarding the characteristics of the product concerned (in this case the call option). This indicates that the equation (2.16) has a generic value, thus any derivative product whose value is determined by the value of S of the underlying product and the time t satisfies the equation (2.16). In addition, it is noted that in the equation (2.16) the expected return μ of the underlying product is not displayed, but only the variable σ of volatility. This indicates that the value of the contract is independent of the expected return and is determined only by the volatility σ, value S of the underlying product, time and as well as by the interest rate r [1].

#### 2.4.3.1 Closed form solution of Black-Scholes PDE

The solution of the stochastic differential equation (2.16) requires the definition of some boundary conditions that differ according to the financial product under consideration. In the case of European call option pricing, these marginal conditions result from the value of the call option at the end of period T:

$$V(S_T, T) = \begin{cases} (S_T - K), & S_T > K \\ 0, & S_T \leq K \end{cases} \quad \textbf{(2.17)}$$

If at some point in time t before the expiration of the Call, the value of the underlying product is S = 0, then it is apparent from the relation (2.13) that there can be no change in the value of the underlying product, and therefore it is certain that the option will not be exercised at its expiration. Therefore, $V(0,t) = 0$ . On the other hand, if at any time t before the expiration of the call option, the value of the underlying product is $S \to \infty$, then it is almost certain that $(S_T \gg K)$, then will exercise the right by yielding $(S_T - K \approx S_T)$. So, the value of the call option is $V(S,t) \approx S$ when$S \to \infty$.

Under these boundary conditions the solution of the stochastic differential equation of the relation (2.16) gives the value of the call option by:

$$V_{call} = SN(d_1) - Ke^{-rT}N(d_2) \quad \textbf{(2.18)}$$

In addition, using the equation (2.1) which determines the value of a put option in relation to the value of a corresponding call option, the Black & Scholes model can easily be used to determine the value of a put. Since the Black & Scholes model is a continuous time model, the equation (2.1) should be modified using a continuous compounding as follows:

$$V_{call} = S + V_{put} - Ke^{-rT} \quad \textbf{(2.19)}$$

By replacing where the value of the call option according to the model, it follows that:

$$V_{put} = Ke^{-rT}N(-d_2) - SN(-d_1) \quad \textbf{(2.20)}$$

Where:

$$d_1 = \frac{\ln(S/K) + (r + \sigma^2/2)T}{\sigma\sqrt{T}}$$

$$d_2 = \frac{\ln(S/K) + (r - \sigma^2/2)T}{\sigma\sqrt{T}} = d_1 - \sigma\sqrt{T}$$

And

- σ is the annualized standard deviation of the underlying product yields,
- T is the time remaining until the expiration of the call option, expressed in years.

The relations **(2.18)** and **(2.20)** are the mathematical expression of the Black and Scholes model for the European option, call and put, respectively [1].

# Chapter 3:
# Finite Difference and Tridiagonal Solvers

## 3.1 Introduction

In previous chapter, the basic theories of option valuation were presented. The numerical methods for solving the Black and Scholes (BS) partial derivative equation (PDE) are going to be presented. The finite difference (FD) schemes are the explicit, implicit and Crank-Nicolson (CN). Basic concepts of these schemes are analyzed through the approximation of BS PDE.

In addition, implicit and CN schemes for BS model lead to the need to solve linear systems. These linear equation systems can be presented in a form of tridiagonal matrix. Thus, algorithms for solving tridiagonal systems such as LU decomposition, the tridiagonal matrix algorithm (TDMA), also known as the Thomas algorithm, and Cyclic Reduction are described.

## 3.2 Black-Scholes PDE approximation with finite difference methods

The finite difference method is used to approximate differential equations by substituting the latter with difference equations. In this way, the problem ultimately comes to solving a (usually large) system of algebraic equations, usually by applying repetitive techniques. The implementation of this method can be broken down into three basic steps:

a) in the discretization of the space with a suitable node grid, which correspond to the positions of calculating the desired size,
b) in the formulation of difference equations, replacing differential operators with approximate,
c) to solve the difference equations, taking into account the initial and boundary conditions of the problem.

For the first step, the most common type of mesh used is uniform, but different choices prove to be more appropriate in specific cases. Without it being necessary for the discretization step to be the same everywhere, the ease of making a rectangular grid with uniform density throughout the computing space makes it the most common choice (**Figure 6: Discretization mesh of the Black Scholes PDE.**).

Numerical approaches of differential operators are based on the calculation of their values based on the values of the dependent variables at adjacent points. For example, calculating the first derivative of a function f at a point $x_0$ can be approximated by the following expressions, which use values of the function h distance from the point interest (as h is the discrete step):

- Forward difference: $$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h} + O(h)$$

- Backward difference: $$f'(x_0) \approx \frac{f(x_0) - f(x_0 - h)}{h} + O(h)$$

- Central difference: $$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h} + O(h^2)$$

From the above expressions, the third is also the most reliable, since its accuracy is classified as a second order $O(h^2)$ (as opposed to the first order $O(h)$ of the first two expressions) [19]. Note that the precision class of an approach is determined by the class of truncation errors. As for the second derivative of function f, the centrally defined second-order expression has the following three-point form:

- Central difference: $$f''(x_0) \approx \frac{f(x_0 + h) - 2f(x_0) - f(x_0 - h)}{h^2} + O(h^2)$$

The Finite Difference method was applied, for the first time, by the Brennan and Schwartz [13] in the valuation of options. The BS PDE as have been developed in 2.4.3 is:

$$rV = \frac{\partial V}{\partial t} + rS\frac{\partial V}{\partial S} + \frac{1}{2}\frac{\partial^2 V}{\partial^2 S}\sigma^2 S^2 \qquad \textbf{(3.1)}$$

Where:

| | | | |
|---|---|---|---|
| V: | Option price | σ: | Volatility of underlying product |
| S: | Price of underlying asset | K: | Strike price |
| r: | Risk-free interest rate | Payoff: | $\max\{S_T - K, 0\}$ |

The independent variables of the Black-Scholes equation are the value of the underlying product S and the time t that has passed from the signing of the contract. These two variables form a two-dimensional space, where the function describing the valuation of option V (S, t) takes the form of a three-dimensional curve. In this way, instead of trying to find a closed-form expression for the function V (S, t); by the finite difference method each value is approached on the discrete points of the space that form a lattice.

### 3.2.1 Grid selection

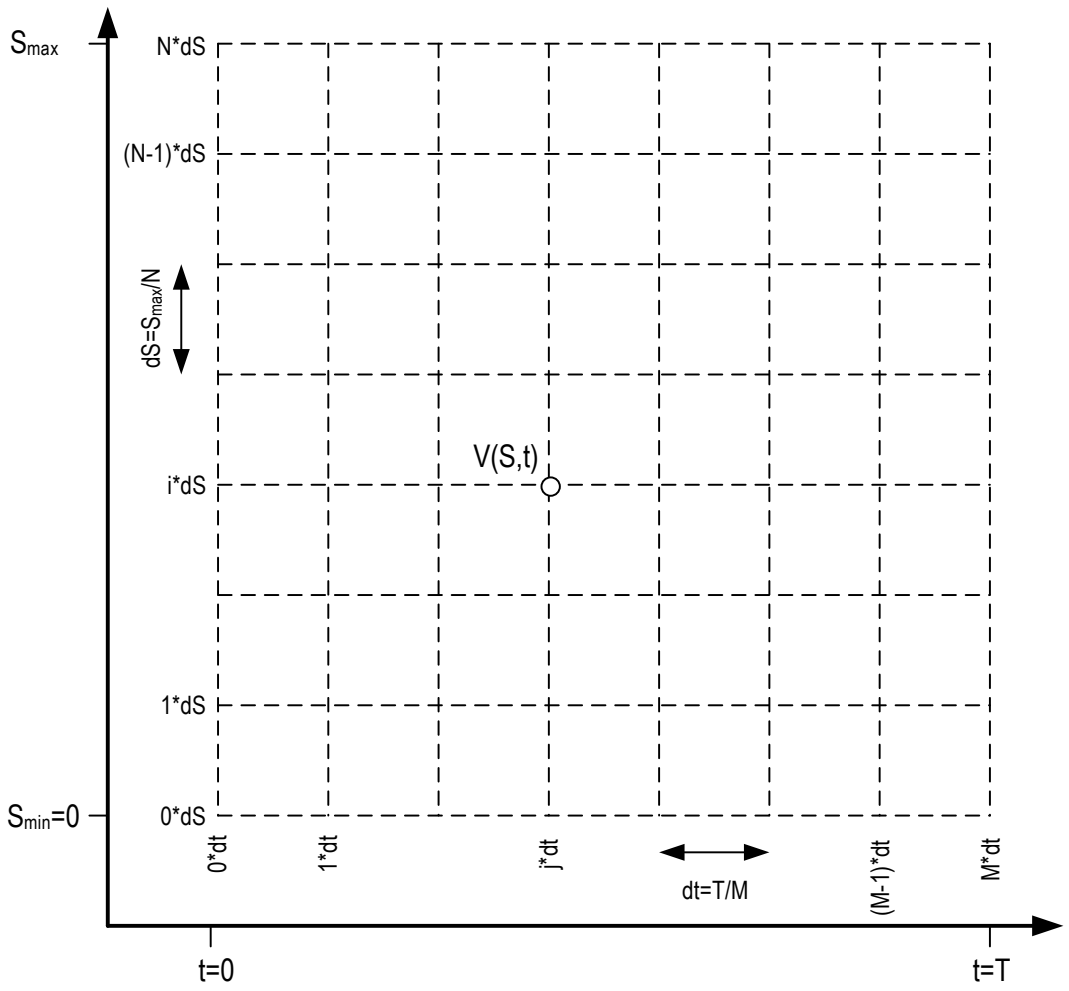
The time t takes values in [0, T] where 0 is the moment of the act of buying or selling the option and T when it matures, that is to say its expiration date. The value of the underlying product S takes values in space [0, ∞] since a product can get a zero or a positive value. In order to be able to define a finite number of distinct points in the product dimension, a maximum value of $S_{max}$ that the product can take must be defined.

In addition to choosing a maximum limit of S, another issue is how many points the grid should have; this often has to do with the type of finite difference method to use. In this case is considered that (S, t) is divided into M point in time dimension and N in the spatial dimension of S, so there is a grid clogged in [0, $S_{max}$] for S and [0, T] for t.

Another element that must been known, before the final form of the grid, is the discretization step. As aforementioned, the most common type of grid used is rectangular because the ease of making a rectangular grid with uniform density throughout the computing space makes it more suitable for its implementation in reconfigurable logic.
For a mesh of uniform density and of size NxM the discretization step is:

- For the dimension of the underlying product: $dS = \frac{Smax}{N}$
- For the time dimension: $dt = \frac{T}{M}$

**Figure 6: Discretization mesh of the Black Scholes PDE.**



The value of the function at the point of the grid ($S_i$, $t_j$) will now be denoted by $V_{i,j}$ and will approximate the actual value at this point $V_{i,j} = V(S_i,\ t_j)$ (**Figure 6**).

So the equation (3.1) takes the following form in the discrete computing space:

$$r\,V_{i,j} = \frac{\partial V}{\partial t} + rS_i\frac{\partial V}{\partial S} + \frac{1}{2}\frac{\partial^2 V}{\partial^2 S}\sigma^2 {S_i}^2 \qquad \textbf{(3.2)}$$

The next step is to apply the finite difference schemes as analyzed in the previous sections to approximate the partial derivatives. Here it is to be noted that in Chapter 2 an analysis was made for all cases of European and American options. So it is necessary to refer again to the boundary conditions of this type of option taking into account the analysis that was made in this chapter with the theory of finite differences.

### 3.2.2 Boundary conditions

The boundary conditions for the Black-Scholes equation in the case of a European call option, expressed in the discrete mesh used by the finite differences and expressing the behavior of the grid boundaries (Dirichlet) are:

For S=0: $$\mathrm{V}(0,\mathrm{t}) = 0 \qquad \textbf{(3.3)}$$

For $S = \mathrm{Smax}$: $$\mathrm{V}(\mathrm{Smax},\mathrm{t}) = \mathrm{Smax} - \mathrm{K} * \mathrm{e}^{(-\mathrm{r}(\mathrm{T}-\mathrm{t}))} \qquad \textbf{(3.4)}$$

Applying the discretization:

For S=0: $$\mathrm{V}_{0,\mathrm{j}+1} = 0 \qquad \textbf{(3.5)}$$

For $S = \mathrm{Smax}$: $$\mathrm{V}_{\mathrm{N},\mathrm{j}+1} = Nds - \mathrm{K} * \mathrm{e}^{-\mathrm{r}(\mathrm{j}+1)\mathrm{dt}} \qquad \textbf{(3.6)}$$

In the second case if the price of the derivative changes linearly in relation to the underlying product, the Von Neumann conditions can be used. For the boundary conditions concerning the value of the derivatives in the system according to Von Neumann:

For S=0: $$\frac{\partial V}{\partial S} = 0 \qquad \textbf{(3.7)}$$

For $S = \mathrm{Smax}$: $$\frac{\partial^2 V}{\partial^2 S} = 0, \frac{\partial V}{\partial S} = 1 \qquad \textbf{(3.8)}$$

Figure 7: Boundary conditions for European call option.

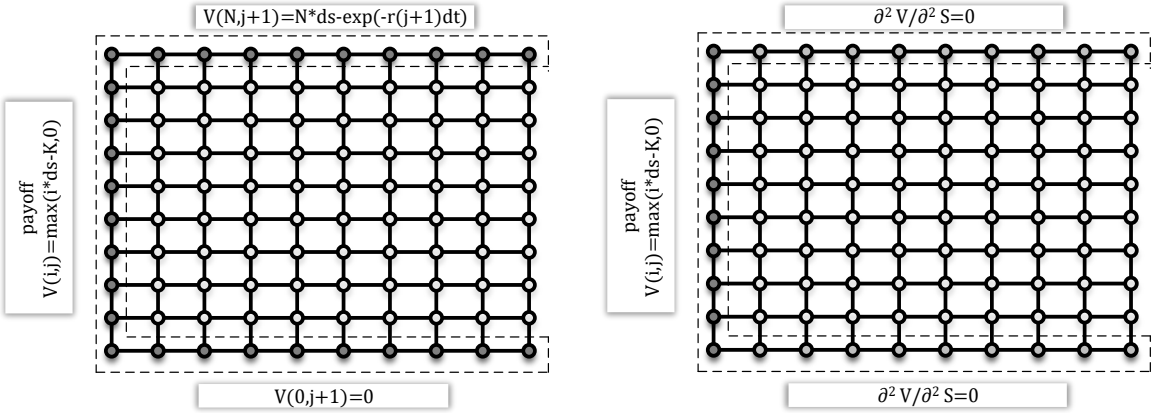



**Table 4: Comparison of boundary conditions**

| | Direchlet | Von Neumann |
|---|---|---|
| Advantages | • More accuracy to the shape of the solution, as their values are known in advance. | • More general and are embedded into the method |
| Disadvantages | • Different for each derivative product<br>• Difficult mathematical formulas<br>• Costly operators for implementation in hardware.<br>• Cost of I/O if implemented in CPU. | • The shape of the solution is not explicitly bound<br>• Truncation Error in time marching<br>• Not applicable if linearity hypothesis does not apply on subject of the differential equations. |

### 3.2.3 Explicit Scheme

The unknown function V in a random node ($S_i$, $t_j$) of the grid, where i denotes the value line of the product and the j time step, is denoted by $V_{i,j} = V(S_i,\ t_j)$. At the same node ($S_i$, $t_j$), the partial derivative of the Black-Scholes equation is approximated with finite difference expressions.

The first derivative over time is replaced by forward difference expression:

$$\frac{\partial V}{\partial t} = \frac{V_{i,j+1} - V_{i,j}}{dt} \tag{3.9}$$

The first and second derivative respect to S, are replaced with center differences. That is,

$$\frac{\partial V}{\partial S} = \frac{V_{i+1,j} - V_{i-1,j}}{2dS} \tag{3.10}$$

And

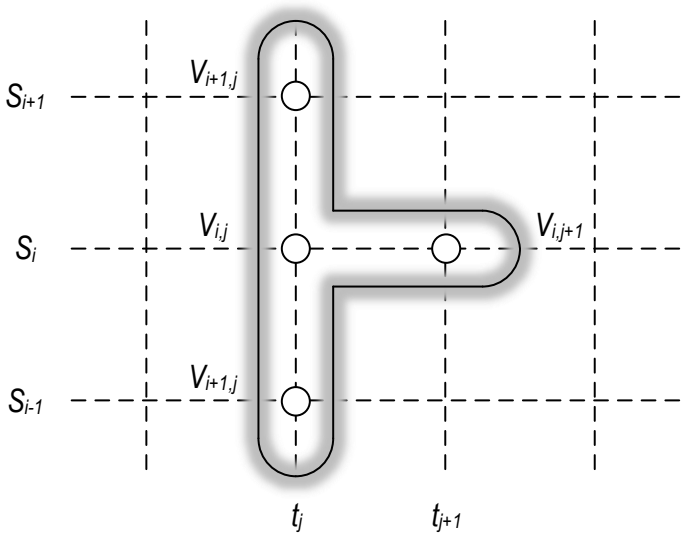
$$\frac{\partial^2 V}{\partial^2 S} = \frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{dS^2} \tag{3.11}$$

Substituting the expressions (3.9)-(3.11) in the equation (3.2) the algebraic equation becomes:

$$r\,V_{i,j} = \frac{V_{i,j+1} - V_{i,j}}{dt} + rS_i \frac{V_{i+1,j} - V_{i-1,j}}{2dS} + \frac{1}{2}\sigma^2 {S_i}^2 \frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{dS^2} \tag{3.12}$$

called finite difference equation.

**Figure 8: Explicit Stencil**




After algebra and separating time steps the equation (3.12) takes the form:

$$V_{i,j+1} = a_i V_{i-1,j} + b_i V_{i,j} + c_i V_{i+1,j} \tag{3.13}$$

Where

$$a_i = \frac{1}{2} \cdot dt \cdot (\sigma^2 i^2 - ri) \tag{3.14}$$

$$b_i = 1 - dt \cdot (\sigma^2 i^2 + r) \quad \textbf{(3.15)}$$

$$c_i = \frac{1}{2} \cdot dt \cdot (\sigma^2 i^2 + ri) \quad \textbf{(3.16)}$$

As shown by the equation (3.13), the calculation of the dependent variable V on the node (i, j + 1) is done directly from the values of V to nodes (i-1, j), (i, j) and (i + 1, j). Schemes such as the formula given by the above equation are named explicit, because moving from one time step to the next is done without the need for solving an algebraic system (Figure 8).

The accuracy of the method can be achieved by limiting the truncation error. The term truncation error is defined as the error made by truncating an infinite sum and approximating it by a finite sum. For example, when using Taylor series expansion, first-order $O(\Delta S)$ schemes neglect all terms after the first derivative while second-order $O(\Delta S^2)$ schemes neglect all terms after the second derivative. The neglected terms constitute the truncation error. Therefore, the higher the scheme order, the lower the truncation error.

In particular, the explicit scheme has spatial truncation error of order $O(\Delta S^2)$ because of the central difference used in S space. And temporal truncation error of order $O(\Delta t)$ because of the forward difference used in time domain.

The necessary condition for the stability of the method is that the factors $a_i$, $b_i$ , $c_i$ should not be negative for all i [13]. It can be proven that ${}^{\partial t}/_{\partial^2 S} \approx 1$ is valid for$b_i > 0$. This condition shows us that the time steps must be equal to the squares of the underlying product. These restrictive conditions create problems in the effective numerical solution of the partial differential equations, which are related to the maximum limit in the size of the time step dt. But when the time step is small, the computation cost is large. This disadvantage is greatly improved by applying the implicit schemes [19].

### 3.2.4 Implicit Scheme

In the implicit scheme the unknown function V in a random node ($S_i$, $t_j$) of the grid, where i denotes the value line of the product and the j time step, is denoted by $V_{i,j} = V(S_i,\ t_j)$. At the same node ($S_i$, $t_j$), the partial derivative of the Black-Scholes equation are approximated with finite difference expressions there are using j+1 points (Figure 9).

The first derivative over time is replaced by finite difference expression:

$$\frac{\partial V}{\partial t} = \frac{V_{i,j} - V_{i,j+1}}{dt} \quad \textbf{(3.17)}$$

The first and second derivative respect to S, are replaced with center differences. That is,

$$\frac{\partial V}{\partial S} = \frac{V_{i+1,j+1} - V_{i-1,j+1}}{2dS} \quad \textbf{(3.18)}$$

And

$$\frac{\partial^2 V}{\partial^2 S} = \frac{V_{i+1,j+1} - 2V_{i,j+1} + V_{i-1,j+1}}{dS^2} \tag{3.19}$$

Substituting the expressions (3.17)-(3.19) in the algebraic equation (3.2) becomes:

$$r\,V_{i,j} = \frac{V_{i,j} - V_{i,j+1}}{dt} + rS_i \frac{V_{i+1,j+1} - V_{i-1,j+1}}{2dS} + \frac{1}{2}\sigma^2 {S_i}^2 \frac{V_{i+1,j+1} - 2V_{i,j+1} + V_{i-1,j+1}}{dS^2} \tag{3.20}$$

After algebra and separating time steps the equation (3.20) takes the form:

$$a_i V_{i-1,j+1} + b_i V_{i,j+1} + c_i V_{i+1,j+1} = V_{i,j} \tag{3.21}$$

Where:

$$a_i = \frac{1}{2} \cdot dt \cdot (ri - \sigma^2 i^2) \tag{3.22}$$

$$b_i = 1 + dt \cdot (\sigma^2 i^2 + r) \tag{3.23}$$

$$c_i = -\frac{1}{2} \cdot dt \cdot (\sigma^2 i^2 + ri) \tag{3.24}$$

The implicit scheme has spatial truncation error of order $O(\Delta S^2)$ because of the central difference used in S space. And temporal truncation error of order $O(\Delta t)$ because of the backward difference used in time domain. The implicit scheme is unconditional stable[19]. The above formulation produces a tridiagonal matrix of the form.

$$\begin{bmatrix} a_1 & b_1 & c_1 & \cdots & \cdots & 0 \\ \vdots & a_2 & b_2 & c_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \cdots & 0 \\ \vdots & \vdots & a_{N-2} & b_{N-2} & c_{N-2} & 0 \\ 0 & 0 & 0 & a_{N-1} & b_{N-1} & c_{N-1} \end{bmatrix} \cdot \begin{bmatrix} V_{0,j+1} \\ V_{1,j+1} \\ V_{2,j+1} \\ \vdots \\ V_{N-2,j+1} \\ V_{N-1,j+1} \\ V_{N,j+1} \end{bmatrix} = \begin{bmatrix} V_{0,j} \\ V_{1,j} \\ V_{2,j} \\ \vdots \\ V_{N-2,j} \\ V_{N-1,j} \\ V_{N,j} \end{bmatrix}$$

The boundary conditions are giving the extra equations that are missing:

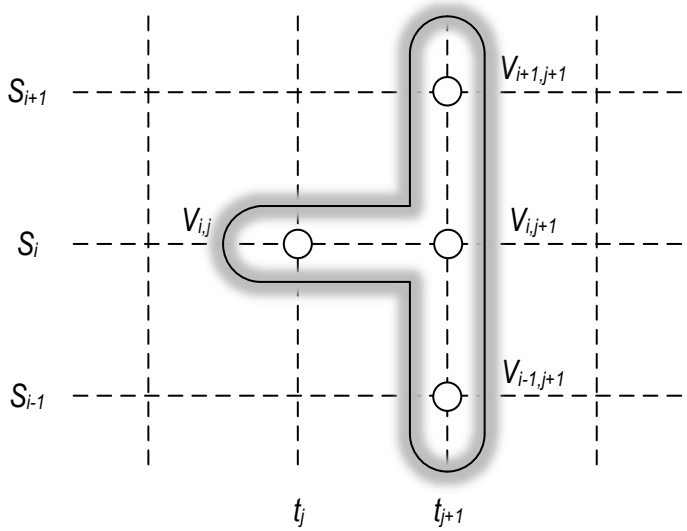
$$V_{0,j+1} = 2V_{1,j+1} - V_{2,j+1}$$
$$V_{N,j+1} = 2V_{N-1,j+1} - V_{N-2,j+1}$$

And the tridiagonal system takes the form:

$$\begin{bmatrix} b_1 + 2a_1 & c_1 - a_1 & \cdots & 0 \\ a_2 & b_2 & c_2 & \cdots \\ \ddots & \ddots & \ddots & \cdots \\ \vdots & a_{N-2} & b_{N-2} & c_{N-2} \\ 0 & 0 & a_{N-1} - c_{N-1} & b_{N-1} + 2c_{N-1} \end{bmatrix} \cdot \begin{bmatrix} V_{1,j+1} \\ V_{2,j+1} \\ \vdots \\ V_{N-2,j+1} \\ V_{N-1,j+1} \end{bmatrix} = \begin{bmatrix} V_{1,j} \\ V_{2,j} \\ \vdots \\ V_{N-2,j} \\ V_{N-1,j} \end{bmatrix}$$

It is often desirable for the error to be of the same order in the variables S and t. So, bigger steps in time marching can be taken with having smaller truncation error. This is accomplished using the Crank-Nicolson method, as it will be seen in the next section.

**Figure 9: implicit Stencil**




### 3.2.5 Crank-Nicolson Scheme

In the explicit and implicit schemes of the previous sections, respectively, the discretization error due to the deduction of terms from the Taylor series is of order $O(\Delta S^2)$ in spatial dimension and of order $O(\Delta t)$ in time dimension. The Crank-Nicolson method [20] has transactions errors of order $O(\Delta S^2)$ and $O(\Delta t^2)$, which are accomplished by approaching the Black-Scholes PDE at the point $(i,\ j+1\ /\ 2)$ between points of $(i,\ j+1)$ and $(i,\ j)$:

$$V_{i,j+1/2} = \frac{V_{i,j+1} - V_{i,j}}{2} \qquad \textbf{(3.25)}$$

Applying central expressions of finite differences in all partial derivatives of first and second order, for the change in time takes the following form:

$$\frac{\partial V}{\partial t} = \frac{V_{i,j+1} - V_{i,j}}{dt} \qquad \textbf{(3.26)}$$

For the change in the price of the underlying product:

$$\frac{\partial V}{\partial S} = \frac{V_{i+1,j+1} + V_{i+1,j} - V_{i-1,j+1} - V_{i-1,j}}{4dS} \qquad \textbf{(3.27)}$$

And

$$\frac{\partial^2 V}{\partial^2 S} = \frac{V_{i+1,j+1} + V_{i+1,j} - 2V_{i,j+1} - 2V_{i,j} - V_{i-1,j+1} - V_{i-1,j}}{2dS^2} \quad \textbf{(3.28)}$$

For the finite difference Crank-Nicolson method, the approximation of differentials for the equation (3.2) becomes:

$$\frac{1}{2}rV_{i,j+1} + \frac{1}{2}rV_{i,j} = \frac{V_{i,j+1} - V_{i,j}}{dt}$$
$$+\frac{1}{2}rids\left(\frac{V_{i+1,j+1} - V_{i-1,j+1}}{2dS}\right) + \frac{1}{2}rids\left(\frac{V_{i+1,j} - V_{i-1,j}}{2dS}\right) \quad \textbf{(3.29)}$$
$$+\frac{1}{4}\sigma^2 i^2 ds^2\left(\frac{V_{i+1,j+1} - 2V_{i,j+1} - V_{i-1,j+1}}{ds^2}\right) + \frac{1}{4}\sigma^2 i^2 ds^2\left(\frac{V_{i+1,j} - 2V_{i,j} - V_{i-1,j}}{ds^2}\right)$$

After algebra and separating time steps the equation (3.29) takes the form:

$$a_i V_{i-1,j+1} + b_i V_{i,j+1} + c_i V_{i+1,j+1} = d_{0_i} V_{i-1,j} + d_{1_i} V_{i,j} + d_{2_i} V_{i+1,j} \quad \textbf{(3.30)}$$

Where:

$$a_i = \frac{1}{4} \cdot dt \cdot (ri - \sigma^2 i^2) \quad \textbf{(3.31)}$$

$$b_i = 1 + \frac{1}{2} \cdot dt \cdot (\sigma^2 i^2 + r) \quad \textbf{(3.32)}$$

$$c_i = -\frac{1}{4} \cdot dt \cdot (ri + \sigma^2 i^2) \quad \textbf{(3.33)}$$

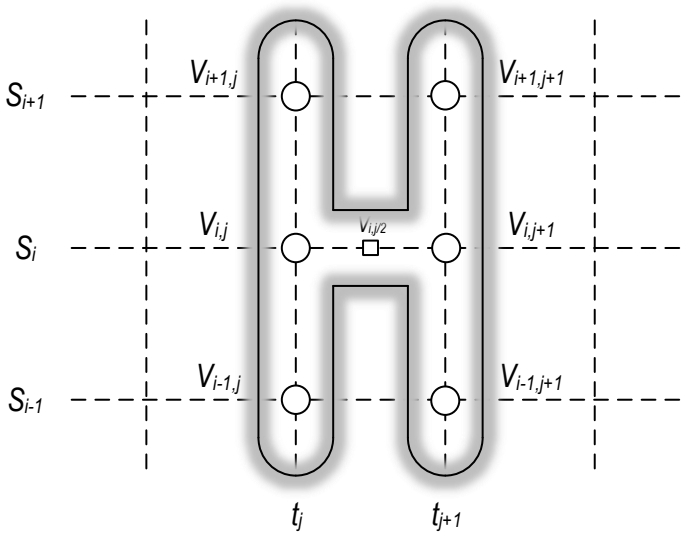
$$d_{1_i} = 1 - \frac{1}{2} \cdot dt \cdot (\sigma^2 i^2 + r) \quad \textbf{(3.34)}$$

$$d_{0_i} = -a_i \quad \textbf{(3.35)}$$

$$d_{2_i} = -c_i \quad \textbf{(3.36)}$$

The Crank- Nicolson stencil is visualized in Figure 10: Crank Nicolson Stencil.

**Figure 10: Crank Nicolson Stencil**




The above formulation produces a tridiagonal matrix of the form.

$$\begin{bmatrix} a_1 & b_1 & c_1 & \cdots & \cdots & 0 \\ \vdots & a_2 & b_2 & c_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \cdots & 0 \\ \vdots & \vdots & a_{N-2} & b_{N-2} & c_{N-2} & 0 \\ 0 & 0 & 0 & a_{N-1} & b_{N-1} & c_{N-1} \end{bmatrix} \cdot \begin{bmatrix} V_{0,j+1} \\ V_{1,j+1} \\ V_{2,j+1} \\ \vdots \\ V_{N-2,j+1} \\ V_{N-1,j+1} \\ V_{N,j+1} \end{bmatrix} =$$

$$\begin{bmatrix} -a_1 & d_1 & -c_1 & \cdots & \cdots & 0 \\ \vdots & -a_2 & d_2 & -c_2 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \cdots & 0 \\ \vdots & \vdots & -a_{N-2} & d_{N-2} & -c_{N-2} & 0 \\ 0 & 0 & 0 & -a_{N-1} & d_{N-1} & -c_{N-1} \end{bmatrix} \cdot \begin{bmatrix} V_{0,j} \\ V_{1,j} \\ V_{2,j} \\ \vdots \\ V_{N-2,j} \\ V_{N-1,j} \\ V_{N,j} \end{bmatrix}$$

The boundary conditions are giving the extra equations that are missing:

$$V_{0,j+1} = 2V_{1,j+1} - V_{2,j+1}$$

$$V_{N,j+1} = 2V_{N-1,j+1} - V_{N-2,j+1}$$

And the tridiagonal system takes the form:

$$\begin{bmatrix} b_1+2a_1 & c_1-a_1 & \cdots & 0 \\ a_2 & b_2 & c_2 & \cdots \\ \ddots & \ddots & \ddots & \cdots \\ \vdots & a_{N-2} & b_{N-2} & c_{N-2} \\ 0 & 0 & a_{N-1}-c_{N-1} & b_{N-1}+2c_{N-1} \end{bmatrix} \cdot \begin{bmatrix} V_{1,j+1} \\ V_{2,j+1} \\ \vdots \\ V_{N-2,j+1} \\ V_{N-1,j+1} \end{bmatrix} =$$
$$\begin{bmatrix} d_1-2a_1 & -c_1+a_1 & \cdots & 0 \\ a_2 & d_2 & c_2 & \cdots \\ \ddots & \ddots & \ddots & \cdots \\ \vdots & a_{N-2} & d_{N-2} & c_{N-2} \\ 0 & 0 & -a_{N-1}+c_{N-1} & d_{N-1}-2c_{N-1} \end{bmatrix} \cdot \begin{bmatrix} V_{1,j} \\ V_{2,j} \\ \vdots \\ V_{N-2,j} \\ V_{N-1,j} \end{bmatrix}$$

The Crank-Nicolson scheme needs to solve a tridiagonal system for each time step and refresh the right-hand member of the equation with new values, where 0 <i <N . Thus, this is the most demanding finite difference scheme than the three analyzed in terms of computational complexity. As mentioned at the beginning of the paragraph, an advantage is the second order precision in temporal change, and the fact that the scheme converges unconditionally. The precision in the direction of the underlying product remains of second order [19]. The Crank-Nicolson scheme is implemented in this thesis.

## 3.3 Algorithms for Solving Linear Equation Tridiagonal Systems

In the previous sections an analysis of Black-Scholes was made using finite difference method. As discussed in the sections (3.2.4) and (3.2.5), the implicit and Crank-Nicolson methods respectively lead to a tridiagonal system of linear equations which have to be solved in every time step. This requires the use of numerical linear algebra methods.

Such systems can be solved directly by more efficient algorithms than those used for more general linear systems, such as the Gauss Elimination [21]. Some algorithms for solving this kind of linear systems are LU decomposition [22], the tridiagonal matrix algorithm (TDMA or Thomas) [23],[24] which is a variant of LU decomposition for tridiagonal systems and Cyclic Reduction [25]. The following sections will present these methods.

### 3.3.1 LU decomposition

The LU decomposition or factorization method refers to the problem of breaking a table A in the form

$$A = LU$$

Where L is lower triangular and the U is the upper triangular table. If the table A of a linear system $Ax = e$ can be broken in this way, then the system is written $LUx = e$ and, as is obvious, its solution is to solve two simple systems: $Ly = e$ for y with forward substitution and $Ux = y$ for x with backward substitution.

The procedure can be summarized as follow:

- Given A, find L and U so that A = LU. Hence LUx = e.
- Let y = UX so that Ly = e. Solve this triangular system for y .
- Finally solve the triangular system Ux = y for x.

The benefit of this approach is that tridiagonal systems need to be solved only once.

#### 3.3.1.1 The tridiagonal matrix algorithm (TDMA or Thomas)

A tridiagonal linear system of size N that has the following form:

$$\begin{bmatrix} b_0 & c_0 & \cdots & 0 \\ a_1 & b_1 & c_1 & \cdots \\ \ddots & \ddots & \ddots & \cdots \\ \vdots & a_{N-2} & b_{N-2} & c_{N-2} \\ 0 & 0 & a_{N-1} & b_{N-1} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-2} \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} e_0 \\ e_1 \\ \vdots \\ e_{N-2} \\ e_{N-1} \end{bmatrix}$$

Thomas algorithm is implemented as shown below:

Algorithm 1: Thomas algorithm

$b'_0 = b_0$
$e'_0 = e_0$

$for\ = 1\ to\ N-2\ do$:
    #decomposition
    $a'_i = a_i / b'_{i-1}$
    $b'_i = b_i - a'_i * c_{i-1}$
    #forward substitution
    $y_i = e_i - a'_i * y_{i-1}$
$end\ for$

$x_{N-1} = e_{N-1} / b'_{N-1}$
$for\ = N-2\ to\ 0\ do$:
    $x_i = (y_i - c_i * x_{i+1}) / b'_i$
$end\ for$

This method is efficient in the case of many systems with common A and different right hand side e, so the factorization is performed only once. The basis for the development of the LU decomposition method is the Gauss method. The computing time for tridiagonal LU factorization is linear with respect to system size. However, this algorithm, though fast and simple, cannot be parallelized without extensive transformations, since there are data interdependencies in each loop of iteration both in the forward replacement, and backward replacement. Only the last two operations in the same loop iteration can be paralleled. Thus, this algorithm cannot scale into massively parallel systems. This, algorithm is used for coarse grain parallelization such as computing many different options simultaneously [26].

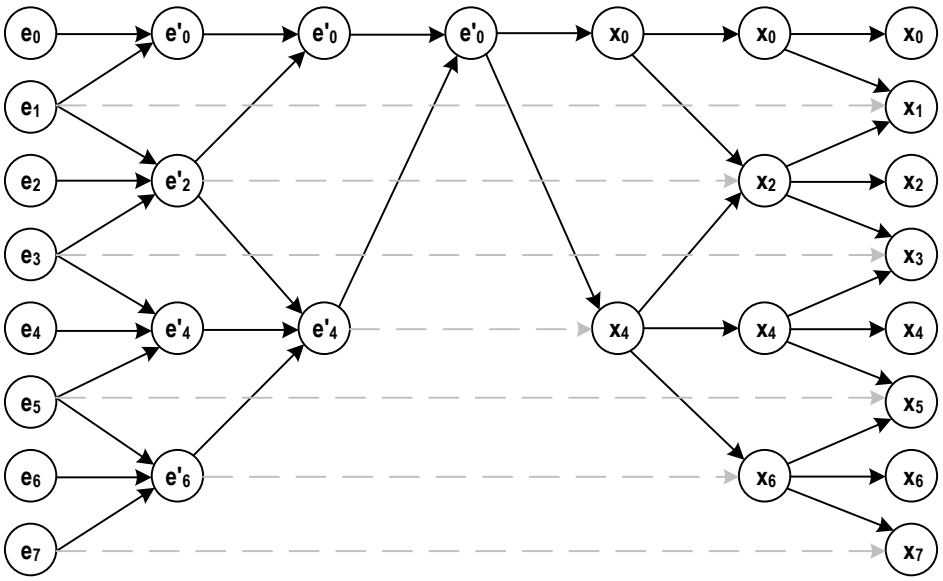
Another algorithm for solving tridiagonal systems is Cyclic Reduction, which together with a variation is described in detail in the next section.

### 3.3.2 Cyclic Reduction (CR)

Cyclic reduction is a family of methods for solving tridiagonal systems presented by Hockney [25]. The basic idea behind these methods is to transform a system of linear equations into two (or more) systems of linear equations that are independent of each other so that they can be solved independently.

The Cyclic Reduction algorithm is an iterative technique for solving tridiagonal systems. The algorithm is divided into two phases, initially removing half of the unknowns, ie the even elements of the table and creating a new system of half size only with the odd unknowns, until it reaches a system of size one, where it can easily be calculated the unknown variable. It is then calculated by reversing the matrix elements that were deleted, using the already calculated units. These two phases are called forward reduction and backward substitution respectively.

**Figure 11: Cyclic Reduction calculation scheme.**




The operations of the algorithm for each step of forward reduction, is done by the following equations:

$$a_i' = -\frac{a_i \cdot a_{i-1}}{b_{i-1}} \quad \textbf{(3.37)}$$

$$b_i' = b_i - \frac{a_i \cdot c_{i-1}}{b_{i-1}} - \frac{c_i \cdot a_{i+1}}{b_{i+1}} \quad \textbf{(3.38)}$$

$$c_i' = -\frac{c_i \cdot c_{i+1}}{b_{i+1}} \quad \textbf{(3.39)}$$

$$e_i' = e_i - \frac{a_i \cdot e_{i-1}}{b_{i-1}} - \frac{c_i \cdot e_{i+1}}{b_{i+1}} \quad (3.40)$$

**Or:**

$$a_i' = -a_{i-1} \cdot k_1 \quad (3.41)$$

$$b_i' = b_i - c_{i-1} \cdot k_1 - a_{i+1} \cdot k_2 \quad (3.42)$$

$$c_i' = -c_{i+1} \cdot k_2 \quad (3.43)$$

$$e_i' = e_i - e_{i-1} \cdot k_1 - e_{i+1} \cdot k_2 \quad (3.44)$$

**Where:**

$$k_1 = \frac{a_i}{b_{i-1}} \quad (3.45)$$

$$k_2 = \frac{c_i}{b_{i+1}} \quad (3.46)$$

After a step of CR forward reduction, redundant unknown variables and zeros can be removed, and a half-size matrix is formed of the remaining unsolved equations. Each step of the backward substitution, defined in Eq. (3.47), solves for unknown variables by substituting solutions obtained from the smaller system.

$$\boldsymbol{x_i = \frac{e_i' - a_i' \cdot x_{i-1} - c_i' \cdot x_{i+1}}{b_i'}} \quad (3.47)$$

Figure 11 depicts the calculation scheme for system size N=8. The total steps for this procedure are $2*\log_2(N)$.

#### 3.3.2.1 Cyclic Reduction with normalized diagonal (Norm-CR)

A variant of CR algorithm [27] is to normalize the main diagonal of the system to the unit by dividing all original coefficients of the system with $b_i$.

$$\begin{bmatrix} 1 & c_0/b_0 & \cdots & 0 \\ a_1/b_1 & 1 & c_1/b_1 & \cdots \\ \ddots & \ddots & \ddots & \cdots \\ \vdots & a_{N-2}/b_{N-2} & 1 & c_{N-2}/b_{N-2} \\ 0 & 0 & a_{N-1}/b_{N-1} & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-2} \\ x_{N-1} \end{bmatrix} = \begin{bmatrix} e_0/b_0 \\ e_1/b_1 \\ \vdots \\ e_{N-2}/b_{N-2} \\ e_{N-1}/b_{N-1} \end{bmatrix}$$

The operations of the *Norm-CR* algorithm for each step of forward reduction, is done by the following equations:

$$temp = \quad 1/(1 - a_i \cdot c_{i-1} - c_i \cdot a_{i+1}) \tag{3.48}$$

$$a_i' = \quad -a_i \cdot a_{i-1} \cdot temp \tag{3.49}$$

$$c_i' = \quad -c_i \cdot c_{i+1} \cdot temp \tag{3.50}$$

$$e_i' = \quad (e_i - a_i \cdot e_{i-1} - c_i \cdot e_{i+1}) \cdot temp \tag{3.51}$$

And for the backward substitution:

$$x_i = \quad e_i' - a_i' \cdot x_{i-1} \; - c_i' \cdot x_{i+1} \tag{3.52}$$

### 3.3.3 Comparison of tridiagonal solver algorithms

The Thomas algorithm is given in algorithm 1, it has a complexity of O(N) and requires a total of 8N arithmetic operations to solve an N-tridiagonal system. Cyclic reduction requires log(N) steps, each of which requires O(N) operations, so total work is O(N log(N)). Serially, cyclic reduction is therefore inferior to Thomas, which require only O(N) work for a tridiagonal system. But in parallel, cyclic reduction can exploit up to N-fold parallelism and requires only O(log(N)) time in best case.

The number of elementary operations of Thomas, CR and the implemented Norm-CR, for a loop pass per phase can be seen in Table 5. Table 6 depicts the total number of operations for the aforementioned algorithms. It can be observed that the changes in CR algorithm by normalizing the main diagonal have increased the computational load from 17(N) to 18(N). Nevertheless, the divisions with high latency from three have been reduced to one.

**Table 5: Number Of Operations For A Loop Pass Per Phase**

| | Forward | | | Backward | | |
|---|---|---|---|---|---|---|
| | #mult | #add/#sub | #div | #mult | #add/#sub | #div |
| Thomas | 2 | 2 | 1 | 1 | 1 | 1 |
| CR | 6 | 4 | 2 | 2 | 2 | 1 |
| Norm_CR | 9 | 4 | 1 | 2 | 2 | 0 |

**Table 6: Total Number Of Operations**

| | #mult | #add/#sub | #div | #total |
|---|---|---|---|---|
| Thomas | 3(N) | 3(N) | 2(N) | 8(N) |
| CR | 8(N) | 6(N) | 3(N) | 17(N) |
| Norm_CR | 11(N) | 6(N) | 1 (N) | 18(N) |

Here, it is worth noting that by applying the normalization there is a gain of one operator with lower latency, with trade-off in accuracy and stability of the entire system. Changing the properties of the tridiagonal matrix must be handled with concern. Because there weren't any accuracy analysis, in bibliography, for this variant of Norm-CR a precision and robustness analysis of the entire system can be found in Chapter 5:.

Before that, in the next chapter there is an effort to record all the previous work that has been done in the field.

# Chapter 4:
# Hardware Acceleration in Option Pricing and related work

## 4.1 Introduction

In this chapter an attempt is made to record the pre-existing research in the field of acceleration option pricing with hardware. The research is focused on papers with FPGA's implementations or designs of all option pricing models. There is also an extensive analysis of the related literature on our approach to the problem. In the last sections of this chapter, hardware systems that have applied solvers for tridiagonal systems of linear equations are recorded.

The contribution of this work is to document prior research in the field since the last decade, so the reader can have direct access to the information and develop comprehensive view of the field.

## 4.2 FPGA based option pricing accelerators

This section records the research that has been done since 2005 in acceleration of option pricing on reconfigurable logic devices (FPGAs). The collection of bibliographic references has been made from many different sources and from our knowledge is the first review in the field.
The references for FPGA based option pricing accelerators are presented in chronological order, so it can be obvious to identify the evolution in the field. In Table 7 the aggregated information is presented with seven columns. The first two columns refer to the citation and year of the publication. The third column shows the financial pricing models that have been implemented.

The abbreviations used:

- MC → Monte Carlo simulation,
- QUAD → QUADrature,
- BT → Binomial Tree,
- TT → Trinomial Tree,
- E-FD → Explicit Finite Difference,
- I-FD → Implicit Finite Difference,
- CN-FD → Crank -Nicolson Finite Difference.

The other columns have information's about the specific implementation of the proposed methods, such as the targeted device, frequency of the design, the Hardware Description Language or High Level Synthesis tools. The last column "Acc" stand for accuracy of the design or algorithm, it is "yes" if the work contains accuracy analysis.

**Table 7: Chronologically sorted FPGA based option pricing accelerators.**

| Reference | Year | Model | Device | Freq | Method | Acc |
|---|---|---|---|---|---|---|
| Zhang et al. [28] | 2005 | MC | XC2VP30 (90nm) | 50Mhz | VHDL | No |
| Thomas et al. [29] | 2006 | MC | XC4VSX55 (90nm) | - | Handel-C | No |
| Morris, Aubury [30] | 2007 | MC | XV4LX160 (90nm) | 61Mhz | Handel-C | Yes |
| Tian et al. [31] | 2008 | MC | XV4FX100 (90nm) | 53Mhz | Verilog-HDL | Yes |
| Tse et al. [32],[33],[34] | 2009 | QUAD | XC4VLX160 (90nm) | 100Mhz | Handel-C | No |
| Jin et al. [35],[36] | 2009 | BT,TT | XC4VSX55 (90 nm) | 76Mhz | Handel-C | No |
| Jin, Thomas, Luk [37] | 2009 | E-FD | XC4VLX160 (90nm) | 106Mhz | Handel-C | No |
| Wynnyk, Magdon-Ismail [38] | 2009 | BT | EP3SE260 (65nm) | 150Mhz | Verilog-HDL | No |
| Tian , Benkrid [39] | 2010 | Q-MC | XC4VFX100 (90nm) | 180Mhz | Verilog-HDL | Yes |
| Tse et al. [40] | 2010 | MC | XC5VLX330T (65nm) | 200Mhz | VHDL | No |
| de Schryver et al. [41] | 2011 | MC | XC5VFX70T (65nm) | 100Mhz | Visual-HDL | No |
| Jin, Luk, Thomas [42] | 2011 | | framework | | | |
| Jin, Luk, Thomas [43] | 2011 | E-FD | XC6VLX550T (40nm) | 310Mhz | HDL | No |
| Becker et al. [44],[45] | 2011 | E-FD | XC6VLX760 (40nm) | - | VHDL | Yes |
| Chatziparaskevas et al. [27] | 2012 | E-FD,CN-FD | XC5VSX240T (65nm) | 145Mhz | VHDL | No |
| Sridharan et al. [46] | 2012 | MC | StratixIVE530 (40nm) | 125Mhz | - | No |
| Chow et al. [47] | 2012 | MC | XC6VSX475T (40nm) | 175Mhz | - | Yes |
| de Schryver et al. [48] | 2013 | ML-MC | XC6VLX240T (40nm) | 120Mhz | Vivado HLS | No |
| Sanchez-Roman et al. [49] | 2013 | MC | StratixV5SGSD8 (28nm) | 142Mhz | Impulse C (HLS) | No |
| Inggs et al. [50] | 2013 | | framework | | | |
| Morales et al. [51] | 2014 | BT | EP4SGX530 (40nm) | 162MhZ | OpenCL | Yes |
| Palmer [52] | 2014 | I-FD | Zynq7020 (28nm) | 100Mhz | VHDL | Yes |
| Brugger et al. [53] | 2014 | ML-MC | Zynq7020 (28nm) | 100Mhz | Vivado HLS | No |
| Inggs et al. [54] | 2014 | MC | Zynq7Z045 (28nm)<br>StratixVGXA7(28nm)<br>XC6VSX475T (40nm) | 100Mhz<br>250Mhz<br>200Mhz | Vivado HLS<br>OpenCL<br>MaxCompiler | No |
| Tavakkoli, Thomas [55] | 2014 | Systolic BT | XC7VX980T (28nm) | 150Mhz | VHDL | No |
| Laszlo et al. [56] | 2015 | E-FD,I-FD | XC7VX690T (28nm) | 234Mhz | Vivado HLS | No |
| Varela et al. [57] | 2015 | MC | ZynqZC702 (28nm) | 100Mhz | Vivado HLS | No |
| Ma, Muslim, Lavagno [58] | 2016 | MC | XC5VLX330T (65nm)<br>XC7VX690T (28nm) | 125Mhz<br>165Mhz | Vivado HLS<br>Vivado HLS | No |
| Pham et al. [59] | 2016 | MC | XC6VSX475T (40nm) | 140Mhz | MaxCompiler | No |
| Stamoulias et al. [60] | 2017 | BS<br><br>B76<br><br>BT | XCKU060 (20nm) | 262Mhz<br>277Mhz<br>237Mhz<br>273Mhz<br>253Mhz<br>225Mhz | VHDL<br>Vivado HLS<br>VHDL<br>Vivado HLS<br>VHDL<br>Vivado HLS | Yes |
| Fabry, Thomas [61] | 2017 | B-MC | XCVU065 (20nm) | 300Mhz | Vivado HLS | Yes |
| Tavakkoli, Thomas [62] | 2017 | Systolic BT | XC7VX980T (28nm) | 100Mhz | HLS | Yes |
| Muslim et al. [63] | 2017 | MC | XC7VX690T (28nm) | 200Mhz | OpenCL | No |

In Figure 12 a statistical analysis of the literature is presented, where some basic observations can be made:

- a) As it can be seen in this graph with the papers per year, from 2005 to 2008 there is only one work per year, only targeting Monte Carlo methods. Afterwards, from 2009 until 2017 there is an average of 3.1 papers per year, with a peak in 2015, where five papers were published. From this can be derived the conclusion that the research topic of FPGA based option valuators has a steady demand after the economic crisis of 2008.

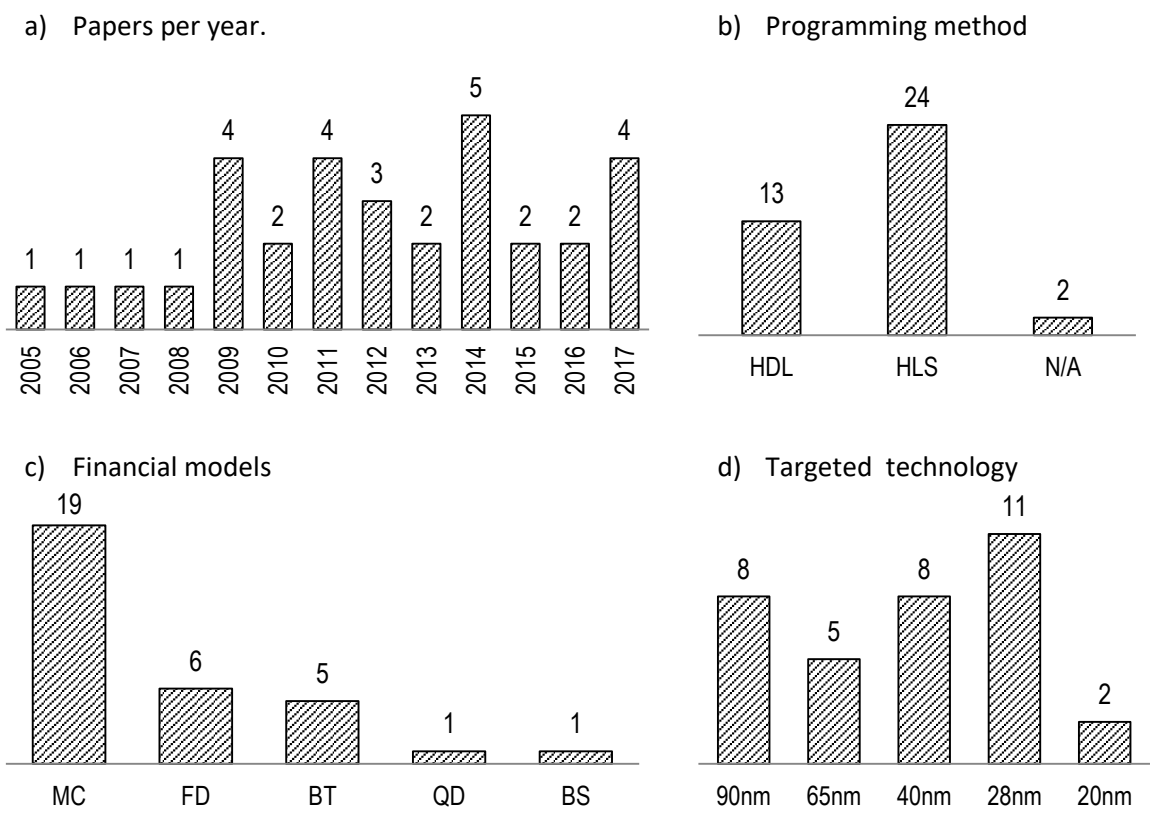
- b) In the second graph of the figure 12 is the frequency of the programming methods that have been used in the literature. Here, it can be observed that High-Level-Synthesis (HLS) is used about double times than Hardware-Description-Language (HDL). This is a trend that applies to every FPGA based research independently of the specific topic or field that is being tackled. Specially, since 2013 when Xilinx launched the VivadoHLS tool most of the researches were using this tool (see **Table 7**).

- c) The third graph shows the financial models that have been implemented on FPGAs. It is clear that Monte Carlo is by far the most implemented method in the literature. The main reason for this is that the Monte Carlo method it is not just an algorithm but it can be characterized more as a procedure. Thus, it gives more flexibility to apply different algorithms in every step of the process. Also, the MC can be used for pricing any kind of derivatives. Nevertheless, it must be mentioned , as Jin et al. [42] suggest, that the FPGA based Monte Carlo solver should only be used when there are no other solvers available. The other models found in the literature are Financial Differences and Binomial Trees. Also, there are one work of Quadrature model and one with Black Scholes analytical solution.

- d) The last graph of the same figure shows the technology of FPGAs. Here, the use of new technology follows the timeline. Since 2005 the structure width of the FPGAs changed from 90nm to 20nm.

Afterwards, the statistical analysis it is necessary to explore further the literature. The way this has been done is by classifying related works based on the methodology. Thus, there are three major categories the Monte Carlo methods, the Finite Differences and the binomial trees, which are going to be discussed further.

**Figure 12: Statistical analysis of literature of FPGA option pricing.**




### 4.2.1 Monte Carlo based works

The first work this study covers was Zhang et al. [28]. They proposed a generic architecture for accelerating MC simulation using an on-chip processor, and a hardware path generator. Also, an illustration of using a generalized number system optimization package named Computer Arithmetic Synthesis Tool (CAST) was used to provide evaluation of the minimum resources required to produce at least 4 decimal place accuracy (as required in financial applications). They examined fixed-point and floating-point number representation. Additionally, their work contribute specialization of the proposed generic architecture to support financial computations based on the BGM interest rate model, with methods for generating Gaussian distributed random numbers, for supporting fast division, and for pipelining the MC simulation. The evaluation of their MC processor showed that an implementation involving a Xilinx XC2VP30 device at 50 MHz is over 25 times faster than a Pentium processor at 1.5 GHz.

In the same concept of Monte Carlo simulation Thomas et al. [29] exploited the nature of MC model and the ability of FPGA for extensive parallelization, by modularizing the individual simulation procedures and replicating them as many times needed to achieve performance. Application of their proposed methodology applied to five different Monte- Carlo simulations

such as random walk, random jump, log-normal walk, dual-asset value at risk, and the GARCH model. Their results showed that hardware implementations on a Xilinx Virtex-4 XC4VSX55 device can run on-average over 80 times faster than software on a 2.66GHz Xeon PC.

Also, Morris and Aubury [30] compared the performance, in terms of acceleration and accuracy, of various fixed and floating point FPGA implementations of the European option benchmark. These were compared to implementations using other devises such as GPUs, Cell BE and CPU. Furthermore, the paper introduced 'HyperStreams' a high-level abstraction, built on the Handel-C programming language.

Baxter et al. [64] presented a generic supercomputer built of 64 FPGAs called Maxwell. One of the case studies, among others, was the implementation in this supercomputer of the Monte-Carlo simulation. The Monte-Carlo kernel was repeated 10 times in a FPGA device and edited by 16 FPGAs. The results showed that FPGAs can overcome more than two orders of magnitude CPU. In the same supercomputer Maxwell Tian et al. [31] implemented the Monte-Carlo simulation using the Box-Muller model to create random paths and GARCH model to compute stochastic volatility. They reported a speedup of 340x by the FPGA implementation versus an equivalent software implementation running on the 2.8 GHz Xeon processors. In another work of Tian and Benkrid [39] a new model for generating random number is tested. Their method used quasi-random or low-discrepancy numbers as random sample sets. Real hardware implementations on the Maxwell machine showed that FPGAs outperform equivalent GPP-based software implementations by 2 orders of magnitude, with the speed-up scaling linearly with the number of processing nodes used (FPGAs/GPPs). The same implementations showed that FPGAs achieve a 3x speedup compared to equivalent GPU-based implementations.

In the work of Tse et al. [32], [33], [34] the Quadrature numeric integration method was applied. This method can be used for derivatives with more than one underlying product. An automated system for creating architectural material was presented to examine the increase in system dimensions. The results show acceleration 23 times in FPGA in relation to the CPU. The GPU performance is roughly the same as FPGA.

The aforementioned works were a first attempt to build option pricing systems on FPGAs based on different hardware architectures and Monte Carlo algorithms. The next stage on the literature was to value more complex options like Asian, Barrier and other exotic derivatives. Also, in this next phase of the field works are characterized by the use of HLS tools.

The work of Tse et al. [40] proposed an FPGA-accelerated Asian option pricing solution, using a highly-optimized parallel Monte-Carlo architecture. The result of this work reported that an implementation of their architecture in a Virtex-5 xc5vlx330t FPGA at 200MHz was 313 times faster than a multi-threaded software implementation running on an Intel Xeon E5420 quad-core CPU at 2.5GHz; it was also 2.2 times faster than the Tesla C1060.

Schryver et al. [41] used an HLS tool called VisualHDL to implement a Heston model simulating single asset barrier options. A Virtex V family FPGA outperformed a 2 GHz Core 2 Duo processor by a factor of 21. Extending the work of Schryver et al. to multi-asset barrier options Sridharan et al. [46] in their design utilized the full truncation Euler discretization method.

They used payoff calculator kernels to compute various payoffs such as vanilla portfolios, barriers, look-backs, etc. Also, they had an early termination condition of "out" barrier options to efficiently schedule MC paths across multiple cores in a single FPGA and across multiple FPGAs. The target platform was Novo-G, a reconfigurable supercomputer housed at the NSF Center for High-Performance Reconfigurable Computing (CHREC), University of Florida. Their design was validated and achieved an average speedup 189 on a Stratix IV FPGA. In the same concept Schryver et al. [48] presented a multi-level Monte Carlo architecture for FPGAs. The multi-level method iteratively runs Monte Carlo simulations by adjusting the simulation numbers and precisions, and has shown reduce of the computational complexity compared to single-level methods by up to 50%. For a Xilinx Virtex-6 FPGA they report that can simulate up to 100 million of time steps in an asset path simulation with less than 3.6 W. Continuing the research of Multi-Level Monte Carlo simulations on FPGA boards Brugger et al. [53] presented energy-efficient modular option pricing framework called HyPER that is generically applicable to all kinds of hybrid CPU/FPGA platforms. They reported that the implementation for barrier options on the Xilinx Zynq 7020 with HyPER platform was 3.4x faster and 36x more power- efficient than a highly tuned software reference on an Intel Core i5 CPU.

Varela et al. [57] presented a way to price multi-dimensional American options (highly involved in risk management) targeting heterogeneous CPU/FPGA systems. They demonstrated how architectural limitation of the Longstaff-Schwartz algorithm was solved by means of an algorithmic transformation employing the Brownian Bridge technique. They concluded that the system on FPGAs achieved a 2x improvement in runtime compared to the state-of-the-art solution and 1.8x more energy efficient than the same reference.

To achieve even better performance researcher started to exploit and other advantages of FPGAs, such as custom precision. An example is the work of Chow et al. [47] which introduced a mixed precision methodology applicable to any Monte Carlo (MC) simulation. It involved the use of data-paths with reduced precision, and the resulting errors were corrected by auxiliary sampling. An analytical model was developed for a reconfigurable accelerator system with a field-programmable gate array (FPGA) and a general purpose processor (GPP). Optimization based on mixed integer geometric programming was employed for determining the optimal reduced precision and optimal resource allocation among the MC data-paths and correction data-paths. Experiments showed that the mixed precision methodology requires up to 11 % additional evaluations while less than 4 % of all the evaluations are computed in the reference precision; the resulting designs were up to 7.1 times faster and 3.1 times more energy efficient than baseline double precision FPGA designs, and up to 163 times faster and 170 times more energy efficient than quad-core software designs optimized with the Intel compiler and Math Kernel Library. Their methodology also produced designs for pricing Asian options which were 4.6 times faster and 5.5 times more energy efficient than NVIDIA Tesla C2070 GPU implementations.

The next phase of Monte Carlo FPGA based option pricing was the use of HLS tools and Heterogeneous Computing. This was necessary so the aggregated knowledge of all this year of research to become accessible to non-FPGA users and be business ready as automated pricing systems. The follow works are examples of that trend.

Sanchez-Roman et al. [49] proposed an FPGA implementation of a Monte-Carlo method for pricing Asian Options using Impulse C and floating-point arithmetic. In an Altera Stratix-V FPGA, a 149x speedup factor was obtained against an OpenMP-based solution in a 4-core Intel Core i7 processor.

Inggs et al. [50] presented the Forward Financial Framework ($F^3$), an application framework for describing and implementing forward looking financial computations on high performance, heterogeneous platforms. $F^3$ allows the computational finance problem specification to be captured precisely yet succinctly, and then automatically creates efficient implementations for heterogeneous platforms, utilizing both multi-core CPUs and FPGAs. The automatic mapping of a high-level problem description to a low-level heterogeneous implementation is possible due to the domain-specific knowledge which is built in $F^3$, along with a software architecture that allows for additional domain knowledge and rules to be added to the framework. The processor used was an Intel Core i7 870 running at 2.93 GHz with an 8192 KB Cache and the FPGA platform was a Maxeler Max3 Card, in particular the Max3424A which makes use of a Xilinx Virtex 6 XC6VSX475T. Their results report that $F^3$ achieves comparable speed and energy efficiency to external manual implementations. Further, the domain-knowledge guided partitioning scheme suggests a partitioning of subtasks that is 13% faster than the average, while exploiting domain dependencies to reduce redundant computations results in an average gain in efficiency of 27%.

In an another work of Inggs et al. [54] they tested their $F^3$ framework using HLS tools such as Xilinx's Vivado HLS, the Altera OpenCL SDK and Maxeler's MaxCompiler. They concluded that the tools offered by Maxeler and Altera are well-suited for accelerating parallel-friendly algorithms such as the Monte Carlo pricing algorithm, as parallelism can be made explicit fairly easily. Xilinx's offering however was better suited to small, functional unit prototyping and will not provide optimal results if used by developers with insufficient hardware design experience. When they applied direct source code translation did not meet performance expectations, when a combination of techniques applied such as exploiting task or pipeline parallelism as well as C-slowing, an acceleration of up to 220 times was achieved using these tools. Compared to the 31 times improvement showed by an optimized Multicore CPU implementation, the 60 times improvement by a GPU and 207 times by a Xeon Phi.

In the same content of exploring High Level Synthesis Ma, Muslim and Lavagno [58] implemented the Black-Scholes and Heston models using OpenCL/C++, to allow direct comparison between GPU and FPGA implementations. The comparison of performance and energy consumption between GPU and FPGA was done with previous results from the literature. In particular, they showed that energy per computation by using an FPGA can be from 5.9% to 9.8% (depending on the algorithm) as much as that by using an GPU as an accelerator for financial models, while performance can be from 1.71X to 2.56X as fast as the GPUs. They also concluded that the Virtex-7 FPGA had a better overall performance than the advanced GPUs in option pricing problems, which is computation-bounded, rather than memory-bounded. On the energy efficiency aspect, the FPGA is 10X more frugal than the GPUs. The implementations in this work were also better than those in previous works in the domain of FPGA acceleration of financial models. For the Black-Scholes model of the Asian option problem, 2.4X of the performance and 41.7% of energy consumption were obtained compared to a previous manual RTL design. For the Heston model of the European barrier

option, this paper had achieved 2.56X of the performance and 78.1% of energy consumption of a previous implementation designed via HLS. Another use of HLS with OpenCL can be found in Muslim et al. [63].

Another attempt to framework the automatic build of reconfigurable accelerators for option pricing was the work of Pham et al. [59]. Their framework was based on tools such as XML format for the option pricing request, the design flow and optimization framework were developed using Python and Maxeler IDE. The generic Pricing Engines were developed by Maxj data flow language, the Java High Level Synthesis language from Maxeler, and took advanced of C-slow optimization techniques. The experiment results were obtained by implementing and running the engines on Maxeler Workstation model MAX3424A, which features with a Xilinx Virtex-6 SX475T FPGA device and Intel Core i7 870 2.93 GHz with 16GB RAM. As a result, they reported a speedup of 2 orders of magnitude compared to SW implementations.

Before the literature review goes to the other models, featuring in the analysis, the work of Fabry and Thomas it must be mentioned [61]. They introduced a hybrid model that combines Binomial with Monte Carlo simulation. They used discrete-space random walk over a binomial lattice, rather than the continuous space-walks used by existing approaches. They used limited-precision fixed-point arithmetic for the generation of paths to achieve vast parallelization, it was found that the size of a discrete-space MC engine can be kept to 370 Flip-Flops and 233 Lookup Tables, allowing up to 3,000 variance-reduced MC cores in one FPGA. The combination of a highly parallelizable architecture and model-specific optimizations means that the binomial pricing technique allows for a 50x improvement in throughput compared to existing FPGA approaches, without any reduction in accuracy.

### 4.2.2 Trees based works

Tree based methods are used most commonly for pricing American options that are difficult to price using Monte-Carlo methods. The first, in our knowledge, that implemented the binomial and trinomial tree method for pricing options on FPGAs were Jin et al. [35],[36]. They used Handel-C source code which was synthesized to EDIF using the Celoxica DK5 suite, which supports HyperStreams. Xilinx ISE 9.2i project navigator was used to place and route the design. Their results showed that the tree-based models executing on a Virtex 4 FPGA at 82.7 MHz with fixed-point arithmetic could run over 160 times faster than a Core2 Duo processor at 2.2 GHz. The FPGA implementation was two times faster than the nVidia Geforce 7900GTX processor with 24 pipelines at 650 MHz, and 27%–35% slower than the nVidia Geforce 8600GTS processor with 32 Pipelines at 1450 MHz.

Wynnyk and Magdon-Ismail [38] presented a vector FPGA design which prices American options by using 4-way parallelism on a single-asset evaluation. They used memory management techniques to solve single trees of up to 64000 time steps in double precision; these techniques achieved a 73x speedup over an optimized CPU implementation as they reported. Their design was coded in Verilog. The work in [51] by Morales et al. is another vector solver (NV=8) of American options with medium-sized binomial trees (n=1024) using GPUs and energy-efficient FPGA implementations. They use the OpenCL language and Altera's

OpenCL compiler to synthesize their design. The implementation could evaluate more than 2000 options/s with an average power less than 20W.

Tavakkoli and Thomas in [55] provided an American option pricer based on a systolic reconfigurable architecture with n systolic cells. The work presented fixed-point FPGA solvers with hand-coded RTL blocks and FloPoCo [65] floating-point cores. Their design achieved, on a Xilinx Virtex-7 xc7vx980t FPGA for a single option with 768 time steps, to be priced with a latency of less than 22 micro-seconds and a pricing rate of more than 100 K options/sec. Although their design gained considerable accelerations, both in terms of latency and throughput, compared to scalar and vector approaches, relocation of the hand-coded RTL solution to other data precisions limited the productivity of this design. So they came up with a new work [62] that presented a formal mathematical framework that captures a large class of binomial-tree problems, and provided a systolic data-movement template that maps the framework into digital hardware. Their solution is based on a fully-automated design flow, which takes C-level user descriptions of binomial trees, with custom data types and tree operations, and automatically generates fully-pipelined reconfigurable hardware solutions in FPGA bit-stream files. Their results indicate that on a Xilinx Virtex-7 xc7vx980t FPGA at a 100-MHz clock frequency can be required 54-μs latency to solve three 876-step 32-bit fixed-point American option binomial trees; with a pricing rate of 114k trees/s. Except the updated results they also provide a brief comparison with previous works in this paper.

Finally, in the work of Stamoulias et al. [60] comparison between VHDL and HLS implementations is made, with three different financial models (Black & Scholes, Black-76 and Binomial). The latter financial model is Binomial trees. They concluded that HLS can achieve higher accuracy due to the floating point, but requires up to 20% higher number of resources in terms of DSPs while the fixed-point implementations developed in HDL can save significant space in terms of resources but with limited accuracy compared to the software code.

### 4.2.3 Finite Differences based works

As it was presented in Chapter 3: there are three schemes of finite differences: explicit, implicit and Crank-Nicholson. The latter can be interpreted as the combination of explicit and implicit scheme. The big difference between explicit and implicit is the computational effort needed for these two approaches. Explicit can solve straightforward the Black Scholes PDE by going forward in the dimension of time; in contrast implicit involves solving a system of linear equations. Thus, the literature is divided in two subcategories: Explicit Finite Differences (E-FD) and Implicit or Crank-Nicolson Finite Differences (I-FD or CN-FD).

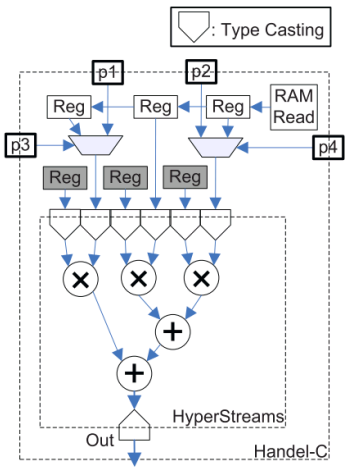
#### 4.2.3.1 *Explicit Finite Differences (E-FD)*

The first work that examined the E-FD scheme on FPGAs, in pricing option concept, was the work of Jin, Thomas and Luk [37]. They defined two levels of parallelism for the E-FD model:

- Coarse granularity to be the ability to valuate different options at the same time.
- Fine granularity to be the ability to valuate different nodes in a grid at the same time.

The higher the coarse granularity, the more options can be priced at the same time; the higher the fine granularity, the faster the valuation speed per option. Their FPGA implementation of the Fine Core logic for the E-FD option pricing model was based on HyperStreams and the Handel-C programming language.

**Figure 13 : The data flow of the E-FD hardware implementation on FPGA (Source: Jin et al. [37])**




In results they mentioned that the peak performance occurs when aggregation of all Fine Cores is made into one Coarse Core, as no logic is wasted to glue the Coarse Cores. The implementation on a Xilinx xc4vlx160 FPGA at 106MHz for the 32-bit single-precision offers a 1.5 times acceleration over the software on an Intel Pentium 4 at 3.6 GHz, while the 64-bit double precision version offers 1.2 times speedup. Eight single-precision Fine Cores can be replicated on the xc4vlx160, and was estimated to achieve 12.2 times acceleration. The authors did not report absolute time measurements for their implementation, neither accuracy analysis.

Continuing their research Jin et al. [43] extended the previous work by presenting a unifying framework for describing and automatically implementing financial explicit finite difference procedures in reconfigurable hardware. The implementation of the framework was based on a Virtex-6 XC6VLX550T device and Flopoco [65] floating point library was used to generate the floating point pipelines. Results showed that an implementation targeting a Virtex-6 device at 310MHz is more than 24 times faster than a software implementation fully optimized by the Intel compiler on a four-core Xeon CPU at 2.66GHz.

To achieve better performance for the E-FD scheme, the same team of Becker, Jin, Thomas and Luk proposed in [44] , the reconfiguration of slowly changing constants in an explicit finite difference solver for option pricing. The approach for this was that a specialized circuit can be design with use of fixed-coefficient multipliers, because the coefficients α, β and γ are constant throughout the pricing of one option. Their design was implemented on the Xilinx Virtex-6 XC6VLX760 FPGA using the FloPoCo [65] to generate fixed-point arithmetic cores,

which then applied with the dynamic constant reconfiguration approach. The fixed-point error analysis was based on the MPFR library [66] and they provided in the paper [45]. Their results showed that by applying the dynamic designs on Coarse grain parallelism can significantly reduce execution times. In the case of 16 Coarse Cores, the execution time for pricing one option was reduced from 26.8 ms (static design) to 5.7 ms (reconfigurable design) which represents a speed-up by a factor of 4.7.

Finally, Jin, Luk and Thomas [42] proposed a framework that could compare different financial on different FPGA hardware implementations. They provide analytical functions to calculate performance metrics such as:

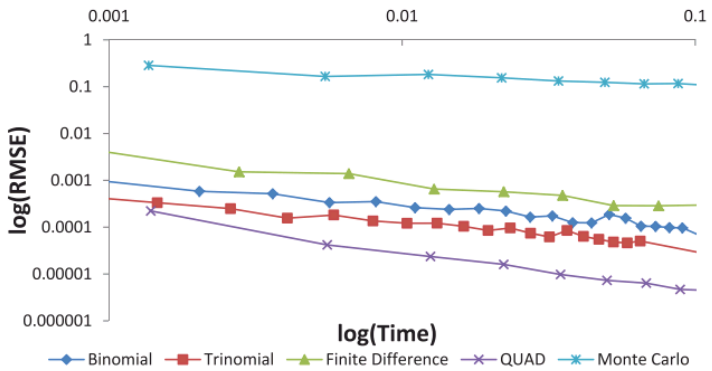
- Execution time (e.g. in seconds), which is based on the assumption that FPGA implementations are fully pipelined and they link the clock rate to execution time with a metric called Asset Price Observation (APO) points.

- Accuracy (e.g. relative error to the reference), they measuring it by utilizing a Monte Carlo simulation technique with randomly generated parameters for the target solver and measuring the root mean squared error over this random sample.

- Resource consumption (e.g. in LUTs and DSPs), by approximating with a methodology that links resource consumption of all the arithmetical operators and aggregates them based on the complexity of the mathematical model.

The two first bullets can be combined (Figure 14) to provide characteristics curves of the different FPGA solvers.

**Figure 14 : Root mean squared error against time for European option (Source: Jin et al. [42])**



In the results of the implemented framework, they conclude that:

- The quadrature method implemented on an XC4VLX160 device produces the most accurate result in terms of RMSE the fastest for European options.
- The trinomial tree tends to be more accurate than binomial tree on a XC4VLX160 device for European options, but they tend to converge to the same result when number of APO points gets larger.
- The explicit finite difference solver provides a more accurate result than the tree based method when number of APO points reaches 10K.

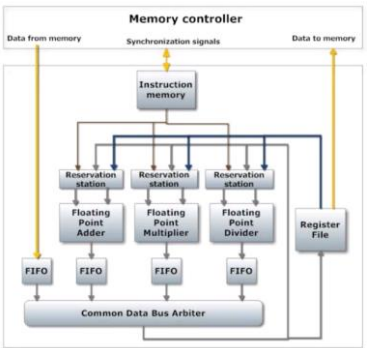
In practice the quadrature solver should be used if applicable. Otherwise the tree based solvers should be used if the result is time critical and the explicit finite difference method should be used if the result is accuracy critical.

#### 4.2.3.2 *Implicit or Crank-Nicolson Finite Differences (I-FD or CN-FD)*

The previous subsection covered the literature of E-FD schemes for option pricing in FPGAs. This section will cover the few works that implemented I-FD or CN-FD schemes. These schemes need a solution of tridiagonal linear systems in every time step, thus a categorization can be based on the algorithm used to solve the system. But, only papers in the field of option pricing are going to be mentioned here, general tridiagonal solvers on hardware are discussed in next section.

The first paper, in our knowledge, that implemented the CN-FD on FPGA was the work of Chatziparaskevas et al. [27]. They used as tridiagonal solver a variant of Cyclic Reduction with normalization of main diagonal (Norm-CR). The hardware architecture was based on a FPGA-based parallel processor, which included a pipelined FPU consisting of a single adder, divider and multiplier. This FPU could compute in serial execution a number of loops of the forward and backward phase. The parallel execution was based on batches of loops in separate FPUs. The architecture for the FPU is showed in the next figure.

**Figure 15 : Internal architecture of Floating Point Unit (Source: Chatziparaskevas et al. [27])**

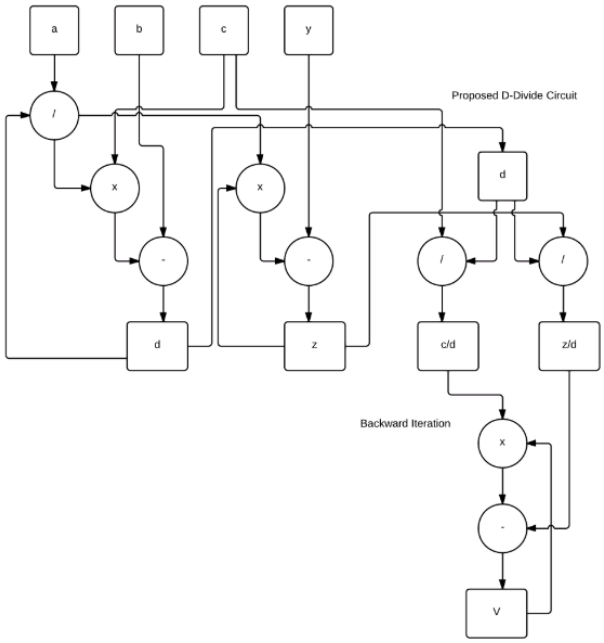


In their results there is no reference for accuracy and absolute timings. The design had been implemented on a Xilinx Virtex5 FPGA XC5VSX240T device, with clock frequency of 145 MHz. FP multipliers, adders and dividers had selected latencies of 4, 8 and 14 clock cycles respectively. They reported that 64 cores could fit in the device and achieved speedup 3.7x over LU CPU implementation.

The aforementioned paper is the base of our research, as the same algorithm (Norm-CR) is used for solving the tridiagonal systems produced by the CR-FD scheme for option pricing with Black-Scholes PDE. Details for the hardware architecture are unfolded in the next Chapters.

Another work, Palmer and Thomas [26], tried to provide a design and implementation of the TDMA or Thomas algorithm optimized for hardware acceleration on FPGA. The hardware based algorithm optimized to achieve overall complexity from 8N down to 5N arithmetic operations, and memory overheads to only 2 N-length vectors per N-tridiagonal system to be solved (see Figure 16). The parallelization of the procedure aroused from coarse replication of the Thomas Core that can solve multiple tridiagonal systems. They also provide a theoretical analysis for fixed-point arithmetic used in the design and how affects the error propagation. No results were provided for the implementation as they reference a problem with the vendor hardware compiler.

**Figure 16 : Data dependency graph for the Thomas algorithm (Source: Palmer and Thomas [26])**




The pending results were presented by Palmer in [52]. The hardware, that Thomas algorithm was implemented, was the ZedBoard Xilinx Zynq7020 Evaluation Kit. Four different settings

were compared, one floating-point design with maximum clock frequency of 100MHz and three different fixed-point at 200MHz. The average time (ms) for computing the solution to tridiagonal systems (N=100) of FPGA Thomas solver for I-FD scheme can be seen in the following Figure 17.

**Figure 17 : Timing of FPGA Thomas solver on Zynq7020 (Source: Palmer [52])**

| | Max Throughput | Min Throughput |
|---|---|---|
| CPU(2.6GHz) | 0.020ms(1x) | 0.020ms(1x) |
| Floating | 0.0012ms(16x) | 0.063ms(0.31x) |
| Fixed[2,30] | 0.00055ms(36x) | 0.040ms(0.50x) |
| Fixed[2,22] | 0.00055ms(36x) | 0.036ms(0.55x) |
| Fixed[2,14] | 0.00057ms(35x) | 0.028ms(0.72x) |

Finally, the third paper which present an FPGA solver for the Black-Scholes PDE, using I-FD scheme is the work of Laszlo et al. [56]. They examined further the architectural, programmability and development issues regarding CPU, GPU and FPGA architectures. They proposed a parallel processor for implicit solver utilizing the Thomas algorithm (TDMA). The parallelization of the method was achieved by coarse grain parallelism, that is, independent processors are performing the calculation of independent options. Some of the implementations details the authors had reference are:

- Each of these processors is capable of pipelining the computation of more options into the same processor with the associated cost of storing the temporary ($c*$, $d*$) arrays of each option.

- The number of options that can be pipelined is defined by the depth of the forward sweep of the Thomas algorithm, which was 67 clock cycles, without referencing if it was on double or single floating point format.

- The temporary storage was implemented in the available Block RAM memories, but due to the deep pipeline the BRAM memory requirement limits the number of deployable processors.

The targeted FPGA device was Xilinx Virtex 7 VX690T. The replication of the processor cores was 11 and 5, for single and double precision respectively. The authors had reported results in terms of resources and performance for both formats of floating point; single and double respectively (see Figure 18).

They also reference a problem with the HLS compiler that failed to recognize that no data hazard exist between $c^*_i$ and $c^*_{i-1}$ in the two consecutive iterations of the forward pass in the tridiagonal algorithm and therefore refused to properly pipeline the loop. They patched the problem by using a secondary temporary array to store a copy of the temporary array.

**Figure 18 : Resource and Performance of FPGA Thomas solver (Source: Laszlo et al. [56])**

TABLE I. FPGA RESOURCE STATISTICS FOR A SINGLE PROCESSOR

| | # BRAM | | # DSP slice | | Clock [ns] | | $\times 10^6$ Ticks | |
|---|---|---|---|---|---|---|---|---|
| | SP | DP | SP | DP | SP | DP | SP | DP |
| Explicit | 24 | 64 | 1200 | 3570 | 4.09 | 4.01 | 334 | 315 |
| Implicit | 256 | 512 | 37 | 94 | 4.26 | 4.3 | 14.2 | 12.6 |

Note: SP - Single Precision, DP - Double Precision

TABLE II. PERFORMANCE - SINGLE PRECISION

| | ps/element | | | GFLOPS | | | GFLOPS/W | | |
|---|---|---|---|---|---|---|---|---|---|
| | C | G | F | C | G | F | C | G | F |
| Explicit | 15.2 | 2 | 17.4 | 394 | 3029 | 344 | 1.46 | 12.9 | 8.6 |
| Implicit | 142.7 | 14.5 | 766 | 139 | 1849 | 26 | 0.51 | 7.9 | 0.65 |

Note: C - 2 Xeon CPUs, G - Tesla K40 GPU, F - Virtex 7 FPGA

TABLE III. PERFORMANCE - DOUBLE PRECISION

| | ps/element | | | GFLOPS | | | GFLOPS/W | | |
|---|---|---|---|---|---|---|---|---|---|
| | C | G | F | C | G | F | C | G | F |
| Explicit | 29.8 | 4.1 | 48.2 | 201 | 1463 | 124 | 0.74 | 6.2 | 3.1 |
| Implicit | 358.8 | 43.5 | 1748 | 48 | 892 | 9.8 | 0.18 | 3.8 | 0.24 |

Note: C - 2 Xeon CPUs, G - Tesla K40 GPU, F - Virtex 7 FPGA

In conclusion, the aforementioned four papers provide insights about architectures on FPGAs, for solving the Black-Scholes PDE with implicit finite difference scheme, by using a tridiagonal solver as core of their designs.

Nevertheless, the volume of the literature is too low to make safe assumptions about the best alternative to address the problem. Also, the results of the works are not clear enough, as there is no specific benchmark to evaluate each on with the others, because either timing is not in absolute numbers (ms) and is expressed with speedup terms; either the error analysis is theoretical or is not presented at all.

This subsection can be characterized as related work to this research. However, the disadvantages mentioned in the previous paragraph and the fact that there wasn't found an efficient FPGA implementation of Cyclic Reduction for option pricing, were a motivation to extend further the literature review to thoroughly cover the tridiagonal solver accelerators; as it is the part of the procedure with the biggest complexity. Thus, in the next section some interest works of the literature for FPGA and GPU based tridiagonal solvers are presented.

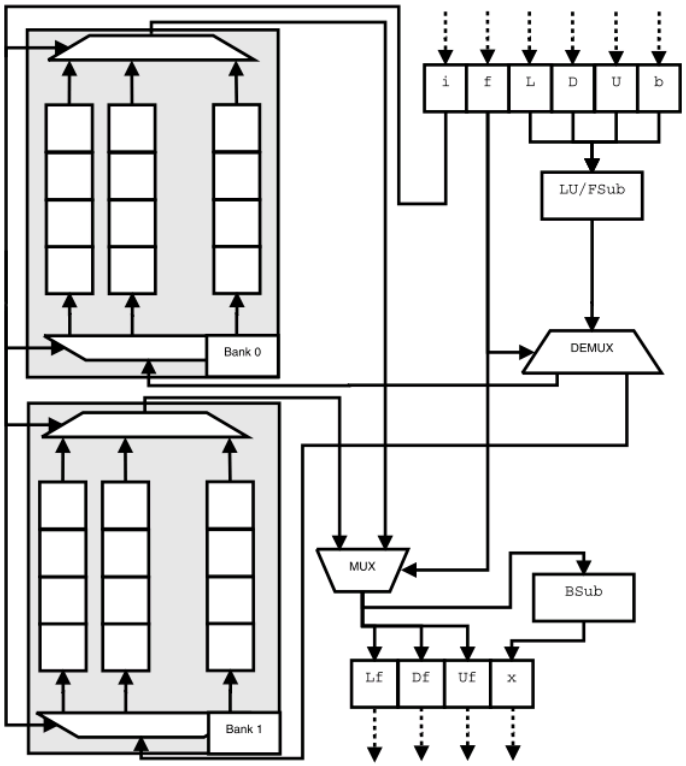
## 4.3 FPGA and GPU based tridiagonal solvers

This section is dedicated to tridiagonal solvers implemented on FPGAs or GPUs. The search of relative literature has been done with Cyclic Reduction being in the center of this research, as it is the algorithm that is used in this thesis.

Here the classification of the literature is following a two level hierarchy. The first is the targeted hardware device, i.e. FPGA or GPU, and the second level is characterized by the algorithms that are presented in the papers, i.e. TDMA or Thomas and Cyclic Reduction.

The FPGAs implementations are few in numbers and only the TDMA solver is consider in the literature. In the paper [67] of Oliveira et al. a custom processor is proposed for the TDMA. This solver was intended for use in Computational Fluid Dynamics (CFD) application, thus is specific to that particular CFD. Warne et al. in [68] proposed a more general architecture for the TDMA on FPGAs. They designed a custom circuit that is well suited for applications that require solving many independent tridiagonal systems (see Figure 19). The design supports user specified precision thought the use of a custom floating point VHDL.

**Figure 19 : TDMA solver pipeline (Source: Warne et al. [68])**



In [69] they provide results for the same design implemented on Xilinx and Altera devices with Vivado HLS and OpenCL respectively. For a tridiagonal system of size 1024 the results in terms of accuracy and speed are presented in the next figure.

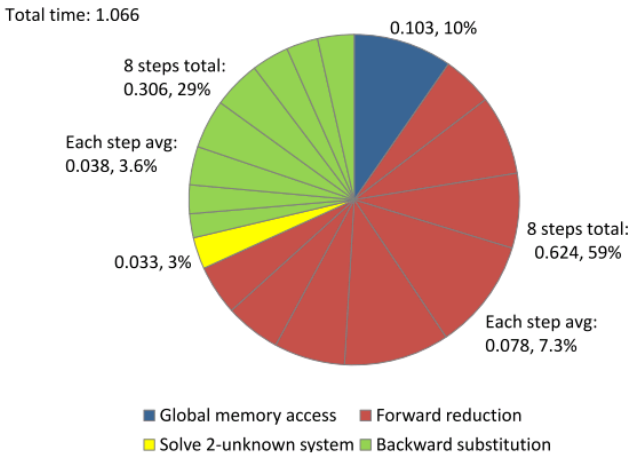
**Figure 20 : Results of TDMA solver pipeline (Source: Warne et al. [69])**

Table 2: Run-time Comparison and Solution Error

| | VHDL | Altera OCL[1] | Xilinx HLS |
|---|---|---|---|
| Run-time (secs) | 0.000671 | 0.002752 | 0.000465 |
| $\|\|\mathbf{x}-\mathbf{b}\|\|_\infty$ | 1.79e-04 | 6.60e-05 | 4.40e-05 |

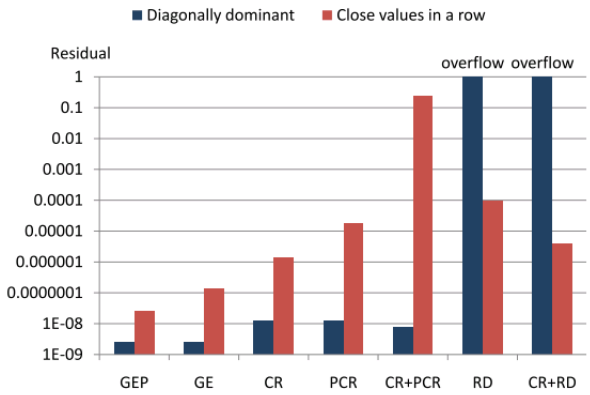
Unlike the FPGAs based works, GPUs have plenty implementations of tridiagonal solvers as it is common in many graphic problems to solve multiple times tridiagonal systems [70]. In paper [71] Zhang et al. provided a detailed comparison of the performance of three parallel algorithms and their hybrid variants for solving tridiagonal linear systems on a GPU: cyclic reduction (CR), parallel cyclic reduction (PCR) and recursive doubling (RD). The results for the CR are depicted in figure below.

**Figure 21 : Timings in milliseconds of CR for problem size 512x512 (Source: Zhang et al [71])**




Also they measured the residual error for the algorithms in single floating precision (see Figure 22). The CR is stable both for diagonally dominant matrices and matrices with close values in a row.

**Figure 22 : Accuracy Analysis of tridiagonal solvers (Source: Zhang et al [71])**




In contrast, Göddeke and Strzodka in [72] proved that attempting to solve the system in single precision fails, while double precision suffices to reduce the error of the finite scheme; and hence, to guarantee the result accuracy. In single precision, however, further refinement of the increases the error again: The additional refinement results in a solution that is objectively worse, although more unknowns are involved and hence more work is performed. Adding to the detriment, this behavior is difficult to notice without ground truth, because the solver still converges and reduces the residuals as expected. They implement the CR with optimizations in memory and they declare that execution time is 0.445 ms for 513 systems of 513-unknowns on the same GPU (NVIDIA GTX 280).

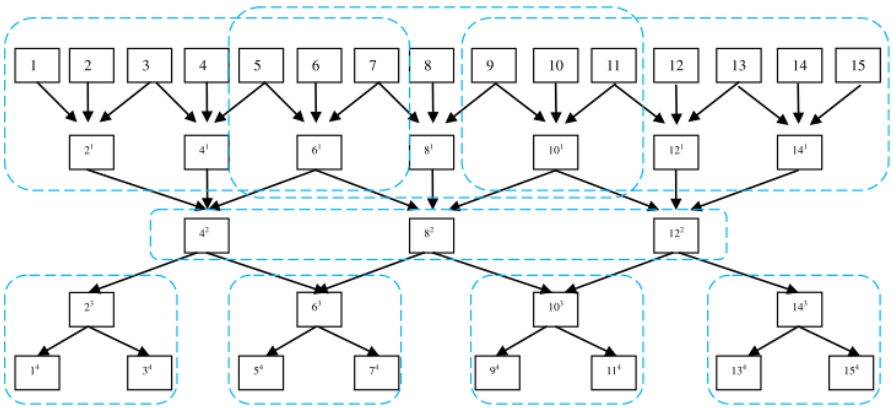
Another work that implement CR on GPU (NVIDIA GeForce GTX 295) is [73] of Quesada-Barriuso et al. They analyzed the cyclic reduction method, as example of fine-grained parallelism, and Bondeli's algorithm, as a coarse-grained example of parallelism. Both algorithms were implemented for GPU architectures using CUDA and multi-core CPU with shared memory architectures using OpenMP. The results for the different implementations for a $2^{20}$-equations tridiagonal system using single floating point's arithmetic are shown in the next figure.

**Figure 23 : Performance of different tridiagonal solvers (Source: Quesada-Barriuso et al. [73])**

| Algorithm | Thomas | CR | Bondeli |
|---|---|---|---|
| Sequential | | | |
| Time | 0.0547 s | ——- | ——- |
| GFLOPS | 0.14 | | |
| OpenMP | | | |
| Time | —— | 0.0507 s | 0.0365 s |
| GFLOPS | | 0.44 | 0.78 |
| CUDA | | | |
| Time | —— | 0.0029 s | 0.0058 s |
| GFLOPS | | 15.33 | 17.50 |

In [74] Zhao and Yu developed a GPU shared memory-based chunked cyclic reduction (see Figure 24) under the constraint of the capacity of the shared memory. Computational results showed that GPU shared memory chunked cyclic reduction exhibits high efficiency by Nvidia TITAN with 48k shared memory, and GPU shared memory chunked cyclic reduction can solve a tridiagonal system with 262,144-by-262,144 coefficient matrix in 1.768 ms.

**Figure 24 : Chunking for cyclic reduction of a 16x16 tridiagonal system (Source: Zhao and Yu [74])**




Other interesting works for various tridiagonal solvers on GPUs can be found in [75], [76], [77] and [78] .

## 4.4 Summary

In conclusion, there is a rich literature for option pricing on FGPAs, but few works have been done in Implicit Finite Different Scheme and only one for the Crank-Nicolson. Cyclic Reduction has many implementations on GPUs but it hasn't been ported on FGPA.

This research continues in the next chapter with accuracy analysis of Crank-Nicolson scheme for solving Black-Scholes PDE with use of the Normalized Cyclic Reduction variant.

# Chapter 5:
# Accuracy Analysis of Crank-Nicolson Finite Difference Method with Normalized-Cyclic Reduction as a Tridiagonal Solver for Option Pricing

## 5.1 Introduction

Since numerical instability of algorithms can be derived from many sources such as truncation error of the Finite Difference scheme, round-off errors of floating precision, arithmetic overflow and ill-conditioned problems, a process has to be designed to measure these risks. In this chapter experiments for measuring the precision of the proposed method had been designed and implemented. These experiments are divided in two categories, the first category is based on a mixed-precision process and the second category is a process to determine a custom precision of floating point arithmetic that the method produces good enough results.

As described in Chapter 4:, some research works provide accuracy analysis for option pricing either theoretical or experimental. In our opinion, in such systems that combine heavy mathematical background and numerical analysis methods with heterogeneous computing systems, an accuracy analysis must always be performed. A work that describes clear well the behavior of errors in finite difference methods for option pricing is [79].

The work of Goddeke and Strzodka [72] analyzes the failure of single precision in Cyclic Reduction on a GPU implementation, also in Conjugate Gradient Solver [80] for general PDE solvers on FPGAs, and suggests a mixed-precision process called iterative refinement [81]. From this work the first category of experiments were designed, where three different approaches are measured in various error metrics. The first is single precision, the second approach is a mixed-precision where initialization of parameters of the tridiagonal system are computed in double precision and the solver makes computations in single precision, last all computation are made in double precision.

## 5.2 Error metrics

Several different error measures were used to calculate the errors. To measure the error across the solution vector, the l2 norm was used to measure two kinds of errors:

$$absolute\ error = \|\hat{x} - x\|_2 \tag{5.1}$$

$$relative\ error = \frac{\|\hat{x} - x\|_2}{\|x\|_2} \tag{5.2}$$

Where,

$$\|x\|_2 = \left(\sum |x_i|^2\right)^{1/2}$$

Also to measure the quality of the solution at values near the money, as these are the intervals in the solution vector that the market is interested in, the absolute and relative error was measured at points left and right of the strike price:

$$absolute\ error = |\hat{x} - x| \tag{5.3}$$

$$relative\ error = \frac{|\hat{x} - x|}{|x|} \tag{5.4}$$

Where $\hat{x}$ is the approximation of the method and $x$ is the true value obtained by the analytical solution of Black-Scholes model as described in section 2.4.3.1.

For the second sets of experiments, with the MPFR library where the arithmetic precision is determined, the root mean squared error is used:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{x}_i - x_i)^2} \tag{5.5}$$

The RMSE is a quadratic scoring rule that also measures the average magnitude of the error. It's the square root of the average of squared differences between prediction and actual observation. Taking the square root of the average squared errors has some interesting implications for RMSE. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. This means the RMSE should be more useful when large errors are particularly undesirable.

## 5.3 First set of experiments

There are four experiments in this first set, which try to explore the behavior of the proposed method in terms of accuracy. These are:

1. Which boundary conditions?
2. Which system size?
3. How many time steps?
4. Different parameters

The first three experiments have same parameters for the European Call Option and the last one different.

### 5.3.1 Experiment 1: Which boundary conditions?

In this experiment, the goal was to ascertain how different the algorithm works for the different boundary conditions. The errors were measured with the l2 norm in two cases, one for the whole vector and the second for the underlying product values from 45 to 55. The values that do not belong to some point of the lattice are found by linear interpolation.

The first set of experiments was made with values for the European Call Option (see 3.2):

Smax = 100, K = 50, σ =0.3, r = 0.01, T = 1

The experiments were:

| N | M | ds | dt |
|---|---|---|---|
| 9 | 8 | 100/9 | 1/8 |
| 17 | 16 | 100/17 | 1/16 |
| 33 | 32 | 100/33 | 1/32 |
| 65 | 64 | 100/65 | 1/64 |
| 129 | 128 | 100/129 | 1/128 |
| 257 | 256 | 100/257 | 1/256 |
| 513 | 512 | 100/513 | 1/512 |
| 1025 | 1024 | 100/1025 | 1/1024 |
| 2049 | 2048 | 100/2049 | 1/2048 |
| 4097 | 4096 | 100/4097 | 1/4096 |
| 8193 | 8192 | 100/8193 | 1/8192 |
| 16385 | 16384 | 100/16385 | 1/16384 |
| 32769 | 32768 | 100/32769 | 1/32768 |

N here corresponds to the lattice method and is chosen in such a way as to give a tridiagonal system of $2^n$ size and M is the corresponding number of time steps.

#### 5.3.1.1 *Relative error (l2-norm) near strike price*

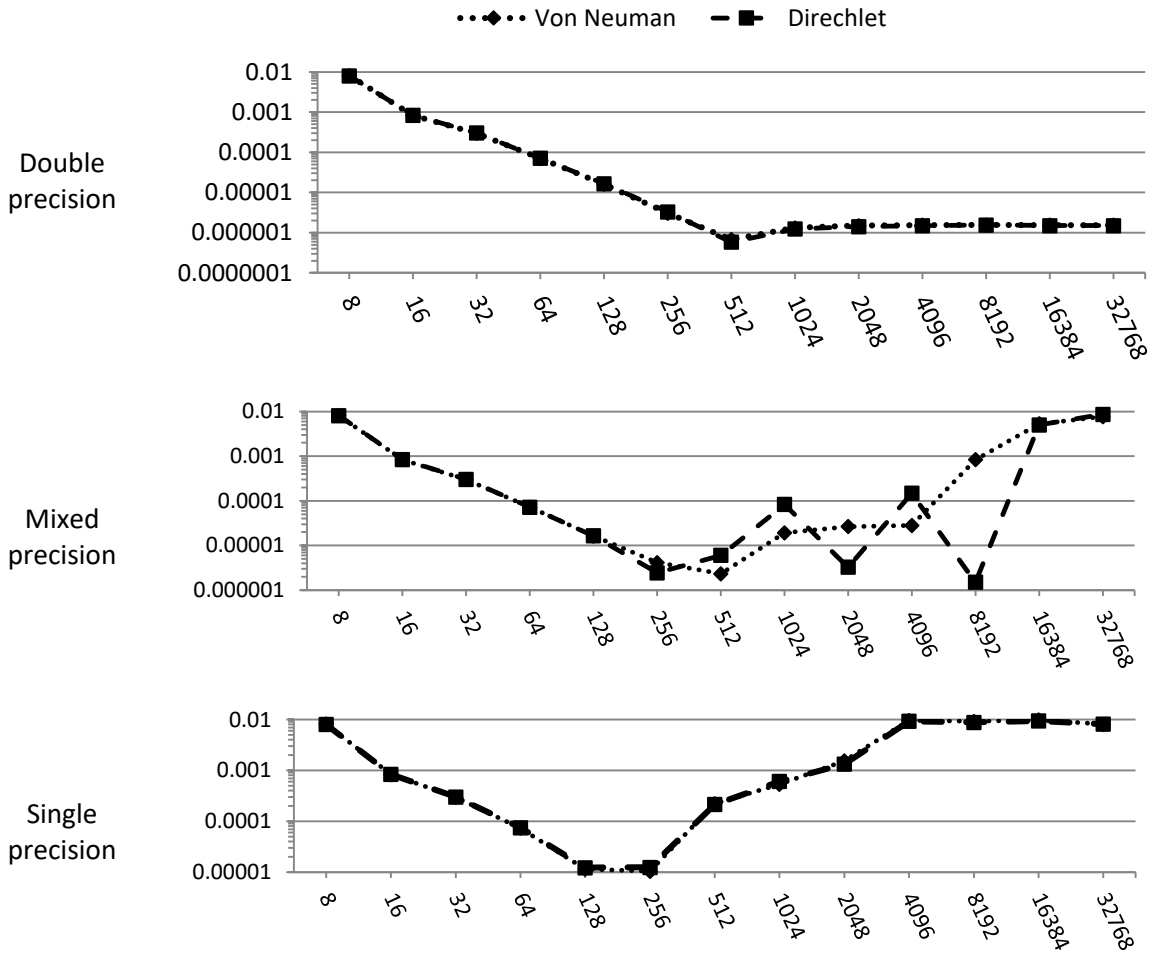
The relative error for the two cases of boundary conditions is shown in the following tables:

**Table 8: Relative error near strike price of European call option.**

| Von Neumman | | | Direchlet | | |
|---|---|---|---|---|---|
| double | mixed | single | double | mixed | single |
| 8.14599E-03 | 8.14622E-03 | 8.14642E-03 | 8.02717E-03 | 8.02708E-03 | 8.02735E-03 |
| 8.30628E-04 | 8.30196E-04 | 8.30413E-04 | 8.40478E-04 | 8.40727E-04 | 8.40349E-04 |
| 2.98362E-04 | 2.98165E-04 | 2.98235E-04 | 3.02087E-04 | 3.02190E-04 | 3.02130E-04 |
| 7.03640E-05 | 7.16543E-05 | 7.36810E-05 | 7.15014E-05 | 7.17693E-05 | 7.51174E-05 |
| 1.60649E-05 | 1.62187E-05 | 1.15297E-05 | 1.65372E-05 | 1.63748E-05 | 1.22624E-05 |
| 3.06370E-06 | 4.12824E-06 | 1.05068E-05 | 3.30894E-06 | 2.43035E-06 | 1.24487E-05 |
| 6.73908E-07 | 2.32333E-06 | 2.23407E-04 | 5.81642E-07 | 5.96971E-06 | 2.15250E-04 |
| 1.32458E-06 | 1.91304E-05 | 5.36277E-04 | 1.23176E-06 | 8.28769E-05 | 6.10406E-04 |
| 1.49318E-06 | 2.64121E-05 | 1.54130E-03 | 1.42512E-06 | 3.26163E-06 | 1.33036E-03 |
| 1.53934E-06 | 2.79831E-05 | 9.65004E-03 | 1.48367E-06 | 1.46631E-04 | 9.30834E-03 |
| 1.54954E-06 | 8.29981E-04 | 9.19101E-03 | 1.54954E-06 | 1.50014E-06 | 8.81434E-03 |
| 1.54851E-06 | 5.28387E-03 | 9.81547E-03 | 1.50227E-06 | 4.98263E-03 | 9.42968E-03 |
| 1.54710E-06 | 7.63115E-03 | 8.10463E-03 | 1.50258E-06 | 8.58467E-03 | 8.14261E-03 |

The figures are presented with the corresponding precision and the relative error. It can be noticed that in double precision Direchlet boundary condition have the same effect as Von Neuman for the vector near the exercise price. Subsequently, in mixed precision procedure, it is noticed that, up to size N = 512 the algorithm converges after that system size the error grows again, the Direchlet conditions give a strange behavior to the algorithm, depending on the discretization of the mesh as can be seen in the following figure. Last, for the single precision there is no difference between the two different ways of calculating boundary conditions, but it appears that above system size N = 256 the algorithm stops producing exploitable results.

**Figure 25: log relative error (l2-norm) near strike price versus system size.**




### 5.3.1.2 *Relative error (l2-norm) of the solution vector*

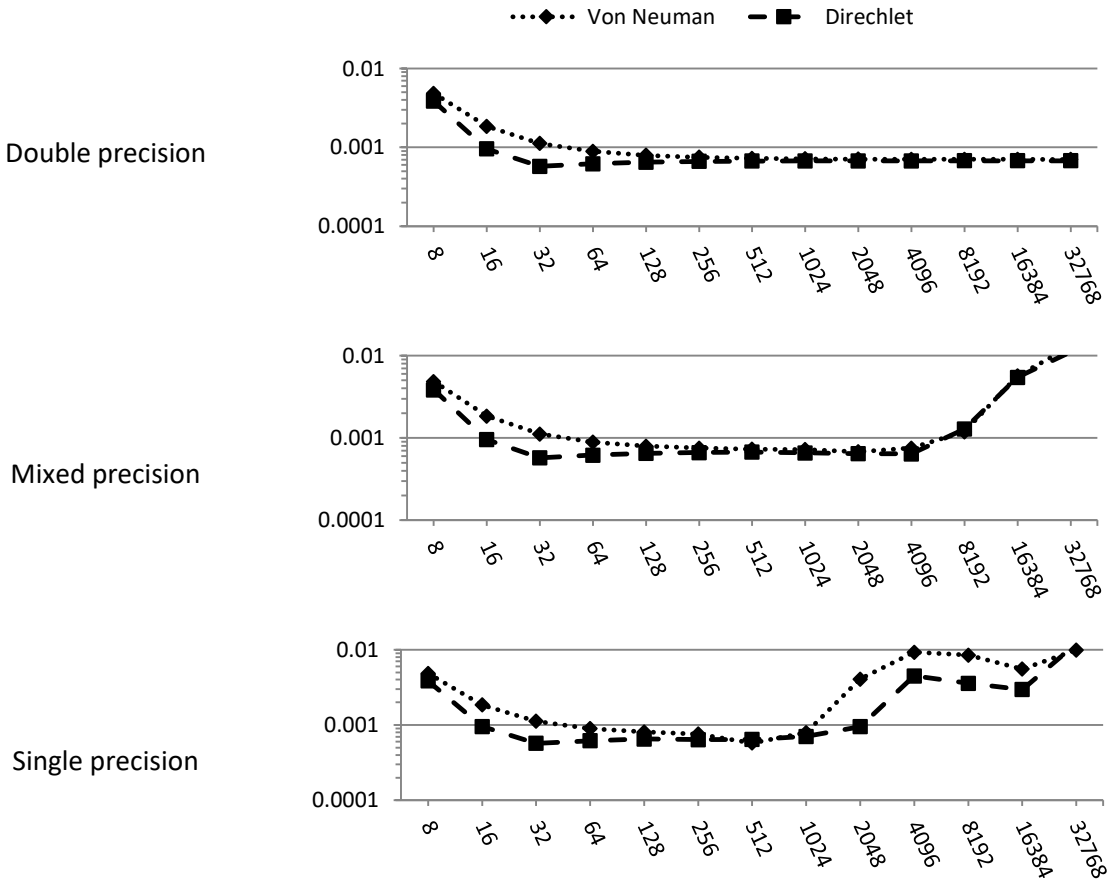
Then the errors for the entire solution vector are presented:

**Table 9: Relative error for solution vector of European call option.**

| Von Neumman | | | Direchlet | | |
|---|---|---|---|---|---|
| double | mixed | single | double | mixed | single |
| 4.84475E-03 | 4.84505E-03 | 4.84510E-03 | 3.85424E-03 | 3.85429E-03 | 3.85427E-03 |
| 1.84694E-03 | 1.84704E-03 | 1.84815E-03 | 9.56507E-04 | 9.56729E-04 | 9.56497E-04 |
| 1.11829E-03 | 1.11832E-03 | 1.12059E-03 | 5.72960E-04 | 5.72937E-04 | 5.72884E-04 |
| 8.90308E-04 | 8.89097E-04 | 8.97297E-04 | 6.18929E-04 | 6.18963E-04 | 6.18998E-04 |
| 7.94075E-04 | 7.94420E-04 | 8.08792E-04 | 6.49852E-04 | 6.50600E-04 | 6.52963E-04 |
| 7.48155E-04 | 7.55384E-04 | 7.60216E-04 | 6.63416E-04 | 6.63933E-04 | 6.39119E-04 |
| 7.25943E-04 | 7.31925E-04 | 5.77633E-04 | 6.69536E-04 | 6.73679E-04 | 6.43812E-04 |
| 7.14268E-04 | 7.25708E-04 | 7.91591E-04 | 6.72422E-04 | 6.58684E-04 | 7.06323E-04 |
| 7.08654E-04 | 6.81829E-04 | 4.07170E-03 | 6.73821E-04 | 6.45955E-04 | 9.50025E-04 |
| 7.05851E-04 | 7.54879E-04 | 9.26380E-03 | 6.74510E-04 | 6.41741E-04 | 4.48241E-03 |
| 7.04449E-04 | 1.19082E-03 | 8.48195E-03 | 6.74852E-04 | 1.28191E-03 | 3.57771E-03 |
| 7.03750E-04 | 5.66508E-03 | 5.54274E-03 | 6.75022E-04 | 5.46239E-03 | 2.95378E-03 |
| 7.03398E-04 | 1.29623E-02 | 9.90065E-03 | 6.75107E-04 | 1.10744E-02 | 1.20695E-02 |

Same as before the figures show the error for the three precision (double, mixed and single).

**Figure 26: Relative error (l2-norm) for the solution vector.**



In the overall vector, Direchlet boundary conditions seem to work better, but the gain is very small compared to the advantages of Von Neumman. The result of the experiment shows that the decision to choose Von Neumman boundary conditions is correct, because at bigger systems of size N there is no practically any difference.

However, some worrying phenomena have arisen for the algorithm behavior. At first sight it seems to be a problem of arithmetic precision, but it may also come from other type of errors, such as discretization errors. For this reason, a more extensive analysis needs to be done.

### 5.3.2 Experiment 2: Which system size?

Following the first experiment results, in the second experiment the algorithm has been changed so that it could solve systems of size $2^n$-1. This has been done to measure the effect of arithmetic precision on space discretization of S. So there are two systems of size $2^n$-1 and $2^n$, the former can be perfectly represented by floating point arithmetic as ds is calculated by dividing with number of power of two. Thus, no information is lost due to round off errors of floating arithmetic.

The comparison between these two versions of the same algorithm is done exactly at the exercise price as a point measurement, while the error for the whole vector continues to be calculated in this and in all subsequent experiments.

The scope of this experiment is to exclude some sources of errors and to see in absolute terms the behavior of the error. The errors in all experiments are derived from the difference between the analytical solution and the approximation solution of the Black-Scholes model for European call option. The same parameters for the option have been used as in the first experiment.

The reference solution for this experiment is from BS closed form solution:

**Exact solution = 6.18413565**

**Table 10: Price approximation of European call option for different discretization and precision.**

| System size $2^n$ | | | System size $2^n$-1 | | |
|---|---|---|---|---|---|
| double | mixed | single | double | mixed | single |
| 6.60771900 | 6.60771680 | 6.60771596 | 5.587887165 | 5.58788681 | 5.587884426 |
| 6.30098177 | 6.30097699 | 6.30098009 | 6.055699279 | 6.055704117 | 6.055699348 |
| 6.21501272 | 6.21501112 | 6.21501207 | 6.152834152 | 6.152838707 | 6.152839661 |
| 6.19207869 | 6.19208980 | 6.19210410 | 6.176349122 | 6.176352978 | 6.176347733 |
| 6.18614537 | 6.18614578 | 6.18610024 | 6.182185393 | 6.182200909 | 6.18214035 |
| 6.18463590 | 6.18460250 | 6.18466663 | 6.183642177 | 6.183616638 | 6.183702469 |
| 6.18425530 | 6.18424869 | 6.18562269 | 6.184006391 | 6.184035301 | 6.185444832 |
| 6.18415982 | 6.18427801 | 6.18106198 | 6.184097527 | 6.18379879 | 6.181346893 |
| 6.18413594 | 6.18398738 | 6.17630887 | 6.184120357 | 6.184527397 | 6.180389404 |
| 6.18412998 | 6.18391466 | 6.24365950 | 6.184126087 | 6.184597015 | 6.246400356 |
| 6.18412851 | 6.17915797 | 6.24124098 | 6.184127532 | 6.18431139 | 6.243336678 |
| 6.18412814 | 6.15170574 | 6.24446893 | 6.184127899 | 6.171162605 | 6.220122814 |
| 6.18412805 | 6.14018035 | 6.14028072 | 6.184127991 | 6.099887848 | 6.132685661 |

### 5.3.2.1 *Absolute error at strike price*

The accuracy at the exercise value was measured with absolute error while for the whole vector the relative error l2-norm was used. The following tables show the absolute error in the exercise price for the two types of systems.

**Table 11: Absolute error at strike price of European call option.**

| System size $2^n$ | | | System size $2^n$-1 | | |
|---|---|---|---|---|---|
| double | mixed | single | double | mixed | single |
| 4.23583E-01 | 4.23581E-01 | 4.23580E-01 | 5.96251E-01 | 5.96249E-01 | 5.96248E-01 |
| 1.16846E-01 | 1.16841E-01 | 1.16844E-01 | 1.28436E-01 | 1.28432E-01 | 1.28436E-01 |
| 3.08771E-02 | 3.08755E-02 | 3.08764E-02 | 3.13015E-02 | 3.12969E-02 | 3.12960E-02 |
| 7.94305E-03 | 7.95415E-03 | 7.96845E-03 | 7.78653E-03 | 7.78267E-03 | 7.78792E-03 |
| 2.00972E-03 | 2.01013E-03 | 1.96460E-03 | 1.95026E-03 | 1.93474E-03 | 1.99530E-03 |
| 5.00249E-04 | 4.66850E-04 | 5.30985E-04 | 4.93472E-04 | 5.19010E-04 | 4.33180E-04 |
| 1.19656E-04 | 1.13037E-04 | 1.48704E-03 | 1.29257E-04 | 1.00347E-04 | 1.30918E-03 |
| 2.41682E-05 | 1.42363E-04 | 3.07367E-03 | 3.81217E-05 | 3.36859E-04 | 2.78876E-03 |
| 2.88208E-07 | 1.48270E-04 | 7.82678E-03 | 1.52920E-05 | 3.91748E-04 | 3.74624E-03 |
| 5.66539E-06 | 2.20987E-04 | 5.95238E-02 | 9.56144E-06 | 4.61367E-04 | 6.22647E-02 |
| 7.14231E-06 | 4.97768E-03 | 5.71053E-02 | 8.11645E-06 | 1.75741E-04 | 5.92010E-02 |
| 7.50602E-06 | 3.24299E-02 | 6.03333E-02 | 7.74966E-06 | 1.29730E-02 | 3.59872E-02 |
| 7.59524E-06 | 4.39553E-02 | 4.38549E-02 | 7.65725E-06 | 8.42478E-02 | 5.14500E-02 |

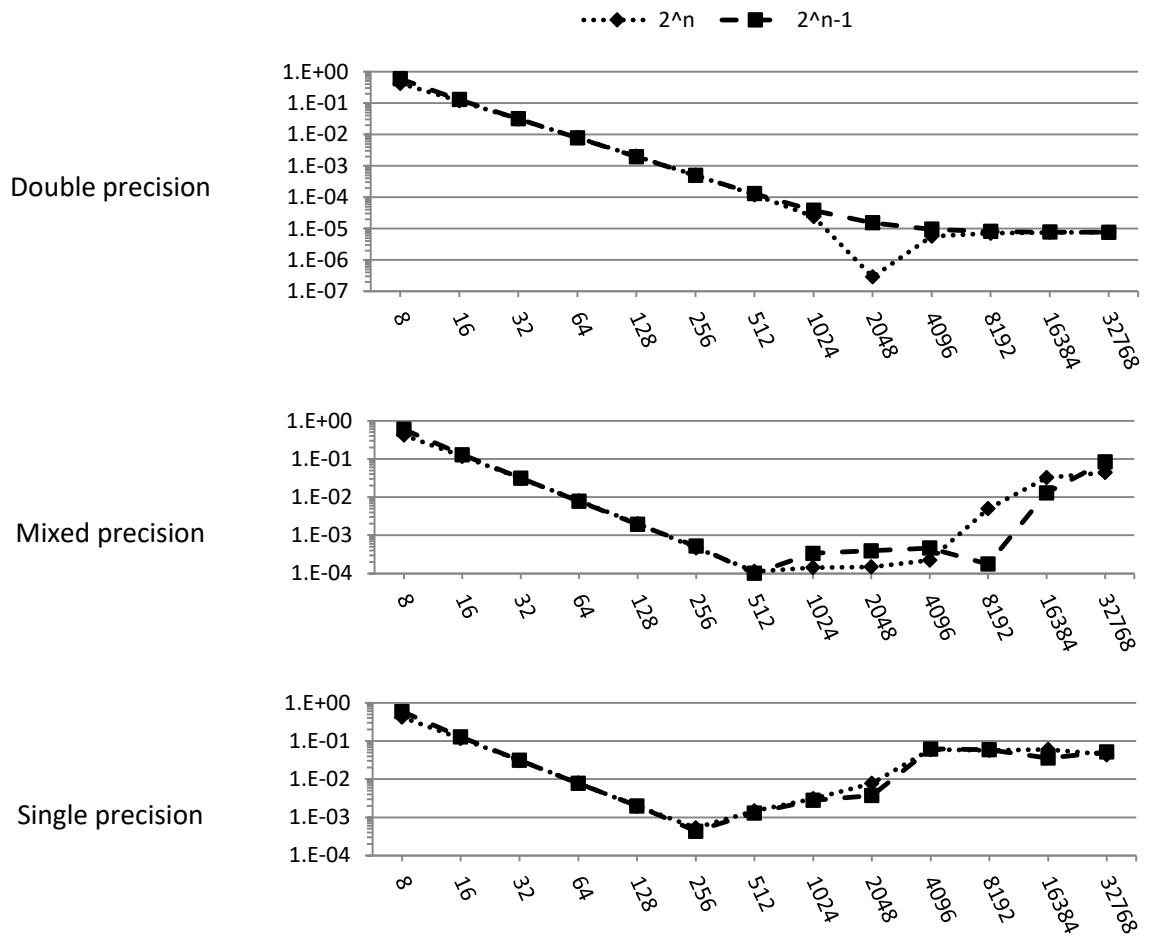
The figure below shows the absolute error at strike price. The results appear to have exactly the same behavior as the previous experiment. For system size 2048 this extreme value is formed with a value of K = 50 and Smax = 100 in the system $2^n$-1 because the value S = 50 in

the discretization belongs to a lattice point, whereas in the system $2^n$ it is calculated by linear interpolation. For this set of measurements, the algorithm works perfectly up to the 2048 system where for each doubling system the error is doubling, whereas after this size the system converges to an error of less than 0.00001.

In the mixed precision process there is no great difference between the two types of systems. Here the systems work the same up to size 512 and for larger ones they have similar behavior. The algorithm works correctly up to 512 size where it doubles and here the error for each doubling of the system, as shown in the figure below.

Exactly, the same behavior as mixed precision is also present in the single precision with the only difference that the algorithm stops working correctly in an order of magnitude earlier, ie in system 256, as shown in the figure.

**Figure 27: Absolute error in log scale at strike price versus system size.**




### 5.3.2.2 *Relative error (l2-norm) of the solution vector*

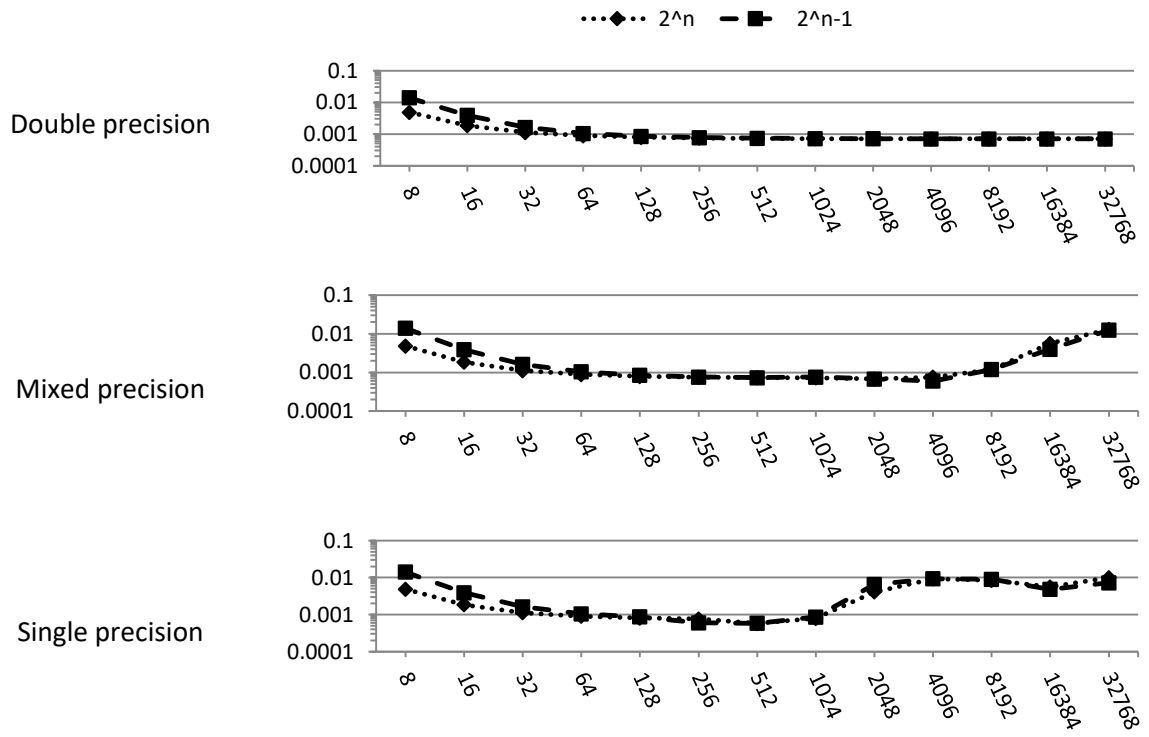
Below are the tables with the relative error across the vector of the solution, for the different sizes and different representation precisions.

**Table 12: Relative error for solution vector of European call option.**

| System size $2^n$ | | | System size $2^n$-1 | | |
|---|---|---|---|---|---|
| double | mixed | single | double | mixed | single |
| 4.84475E-03 | 4.84505E-03 | 4.84510E-03 | 1.40259E-02 | 1.40261E-02 | 1.40263E-02 |
| 1.84694E-03 | 1.84704E-03 | 1.84815E-03 | 3.88609E-03 | 3.88544E-03 | 3.88660E-03 |
| 1.11829E-03 | 1.11832E-03 | 1.12059E-03 | 1.61936E-03 | 1.61883E-03 | 1.61864E-03 |
| 8.90308E-04 | 8.89097E-04 | 8.97297E-04 | 1.04093E-03 | 1.04250E-03 | 1.04525E-03 |
| 7.94075E-04 | 7.94420E-04 | 8.08792E-04 | 8.52094E-04 | 8.50744E-04 | 8.63026E-04 |
| 7.48155E-04 | 7.55384E-04 | 7.60216E-04 | 7.73792E-04 | 7.65470E-04 | 6.01098E-04 |
| 7.25943E-04 | 7.31925E-04 | 5.77633E-04 | 7.37585E-04 | 7.33991E-04 | 5.83041E-04 |
| 7.14268E-04 | 7.25708E-04 | 7.91591E-04 | 7.20117E-04 | 7.60407E-04 | 8.56222E-04 |
| 7.08654E-04 | 6.81829E-04 | 4.07170E-03 | 7.11534E-04 | 6.82754E-04 | 6.55896E-03 |
| 7.05851E-04 | 7.54879E-04 | 9.26380E-03 | 7.07279E-04 | 6.01126E-04 | 9.30001E-03 |
| 7.04449E-04 | 1.19082E-03 | 8.48195E-03 | 7.05161E-04 | 1.18990E-03 | 8.96651E-03 |
| 7.03750E-04 | 5.66508E-03 | 5.54274E-03 | 7.04106E-04 | 4.00538E-03 | 4.79462E-03 |
| 7.03398E-04 | 1.29623E-02 | 9.90065E-03 | 7.03580E-04 | 1.24239E-02 | 7.18956E-03 |

And as chart form are presented in the following figure:

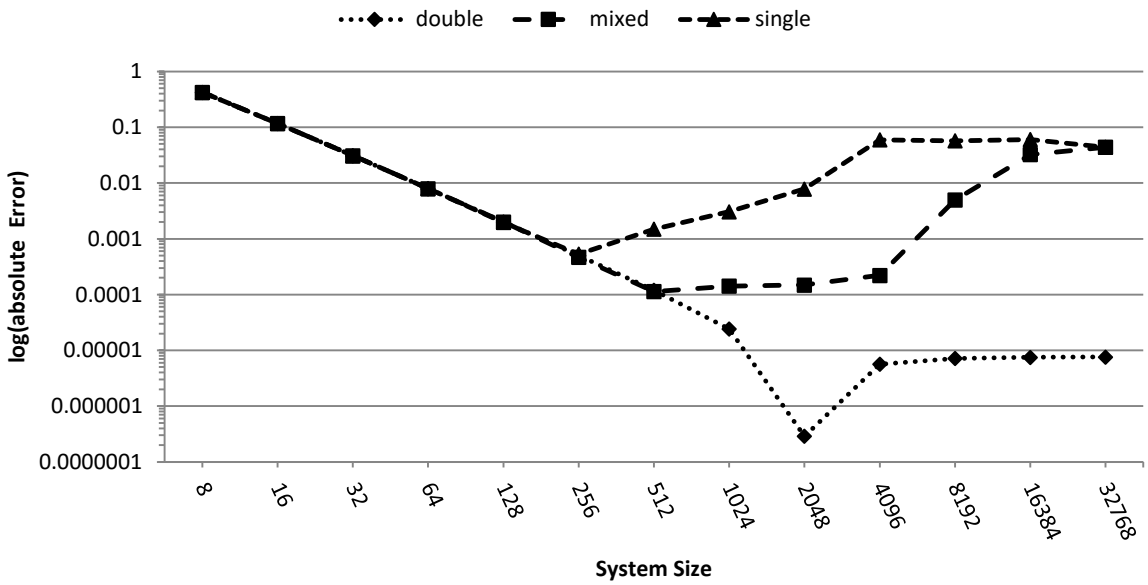
**Figure 28: Relative error (l2-norm) for the solution vector versus system size.**



It can be observed that the tridiagonal system size does not play a role if it is odd or even number, eg 512 or 511. In terms of relative error, it seems to work better for small grids the $2^n$ tridiagonal system size. The analysis will continue with this size because architecture in Hardware is designed for $2^n$ tridiagonal system size.

The graph below shows the absolute error in the exercise price only for the above size case. It seems clear that the failure of single precision to reduce the error after 256. It is encouraging that with a simple input of data from the phase initialization of coefficients a, c, d, e to double precision leads the algorithm to function as it should for a even doubling a system. This fact shows us that with a better mixed precision process it can deliver better results.

**Figure 29: Absolute error in log scale at strike price versus system size.**




### 5.3.3 Experiment 3: How many time steps?

The two previous experiments responded to questions that had to do with what boundary conditions should be used and the role of $2^n$ or $2^n$-1 sized tridiagonal system. Also, some initial conclusions were extracted for the interdependence of the precision with the errors. The increase for above a system size makes the solution worsen.

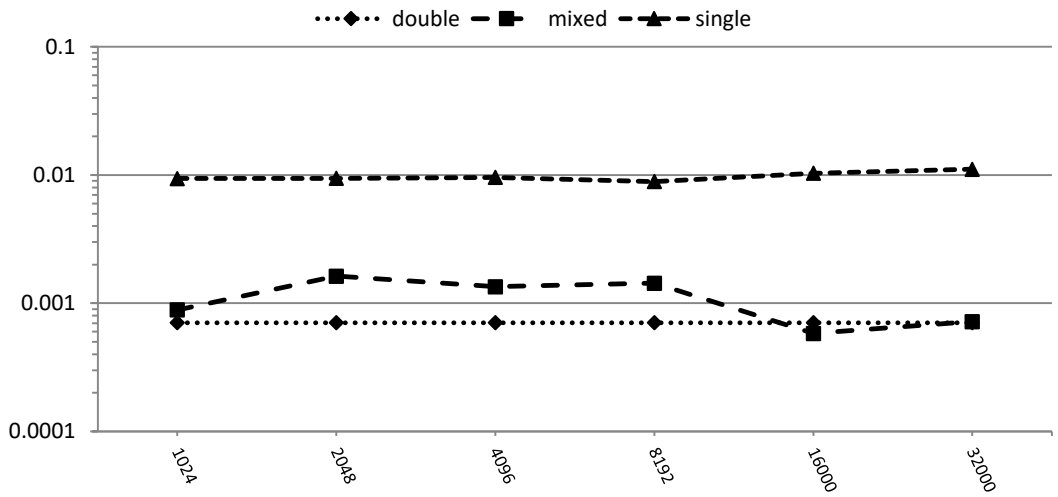
In this experiment, the question of how many time steps the algorithm in order to achieve a good solution is examined. This was based on the choice of Crank Nicolson's scheme versus the implicit, that in the course of time, larger dt steps can be made.

For each system size from N= $2^8$ = 256 to N= $2^{14}$ = 16384, different dt = T / M discrete time steps are tested. The absolute error is measured at three values, from left and right of the

exercise price, the underlying product values S = 45 and S = 55 respectively, as well as the exercise price. For completeness purposes, other types of errors are also calculated.

The figure below shows that for a tridiagonal system size of N = 8192 as the time steps are increased the mixed precision loses precision and then gained again across the solution vector. This indicates instability for the results for mixed precision. Double precision achieved the same accuracy in every setting of this experiment.

**Figure 30: Relative error (l2-norm) for the solution vector versus time steps.**

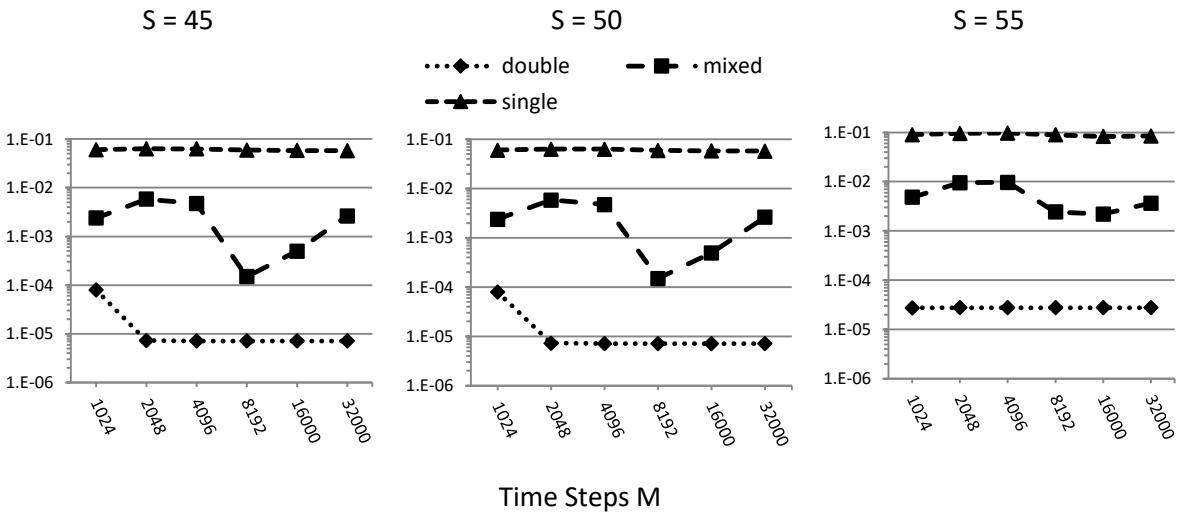



In the charts below, it can be noticed that for system size N = 8192 only 2048 time step are needed to achieved the best absolute error.

In this experiment, it turned out that the algorithm works just as well for fewer iterations than M = N. This was expected and was the main reason that Crank Nicolson was chosen as a finite difference over implicit, because it can do bigger dt steps.

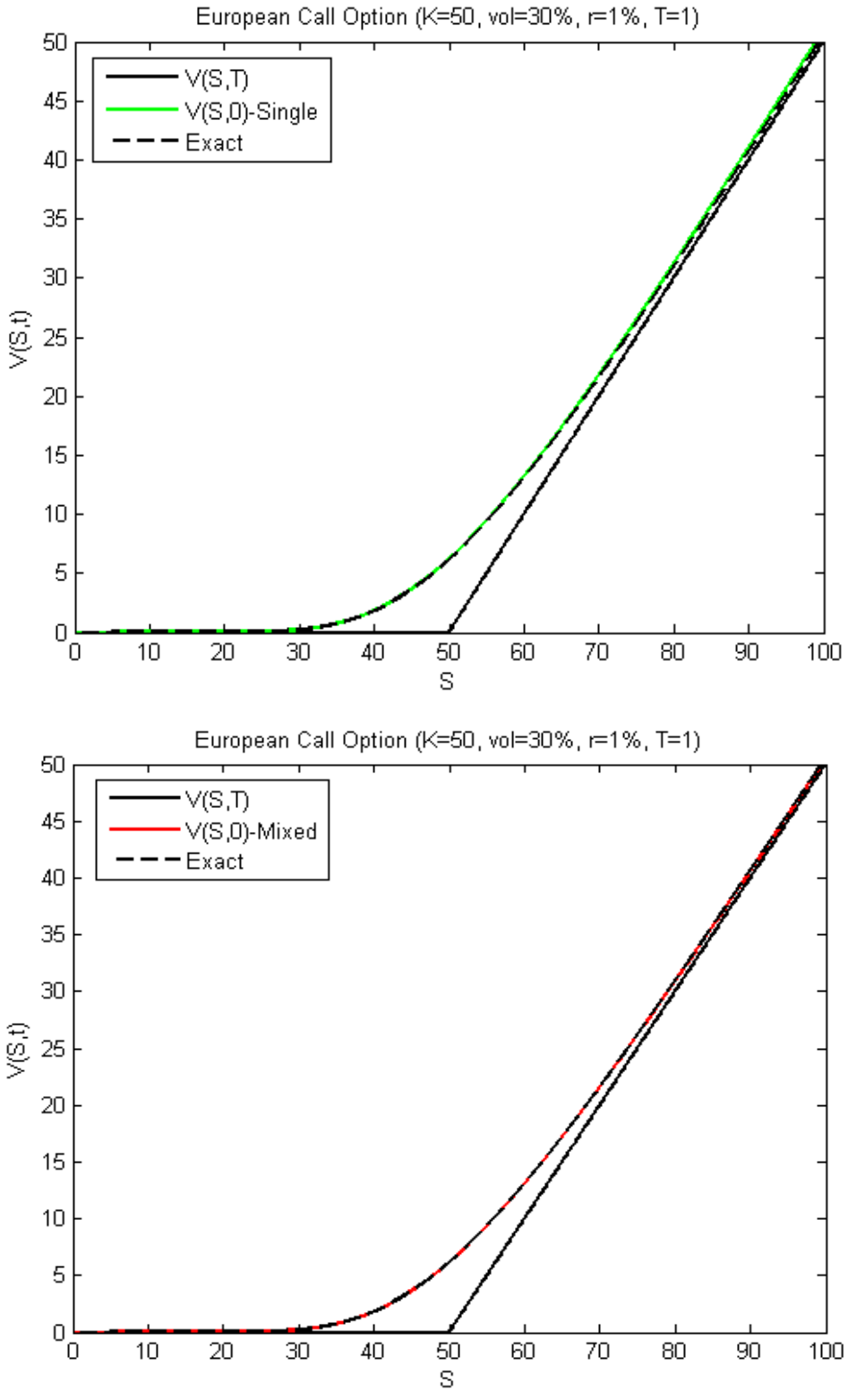
Also this experiment showed the weakness and the complete failure to produce prices for stock options that can be used by the market due to single precision.

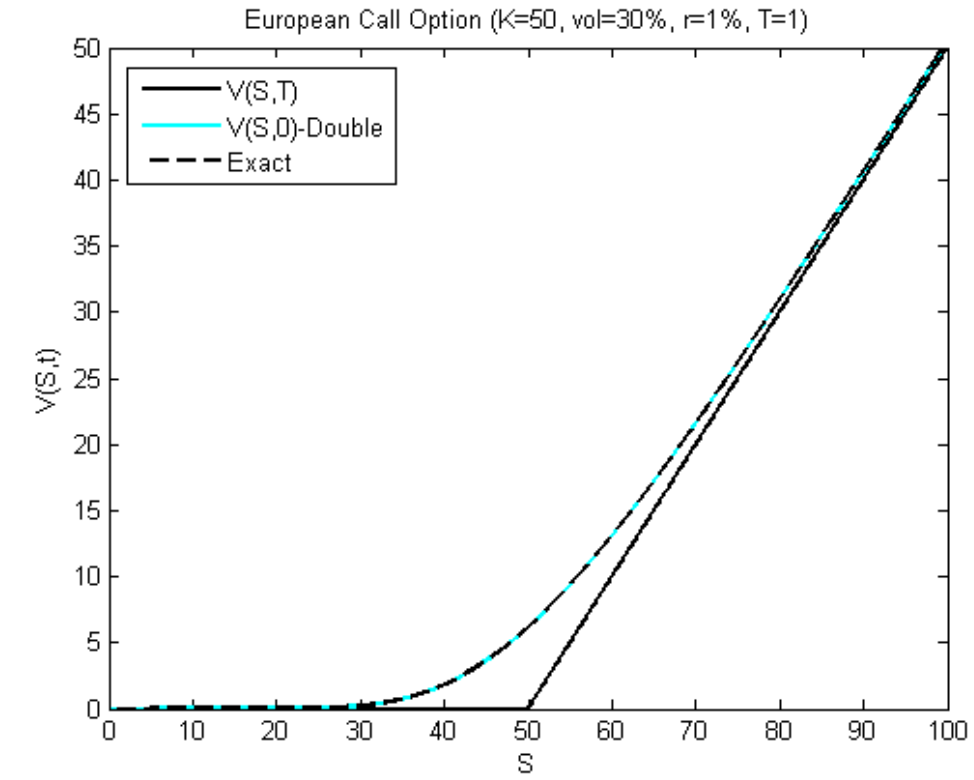
**Figure 31: Absolute error in log scale around strike price with N = 8192 versus time steps.**

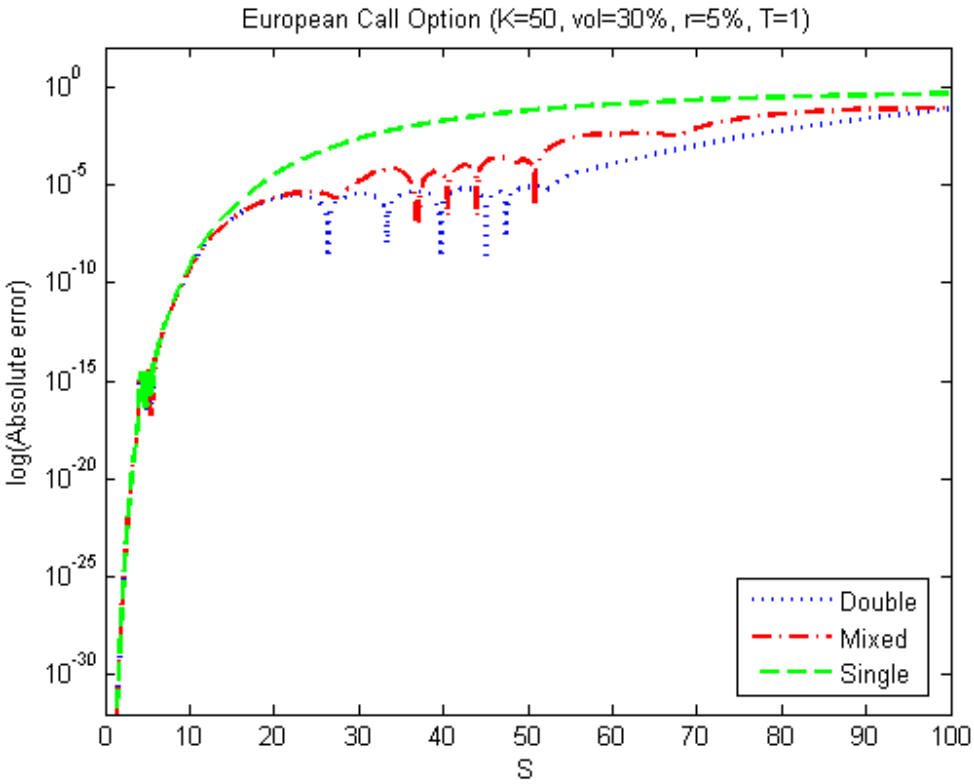


### 5.3.3.1 *Payoff functions and vector error*

**Figure 32: Payoff functions for European call option with different arithmetic precision and absolute error.**




European Call Option (K=50, vol=30%, r=1%, T=1)
V(S,T)
V(S,0)-Double
Exact
V(S,t)
S
0
5
10
15
20
25
30
35
40
45
50
0
10
20
30
40
50
60
70
80
90
100


European Call Option (K=50, vol=30%, r=5%, T=1)
log(Absolute error)
S
Double
Mixed
Single
10^0
10^-5
10^-10
10^-15
10^-20
10^-25
10^-30
0
10
20
30
40
50
60
70
80
90
100

### 5.3.4 Experiment 4: Different parameters

Next, knowing the problems for single precision, what is being sought was to look at whether the same behavior continues for different option parameters. The parameters that changed here are Smax = 500 from 100, this was a choice for measuring the errors far enough from the boundaries. Also, various values for interest rate r from 1% to 20% are also tested. The other parameters remain constant as in the other experiments. As far as the discretization is concerned, N = M was chosen for this experiment.

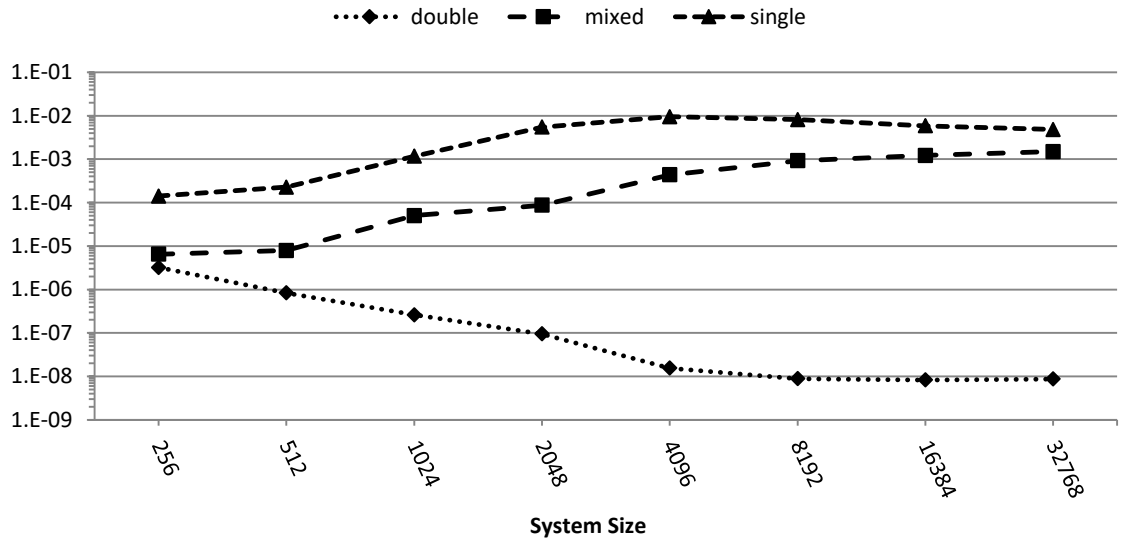
#### 5.3.4.1 *Increase of Smax = 500*

In this first test, only Smax has changed:

Smax = 500, K = 50, σ =0.3, r = 0.01, T = 1

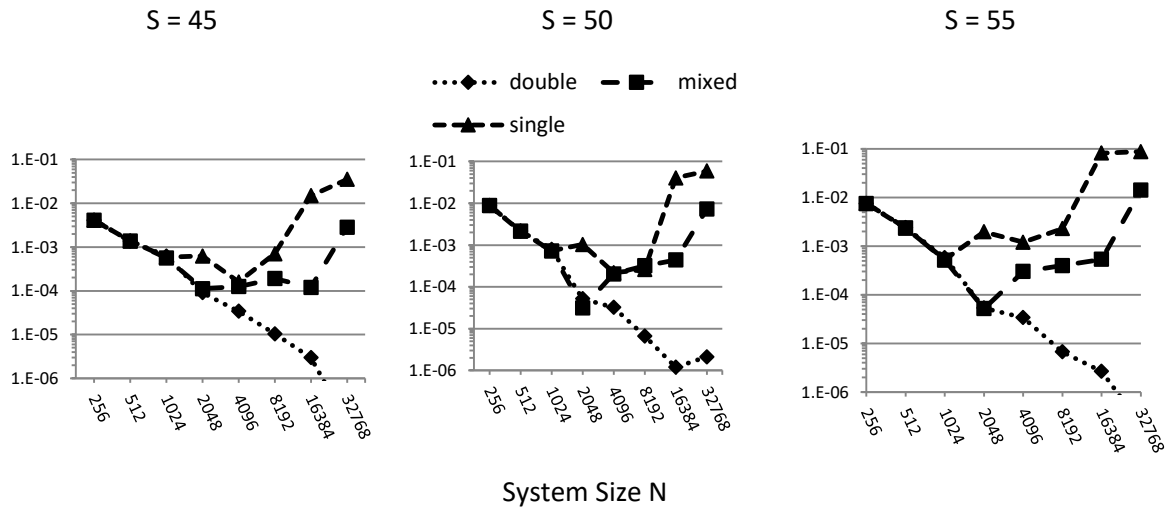
It can be observed that the relative norm l2 error has different behavior than previous experiments. Note that for double precision it must reach N = 8192 to stabilize at 1E-08 the error. The other two precisions started from a good accuracy in terms of relative error, and as the system grows, the error was increased:

**Figure 33: Relative error for Smax = 500.**




In terms of absolute error, measuring the same values as in previous experiment 3, it can be observed that the algorithms are all working correctly for larger systems for Smax = 500 rather than Smax = 100. This is perfectly reasonable since the discretization spatial step ds = Smax / N has grown. The same behavior it is observed with the previous experiments the different versions of the algorithm.

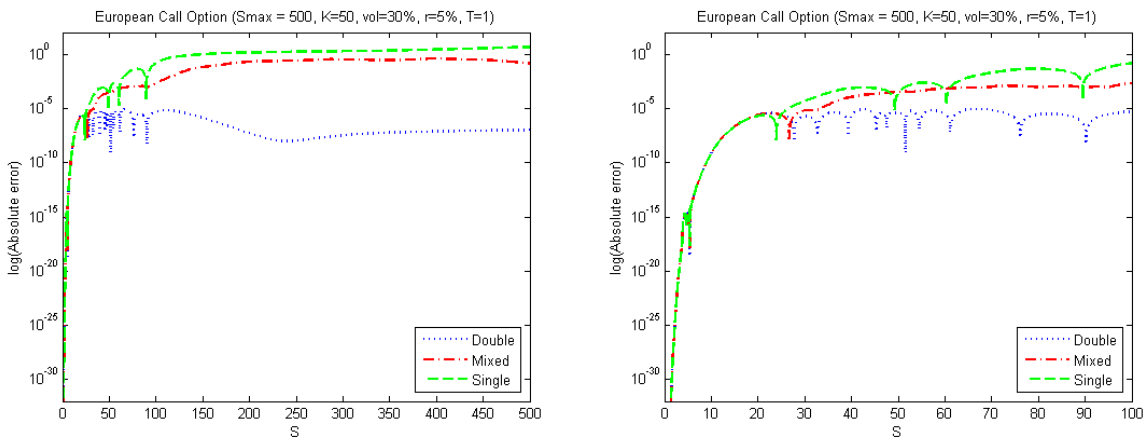
**Figure 34: Absolute error in log scale around strike price for Smax = 500 .**




Next, the absolute error for N = M = 8192 is displayed on a logarithmic scale, compared to the same execution for Smax = 100. It can be noticed, that close to the exercise value gives the same quality of solution while the algorithm gives better solution quality for the rest vector, for double precision. Therefore, it seems that there is a golden ratio for choosing between a K-value and an upper barrier of the value of the underlying Smax product.

In the other two executions of the code, the same phenomena emerged; the failure to produce a good solution at any point in the vector, as shown in the figure below left for the whole vector, right zoomed in showing only up to S = 100.

**Figure 35: Log scale absolute error European call option with Smax = 500.**



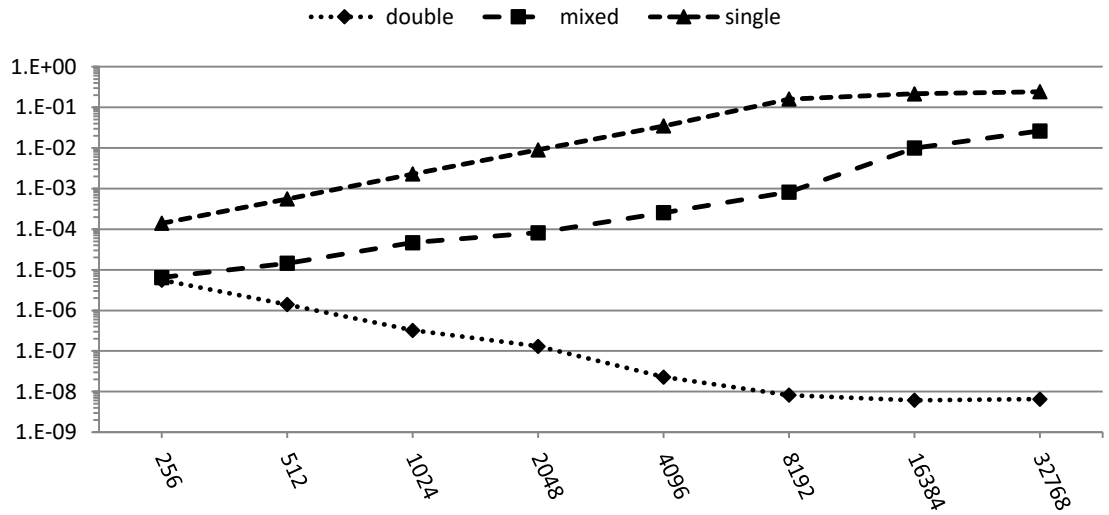
### 5.3.4.2 *Increase of interest rate from 1% to 20%*

Increasing the interest rate from r = 1% to r = 20% does not change the error behavior compared to the previous settings. The different implementations produce a smoother error results with more diverging effect. The experiment was done for all the intermediate interest rates (i.e 5%, 10%, 20%) results for r=20% are presented.

Smax = 500, K = 50, σ =0.3, r = 0.2, T = 1

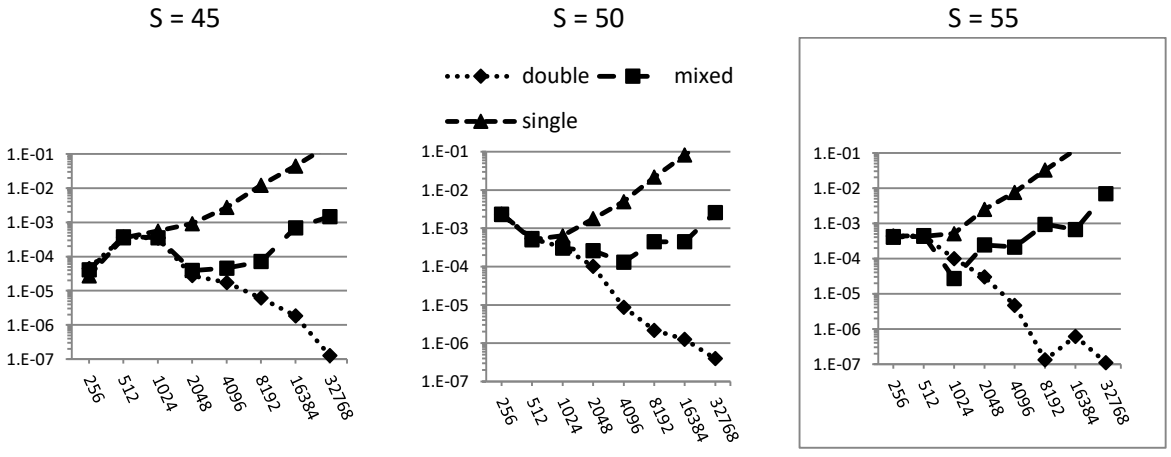
The relative error across the solution vector:

**Figure 36: Relative error for Smax = 500 and r=20%.**



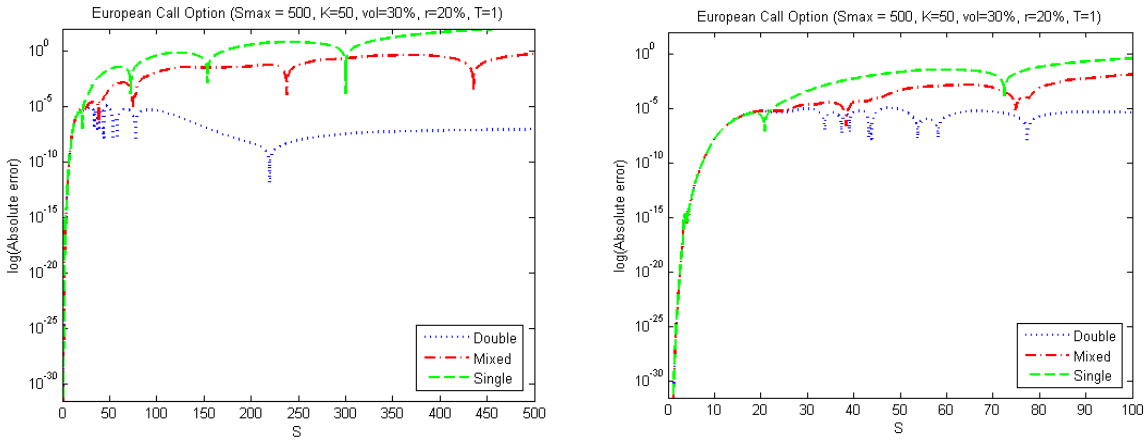

The absolute errors in prices are presented. It now seems that for this return and risk settings the single precision does not work neither for small size systems.

**Figure 37: Absolute error in log scale around strike price for Smax = 500 and r = 20%.**



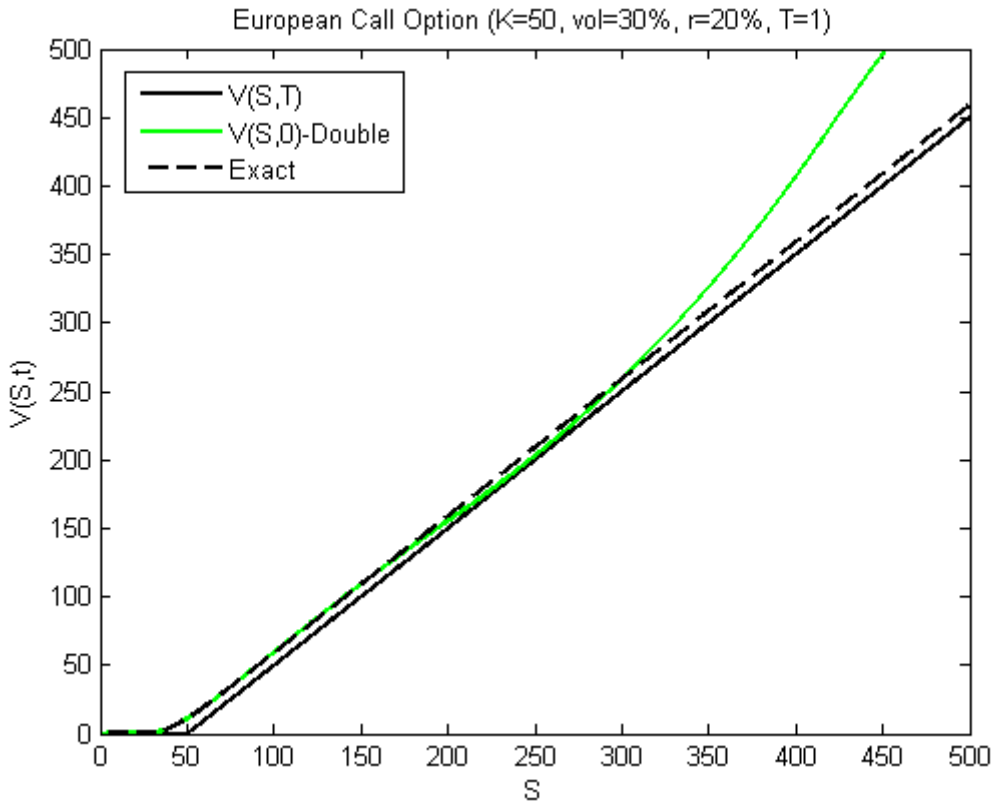
The absolute error for the solution vector for N = 8192 and M = 8192 is shown in the figure below, at the left is Smax = 500 and the same graph right only up to S = 100:

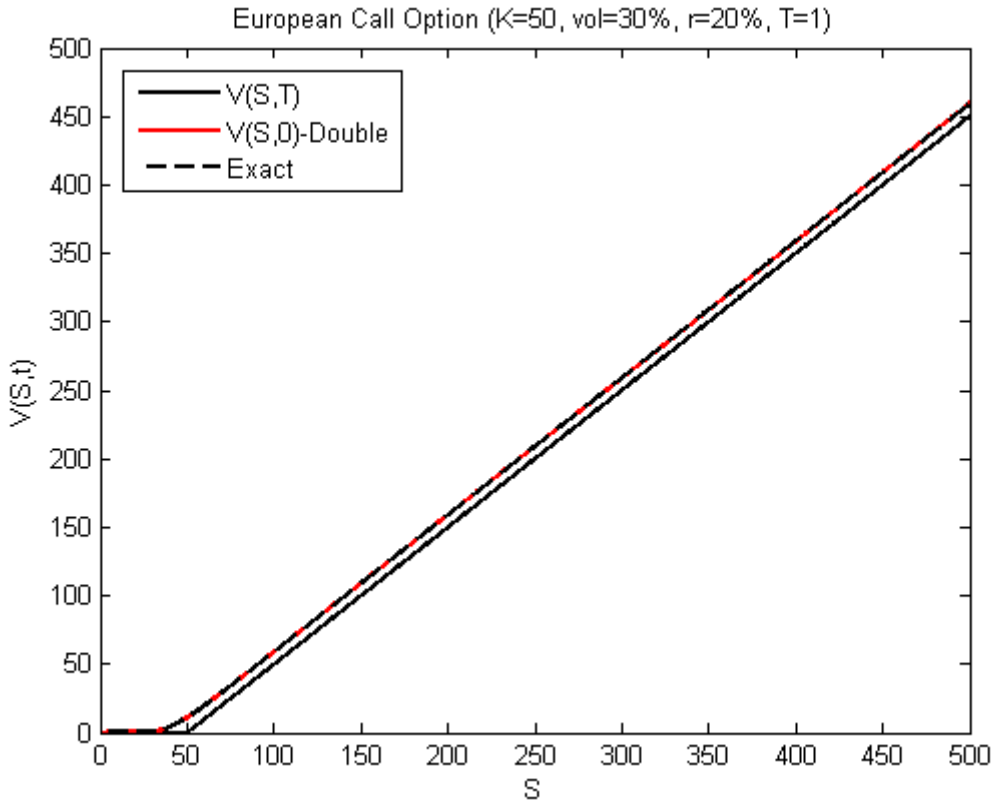
**Figure 38: Log scale absolute error European call option with Smax = 500 and r=20%.**

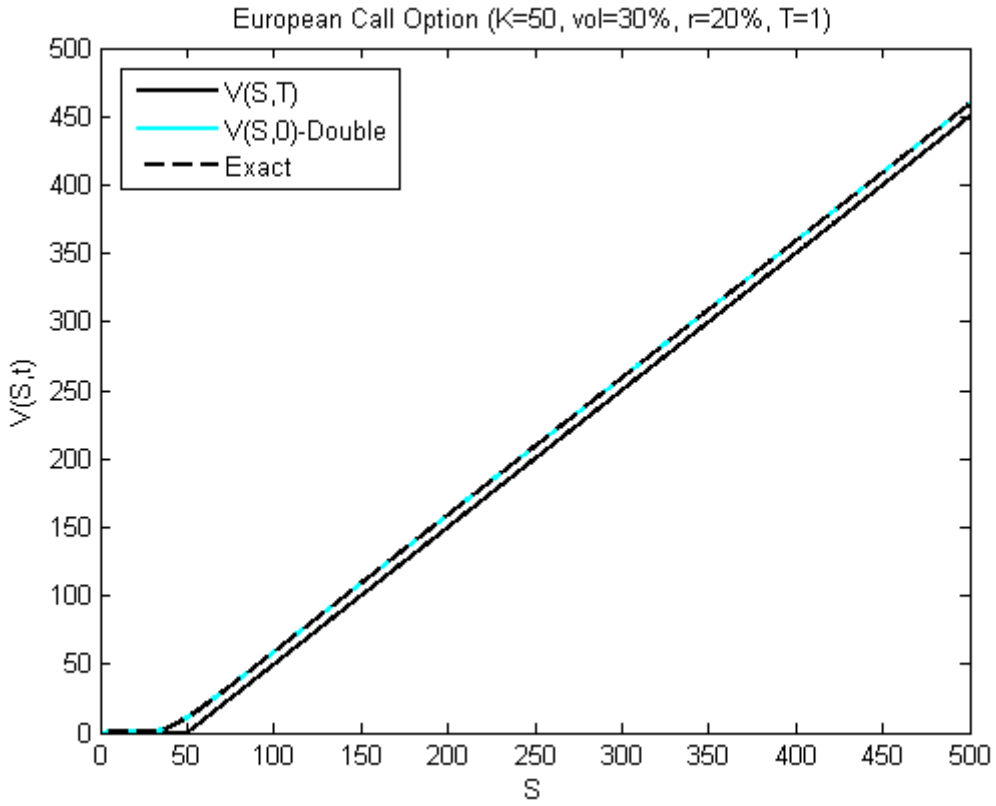



It can be observed that for a single precision the numerical solution goes far beyond analytical.

**Figure 39: Payoff functions for European call option with Smax = 500 and r = 20%.**




European Call Option (K=50, vol=30%, r=20%, T=1)
V(S,T)
V(S,0)-Double
Exact
V(S,t)
S
0
50
100
150
200
250
300
350
400
450
500


European Call Option (K=50, vol=30%, r=20%, T=1)
V(S,T)
V(S,0)-Double
Exact
V(S,t)
S
0
50
100
150
200
250
300
350
400
450
500

## 5.4 Arithmetic precision experiment with MPFR

In the previous experiments it was clearly shown that parameter selection is of great importance in order to produce a quality solution that can be used for a range of option values. Again, it turns out that the system is not working for single precision.

The analysis so far has clearly shown that the combination of Crank-Nicolson, norm-CR and single precision does not work. So, an arithmetic precision that produce acceptable results for the market must be found. Because resources are finite in FPGAs and also the performance deteriorates from single to double precision about 25% [43]. Thus, an experiment was design to determine floating point arithmetic precision that passes the market threshold and balance the tradeoffs accuracy with performance.

### 5.4.1 The MPFR Library

The Multiple Precision Floating-Point Reliably is a GNU portable C library for arbitrary-precision binary floating-point computation with correct rounding, based on GNU Multi-Precision Library. The computation is both efficient and has a well-defined semantics: the functions are completely specified on all the possible operands and the results do not depend on the platform. This is done by copying the ideas from the ANSI/IEEE-754 standard for fixed-precision floating-point arithmetic [66].

More precisely, its main features are:

- Support for special numbers: signed zeros (−0), infinities and not-a-number (a single NaN is currently supported).
- Each number has its own precision (in bits since MPFR uses radix 2). The floating-point results are correctly rounded to the precision of the target variable, in any of the four IEEE-754 rounding modes.
- Supported functions: MPFR implements all mathematical functions from C99 and other usual mathematical functions: the logarithm and exponential in natural base, base 2 and base 10, the log(1+x) and exp(x)−1 functions (log1p and expm1), the six trigonometric and hyperbolic functions and their inverses, the gamma, zeta and error functions, the arithmetic geometric mean, the power (xy) function. All those functions are correctly rounded over their complete range.

#### 5.4.1.1 *Implementation of Crank-Nicolson scheme with MPFR*

In order to make further experiments for option pricing accuracy of the proposed method, all the algorithms had to be implemented again with MPFR library. The code is parameterized to change precision. All functions were design in double precision except Forward Phase (FP), Backward Phase and Right Hand Side “e” Phase, which are calculated in custom precision. Below is the code of inner loop of RHS “e” Phase and shows how the operations work with MPFR.

```
mpfr_mul(d0_u, d0[l-1], u[l-1], MPFR_RNDN);//d0[l-1]*u[l-1]
mpfr_mul(d1_u, d1[l-1], u[l], MPFR_RNDN);//d1[l-1]*u[l]
mpfr_mul(d2_u, d2[l-1], u[l+1], MPFR_RNDN);//d2[l-1]*u[l+1];
mpfr_add(add1_du, d0_u, d1_u, MPFR_RNDN);//d0[j-1]*u[j-1] + d1[j-1]*u[j]
mpfr_add(e[l-1], add1_du, d2_u, MPFR_RNDN);//e[j-1] =(d0[j-1]*u[j-1] +d1[j-1]*u[j])+d2[j-1]*u[j+1];
```

### 5.4.2 Monte Carlo simulation with custom precision

Due to the unpredictable nature of solution convergence, solver accuracy cannot be measured accurately in a single run as in the analysis performed in section 5.3. To minimize the possible biases from individual runs we followed the work [42] made by Wayne Luk's team at Imperial College, funded by J.P. Morgan and FP7. An attempt has been made to compare all the different FPGA-based valuation models, not in a speed up terms, but to an absolute degree. The trade-off between precision and time was measured. The comparison methodology that has been proposed was a Monte Carlo simulation with random parameter generation for options. The performance of each model was measured with the root mean square error (RMSE).

In this work there is an effort to follow the same path. The differences with the aforementioned work are that: the arithmetic precision is taken into account using MPFR library, they measured RMSE with double precision. Also, the performance comparison is not presented with the time but with the spatial step of discretization Δs. The process that has been followed is:

1. Set the parameters for Option and precision. Here the choices for the experiment were: Smax = 200, S = [50, 100], K = S ± [0, 10], σ = [0.1, 0.3], r = [0.02, 0.2], T =1.0, arithmetic precision from s24e8 to s53e11. For each system size N the same procedure was followed with dt = 0.001.
2. Generate random values for the option parameters with variables boundaries as initialize in step 1.
3. Calculate the price with numerical scheme of Crank- Nicolson
4. Calculate the price for the same parameters with analytical form of Black-Scholes.
5. Calculate the RMSE with equation (5.5).
6. Do 2-5 for all precision and system sizes.

All functions in the experiment were implemented in MPFR. For each set of the selected parameters 1000 trials have been done. Also, the selection of dt = 0.001 denotes that it is 1000 time steps per option, so the solver workload was from 1.6K to 16,384K grid points for each experiment. The results are presented in **Table 13**.

**Table 13: RMSE of European call option for precision against Δs with dt = 0.001.**

| Δs | 32bit-Single | 40bit | 48bit -Extended | 64bit -Double |
|---|---|---|---|---|
| 11.76 | 6.93E-01 | 7.11E-01 | 6.20E-01 | 7.92E-01 |
| 6.06 | 4.84E-02 | 4.64E-02 | 4.01E-02 | 4.32E-02 |
| 3.08 | 2.68E-02 | 7.99E-03 | 6.55E-03 | 6.61E-03 |
| 1.55 | 3.89E-02 | 1.48E-03 | 1.41E-03 | 1.40E-03 |
| 0.78 | 5.13E-02 | 3.90E-04 | 3.67E-04 | 3.57E-04 |
| 0.39 | 1.74E-01 | 1.80E-04 | 1.10E-04 | 1.00E-04 |
| 0.20 | 1.00E+00 | 6.93E-04 | 3.05E-05 | 2.75E-05 |
| 0.10 | - | 3.76E-03 | 1.28E-05 | 1.01E-05 |
| 0.05 | - | - | 3.58E-05 | 4.71E-06 |
| 0.02 | - | - | 1.55E-04 | 2.25E-06 |
| 0.01 | - | - | - | 2.00E-06 |

**Figure 40: RMSE for precision against Δs with dt = 0.001.**

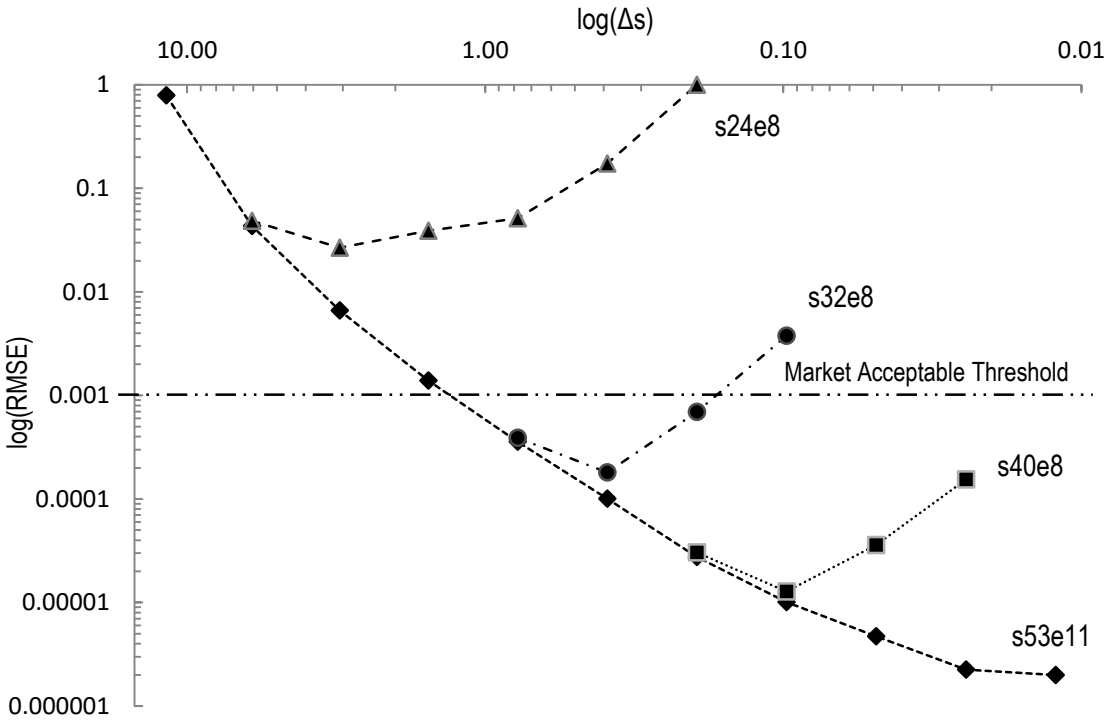



Figure 40 depicts RMSE against Δs (spatial time step) with dt = 0.001 for all the experiments. The market acceptable threshold is to produce an option pricing with less than $10^{-3}$ error. So, here the line for RMSE was drawn at 0.001. It must be noticed that RMSE penalizes errors that are away from the mean value, because of the squared error. This can be interpreted that the solver may have better results than those given by the graph but may failed to produce good enough results for some cases. Nevertheless, the above analysis can be taken as the working characteristics of our proposed algorithm.

To summarize, the Crank - Nicolson scheme with use of norm-CR can achieve market acceptable results with Δs < 1. The minimum arithmetic precision needed for achieving this accuracy is 48bits. With this precision good behavior of the algorithm is maintained in terms of accuracy.

In the next chapter, the 48bits arithmetic precision is going to be discussed along with hardware decisions that have been made and how it affects the performance of the FPGA option solver.

# Chapter 6:
# Hardware Architectures and Design Decisions

## 6.1 Introduction

As described in previous chapters, **normalized cyclic reduction algorithm** has been chosen for solving the tridiagonal system produced by Crank-Nicolson scheme. In this chapter, the hardware architectures and the design decisions are going to be presented.

First of all, the mathematical operations and the modeling of the algorithm will be explained. Secondly, the implemented architectures will be shown. The first architecture is simple, non-pipeline, the second one includes a three operand adder and the last, that is the proposed one, has fused multiply adder. Each design had been implemented in different periods of time with different technology of FPGAs, different precision of floating-point numbers and different versions of Xilinx tools.

## 6.2 Modeling the Crank-Nicolson scheme

The solution to the Crank-Nicolson Finite Difference Scheme is divided into two parts:

- In solving the tridiagonal linear system:

$$a_i * x_{i-1}^{n+1} + b_i * x_i^{n+1} + c_i * x_{i+1}^{n+1} = e_i^n$$

- And in the renewal of the right member “e” of the above equation through the relationship:

$$e_i^n = d_{0,i} * x_{i-1}^n + d_{1,i} * x_i^n + d_{2,i} * x_{i+1}^n$$

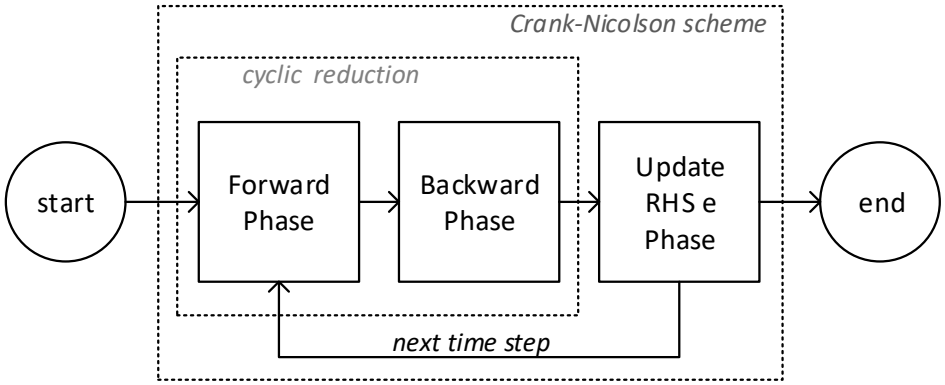
These two procedures must be performed for each time step n.

The solution of the tridiagonal linear system is done with a variant of the classic cyclic reduction algorithm, having the main diagonal $b_i$=1. The algorithm, in turn, is also divided into two phases:

- Forward Phase or forward elimination
- Backward Phase or backward substitution

The procedure for the numerical solution of the Black-Scholes equation is shown in the figure.

**Figure 41: Flow diagram of Crank-Nicolson scheme.**



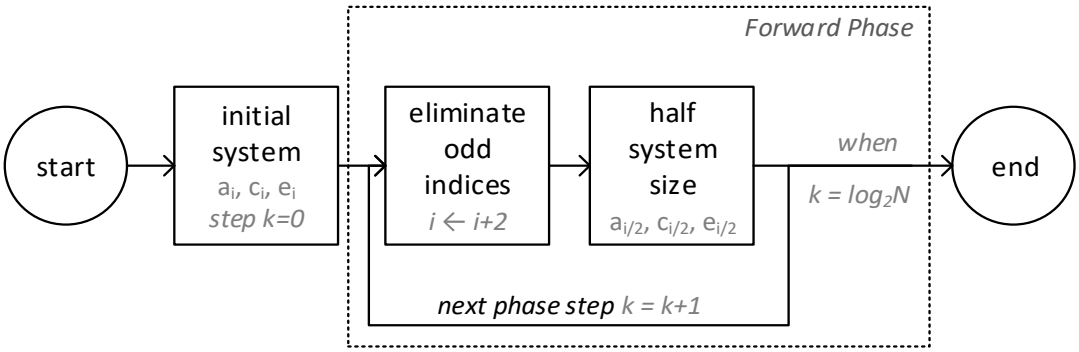

In the subsections below, the three phases are analyzed separately and their operations are modeled.

### 6.2.1 Forward Phase

In Forward Phase (FP) the steps of the algorithm are:

1. The elements $a_{0,i}$ , $c_{0,i}$ , $e_{0,i}$ are read for each i , from 0 to N-1 of an initial system size N, step k=0.
2. A new system of size N/2 is calculated, eliminating the even matrix rows through the algorithm operations, step k=1.
3. The elements $a_{k,i}$ , $c_{k,i}$ , $e_{k,i}$ of reduced system are read and a new system of size $N_k/2$ is calculated, step k=k+1.
4. Step 3 is being repeated until system becomes of size $N_k=1$, step $k=\log_2 N$. The output of FP is all the reduced systems.

**Figure 42: Flow diagram of Forward Phase.**

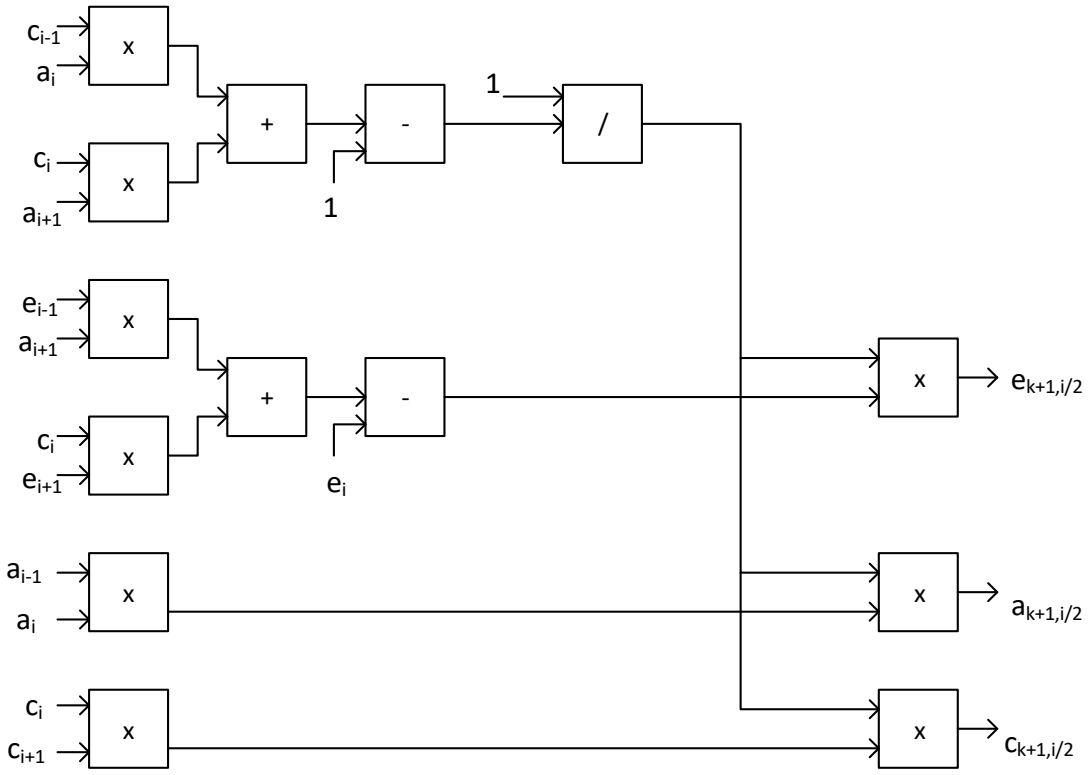


The operations executed for the process of reduction, as discussed above, for a loop iteration of the algorithm, are as follows:

$$temp = 1/(1 - c_{i-1} * a_i - c_i * a_{i+1})$$
$$a_{k+1,i/2} = -temp * a_{i-1} * a_i$$
$$b_{k+1,i/2} = 1$$
$$c_{k+1,i/2} = -temp * c_i * c_{i+1}$$
$$e_{k+1,i/2} = temp * (e_i - e_{i-1} * a_i - c_i * e_{i+1})$$

As shown in the figure, the sequence of operations, 2 simultaneous multiplications → addition → subtraction, is repeated twice, one for calculating the "temp" variable and one for calculating "$e_i$".

**Figure 43: Modeling the operations of Forward Phase.**



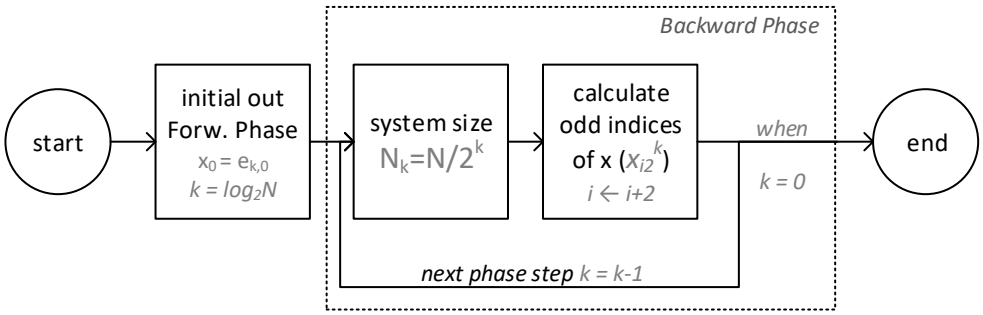

### 6.2.2 Backward Phase

Backward Phase (BP) starts when Forward Phase is over. In detail, the steps of this phase are as follows:

1. Element $x_0=e_{k,0}$ is calculated, initial step $k=\log_2 N$.

2. The odd elements $a_{k,i}$, $c_{k,i}$, $e_{k,i}$ and $x_0$ are read and $x_{N/2}$ is calculated, step $k=\log_2 N - 1$.
3. For each i, from 1 to $N_{k-1}$, new elements x are calculated, using variables x of the previous steps and the odd elements of this step $a_{k,i}$, $c_{k,i}$, and $e_{k,i}$.
4. Step 3 is being repeated until algorithm step becomes k=0, and all variables of array x of size N are calculated.

**Figure 44: Flow diagram of Backward Phase of Cyclic Reduction.**

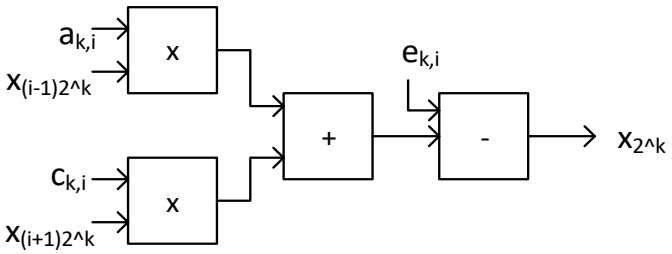



The operations executed for the process of substitution, as discussed above, for a loop iteration of the algorithm, are as follows:

$$x_{i2^k} = e_{k,i} - a_{k,i} * x_{(i-1)2^k} - c_{k,i} * x_{(i+1)2^k}$$

As shown in the figure, for each $x_{i2^k}$ element, 2 multiplications, 1 addition, 1 subtraction are needed, in the same order as the FP.

**Figure 45: Modeling the operations of Backward Phase.**



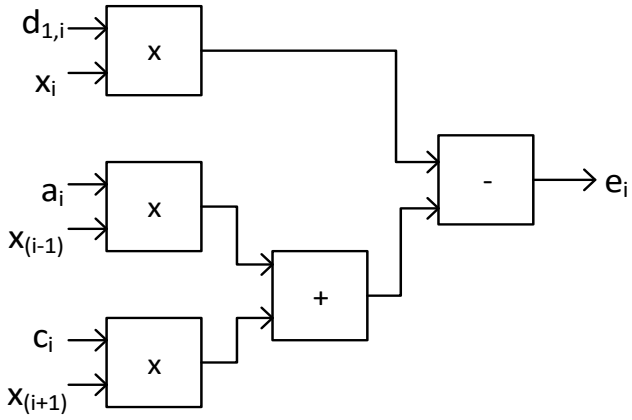

### 6.2.3 Update Right Hand Side "e" Phase

Update "e" Phase (UpeP) starts after the procedure of solving the tridiagonal linear system has finished. As soon as BP calculates all variables of x array, the right side of the system is renewed through the operations:

$$e_i = -a_i * x_{i-1} + d_{1,i} * x_i - c_i * x_{i+1}$$

As shown in the figure, for each $e_i$ element, 3 multiplications, 1 addition, 1 subtraction are needed, in the same order as the BP.

**Figure 46: Modeling the operations of Update RHS e Phase.**

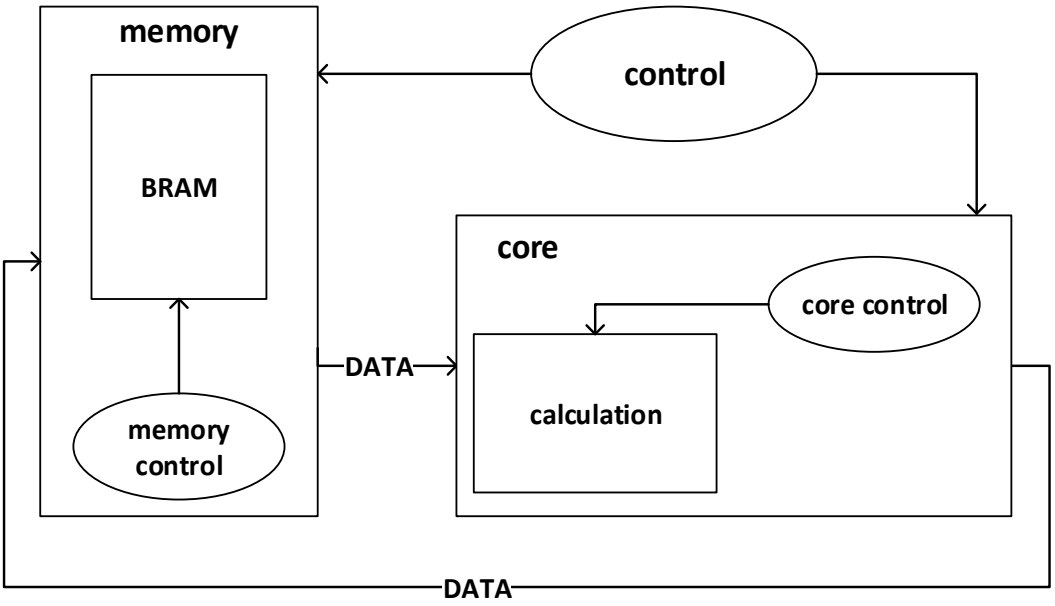



After UpeP is finished, arrays a, c and the updated e form a new tridiagonal linear system that needs to be solved, for the next time step m of Crank-Nicolson scheme.

## 6.3 The first two architecture designs

At the beginning of the first design, the first issue to be addressed was the data. From Crank-Nicolson scheme, there is a tridiagonal matrix of size N (a, c, e), an array of the same size (d) and the solution array (x) with floating-point numbers. It was decided that all these vectors would be stored on FPGA's BRAMs and there will be a basic core where the mathematical operations are done. As shown in Figure 47, the data are read from memories and go into the core. The results of the core's calculations will be stored back to the BRAMs. This procedure will be repeated M times, where M is the number of time steps chosen in Crank-Nicolson. New data overwrite the old ones where permitted. There is also two-level hierarchy in control units. At the bottom level, two different FSMs control the data storage and the core and at the top level there is one FSM to rule them all. This dataflow is kept in all three designs that will be presented in the next sections.

**Figure 47: Top block diagram.**




The second issue was the calculation unit and the arrangement of operations. From the analysis of previous sections, it is obvious that the forward phase is the most resources consuming phase and needs 14 floating-point operations, while backward phase needs only 4 and update “e” only 5. Also, there is a pattern in the sequence of operations in all three phases; simultaneous multiplications followed by addition and subtraction. As a result, the initial approach for the design was to follow exactly the modeling of operations in forward phase and the other two phases would be covered. However, if this approach had been utilized, the design would have required many floating-point operators and DSPs, which would have been used only in forward phase and have remained idle during the other two phases. In a design like this, it could have been difficult to add many cores and make it parallel. Therefore, the next goal was to reduce floating-point operators and reuse them wherever needed.

### 6.3.1 First architecture design

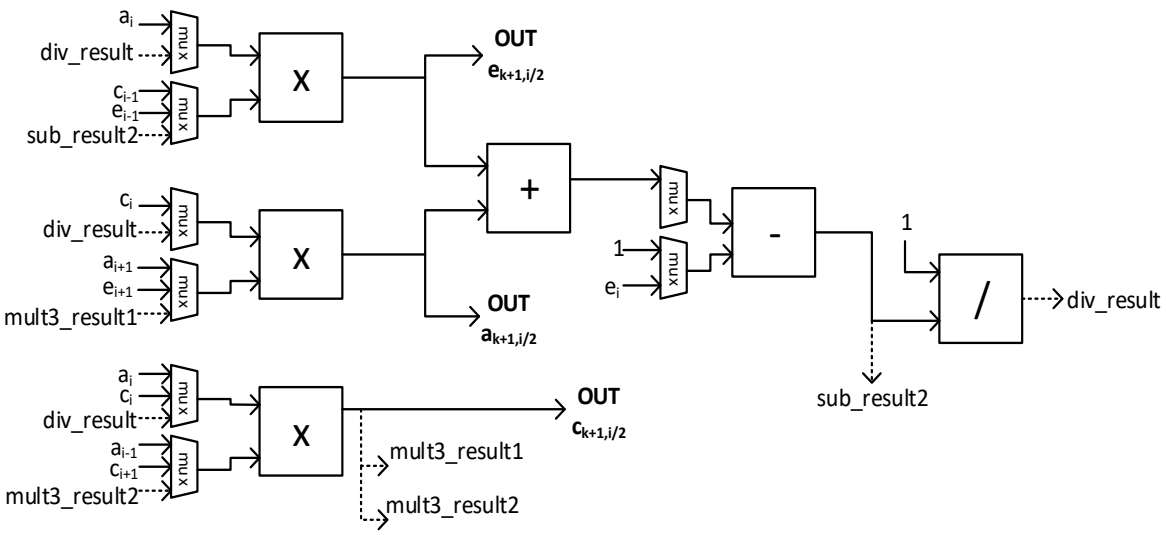
In the first architecture design, each element of the original tridiagonal system was stored in a single memory location. Variables a, c, and e are put into three separate BRAMs, at the positions 0 to N-1, where N is the system size. Every new (half-sized) system that is produced in each k phase step of FP is stored in the same memories as the initial. The first element of each “new” system is stored in the next position of the last item of the old system. For example, for phase step k=1, new elements are stored from position N and beyond. The elements of vectors x and d are stored in two BRAMs of size N, same size as the vectors’.

It was noted that variable *temp* has to be calculated first because it is necessary to export all three new *a, c* and *e*. Thus, the operations from modeling were reordered taking into account three things: 1) some of the FPOs must be used more than one time in FP in order to reduce the needed resources, 2) keep the basic sequence: two parallel multiplications followed by addition and subtraction, because it appears in all three phases, 3) the number of FPOs that stay idle in BP and UpeP must be the minimum possible.

#### 6.3.1.1 Forward Phase

Initially, in calculation unit were used 2 multipliers, 1 add unit, 1 sub unit and 1 divider. However, the large number of multiplications during FP, led to the addition of a third multiplier. An FPO per operation is used. The FPOs that have been used were generated with the Xilinx Floating-Point Operator v5.0.

**Figure 48: Forward Phase of first design.**




The architecture shown in Figure 48 is the block diagram for core's calculation unit. In the other two phases some of these FPOs stay idle. This happens in each architecture design. The calculation unit is explained separately at each phase, but the overall circuit is the same one with FP and the other two phases use the same resources.

The design in Figure 48 is not pipelined but has registers to keep values from intermediate results that need to be used for the final results. For simplicity, they are not shown in the scheme.

It must be noted, that this first and simple architecture was designed before the numerical analysis, that was presented on chapter 4, and the floating-point numbers are single-precision, 32 bits. The latency of Xilinx's FPOs, as well as the number of DPSs and LUTs used

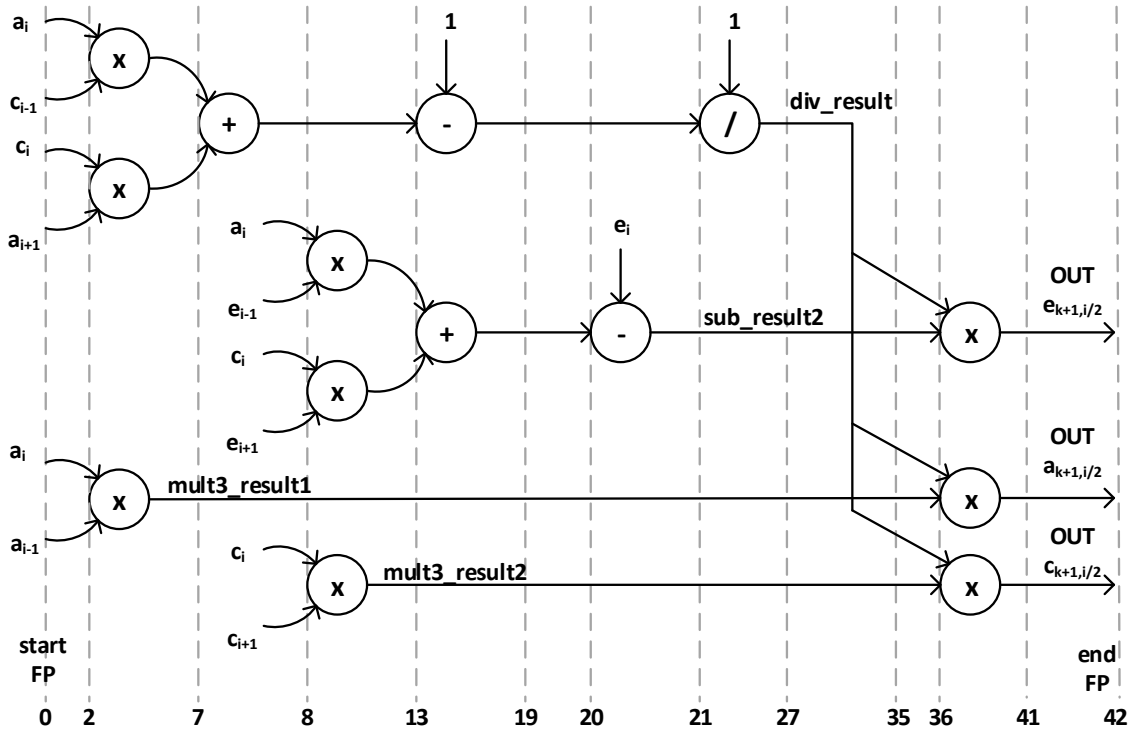
are different for different precisions. At the beginning of the design, the FPOs had the maximum latency values. After the system was completed and the clock period was estimated, the latency gradually decreased, a feature provided by Xilinx's FPO. After each decrease, Synthesis and Implementation were running and the clock value was checked. The combination of the smaller latency that did not increase the clock period more than 5 ns is that of the Table 14. The purpose of this method was to find the critical path inside an FPO, in order to make sure that no more optimization could have been done. This method was used at all the later designs, and from this point on, the number of cycles presented is the final.

**Table 14: Latency of FPOs for first architecture.**

| floating point operator | maximum latency | used latency |
|---|---|---|
| multiplier | 8 | 5 |
| adder | 12 | 6 |
| sub | 12 | 7 |
| divider | 28 | 14 |

The number of cycles to produce a result in Forward Phase, for single precision numbers, is 42 and is shown in Figure 49. Any extra cycle or register that was added in the datapath, had been done at the end of the design, taking into account the critical path and the final clock frequency. Figure 49 shows the sequence of FP operations in time, not the FPOs, and latency is measured in cycles.

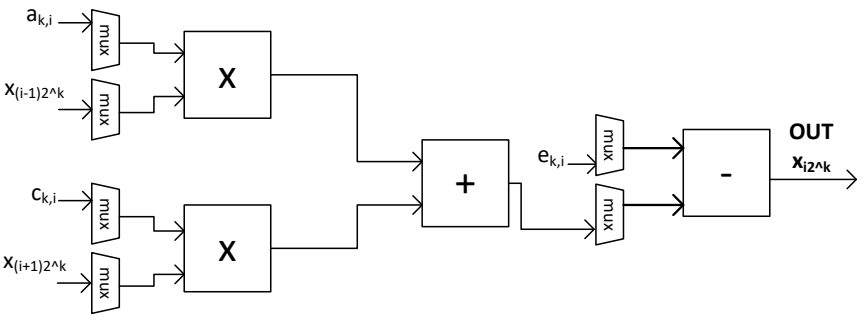
**Figure 49: Datapath with latency for First architecture of the Forward Phase.**




### 6.3.1.2 Backward Phase

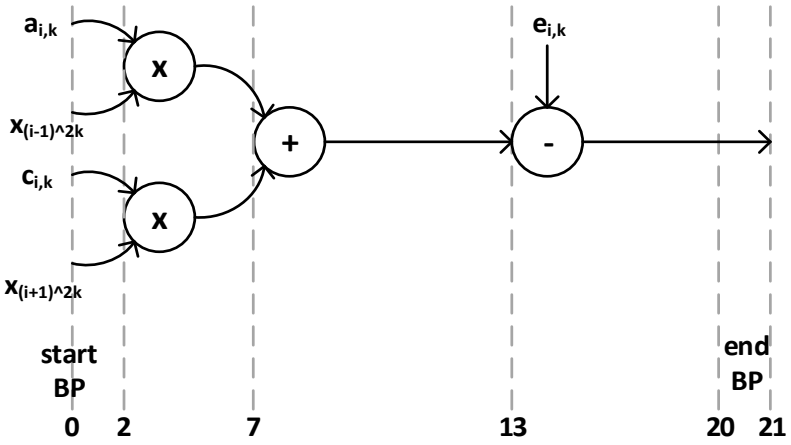
Backward Phase (BP) starts when FP is over. As shown in the figure TADE, in step $k = \log_2 N$, $e_0$ is equal to $x_0$. In each of the following steps, k = k-1, the variables x are used that have been calculated in the previous steps, and also the variables $a_{k,i}$, $c_{k,i}$, $e_{k,i}$, that have been calculated in FP of the same step. For Backward Phase, the order of operations is the same as for Forward. Therefore, no changes were made to the basic calculation unit; simply the variables were linked to the FPOs.

**Figure 50: First architecture of the Backward Phase.**



The number of cycles to produce a result in Backward Phase, for single precision numbers is 21 and is shown in Figure 51. The figure shows the sequence of BP operations in time, not the FPOs, and latency is measured in cycles.

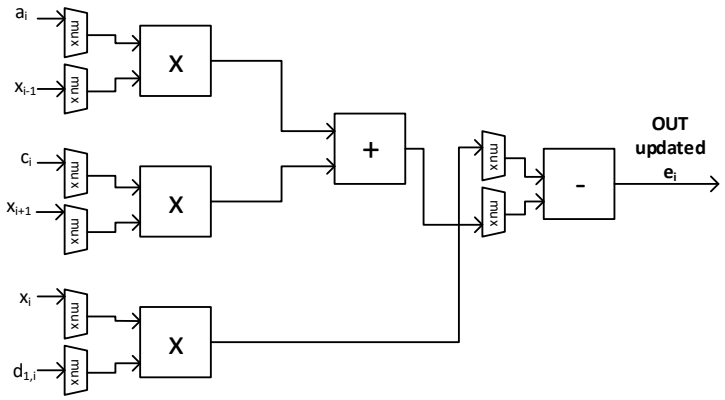
**Figure 51: Latency for First architecture of the Backward Phase.**




### 6.3.1.3 Update Right Hand Side "e" Phase

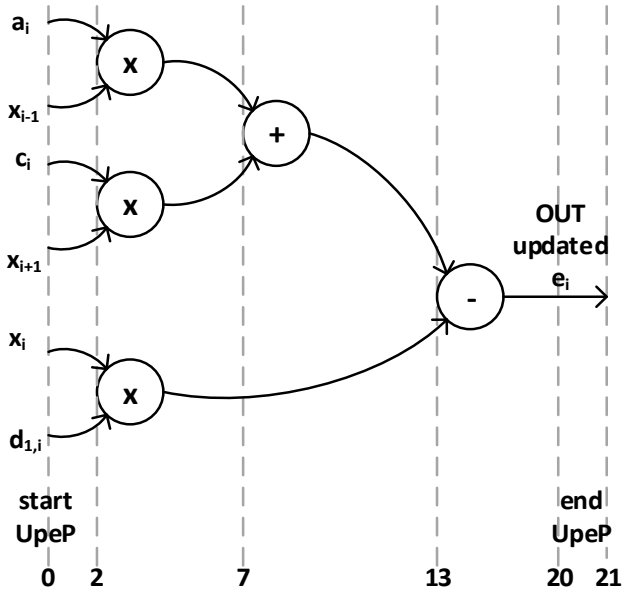
Update RHS "e" Phase (UpeP) starts after the end of Backward Phase. The elements *a* and *c* of the original system, the elements of vector *d* and the calculated *x* export the new *e*, which is stored in memory E, to positions 0 to N-1, overwriting the vector that had been initially stored there. For Update RHS "e" Phase, the order of operations is the same as BP and FP. Therefore, there was no need to add anything to the basic calculation unit.

**Figure 52: First architecture of the Update "e" Phase.**



The number of cycles to produce a result in UpeP, for single precision numbers is 21, same as BP and is shown in Figure 53.

**Figure 53: Latency for First architecture of the Update "e" Phase.**

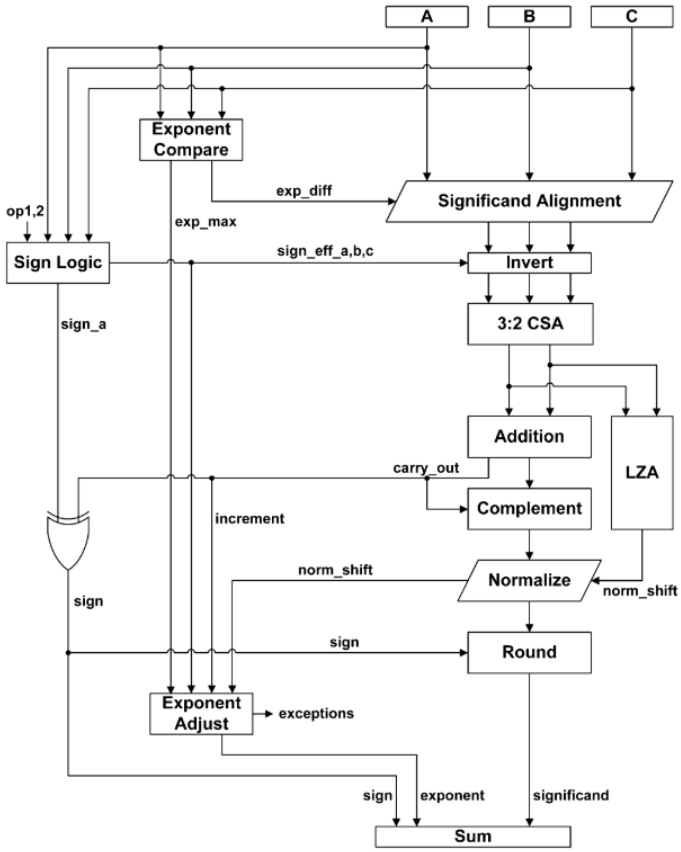



### 6.3.2 Second architecture design with custom 3-operand adder

In the second architecture design, the first optimization was to make the datapath pipelined. The designs that will be presented, from now on, are pipelined. Three multipliers were added, increasing the total number of multipliers to 6, from 3. These multipliers were used only in FP. Because of the sequence of the operations, with the addition of 3 multipliers, the calculation unit receives new input numbers every two cycles in FP. In order to make the unit receive new input in every cycle, there were needed 9 multipliers total, an option that would increase the resources without much to gain. In BP and UpeP, the pipeline rate is one cycle.
Subsequently, the next optimization was about FPOs. The idea to create custom floating-point operators that make the same plain operations as the ones that were already used was abandoned quickly, as pointless. There was an attempt to create a custom multiplier but it was hard to make it better, faster, and with fewer resources than Xilinx's ip core. The idea to find new floating-point operators that make complex and fused operations was attractive and it was further explored.

The first FPO that was tried was a 3-operand adder, based on [82], [83] and [84]. The 3 operand adder that was created had latency 8 cycles, compared to the adder-sub units that had total latency 13 cycles which was a decrease. In Figure 54 block diagram of operator's design is shown.

**Figure 54: Three operand adder (Source: [84]).**




Clock frequency of total design did not increase; the critical path was still inside Xilinx's FPO. With the usage of 3 operand adder, in place of an adder and a sub unit, the total latency of each phase reduced to 35 cycles (from 42) for FP and to 13 cycles (from 21) for BP and UpeP. These numbers were calculated for single precision floating-point adders. The decrease of latency was sufficient in BP and UpeP but not in FP.

The following figures show the block diagram and latency of each phase.

### 6.3.2.1 Forward Phase

**Figure 55: Architecture of the FP with 3-operand adder.**

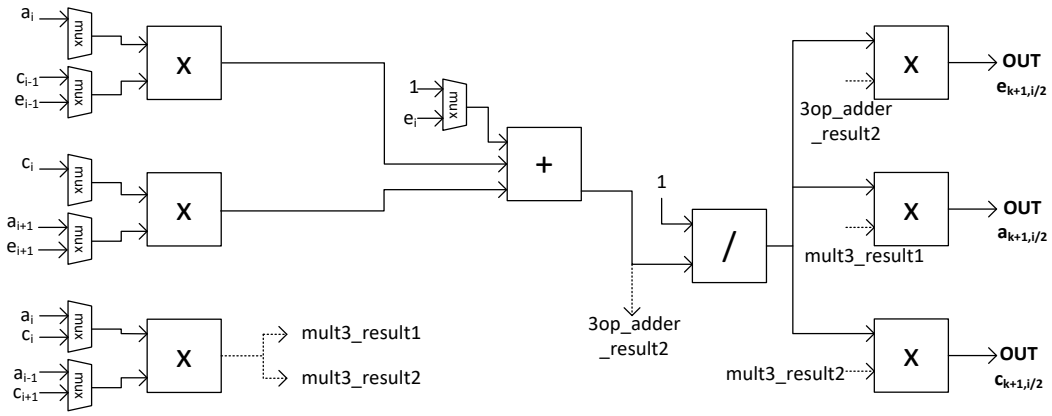
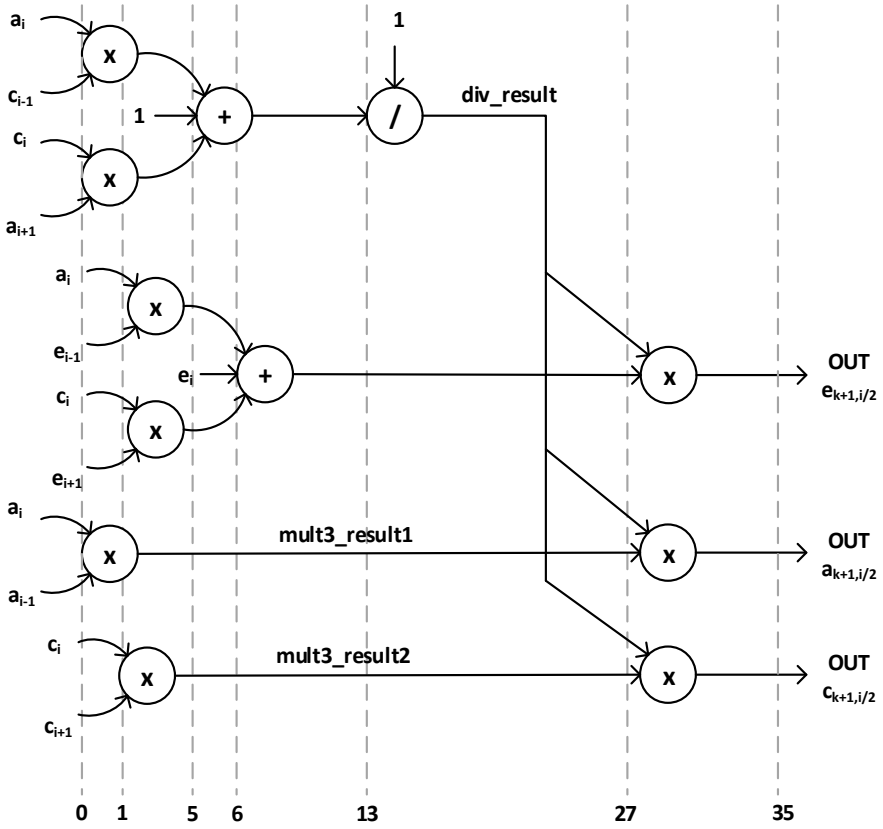



**Figure 56: Latency of the FP with 3-operand adder.**



### 6.3.2.2 Backward Phase

**Figure 57: Architecture of the BP with 3-operand adder.**

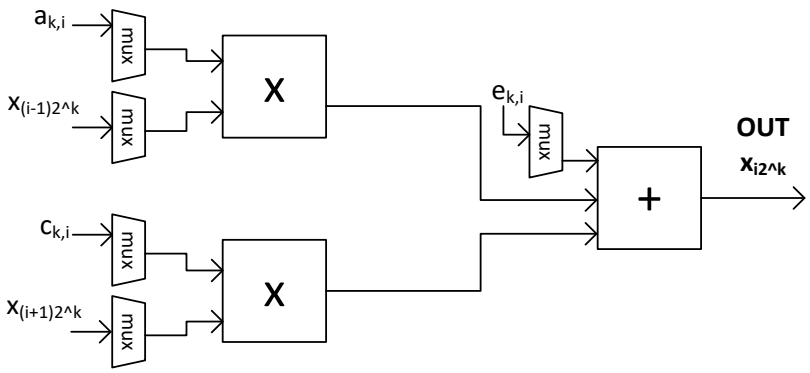
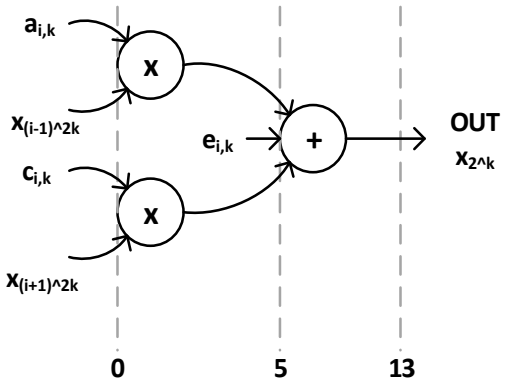



**Figure 58: Latency of the BP with 3-operand adder.**



### 6.3.2.3 Update Right Hand Side “e” Phase

**Figure 59: Architecture of the UpeP with 3-operand adder.**

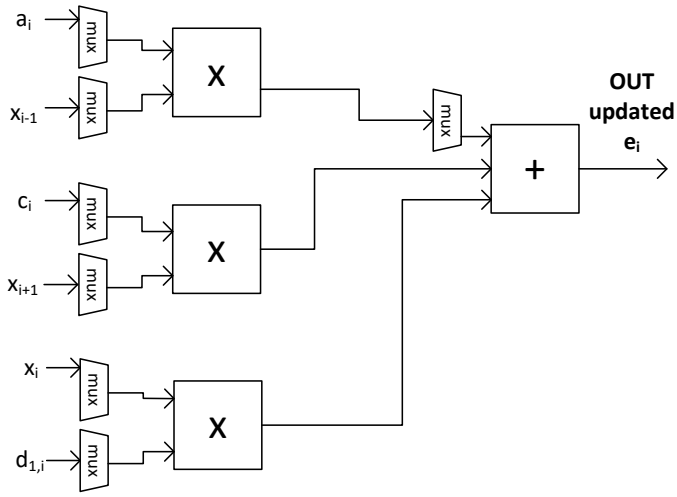
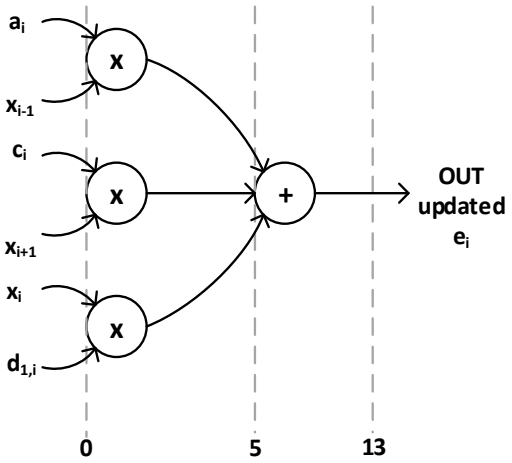



**Figure 60: Latency of the UpeP with 3-operand adder.**



## 6.4 The proposed architecture with fused multiply-adder operator

The two previous designs were implemented in single precision arithmetic, before the error analysis, shown in Chapter 4. This analysis has led to the need of increasing the arithmetic precision in order to achieve valuable results. Thus, the proposed architecture must include custom precision arithmetic, with the final goal being the achievement of better or similar performance to the previous designs. In the next subsections, design decisions are going to be discussed.

### 6.4.1 Modeling design with FMA

The next FPO that was studied, and it showed to fit better with the sequence of operations, was the fused multiply-add unit. Xilinx, on floating-point ip core version 7.1 has incorporated this operation. Modeling of operations for all three phases is shown in Figure 61, Figure 62, and Figure 63.

**Figure 61: Modeling the FP with fused multiply adder operand.**

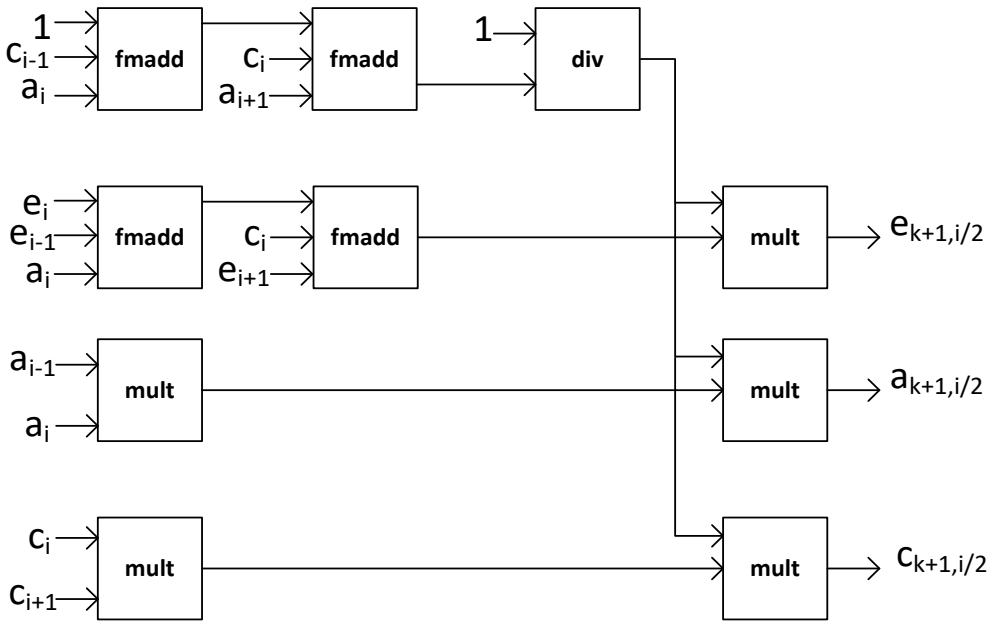


**Figure 62: Modeling the BP with fused multiply adder operand.**

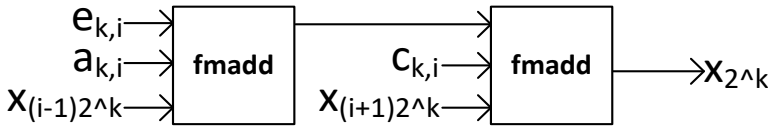
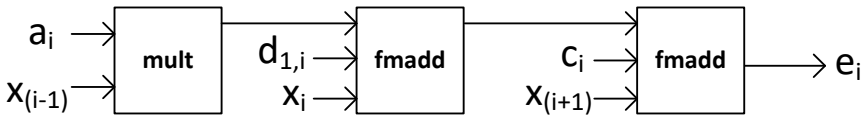



**Figure 63: Modeling the UpeP with fused multiply adder operand.**




It is obvious from the figures that the number of operations is reduced for all phases. In Table 15 the number of operators that are needed is presented, for every modeling that has been made, for all phases.

**Table 15: Number of operations per architecture.**

| modeling \ phase | FP #operations | BP #operations | UpeP #operations |
|---|---|---|---|
| first-simple | 14 | 4 | 5 |
| with 3-operand adder | 12 | 3 | 4 |
| with fmadd | 10 | 2 | 3 |

It had to be examined whether the latency was also reduced. Xilinx's floating-point operator v.7.1 offers many options regarding the arithmetic precision, the number of DSPs to be used and the latency of the core. In Table 16, the number of DSPs and the maximum latency are presented, for single precision and for the custom precision that has been chosen.

**Table 16: Number of DSPs and Cycles per operator per precision.**

| floating point operator v7.1 | precision | DSP48E1 | maximum latency |
|---|---|---|---|
| fused multiply-add | 32 bits | 2 | 16 cycles |
| fused multiply-add | 32 bits | 4 | 19 cycles |
| fused multiply-add | 48 bits | 5 | 21 cycles |
| fused multiply-add | 48 bits | 8 | 24 cycles |
| fused multiply-add | 64 bits | 10 | 26 cycles |
| fused multiply-add | 64 bits | 13 | 29 cycles |
| multiplier | 32 bits | 0 | 8 cycles |
| multiplier | 32 bits | 1 | 8 cycles |
| multiplier | 32 bits | 2 | 8 cycles |
| multiplier | 32 bits | 3 | 6 cycles |
| multiplier | 48 bits | 0 | 8 cycles |
| multiplier | 48 bits | 6 | 12 cycles |
| multiplier | 48 bits | 7 | 13 cycles |
| multiplier | 64 bits | 0 | 9 cycles |
| multiplier | 64 bits | 6 | 12 cycles |
| multiplier | 64 bits | 7 | 12 cycles |
| multiplier | 64 bits | 8 | 13 cycles |
| divider | 32 bits | 0 | 28 cycles |
| divider | 48 bits | 0 | 44 cycles |
| divider | 64 bits | 0 | 57 cycles |

For each precision, single and the custom that has been chosen, every possible combination of values among the two basic characteristics (DSPs and latency) was tried.

- The primary goal was to determine the critical path inside an FPO and to keep the clock frequency above 250MHz.

- A secondary goal was the usage of DSPs to be the minimum possible, which would allow fitting the design in various FPGA devices, small or large. It turns out that to achieve replication of the basic calculation unit at least 32 times (i.e. have a design with 32 cores) in small devices, the number of DSPs must be examined with caution. After exhausting trials, for 48 bits precision, the values that have been chosen are shown in Table 17.

**Table 17: Used DSPs and latency per operator.**

| floating point operator v7.1 | precision | DSP48E1 | Used latency |
|---|---|---|---|
| fused multiply-add | 48 bits | 5 | 8 cycles |
| multiplier | 48 bits | 6 | 5 cycles |
| divider | 48 bits | 0 | 22 cycles |

### 6.4.2 Proposed architecture with FMA

Here, the hardware architectures are presented for each phase in figures Figure 64, Figure 65 and Figure 66. At this point, it was left to decide how many ip cores could be replicated on FPGA board, taking into account the aforementioned designs' goals. Therefore, 2 fmadd, 3 multipliers and 1 divider were used. For simplicity, in all the figures, the pipeline registers are omitted.

**Figure 64: Proposed Architecture of the FP with fused multiply adder operand.**

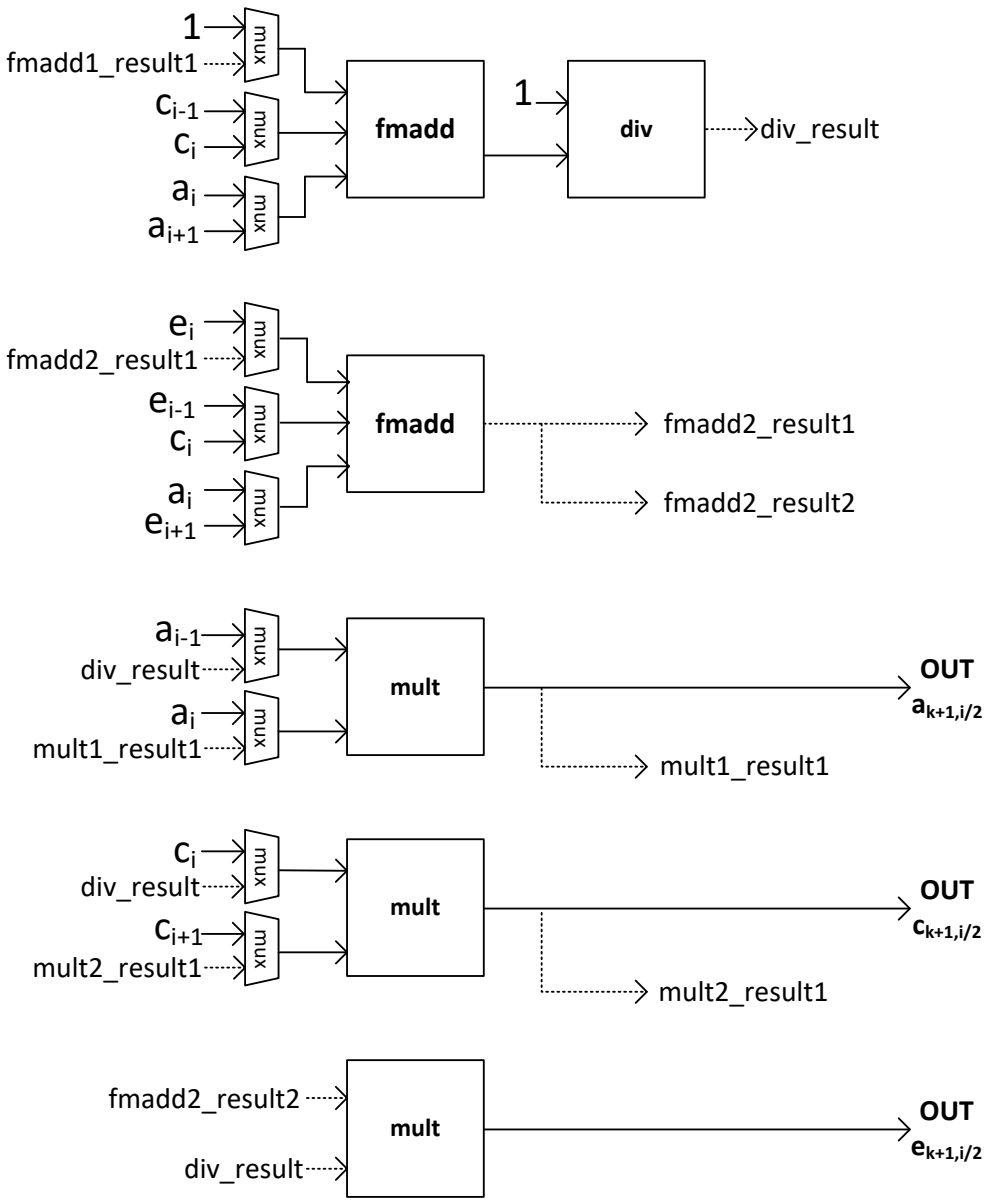


**Figure 65: Proposed Architecture of the BP with fused multiply adder operand.**

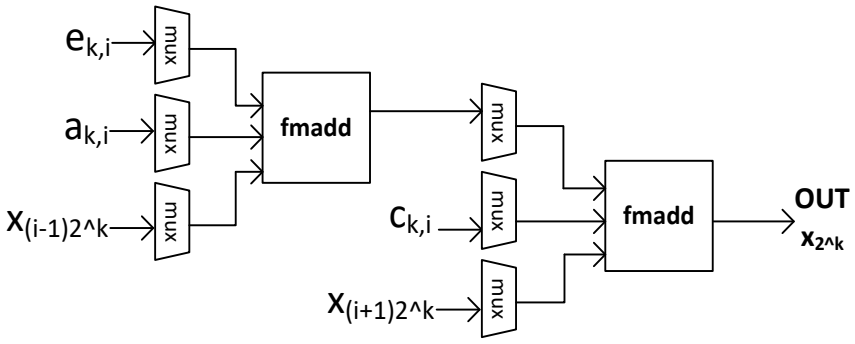
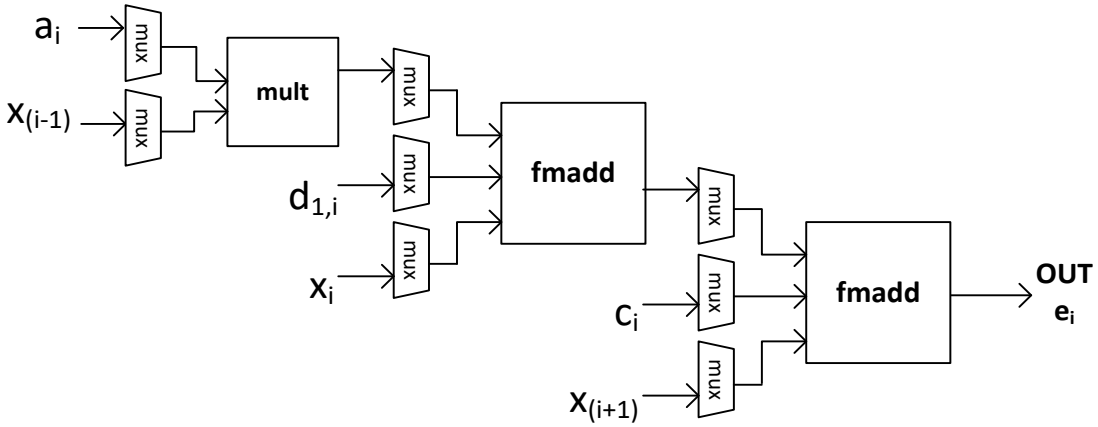



**Figure 66: Proposed Architecture of the UpeP with fused multiply adder operand.**



### 6.4.3 Latency of the proposed architecture with FMA per CR phase

In figures Figure 67, Figure 68 and Figure 69 is shown the number of cycles that are needed to produce a result in each phase. For FP this number is 44 cycles, for BP is 16 and for UpeP is 21.

**Figure 67: Latency of proposed architecture of FP with FMA.**

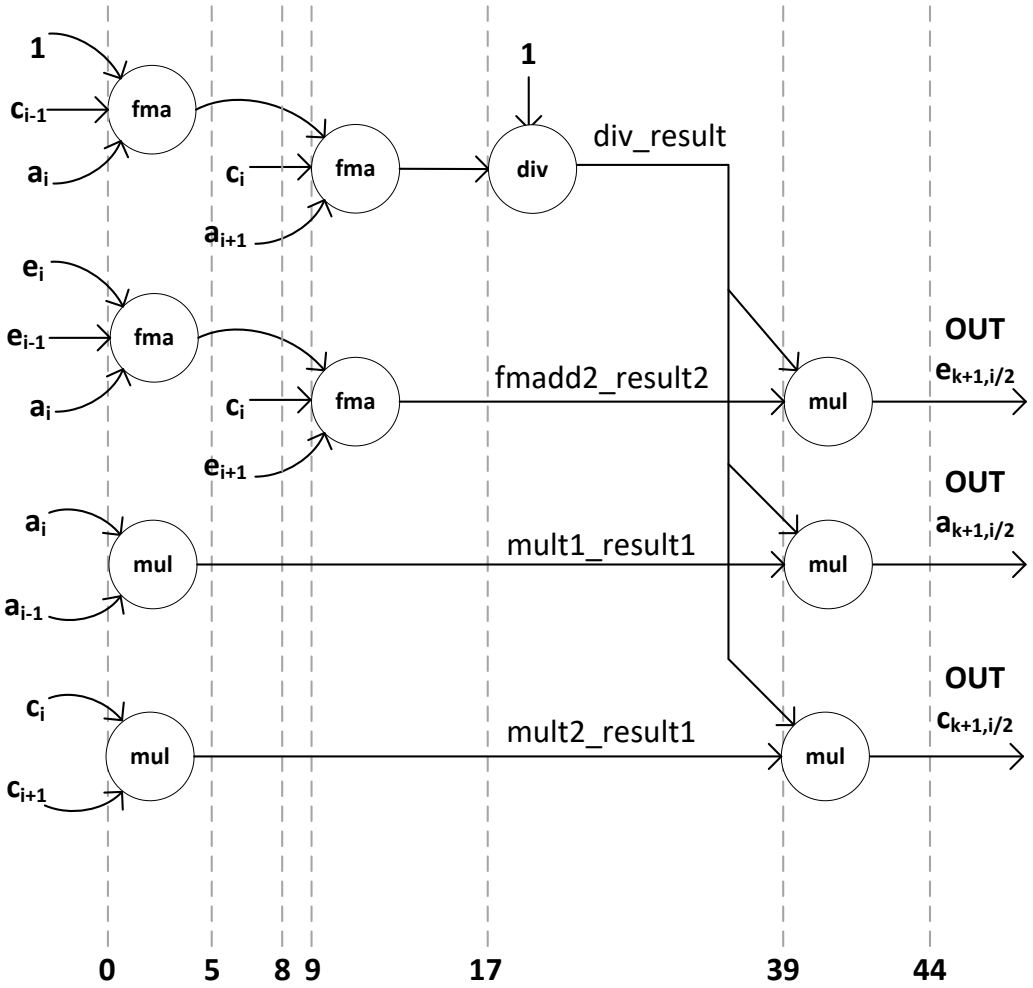


Figure 68: Latency of proposed architecture of BP with FMA.

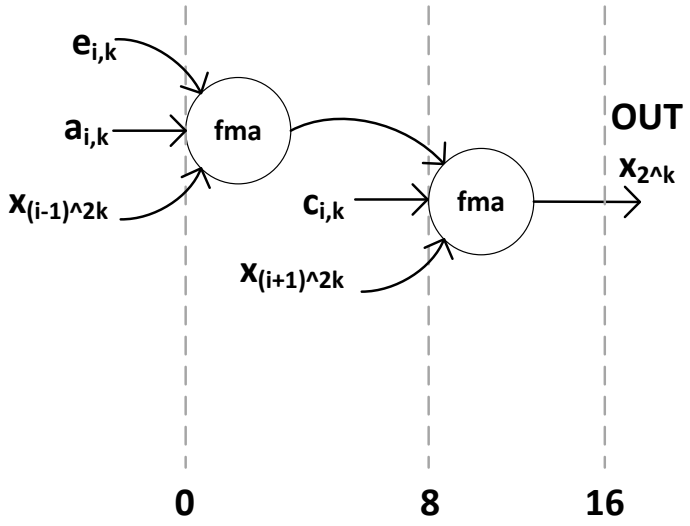
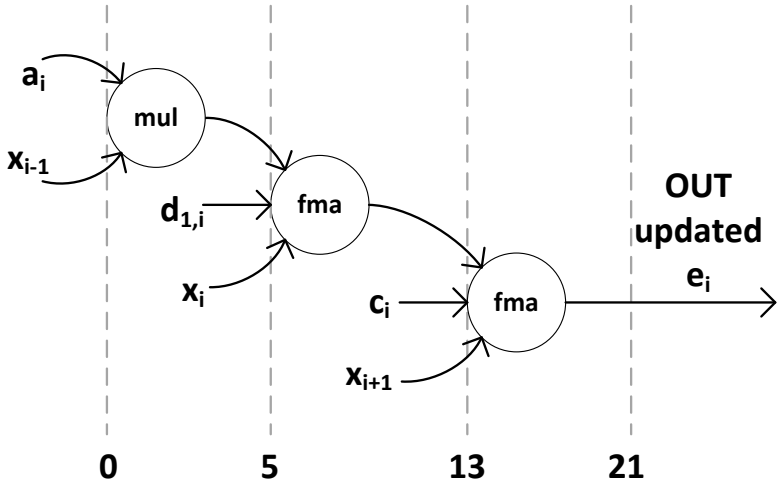



Figure 69: Latency of proposed architecture of UpeP with FMA.

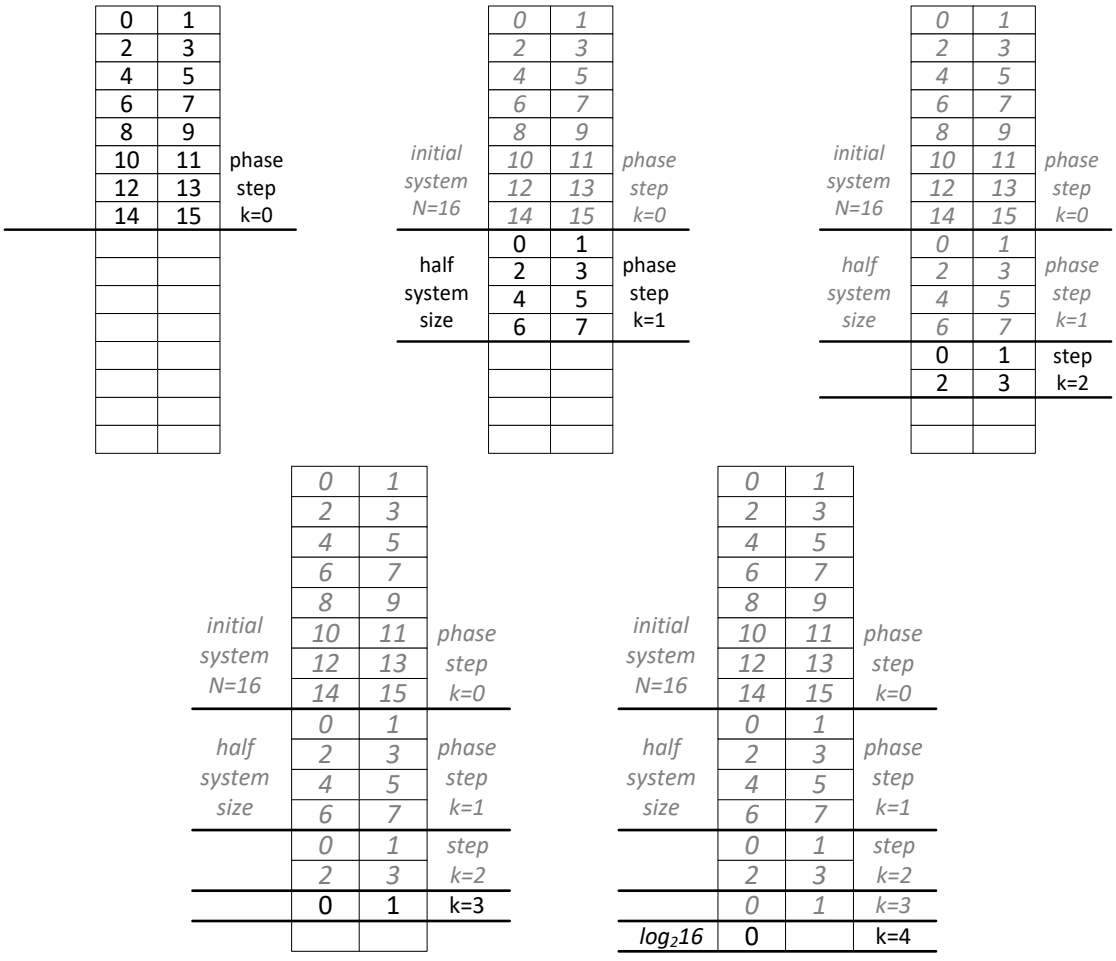


### 6.4.4 Data Storage

A challenging issue of Cyclic Reduction is the data storage; the manipulation of the memory scheme can make the difference in terms of performance [71], [74]. In this section memory optimizations are discussed for the proposed architecture.

Memory reading is done in series at each cycle. In previous designs, on element was stored in a memory position. In order to reduce one cycle at the beginning of every FP phase step, it would be better to store two elements in a memory position, so two elements would be read in one cycle. Figure 70 shows how elements of *a*, *c*, *e* vectors of size 16 are being written in the BRAMs, at each phase step k of FP.

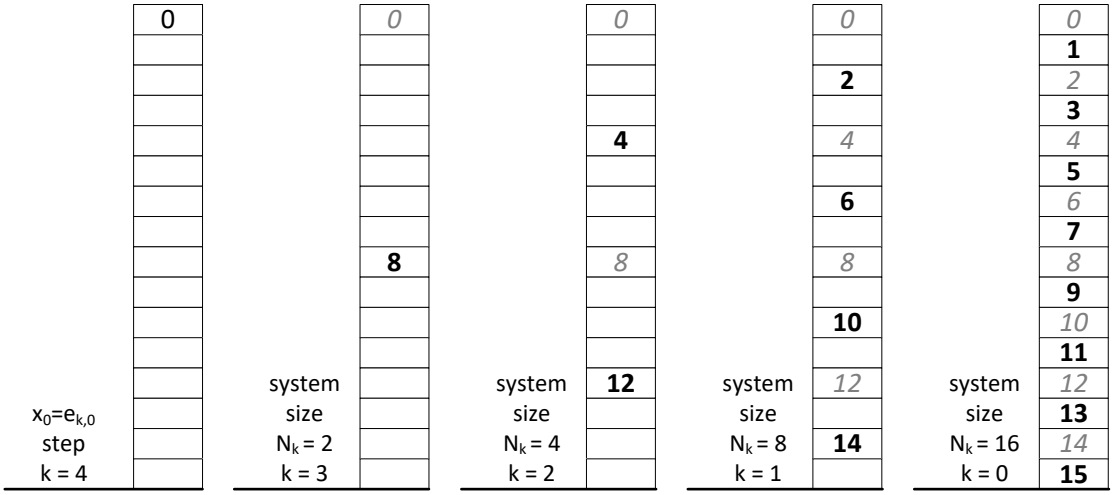
**Figure 70: Shared memory for FP.**




The elements of vector x are stored in a BRAM of size N, same size as the system. Figure 71 shows how elements of vector *x*, of size 16, are written in the BRAMs, at each phase step of BP. As it can be seen, reading and writing the variables x is not done in series. Thus, complex

counters and a separate control unit (see Figure 47) have been implemented to help the pointers get the right address.

**Figure 71: Shared memory for BP.**




For UpeP, a new vector *d* is used, only for reading, which is stored in a separate BRAM, mem D. The updated e variables that are calculated in this phase are stored in mem E, overwriting the old system.

After the design had been completed for 1 core, the goal was to replicate the basic core to achieve a multicore architecture to calculate large tridiagonal systems faster. The number of units that needed to be added is based on the FPGA that is being used and its resources.

An initial approach, which was implemented from early in the first hardware architecture, was that all cores use one large shared memory. The disadvantage of this design was the storage of data in BRAMs and the interconnection of BRAMs with the units. Using one memory for all units and in series reading and writing data from and to them was time-consuming. The solution was to store the vectors in more than one memories, specifically each core should have its own memory.

In a later version of this design, the coefficients of the tridiagonal system to be solved were separated in many memories. An example is shown in Figure 72. The initial system is of size 32 and the number of cores is four, hence each memory has a sub-system of size 8. In this way, each memory is attached to one core and at the same time each core has its own tridiagonal system which is reduced in half and stored back to its own memory at the next positions.

The basic idea was that all the pairs memory-core would work just like the nodes from cyclic-odd even reduction. With this way a core can get data from both its memory and other certain memories (not from all of them), but the results it exports, go only to its own memory.

In the example of Figure 72, until phase step k=3 all cores take inputs and produce results that are being written to their own memories. On step k=4 only cores 0 and 2 are working while on the last step only core 0 calculates the last value. The cores are working in a way that simulates the cyclic reduction algorithm. The same method is applied to BP too, as it can been seen in Figure 73.

**Figure 72: Distributed memory per Core for FP.**

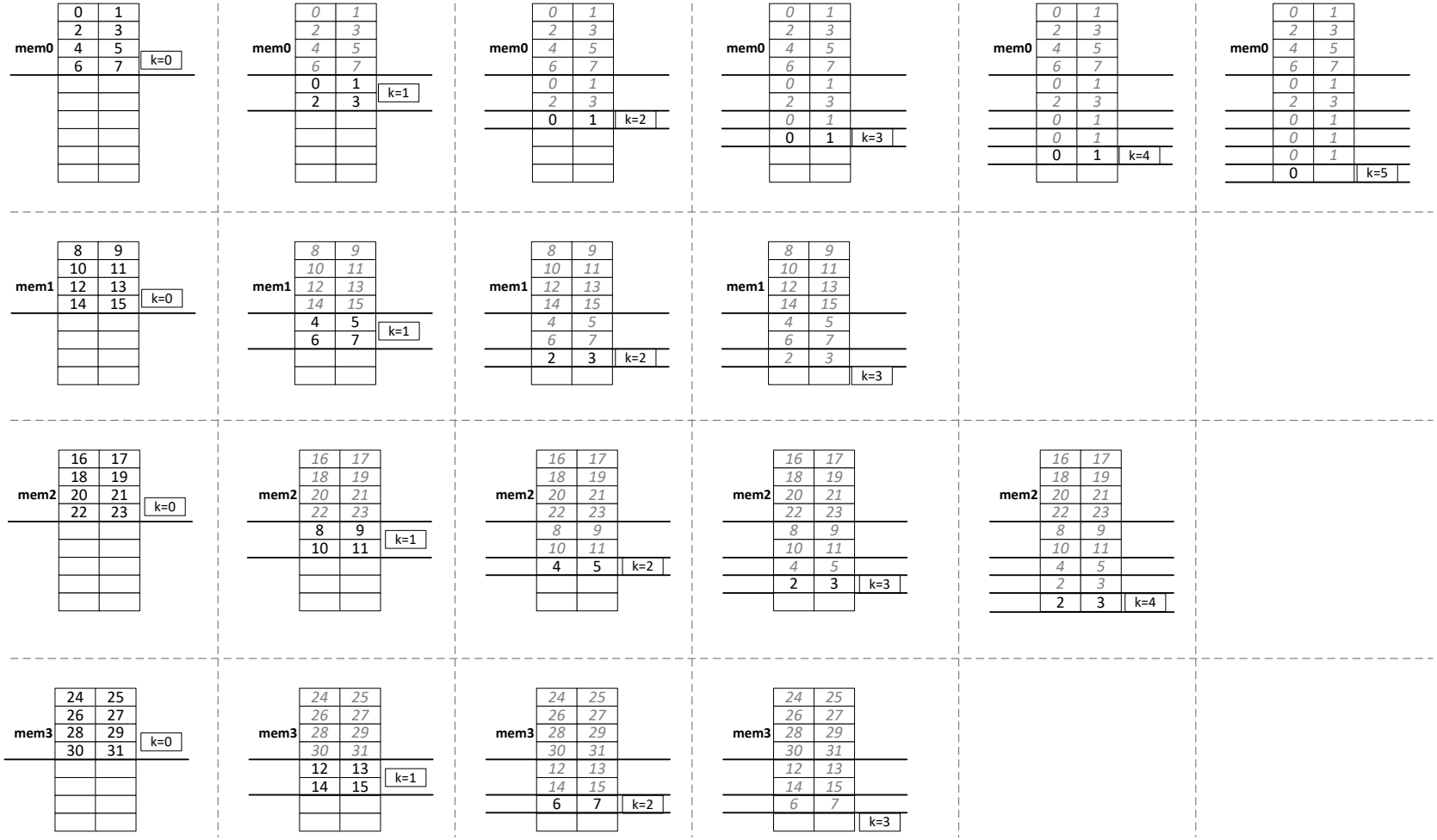
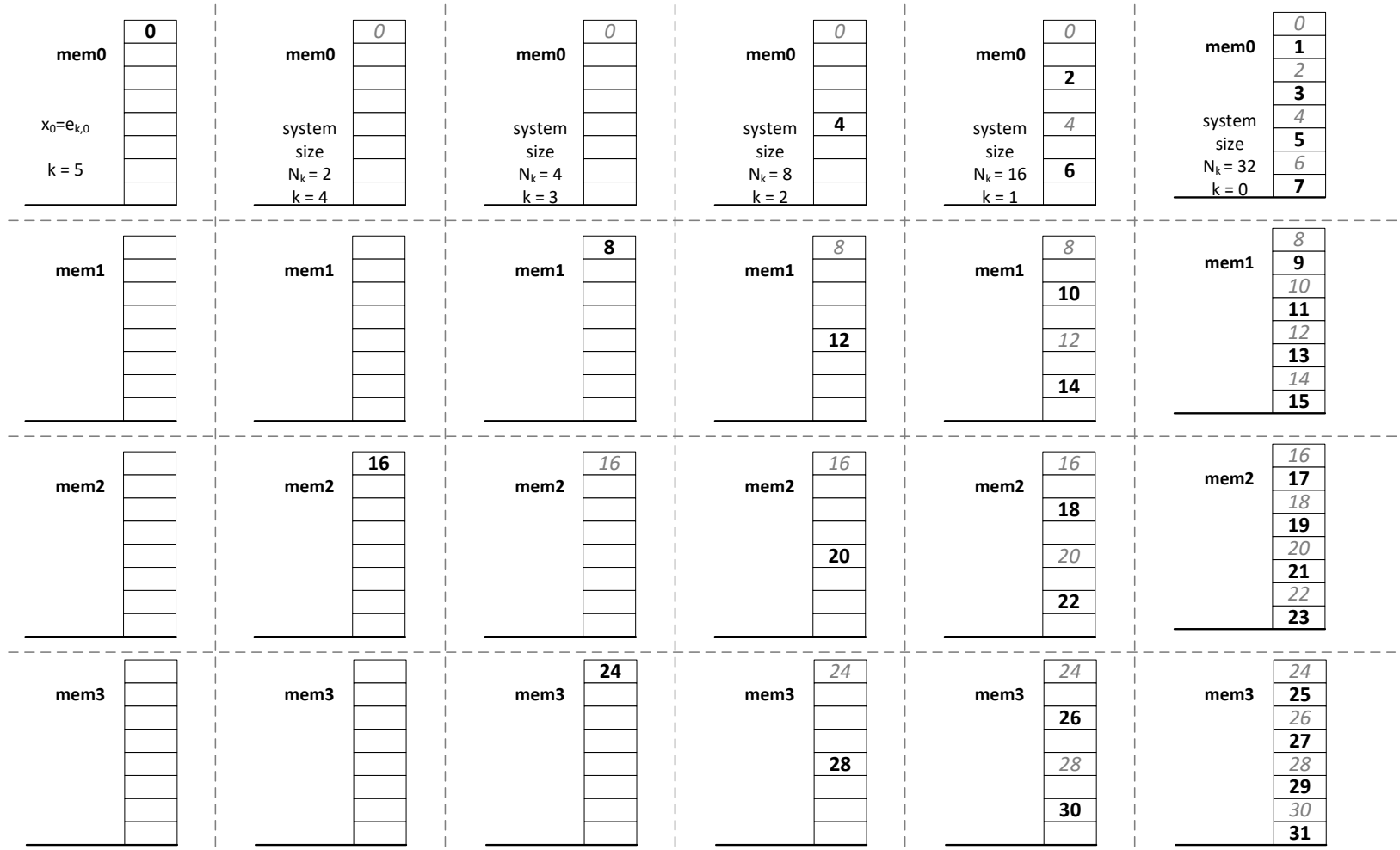


**Figure 73: Distributed memory per Core for BP.**



## 6.5 Summary and results

In previous sections, three different hardware architectures were presented and designs' decisions were thoroughly discussed. Here, implementation details are going to be presented with their results. It must be reminded that first (naïve) and second (3-opadder) designs were implemented in 32 bit single floating point arithmetic on different Xilinx FPGAs boards, xc5vlx330T with clock frequency 210.421MHz and xc6vsx475T with different with clock frequency 250.247MHz respectively. The proposed design (FMA) was implemented with 48bits custom precision on Ultrascale xcvu9p with clock frequency 263MHZ. Resources utilization is summarized in the next Table 18.

**Table 18: Utilization Summary for all Designs**

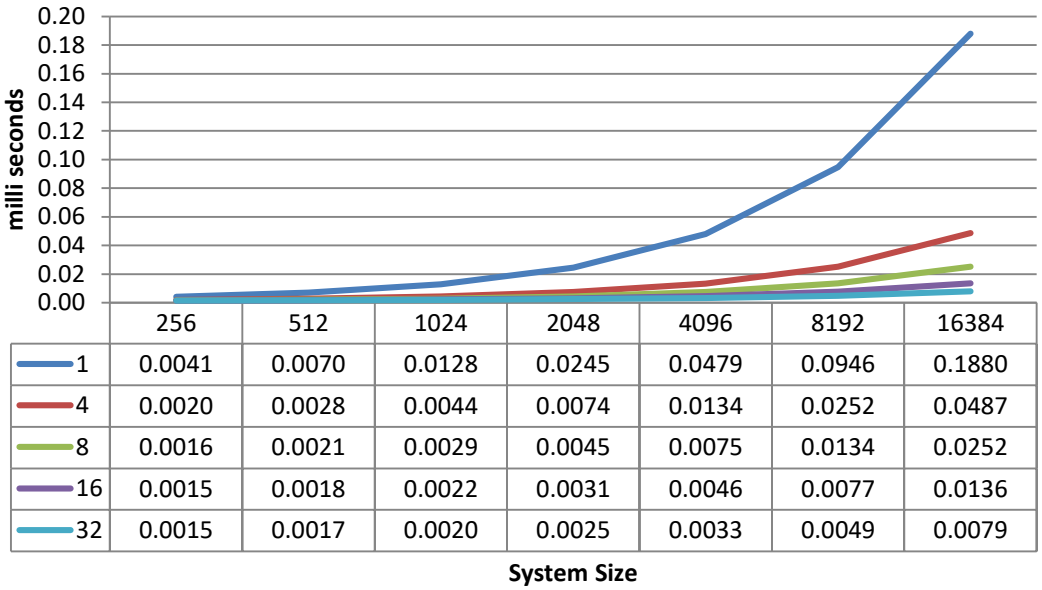
| Device Utilization Summary - XC5VLX330T | | | | | |
|---|---|---|---|---|---|
| *32-bit precision* | | Number of cores | | | |
| *non-pipeline* | | 1 | | 4 | |
| Resource | Available | Used | Util. | Used | Util. |
| LUTs | 207360 | 3442 | 1,66% | 13985 | 6,74 |
| Regs | 207360 | 5025 | 2,43% | 12749 | 6,18 |
| BRAM | 288 | 58 | 20,13% | 80 | 27,77% |
| DSPs | 192 | 3 | 1,57% | 12 | 6,25% |

| Device Utilization Summary - XC6VSX475T | | | | | |
|---|---|---|---|---|---|
| *32-bit precision* | | Number of cores | | | |
| *pipeline* | *3op adder* | 1 | | 4 | |
| Resource | Available | Used | Util. | Used | Util. |
| LUTs | 297600 | 8656 | 2.91% | 39817 | 13.38% |
| Regs | 595200 | 5294 | 0.89% | 26470 | 4.45% |
| BRAM | 1064 | 61 | 5.73% | 72 | 6.77% |
| DSPs | 2016 | 6 | 0.30% | 24 | 1.19% |

| Device Utilization Summary - XCVU9P | | | | | |
|---|---|---|---|---|---|
| *48-bit precision* | | Number of cores | | | |
| *pipeline* | *fmadd* | 1 | | 4 | |
| Resource | Available | Used | Util. | Used | Util. |
| LUTs | 2364480 | 7135 | 0.30% | 32821 | 1.38% |
| Regs | 1182240 | 6662 | 0.56% | 33310 | 2.81% |
| BRAM | 2160 | 46 | 2.11% | 52 | 2.41% |
| DSPs | 6840 | 25 | 0.37% | 100 | 1.46% |

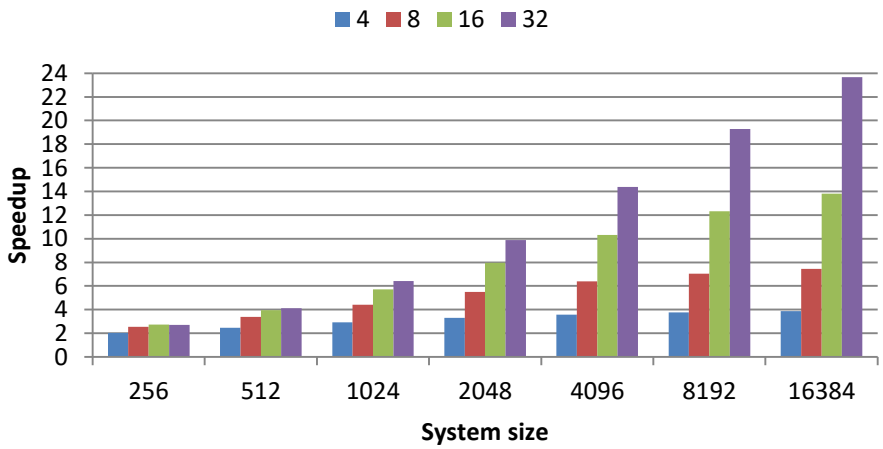
The rest of the performance analysis is presented only for the proposed architecture with the FMA operator, as the other two don't provide usable results for option pricing the comparison in absolute numbers cannot be valid. For comparability reasons with other or future works the results are presented separately for Crank-Nicolson and Cyclic Reduction. For the normalized Cyclic Reduction the timings results are in milliseconds and are showed in Figure 74. Here, results are for solving one tridiagonal system per system size.

**Figure 74: Timings (milliseconds) for Cyclic Reduction.**




| | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 |
|---|---|---|---|---|---|---|---|
| 1 | 0.0041 | 0.0070 | 0.0128 | 0.0245 | 0.0479 | 0.0946 | 0.1880 |
| 4 | 0.0020 | 0.0028 | 0.0044 | 0.0074 | 0.0134 | 0.0252 | 0.0487 |
| 8 | 0.0016 | 0.0021 | 0.0029 | 0.0045 | 0.0075 | 0.0134 | 0.0252 |
| 16 | 0.0015 | 0.0018 | 0.0022 | 0.0031 | 0.0046 | 0.0077 | 0.0136 |
| 32 | 0.0015 | 0.0017 | 0.0020 | 0.0025 | 0.0033 | 0.0049 | 0.0079 |

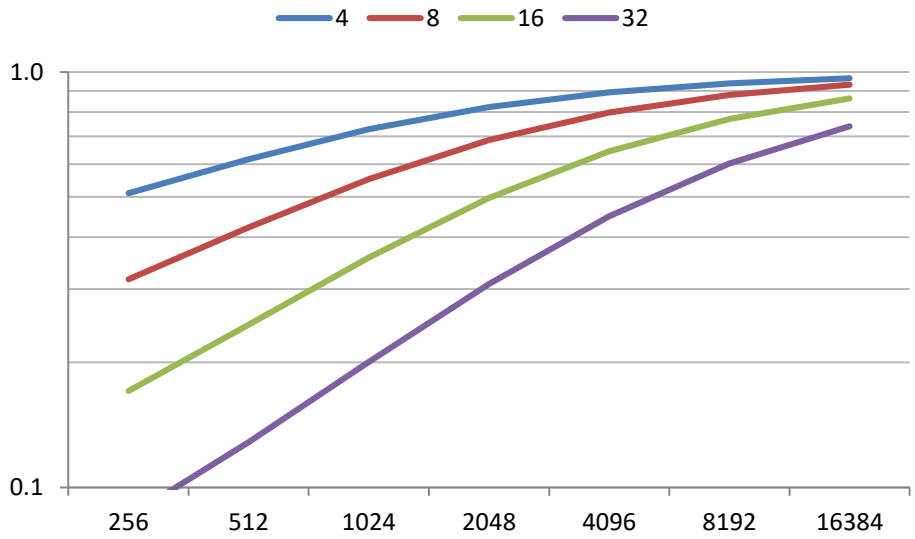
**Figure 75: Speed up for Cyclic Reduction.**



This figure 75 shows the speed up from 1 core for different system sizes. It can be observed that it cannot achieve linear increase of speedup when the system size and the number of cores are doubled. Also, for small system sizes it can be observed that there is no speedup with increase of #cores. For large system sizes the FPGA implementation with 32 cores can speed up the process, from one core, 23 times.

Next figure 76 shows the parallel efficiency (Speedup/#cores) for the same architectures. In order to achieve parallel efficiency above 90%, it must be used low number of cores with large system size. As an example, this stands for 4 cores with system size 4096.

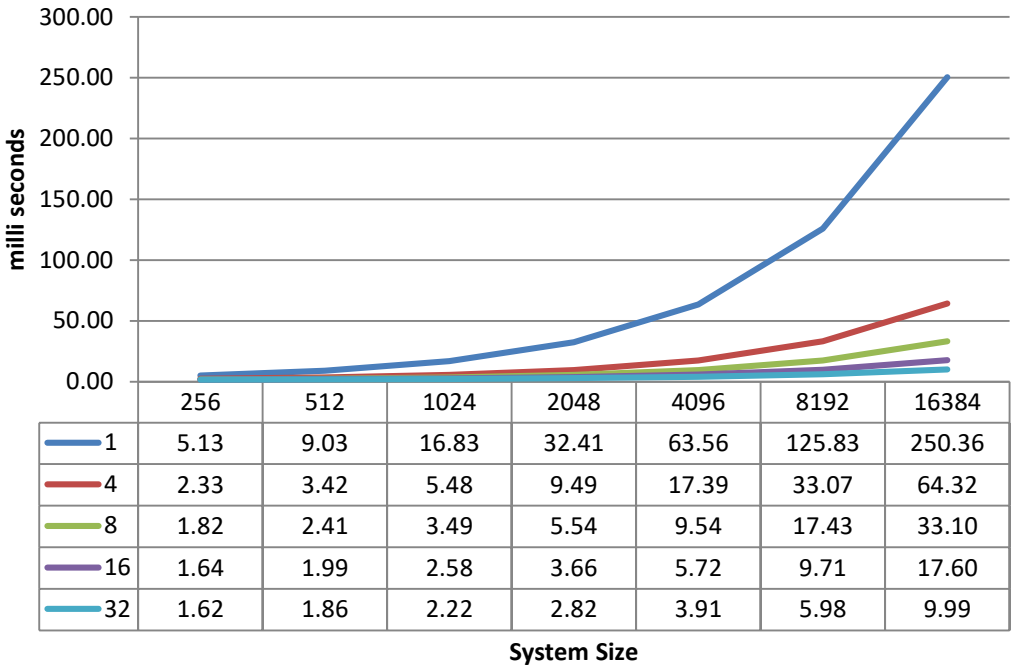
**Figure 76: Efficiency for Cyclic Reduction.**




Overall the implementation of normalized Cyclic Reduction algorithm can solve a tridiagonal system of size 16384x16384 in 0.0079ms with parallel efficiency of 75%.

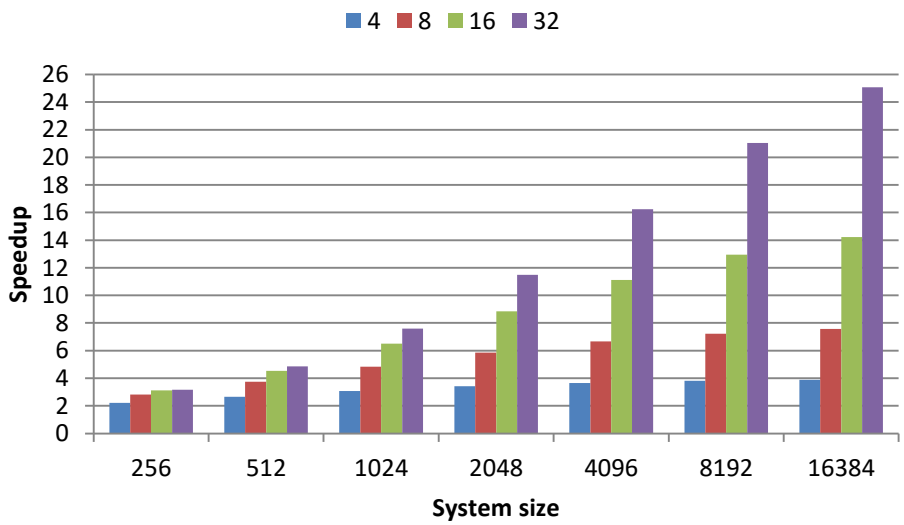
These were the results for Cyclic Reduction solver, which is the implicit part of the Crank-Nicolson Finite Difference method. To obtain the results for the whole Crank-Nicolson scheme the Update e Phase must be considered, as it is the explicit part of CN-FD. In Figure 77, timing results are presented in millisecond for the CN-FD for various system sizes. It must be noticed that, these results are for pricing an option with M = 1000 steps for the discretization in the direction of time. This choice had been made for reasons of consistency with the accuracy analysis of the Subsection of 5.4.2.

**Figure 77: Timings (milliseconds) for Crank-Nicolson.**



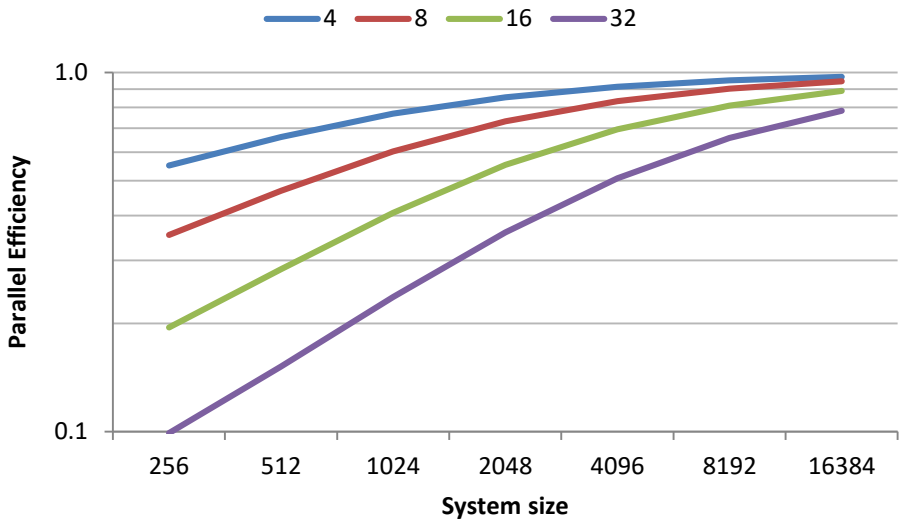

| | 256 | 512 | 1024 | 2048 | 4096 | 8192 | 16384 |
|---|---|---|---|---|---|---|---|
| 1 | 5.13 | 9.03 | 16.83 | 32.41 | 63.56 | 125.83 | 250.36 |
| 4 | 2.33 | 3.42 | 5.48 | 9.49 | 17.39 | 33.07 | 64.32 |
| 8 | 1.82 | 2.41 | 3.49 | 5.54 | 9.54 | 17.43 | 33.10 |
| 16 | 1.64 | 1.99 | 2.58 | 3.66 | 5.72 | 9.71 | 17.60 |
| 32 | 1.62 | 1.86 | 2.22 | 2.82 | 3.91 | 5.98 | 9.99 |

**Figure 78: Speed up for Crank-Nicolson.**



In Figure 78 can be observed that there is the same behavior for CN-FD with the CR, but the speed up for a system size 16384 with 32 cores is 25x, when the CR for the same settings had 23x speedup. This is because Update e Phase utilizes all the cores, in contrast of CR which is bounded by the system size and the number of cores. The same explanation stands for the parallel efficiency which is slightly better than CR (Figure 76).

**Figure 79: Efficiency for Crank-Nicolson.**




Combining the above results with the error analysis interest conclusions can be exported. In Figure 80 the log of RMSE against the time in milliseconds is presented. For this graph some assumptions were made. From the accuracy analysis it is known that the 48bit precision for a system size and above, even if produce market acceptable error, worsen the quality of the solution. Thus, it has no meaning to include points, in this graph, that have the same error but take more time. So, for system size up to 2048, the values of RMSE for 48bit precision are used, and for 4096 to 16384 the RMSE in double precision are used. This change gives the opportunity to obtain characteristic operation curves and extrude useful conclusion for the Option Pricing Accelerator. Similarly assumptions were made in [42], hardware implementation was in single precision and RMSE was obtained from double precision.

Figure 81 shows the Parallel efficiency against time (milliseconds) in log scale. It can be observed that to achieve efficiency around 80% for any multicore design are needed 10ms to produce a value for the Option.

**Figure 80: log(RMSE) against log(time in ms) for Crank-Nicolson.**

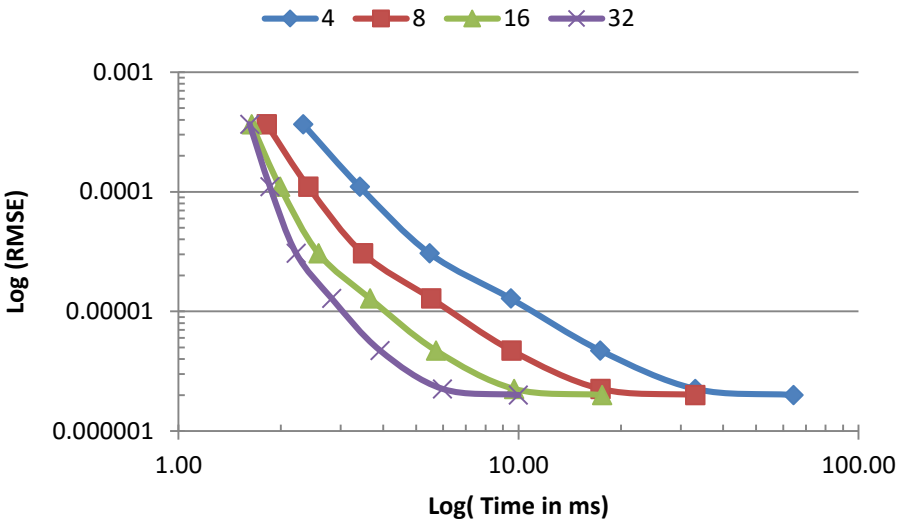
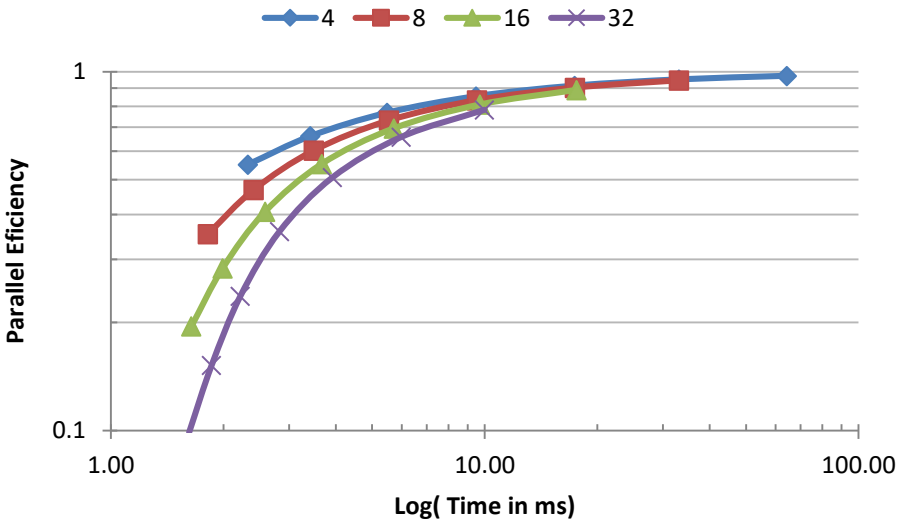



**Figure 81: log(parallel efficiency) against log(time in ms) for Crank-Nicolson.**




Operational limits of the hardware designs are examined through the application of different scenarios, for real situations that may arise in the finance sector, especially for Option Pricing. The criteria for the evaluation of every scenario are Time, Accuracy and Efficiency.

- Scenario 1: Price an option as fast as it possible.

| Cores | Time | Accuracy | Efficiency |
|---|---|---|---|
| 4 | 2.33ms | 3.67E-04 | 55.0% |
| 8 | 1.82ms | 3.67E-04 | 35.3% |
| 16 | 1.64ms | 3.67E-04 | 19.5% |
| 32 | 1.62ms | 3.67E-04 | 10.0% |

In this scenario, designs with 8, 16, 32 cores have small differences in time, with same acceptable accuracy and differ in the parallel efficiency. A rational choice would be 16 cores, as it is 0.02ms slower than 32 cores but has double efficiency.

- Scenario 2: Price an option as fast as it possible with maximum accuracy.

| Cores | Time | Accuracy | Efficiency |
|---|---|---|---|
| 4 | 33.07ms | 2.25E-06 | 95.1% |
| 4 | 64.32ms | 2.20E-06 | 97.3% |
| 8 | 17.43ms | 2.25E-06 | 90.3% |
| 8 | 33.10ms | 2.20E-06 | 94.6% |
| 16 | 9.71ms | 2.25E-06 | 80.1% |
| 16 | 17.60ms | 2.20E-06 | 88.9% |
| 32 | 5.98ms | 2.25E-06 | 65.8% |
| 32 | 9.99ms | 2.20E-06 | 78.3% |

In this scenario, the first observation is that a gain of 0.05E-06 for RMSE makes almost double the time of computations. Thus, if this improvement of accuracy is not critical for the application, the best choice for the preferences that fulfill this scenario is design with 32 cores, where the pricing is done in 5.98ms with efficiency 65.8%.

- Scenario 3: Price an option under 10ms with Efficiency above 75%.

| Cores | Time | Accuracy | Efficiency |
|---|---|---|---|
| 4 | 5.48ms | 3.05E-05 | 76.8% |
| 4 | 9.49ms | 1.28E-05 | 85.4% |
| 8 | 9.54ms | 4.71E-06 | 83.3% |
| 16 | 9.71ms | 2.25E-06 | 80.1% |
| 32 | 9.99ms | 2.20E-06 | 78.3% |

All the above alternatives are covering the criteria thresholds. It is up to the designer of the option pricing system, who has financial domain knowledge, to choose the most appropriate setting. An alternative that works without losses for 48bit precision is the 4 cores with time = 9.49ms, RMSE = 1.28E-05 and Efficiency = 85.4%.

Overall, to achieve near 80% efficiency every core of any multicore architecture must be fed with a system size of 512.

# Chapter 7:
# Conclusions and Future work

This work presented a hardware accelerator on FPGAs for Option Pricing using Crank-Nicolson Finite Difference scheme with normalized Cyclic Reduction algorithm as Tridiagonal Solver. Low-level optimizations were implemented to produce a parallel system that scale ups efficiently. These optimizations, such as custom precision arithmetic, fused operators, pipelined designs etc., were on level of hardware design and had scope to produce a highly parallel hardware architecture.

Three different hardware architectures for the main computation core were presented. First a naïve non-pipelined at 32bit precision, next a pipelined architecture using a custom 3 operand adder, also at 32bit precision and finally the proposed hardware architecture with a Fused Multiply Addition operator (FMA or fmadd). This architecture was implemented with 48bit precision, which was selected after taking into account error analysis with MPRF library and design decisions. The implementation was on a Xilinx FPGA device, Ultrascale xcvu9p, and achieved clock frequency 263MHZ.

The results of this research showed that a Tridiagonal System of size 16384x16384 can be solved by normalized Cyclic Reduction on FPGA in 0.0079ms with parallel efficiency of 75% with 32 cores architecture. A European call option can be valued, from the FPGA based accelerator of the Crank-Nicolson finite difference scheme with 4 cores, in 9.49ms with RMSE error at 1.28E-05 and parallel efficiency of 85.4%.

For future work, the followings are recommended:

- **Coarse grain parallelism** must be explored. This work did a fine grain parallelism by trying to replicate cores that can price an option as faster as it could with the Crank-Nicolson method. Results showed that 4 cores are enough to achieve good results utilizing only 2.81% of Regs resource, this means that about 30 replications of the 4core design can be made. Thus, 30 options can be priced at the same time.

- **Cloud implementation** must be explored. This was the reason for choosing the specific Xilinx FPGA device (Ultrascale xcvu9p) as Amazon Cloud uses them in F1 instances. For 1 FPGA the cost is 1.65$/per hour, taking account fine grain and coarse grain parallelism the throughput of the system would make this hardware Crank-Nicolson finite difference accelerator very competitive.

# Bibliography


[1] J. C. Hull, *Options, Futures and Other Derivatives*, vol. 1542, no. 9. 2015.

[2] J. V. Duca, "Subprime Mortgage Crisis | Federal Reserve History," *Federal Reserve Bank of Dallas*, 2013. [Online]. Available: https://www.federalreservehistory.org/essays/subprime_mortgage_crisis. [Accessed: 01-Aug-2018].

[3] M. Fleming and A. Sarkar, "The Failure Resolution of Lehman Brothers," *FRBNY Econ. Policy Rev.*, vol. 20, no. March, pp. 1–54, 2014.

[4] J. C. Shambaugh, "The Euro's Three Crises," *Brookings Pap. Econ. Act.*, vol. 2012, no. 1, pp. 157–231, 2012.

[5] A. Afonso, D. Furceri, and P. Gomes, "Sovereign credit ratings and financial markets linkages: Application to European data," *J. Int. Money Financ.*, vol. 31, no. 3, pp. 606–638, Apr. 2012.

[6] P. Boumparis, C. Milas, and T. Panagiotidis, "Economic policy uncertainty and sovereign credit rating decisions: Panel quantile evidence for the Eurozone," *J. Int. Money Financ.*, vol. 79, pp. 39–71, Dec. 2017.

[7] ECB Bulletin, "The Information Content of Option Prices During the Financial Crisis," European Central Bank, 2011.

[8] Standard & Poor's, "United States of America Long-Term Rating Lowered To ' AA + ' On Political Risks And Rising Debt Burden ; Outlook Negative," vol. 5 August, pp. 1–8, 2011.

[9] F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities," *J. Polit. Econ.*, vol. 81, no. 3, pp. 637–654, 1973.

[10] R. C. Merton, "Theory of rational option pricing," *Bell J. Econ. Manag. Sci.*, vol. 4, no. 1, pp. 141–183, 1973.

[11] J. C. Cox, S. A. Ross, and M. Rubinstein, "Option pricing: A simplified approach," *J. financ. econ.*, vol. 7, no. 3, pp. 229–263, 1979.

[12] J. C. Cox and M. Rubinstein, *Options Markets*. Prentice-Hall, 1985.

[13] M. Brennan and E. Schwartz, "Finite Difference Methods and Jump Processes Arising in the Pricing of Contingent Claims : A Synthesis," *J. Financ. Quant. Anal.*, vol. 13, no. 3, pp. 461–474, 1978.

[14] G. Courtadon, "A More Accurate Finite Difference Approximation for the Valuation of Options," *J. Financ. Quant. Anal.*, vol. 17, no. 5, pp. 697–703, 1982.

[15] J. Hull and A. White, "Valuing Derivative Securities Using the Explicit Finite Difference Method," *J. Financ. Quant. Anal.*, vol. 25, no. 1, p. 87, 1990.

[16] P. Boyle, "Options: a Monte Carlo approach," *J. financ. econ.*, vol. 4, no. 3, pp. 323–338, 1977.

[17] G. Barone-Adesi and R. E. Whaley, "Efficient Analytical Approximation of American Option Values," *J. Finance*, vol. XLII, , no. 2, pp. 301–320, 1987.

[18] F. A. Longstaff and E. S. Schwartz, “Interest Rate Volatility and the Term Structure : A Two-Factor General Equilibrium Model,” *J. Financ.*, vol. 47, no. 4, pp. 1259–1282, 1992.

[19] W. F. AMES, *Numerical Methods for Partial Differential Equations*, 3rd ed. ACADEMIC PRESS, 1992.

[20] J. Crank and P. Nicolson, “A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type,” *Math. Proc. Cambridge Philos. Soc.*, vol. 43, no. 1, pp. 50–67, 1947.

[21] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, vol. 50. 1997.

[22] J. R. Bunch and J. E. Hopcroft, “Triangular factorization and inversion by fast matrix multiplication,” *Math. Comput.*, vol. 28, no. 125, pp. 231–231, 1974.

[23] L. . Thomas, “Elliptic Problems in Linear Differential Equations over a Network,” Watson Sci. Comput. Lab Report,Columbia University, New York, 1949.

[24] S. D. Conte and C. De Boor, *Elementary numerical analysis: an algorithmic approach*, no. July. McGraw-Hill, 1972.

[25] R. W. Hockney, “A Fast Direct Solution of Poisson’s Equation Using Fourier Analysis,” *J. ACM*, vol. 12, no. 1, pp. 95–113, 1965.

[26] S. Palmer and D. Thomas, “Accelerating Implicit Finite Difference Schemes Using a Hardware Optimised Implementation of the Thomas Algorithm for FPGAs,” Computational Finance, Feb. 2014.

[27] G. Chatziparaskevas, A. Brokalakis, and I. Papaefstathiou, “An FPGA-based parallel processor for Black-Scholes option pricing using finite differences schemes,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, 2012, pp. 709–714.

[28] G. L. Zhang, P. H. W. Leong, C. H. Ho, K. H. Tsoi, D. U. Lee, C. C. C. Cheung, R. C. C. Cheung, and W. Luk, “Reconfigurable acceleration for Monte Carlo based financial simulation,” in *Proceedings - 2005 IEEE International Conference on Field Programmable Technology*, 2005, vol. 2005, pp. 215–222.

[29] D. B. Thomas, J. A. Bower, and W. Luk, “Hardware architectures for Monte-Carlo based financial simulations,” *Proc. - 2006 IEEE Int. Conf. F. Program. Technol. FPT 2006*, pp. 377–380, 2006.

[30] G. W. Morris and M. Aubury, “Design space exploration of the European option benchmark using hyperstreams,” *Proc. - 2007 Int. Conf. F. Program. Log. Appl. FPL*, pp. 5–10, 2007.

[31] X. Tian, K. Benkrid, and X. Gu, “High Performance Monte-Carlo Based Option Pricing on FPGAs,” *Eng. Lett.*, vol. 16, no. 3, pp. 434–442, 2008.

[32] A. H. T. Tse, D. Thomas, and W. Luk, “Accelerating quadrature methods for option valuation,” *Proc. - IEEE Symp. F. Program. Cust. Comput. Mach. FCCM 2009*, pp. 29–36, 2009.

[33] A. H. T. Tse, D. B. Thomas, and W. Luk, “Option pricing with multi-dimensional quadrature architectures,” in *Proceedings of the 2009 International Conference on Field-Programmable Technology, FPT’09*, 2009, pp. 427–430.

[34] A. H. T. Tse, D. Thomas, and W. Luk, "Design exploration of quadrature methods in option pricing," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 20, no. 5, pp. 818–826, 2012.

[35] Q. Jin, D. Thomas, W. Luk, and B. Cope, "Exploring reconfigurable architectures for binomial-tree pricing models," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 4943 LNCS, pp. 245–255.

[36] Q. Jin and D. Thomas, "Exploring reconfigurable architectures for tree-based option pricing models," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 2, no. 4, pp. 1–17, 2009.

[37] Q. Jin, D. Thomas, and W. Luk, "Exploring reconfigurable architectures for explicit finite difference option pricing models," in *Field Programmable Logic and Applications*, 2009, pp. 73–78.

[38] C. Wynnyk and M. Magdon-Ismail, "Pricing the American Option Using Reconfigurable Hardware," in *2009 International Conference on Computational Science and Engineering*, 2009, no. 2, pp. 532–536.

[39] X. Tian and K. Benkrid, "High-performance quasi-monte carlo financial simulation: FPGA vs. GPP vs. GPU," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 4, pp. 1–22, 2010.

[40] A. H. T. Tse, D. B. Thomas, K. H. Tsoi, and W. Luk, "Efficient reconfigurable design for pricing asian options," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 4, p. 14, 2010.

[41] C. De Schryver, I. Shcherbakov, F. Kienle, N. Wehn, H. Marxen, A. Kostiuk, and R. Korn, "An energy efficient FPGA accelerator for Monte Carlo option pricing with the Heston model," *Proc. - 2011 Int. Conf. Reconfigurable Comput. FPGAs, ReConFig 2011*, pp. 468–474, Nov. 2011.

[42] Q. Jin, W. Luk, and D. Thomas, "On comparing financial option price solvers on FPGA," in *Proceedings - IEEE International Symposium on Field-Programmable Custom Computing Machines, FCCM 2011*, 2011, no. Mc, pp. 89–92.

[43] Q. Jin, W. Luk, and D. Thomas, "Unifying finite difference option-pricing for hardware acceleration," in *Proceedings - 21st International Conference on Field Programmable Logic and Applications, FPL 2011*, 2011, pp. 6–9.

[44] T. Becker, Q. Jin, W. Luk, and S. Weston, "Dynamic constant reconfiguration for explicit finite difference option pricing," in *Proceedings - 2011 International Conference on Reconfigurable Computing and FPGAs, ReConFig 2011*, 2011, pp. 176–181.

[45] Q. Jin, T. Becker, W. Luk, and D. Thomas, "Optimising explicit finite difference option pricing for dynamic constant reconfiguration," in *Proceedings - 22nd International Conference on Field Programmable Logic and Applications, FPL 2012*, 2012, pp. 165–172.

[46] R. Sridharan, G. Cooke, K. Hill, H. Lam, and A. George, "FPGA-based reconfigurable computing for pricing multi-asset barrier options," *Symp. Appl. Accel. High-Performance Comput.*, pp. 34–43, 2012.

[47] G. C. T. Chow, A. H. T. Tse, Q. Jin, W. Luk, P. H. W. Leong, and D. B. Thomas, "A mixed

precision Monte Carlo methodology for reconfigurable accelerator systems," *Proc. ACM/SIGDA Int. Symp. F. Program. Gate Arrays - FPGA '12*, p. 57, 2012.

[48] C. de Schryver, P. Torruella, and N. Wehn, "A Multi-Level Monte Carlo FPGA Accelerator for Option Pricing in the Heston Model," *Des. Autom. Test Eur. Conf. Exhib. (DATE), 2013*, pp. 248–253, 2013.

[49] D. Sanchez-Roman, V. Moreno, S. Lopez-Buedo, G. Sutter, I. Gonzalez, F. J. Gomez-Arribas, and J. Aracil, "FPGA acceleration using high-level languages of a Monte-Carlo method for pricing complex options," *J. Syst. Archit.*, vol. 59, no. 3, pp. 135–143, 2013.

[50] G. Inggs, D. Thomas, and W. Luk, "A heterogeneous computing framework for computational Finance," *Proc. Int. Conf. Parallel Process.*, pp. 688–697, 2013.

[51] V. M. Morales, P.-H. Horrein, A. Baghdadi, E. Hochapfel, and S. Vaton, "Energy-efficient FPGA implementation for binomial option pricing using OpenCL," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2014*, 2014, pp. 1–6.

[52] S. Palmer, "Accelerating Implicit Finite Difference Schemes Using a Hardware Optimized Tridiagonal Solver for FPGAs," *ArXiv ID:402.5094*, Feb. 2014.

[53] C. Brugger, C. De Schryver, and N. Wehn, "HyPER: A runtime reconfigurable architecture for monte carlo option pricing in the Heston model," *Conf. Dig. - 24th Int. Conf. F. Program. Log. Appl. FPL 2014*, 2014.

[54] G. Inggs, S. Fleming, D. Thomas, and W. Luk, "Is high level synthesis ready for business? A computational finance case study," in *Proceedings of the 2014 International Conference on Field-Programmable Technology, FPT 2014*, 2014, pp. 12–19.

[55] A. Tavakkoli and D. B. Thomas, "Low-latency option pricing using systolic binomial trees," in *Proceedings of the 2014 International Conference on Field-Programmable Technology, FPT 2014*, 2014, vol. 1, pp. 44–51.

[56] E. Laszlo, Z. Nagy, M. B. Giles, I. Reguly, J. Appleyard, and P. Szolgay, "Analysis of parallel processor architectures for the solution of the Black-Scholes PDE," *Proc. - IEEE Int. Symp. Circuits Syst.*, vol. 2015–July, no. 1, pp. 1977–1980, May 2015.

[57] J. A. Varela, C. Brugger, C. De Schryver, N. Wehn, S. Tang, and S. Omland, "Exploiting the brownian bridge technique to improve longstaff-schwartz American option pricing on FPGA systems," in *International Conference on ReConFigurable Computing and FPGAs, ReConFig 2015*, 2015.

[58] L. Ma, F. Bin Muslim, and L. Lavagno, "High Performance and Low Power Monte Carlo Methods to Option Pricing Models via High Level Design and Synthesis," in *2016 European Modelling Symposium (EMS)*, 2016, pp. 157–162.

[59] N. K. Pham, K. Mi, M. Aung, and A. Kumar, "AUTOMATIC FRAMEWORK TO GENERATE RECONFIGURABLE," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2016, pp. 1–8.

[60] I. Stamoulias, C. Kachris, and D. Soudris, "Hardware accelerators for financial applications in HDL and High Level Synthesis," in *2017 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2017, pp. 278–285.

[61] P. Fabry and D. Thomas, “Efficient Reconfigurable Architecture for Pricing Exotic Options,” *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, no. 4, p. 29:1--29:22, 2017.

[62] A. Tavakkoli and D. B. Thomas, “A High-Level Design Framework for the Automatic Generation of High-Throughput Systolic Binomial-Tree Solvers,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 26, no. 2, pp. 341–354, Feb. 2017.

[63] F. Bin Muslim, L. Ma, M. Roozmeh, and L. Lavagno, “Efficient FPGA Implementation of OpenCL High-Performance Computing Applications via High-Level Synthesis,” *IEEE Access*, vol. 5, pp. 2747–2762, 2017.

[64] R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. Mccormick, G. Smart, R. Smart, A. Cantle, and R. Chamberlain, “Maxwell – a 64 FPGA Supercomputer,” no. Ahs, pp. 1–8, 2007.

[65] F. De Dinechin and B. Pasca, “Designing Custom Arithmetic Data Paths with FloPoCo,” *IEEE Des. Test*, vol. 28, no. 4, pp. 18–27, 2011.

[66] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann, “MPFR: A Multiple-Precision Binary Floating-Point Library With Correct Rounding,” *ACM Trans. Math. Softw.*, vol. 33, no. 2, p. 13–es, 2007.

[67] F. Oliveira, C. S. Santos, F. A. Castro, and J. C. Alves, “A Custom Processor for a TDMA Solver in a CFD Application,” in *Reconfigurable Computing: Architectures, Tools and Applications*, 2008, vol. 4943, no. Lecture Notes in Computer Science, pp. 63–74.

[68] J. D. Warne, N. A. Kelson, and R. F. Hayward, “Solving Tri-Diagonal Linear Systems using Field Programmable Gate Arrays,” in *4th International Conference on Computational Methods (ICCM2012)*, 2012, no. November, pp. 25–28.

[69] D. J. Warne, N. A. Kelson, and R. F. Hayward, “Comparison of high level FPGA hardware design for solving tri-diagonal linear systems,” in *Procedia Computer Science*, 2014, vol. 29, pp. 95–101.

[70] M. Kass, A. Lefohn, and J. Owens, “Interactive Depth of Field Using Simulated Diffusion on a GPU,” *Computing*, 2006.

[71] Y. Zhang, J. Cohen, and J. D. Owens, “Fast tridiagonal solvers on the GPU,” *ACM SIGPLAN Not.*, vol. 45, no. 5, p. 127, 2010.

[72] D. Göddeke and R. Strzodka, “Cyclic reduction tridiagonal solvers on GPUs applied to mixed-precision multigrid,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 1, pp. 22–32, Jan. 2011.

[73] P. Quesada-Barriuso, J. Lamas-Rodríguez, D. B. Heras, M. Bóo, and F. Argüello, “Selecting the Best Tridiagonal System Solver Projected on Multi-Core CPU and GPU Platforms,” *Int. Conf. Parallel Distrib. Process. Tech. Appl. PDPTA2011*, vol. 2, no. 2, pp. 839–845, 2011.

[74] D. Zhao and J. Yu, “Efficiently solving tri-diagonal system by chunked cyclic reduction and single-GPU shared memory,” *J. Supercomput.*, vol. 71, no. 2, pp. 369–390, 2014.

[75] A. Davidson, Y. Zhang, and J. D. Owens, “An auto-tuned method for solving large tridiagonal systems on the GPU,” *Proc. - 25th IEEE Int. Parallel Distrib. Process. Symp. IPDPS 2011*, pp. 956–965, 2011.

[76] L. W. Chang, J. A. Stratton, H. S. Kim, and W. M. W. Hwu, "A scalable, numerically stable, high-performance tridiagonal solver using GPUs," *Int. Conf. High Perform. Comput. Networking, Storage Anal. SC*, pp. 1–11, Nov. 2012.

[77] M. Giles, E. Laszlo, I. Reguly, J. Appleyard, and J. Demouth, "GPU Implementation of Finite Difference Solvers," *Proc. WHPCF 2014 7th Work. High Perform. Comput. Financ. - Held conjunction with SC 2014 Int. Conf. High Perform. Comput. Networking, Storage Anal.*, pp. 1–8, 2014.

[78] E. Laszlo, M. Giles, and J. Appleyard, "Manycore Algorithms for Batch Scalar and Block Tridiagonal Solvers," *ACM Trans. Math. Softw.*, vol. 42, no. 4, pp. 1–36, 2016.

[79] A. Sottoriva and B. R. June, "Investigating Finite Difference Methods for Option Pricing," 2007.

[80] R. Strzodka and D. Göddeke, "Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components," *Proc. - 14th Annu. IEEE Symp. Field-Programmable Cust. Comput. Mach. FCCM 2006*, pp. 259–270, 2006.

[81] C. Moler, "Iterative refinement in floating point," *J. ACM*, no. 2, pp. 316–321, 1967.

[82] A. F. Tenca, "Multi-operand floating-point addition," *Proc. - Symp. Comput. Arith.*, pp. 161–168, 2009.

[83] T. Yao, D. Gao, X. Fan, and X. Ren, "Three-operand floating-point adder," *Proc. - 2012 IEEE 12th Int. Conf. Comput. Inf. Technol. CIT 2012*, pp. 192–196, 2012.

[84] J. Sohn and E. E. Swartzlander, "A Fused Floating-Point Three-Term Adder," *IEEE Trans. CIRCUITS Syst. Regul. Pap.*, vol. 61, no. 10, pp. 2842–2850, 2014.